applied sciences

MDPI

Article

Real-Time Hybrid Navigation System-Based Path Planning and Obstacle Avoidance for Mobile Robots

Phan Gia Luan and Nguyen Truong Thinh *

Department of Mechatronics, HCMC University of Technology and Education, Ho Chi Minh 700000, Vietnam; pgluan95@gmail.com

* Correspondence: thinhnt@hcmute.edu.vn; Tel.: +84-903-675-673

Received: 14 March 2020; Accepted: 7 May 2020; Published: 12 May 2020

check for updates

Abstract: In this work, we present a complete hybrid navigation system for a two-wheel differential drive mobile robot that includes static-environment- global-path planning and dynamic environment obstacle-avoidance tasks. By the given map, we propose a multi-agent A-heuristic algorithm for finding the optimal obstacle-free path. The result is less time-consuming and involves fewer changes in path length when dealing with multiple agents than the ordinary A-heuristic algorithm. The obtained path was smoothed based on curvature-continuous piecewise cubic Bézier curve (C^2 PCBC) before being used as a trajectory by the robot. In the second task of the robot, we supposed any unforeseen obstacles were recognized and their moving frames were estimated by the sensors when the robot tracked on the trajectory. In order to adapt to the dynamic environment with the presence of constant velocity obstacles, a weighted-sum model (WSM) was employed. The 2D LiDAR data, the robot's frame and the detected moving obstacle's frame were collected and fed to the WSM during the movement of the robot. Through this information, the WSM chose a temporary target and a C^2 PCBC-based subtrajectory was generated that led the robot to avoid the presented obstacle. Experimentally, the proposed model responded well in existing feasible solution cases with fine-tuned model parameters. We further provide the re-path algorithm that helped the robot track on the initial trajectory. The experimental results show the real-time performance of the system applied in our robot.

Keywords: hybrid navigation system; weighted-sum model; a heuristic algorithm; piecewise cubic Bézier curve; mobile robot

1. Introduction

1.1. Motivation

Jobs in the service sector are gradually being occupied by robots because of their superior capabilities over humans, such as tireless work, low cost, robust and high precision. In order for robots to be flexible and meet work needs, service robots are often equipped with a movable platform. Due to many objective factors in the robot workspace, such as obstacle-rich workspace, environments with human presence, their mobility in indoor or outdoor workspaces is greatly limited without a matching navigation system. Navigation systems instruct a robot to go to a desired location safely, as they are capable of helping the robot respond to changes in the operating environment during travel. With the requirements of increasingly capable robots, navigation systems must also be complete and more accurate. Therefore, many navigation schemes have been rapidly developed depending on a particular environment and the structure of the robot. Currently, navigation systems for robots that operate indoors can navigate the robot move to a target with a given map. However, robots do not even have the ability to react to dynamic environments or in the presence of humans. Developing an autonomous robot that works in dynamic environments requires navigation systems that are able to

collect information from the environment surrounding the robot, simultaneously process the given map and continuously re-plan motion for the robot in order to safely reach the target. Most currently used navigation systems for service robots can be roughly categorized into two main types: global path planning and obstacle avoidance, which are introduced in detail in this study [1]. With a given map, path planning traces a trajectory from the robot to the desired position that meets the safety and kinematic constraints of the robot. In addition, the path-planning method needs to be optimized to some criteria. However, in the presence of unforeseen obstacles, the robot may not be able to reach the desired target. On the other hand, obstacle-avoidance methods help a robot identify obstacles by using sensor information and updates the direction of movement to avoid obstacles and safely reach the target. In complex environments with many obstacles, an obstacle-avoidance method will be ineffective if the target is far away from the robot. This study proposes a navigation solution for a service robot that works in our laboratory. The robot has a two-wheel differential drive platform for movement, light detection and ranging sensor (LiDAR) mounted in the platform for vision.

1.2. Related Work

1.2.1. Global Path Planning Approach

Many global path-planning methods have been studied, each with its own criteria for optimization that depend on the particular environment. It is possible to divide path planning into two categories by decision-making based classification: deterministic (A-heuristic, D algorithm, etc.) and non-deterministic (particle swarm optimization, rapidly exploring, random tree, ant colony optimization, etc.). The artificial potential field [2] is a deterministic algorithm that simulates a magnetic field with a target acting as an attractive field and obstacles acting as the repulsive field. The robot acts as a particle under the effect of these fields. The advance of [2] is to find a suitable path without stacking caused by the local minima of its optimization problem (cul-de-sac case) and reducing the oscillation of the path (zigzag). A remarkable point is the regression search method that looks for nonadjacent points as long as the line between them does not exist an obstacle and remove the remaining points. Based on that, an optimized-length path can be obtained which original APF cannot produce directly. The genetic algorithm is presented in [3] that can be considered as a search heuristic that is inspired by Charles Darwin's theory. The algorithm presented in [3] could not find the shortest path since it uses the Manhattan distance instead of Euclidean distance for fitness value. A major drawback of the genetic algorithm is that it is computationally expensive for finding an optimal path. PSO of Ferguson splines were applied in [4] for path planning. Using the algorithm, the robot can find a trajectory directly. However, with a complex map, finding a solution for the problem requires many particles (particles are used to construct spline segment), i.e., the calculation becomes very difficult without prior sub-targets. The A-heuristic algorithm was introduced in [5,6] that was used to solve the global path-planning task with the given map. The highlight in [5] is the consideration of unsafe diagonal movements, sharp turns and robot size that the original algorithm does not consider. Turning score was added to the weight of each cell in [6] to improve the optimal motion of the robot. Wu, et al., present the interesting control system for the global path planning that can adapt to the presence of the static unforeseen obstacles in [7]. Their algorithm is based on the VFH* that directly produces the smooth trajectory. However, in many cases, these trajectories just can be applicable for omnidirectional mobile robots. In [8], Kazem, et al., modified the VFH method and applied neuron network learning model in order to enhance ability to react with critical situations of robot.

A-heuristic algorithm is the simple, time-efficient algorithm and its performance is acceptable. Furthermore, special features of this algorithm are easily modified and its result is always existing for the reachable target. However, in mobile-robot path planning, we need to consider the size and kinematics constraint of the robots in order to generate the trajectory for it. We will consider these problems in Section 3.

1.2.2. Moving Obstacle-avoidance Approach

Analysis of the time–space relative trajectory of the robot and obstacle is employed in [9]. The result was obtained by finding the intersection between reachable cone and avoiding line that clearly demonstrated in [9]. However, the shape of the reachable cone depends on the robot's linear speed during the robot's movement on a local path. In order to obtain the optimal solution between many criteria, Yamada, et al., used like-WSM which takes the distance between the robot and obstacle, time to reach to the minimum distance between the robot and obstacle, reachable maximum linear speed and offset angle between the robot heading and the robot via point direction into account. Ferrara, et al., proposed mobile robot navigation in dynamic environments by using time-varying harmonics APF in [10]. A fuzzy controller used to follow the given path and avoid obstacles was introduced in [11]. Fuzzy is a common approach of many control tasks of robots since it is flexible and has the capability to handle problems with imprecise and incomplete data. Mitrovic, et al., attempted to indicate the evading static obstacle ability. With the proposed approach in [12], the robot also can avoid moving obstacles in some cases: overtaking, directly passing, but cannot have a good performance in the arbitrary situation. A few recent works have come up with novel methods such as no-target obstacle-avoidance for mobile robot [13]. Kong, Haiyi, et al., introduced the No-Target Bug1 and No-Target Bug2 algorithms for online-path planning. Since these algorithms are based on APF, they hardly adapt to the dynamic environment where the moving obstacles are presented.

In our case, we proposed an analysis time–space relative trajectory method for determining collision and weighted-sum model for obtaining optimal local path since it is simple, fast and results are acceptable.

1.3. Contributions

This work has two main contributions. First, by presenting a multi-agent A-heuristic algorithm and piecewise cubic Bézier curve, the robot trajectory was planned. A re-path algorithm was deployed to help the robot track on the planned trajectory. It greatly reduced the processing time of the robot and maximized its speed. The second contribution is the capacity of the robot that can avoid moving obstacles by using a weighted-sum model and context analysis method. In a not-too-complicated case, this model could optimize the length of the robot trajectory very quickly, and the collision rate was quite low.

2. Review of Hybrid Navigation Systems

The main objective of this study was to provide a system in order to manipulate autonomous robots that move from initial pose $\mathbf{P}_R(0)$, $\psi_R(0)$ to the desired location and heading $\mathbf{P}_D$, ψ_D with a given map. In order to archive this, the robot must have a suitable navigation scheme. According to [14], the robot's navigation scheme can be determined through four main tasks:

- Collect the data about the robot's workspace in the form of a map;
- Plan a collision-free path from the original position of the robot to the desired position;
- Generate trajectory for the robot that meets the velocity and acceleration limits of the robot;
- Avoid unforeseen obstacles—be they static or dynamic—and keep tracking initial trajectory.

This study assumes that the robots can localize themselves and obstacles can be detected by the robot on the known map. This work only focuses on three main tasks: path planning, trajectory generation and obstacle-avoidance with the processed sensor data. We propose a hybrid navigation system as in Figure 1. The advantages of this kind of system is clearly demonstrated in the study [14] and have good reality performance in [15]; its constituents can be summarized as follows:

- Deliberative layer uses prior-data in the form of map to trace a path from the position of the robot to the desired position. This type of path is called a global path. This layer is responsible for the path-planning task;
- Reactive layer uses information extracted from sensors to evaluate and decide to create a path to help the robot respond to unforeseen obstacles. This type of path is called the local path. This layer undertakes the obstacle-avoidance task;
- Executive layer decides which path to follow, while controlling the robot to follow the selected path.

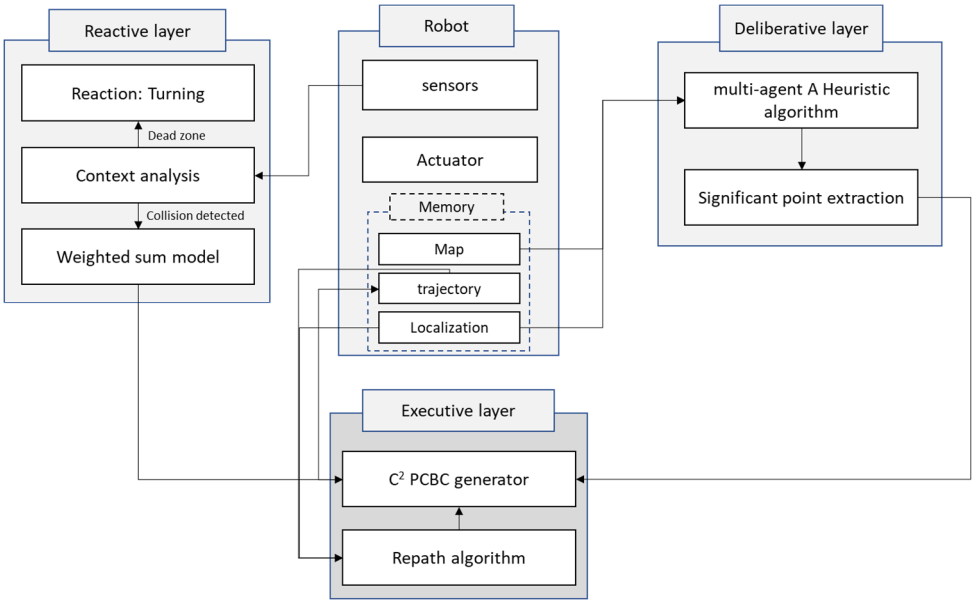

Figure 1. Flowchart of our proposed hybrid navigation system.

In the Sections 4–7, we propose methods to solve each task of the robot individually to complete the proposed navigation system seen in Figure 1. In Section 3, we briefly introduce the robot kinematic model. In Section 4, we introduce the A-heuristic algorithm for path-planning and propose our modification based on this algorithm. In Section 5, we apply the PCBC for smoothing the planned path. In Section 6, we introduce the problem in discrete control and explain our re-path algorithm for the trajectory tracking task. In Section 7, we focus on constructing the reactive layer of our navigation system that helps the robot to perceive and choose the appropriate path for avoiding moving obstacles.

3. Two-Wheel Differential Drive Platform Kinematics

This research is based on a simple 2-wheel differential drive mobile platform that has two independently controlled wheels and one omnidirectional wheel for balancing. The robot moves on the flat lane, localization by LiDAR and can be express by relation between reference frame {W} and the mobile robot's own frame {R}. Robot frame {R} = ($\mathbf{P}_R(t)$,$\psi_R(t)$) is defined by position vector of robot respect to reference frame and robot heading, respectively. Let ($V_R(t)$, $\omega_R(t)$) be the instantaneous linear speed and angular speed of robot body based on {W}, ($\theta_l(t)$,$\theta_r(t)$) be the left and right wheel angular speed, r be the radius of the driving wheel, L be a distance between two wheels. Because each command of a CPU just can accelerate a circular motion only one wheel independently, the control equation based on robot motion has form like as.

$$\begin{bmatrix} \theta_r(t) \\ \theta_l(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{r_w} & -\frac{l_w}{2r_w} \\ \frac{1}{r_w} & \frac{l_w}{2r_w} \end{bmatrix} \begin{bmatrix} V_R(t) \\ \omega_R(t) \end{bmatrix} \quad (1)$$

On the other hand, in this work, robot will continuously move along the smoothed path. The necessary condition in the path design procedure is a twice differentiable path. The given smoothed path featured by its curvature at each point on the curve. Let the curve **C**(t) is the desired path of the robot, we can archive the curvature by below equation.

$$\kappa(t) = \frac{x_{\mathbf{C}}(t)\ddot{y}_{\mathbf{C}}(t) - \ddot{x}_{\mathbf{C}}(t)y_{\mathbf{C}}(t)}{\left(\sqrt{x_{\mathbf{C}}^2(t) + y_{\mathbf{C}}^2(t)}\right)^3} \tag{2}$$

Because the unit of curvature does not involve time and is supposed to be intrinsic to a curve **C**(t), it is independent of the speed of the robot. Let V_S(t) and ω_S(t) be the desired linear velocity and angular velocity of robot when moving along the desired path. According to Equation (2), we have:

$$\kappa(t) = \frac{\omega_S(t)}{V_S(t)} \tag{3}$$

4. Path Planning

4.1. Modified A-Heuristic Algorithm (Path Finding)

A-star searching is the most popular grid-based path-planning algorithm because it is effective and low-cost for processing times. A-star is similar to Dijkstra's algorithm, but it uses a heuristic approach that makes A-star searching faster than others without it. The A-heuristic algorithm searches an optimal path by giving a prior map, starting point and destination point. It adds the heuristics function to the weight of every candidate solution. By guesting the distance of every possible route to the target point, it can skip checking many undesirable solutions and make the algorithm faster. The weight of each point can be estimated by the equation below.

$$f_n = g_n + h_n \tag{4}$$

where h_{n} is the distance between a current checking point to the target point; g_{n} is the length of the route between the initial point and the current checking point that contains the last checking point; f_{n} is the total weight of the current checking point.

This research uses the A-star algorithm for path planning of mobile robot given the prior data. The data were taken before and converted to a binary map which can be expressed as the below matrix.

$$\mathbf{M}_{i,j} = \begin{cases} 1 & (i,j) \in obsSet \\ 0 & otherwise \end{cases} \tag{5}$$

A-star searches for the optimal path based on a grid-based map. However, we cannot search point-by-point because the memory of the processor will be exceeded and require a long processing time for a large map. For this reason, a grid is generated based on the robot's position and size. Every spot in the grid contains many points; this is the sub-binary matrix of the entire map. Its size equal to twice the size of robot. For convenience, we consider each spot as the point of reduced-resolution map and be evaluated by the sum of all value of points in spot. The spot matrix (6) acts as the binary map with value 1 (spot contains partial obstacle) and value 0 (free collision spot).

$$Spot_{x,y} = \begin{cases} 1 & \sum_{(i,j)\in[x-s,x+s]\times[y-s,y+s]} \mathbf{M}_{i,j} > 0 \\ 0 & otherwise \end{cases} \tag{6}$$

If robot searches spot-by-spot on the map, the case that the next spot and the current spot lie on the diagonal line, the partial unobservable route appeared during the check (Figure 2).

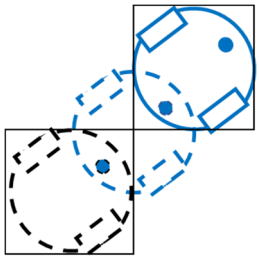

Figure 2. Robot lies in the middle of spots and moves in a diagonal line.

To overcome this problem, one square searching window is defined by four spots; the robot is in the middle of this searching window instead of the spot (Figure 3a). By searching spot-by-spot, the search window can handle the partially unobservable route in the diagonal line for robot (Figure 3b). When the searching window comes to any spot, it will evaluate whether the spot does or does not contain a obstacle. If a spot contains an obstacle, the search window does not evaluate that anymore.

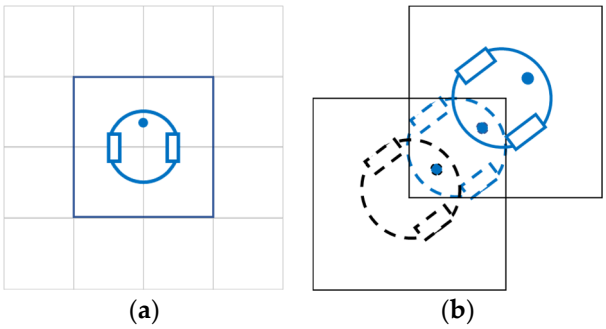

Figure 3. (**a**) Robot lies in the middle of the search window; (**b**) robot moves in a diagonal line without partial unobservable route.

In this work, we also accelerate the searching speed of A-heuristic algorithm by using parallel handling. The A-star algorithm concurrently checks multiple windows (multi-agents) at the same time. These agents must meet the requirements:

- Each agent's current checking window does not have the same location with the other in the current checking set;
- Current checking set has the lowest total f-score in open set;
- Number of current checking windows does not exceeded the processing time.

Each current checking window takes into account 8 neighbor windows. The set of all neighbor windows may have redundant windows due to overlap. If any neighbor window overlaps with another in the neighbor set, the A-star will consider its g-score. The algorithm will select the neighbor window that has the lowest g-score in the overlapping window set to take account in the next step. Other steps in the multi-agent A-star algorithm are similar to the origin. However, these agents share the open and closed sets. By this method, the algorithm can be faster, and memory also does not accumulate exponentially which results in Figure 4.

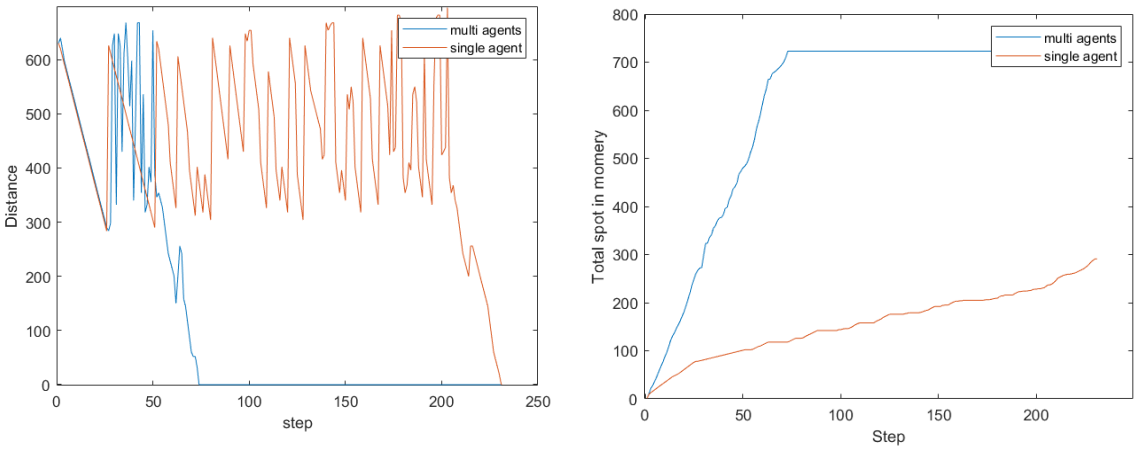

Figure 4. (**a**) Number of steps that the window which has the lowest h score of open set converges the target spot; (**b**) total spots of open set and closed set during the search.

4.2. Significant Points Extraction

$$PathSet = \{\mathbf{P}(0), \mathbf{P}(1), \ldots, \mathbf{P}(n-1)\} \tag{7}$$

The above path point set is produced by A-heuristic algorithm (illustrated in Figure 5a) still needs to be reprocessed before the smoothing path algorithm plays its role. Since A-heuristic has a limit in searching direction and distance between consecutive checking points, it produces the oscillated path that makes robot difficultly continuously follows [16], i.e., directly smoothing this path point set for robot trajectory produce large curvature trajectory and make robot decelerates to follow trajectory or slipped out trajectory many times. One more significant reason is reducing memory costs when smoothing path because the greater number of path point set causes increasing memory using to produce a smooth trajectory. First, by removing points in the same line, we can greatly reduce the number of points in the path point set. The significant point set can be present as (8) and illustrated in Figure 5b corresponding to the current example.

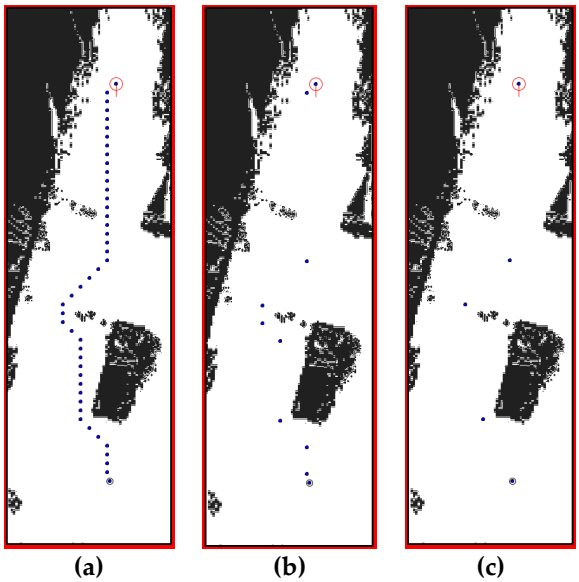

Figure 5. (**a**) Path point set is produced by A-Heuristic algorithm; (**b**) significant point set; (**c**) reduced significant point set by using regression search method.

$$significantSet = \{\mathbf{P}(i) \big| \mathbf{P}(i+1) - 2\mathbf{P}(i) + \mathbf{P}(i-1) \neq 0, \mathbf{P}(i) \in pathSet\} \cup \{\mathbf{P}(0), \mathbf{P}(n-1)\} \quad (8)$$

$$k = \{1, 2, \ldots, n-2\}$$

Geometrically, the A-heuristic algorithm produces redundant triangular paths caused by its limit in the searching direction. The shortening triangular path by the regression search method is employed. The robot sequentially checks the segmented line between the pairs of point $\{\mathbf{P}_i, \mathbf{P}_{i+2}\}$ with the obstacle-scaled map $\mathbf{M}'_{i,j}$. If the line segment does not contain any obstacle the point $\mathbf{P}_{i+1}$ will be removed, otherwise, the algorithm will continuously be checking $\{\mathbf{P}_{i+1}, \mathbf{P}_{i+3}\}$. This process will be repeated until no points are disqualified. The Figure 5c illustrates the significant point set in Figure 5b that is applied the regression search method.

5. Trajectory Generation by Piecewise Cubic Bézier Curve

5.1. Background

Named for Pierre Bézier, Bézier curves employ two endpoints called anchor points which defines the span of the line segment, which at least one additional point called control points—points used to give curviness to the original line segment. Bézier curve **B** is a parametric curve that can be expressed as the polynomial function (9) of degree n based on n+1 control point.

$$\mathbf{B}(u) = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} (1-u)^{n-i} u^i \mathbf{P}_i \quad (9)$$

where, $u \in [0, 1]$ with $\mathbf{P}_0$, $\mathbf{P}_n$ is anchor points.

Directly using the original Bézier curve to smooth robot trajectory turns out many shortcomings. First, generating an n-order Bézier curve in order to smooth the path based on n+1 significant point cause difficultly to control curviness. Second, as the order of the equation increases, the complexity, and therefore the processing time increase. Third, Bézier curves always pass through anchor points and contained in convex hull constructed by the control point set. Therefore, the curve rarely passes through points in the significant point set. This property causes a great difference between the curve and initial path set produced by A-heuristic algorithm.

There are many ways to tackle this problem, divide and conquer is the basic approach. Robot trajectory is disjointed to many curve segments. The number of curve segments is based on the number of points of significant point set. By adding k control points to each curve segment, it becomes k+1 order Bézier curve that can be presented in Equation (10). Each curve segment passes through 2 anchor points which are points in significant point set. The new curve is the piecewise (k+1) order Bézier curve and passes through all points of significant point set. The second anchor point (endpoint) of the vth curve shares the first anchor point (start point) of the (v+1)th curve (Equations (11) and (12)). This reason makes any piecewise Bézier curve becomes the continuous function over its domain.

$$\begin{aligned} &\mathbf{B}(u, v) = (1-u)^{k+1} \mathbf{P}_A(v) + \sum_{j=1}^{k} \begin{pmatrix} k+1 \\ 1 \end{pmatrix} (1-u)^{k-j+1} u^j \mathbf{C}_j(i) + u^{k+1} \mathbf{P}_A(v+1) \\ &u \in [0, 1], v \in \{0, 1, \ldots, n-1\} \end{aligned} \quad (10)$$

$$\mathbf{B}(0, v) = \mathbf{P}_A(v) \quad (11)$$

$$\mathbf{B}(1, v) = \mathbf{B}(0, v+1) \quad (12)$$

5.2. C^2 Continuous PCBC

To turn the Bézier curve path into a robot trajectory, the curve must meet robot kinematic constraints. Oscillated curves can make robots slip out of the desired trajectory. On the other hand, different from car-like robots, differential drive robot can track on C0 curve, but it is difficult to establish the controller for robot to track on it. The velocity and acceleration of robot must be continuous, thus, curve with continuous curvature meets 2 wheels differential drive mobile robot kinematic. Each curve segment is the distinct Bézier curve which has k+1 order polynomial function, i.e., each curve segment is C_{k+1} function. The problem comes from the point that links 2 segments. The entire curve is undifferentiable at these points. Therefore, we must define Equations (13) and (14) to satisfy the kinematic constraint of robot.

$$\dot{\mathbf{B}}(1,v) = \dot{\mathbf{B}}(0,v+1) \tag{13}$$

$$\ddot{\mathbf{B}}(1,v) = \ddot{\mathbf{B}}(0,v+1) \tag{14}$$

Cubic Bézier curve is the suitable solution because it has a 3rd-order polynomial function and it is not-too-complicated. By setting each segment to be cubic Bézier curve, we have the matrix-form of entire PCBC that presented in (15).

$$\mathbf{B}(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{B}(0,v) \\ \mathbf{C}_1(v) \\ \mathbf{C}_2(v) \\ \mathbf{B}(1,v) \end{bmatrix} \tag{15}$$
$$u \in [0,1], v \in \{0,1,\ldots n-1\}$$

According to Equations (13) and (15) and (14) and (15), we have the Equations (16) and (17), respectively.

$$C_2(v) = C_1(v+1) \tag{16}$$

$$\mathbf{C}_1(v) + 4\mathbf{C}_1(v+1) + \mathbf{C}_1(v+2) = 4\mathbf{P}_A(v+1) + 2\mathbf{P}_A(v+2) \tag{17}$$

Equations (16) and (17) can be expanded into the system with n-2 equations and n variables. To constrain the last 2 variables $\mathbf{C}_1$(0) and $\mathbf{C}_2$(n-2), Equations (18) and (19) are employed with ψ_A(0) and ψ_A(n-1) are the initial angle of robot and destination angle, respectively.

$$\mathbf{C}_1(0) = \mathbf{P}_A(0) + \begin{bmatrix} \cos(\psi_A(0)) \\ \sin(\psi_A(0)) \end{bmatrix} \frac{\|\mathbf{P}_A(1) - \mathbf{P}_A(0)\|}{3} \tag{18}$$

$$\mathbf{C}_1(n-1) = \mathbf{P}_A(n-1) + \begin{bmatrix} \cos(\psi_A(n-1)) \\ \sin(\psi_A(n-1)) \end{bmatrix} \frac{\|\mathbf{P}_A(n-1) - \mathbf{P}_A(n-2)\|}{3} \tag{19}$$

Experimentally, $\boldsymbol{C}_1$(0) and $\mathbf{C}_1$(n-1) determined by one-third of the Cartesian length between the corresponding point and the neighbor point in a significant point set. $\mathbf{C}_1(n$-1) is not used to construct the PCBC, but the insertion of it completes the system of Equations (16) and (17). The complete system of equation can be express as Equation (20).

$$\begin{bmatrix} 1 & 0 & & & \cdots & & & 0 \\ 1 & 4 & 1 & 0 & & & & \vdots \\ 0 & 1 & 4 & 1 & 0 & & & \\ \vdots & & & & \ddots & & & \\ & & & & & 0 & 1 & 4 & 1 \\ 0 & \cdots & & & & & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{C}_1(0) \\ \mathbf{C}_1(1) \\ \mathbf{C}_1(2) \\ \vdots \\ \mathbf{C}_1(n-2) \\ \mathbf{C}_1(n-1) \end{bmatrix} = \begin{bmatrix} \mathbf{C}_1(0) \\ 4\mathbf{P}_A(1) + 2\mathbf{P}_A(2) \\ 4\mathbf{P}_A(2) + 2\mathbf{P}_A(3) \\ \vdots \\ 4\mathbf{P}_A(n-2) + 2\mathbf{P}_A(n-1) \\ \mathbf{C}_1(n-1) \end{bmatrix} \tag{20}$$

By solving the system of Equation (20), all control points can be obtained. The PCBC now has continuous curvature over the entire curve and only depends on the set of anchor points, start angle and destination angle. The significant point set in Figure 5c that fed to Equation (20) results in the continuous curvature over the entire path illustrated in Figure 6.

The arc length of the vth segment given u = μ can be found by Equation (21).

$$l(\mu, v) = \int_0^{\mu} \|\dot{\mathbf{B}}(u, v)\| du \tag{21}$$

The arc length of entire PCBC respect to curve parameters μ, v can be obtain by sum up total arc length of ($\{v\}_0{}^{v-1}$)th curve segments and arc length of the vth curve segment given u=μ that presented in Equation (22).

$$L(\mu, v) = \sum_{v=0}^{v-1} l(1, v) + l(\mu, v) \tag{22}$$

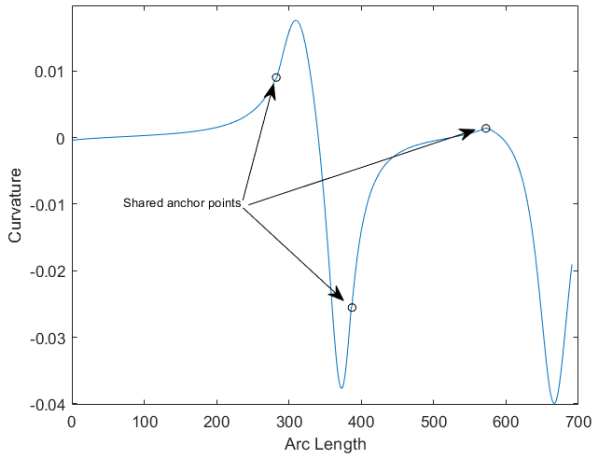

Figure 6. Curvature of piecewise cubic Bézier curve (C^2 PCBC) are continuous over the entire curve.

6. Discrete Time Control Method

PCBC is the curve that has a parameter that does not respect to arc length [17]. Therefore, it cannot be used directly as a robot trajectory. To solve this problem, reparameterization is required. This study employs a local linear approximation approach for approximating the curve parameter based on the arc length.

Based on Equations (21) and (22), by sampling curve parameter we can obtain Table 1 for converting between arc length and curve parameter.

Let L_R(t) be the total distance traveled by the robot from its initial position to $\mathbf{P}_R$(t). Suppose that robot have followed the curve, from the table above, we can approximate the value of the curve parameter u and v respect to t.

$$\begin{aligned} u(t) &= u(L_R(t)) \\ v(t) &= v(L_R(t)) \end{aligned} \tag{23}$$

Let $\mathbf{P'}_R$(t) = $\mathbf{P^B_R}$(t) = $\{u(t), i(t)\}$ be the desired position of the robot on curve respect to curve parameter at time t. Let $\kappa(t) = \kappa(u(t), i(t))$ be curvature of curve at time t. According to Equation (2), $\kappa(t)$ can be achieve by equation below.

$$\kappa(t) = \frac{\dot{\mathbf{B}}(u(t), v(t)) \times \ddot{\mathbf{B}}(u(t), v(t))}{\|\dot{\mathbf{B}}(u(t), v(t))\|^3} \tag{24}$$

Table 1. Data for converting between arc length and curve parameter.

v	u	l(u,v)	L(u,v)
0	**0**	**0**	**0**
0	0.01	0.01	0.01
0	0.02	2.8427	2.8427
0	0.03	5.6792	5.6792
0	0.04	8.5367	8.5367
0	0.05	11.3858	11.3858
.	.	.	.
.	.	.	.
3	0	0	687.7763
3	0.01	0.001	687.7773
3	0.02	1.8897	689.6662
3	0.03	3.7465	691.5228
.	.	.	.
.	.	.	.
3	0.98	115.2653	803.0417
3	0.99	116.2936	804.07
3	1	117.343	805.1194

Let VS(t), ωS(t) is the desired linear and angular speed of robot at time t; $\theta_{R_S}(t)$, $\theta_{L_S}(t)$are the desired right and left wheel angular speed of robot, $\theta_{\max}$ is the maximum angular speed that robot gear can achieve. According to Equations (1) and (3), if $\kappa(t) > 0$ then set $\theta_{R_S}(t) = \theta_{\max}$ and $\theta_{L_S}(t)$ by the Equation (25).

$$\theta_{L_S}(t) = \frac{2 - \kappa(t) l_w}{2 + \kappa(t) l_w} \theta_{R_S}(t) \tag{25}$$

If $\kappa(t) < 0$ then set $\theta_{L_S}(t) = \theta_{\max}$ and $\theta_{R_S}(t)$ based on Equation (25). According to Equation (1) we can obtain $V_S(t)$ and ω_S(t).

According to Equations (24) and (25), the robot can constrain its speed when moving on a trajectory. However, as the discussion in [18], maintaining maximum speed on one gear causes the robot to easily slip out of the desired trajectory when the robot goes through a trajectory segment which has large curvature due to its failure to respond to the acceleration in the other gear. That reason causes deviations between V_S(t), ω_S(t) and V_R(t), ω_R(t). Together with the open system, the robot cannot determine the error between its actual position and desired position on the curve that causes cumulative errors described in Figure 7. Therefore, the desired position of the robot on the curve will be different from the actual position $\mathbf{P}'_R(t) \neq \mathbf{P}_R(t)$. To be able to create a closed-loop system, the robot will refer to its true position by localization sensors, while the desired position on the trajectory is determined by the parameter of the curve determined by Equation (15) and (23). By the position error, the robot needs to be navigated for returning initial trajectory. To overcome this problem, we propose a method of re-path algorithm summarized in Figure 8. Instead of changing the robot behavior, our method changes shape of initial trajectory. Let $\varepsilon(t)$ be the position error of the robot. If $|\varepsilon(t)| > \varepsilon_{max}$ then based on the main trajectory, the robot will reproduce a new path that does not differ from the remaining path to help the robot return desired position. The new path receives the actual current position of the robot is the initial point, $\mathbf{P}_{\text{back}}(t)$ is the point located in the curve and it is added to the new path. points in path point set that the robot has not passed through are also added to the new path. The C^2 PCBC generator takes its role and generates a new trajectory called a reshaped trajectory. The curve parameters which localize $\mathbf{P}_{\text{back}}(t)$ can be achieved by Equation (26).

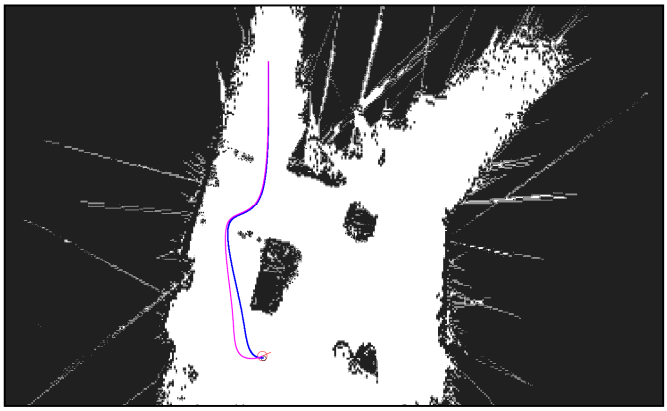

Figure 7. Cumulative errors of actual robot position respect to reference trajectory.

$$\begin{aligned} u_{back}(t) &= u(L_R(t) + \Delta L_{back}) \\ v_{back}(t) &= v(L_R(t) + \Delta L_{back}) \end{aligned} \tag{26}$$

If $i_{\text{back}}(t)$ and $u_{\text{back}}(t)$ do not exceed maximum parameters of PCBC, $\mathbf{P}_{back}(t)$ can be obtained by equation below:

$$\mathbf{P}_{back}(t) = \mathbf{B}(u_{back}(t), v_{back}(t)) \tag{27}$$

Otherwise:

$$\mathbf{P}_{back}(t) = \mathbf{P}(n-1) \tag{28}$$

where $\mathbf{P}(n-1)$is the given target position of the robot. Figure 9 illustrates the case in that the robot slip out of trajectory and attempting to get back the initial trajectory by employing the re-path algorithm.

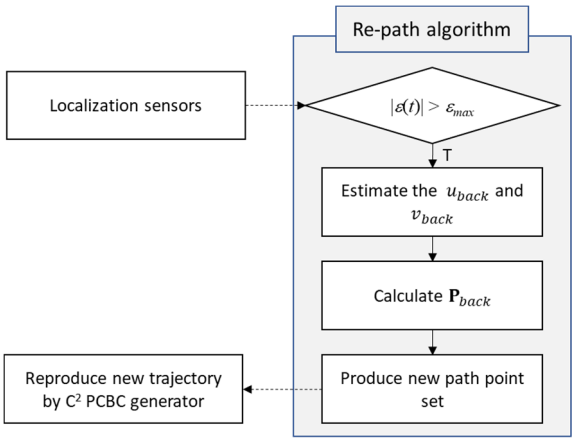

Figure 8. Re-path algorithm flow chart.

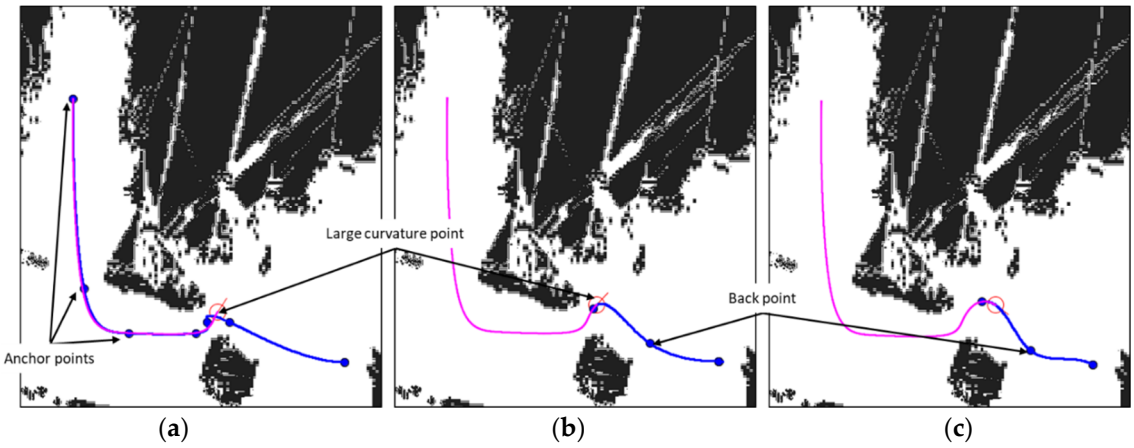

Figure 9. (**a**) Robot moves to the position where curvature is large enough, and deviates from desired trajectory; (**b**) After robot deviates from the trajectory, robot will create a subtrajectory to the back point, (**c**) The robot continues to fail to meet the acceleration condition caused by large curvature of the first created subtrajectory, the robot repeatedly create subtrajectory until it returns to original path.

7. Obstacle-Avoidance

7.1. Detect Collision by Using Gradient Descent and Context Analysis Based Obstacle-Avoidance Scheme

For convenience, we assume that the robot sensors have already detected moving obstacles when it moves over the scanning area. The information from sensors can obtain which includes a radius of the circular boundary of obstacle body, position of center of boundary and constant linear velocity. It has two distinct cases when the robot detects the object:

- Obstacle passes over the dead zone.
- Obstacle passes over the scanning zone, but not dead zone. These zones of sensors were introduced in Figure 10.

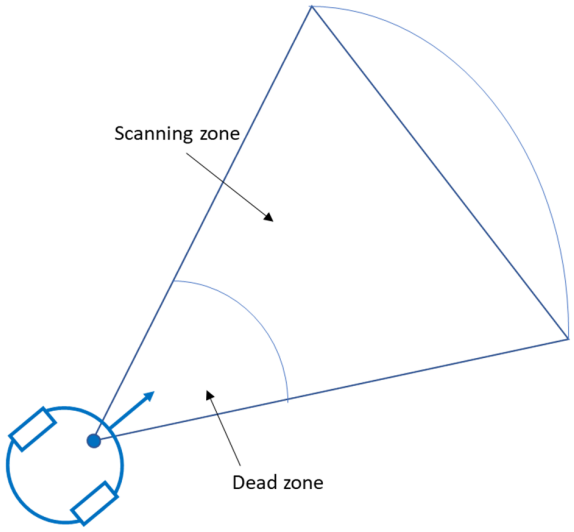

Figure 10. Illustrating dead zone and scanning zone of the robot.

Each case has different behaviors to avoid the obstacle based on context analysis and is illustrated in Figure 11. In the first case, when an obstacle suddenly passes over the dead zone (unpredictable case), the robot will try to stop and turn to the left until the obstacle does not present in the dead zone anymore. In the second case, when an obstacle passes over the scanning zone, the robot will check

whether a collision is occurring. If the current trajectory were free-obstacle, the robot would continue to track on it. Otherwise, the robot must change the velocity and plans a new path. In this work, we assume the robot is on the emergency situation and needs to move as fast as possible. Therefore, in the second case, to an avoid obstacle, the robot must change the velocity direction instead of slowing down. The robot will generate a new path that will help the robot dodge the moving object and lead the robot back to the initial path at some point further. The new path has optimized arc length. Let $\{O\} = (\mathbf{P}_O(t), \psi_O)$ be the obstacle frame respect to reference frame, obstacle moving with constant velocity $\mathbf{V}_O$ and R_O be a radius of the circular boundary. If the distance between obstacle and the robot is equal or less than sum of radius of them then collision between the obstacle and the robot will occur. Let $\mathbf{P}_O^R(t)$ is the relative position of object respect to the robot position determined by Equation (29).

$$\mathbf{P}_O^R(t) = \mathbf{P}_O(t) - \mathbf{P}_R(t) \tag{29}$$

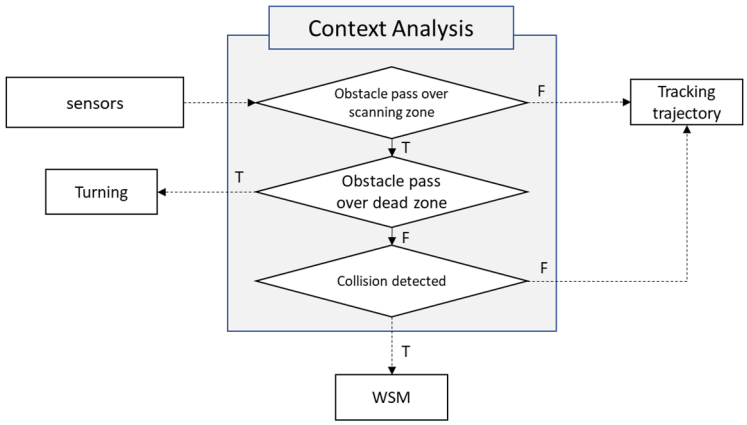

Figure 11. Context analysis flow chart.

The path is safe if the minimum distance between the robot and moving obstacle is greater than sum of radius of them. This condition is held by the inequation below.

$$\min_{t \in [t_0, \infty)} \|\mathbf{P}_O^R(t)\| > R_R + R_O \tag{30}$$

However, the derivative of Equation (29) is very complicated. Solution of inequation (30) can be obtained by using the univariate gradient descent algorithms with initial solution candidate t_0 is assigned by the time when the robot detects obstacle and initial learning rate is γ_0. By Equations (31) and (32), the robot can find out candidate solution very fast.

$$t_{n+1} = t_n - \gamma_n \|\mathbf{P}_O^R(t_n)\| \tag{31}$$

$$\gamma_n = \frac{t_n - t_{n-1}}{\|\mathbf{P}_O^R(t_n)\| - \|\mathbf{P}_O^R(t_{n-1})\|} \tag{32}$$

According to the Equation (29) and established candidate solution of (31), the shortest distance between obstacle and the robot is obtained. By the condition (30), the robot can decide to continue following the initial path or getting a new path. If the robot detects the collision, the robot will construct a new path by following these steps:

- Decide offset angle for a new direction to avoid the obstacle;
- Repeat step 1 until the obstacle has not been observed anymore;
- Interpolate back point which is inside the initial path in order to help the robot get back main trajectory;

- Construct reshaped trajectory.

In step one, we propose the multi-objective optimization problem to decide offset angle.

7.2. Decide Offset Angle by Using WSM

Let α_r, l_r be angular measure range and long-range of sensor. Let $\alpha(t) \in [-\alpha_r/2, \alpha_r/2]$ be the desired offset angle of the robot at t when the robot detected the collision, The range of offset angle is based on angular measure range of sensor because the robot keeps moving at high speed that causes dangerous case when the robot turns to the unobservable direction. The offset angle is selected based on these below functions and conditions:

Function 1. *$l(\alpha,t)$ is the distance that pulsed light wave emitted at α was traveled at time t subtracts dead zone radius. This distance is extracted from sensors data;*

Function 2. *$d^R_{O\min}(\alpha,t)$ is the smallest distance from the robot to the estimated relative trajectory; between the object and the robot at an alpha angle and time t;*

Function 3. *$d^R_{B\min}(\alpha,t)$ is the smallest distance from the new estimated robot position to the desired trajectory when the robot moves in the direction formed by α;*

Function 4. *$t_\alpha(\alpha,t)$ is the time it takes a robot to achieve compensation angle α at time t when the robot is moving with $V_R(t)$, $\omega(t)$;*

Condition 1. *If $l(\alpha,t)$ is maximized at α, α will be selected;*

Condition 2. *If $d^R_{O\min}(\alpha,t)$ is maximized at α, α will be selected;*

Condition 3. *If $d^R_{B\min}(\alpha,t)$ is minimized at α, α will be selected;*

Condition 4. *If $t_\alpha(\alpha,t)$ is minimized at α, α will be selected;*

These conditions involve four objective functions (33) that must be optimized simultaneously.

$$\left.\begin{array}{l} \max\limits_{\alpha} l(\alpha,t) \\ \max\limits_{\alpha} d^R_{O\min}(\alpha,t) \\ \min\limits_{\alpha} d^R_{B\min}(\alpha,t) \\ \min\limits_{\alpha} t_\alpha(\alpha,t) \end{array}\right\} subject\ to\ \alpha \in \left[-\frac{\alpha_r}{2}, \frac{\alpha_r}{2}\right] \tag{33}$$

A very common method to ranking candidate solutions against a set of n-objective functions is the weighted-sum method. WSM illustrated in Figure 12 turns original n-objective functions into a single objective function. This method's features are described in detail in [19]. We call the needed maximizing function is a beneficial factor and needed minimizing function is a non-beneficial factor. Before WSM takes its role, all factors should be normalized. A beneficial factor will be divided by its optimal value. Inverse of a non-beneficial factor will be multiplied by its optimal value. Let w_1, w_2, w_3, w_4 are weight values of WSM, respectively that meet condition $w_1 + w_2 + w_3 + w_4 = 1$. By WSM we obtain the total weighted factor $f(\alpha,t)$.

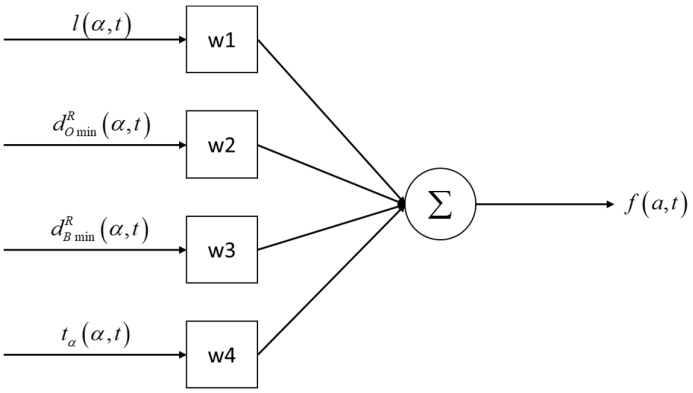

Figure 12. Total weighted factor is recurred by WSM.

The selected α that optimal the total weighted factor. By this value, the new subtrajectory is generated.

Let R_α be the radius of circle those centers located at current position of the robot, $\mathbf{P}_\alpha(t)$ be any point on the circle can be determined via:

$$\mathbf{P}_\alpha(t) = R_\alpha \begin{bmatrix} \cos(\psi_R(t) + \alpha(t)) \\ \sin(\psi_R(t) + \alpha(t)) \end{bmatrix} + \mathbf{P}_R(t) \tag{34}$$

$\mathbf{P}_\alpha(t)$ is also the destination point of the subtrajectory. Hence, there we have a new anchor point set that includes $\mathbf{P}_R(t)$ and $\mathbf{P}_\alpha(t)$ to produce subtrajectory. Figure 13 illustrates the subtrajectory set produced by the $\mathbf{P}_\alpha(t)$ with different offset angles. However, the value of R_α depends on the current parameters and the kinematic constraint of the robot. R_α must meet the worst case when the robot is tending to turn right or left sharply, but the offset angle has value in the opposite site respect to the robot heading, i.e., the robot responds well at one-sided of the boundary of the range offset angle and does not at the other side. Experimentally, we can obtain R_α value by simulating the worst-case many times and choosing the shortest one that meets the case.

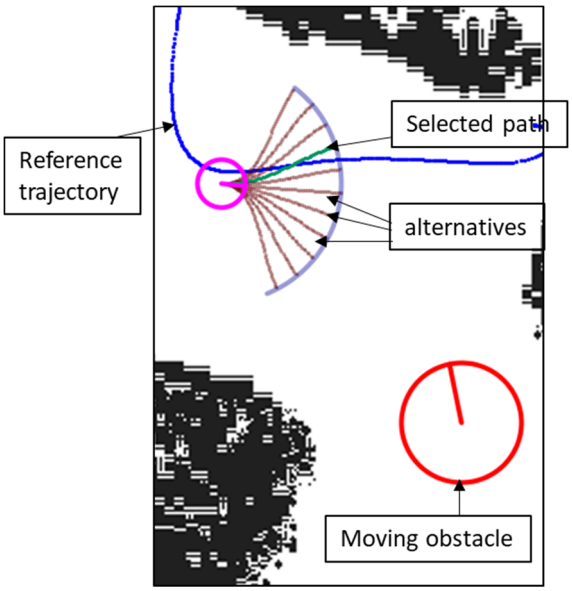

Figure 13. Visualization of sub-trajectories constructed by alternatives of the weighted-sum model.

At every time step, robots will collect data and extract α values from WSM, create subtrajectory and continuously adjust until the obstacle is no longer within the sensor visibility. Once safe, the robot will return to the original trajectory. The summary of WSM used in the obstacle-avoidance task is illustrated in Figure 14.

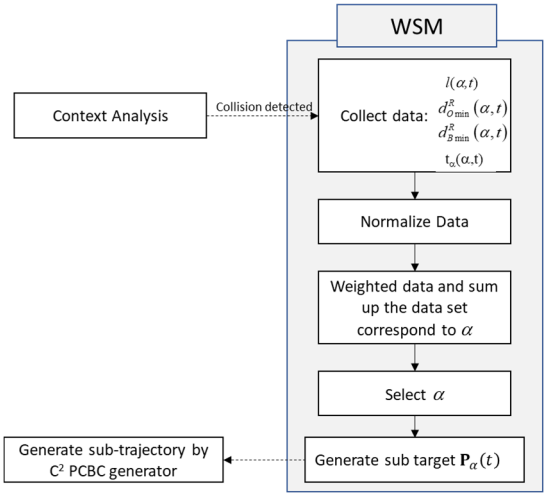

Figure 14. The WSM flow chart.

7.3. Returning Main Trajectory by Reshaping Curve Algorithm

After the robot no longer see the observed obstacle, the robot will go back to the main trajectory. For easily getting back to the main trajectory, we will interpolate a retuning point on the path that has fixed distance ΔL from it to the current position of the robot. Let $L(u_i,i_R)$ be the total distance which the robot has moved, $L(u_b, i_b)$ is the arc length of curve from the initial point to the returning point that defined by Equation (35).

$$L(u_b, i_b) = L(u_R, i_R) + \Delta L \tag{35}$$

By the approximating table and the Equations (26)–(28) we obtain the returning point position. In the current example, Figure 15 illustrates the re-path algorithm (illustrated in Figure 8) implementation when the robot exceeds the allowed position error.

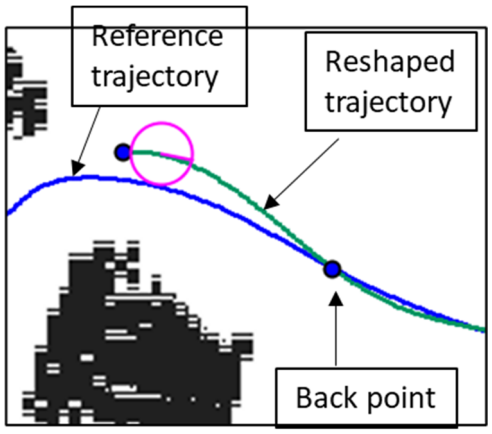

Figure 15. Visualization of re-pathed trajectory.

8. Experiments and Discussions

The robot (Figure 16) performs experiments in the lab with the given pre-built map (Figure 17). The robot will be placed in two different scenarios are shown in Table 2. The robot starts at its station and moves to the workpiece-collection point. In the second scenario, the robot leaves the post-processing point to move to the storage station. The following picture shows a detailed description of the lab's recorded map:

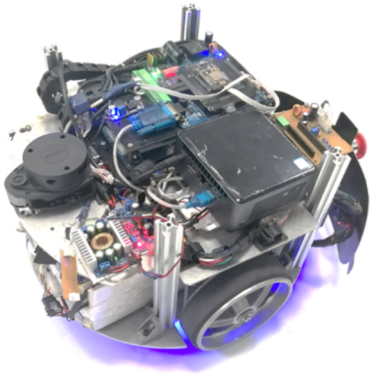

Figure 16. Mobile platform for experiments.

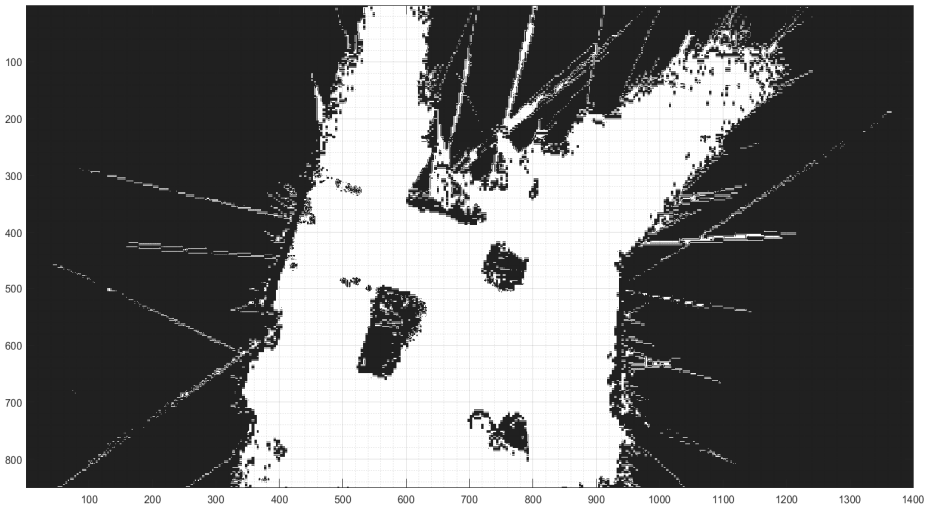

Figure 17. The map of lab pre-built by the robot.

Table 2. Robot's task for each scenario.

Scenario #	Initial Position	Initial Heading	Destination Position	Destination Heading
1	(560, 120)	90 degree	(900, 450)	0 degree
2	(900, 250)	−90 degree	(500, 625)	0 degree

According to our distributions, we divide our experiments into three parts: path-planning performance, tracking trajectory by re-path algorithm performance and moving obstacle-avoidance performance.

8.1. Path Planning Performance

In the first experiment, we will investigate the processing speed, memory usage and length of paths produced by original A-heuristic and multi-agent A-heuristic path planning with the presence or absence of the significant points extraction algorithm (SPEA). The memory usage of each algorithm is measured by the number of checked spots.

In Tables 3 and 4, respectively, the indices obtained by the algorithms through the two scenarios mentioned above are presented. The trajectories produced by fed the path point set of different algorithms to the C2 PCBC generator are also illustrated in Figure 18. In both scenarios, we see the superiority of the ability to optimize length of the path produced by the original algorithm even with or without the presence of SPEA compared to a multi-agent algorithm. However, A-heuristic in the presence of SPEA not only reduces the size of the path set, but also reduces the length of the path. Therefore, the variance of path length produced by two algorithms is no longer a problem.

Furthermore, the presence of SPEA reduce the oscillation and curvature–magnitude of the path that presented in Figure 18. In the first scenario, when the robot uses a multi-agent algorithm, its processing time is reduced by 58.3%, but the number of spots checked is 3.2 times that of original A-heuristic.

In the second scenario, with average numbers of agents is 9.7, 119% more spot-checked and processing time only decreased by 11.1%, this is the case where the multi-agent algorithm performs not really well compared to the amount of memory it consumes.

Table 3. Indices obtained by original A-heuristic algorithm and multi-agent A-heuristic algorithm with or without the presence of a reduction path algorithm in the first scenario.

Algorithm	Processing Time (ms)	Length of Path (cm)	Total Checked Spot (Spot)	Path Set Size (Spot)
Original A heuristics	12	603.1909	251	39
Multi-agent A heuristics	5	657.7839	804	40
Original A heuristics with significant points extraction algorithm	N/A	601.83997	N/A	7
Multi-agent A heuristics with significant points extraction algorithm	N/A	602.94867	N/A	6

Table 4. Indices obtained by original A-heuristic algorithm and multi-agent–heuristic algorithm with or without the presence of a reduction path algorithm in the second scenario.

Algorithm	Processing Time (ms)	Length of Path (cm)	Total Checked Spot (Spot)	Path Set Size (Spot)
Original A heuristics	18	692.5779	618	40
Multi-agent A heuristics	16	704.1758	1416	40
Original A heuristics with significant points extraction algorithm	N/A	666.4103	N/A	5
Multi-agent A heuristics with significant points extraction algorithm	N/A	668.11456	N/A	5

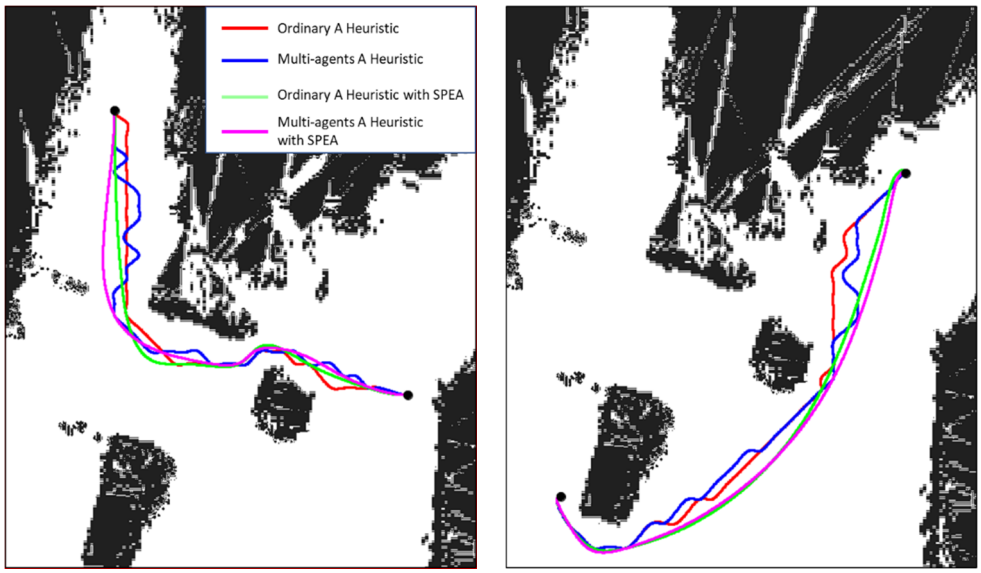

Figure 18. (**a**) Trajectories produced by fed the path point set of different algorithms to the C2 PCBC generator in the first scenario, (**b**) in the second scenario.

8.2. Tracking Trajectory by Re-Path Algorithm Performance

As mentioned in Section 5, the re-path algorithm is employed to control the position error of the robot's position compared to the desired position located on the initial trajectory. However, when the robot generates a subtrajectory, the main trajectory itself has a variation in curvature between it and

the initial trajectory. The reason is keeping continuous curvature at $\mathbf{P}_{back}(t)$. If the curvature is greatly changed on the whole main trajectory, the robot will not be able to respond to the position error because of the difference between changed main trajectory and initial trajectory. However, the re-path algorithm only affects a small segment around $\mathbf{P}_{back}(t)$, which can be demonstrated by the series of curvature of main and reshaped trajectory respect to arc length in Figures 19 and 20, according to the first and second scenario, respectively.

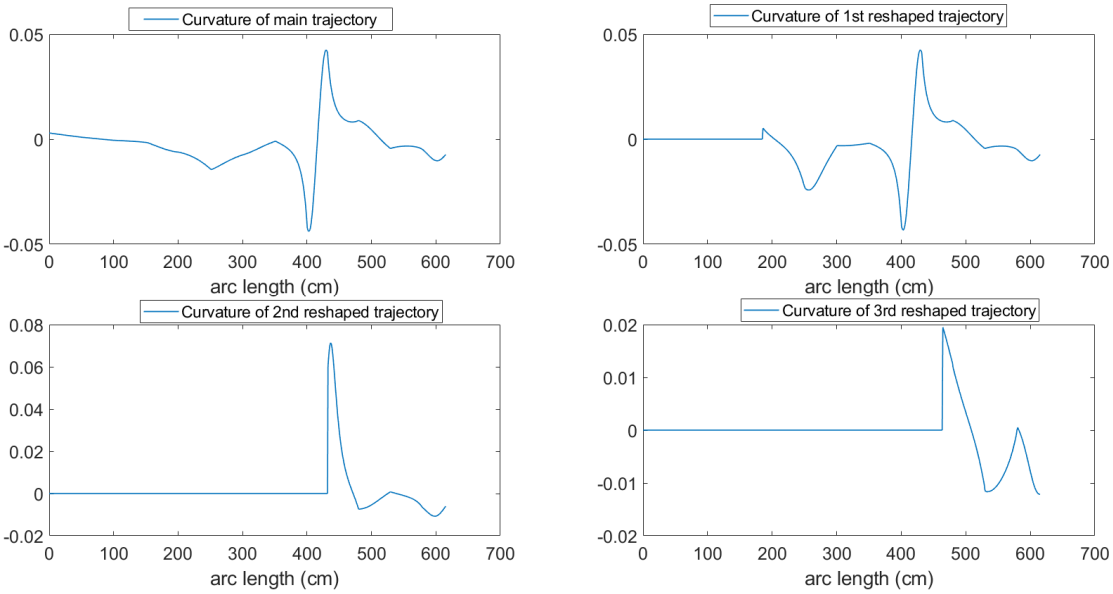

Figure 19. Trajectories' curvature of the first scenario.

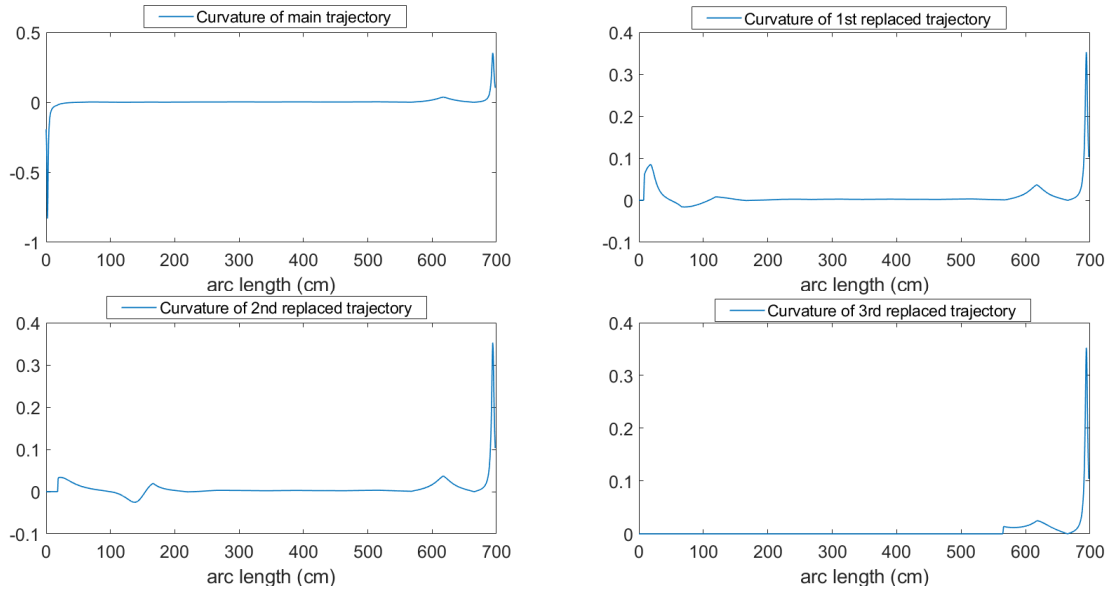

Figure 20. Trajectory curvature of the second scenario.

In both scenarios, the robot re-paths three times at the points where the accumulated position error (Figures 21 and 22) exceeds the allowed deviation on its entire trajectory from start position to destination position.

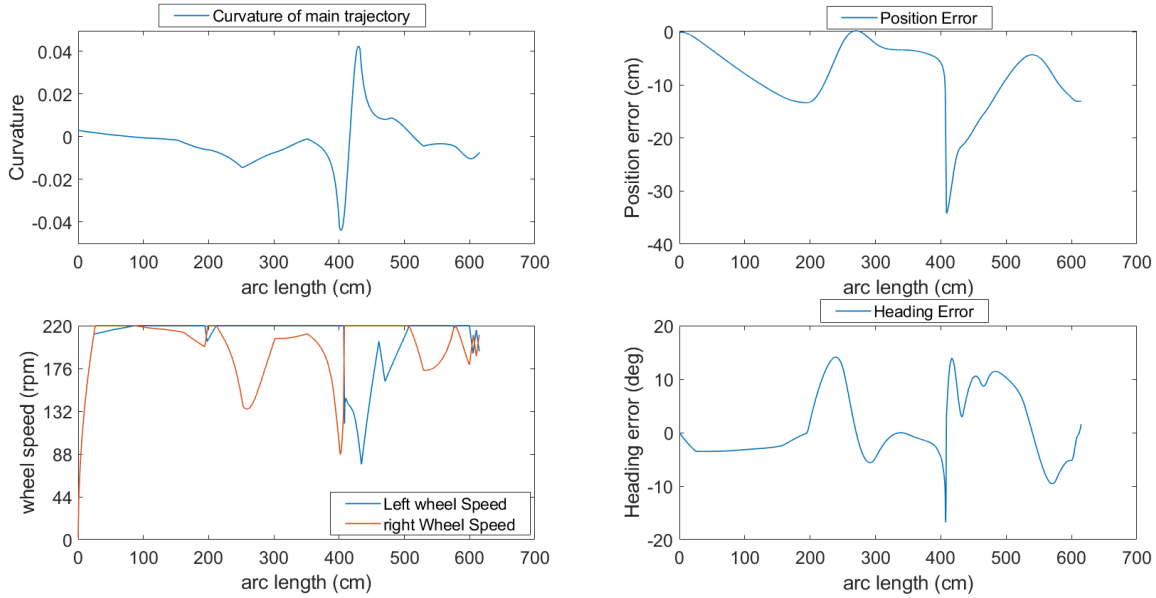

Figure 21. Left and right wheel angular speed, position error and heading error of the robot's pose, with respect to desired pose on trajectory of the first scenario.

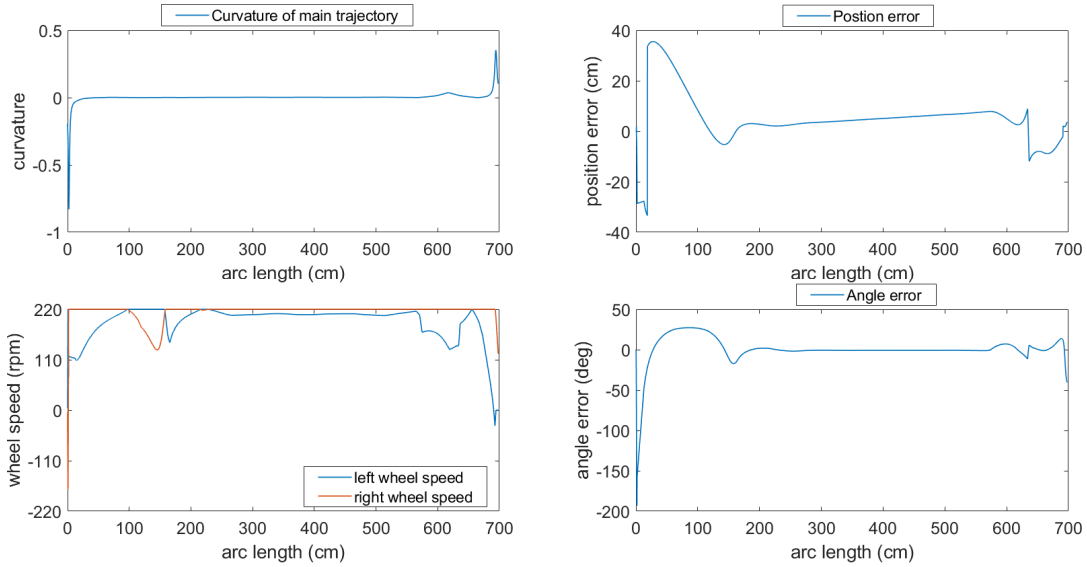

Figure 22. Left and right wheel angular speed, position error and heading error of the robot's pose, with respect to desired pose on trajectory of the second scenario.

In this experiment, the geometric parameters and kinematic limits of the robots are shown in the following Table 5.

Table 5. Robot geometric parameters and kinematic limits.

Parameter	Value
R_R	0.4 m
l_w	0.36 m
r_w	0.06 m
$\theta_{\max}$	220 rpm
$\theta''_{\max}$	2.2 rad/s^2

Let ε_{max} = 10 cm. In the first scenario, the position on the trajectory that has arc length L(u,i) ≈ 4 m has the corresponding curvature is −0.048 m^{-1} and the position has arc length $L(u,i)$ ≈ 4.4 (m) has the corresponding curvature is 0.048 m^{-1}, The average rate of change of curvature respect to arc length is approximately 0.24 m^{-2}. The robot does not meet the desired velocity at this point, so it slips off the original trajectory. We can easily see the deviation in position and heading of the robot with respect to the desired position and heading at this point on the position error and heading error graphs. However, by the third and fourth re-paths, the robot gradually takes control of the error. In the second scenario, In the initial position, the robot has an initial heading in the opposite direction of the path, it is easy to see that the robots cannot meet the angular speed at this point, leading to large heading errors and position error in the beginning. By using the re-path algorithm, the robot quickly returns to its original trajectory by first and second re-paths. It illustrated in Figure 20.

8.3. Test Moving Obstacle-Avoidance Ability

To be able to verify the moving obstacle avoiding ability, in the first scenario, we simulate two obstacles with their initial position, velocity and size as Table 6.

Table 6. Virtual obstacle parameters for testing moving obstacle avoiding ability in first scenario.

	1st Obstacle	**2nd Obstacle**
Initial position	(585, 400)	(850, 300)
Velocity	(−0.1, −0.5)	(0, 0.1)
Size (cm)	40	60

Experimentally, we set the weight values when the robot detect collision is w_1 = 0.3, w_2 = 0.15, w_3 = 0.33, w_4 = 0.22. Figure 23 is a snapshot of the robot's movement from the starting point to the target based on tracking the previously created trajectory in the first scenario. In the process of moving in a given trajectory, the robot must dodge these simulated obstacles. When no obstacle is presented, the robot moves along the given trajectory. If any obstacle that falls into the scanning zone then the robot will check for collision against that obstacle. In the given situation, both obstacles certainly collide with the robot when it maintains the original trajectory. Therefore, when these obstacles are detected by the robot, the robot is forced to update offset angle to avoid obstacles while establishing a subtrajectory from this parameter. The offset angle is constantly updated until the detected obstacle leaves the robot's scanning zone. Right at that time, by re-path algorithm, the robot tries to return to the original trajectory. We also have some snapshot of the robot (Figure 24) that presents the performance of the robot in the second scenario without the presence of unforeseen obstacles.

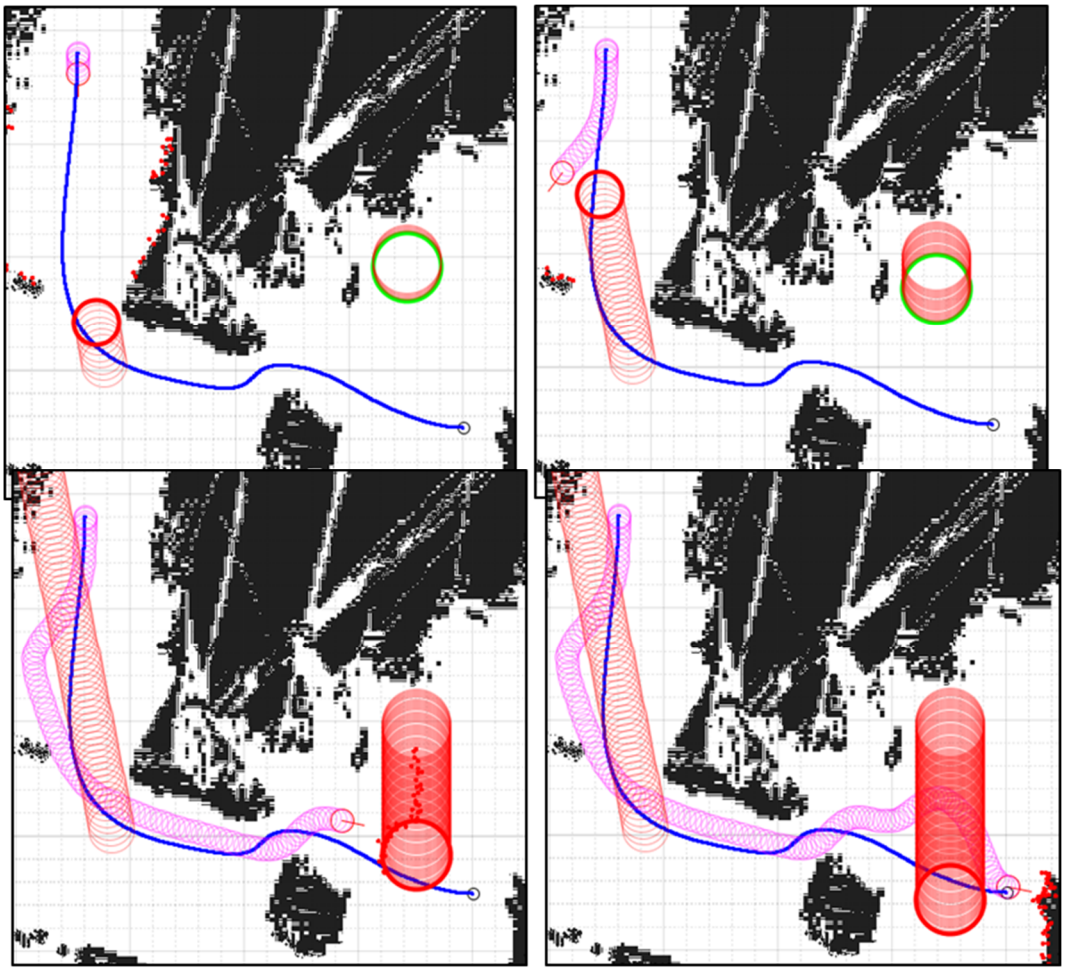

Figure 23. Reference trajectory of the robot is the blue curve, detected obstacle has red circular boundary, non-observed obstacle has green circular boundary.

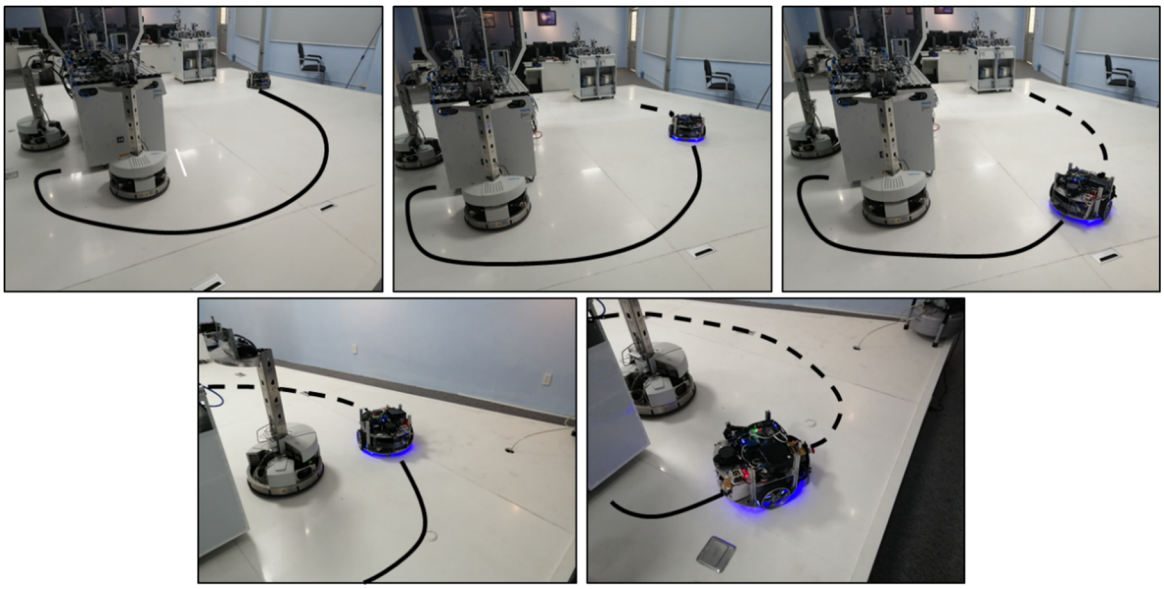

Figure 24. Snapshot series of the trajectory tracking performance of the robot in workspace.

9. Conclusions

In this study, we proposed series algorithms for navigating the autonomous service robot in an indoor environment in the presence of moving obstacles. These methods meet the requirements of real-time performance, flexibility. In tracking trajectory task, instead of doing the velocity plan for the robot, we present the reshape trajectory method. That method can perform in real time because it very flexible and does not cause many changes in curvature of the entire trajectory. In moving obstacle-avoidance task, our algorithm performs very well in many cases. However, in some specific cases, the robot is easily stuck in local minimum when optimizing WSM. The proposed algorithm is not strictly time-optimal or distance-optimal because the path constructed by the solution extracted from WSM is based on weight values that subjectively evaluated which makes it almost impossible to have the optimal path.

For a better version, further works may include the following topics:

- Using neural networks to evaluate the weighted values of WSM;
- In the path-planning algorithm, the final path does not have enough space for construct the Bézier curve in some specific case when path is constructed in a narrow area. We proposed the bubble generation method that same as in the [20]. They can easily constrain the extension of the Bézier curve around the original path;
- Restricting the maximum velocity when the robot encounters sections of trajectory with large curvature.

Author Contributions: Conceptualization, N.T.T.; Formal analysis, P.G.L.; Investigation, P.G.L.; Methodology, N.T.T.; Project administration, N.T.T.; Supervision, N.T.T.; Visualization, P.G.L.; Writing–original draft, P.G.L.; Writing–review & editing, N.T.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by Ho Chi Minh City University of Technology and Education, Vietnam.

Conflicts of Interest: The authors declare no conflict of interest

References

1. Coste-Manière, È.; Simmons, R. Architecture, the Backbone of Robotic Systems. In Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, CA, USA, 24–28 April 2000.
2. Li, G.; Yamashita, A.; Asama, H.; Tamura, Y. An efficient improved artificial potential field based regression search method for robot path planning. In Proceedings of the 2012 IEEE International Conference on Mechatronics and Automation, Chengdu, China, 5–8 August 2012; IEEE: Piscataway, NJ, USA, 2012.
3. Taharwa, A.; Alweshah, M.; Al-Taharwa, I.; Sheta, A. A mobile robot path planning using genetic algorithm in static environment. *J. Comput. Sci.* **2008**, *4*, 341–344. [CrossRef]
4. Saska, M.; Macas, M.; Přeučil, L.; Lhotská, L. Robot path planning using particle swarm optimization of Ferguson splines. In Proceedings of the 2006 IEEE Conference on Emerging Technologies and Factory Automation, Prague, Czech Republic, 20–22 September 2006; IEEE: Piscataway, NJ, USA, 2006.
5. ElHalawany, B.M.; ABDEL-Kader, H.M.; Eldien, A.S.T.; Elsayed, A.E.; Nossair, Z.B. Modified a* algorithm for safer mobile robot navigation. In Proceedings of the 2013 5th International Conference on Modelling, Identification and Control (ICMIC), Cairo, Egypt, 31 August–2 September 2013; IEEE: Piscataway, NJ, USA, 2013.
6. Sudhakara, P.; Ganapathy, V. Path Planning of a Mobile Robot using Amended A-Star Algorithm. *Int. J. Control Theory Appl.* **2016**, *9*, 489–502.
7. Wu, M.; Dai, S.-L.; Yang, C. Mixed Reality Enhanced User Interactive Path Planning for Omnidirectional Mobile Robot. *Appl. Sci.* **2020**, *10*, 1135. [CrossRef]
8. Kazem, B.; Hamad, A.H.; Mozael, M. Modified vector field histogram with a neural network learning model for mobile robot path planning and obstacle avoidance. *Int. J. Adv. Comput. Technol.* **2010**, *2*, 166–173.
9. Yamada, T.; Sa, Y.L.; Ohya, A.; Yamada, T. Moving obstacle avoidance for mobile robot moving on designated path. In Proceedings of the 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, Korea, 30 October–2 November 2013; IEEE: Piscataway, NJ, USA, 2013.

10. Ferrara, A.; Rubagotti, M. Sliding mode control of a mobile robot for dynamic obstacle avoidance based on a time-varying harmonic potential field. In Proceedings of the ICRA 2007 Workshop: Planning, Perception and Navigation for Intelligent Vehicles, Rome, Italy, 14 April 2007; Volume 160.
11. Sharma, K.D.; Chatterjee, A.; Rakshit, A. A PSO–Lyapunov hybrid stable adaptive fuzzy tracking control approach for vision-based robot navigation. *IEEE Trans. Instrum. Meas.* **2012**, *61*, 1908–1914. [CrossRef]
12. Mitrović, S.T.; Djurovic, Z.M. Fuzzy-Based Controller for Differential Drive Mobile Robot Obstacle Avoidance. In Proceedings of the 7th IFAC Symposium on Intelligent Autonomous Vehicle, Lecce, Italy, 6–8 September 2010; Volume 7.
13. Kong, H.; Yang, C.; Li, G.; Dai, S.-L. A sEMG-Based Shared Control System With No-Target Obstacle Avoidance for Omnidirectional Mobile Robots. *IEEE Access* **2020**, *8*, 26030–26040. [CrossRef]
14. Kunchev, V.; Jain, L.C.; Ivancevic, V.G.; Finn, A. Path planning and obstacle avoidance for autonomous mobile robots: A review. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*; Springer: Berlin/Heidelberg, Germany, 2006.
15. Sgorbissa, A.; Zaccaria, R. Planning and obstacle avoidance in mobile robotics. *Robot. Auton. Syst.* **2012**, *60*, 628–638. [CrossRef]
16. Duchon, F.; Babinec, A.; Kajan, M.; Beno, P.; Florek, M.; Fico, T.; Jurišica, L. Path planning with modified a star algorithm for a mobile robot. *Procedia Eng.* **2014**, *96*, 59–69. [CrossRef]
17. Simons, D.P.; Scott, S.S. Arc-length reparameterization. U.S. Patent No. 6,115,051, 5 September 2000.
18. Hwang, J.-H.; Arkin, R.C.; Kwon, D.-S. Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No. 03CH37453), Las Vegas, NV, USA, 27–31 October 2003; IEEE: Piscataway, NJ, USA, 2003; Volume 2.
19. Yang, X.-S. *Nature-inspired Optimization Algorithms*; Elsevier: Amsterdam, The Netherlands, 2014.
20. Zhu, Z.; Schmerling, E.; Pavone, M. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In Proceedings of the 2015 54th IEEE Conference on Decision and Control (CDC), Osaka, Japan, 15–18 December 2015; IEEE: Piscataway, NJ, USA, 2015.