Motion Planning of Differential Drive Mobile Robot Using Circular Arc

Wan Zafira Ezza Wan Zakaria, Ahmad Ramli*, Md Yushalify Misro and Mohd Nadhir Ab Wahab

***Abstract*—In modern applications, mobile robots are important dynamic systems. Motion planning is a crucial responsibility for these robots as it enables them to move from one place to another safely and efficiently. Extensive research has been conducted on motion planning for static scenarios. However, research is limited when the scenarios are dynamic because it is difficult and expensive to continuously replan the robot's moves to ensure its safety. This paper presents the circular arc path algorithm for sensorless differential drive mobile robot movement, given that the initial point and endpoint are defined. The differential drive mobile robot, which is adopted for various vehicles and robots, is very popular due to its simple operation and robustness. A mobile robot with two driving wheels that uses a differential drive splits its overall velocity between its left and right wheels. Accordingly, the radius of the circle that forms after movement between two points is the main factor that determines the left and right velocity of the robot's wheels. Through this algorithm, all points will be interpolated with G^1 continuity, which only requires an initial direction to move. Robot simulation is performed using CoppeliaSim, and the algorithm is developed using Remote API functions in Matlab.**

***Index Terms*—differential drive, circular arcs, robot simulation, motion planning, mobile robot**

I. Introduction

MOTION planning, in the context of robotics and autonomous systems, refers to the process of determining a sequence of motions or actions that enable a robot or vehicle to move from an initial state to a desired goal state while satisfying various constraints and objectives [1]. Motion planning algorithms take into account the kinematics and dynamics of the robot, as well as factors such as velocity limits, accelerations, and control inputs. The goal of motion planning is to determine the appropriate control commands or control signals that enable the robot to execute the planned path while satisfying the physical constraints and operational requirements. It plays a fundamental role in various applications, including robot manipulation such as in [2], autonomous navigation, unmanned aerial vehicles (UAVs), self-driving cars, industrial automation, and many others. It enables robots and autonomous systems to plan and execute their motions effectively, ensuring efficient and safe interactions with the surrounding world.

A differential drive mobile robot is a drive system in which each side of the vehicle's wheels or tracks is driven separately, and turning is accomplished by driving the left and right wheels at separate velocities [3]. The popularity of this driving system stems from its simplicity (no explicit turning mechanism) and agility (it can turn in extremely small radii of curvature). It has been widely used in a range of robot simulations for motion planning. In general, there are two types of controllers for mobile robots: kinematic controllers, which govern the driving wheel velocities for mobile robots, and dynamic controllers, where the control command is the driving torques. A kinematic controller for a two-wheel mobile robot with a differential drive is presented in [4].

The majority of current differential drive mobile robots are equipped with various types of sensors, including cameras (or more generally, visual sensors) [5], [6], sonar sensors [7], Light detecting and ranging (LiDAR) sensors [8] and tactile sensors [9], among others, to observe the environment. However, for our research, we solely rely on CAGD (Computer-Aided Geometric Design) knowledge without the assistance of sensors for the study of motion planning. In traditional motion planning, it is typically assumed that we have knowledge, at least partially, of the pose (position and orientation) of the target and/or the robot.

Any motion controller would be incomplete without a circular motion controller. The circular interpolation controller drives two motors with changing velocities and directions based on the quadrant of motion to achieve a circular movement from the start to end coordinates. A succession of short steps is used to imitate circular motion [10]. Most circular interpolation methods utilize parametric sine and cosine functions for the necessary calculations. Parametric functions are necessary due to their high level of numeric precision, but they also require additional hardware and time to be applicable in real-time applications.

Research on the topic of smooth circular arc interpolation for parametric curves has been active in recent years. A novel two-stage interpolation algorithm is proposed in [11], with the first stage being rough interpolation without linear approximation and the second stage being circular arc fine interpolation, which is an improvement above traditional interpolation techniques. The suggested approach has nearly little theoretical error, according to simulation findings. Due to the linear approximation error, a contradiction between velocity and accuracy occurred.

The problem of constructing circular motion from one point to another point in Cartesian space (x, y, z), which entails the robot's travel from the beginning to the final

Manuscript received April 13, 2023; revised October 12, 2023.

This paper was supported by Ministry of Higher Education Malaysia through the Fundamental Research Grant Scheme with Project Code: FRGS/1/2020/STG06/USM/02/4

W.Z.E Wan Zakaria a lecturer at the School of Industrial Technology, Universiti Sains Malaysia (USM), 11800 Pulau Pinang, Malaysia. e-mail: (ezzafira@usm.my).

A. Ramli is an Associate Professor at the School of Mathematical Sciences, Universiti Sains Malaysia (USM), 11800 Pulau Pinang, Malaysia. (Corresponding Author; phone: +604-653-4773; fax: +604-657-0910; e-mail: alaramli@usm.my.

M. Y. Misro is a lecturer at the School of Mathematical Sciences, Universiti Sains Malaysia(USM), 11800 Pulau Pinang, Malaysia. e-mail: (yushalify@usm.my).

M. N. Ab Wahab is a lecturer at the School of Computer Sciences, Universiti Sains Malaysia (USM), 11800 Pulau Pinang, Malaysia. e-mail: (mohdnadhir@usm.my).

position, was investigated by [12]. The difficulty of creating point-to-point circular motion in multidimensional space was also explored. However, the point-to-point (PTP) method in Cartesian space is used to generate a circular pathway, which becomes computationally burdensome as the number of PTP increases; more time will be required. On the other hand, the smoothness of the circular path improves when there are more points. A parametric equation of a circle is utilized to achieve circular motion.

The topic of producing a trajectory capable of avoiding probable impediments in a blending event with a non-holonomic assumption is the focus of [13]. This study proposes a method that combines a basic circular path tool with a corridor generation algorithm. It places the problem in a workspace where a blending trajectory is needed to link tangent routes and avoid obstructions. To address this issue, the system attempts to create a collision-free corridor that can serve as a safe path-taking region.

In order to execute an ideal smooth path and enable smooth turning of the robot movement, [14] introduces new modified circular arcs to replace path segment joints. One of the most widely used applications for calculating navigational paths is the Probabilistic Roadmap (PRM), which is employed in their approach. The arc fillet approach presented in this study is particularly strong for smoothing the robot's navigational path, giving it a unique characteristic.

On the other hand, [15] introduces a novel concept for creating a line-following algorithm for a mobile differential drive robot. The paper explains in detail how the radius of the path's curvature is determined using geometry, and based on that information, the differential drive kinematics establish the desired difference between the angular velocities of the two wheels. This robot measures how much a circular arc deviates from a straight line using a variety of IR-reflecting sensors. However, our study does not involve any sensors, as the differential drive mobile robot will move based on the velocity of the two wheels and time.

In this research, we propose a circular-arc-based approach for point interpolation in route planning of sensorless differential drive mobile robots, building upon the work of [16]. The focus is on finding a path using two circular arcs, considering that the angles of the start and endpoint are different. Thus, a path based on circular arcs that interpolate two or more points with G^1 continuity is constructed, referring to [17], where a smooth path consisting of a linear path and a circular arc path with G^1 continuity, tolerance restrictions, shape preservation, and computational efficiency is formed through simulations and real robot experiments. We also propose an algorithm to compute the velocities of the wheels and the travel time based on curvature [18]. This approach is implemented and evaluated on a simulator to observe its accuracy and efficiency. Experimental results have been applied to the Pioneer 3-DX mobile robot in CoppeliaSim.

II. Mobile Robots

This section shows information about the mobile robot used in simulation with CoppeliaSim. The components to move the mobile robot are discussed.

A. Pioneer 3-DX

Pioneer 3-DX is a two-wheel, two-motor differential drive robot that is excellent for use in a laboratory or classroom. The robot comes with a front SONAR, a single battery, and wheel encoders. Pioneer research robots are the most widely used intelligent mobile robots in education and research across the world [19].

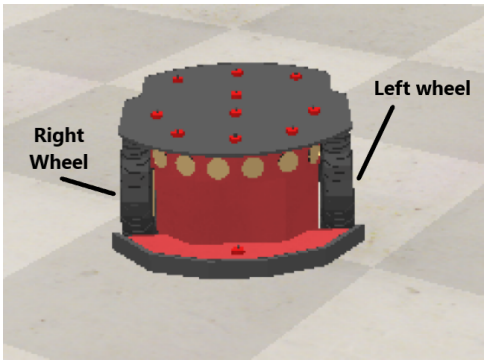


Fig. 1: Pioneer 3-DX simulated model.

Figure 1 depicts the simulated Pioneer 3-DX, which was utilized in our CoppeliaSim simulation. The Pioneer 3-DX is equipped with two wheels that control the robot's motion. Varied velocities applied to the wheels result in different directions and operations. The robot's movement is generated using the remote API function in MATLAB, establishing a connection with the robot. Table I presents the robot's motion data based on the velocities of its wheels.

TABLE I: The movement of pioneer 3-DX robot

Velocity of left wheel, V_L **and right wheel,** V_R	**Operations**
$V_L < V_R$	Counter-clockwise circular movement
$V_L > V_R$	Clockwise circular movement
$V_L = V_R = 0$	Stop

Note that the circular movement creates a certain radius based on the speed. If we wish for the robot to rotate at a particular point without creating a circular arc, we can let $V_L = -V_R$ whereas V_L and V_R are the velocity of left and right wheels respectively.

B. Differential drive mobile robot

The differential drive robot consists of a platform equipped with a front caster and a pair of rear coaxial drive wheels for isotatic equilibrium. Each of these drive wheels is independently driven by a DC motor, which is in turn energized by a control voltage. By varying the power applied to the motors, the differential-wheeled mobile robot can be made to move in a straight line or trace different trajectories such as curves and circles. Deriving a precise mathematical model is a crucial part of designing any control system [20].

The kinematics concept of a differential drive mobile robot is shown in Figure 2. To generalize this approach to any robot with two wheels, we would not be using any additional features such as sensors. The following model assumptions are taken into account: (1) The robot's mass center is at the geometric center of the body frame. (2) The robot moves

without depending on sensors. (3) The robot is moving on a solid ground surface with two wheels that are always in contact with it.

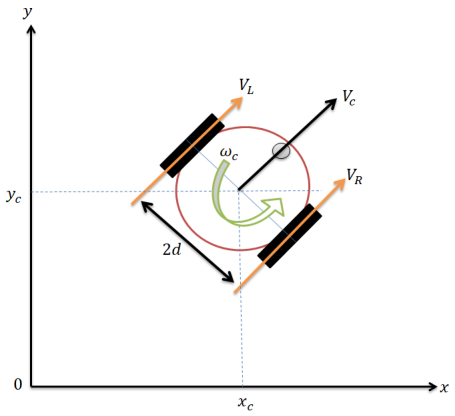


Fig. 2: The kinematics schematic of differential drive mobile robot (This figure is redrawn and simplified from [20]. Permission has been obtained)

Let (x,y) represent the coordinates of the robot platform's center of mass, and θ represents the angle between the direction of the robot's travel and the x-axis. The angular velocities of the wheels are determined by the robot center's linear and angular velocities, represented by V_c and $\omega_c = \theta$ respectively [21]. The wheels are separated by a distance of 2d. Different velocities in the wheels will result in different directions and operations. The movement is generated by the remote API function in MATLAB, which is connected to the robot.

III. Methodology Used and Results

A. Circular Arcs Interpolation

The trajectory of a point moving in the plane with coordinates $C(t) = (x(t), y(t))$, where $x(t)$ and $y(t)$ are functions of the parameter t, is known as a parametric curve. For each value of t, we obtain a point on the curve.

In this paper, we utilize a circular arc path. The parametric equations for a circle of radius r and center (a, b), representing the functions $x(t)$ and $y(t)$ respectively, are given by:

$$\begin{aligned} x(t) &= a + r\cos t \\ y(t) &= b + r\sin t \end{aligned} \tag{1}$$

and its first derivative is given by

$$\begin{aligned} x'(t) &= -r\sin t \\ y'(t) &= r\cos t \end{aligned} \tag{2}$$

So, given any two points, $P_1 = (p_{x,1}, p_{y,1})$ and $P_2 = (p_{x,2}, p_{y,2})$, we aim to interpolate these points with a specified tangent direction at P_1, denoted as $C_1'(t_1) = (d_{x,1}, d_{y,1})$.

By substituting the values into Equation 1, we can determine the radius (r), centers (a and b), and phases (t_1 and t_2). The equation will then become:

$$\begin{aligned} p_{x,1} &= a + r\cos t_1 \\ p_{y,1} &= b + r\sin t_1 \\ p_{x,2} &= a + r\cos t_2 \\ p_{y,2} &= b + r\sin t_2 \\ d_{x,1} &= -r\sin t_1 \\ d_{y,1} &= r\cos t_1 \end{aligned} \tag{3}$$

Note that there are only 5 parameters to be solved: a, b, r, t_1, and t_2. However, there are 6 equations in Equation 3. Therefore, we transform the coordinate system by rotation such that either $d_{x,1}$ or $d_{y,1}$ equals zero. By doing so, we can eliminate one of the last two equations. This is because $\sin 0 = 0$ when the direction is on the y-axis, and $\cos\frac{\pi}{2} = 0$ when the direction is on the x-axis.

We demonstrate our algorithm by interpolating the first three points using a circular arc, as shown in Figure 3. To simplify our discussion, we intentionally choose the robot's direction to be facing the y-axis and the x-axis. However, we will generalize it in the later section.

We propose Algorithm 1 to interpolate 3 points using a circular arc, as shown in Figure 3. This algorithm can be easily extended to include more points by repeating the iteration. The notations used in Algorithm 1 are as follows: $C_i = (x(t_{i,j}), y(t_{i,j}))$, where i denotes the i-th curve and j denotes the j-th point. Specifically, when $j = 1$, it represents the starting point, and when $j = 2$, it represents the end point. We will demonstrate the results for more points in a later section.

In the algorithm, we emphasize the need to rotate the points to reduce Equation 3 to 5 equations. This can be achieved using matrix transformation rules by transitioning the point to the origin and rotating it to either the x-axis or the y-axis. The rotation in polar coordinate form can be referred to in [22].

Technically, the robot can stop at the endpoint of the first curve and start with different velocities for the next curve. Therefore, G^1 continuity [23] would be sufficient. Algorithm 1 only requires incorporating the initial direction into the movement.

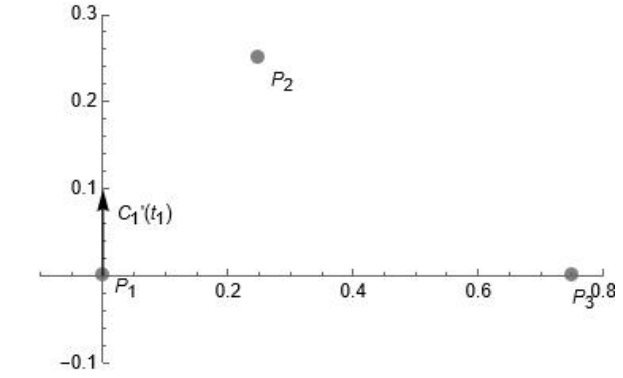
Each circular arc is connected by G^1 continuity, where the endpoint direction of the first curve, $C_1'(t_2) = (d_{x,2}, d_{y,2})$, is equal to the start point direction of the second curve, $C_2'(t_1) = (d_{x,1}, d_{y,1})$. The figure below illustrates the interpolation of 2 circular arcs.

B. Velocity and time for differential drive mobile robot simulation

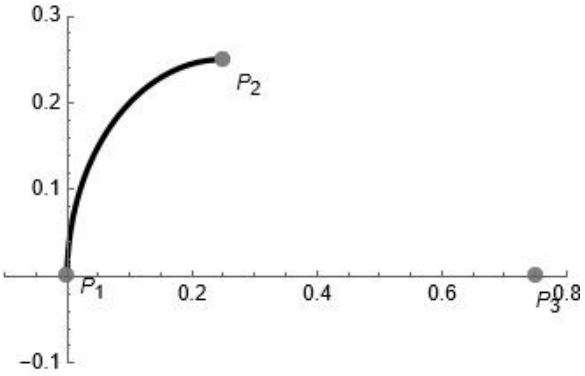
It is computationally easier to construct a path that interpolates the desired points. However, applying this information to a robot simulator requires further assessment. Assuming that a robot is facing the initial direction, it will move to the next point by traversing the circular arc with a specific radius. The radius can be controlled by adjusting the velocities of its left and right wheels.

Suppose the relationship between V_L and V_R and the center velocity is as follows:

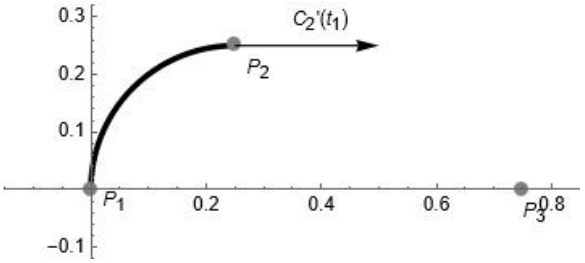
$$V_c = \frac{(V_L + V_R)}{2} \tag{4}$$



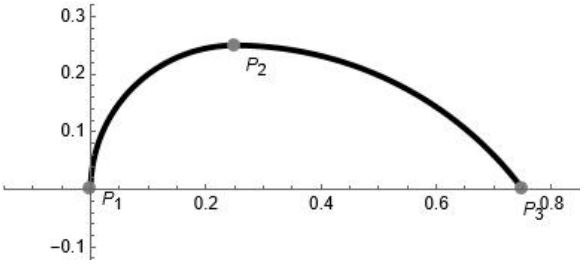
(a) Three points and the first direction



(b) First Circular arc.



(c) Next direction.



(d) Two circular arc.

Fig. 3: Illustration of circular arc interpolation.

When turning clockwise,

$$\begin{aligned} V_L &= V_c + d\omega_c \\ V_R &= V_c - d\omega_c \end{aligned} \tag{5}$$

When turning counter-clockwise,

$$\begin{aligned} V_L &= V_c - d\omega_c \\ V_R &= V_c + d\omega_c \end{aligned} \tag{6}$$

By referring to [18], where they evaluate the Bezier curve formula and the curvature, we implement it into a circular arc to express the linear velocity, V_c, and angular velocity.ω_c as follows:

Algorithm 1 Circular Arc Interpolation

Input

- `Three points,` $P_1 = (p_{x,1}, p_{y,1})$, $P_2 = (p_{x,2}, p_{y,2})$ `and` $P_3 = (p_{x,3}, p_{y,3})$
- `The direction where the robot is facing,` $C_1'(t_1) = (d_{x,1}, d_{y,1})$

Output

- `Radius,` r
- `Center,` (a, b)
- `Direction,` $C_1'(t_2) = (d_{x,2}, d_{y,2})$
- `Phase` t_1 `and` t_2

1: Step 1 (As shown in Figure 3a) : Substitute $(p_{x,1}, p_{y,1})$, $(p_{x,2}, p_{y,2})$ and $(d_{x,1}, d_{y,1})$ into Equation 3. Note that in this example, $d_{x,1} = 0$. Solve the first four equations and the last equation to obtain the center, phase and radius of a circular arc that interpolates both points.
2: Step 2 (As shown in Figure 3b)) : First circular arc, $C_1(t)$ is obtained.
3: Step 3 (As shown in Figure 3c): Set the direction of the next curve $C_2'(t_1) = C_1'(t_2)$. If either $(d_{x,2}, d_{y,2})$ is equal to 0, we may proceed. Otherwise, rotate the whole system to the x-axis or y-axis for computation purposes.
4: Repeat Step 1.
5: Step 4 (As shown in Figure 3d): Next circular arc, $C_2(t)$ is obtained.

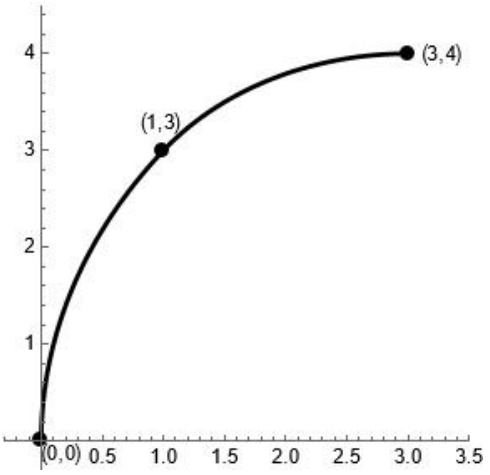


Fig. 4: Circular interpolation of 2 circular arcs.

$$V_c = \sqrt{(x'[A])^2 + (y'[A])^2} \tag{7}$$

and

$$\omega_c = \frac{x'[A]y"[A] - y'[A]x"[A]}{(x'[A])^2 + (y'[A])^2} \tag{8}$$

We choose $d = 0.1905$ referring to the specifications of the Pioneer 3DX robot [19]. The time, t, for each velocity is determined as follows:

$$t = \frac{V_c}{S} \tag{9}$$

where S is the Euclidean distance between two points.

C. Robot simulation

In this part, we present a robot simulation as a continuation of the output in Algorithm 1. The simulation is run on a computer with an Intel(R) Core(TM) i3-8100 CPU @ 3.60GHz 3.60 GHz. Since Algorithm 1 produces the radius for each circular arc, we substitute different values of linear and angular velocity, V_c and ω_c respectively, into Equations 9 and 5 to obtain the velocities for both wheels.

We implement this simulation on CoppeliaSim by providing the values of V_L, V_R, and time corresponding to its speed in Matlab. The motivation behind our simulation is to observe whether the robot can move on the projected path without the assistance of any additional elements, such as a sensor.

We separate our simulation into 5 cases. Each case show a different points for the robot to pass through as follows:

- Case 1, the robot need to pass through 2 points which are $P_1 = (0, 0)$ and $P_2 = (1, 3)$.
- Case 2, the robot need to pass through 3 points which are $P_1 = (0, 0)$, $P_2 = (1, 3)$ and $(2, 6)$.
- Case 3, the robot need to pass through 3 points which are $P_1 = (0, 0)$, $P_2 = (1, 3)$ and $(3, 4)$.
- Case 4, the robot need to pass through 3 points with larger radius on the second arc which are from $P_1 = (0, 0)$, $P_2 = (2, 0.5)$ and $(4, 1)$.
- Case 5, the robot need to pass through 4 points which are $P_1 = (0, 0)$, $P_2 = (1, 3)$, $(2, 6)$ and $(4, 9)$.

In Figure 5, we show the path generated from the simulation and compare it with the interpolated curve obtained from Algorithm 1. It is important to note that the interpolated curve is constructed from a circular arc.

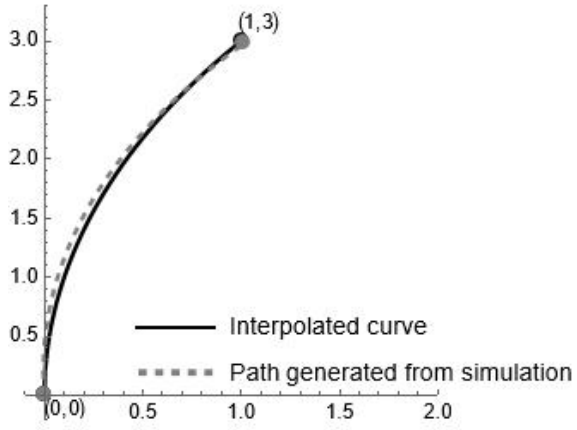
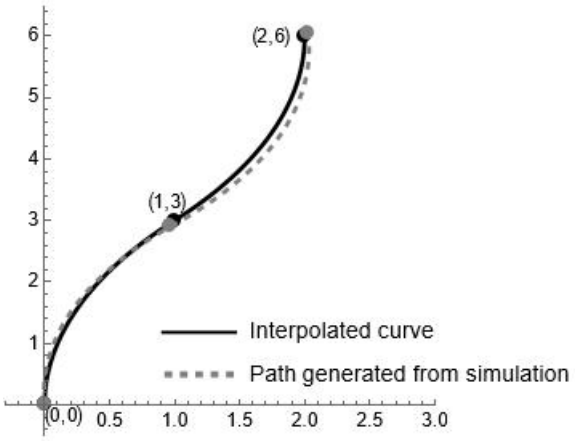


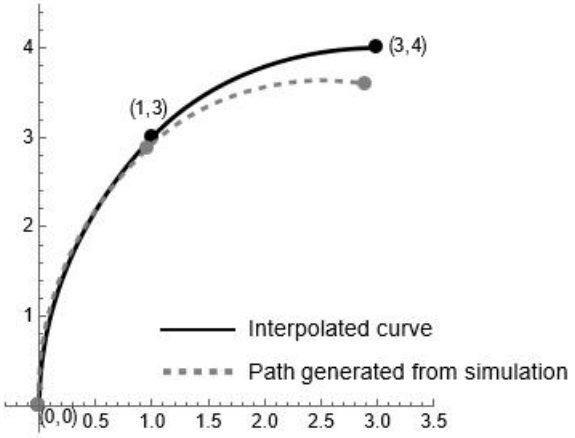
Fig. 5: Case 1: One segment of circular arc

The robot simulation in Figure 5 moves from $(0, 0)$ to $(1, 3)$ in the right direction. According to Algorithm 1, we obtained a radius of $r = 5$. We computed the velocities and obtained $V_L = 5.1905$ and $V_R = 4.8095$. The time required to complete the circular arc in the simulator is 247.56s. As shown in the figure, the path generated from the simulation closely follows the expected interpolated curve. The simulation endpoint is (1.0160, 2.981), while the endpoint from the interpolated curve is (1.0000,3.0000), resulting in an error of 0.0248.

Next, we proceed to run the robot simulation for the second circular arc. We set two different endpoints: $(2, 6)$ for



(a) Case 2: The second circular arc moves counter-clockwise from $(1, 3)$ to $(2, 6)$



(b) Case 3: The second circular arc moves clockwise from $(1, 3)$ to $(3, 4)$

Fig. 6: Comparison of interpolated and path generated from the simulation

counterclockwise motion and $(3, 4)$ to continue the clockwise motion. The results are shown in Figure 6.

The red and green paths in Figure 6 represent the paths generated from the simulation. The different directions of the second segment of the circular arc are shown for comparison of the simulation results. Table II presents the list of velocities used for the left and right wheels, V_L and V_R, for the red path in Figure 6a.

TABLE II: Velocity left and right wheels, V_L and V_R and time for path Figure 6a

Points	V_L	V_R
$(0, 0)$ to $(1, 3)$	5.1905	4.8095
$(1, 3)$ to $(2, 6)$	4.8905	5.1905

Figure 6a displays the interpolated curve with different directions continuing the first segment of the circular arc from Figure 5. The path generated from the simulation (red path) in Figure 6a closely follows the interpolated curve, with an error difference of less than 0.5 as shown in Table III. The total time taken to complete the entire path is 495.055s.

Table IV presents the list of velocities used for the left and right wheels, V_L and V_R, for the red path in Figure 6b.

TABLE III: The comparison error between interpolated and simulation coordinate for path Figure 6a

Points	Interpolated coordinate	Simulation coordinate	Error
P_1	$(1, 3)$	$(0.9708, 2.9056)$	0.0988
P_2	$(2, 6)$	$(2.0238, 6.0354)$	0.0426

TABLE IV: Velocity left and right wheels, V_L and V_R and time for path Figure 6b

Points	V_L	V_R
$(0, 0)$ to $(1, 3)$	5.1905	4.8095
$(1, 3)$ to $(3, 4)$	2.6905	2.3095

Figure 6b depicts the interpolated curve with the same directions continuing the first segment of the circular arc from Figure 5. The path generated from the simulation (green path) for Figure 6b closely follows the interpolated curve in the first segment, with a small error difference as shown in Table V. However, in the second segment, it deviates slightly in the right direction before continuing along the intended path. The total time taken to complete the entire path is 464.77s.

TABLE V: The comparison error between interpolated and simulation coordinate for path in Figure 6b

Points	Interpolated coordinate	Simulation coordinate	Error
P_1	$(1, 3)$	$(0.8894, 2.7591)$	0.1221
P_2	$(3, 4)$	$(2.9007, 3.5996)$	0.4125

Furthermore, our algorithm was applied to two circular arcs, each exhibiting distinct characteristics: the first arc displays a clearly defined circular shape, whereas the second arc appears to resemble a straight line from its second point. Technically, the second arc is still a circular arc but has an extremely large radius. The results are shown in Figure 7.

The red and blue paths in Figure 7 represent the paths generated from the simulation. Table VI presents the list of velocities used for the left and right wheels, V_L and V_R, for the red path in Figure 7.

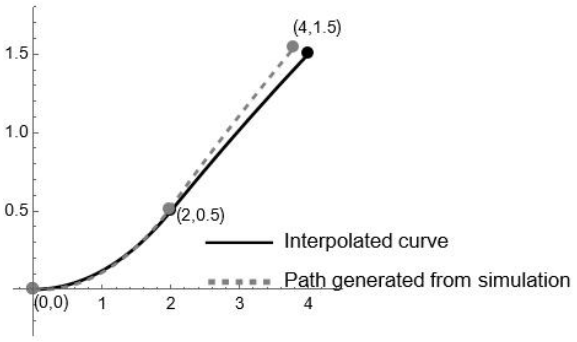


Fig. 7: Case 4: Two segments of circular arcs. The second arc exhibits a straight line behavior but is represented by a large radius.

TABLE VI: Velocity left and right wheels, V_L and V_R and time for path Figure 7

Points	V_L	V_R
$(0, 0)$ to $(2, 0.5)$	4.0595	4.4405
$(2, 0.5)$ to $(4, 1.5)$	37.9008	37.5198

In Table VI, we can see that the velocity V_L and V_R are almost equal as it traverse a straight line.

As anticipated, the robot adhered quite faithfully to the trajectory of the first arc. However, it exhibited a slight deviation when traversing the second arc. The error at the end for the second arc is 0.2086 as shown in Table VII.

As shown in previous cases, the error propagates once the robot traverses the second arc.

TABLE VII: The comparison error between interpolated and simulation coordinate for path in Figure 7

Points	Interpolated coordinate	Simulation coordinate	Error
P_1	$(2, 0.5)$	$(1.9771, 0.5105)$	0.0251
P_2	$(4, 1.5)$	$(3.7957, 1.5428)$	0.2086

Next, we tested our algorithm on 3 circular arcs. Figure 8 illustrates the interpolated curve and the path obtained from the robot simulation.

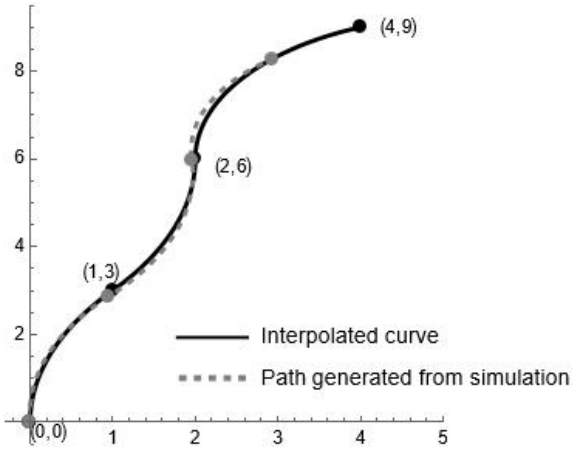


Fig. 8: Case 5: Three segments of circular arc.

The error for each point is calculated as shown in Table VIII.

TABLE VIII: The comparison error between interpolated and simulation coordinate for path Figure 8

Points	Interpolated coordinate	Simulation coordinate	Error
P_1	$(1, 3)$	$(0.9434, 2.8573)$	0.1535
P_2	$(2, 6)$	$(1.9636, 5.9681)$	0.0484
P_3	$(4, 9)$	$(2.9276, 8.2750)$	1.2944

Notably, it's observed that the simulated path terminates before reaching the endpoint indicated in the interpolated curve. The error, following the utilization of three circular arcs, is recorded at 1.2944. The entirety of the simulation process required a total time of 790.66 seconds.

IV. Error Analysis

In this section, we analyze the errors in our simulation. In the previous section, we demonstrated the simulation for a specific run. However, each time a simulation is conducted, the results may vary due to factors such as the computer's capacity at that particular moment.

We analyze the errors at the endpoints for three different cases: one segment, two segments, and three segments of circular arcs. Respectively for each case, we have selected Case 1 (as in Figure 5), Case 2 (as in Figure 6), and Case 5 (as in Figure 8) for further analysis. We run these three cases 30 times in the simulation to observe the error pattern.

Figure 9 shows the histogram of error versus frequency between interpolation and simulation for Case 1: one circular arc in Figure 5.

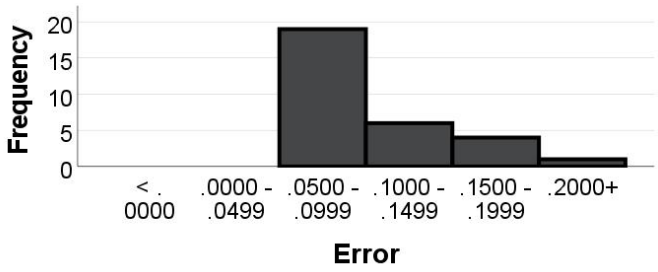


Fig. 9: The histogram of simulation error for Case 1.

Next, we run another 30 simulations for Case 2: two circular arcs in Figure 6, and the error frequency is shown in Figure 10.

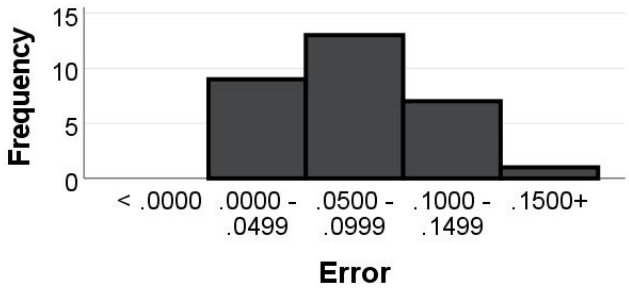


Fig. 10: The histogram of simulation error for Case 2.

We then continue with Case 5 in Figure 8, involving three circular arcs, and the histogram of error is shown in Figure 11.

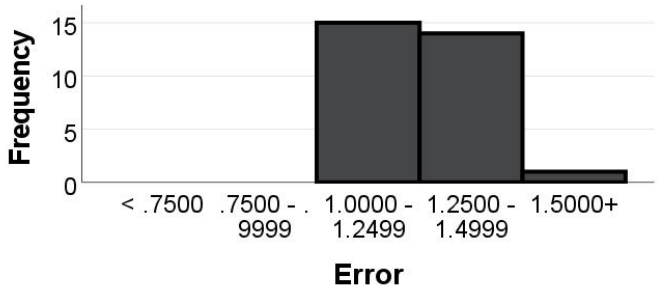


Fig. 11: The histogram of simulation error for Case 5.

Table IX shows the errors and variances for each cases. As we observed previously, the error increases when the number of arcs increases. However, for all cases, the variances are low (<0.02), indicating that the simulation remains consistent even after multiple runs.

TABLE IX: The maximum, minimum, average and variance of error for Case 1 (5), Case 2 (6), and Case 5 (8)

Case	Minimum	Maximum	Average	Variance
Case 1	0.0679	0.2063	0.1077	0.001
Case 2	0.0231	0.1540	0.0741	0.001
Case 5	1.0226	1.7480	1.2225	0.018

V. Discussion

During the development and experimental path used in the simulation, we encountered difficulties in finding suitable velocities and times for each wheel traveled by the robot in the simulator. It is important to note that a real robot would require a different approach to determine the velocity and time solutions. This challenge can be addressed by analyzing the relationship between velocity, time, and the curvature of a circle, as outlined in [18]. By using this approach, the path can be simulated with the required velocity and time for each circular arc connection, without the need for the initial rotation of the robot.

This approach yields similar paths in both interpolation and simulation. The small error between the interpolated points and the points resulting from the robot simulation for two circular arcs with different directions confirms this. However, when using a path with two circular arcs in the same direction or three circular arcs, we observed that the slight movement of the robot stopping before continuing to the next circular arc has an impact on the robot simulation's ability to reach the desired destination.

From the robot simulation, it is generally observed that the robot closely follows our interpolated curve. However, the error may increase when multiple arcs are used. This may be caused by skidding [24].

At the end of the first segment in Case 3 (Figure 6b), the movement of the robot appears to deviate. This is due to the fact that the tangent at the endpoint of the first segment during simulation does not align with the expected direction. Since we rely on the initial tangent to be the same as the end tangent of the first segment computationally, any error at the end of the first segment will propagate to the second segment and subsequent segments.

It should be noted that the curve from the simulation may slightly differ each time the simulation is run. We observed that errors may occur for a few reasons. Firstly, when we tested the algorithm on a computer with lower capacity (Intel(R) Celeron(R) N4000 CPU @ 1.10GHz 1.10 GHz), the results showed variations in time and endpoint. Therefore, we deduced that computer capacity may yield slightly different results. However, our calculations in Algorithm 1 remain valid.

Secondly, as our simulation runs on the connection between Matlab and CoppeliaSim simultaneously, there is a possibility of a communication breakdown during the simulation, leading to errors. The connection may be interrupted if there are other ongoing applications running at the same time.

VI. Conclusion

In this paper, we have developed a method for constructing paths using circular arcs in the context of a differential drive mobile robot simulation application. Our approach allows users to generate interpolated paths by providing the initial coordinates and direction while maintaining G^1 continuity between connected curves. The code implementation we have presented is beneficial for users seeking to move their robots along circular trajectories by simply specifying the speed and time for the left and right wheels.

Through our experiments, we have demonstrated that the simulation closely follows the planned curve and successfully interpolates the given points for scenarios involving two and three circular arcs.

There are several directions for future research that we would like to highlight. Firstly, it would be valuable and insightful to implement our code on a real robot to assess any additional limitations or challenges that may arise in a practical setting. This real-world experimentation can provide valuable insights into the applicability and performance of our approach.

Secondly, when constructing paths using two circular arcs, it is important to note that mathematically, there is a lack of curvature continuity between the arcs, which can result in a discontinuity in terms of speed (as discussed in [25], [23], and [26]). Exploring alternative curve types that allow for curvature continuity could potentially lead to smoother robot movements and improve overall path quality.

These areas of future research have the potential to further enhance the practicality and performance of our method, expanding its applicability to real-world scenarios and addressing potential limitations related to curvature continuity.

References

[1] "Motion Planning vs Path Planning — shaderobotics.com," https://www.shaderobotics.com/posts/motion-planning-vs-path-planning, [Accessed 10-Jul-2023].

[2] J. Shen, W. Zhang, Y. Ye, Y. Zhu, and X. Ye, "Adaptive neural network control of space flexible robot based on calculated torque for non-cooperative targets." *IAENG International Journal of Computer Science*, vol. 50, no. 2, pp. 525–536, 2023.

[3] N. Uddin, "A two-wheeled robot trajectory tracking control system design based on poles domination approach." *IAENG International Journal of Computer Science*, vol. 47, no. 2, pp. 154–161, 2020.

[4] A. Nagy, G. Csorvási, and D. Kiss, "Path planning and control of differential and car-like robots in narrow environments," in *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2015, pp. 103–108.

[5] A. Gorbenko and V. Popov, "Visual landmark selection for mobile robot navigation," *IAENG International Journal of Computer Science*, vol. 40, no. 3, pp. 134–142, 2013.

[6] H. H. Triharminto, O. Wahyunggoro, T. B. Adji, A. Cahyadi, I. Ardiyanto *et al.*, "Local information using stereo camera in artificial potential field based path planning," *IAENG International Journal of Computer Science*, vol. 44, no. 3, pp. 316–326, 2017.

[7] P. Paral, S. Ghosh, A. Chatterjee, and S. K. Pal, "Automatic relevance determination kernel-embedded gaussian process regression for sonar based human leg localization with a mobile robot," *IEEE Sensors Letters*, 2022.

[8] J. Iqbal, R. Xu, S. Sun, and C. Li, "Simulation of an autonomous mobile robot for lidar-based in-field phenotyping and navigation," *Robotics*, vol. 9, no. 2, p. 46, 2020.

[9] C. T. Nnodim, A. El-Bab, B. W. Ikua, and D. N. Sila, "Design and simulation of a tactile sensor for fruit ripeness detection," *Proc. World Cong. Eng. Comp. Sci*, vol. 2019, pp. 390–395, 2019.

[10] J. Liao, Z. Chen, and B. Yao, "Adaptive robust control of skid steer mobile robot with independent driving torque allocation," in *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2017, pp. 340–345.

[11] G. Wang and G. Ye, "Novel circular interpolation algorithm for high-accuracy positioning systems," in *2016 IEEE International Conference on Information and Automation (ICIA)*. IEEE, 2016, pp. 222–227.

[12] Z. Zhang, A. Beck, and N. Magnenat-Thalmann, "Human-like behavior generation based on head-arms model for robot tracking external targets and body parts," *IEEE transactions on cybernetics*, vol. 45, no. 8, pp. 1390–1400, 2014.

[13] H. Ren and F. Katsuki, "Circular arc based obstacle avoiding blending trajectory plan," in *2020 5th International Conference on Control and Robotics Engineering (ICCRE)*. IEEE, 2020, pp. 15–18.

[14] M. K. Ouach and T. Eren, "Prm path smoothening by circular arc fillet method for mobile robot navigation," *arXiv preprint arXiv:2112.03604*, 2021.

[15] J. Singh and P. S. Chouhan, "A new approach for line following robot using radius of path curvature and differential drive kinematics," in *2017 6th International Conference on Computer Applications In Electrical Engineering-Recent Advances (CERA)*. IEEE, 2017, pp. 497–502.

[16] M. Effati and K. Skonieczny, "Circular arc-based optimal path planning for skid-steer rovers," in *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*. IEEE, 2018, pp. 1–4.

[17] S. He, Y. Deng, C. Yan, Z. Gao, and C.-H. Lee, "A tolerance constrained robot path circular interpolation method for industrial scara robots," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 235, no. 6-7, pp. 1061–1073, 2021.

[18] E. Broman and I. Rossing, "A system for procedural camera movements for navigation in astrographics," 2020.

[19] I. A. Technology, "Pioneer3-dx," https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf, 2011, [Accessed 10-Jul-2023].

[20] A. Rosales, G. Scaglia, V. Mut, and F. di Sciascio, "Formation control and trajectory tracking of mobile robotic systems–a linear algebra approach," *Robotica*, vol. 29, no. 3, pp. 335–349, 2011.

[21] S. L. Francis, S. G. Anavatti, and M. Garratt, "Real-time path planning module for autonomous vehicles in cluttered environment using a 3d camera," *International Journal of Vehicle Autonomous Systems*, vol. 14, no. 1, pp. 40–61, 2018.

[22] K. Kuttler, Sep 2022. [Online]. Available: https://math.libretexts.org/Bookshelves/Linear_Algebra/A_First_Course_in_Linear_Algebra_(Kuttler)/05\%3A_Linear_Transformations/5.04\%3A_Special_Linear_Transformations_in_R

[23] W. Z. E. W. Zakaria, A. Ramli, and J. M. Ali, "Bezier curves interpolation with end point constraints on road map," in *AIP Conference Proceedings*, vol. 2184, no. 1. AIP Publishing LLC, 2019, p. 060060.

[24] D. Wang and C. B. Low, "Modeling and analysis of skidding and slipping in wheeled mobile robots: Control design perspective," *IEEE Transactions on Robotics*, vol. 24, no. 3, pp. 676–687, 2008.

[25] N. Othman, U. Reif, A. Ramli, and M. Misro, "Manoeuvring speed estimation of a lane-change system using geometric hermite interpolation," *Ain Shams Engineering Journal*, vol. 12, no. 4, pp. 4015–4021, 2021.

[26] Y. Zhang, P. Ye, J. Wu, and H. Zhang, "An optimal curvature-smooth transition algorithm with axis jerk limitations along linear segments," *The International Journal of Advanced Manufacturing Technology*, vol. 95, pp. 875–888, 2018.