Teach and Repeat for Path Planning using Bézier Curves

1st Jardel dos Santos Dyonisio
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0009-0003-7722-7203

2nd Luís Felipe Milczarek
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0009-0008-7693-034X

3rd Nicolas Freitas Dias
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0009-0009-9970-0501

4th Stephanie Loi Brião
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0000-0001-9345-2038

5th Gabriel Amaral Dorneles
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0009-0001-6554-5832

6th Igor Pardo Maurell
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0000-0002-4376-9544

7th Pedro Miranda Pinheiro
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0000-0002-8564-458X

8th Rodrigo da Silva Guerra
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0000-0003-4011-0901

9th Paulo Lilles Jorge Drews-Jr
Center for Computational Science
Univer. Federal do Rio Grande (FURG)
Rio Grande, Brazil
0000-0002-7519-0502

***Abstract*—In the service robots field, a mobile robot must be able to navigate and repeat previously defined paths. This paper aims to develop a technique capable of teaching and repeating (T&R) a path taken by a mobile robot. We evaluated using a differential robot. The developed system is integrated with the ROS framework and uses the mobile base, which is built with self-balancing scooter components. When changes need to be made in the environment, a user must generate a new path by teleoperating the robot, and the system can repeat new paths autonomously. The proposed approach consists of using Bézier curves to represent the paths taken by the user. The proposed method implements a T&R technique to navigate a mobile robot. Results show the worst maximum average of percentage error (MPE) using this method was only 1.361% in adapting the Bézier curve to the points where the robot was taught, satisfactorily representing the path. In the repeating process, the worst MPE was 9.983%.**

***Index Terms*—Teach and repeat, Autonomous Mobile Robot, Planning route.**

I. Introduction

With technological advancements, service robots are utilized in new daily contexts, such as surveillance, inspection, restaurants, and logistics. Their usage spans from residences to industries and hospitals, exemplified by Enchanted Tool's Mirokai[1]. Various platforms like Pal Robotics' TiaGO[2], Toyota's HSR[3], and Softbank's Pepper[4] offer a range of capabilities, each focusing on a specific market segment like social interaction, home care, or business. A robot capable of performing tasks such as transporting objects within a building enables humans to concentrate on higher-level activities, particularly in workplace settings. Thereby alleviating staff burden [9].

One important capability for service robots is autonomous navigation, which involves avoiding collisions and finding the optimal path to perform their tasks. Path planning is a fundamental aspect of mobile robotics in this context, involving determining how to navigate to a target goal within a known or unknown environment. The algorithm must consider variables such as dynamic obstacles and static environment features. Although there are multiple possible approaches to path planning [1], one widely used method in known environments is the Dijkstra algorithm [3]. It calculates the shortest path to a point in space. However, while determining the shortest possible path is often the best solution, in environments with high dynamics and predetermined goal positions, a predefined path may also be an effective solution.

In addition to path planning, local navigation is equally important in mobile robotics, and the dynamic window approach (DWA) [3] local planner is widely adopted for this purpose. Integrated within the ROS Navigation Stack, this planner enables to navigate the environment autonomously.

This work was supported by Brazilian Agencies CNPq, FAPERGS, CAPES, PRH-ANP, and FINEP.

[1]https://enchanted.tools/

[2]https://pal-robotics.com/robots/tiago/

[3]https://global.toyota/en/detail/8709541

[4]https://us.softbankrobotics.com/pepper

Despite their wide application, there are some limitations to saving and repeating a predefined path. While the navigation stack [6] [5] can be used to use each point traversed by the robot as a future objective, creating a continuous curve is important but challenging due to the stack's inability to receive a predefined path.

This paper proposes a teach-and-repeat (T&R) technique based on Bézier curves applied to a low-cost platform [2] with odometry and an inertial measurement unit (IMU). The platform uses an Extended Kalman Filter to fuse both sensors to estimate the robot's pose. The T&R technique involves two main phases: in the teaching phase, an operator manually guides the robot to record and learn the path. In the repeat phase, the robot autonomously executes the task based on the learned data. The entire path is represented by a Bézier curve, which allows for precise path following. This approach ensures that the robot can replicate the taught paths.

II. Related Work

This section provides an overview of existing research and relevant developments. It explores previous work and approaches in mobile robotics, particularly focusing on navigation, path planning, and robot localization methods.

A. Teach and Repeat for Mobile Robots

The work developed by [8] presents a T&R navigation system that utilizes sliding-mode control to address uncertainties such as sensor noise and wheel-terrain interactions. The system relies on odometry and a monocular camera for navigation. While this approach has demonstrated a good performance in trajectory error and stability, achieving an average error of less than 5cm, it differs from our approach by incorporating camera data, adding complexity and reliance on visual input that is not present in our method. In contrast, our work exclusively uses odometry and IMU readings, simplifying the system and reducing reliance on external environmental features.

Another application developed by [14] covers the automation of logistical tasks for small batches and flexible production processes. The emphasis is on creating intuitive and easy-to-use systems, allowing unskilled workers to naturally instruct transport systems. To this end, the authors present a mobile robot with a laser scan to match the robot relative to a taught trajectory, avoiding needing a globally consistent metrical map. This reduces setup complexity and the potential errors from grid maps. Using the Pioneer robot based on odometry, the maximum error was 40%, while using the omniRob robot, the maximum error was 15%. Instead, our work represents the path using Bézier curves, which allow for smooth and continuous path representation.

B. ROS Navigation Packages

The navigation stack is widely used in the robotics field and is a software set that enables robots to navigate in unknown and known environments autonomously. It includes global and local planners, cost maps, and localization, allowing the robot to move on the map. Based on this tool, users can specify a goal for the robot to reach. However, while the user defines the goal, the planner autonomously determines the robot's path to reach that goal. In other words, users may not have direct control over the exact path the robot executes because it is dynamically determined by the planner based on real-time environmental conditions and constraints.

There is a function available that enables the robot to follow waypoints, allowing the user to specify points the robot must reach to conclude that determined path. However, in the same way, the users do not define the entire path or how the path is executed. Instead, the system autonomously determines the most efficient route and execution strategy based on the provided waypoints, optimizing the robot's real-time trajectory.

To address this challenge, the move_base_flex package was developed [11]. This package enables users to convert a set of coordinates into a path and execute them using the mbf_msgs/GetPath and mbf_msgs/ExePath actions. However, this method does not treat the path as a curve nor provide a parameterized curve or enable the curve to avoid obstacles while preserving its essential characteristics.

III. Method

This section presents the T&R process used to enable the robot to navigate along a predefined path autonomously. The method is divided into two main stages: the teaching phase, in which the robot's 2D coordinates are recorded and stored as it is manually guided through the desired path, and the repeating phase, in which the robot uses this information to replicate the path autonomously.

The proposed method adapts a Bézier curve to represent the taught path, generating control points that define the curve corresponding to the path taken. During the repeating phase, n equidistant reference points are established along the Bézier curve, which are used with the local planner's paths. The robot continuously evaluates the set of points, calculates the cost of following each possible path, and selects the local planner path with the lowest cost to guide its movement. As the robot advances and reaches a reference point on the Bézier curve, that point is marked as reached, and a new point is added to the end of the sequence of n points. This process is repeated until the robot has successfully followed the entire path.

A. Teaching phase

The teaching process involved the development of the teaching node, responsible for monitoring the ROS topic '/robot_pose_ekf/odom_combined' provided by the Extended Kalman Filter (EKF) package. This package performs fusion between odometry and IMU data, returning information in the message type PoseWithCovarianceStamped. As the robot moves, the ROS node captures details such as its spatial position (x and y coordinates) and orientation. The x and y coordinates data are then stored in separate files, each representing a distinct path taken by the robot. This file structure enables users to select which path the robot should execute.

B. Fitting and generating the Bézier curve

This step is based on the algorithm PiecewiseG1BézierFit [4], which aims to fit a curve from certain points. This algorithm, originally written in Matlab source code, was adapted for Python[5] using NumPy[6] to replace certain functions present in Matlab. This portability was intended to facilitate integration with ROS and its efficiency.

The importance of the fitting method for the work is its capability to fit a Bézier curve from a set of data points. Figure 1 illustrates the fitting of Bézier curves to a dataset. [4] offers the "globally optimized only" fitting method, which defines only a single Bézier curve for the entire path. Despite being a simpler method, it is effective and suitable for our method.

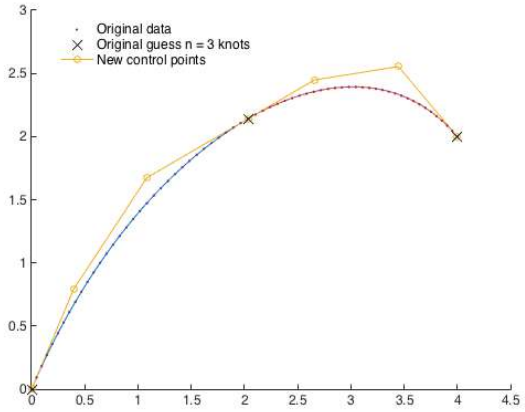

Fig. 1. Fitting a Bézier curve to a dataset using the PiecewiseG1BézierFit algorithm. The figure shows the original data points (dots), the initial guess with 3 knots (crosses), and the newly calculated control points (circles) that define the final Bézier curve (smooth line). This approach optimizes the curve fitting process, allowing the curve to closely follow the shape of the data while maintaining smoothness and continuity [12].

In addition to the benefits mentioned, Bézier curves provide parameterizing paths with several advantages, such as being easy to represent mathematically, having smooth and controllable curvature properties, and being able to be adjusted to different control point configurations. This makes the planning and navigation process more flexible, allowing it to adapt to different situations and scenarios.

C. Generating local planner paths

The local planner is based in the DWA local planner, generating lines to possible local paths for the robot to emulate a future behavior. Figure 2 illustrates how this process works.

Once the Bézier curve is adapted, it acts as a base to implement the repetition of the taught path. The method chosen for this work generates local paths for the robot. This process was essential for the robot to perform the repeat path stage. This involves simulating the mobile robot's future behavior by calculating its kinematics using the Jacobian matrix, which

[5]Python programming language https://www.python.org/.

[6]NumPy library https://numpy.org/

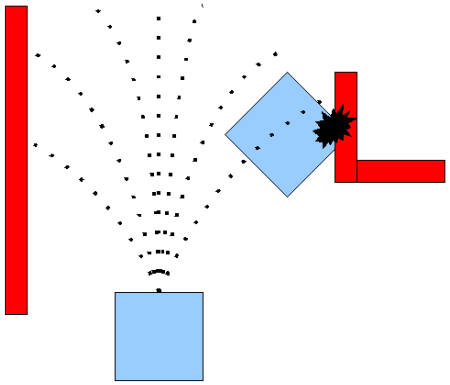

Fig. 2. Demonstration of the Dynamic Window Approach (DWA) applied to a mobile robot. In the figure, the robot (in blue) calculates potential trajectories (dotted lines), considering its kinematics and the obstacle constraints (in red) in the environment. The robot simulates future movements to determine the best path to follow, avoiding collisions and ensuring safe navigation. The potential collision point is shown in black. [7]

provides information on how it behaves when performing. Based on it, we create local paths that are in line with the robot's characteristics.

D. Calculating differential robot kinematics

To generate the local paths according to the robot's kinematics, some variables must be considered: $tyre\_radius$, which represents the wheel's radius, and $robot\_yaw$, which represents the robot's yaw angle [10].

$$f_{l0} = tyre\_radius \cdot \cos(robot\_yaw) \tag{1}$$

$$f_{a0} = tyre\_radius \cdot \sin(robot\_yaw) \tag{2}$$

Using the derived equations, 1 and 2, the Jacobian matrix is formulated as a 2×2 matrix, Eq. 3. This matrix relates the robot's linear velocities in the X and Y directions and its angular velocity with the wheel velocities.

$$J = \begin{bmatrix} f_{l0} & 0.0 \\ f_{a0} & 0.0 \end{bmatrix} \tag{3}$$

Thus, generating n curves in the local planner was possible according to the robot constraints. The operator can define the number of local paths, as can the distance between the points. With this method, we define the best local path for the robot based on the situation at the time. Figure 3 shows the local paths, with each curve being the vertical set of points.

Before publishing the robot's velocity, the robot's kinematics should be considered. The mobile base operates as a differential drive system. The following equations describe how to apply the velocity commands to the left 4 and right 5 wheels:

$$v_{\text{left}} = v_{\text{linear}} - v_{\text{angular}} \times \frac{d_{\text{btw\_wheels}}}{2} \tag{4}$$

$$v_{\text{right}} = v_{\text{linear}} + v_{\text{angular}} \times \frac{d_{\text{btw\_wheels}}}{2} \tag{5}$$

where v_{linear} represents the linear velocity command and angle is the angular velocity command. $d_{\text{btw\_wheels}}$ denotes the distance between the two wheels.

Once the velocities for the left and right wheels are determined, the robot's overall linear and angular velocities can be computed as follows: equations 6 and 7.

$$v_{\text{linear}} = \frac{v_{\text{left}} + v_{\text{right}}}{2} \tag{6}$$

$$v_{\text{angular}} = \frac{v_{\text{right}} - v_{\text{left}}}{d_{\text{btw\_wheels}}} \tag{7}$$

These equations provide a framework for calculating the robot's velocity based on the desired linear and angular velocities and the physical characteristics of the differential drive system.

E. Defining the best local path

To define the best local path to perform, the system calculates the cost of each local path to repeat the predefined path. This calculation involves comparing the robot's actual position about the reference point with the position predicted in the corresponding local path. The first reference points on the Bézier curve have higher weights (weight = number of points), and each point is subtracted - 1 from the weight. Hence, the points closest to the mobile robot significantly influence local path selection decisions. This ensures the robot considers the closest points more important during the repeating process. Figure 3 shows the local paths.

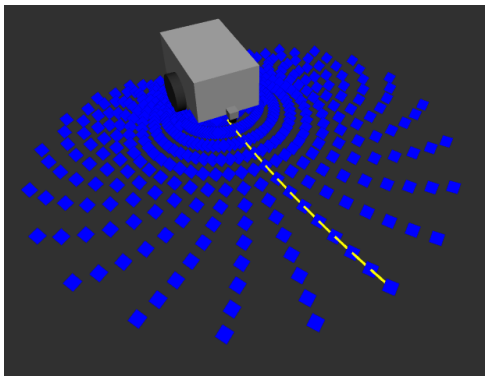

Fig. 3. Visualization of the local planner's possible paths considering the robot's dynamic limitations. The blue lines represent all potential paths that the robot could take. Among these, the yellow path is selected for execution, as it has the lowest cost. This cost is determined by weighing the robot's actual position relative to the predefined Bézier curve, emphasizing the points closer to the robot.

F. Repeting the path

As the robot moves along the path, it constantly monitors its proximity to the nearest reference point on the Bézier curve. In the same way, the reference points on the local paths are also equidistant from each other and defined in a set of n points. When the distance between the robot and the nearest reference point falls below a predefined threshold, that point is considered to have been reached by the robot.

The distance between the robot and the objective point is calculated using the Euclidean distance, Equation 8 to determine whether the vehicle has reached its goal. A threshold is set, and the objective is considered completed if the distance falls below this limit.

$$\text{dist} = \sqrt{(obj_x - pos_x)^2 + (obj_y - pos_y)^2} \tag{8}$$

- dist: Euclidean distance between the target point and the current position.
- obj_x, obj_y: x and y coordinates of the goal point.
- pos_x, pos_y: x and y coordinates of the current position.

The method then updates the reached reference point, removes it from the list of n points, considers its subsequent one the closest point to reach, and adds a new point at position *n + 1*.

With our method, we have n equidistant reference points on both the Bézier curve and the local paths, and as the points are used up, the n reference points move along the Bézier curve until there are no more possible points. This method allows the robot to progress gradually and systematically along the Bézier curve as the reference points are reached and replaced. Figure 4 demonstrates the logic developed for the method.

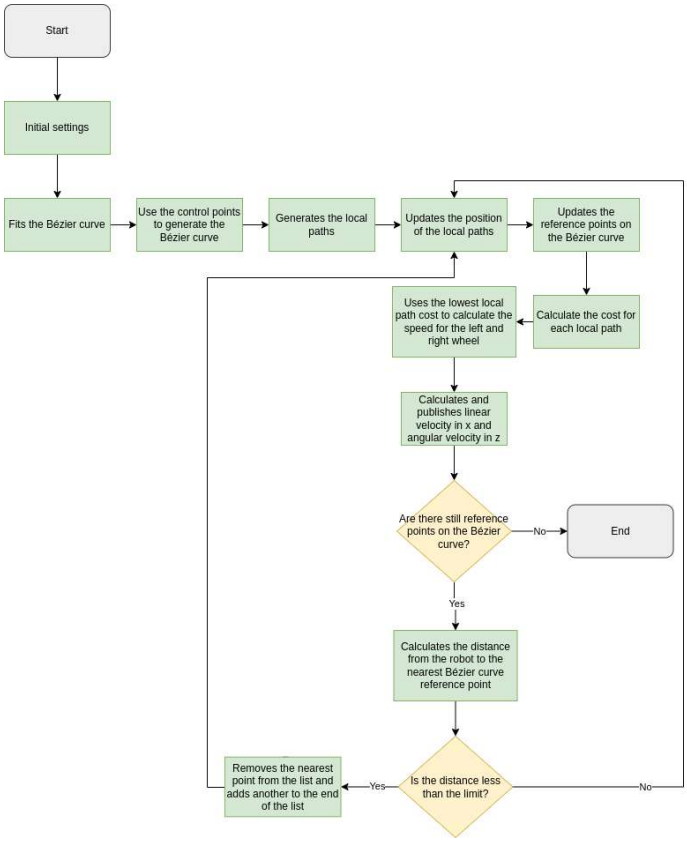

Fig. 4. Flowchart illustrating the repeating stage using Bézier curves and the Dynamic Window Approach.

IV. Experimental Environment

The experiments were conducted in indoor environments, typically found in offices, universities, and hospitals, focusing on corridors that service robots would commonly navigate. Figure 5 illustrates the paths used in these experiments. These spaces are often characterized by narrow corridors connecting various rooms, laboratories, and offices, providing a realistic scenario for testing the robot's capabilities. The design of the experimental paths was inspired by the work of [8], [13], and [14], which used similar path shapes to generate the teach and repeat results for mobile robot navigation systems.

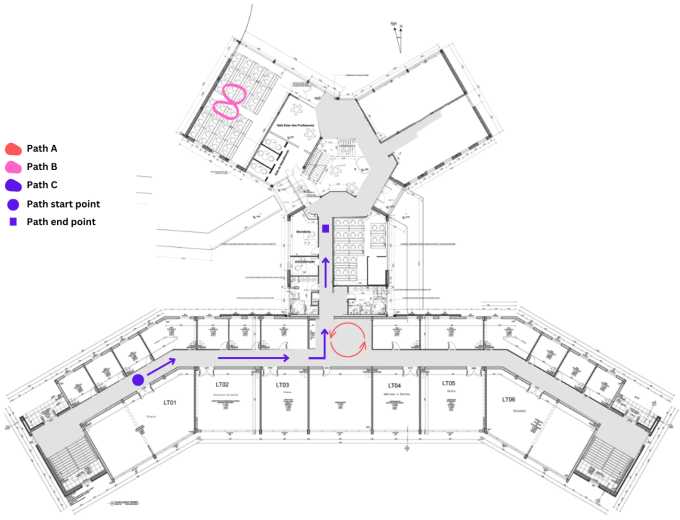

Fig. 5. Environment floor plan used for the experiments. The paths marked as A, B, and C represent different paths to repeat.

The experimental paths included three distinct types:

- **Path A**: A circular path designed to test the robot's performance in continuous turns, which is uncommon in regular service robot operations but useful for testing the algorithm's limits.
- **Path B**: An eighth-shaped path, which, similar to the circular path, is unusual for service robots but serves as a benchmark for testing the system's resilience to complex trajectories.
- **Path C**: A corridor navigation path representing a more realistic scenario that a service robot might encounter in real-world environments.

V. Results

This section presents the results of the experiments on the three paths described in the previous section. Each path was experimented with using the same mobile base and parameters, with three repetitions per path, to ensure consistency. Table I consolidates the experimental results, including the fitting error, repeat mean percentage error (MPE), and the time taken to complete each path.

The mean percentage error (MPE) between the planned and executed paths was calculated to evaluate the accuracy of the paths. The MPE is computed by determining the minimum distance between each point on the executed path and the closest point on the planned path. These minimum distances are averaged to give the mean error, denoted as D_{min}. This average error is compared to the mean distance of the points on the planned path from the origin, denoted as D_{ref}, providing a normalized measure of error as a percentage:

$$\text{MPE} = \left(\frac{D_{min}}{D_{ref}} \right) \times 100 \tag{9}$$

This metric is particularly useful because it provides a relative measure of how closely the executed path follows the planned route, regardless of the scale of the path.

Path	Distance	Experiment	Fit error	Repeat MPE	Time
A	10.841m	1	0.477%	5.748%	1min04sec
		2		6.600%	1min02sec
		3		6.219%	1min02sec
B	23.418m	1	1.361%	9.983%	2min07sec
		2		9.190%	2min14sec
		3		9.814%	2min08sec
C	39.210m	1	0.155%	0.836%	3min17sec
		2		0.783%	3min19sec
		3		0.783%	3min19sec

TABLE I
Comparison of results for different paths tested under various conditions.

- **Path A repeating results**: Path A, the circular path, yielded an average repeat MPE of 6.189%, with the worst case being 6.219% and the best 5.748% (see Figure 6).

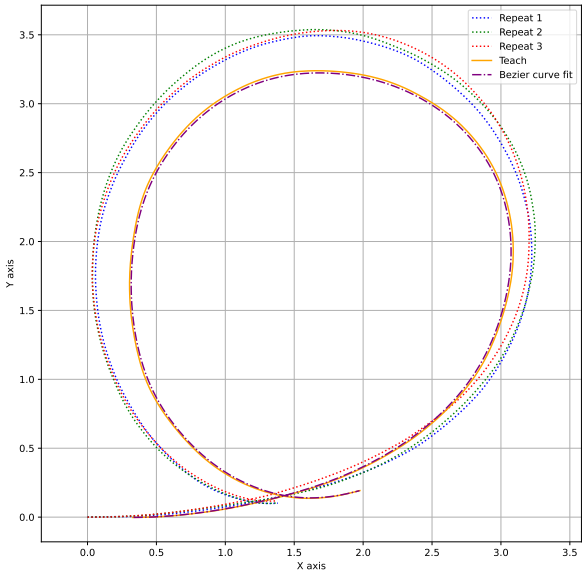

Fig. 6. Results of the three experiments on the circular path.

- **Path B repeating results**: In Path B, the eighth-shaped path, the average repeat MPE was 9.662%, with the worst case reaching 9.983% and the best case 9.190% (see Figure 7).
- **Path C repeating results**: Path C, which simulated corridor navigation, produced the lowest repeat MPE,

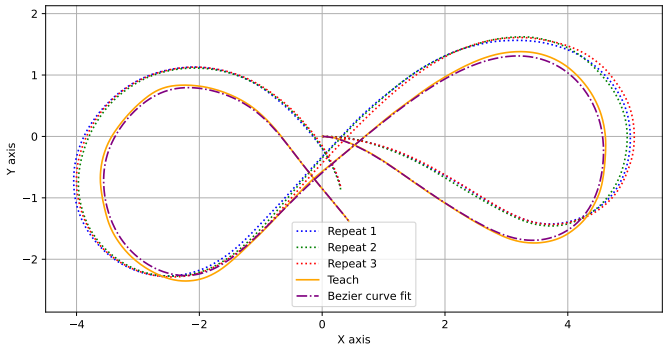

Fig. 7. Results of the three experiments on the eighth-shaped path.

averaging 0.766%. The highest error was 0.836%, while the lowest was 0.679% (see Figure 8).

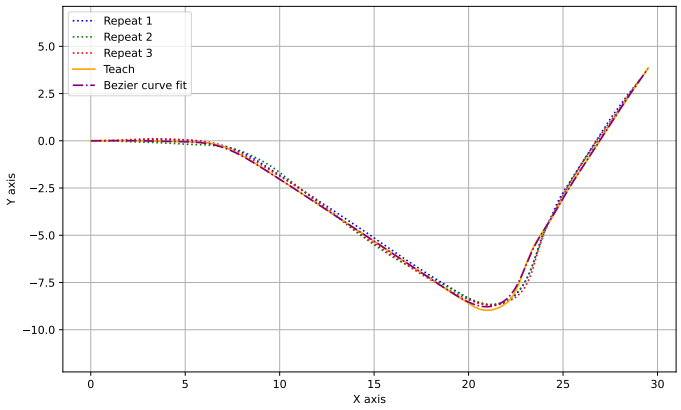

Fig. 8. Results of the three experiments on the corridor path.

VI. Conclusions

This work developed a method based on Bézier curves that implement the T&R technique for path planning in a low-cost mobile base. The system involves a user teaching a differential drive robot by tele-operating or manually guiding it along a specific path. This enables the mobile robot to replicate the same path in a real-world environment appropriate for service robots.

In the method implementing Bézier curves, adapting the curve to the points traveled by the mobile robot showed excellent results, with the worst error recorded in experiments being only 1.361%

The results were positive, considering the experiments used a simple robotic platform, only providing odometry and IMU sensors. These sensors provide accumulation errors that can cause problems during the T&R process, although the maximum MPE was 9.983%.

From the research and development of this work, it was possible to implement the T&R technique in the mobile robot. This technique enables the robot to learn a path and autonomously repeat it. Utilizing the odometry and IMU data available from the platform represents a possible feature of the navigation stack.

Future work could explore using Bézier curves with multiple segments or Bspline for greater path adaptability and dynamic adjustments during navigation. In addition, integrating a location system would increase the reliability of the data and support the implementation of the system over even longer paths.

References

[1] Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W.: Principles of robot motion: theory, algorithms, and implementations. MIT press (2005)

[2] Dias, N.F., Dyonisio, J.D.S., Quadros, L.F.M., Dorneles, G.A., Prado, J.J.P., Rocha, M.Z., Bicho, M.C., Lemos, J.F.S.S., Bezerra, H.C., Teixeira, C.N.S., Maurell, I.P., Guerra, R.S., Drews-Jr., P.L.J.: SHARK: Stable hoverboard-driven autonomous robot kit. In: Proceedings of the V Brazilian Humanoid Robot Workshop (BRAHUR) and VI Brazilian Workshop on Service Robotics (BRASERO) (2024). https://doi.org/10.29327/v-brahur-vi-brasero.869310

[3] Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine **4**(1), 23–33 (1997). https://doi.org/10.1109/100.580977

[4] Lane, E.J.: Fitting data using piecewise G1 cubic bézier curves (1995), https://hdl.handle.net/10945/31580

[5] Macenski, S., Martin, F., White, R., Ginés Clavero, J.: The marathon 2: A navigation system. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2020)

[6] Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., Konolige, K.: The office marathon: Robust navigation in an indoor office environment. In: International Conference on Robotics and Automation (2010)

[7] Marder-Eppstein, E., et al.: DWA Local Planner (2012), https://github.com/ros-planning/navigation

[8] Nourizadeh, P., Milford, M., Fischer, T.: Teach and repeat navigation: A robust control approach (2024). https://doi.org/10.48550/arXiv.2309.15405

[9] Ozkil, A.G., Fan, Z., Dawids, S., Aanes, H., Kristensen, J.K., Christensen, K.H.: Service robots for hospitals: A case study of transportation tasks in a hospital. In: 2009 IEEE International Conference on Automation and Logistics. pp. 289–294 (2009). https://doi.org/10.1109/ICAL.2009.5262912

[10] Paszkowiak, W., Bartkowiak, T., Pelic, M.: Kinematic model of a logistic train with a double ackermann steering system. International Journal of Simulation Modelling (2021), https://api.semanticscholar.org/CorpusID:236283381

[11] Pütz, S., Simón, J.S., Hertzberg, J.: Move Base Flex: A highly flexible navigation framework for mobile robots. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (October 2018), https://github.com/magazino/move_base_flex

[12] Rehm, E.: PiecewiseG1BezierFit. https://gitlab.com/erehm/PiecewiseG1BezierFit (2022)

[13] Rozsypálek, Z., Rouček, T., Vintr, T., Krajník, T.: Multidimensional particle filter for long-term visual teach and repeat in changing environments. IEEE Robotics and Automation Letters **8**(4), 1951–1958 (2023). https://doi.org/10.1109/LRA.2023.3244418

[14] Sprunk, C., Tipaldi, G.D., Cherubini, A., Burgard, W.: Lidar-based teach-and-repeat of mobile robot trajectories. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3144–3149 (2013). https://doi.org/10.1109/IROS.2013.6696803