Research Paper
Computational Methods for Differential Equations
http://cmde.tabrizu.ac.ir
Vol. *, No. *, *, pp. 1-10
DOI:10.22034/cmde.2024.64173.2895

A New Algorithm for Indoor Robot Localization using Turning Function

Pardis Sadatian Moghaddam[1,*], Ali Vaziri[2] and Anita Ershadi Oskouei[3]
[1]Department of Computer Science, Georgia State University, Atlanta, Georgia.
[2]Department of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, USA.
[3]Department of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, USA.

Abstract

One of the complex challenges in the field of artificial intelligence and robotics is enabling robots to accurately determine their position. While this task is simpler in open spaces using antennas, satellites, and other tools, it becomes much more challenging in enclosed environments. Various methods are employed for indoor positioning, one of which involves low-cost rangefinders and polygon mapping around the robot.
This paper presents a new algorithm entitled RLuTF using turning functions and their geometric properties, allowing the robot to determine its position at any given moment.

Keywords. Indoor Positioning, Turning Functions, Robot Localization.
2010 Mathematics Subject Classification. 65L05, 34K06, 34K28.

1. Introduction

The accurate localization of a robot, or any moving object, in indoor environments is of paramount importance, as highlighted in the previous sections. In such environments, where access to satellite-based systems like GPS is unavailable, it becomes crucial to continually determine the precise position of the robot. Without accurate localization, the cumulative error in position estimation can grow significantly over time. This phenomenon, often referred to as "drift," can cause the robot to become essentially blind and lost within just a few minutes, rendering it incapable of completing its tasks effectively.

Even if the initial position of the robot is known with high accuracy, various factors contribute to a gradual increase in error as the robot moves. These factors include computational inaccuracies, rounding errors in data, wheel slippage, wear and tear on the robot's wheels (such as reduced wheel diameter), sensor inaccuracies, and a host of other mechanical and environmental influences. As these errors accumulate, the robot's subsequent position estimates become increasingly unreliable, and after a few minutes of operation, the robot may no longer have an accurate sense of its location within the environment. This progressive loss of positional awareness results in the robot effectively being lost on the map, significantly hindering its ability to perform tasks that rely on knowing its exact location.

In robotics, this problem is critical for achieving autonomy, particularly in environments where there is no access to external location services. As such, developing robust localization methods to minimize or compensate for these errors is a central challenge in robotics research, especially for indoor navigation. Accurate localization is not just a convenience—it is essential for ensuring the reliability and functionality of the robot's operations in real-world scenarios.

Robot localization in indoor environments can be accomplished using a variety of techniques, with some of the most prominent methods relying on computer vision and image processing. In vision-based methods, the robot captures and compares images from its surroundings to determine its position. However, these techniques come with significant computational complexity due to the high volume of data and the need for continuous image analysis. Additionally,

Received: 27 October 2024 ; Accepted: 10 November 2024.
∗ Corresponding author. Email: psadatian1@student.gsu.edu.

vision-based localization can struggle in environments where images change rapidly, leading to less accurate results. Variations in lighting, obstacles, and perspective can further complicate image comparison, making it difficult to achieve reliable positioning in dynamic or cluttered settings.

In addition to the approach presented in this paper, several other methods have been explored in the literature for robot localization in indoor environments. For example, Shamsfakhr et al. [10] proposed a method for mobile robot localization in dynamic and cluttered environments using artificial landmarks. Their approach focuses on utilizing recognizable features within the environment to enable accurate localization even when the surroundings are constantly changing. This method has demonstrated robustness in cluttered spaces where traditional sensor-based methods struggle.

Furthermore, Shamsfakhr and Sadeghi Bigham [11] introduced the Geometrical Scan Registration (GSR) algorithm, which enables fast and robust robot pose estimation. The GSR algorithm uses geometric matching techniques to align scans obtained from sensors with a pre-existing map, allowing for more efficient localization in real-time applications.

Additionally, Padkan et al. [7] explored the use of the Turning Function in a different domain, applying it to fingerprint matching. They developed a novel approach based on the onion peeling method combined with the Turning Function, which proved effective in comparing complex geometric patterns.

These studies complement the approach presented in this paper by highlighting alternative uses of geometric methods for localization and pattern matching.

An alternative method, and the focus of this paper, involves the use of range-finding tools and sensors to construct a polygonal representation of the robot's surroundings. In this approach, distance sensors, such as LIDAR (Light Detection and Ranging), are used to measure the robot's proximity to obstacles in its environment. LIDAR technology enables the robot to perform remote distance measurements by emitting laser beams and analyzing the time it takes for the reflected beams to return. The data collected from these measurements allow the robot to create a polygonal shape around itself, representing the distances to the surrounding objects and walls.

This polygon, derived from the sensor readings, is then compared to a map the robot has stored in its memory. The map contains predefined information about the environment, such as walls, obstacles, and landmarks. By aligning the polygon generated from real-time sensor data with the existing map, the robot can accurately estimate its position within the environment. This process effectively matches the robot's current sensor-based representation of its surroundings with the known layout of the space, allowing it to maintain an accurate sense of location despite the absence of GPS.

LIDAR-based localization offers several advantages, including higher accuracy in environments with static obstacles and the ability to function in a wider range of lighting conditions compared to vision-based systems. However, it also requires sophisticated data processing algorithms to match the real-time sensor data with the stored map, as well as efficient filtering techniques to handle noise and sensor errors. The ability to continuously update the robot's location based on this comparison is key to ensuring that the robot remains aware of its position as it navigates through complex indoor environments.

The most challenging aspect of the localization problem, which is inherently a complex task, lies in identifying a point on the map where, if the robot were hypothetically positioned, the polygon formed around it would exactly match the actual polygon generated by its sensors. This task is computationally intensive because the number of potential points on the map that need to be tested is theoretically infinite, making the problem highly complex in terms of time and computational resources.

To solve this problem, the robot needs to compare the sensor-generated polygon with every possible polygon that could be formed at different points on the map. Since the robot can be located at any position and orientation within the environment, the number of configurations that need to be checked grows exponentially. This makes the problem not only a geometric challenge but also a computational one, as exhaustive search methods would be highly inefficient.

The key to reducing the complexity of this problem lies in finding ways to limit the number of operations required to perform this matching. Rather than testing an infinite number of potential positions and orientations, effective algorithms are needed to reduce the search space to a finite, manageable number of possibilities. One approach to this is discretizing the environment—dividing the map into a finite grid of potential robot locations, thereby transforming the infinite problem into one that can be computationally addressed.

However, even with discretization, the problem remains complex due to the high precision required for the match between the real-world polygon and the map-based polygon. This is where optimization techniques and heuristics come into play. Methods such as iterative closest point (ICP) algorithms, particle filters, or Monte Carlo localization (MCL) can be employed to estimate the robot's position more efficiently. These algorithms aim to converge on the correct position by evaluating only the most likely candidate locations, based on sensor readings and prior knowledge of the robot's movement.

Furthermore, techniques like sensor fusion, which combines data from multiple sensors (e.g., LIDAR, wheel encoders, IMUs), can help in refining the search process by providing more accurate estimates of the robot's movement and reducing the uncertainty in localization. The goal is to minimize the number of potential positions that need to be tested, thus making the problem computationally feasible while maintaining a high degree of accuracy in the robot's localization.

In essence, the real challenge is not just finding the correct position but doing so efficiently in a way that reduces computational overhead. By leveraging smart algorithms, mathematical optimizations, and advanced sensor technologies, the problem can be transformed from an infinite, time-consuming search into a solvable, finite problem within acceptable time constraints.

In this paper, the localization problem is addressed using the Turning Function approach. The Turning Function represents the shape of a polygon as a continuous function of the angles between its edges, allowing for a more efficient comparison of shapes, such as those formed around the robot by its distance sensors. By using this method, the problem of matching the real-world polygon generated by the robot's sensors to a known map is made more tractable, reducing the computational complexity compared to more exhaustive search-based methods.

The complexity of this problem increases dramatically in the Kidnapped Robot scenario. In this more challenging variant of the problem, the robot is randomly placed at an unknown point on the map, and it must determine its position without any prior information about where it might be. This scenario is significantly more difficult because the robot has no reference for its initial position, and therefore, it must perform a comprehensive search over all possible positions and orientations in the environment. The robot must continuously compare its sensor data against the entire map to locate itself, which dramatically increases the computational burden and time required to achieve accurate localization.

In this paper, we address the critical problem of robot localization in indoor environments, where traditional GPS-based methods are not applicable. Precise localization is essential for autonomous robots to navigate and perform tasks efficiently, and the accumulation of localization errors can significantly impair a robot's ability to operate. To solve this problem, we propose an approach that leverages the Turning Function metric to enhance localization accuracy, especially when utilizing sensor-based data such as that from LiDAR.

In section 2, we introduce and define the Turning Function metric, a powerful tool for comparing the shapes of polygons. This metric is particularly well-suited for the problem of robot localization, as it enables the comparison of the robot's surrounding environment—represented as polygons detected by LiDAR sensors—with its internal map. By analyzing the differences between these polygons, the Turning Function allows us to make precise adjustments to the robot's estimated position.

In section 3, we present a novel algorithm for solving the robot localization problem. This algorithm, termed Robot Localization using Turning Function (RLuTF), iteratively refines the robot's position by comparing the virtual polygon (based on the robot's predicted location) and the real polygon (obtained from sensor readings). The algorithm is designed to operate efficiently in real-time environments, offering a balance between accuracy and computational feasibility.

Finally, in section 4, we summarize our findings and discuss the potential applications of the proposed algorithm in real-world robotics. We also highlight future directions for research, particularly in extending the algorithm to more complex scenarios such as the kidnapped robot problem, where the robot must identify its location after being placed in an unknown position.

2. Turning Function

One of the most crucial steps of a shape matching methods is finding the similarity degree between two shapes. Therefore, similarity and distance metrics are two concepts that need to be defined. There are many distance metrics that can be used in shape and polygon matching. Choosing an appropriate distance metric depends on the application. Shape is an important low-level image feature that can be used in matching purposes such as polygon matching. The main challenging issue in shape matching methods is defining a shape representation that is invariant to rotation, translation, scaling, and the curve starting point shift [4]. In the following, we first define distance metric and similarity metric and then we introduce turning function as an efficient shape matching metric.

Distance Metric. Given a set X, a distance metric is a real-valued function $dist(a, b)$ on the Cartesian product $X * X$ if, for any $a, b, c \in X$, it satisfies the following conditions [1, 3].

(1) $dist(a, b) \geq 0$,
(2) $dist(a, b) = dist(b, a)$,
(3) $dist(a, c) \leq dist(a, b) + dist(b, c)$,
(4) $dist(a, b) = 0$ if and only if $a = b$.

Similarity Metric. Given a set X, a similarity metric is a real-valued function $sim(a, b)$ on the Cartesian product $X * X$ if, for any $a, b, c \in X$, it satisfies the following conditions [3]:

(1) $sim(a, b) = sim(y, x)$,
(2) $sim(a, a) \geq 0$,
(3) $sim(a, a) \geq sim(a, b)$,
(4) $sim(a, b) + sim(b, c) \leq sim(, z) + sim(y, y)$,
(5) $sim(a, a) = sim(b, b) = sim(a, b)$ if and only if $a = b$.

The *Turning Function* is a mathematical tool used for comparing the shapes and contours of polygons. It provides a way to capture the geometric structure of a polygon by measuring the cumulative turning angle as a point traverses the polygon's boundary. This function can be particularly useful in tasks such as shape matching and robot localization.

Definition 2.1. Let P be a polygon with vertices $A_1, A_2, \ldots, A_n$. The *Turning Function* $T(\theta)$ represents the total angle turned by a point as it traverses the boundary of the polygon, starting at an initial point on the boundary and moving in a counterclockwise direction. The angle is accumulated at each vertex as the point moves along the edges. Formally, the Turning Function is defined as:

$$T(s) = \int_0^s \kappa(\tau)\, d\tau$$

where s is the arc length along the boundary and $\kappa(\tau)$ is the curvature at each point τ on the boundary.

The function $T(s)$ is periodic, repeating every 2π, and can be normalized to the perimeter of the polygon, making it invariant to transformations such as translation, rotation, and scaling.

2.1. Applications in Robotics and Shape Matching.

The Turning Function has broad applications, particularly in fields such as robotics and computer vision. In **robot localization**, for instance, a robot equipped with sensors such as LiDAR can compute the Turning Function of its surrounding environment. By comparing this function to the stored map of the environment, the robot can accurately estimate its position.

Additionally, the Turning Function is used in **shape matching** algorithms. For example, if two polygons are represented by their respective Turning Functions, a comparison of these functions allows for an efficient way to determine shape similarity. In tasks such as object recognition, the Turning Function can distinguish between different geometric shapes based on their curvature profiles.

2.2. Comparison of Shapes Using Turning Functions.

To illustrate the concept, consider a square and an equilateral triangle. The Turning Function for the square will increase by $\frac{\pi}{2}$ (i.e., 90 degrees) at each vertex, resulting in a stepwise graph. In contrast, the Turning Function for an equilateral triangle will have fewer, larger steps, increasing by $\frac{2\pi}{3}$ (i.e., 120 degrees) at each vertex.

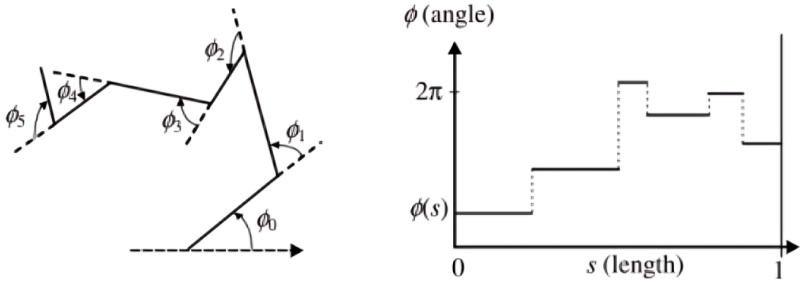

FIGURE 1. Turning Functions for a chain. The x-axis represents normalized perimeter, while the y-axis shows the cumulative angle [13].

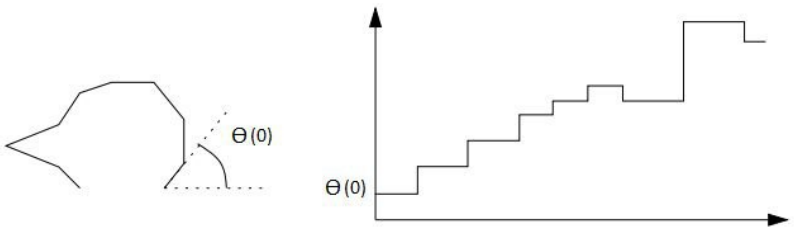

FIGURE 2. Polygonal curve and turning function [12].

By comparing the Turning Functions of two shapes, we can compute a measure of their similarity. For instance, two shapes are considered similar if the difference between their Turning Functions is small. This comparison is often done using a distance metric such as the L_2-norm:

$$d(T_1, T_2) = \left(\int_0^1 (T_1(s) - T_2(s))^2 \, ds \right)^{1/2},$$

where $T_1(s)$ and $T_2(s)$ are the Turning Functions of the two shapes being compared.

2.3. **Relevance to Robot Localization.** In the context of **robot localization**, the Turning Function enables robots to compute the shape of the environment from sensor data and compare it to a known map. By minimizing the difference between the Turning Function derived from sensor readings and the Turning Function of the map, the robot can refine its estimated position with high accuracy. It can be seen many mathematical foundations and applications of the Turning Function in literature [5, 6, 8, 12].

2.4. **Turning Function as a Metric.** The turning function is an easy and standard tool for representing and describing shapes and in particular polygons. Arkin et al. [1] have proposed a method for comparing polygons based on the L_p distance between their turning functions that works for both convex and non-convex polygons. The turning function starts from a certain point, called reference, that is arbitrary selected on the shape's boundary [9]. The function measures the counterclockwise tangent as a function of arc s [1]. $\Theta(0)$ is the angle that reference point makes with x-axis. $\Theta_A(s)$ keeps track of the turning that takes place along the shape, increasing with counterclockwise turns and decreasing with clockwise turns. For a convex polygon, the turning function is monotone and changes in vertices up to $\Theta(0) + 2\pi$ and is constant between two consecutive vertices [1].

The distance between polygons $A1$ and $A2$ is defined as L_p distance between their turning functions, see Figure 3. Consider $\Theta_A 1(s)$ and $\Theta_A 2(s)$ as two turning functions. The distance between their turning functions is defined as:

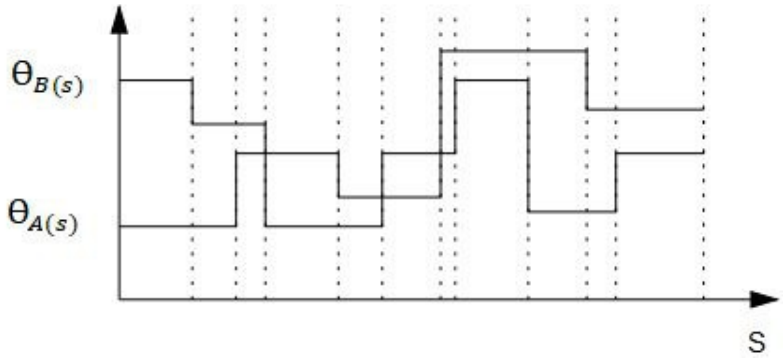

FIGURE 3. The distance between two polygons is calculated using rectangles enclosed by $\theta_A(s)$, $\theta_B(s)$, and dotted lines [13].

$$d(P1, P2) = \|\Theta_P 1 - \Theta_P 2\| = (\int_0^1 |\Theta_P 1(s) - \Theta_P 2(s)|^p ds)^{1/p}. \tag{2.1}$$

Rotating the polygons and choosing different points as the reference point affects the distance. Therefore, the minimum distance over all different reference points and rotation is computed as follows:

$$d(A1, A2) = (\min_{\Theta \in \mathbf{R}, t \in [0,1]} \int_0^1 |\Theta_A 1(s+v) - \Theta_A 2(s) + \theta|^p ds)^{1/p}. \tag{2.2}$$

In the above formula, polygon A is rotated by angle θ and the reference point is shifted by an amount v. The turning function is invariant to scaling, since the polygons is re-scaled and also invariant under translation and rotation because of having no orientation information [2]. These properties makes it ideal for measuring the similarity between shapes and it can be used in computer vision and specially image retrieval. According to [1], comparing shapes based on the turning function runs in $O(mn \log(mn))$ time where m and n are the number of vertices in two polygons.

3. ALGORITHM RLUTF

In contrast, a simpler version of this problem assumes that the robot knows its initial position at the start of its movement. In this version, the robot only needs to correct its position periodically before the accumulated error becomes too large. This is the scenario our algorithm focuses on. Here, we assume that the robot is aware of its initial location, and as it moves through the environment, it recalibrates its position regularly, correcting for the accumulated errors before they grow beyond a tolerable threshold. This periodic correction ensures that the robot's position estimate remains accurate over time.

Our algorithm solves this simpler version of the localization problem by running at regular intervals—every minute in our case. The algorithm works by comparing the robot's actual position, as derived from real-time sensor data, with its estimated position, as determined by its previous calculations and movement predictions. When discrepancies are found between these two positions, the algorithm updates the robot's estimated location, aligning it with the actual location detected by the sensors. This continuous process of matching and updating ensures that the robot's understanding of its position remains precise at all times, even as it moves through the environment.

The Robot Localization using Turning Function (RLuTF) algorithm, as described in this paper, provides an efficient and reliable method for maintaining accurate localization. By leveraging the Turning Function to compare polygons generated by sensor data with known maps, our algorithm ensures that the robot can continuously correct its position in real-time, preventing the accumulation of significant localization errors. This approach strikes a balance between accuracy and computational efficiency, making it well-suited for practical applications in indoor environments where frequent position recalibration is necessary.

In the following sections, we present the details of the RLuTF algorithm, its underlying principles, and its implementation for real-time localization in robotic systems.

Algorithm 1: RLuTF Algorithm

Input: Map of the environment, Or_{old} (previous real position of the robot), polygon of obstacles around the robot using LiDAR (Pr)

Output: Or_{new} (updated real position of the robot)

1. Calculate Ov_{new} (virtual position at the new time step) using Or_{old} and the operations performed;
2. Compute Pv (virtual polygon) assuming the robot is at Ov_{new};
3. Obtain Pr (real polygon) from the sensor data;
4. Compute the difference between Pv and Pr using the Turning Function and calculate the area between them;
5. **if** *the difference is less than a small threshold* ϵ **then**
   - Set $Pr = Pv$;
   - Continue with the movement;
8. **else**
   - a. Select three random points around Ov_{new} within a distance δ;
   - b. Repeat the algorithm from Step 1 for each of these three points;
   - c. Choose the point that yields the smallest ϵ and replace Ov_{new} with this point;
   - d. $\delta = \delta/2$;
   - e. Continue from Step 4;
14. Return $Or_{\text{new}} = Ov_{\text{new}}$;

Symbols and Notation.

- Or_{old}: This symbol represents the previous real position of the robot at the last time step. It is obtained from the robot's last known location before the current movement.
- Ov_{new}: This is the virtual position of the robot at the current time step. It is calculated based on the previous real position (Or_{old}) and the operations or movements the robot has performed since the last time step. This is essentially the robot's predicted position before correction.
- Pv: The virtual polygon formed around the robot assuming it is located at Ov_{new}. This polygon is constructed based on the known map and the assumption that the robot's virtual position is correct.
- Pr: The real polygon obtained from the robot's sensors, specifically using the LiDAR data. This polygon represents the actual distances from the robot to surrounding obstacles or walls at the current time step.
- ϵ: This represents a small threshold used to measure the acceptable difference between the real and virtual polygons. If the difference between Pv and Pr is less than ϵ, the robot's position is considered accurate enough to continue moving without further correction.
- δ: This represents a small distance around the current virtual position (Ov_{new}). If the difference between Pv and Pr exceeds the threshold ϵ, the algorithm selects random points within this distance (δ) to reevaluate the position and find a better match.

Turning Function: This is a mathematical function used to compare two polygons, Pv and Pr. It measures the difference in the shape and orientation of these polygons by calculating the area between them. The Turning Function helps in determining how well the virtual polygon matches the real polygon.

General Structure of the Algorithm Initial Estimation of the Robot's Position: The algorithm begins by calculating the robot's virtual position (Ov_{new}) using its previous real position (Or_{old}) and the operations it has performed. This step uses basic motion models to predict the robot's new position.

Polygon Construction: After determining Ov_{new}, the algorithm constructs a virtual polygon (Pv) around the robot, assuming it is at this new position. Simultaneously, the real polygon (Pr) is obtained from the robot's LiDAR sensors, which provide actual distance measurements to nearby obstacles.

Comparison of Real and Virtual Polygons: The algorithm uses the Turning Function to calculate the difference between the virtual polygon (Pv) and the real polygon (Pr). This step helps to determine if the robot's estimated position is accurate enough by comparing the two polygons.

Position Correction: If the difference between Pv and Pr is below the threshold ϵ, the algorithm assumes that the virtual position is accurate and proceeds with the robot's movement. If the difference exceeds ϵ, the algorithm selects three random points within a small distance δ from Ov_{new} and repeats the position estimation for each point.

Optimization: Out of the three randomly selected points, the point that results in the smallest difference (the lowest ϵ) between Pv and Pr is chosen as the new virtual position (Ov_{new}). This ensures that the robot's position estimate is as accurate as possible.

Return Updated Position: Once the best match is found, the real position of the robot (Or_{new}) is updated to match the optimized virtual position (Ov_{new}), ensuring that the robot's understanding of its location is accurate.

3.1. **Time Complexity.** In this section, we analyze the time complexity of the Robot Localization using Turning Function (RLuTF) algorithm, focusing on its computational cost at each step and providing an overall assessment using asymptotic notation.

Step 1: Initial Position Calculation: The algorithm begins by calculating the virtual position of the robot, Ov_{new}, based on the previous real position Or_{old} and the actions taken by the robot. This calculation typically involves using a motion model, which takes into account the robot's control inputs and sensor readings. Assuming the motion model and sensor update require constant time, the complexity of this step is $O(1)$.

Step 2 & 3: Polygon Construction: The next steps involve the construction of two polygons: 1. The virtual polygon, Pv, generated assuming the robot is at Ov_{new}. 2. The real polygon, Pr, generated using real-time sensor data (e.g., LiDAR).

Both polygons are typically composed of n vertices, where n represents the number of points detected by the sensor (or approximated by the virtual polygon). Constructing a polygon from these points requires linear time relative to n, making the time complexity of this step $O(n)$.

Step 4: Polygon Comparison using Turning Function: In this step, the algorithm compares the real and virtual polygons using the **Turning Function**. The complexity of comparing two polygons using the Turning Function depends on the number of vertices in the polygons. Given that both Pv and Pr have n vertices, the Turning Function comparison typically requires traversing each vertex of both polygons and calculating the difference in angles or areas.

Thus, the time complexity of this comparison step is $O(n)$.

Step 5: Iterative Search for the Optimal Position: If the difference between Pv and Pr exceeds the threshold ϵ, the algorithm performs an iterative search to find the optimal virtual position by selecting three random points within a distance δ from Ov_{new}. For each of these points, the algorithm repeats the previous steps, including recalculating the virtual position, constructing the virtual polygon, and comparing it to the real polygon. Since these steps are repeated for three points, the worst-case time complexity of this step is $O(3 \times n)$.

However, considering that at each step, the operations of each loop may only be repeated a limited number of times, in the worst-case scenario, each localization at each stage could potentially be repeated up to $\frac{\log(\delta/\epsilon)}{2}$ times. Therefore, the worst-case time complexity of this step is $O(n \times \frac{\log(\delta/\epsilon)}{2})$.

If the optimal position is not found after three points, the algorithm may repeat this process, but the number of iterations is usually small (constant). Therefore, we can consider the time complexity of this step to be linear with respect to n.

Step 6: Return Updated Position: Once the best match is found, the algorithm updates the real position of the robot, Or_{new}, to match the optimized virtual position, Ov_{new}. This update is a constant-time operation $O(1)$.

Overall Time Complexity: Given the analysis of each step, we can now calculate the overall time complexity of the RLuTF algorithm. The most computationally expensive step is the polygon comparison, which has a linear

time complexity of $O(n)$. Since this step is performed for three candidate positions in the worst case, the total time complexity of the algorithm is $O(n \times \frac{\log(\delta/\epsilon)}{2})$.

Theorem 3.1. *In the worst case scenario, the time complexity of the RLuTF algorithm is* $O(n \times \frac{\log(\delta/\epsilon)}{2})$ *with respect to the number of vertices* n *in the polygons constructed from sensor data.*

Proof. Each step of the algorithm either requires constant time (e.g., initial position calculation, position update) or linear time with respect to n (e.g., polygon construction, Turning Function comparison). Even when the algorithm iteratively searches three random points, the operations performed at each point involve polygon construction and comparison, which take $O(n)$ time. Since the number of iterations is constant (at most $O(3n \times \frac{\log(\delta/\epsilon)}{2})$.), the overall complexity of the RLuTF algorithm In the worst case scenario, is $O(n \times \frac{\log(\delta/\epsilon)}{2})$. □

The RLuTF algorithm offers an efficient solution to the robot localization problem, with a time complexity that scales linearly with the number of vertices n in the polygons generated by the robot's sensors. This makes the algorithm well-suited for real-time localization in environments where frequent recalculations of the robot's position are necessary.

4. Conclusion

In this paper, we addressed the critical problem of robot localization in indoor environments. Specifically, we examined this problem under the assumption that the robot operates in a local, known area and without access to GPS or external positioning systems. Utilizing the *Turning Function*, a robust metric for comparing polygons, we developed an efficient algorithm that enables the robot to accurately determine its position by comparing sensor-detected obstacles to a pre-existing map.

One of the key strengths of the proposed algorithm lies in its computational complexity. While slightly higher than linear algorithms, it remains computationally feasible for real-time applications in indoor environments. The algorithm achieves this by working solely on polygonal representations of the robot's surroundings, significantly reducing the need for complex and expensive imaging equipment.

By leveraging the Turning Function, the method provides a high-speed localization solution without requiring high-resolution image data, making it particularly suitable for resource-constrained robots. Moreover, our analysis shows that the time complexity of new algorithm RLuTF is of the order $O(n \times \frac{\log(\delta/\epsilon)}{2})$, making it both fast and scalable.

In summary, the algorithm we presented combines the accuracy of geometric comparison with the computational efficiency required for indoor localization tasks, offering a promising solution for real-time robot navigation in complex environments.

References

[1] E. M. Arkin, et al., *An efficiently computable metric for comparing polygonal shapes*, Cornell University Operations Research and Industrial Engineering, 1989.

[2] D. Cakmakov and Emilija Celakoska, *Estimation of curve similarity using turning functions*, International Journal of Applied Mathematics, *15* (2004), 403-416.

[3] S. Chen, B. Ma, and K. Zhang, *On the similarity metric and the distance metric*, Theoretical Computer Science, *410*(24-25) (2009), 2365-2376.

[4] D. Lee, , S. Antani, and L. Rodney Long, *Similarity measurement using polygon curve representation and fourier descriptors for shape-based vertebral image retrieval*, Medical Imaging 2003: Image Processing, SPIE, *5032* (2003).

[5] E. Menegatti, et al., *Omnidirectional vision scan matching for robot localization in dynamic environments*, IEEE transactions on robotics, *22*(3) (2006), 523-535.

[6] R. Ohbuchi, T. Minamitani, and Tsuyoshi Takei, *Shape-similarity search of 3D models by using enhanced shape functions*, International Journal of Computer Applications in Technology, *23*(2-4) (2005), 70-85.

[7] N. Padkan, B. Sadeghi Bigham, and M. R. Faraji, *Fingerprint matching using the onion peeling approach and turning function,* Gene Expression Patterns, *47* (2023), 119299.

[8] J. Philip,*Shape Matching Using Turning Functions*, University of Waterloo, 1999.

[9] B. Sadeghi Bigham and Samaneh Mazaheri, *A Survey on measurement metrics for shape matching based on similarity, scaling and spatial distance*, Data Science: From Research to Application, Springer International Publishing, 2020.
[10] F. Shamsfakhr, B. Sadeghi Bigham, and A. Mohammadi, *Indoor mobile robot localization in dynamic and cluttered environments using artificial landmarks,* Engineering Computations, *36*(2) (2019), 400–419.
[11] F. Shamsfakhr and B. Sadeghi Bigham, *GSR: Geometrical scan registration algorithm for robust and fast robot pose estimation*, Assembly Automation, *40*(6) (2020), 801–817.
[12] R. C. Veltkamp and M. Hagedoorn, *Shape similarity measures, properties and constructions*, International Conference on Advances in Visual Information Systems, Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.
[13] R. C. Veltkamp, *Shape matching: Similarity measures and algorithms*, Proceedings International Conference on Shape Modeling and Applications, IEEE, 2001.