Indoor Wireless Beacon Tracking Using ESP32

Muskan Patnool, Varshini G N
Department of Electrical Engineering,
IIIT Bangalore Comet Foundation,
Bangalore, India 560083
muskan@iiitb.ac.in

***Abstract*—Through this paper, we demonstrate the use of machine learning in beacon tracking using an unmanned ground vehicle (UGV) and a WiFi-enabled microcontroller such as the ESP32. This is done by using received signal strength indicator (RSSI) values from a single target beacon as inputs to the microcontroller. These values are then used for making decisions on UGV positioning using learning algorithms.**

1 Introduction

Positioning, localization and navigation (PLAN) technology has become an essential capability for consumer electronics and autonomous vehicles alike. This fast growing market has seen the entry of big technology industry that was traditionally focused on internet software [**?**]. Use cases of PLAN technologies include object tracking, deep sea exploration, autonomous navigation, medical applications and assistance for the elderly and the specially abled [**?**], [**?**]. Reliable PLAN technology can reduce road accidents, congestion, energy and time consumption. However, many existing PLAN technologies are complex, cater to industrial use cases and use proprietary hardware that might be scarcely available to consumers [**?**], [**?**].

To overcome the above challenges, we propose a simple, effective and robust algorithm for indoor beacon tracking using a single WiFi or BLE beacon and a low-cost easily available microcontroller such as the ESP32 that is suitable for consumer electronics such as robot vacuum cleaners. The main contributions of this paper are as follows.

1) A simple yet robust algorithm for indoor beacon tracking using only RSSI measurements from a single target beacon.
2) A practical implementation of the algorithm on an ESP32 microcontroller mounted on a UGV chassis using open-source frameworks such as PlatformIO.

The remainder of this paper is organized as follows. In Section 2, we present a brief survey of related work in the field of indoor beacon tracking. In Section 3, we formulate the optimization problem corresponding to this task and describe our approach. In Section 4, we present the implementation details of the algorithm on the ESP32 microcontroller. In Section 5, we present the experimental results and performance evaluation of our proposed approach. Finally, in Section 6, we conclude the paper and discuss possible extensions of our work.

2 Related Work

A comprehensive survey of PLAN technology presented in [**?**] categorizes PLAN technologies based on the primary sensors used in navigation and illustrates the trade-offs between accuracy, cost and complexity. Some of these technologies are briefly described below.

A fuzzy controller system consisting of three ultrasonic sensors and one camera to detect obstacles in the robot's vicinity is presented in [**?**]. The navigation logic is implemented on BLE tag modules using the angle of arrival (AOA) method and is suitable for unmanned nursing in hospitals, especially during pandemic outbreaks. [**?**] proposes iDROP, a 3D localization scheme for drones in indoor GPS-denied environments that is robust against noise and multipath fading and provides location estimation with high accuracy. A UGV implemented on the robot operating system (ROS) that uses light detection and ranging (LiDAR) for accurate navigation even in dynamic conditions and is suitable for various indoor robotic use cases such as in warehouses or construction sites is available in [**?**]. [**?**] proposes a scalable navigation system using multiple ESP32 microcontrollers acting as BLE beacons. Here, a neural network models the position of the robot based on the signal strength received from the various beacons and the Levenberg-Marquardt method is used to accurately estimate the position of the robot.

Methods based on artificial intelligence (AI) have gained popularity in recent literature. [**?**] proposes an easy to implement method for multi-object tracking (MOT) using an encoder-decoder mechanism. The proposed method achieves comparable performance with state-of-the-art frameworks on the challenging MOT dataset. In [**?**], deep reinforcement learning (RL) is leveraged to train an agent for tracking dynamically moving objects using unmanned surface vehicles (USV) in marine environments. The trained RL agent performs comparably to the analytically derived trajectory in the steady state. The authors of [**?**] explore multiple RL algorithms using LiDAR inputs on TurtleBot3, showing that the twin delayed deep deterministic policy gradient (TD3) is the most efficient and robust across varied environments.

Many of the above methods are either too complex or too expensive to be implemented on cheap hardware such as an ESP32 microcontroller. Furthermore, a lot of supporting infrastructure needs to be set up to enable autonomous navigation. Our work aims to address these challenges by providing

a lightweight and cost-effective solution for indoor beacon tracking using a single wireless beacon and easily available wireless-capable microcontrollers such as the ESP32.

3 Proposed Navigation System

In this section, we formulate the beacon tracking problem as an optimization problem to estimate the radial distance of the UGV from the beacon. We then propose two solutions to this problem and compare their effectiveness.

3.1 Problem Statement

To estimate (radial) distance to beacon, we use its signal strength which is usually given by the RSSI. The RSSI (in dBm) at radial distance of r metres is defined as

$$C(r) = C(1) - 10\log_{10}(r), \tag{1}$$

where $C(1)$ is the RSSI at a distance of 1 metre from the beacon. Using this definition, the beacon tracking problem can be formulated as the following optimization problem.

$$\max_r C(r) \text{ s.t. } r > 0. \tag{2}$$

The derivative and second derivative of $C(r)$ is

$$C'(r) = -\frac{10}{\ln 10}\frac{1}{r}, \tag{3}$$

$$C''(r) = \frac{10}{\ln 10}\frac{1}{r^2} > 0. \tag{4}$$

Notice that for $r > 0$, $C'(r) < 0$, thus $C(r)$ is a decreasing function of r. This implies that the maximum RSSI is at $r = 0$, as expected. Additionally, note that $C(r)$ is a convex function of r since $C''(r) > 0$ for $r > 0$. Thus, we can use gradient ascent [**?**] to recursively find the point where the RSSI is maximum, which would correspond to the location of the beacon. Using (3), the gradient ascent update equation for the radial distance r is given by

$$r_{n+1} = r_n + \alpha C'(r_n) = r_n - \frac{10\alpha}{r_n \ln 10} \tag{5}$$

where r_i is the radial distance at the i-th step and α is the step size and r_0 is the initial radial distance of the UGV. Since $r_n > 0$, we can see that $r_{n+1} < r_n$ for all n. In other words, the radial distance to the beacon is decreasing with each step. This means that the UGV will converge towards the beacon. However, due to the explosion of the gradient in (3) at $r = 0$, the update steps become larger as r_n decreases, which could lead to overshooting the beacon. We now compare two algorithms that overcome this limitation by using a recursive approach.

3.2 Beacon Tracking Algorithm

The proposed beacon tracking algorithms consists of two stages at each step. The first stage is the *probing stage*, where the UGV measures the RSSI at various points in its vicinity. The second stage is the *decision stage*, where the UGV decides its direction of movement based on the RSSI measurements.

3.2.1 Angular Array RSSI Decision Algorithm (AARD): The algorithm proposed in [**?**] is described in Algorithm 2. The probing phase consists of taking RSSI measurements in place but at various orientations. The UGV reads the RSSI values at 3 locations 90°apart. The decision phase consists of moving in the direction where the signal is the strongest. A major drawback of this approach is that there may not be appreciable difference in the measured RSSI values at the various orientations in the same place, especially if the beacon is far away. This could lead to the UGV making arbitrary moves instead of converging towards the beacon.

Algorithm 1 AARD

Input: RSSI threshold T, number of steps N

while GetRSSI() $< T$ **do**
  $r_F \leftarrow$ GetRSSI().
  Turn left by 90 degrees in place.
  $r_L \leftarrow$ GetRSSI().
  Turn right (from initial orientation) by 90 degrees in place.
  $r_R \leftarrow$ GetRSSI().
  if $r_F \geq r_L$ and $r_F \geq r_R$ **then**
    Turn to the initial direction in place.
  else if $r_L \geq r_F$ and $r_L \geq r_R$ **then**
    Turn to the left direction in place.
  else$r_R \geq r_F$ and $r_R \geq r_L$
    Turn to the right direction in place.
  Move forward in the chosen direction by one step.

3.2.2 Linear Array RSSI Decision Algorithm (LARD): To overcome the issue of arbitrary movement when placed at a large distance from the beacon, the UGV must travel a significant distance in the probing stage to obtain differences in the measured RSSI values so that there is enough information to make a decision that is better than movement in a randomly chosen direction. Thus, in the probing stage of our proposed algorithm, the UGV measures the RSSI at N points in a straight line in its current orientation. In the decision stage, the UGV then moves to the position where the maximum RSSI was measured. If this is at the beginning or end of the probing trajectory, the UGV moves backward or forward respectively. Otherwise, it turns perpendicular to its probing trajectory at that point (either left or right). Our proposed algorithm is described in Algorithm 2. The main shortcoming of this particular algorithm is the amount of movement required in the probing stage for making one simple move in the decision stage.

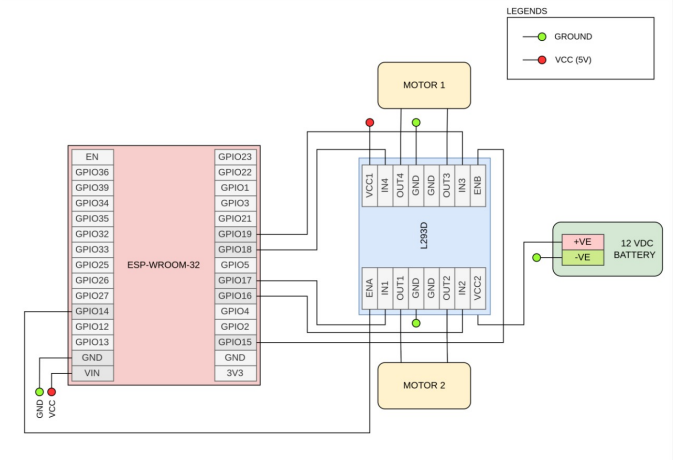

Fig. 1: Wiring Diagram for Beacon Tracking.

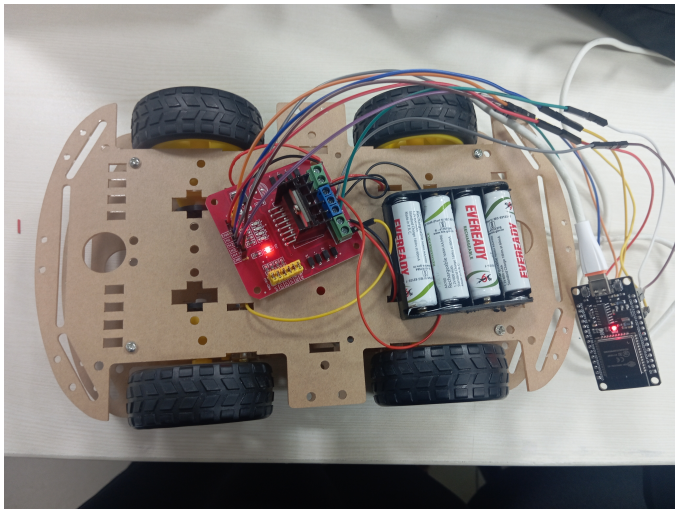

Fig. 2: Result for Beacon Tracking.

Algorithm 2 LARD

Input: RSSI threshold T, number of points N

while GetRSSI() $< T$ **do**
  $r_0 \leftarrow$ GetRSSI().
  for i = 1 to $N-1$ **do**
    Move forward by one step.
    $r_i \leftarrow$ GetRSSI().
  $j \leftarrow \arg\max_i r_i$.
  Move to the position at step i.
  if $i = N-1$ **then**
    Move one step forward.
  else if $i = 0$ **then**
    Move one step backward.
  else
    Turn left.

4 Implementation

Source codes for both of the beacon tracking algorithms discussed above are implemented in C++. They are available on https://github.com/gadepall/ugv-beacon/tree/main/codes.

4.1 Assets

1) UGV chassis with DC motors
2) ESP32 microcontroller with Type-B USB cable
3) L293D Motor Driver IC
4) Breadboard and Jumper Wires
5) Android phone
6) (Optional) USB 2.0/3.0 Hub

5 Results

In the AARD algorithm, the UGV eventually converges close to the beacon (here, the hotspot). However, in some cases, when the RSSI values are too close, the UGV may not be able to track the beacon and may end up in a random direction. The LARD algorithm is more reliable. However, in this case, if there are a lot of nearby obstacles, the UGV may not converge close to the location of the beacon. It may either get physically blocked by the beacon or the signal interference may be too high.

6 Conclusion and Future Work

In this paper, we presented a cost-effective and robust beacon tracking algorithm that uses RSSI measurements to approach the beacon at each step. Our work uses easily available hardware and software tools to implement the algorithm on an ESP32 microcontroller mounted on a UGV chassis. A simple extension to our proposed algorithm could be to combine the AARD and LARD algorithms to reduce the time and total number of steps taken by adjusting the value of (N, θ), which are the number of linear and angular observation points, as the UGV converges to the beacon. This could be done using various heuristics such as the total number of probes or current RSSI measurements. A further extension of this work could be to suitably adapt (5) so that the UGV does not overshoot the beacon.