Koc. J. Sci. Eng., 5(1): (2022) 73-83 https://doi.org/10.34088/kojose.1012914

Kocaeli University

Kocaeli Journal of Science and Engineering

http://dergipark.org.tr/kojose

Natural Navigation System Design for Indoor Mobile Robots

F. Azeez ADEBAYO [1,*], H. Metin ERTUNÇ [2]

[1] *Department of Mechatronics Engineering, Kocaeli University, Kocaeli, 41001, Turkey,* ***ORCID:*** *0000-0002-8160-6949*
[2] *Department of Mechatronics Engineering, Kocaeli University, Kocaeli, 41001, Turkey,* ***ORCID:*** *0000-0003-1874-3104*

Article Info

Research paper

Received : October 21, 2021
Accepted : February 07, 2022

Keywords

Simultaneous Localization and Mapping
Indoor Natural Navigation
Sensor Fusion
PID
Differential Drive Robot

Abstract

Natural navigation simply refers to free navigation without the necessity of tapes, magnets, reflectors, or even wires. Many autonomous vehicles possess this as world maps are readily available and provide a perfect basis for machine learning solutions. However, this is not so much the case for indoor applications. Here, paths are often dynamic and more constrained; therefore, requiring the continuous identification, mapping and localization of the surrounding area. This work focuses on developing an indoor natural navigation system; the localization is achieved with a fusion of the wheel's odometry to the on-board Inertial Measurement Unit (IMU i.e., a combination of relative localization and absolute localization) using Unscented Kalman Filter (UKF) as system's encoder's accumulation of errors is desired to be nullified while employing a PID control in correcting reference state errors. The map is simultaneously constructed using laws of geometry based on static points obtained from a Lidar, subsequently converted to an occupancy grid layout for effective path planning. In operation, tangency is applied in the avoidance of dynamic obstacles. The simulation results obtained in this study confirms the possibility of a simple, educational, indoor navigation system approach easily integrable by other mobile robots of the differential drive model.

1. Introduction

Various studies have gone into autonomous navigation in vehicles for both indoor and outdoor use cases. These studies, while they have without doubt contributed to the success recorded in this area, also offer execution of complex equations and algorithms, which has made its adoption difficult and ultimately, hinder their reproducibility. Indoor mobile robot navigation particularly suffers here as the lack of satellite positioning signals, so discussed in [1] implies its localization and mapping calls for the inclusion of more sensors in its determination which further complicates and compounds the equations.

In this study, we attempt to build a simple indoor differential drive robot, using simplified steps and equations, for a better understanding of the literature, and to ultimately, ensure reproducibility in any coding language. A LiDAR-based navigation technology called CoreSLAM [2] was studied and its variation, BreezySLAM [3] is implemented to generate an occupancy grid map [4] for which the robot will operate. Its feasibility and performance relative to other SLAM techniques is evaluated in [5].

Due to the accumulation of noise leading to drift errors that is accompanied by the use of encoder's wheel odometry alone in localization, a sensor fusion technique with the Unscented Kalman Filter, is applied to combine its results with an IMU [6-7]. The UKF, like the other filters such as Kalman, extended Kalman filter used in combining data of various sensors of similar targets, has been shown in [8] to require very basic governing equations on orientation and position in order to estimate robot states.

In getting the robot to move from one desired place to another, the A-star algorithm from [9] is employed. The algorithm plots the shortest walkable path between nodes, in this study, map pixels, depending on the distance measurement technique selected; Manhattan, Euclidean, etc. Also, the suitability of PID in minimizing error between target and reference robot speed state has been

* Corresponding Author: hazeezadebayo@gmail.com

established in [10] while a simple Tangency approach is selected in avoiding obstacles as the need for a non-infrastructural change and cost-free solution is imperative.

The results obtained in this study showed a 70% improvement in path following accuracy of the sensor fused algorithm over the use of wheel odometry alone. This is further corroborated in [7], where for the square trajectory tested, an accuracy of 66.5% was observed, and a similar result is depicted in [11].

Similar works have also been demonstrated with some variation of the above techniques depending on the choice of sensor, often involving but not limited to; visual SLAM with cameras, sonar-based localization, or some hybrid integration into neural networks [11-15], as well as other combinations with more additions in terms of complexity and constraints, as in the use of RFID, Wi-Fi or Bluetooth whose accuracy and performance depends on implementing other external structures such as receivers [16-19]. The next section will highlight the methodology as well as offer calculation justifications in the design or the differential drive robot. After that, a section on results, followed by conclusions and recommendations marks the end of the paper.

2. Materials and Methods

2.1. Kinematics of Differential Drive Robot

The basic configuration of the mobile robot is with two drive wheels with encoders mounted on them and a free castor wheel for its stability. Drive wheels can be controlled independently, and they maintain a common axis as shown in Figure 1.

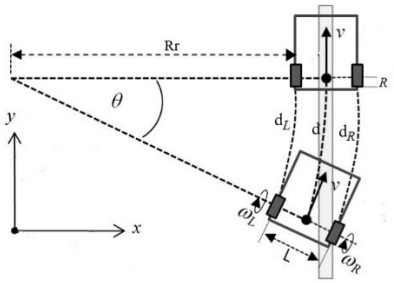

Figure 1. Differential Drive.

The kinematic model of differential drive mobile robot is therefore given by the relations:

$$\dot{x} = \frac{v_R+v_L}{2}\cos\theta = \frac{R}{2}(\omega_R+\omega_L)\cos\theta \quad (1)$$

$$\dot{y} = \frac{v_R+v_L}{2}\sin\theta = \frac{R}{2}(\omega_R+\omega_L)\sin\theta \quad (2)$$

$$\dot{\theta} = \frac{v_R-v_L}{L} = \frac{R}{L}(\omega_R-\omega_L) \quad (3)$$

where; ω_R is rate at which right wheel is turning, ω_L is rate at which left wheel is turning, R is radius of the wheels, and L is the wheel base distance.

Three notable cases associated with the differential drive are also stated below:

- When $v_R = v_L$, the robot moves in a straight line either forward or backward depending on the direction of wheel rotation.
- When $v_R = 0$, the robot moves about the right wheel and about the left wheel when $v_L = 0$. The robot will always move about the wheel with the smallest velocity and while this helps in steering, small errors in the relative velocities between the wheels can affect the robot trajectory.
- When $v_R = -v_L$, the robot rotates about the midpoint of the wheel.

Implementing this model helps us in translating from right and left wheel velocities, v_R and v_L into $\dot{x}$ and $\dot{y}$ describing changes along x and y as well as $\dot{\theta}$ describing changes in the robot orientation. However, since v_R and v_L, do not form inputs that are readily available to us in motion, using the unicycle model our inputs can be designed to be v and ω representing translational and angular velocities respectively. The model is given by:

$$\dot{x} = v\cos\theta \quad (4)$$

$$\dot{y} = v\sin\theta \quad (5)$$

$$\dot{\theta} = \omega \quad (6)$$

Equating both $\dot{x}$ in Eq. (1) and (4) and diving the through by $\cos\theta$. We have;

$$v = \frac{R}{2}(\omega_R+\omega_L) \Longrightarrow \frac{2v}{R} = \omega_R+\omega_L \quad (7)$$

$$\omega = \frac{R}{L}(\omega_R-\omega_L) \Longrightarrow \frac{\omega L}{R} = \omega_R-\omega_L \quad (8)$$

From which v_R and v_L are obtained as:

$$\omega_R = \frac{2v+\omega L}{2R} \implies v_R = \frac{2v+\omega L}{2} \quad (9)$$

$$\omega_L = \frac{2v-\omega L}{2R} \implies v_L = \frac{2v-\omega L}{2} \quad (10)$$

Thus, making our inputs governed by the combined action of both the linear velocity v and the angular velocity w. Of which, feedbacks are obtained from the wheels as ticks representing how many revolutions $s_{(time,t)}$, and by extension, distance d_L, d_R moved by the robot wheels of radius r over a certain amount of time. From these, the position and orientation are calculated by applying geometric techniques [20].

$$d_L = \frac{s_{l(t)}-s_{l(t-1)}}{n} * 2\pi r \quad (11)$$

$$d_R = \frac{s_{r(t)} - s_{r(t-1)}}{n} * 2\pi r \quad (12)$$

where n, is the number of ticks of encoder per revolution.

For analysis, assuming the robot is at some position (x_0, y_0), headed in a direction making an angle θ with the x axis as described in Figure 1. From this, we can write:

$$d_L = R_r \Delta\theta \quad (13)$$
$$d = (R_r + \frac{L}{2})\Delta\theta \quad (14)$$
$$d_R = (R_r + L)\Delta\theta \quad (15)$$

where R_r is the radius of movement, d_L and d_R are arc curvature or distance traveled by the left wheel, right wheel respectively and d is distance described by the center.

Making $\Delta\theta$ and d the subject of the formular yields the relations:

$$\Delta\theta = \frac{d_R - d_L}{L} \quad (16)$$
$$d = \frac{d_R + d_L}{2} \quad (17)$$

If distance traveled, d taken to be so infinitesimally small that it is assumed to be a straight line. We can write the change in x and y to be of the form:

$$\Delta x = d\cos\theta \quad (18)$$
$$\Delta y = d\sin\theta \quad (19)$$

Since coordinates (x_0, y_0) are the known starting point of the robot, the new pose is therefore estimated as:

$$x' = x_0 + d\cos\theta \quad (20)$$
$$y' = y_0 + d\sin\theta \quad (21)$$

However, ideally, we may also describe the position of a robot capable of moving in a particular direction $\theta(t)$ at a given velocity $v(t)$ by integrating Eq. (1), (2) and (3) as:

$$x(t) = \frac{1}{2}\int_0^t (v_R(t) + v_L(t))\cos\theta(t)\,dt \quad (22)$$
$$y(t) = \frac{1}{2}\int_0^t (v_R(t) + v_L(t))\sin\theta(t)\,dt \quad (23)$$
$$\theta(t) = \frac{1}{L}\int_0^t (v_R(t) - v_L(t))dt \quad (24)$$

For the special case of a differential drive robot, the odometry which is the means by which we can obtain or estimate the pose information of the robot as in $(x, y \text{ and } \theta)$, essentially tracking the effect of the wheel velocities and updating pose accordingly, solving the forward kinematics problem is given as:

$$x(t) = x_0 + \frac{1}{2}\int_0^t (v_R(t) + v_L(t))\cos\theta(t)\,dt \quad (25)$$
$$y(t) = y_0 + \frac{1}{2}\int_0^t (v_R(t) + v_L(t))\sin\theta(t)\,dt \quad (26)$$
$$\theta(t) = \theta_0 + \frac{1}{L}\int_0^t (v_R(t) - v_L(t))dt \quad (27)$$

Unfortunately, we cannot simply specify an arbitrary robot pose $(x;\ y;\ \theta)$ and find the velocities that will get us there as in the inverse kinematics problem since each individual wheel contributing to the robot's motion imposes constraints on the robot; which is that it cannot directly slide to the side. Described using Eq. (28) from Figure 2 below;

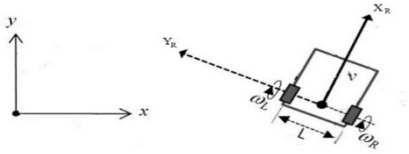

Figure 2. Non-holonomic nature of the differential drive.

$$\dot{Y}_R = \dot{y}\cos\theta - \dot{x}\sin\theta = \left(\frac{R}{2}(\omega_R + \omega_L)\sin\theta\right)\cos\theta - \left(\frac{R}{2}(\omega_R + \omega_L)\cos\theta\right)\sin\theta = 0 \quad (28)$$

here, θ is the angle $\dot{X}_R$ makes with the horizontal axis, x.

Implying that the robot indeed cannot move laterally along its axle, it will require a more complicated set of steering maneuvers. Hence, imposing what is called a non-holonomic constraint. We may also confirm this by checking if the workspace velocity equation in Eq. (28) is integrable. Here we have:

$$f(x, y, \theta) \neq 0 \quad (29)$$

2.2. PID

In section 2.1, we showed that our inputs are functions of both the linear velocity v and the angular velocity w. Since the linear velocity v, can be chosen as a constant valued number that represents the speed limit of our robot, the heading becomes the only controllable parameter as seen in Eq. (6). The difference between the desired direction and the current direction is known as the error and is written in Eq. (30).

$$e = \theta_d - \theta_o \quad (30)$$

where θ_d is the desired orientation which can be obtained using Eq. (31) when waypoints in the form (x_g, y_g) are provided as goal states for the robot and θ_o is the robot's current orientation.

$$\theta_d = \arctan\left(\frac{y_g - y_o}{x_g - x_o}\right) \quad (31)$$

or set as $\theta(t)$ when v_R and v_L are directly inputted as in Eq. (27). If a PID is applied to correct this error, e, as shown in Eq. (32), to produce ω which will help in moving the current state to the desired state, Eq. (33).

$$PID(e) = \mathrm{K_p} e(t) + K_I \int_0^t e(t)dt + K_D \dot{e}(t) \quad (32)$$

$$\omega = \mathrm{PID(e)} \quad (33)$$

where error e, for correctness, is evaluated such that the angle is expected to remain between $-\pi$ and π to avoid ambiguities.

2.3. IMU

Now that we have all that is required to control the robot, it is imperative to state that these models only describe an ideal scenario where there is no drift or slip in the wheels or from the encoder reading. While further operating under the assumption that the time under consideration is infinitesimally small. In practice however, this is not always the case. Although the time can still be kept relatively infinitesimal, the slip cannot be so programmed. To overcome this, an IMU equipped with magnetometer, accelerometer and gyroscope is used externally to track the movement and its results fused. The plane XY is particularly of interest as the rotation is about the Z axis. First, we need to normalize the raw accelerometer data as:

$$accel\_x\_normalized = \frac{accel[x]}{\sqrt{(accel[x]^2 + accel[y]^2 + accel[z]^2)}} \quad (34)$$

$$accel\_y\_normalized = \frac{accel[y]}{\sqrt{(accel[x]^2 + accel[y]^2 + accel[z]^2)}} \quad (35)$$

where acceleration in X, Y and Z directions are given by $accel[x], accel[y]$ and $accel[z]$ with their resulting normalized acceleration in X and Y.

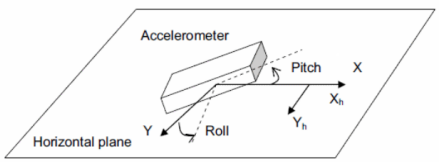

Figure 3. IMU at tilted position.

If the IMU device is tilted, then the pitch and roll angles are not equal to 0°, as seen in Figure 3. The magnetic sensor measurements in all direction will need to be compensated. Hence, we first obtain the pitch and roll angle value as:

$$pitch = \mathrm{asin}(accel\_x\_normalized) \quad (36)$$

$$roll = -\mathrm{asin}(\frac{accel\_y\_normalized}{cos(pitch)}) \quad (37)$$

Next, we calculate the tilt compensated magnetometer in X and Y directions as:

$$magnxcomp = magn[x] * \cos(pitch) + magn[z] * sin(pitch) \quad (38)$$

$$magnycomp = magn[x] * sin(roll) * sin(pitch) + magn[y] * cos(roll) - magn[z] * sin(roll) * cos(pitch) \quad (39)$$

Finally, the earth's magnetic north is measured using the components of the magnetometer obtained above, from which our heading formular can then be written as:

$$heading = 180 * \frac{atan2(magnycomp, magnxcomp)}{\pi} \quad (40)$$

This, as well as the obtained longitudinal acceleration readings will be used in the sensor fusion model.

2.4. Sensor Fusion

The system is modeled in the UKF to have two sensors; IMU's linear acceleration, angular velocity measurement, and heading measurements as well as the encoders reading. The state vector is chosen to be:

$$x_n = \begin{bmatrix} p_x \\ p_y \\ \varphi \\ v \\ \dot{\varphi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} x + vt\cos\theta \\ y + vt\sin\theta \\ \theta + wt \\ v + at \\ w \\ a \end{bmatrix} \quad (41)$$

where p_x is the absolute x position, p_y is the absolute y position, φ is the yaw, v is the longitudinal velocity, $\dot{\varphi}$ is the yaw rate and $\dot{v}$ is the longitudinal acceleration.

Of which Gaussian noise with covariances representing uncertainties in our guess or estimates are subsequently introduced. That is:

$$x_{n+1} = f_n(x_n) + u_n \quad (42)$$

$$y_n = h_n(x_n) + v_n \quad (43)$$

where u_n is the process noise, v_n is the measurement noise, the process model is f_n and measurement model is h_n. x_{n+1} is the state vector and its measurement is denoted by y_n. Here, we wish to estimate the mean and covariances of both the state and its predicted estimated measurement.

2.4.1. Unscented Transformation

Suppose we know the mean $\bar{x}$ and covariance P of the state x, which could be either predicted or filtered estimates, we can find a set of points with a sample mean and covariance equal to $\bar{x}$ and P. the points are called sigma points, which are random in the sense that they depend on the current estimate of the state and state error covariance. We then apply our nonlinear function, Eq. (44), to each of these points. Once these are known, the calculations are deterministic i.e., no random generators are involved. The sample mean and covariance is a good estimate of the true mean and covariance of y. the biggest drawback here being the requirement of a matrix square root that requires $\mathcal{O}(n_x^3)$ operations.

$$y = h(x) \tag{44}$$
$$y^i = h(x^i) \tag{45}$$
$$\hat{y}_u = \frac{1}{2n_x}\sum_{i=1}^{2n_x} y^i \tag{46}$$
$$R_{\tilde{y},u} = \frac{1}{2n_x}\sum_{i=1}^{2n_x}(y^i - \hat{y}_u)(y^i - \hat{y}_u)^T \tag{47}$$

where y^i represents the output of every sigma point, $\hat{y}_u$ is the unscented estimated mean obtained from the sample average and $R_{\tilde{y},u}$ unscented estimated covariance. This also applies to the states in estimating its mean and covariance.

2.4.2. Unscented Kalman Filter

Although, there are three steps for the unscented Kalman filter; the Time update or prediction step, the sigma update and lastly, the measurement update or filter step. The sigma update step can be skipped with a bit of trade off on the accuracy while saving computation i.e., only the time update and measurement step are used and are shown below:
where n represents time. We start with what the mean and covariance initially are at time, $n = 0$:

$$\hat{x}_{0|0} = E[x_o] \tag{48}$$
$$P_{\tilde{x},0|0} = \left|\left|x_0 - \hat{x}_{0|0}\right|\right|^2 \tag{49}$$

Then, for $n = 1, \ldots$ the time update step is performed, then the mean is predicted and covariances of the state are estimated by running the sigma points into the process model, f_n as seen in Eq. (53).

$$x_{n-1}^i = \hat{x}_{n|n-1} + \tilde{x}^i \quad \text{for i} = 1, \ldots, 2\text{n}_\text{x} \tag{50}$$
$$\tilde{x}_n^i = \left[row_i\left(\sqrt{n_x P_{\tilde{X},n-1|n-1}}\right)^T\right] \text{ for i} = 1, \ldots, \text{n}_\text{x} \tag{51}$$
$$\tilde{x}_n^i = -\left[row_i\left(\sqrt{n_x P_{\tilde{X},n-1|n-1}}\right)^T\right] \text{ for i} = \text{n}_\text{x} + 1, \ldots, 2\text{n}_\text{x} \tag{52}$$
$$\hat{x}_n^i = f_n(\hat{x}_{n-1}^i) \tag{53}$$
$$\hat{x}_{n|n-1} = \frac{1}{2n_x}\sum_{i=1}^{2n_x}\hat{x}_n^i \tag{54}$$
$$P_{\tilde{x},n|n-1} = \frac{1}{2n_x}\sum_{i=1}^{2n_x}(\hat{x}_n^i - \hat{x}_{n|n-1})(\hat{x}_n^i - \hat{x}_{n|n-1})^T \tag{55}$$

where a $row_i(A)$ denotes the i^{th} row vector of the matrix A and $\sqrt{n_x P_{\tilde{X},n-1|n-1}}$ is a matrix square root of $(n_x P_{\tilde{X},n-1|n-1})$ such that $\sqrt{n_x P_{\tilde{X},n-1|n-1}}^T\sqrt{n_x P_{\tilde{X},n-1|n-1}} = n_x P_{\tilde{X},n-1|n-1}$. n_x is the dimension of the state vector, $\hat{x}_{n|n-1}$ is the estimate of the mean and covariance $P_{\tilde{X},n-1|n-1}$ of the state error covariance at time $n-1$, lastly, x_{n-1}^i, represents the sigma point. Next, the measurement update is performed. Again, the predicted estimates are taken and then run through the measurement model h_n to obtain the measurement sigma points, y_n^i. Then the predicted estimated value of the measurement, $\hat{y}_{n|n-1}$ which is the sample average of the sigma points are calculated. R_{n-1} covariance representing the gaussian noise for each sensors introduced. It is a diagonal matrix. We also use a covariance $P_{\tilde{x}\tilde{y}}$ that is a combination of the state sigma points and the measurement sigma points to obtain the Kalman gain, K_n. This gain factor is basically used to decide final value for robot state. Subsequently we update the predicted state estimates and covariance.

$$y_n^i = h_n(x_n^i) \tag{56}$$
$$\hat{y}_{n|n-1} = \frac{1}{2n_x}\sum_{i=1}^{2n_x} y_n^i \tag{57}$$
$$P_{\tilde{y}} = \frac{1}{2n_x}\sum_{i=1}^{2n_x}(y_n^i - \hat{y}_{n|n-1})(y_n^i - \hat{y}_{n|n-1})^T + R_{n-1} \tag{58}$$
$$P_{\tilde{x}\tilde{y}} = \frac{1}{2n_x}\sum_{i=1}^{2n_x}(x_n^i - \hat{x}_{n|n-1})(y_n^i - \hat{y}_{n|n-1})^T \tag{59}$$
$$K_n = P_{\tilde{x}\tilde{y}}P_{\tilde{Y}}^{-1} \tag{60}$$
$$\hat{x}_{n|n} = \hat{x}_{n|n-1} + K_n(y_n - \hat{y}_{n|n-1}) \tag{61}$$
$$P_{\tilde{x},n|n} = P_{\tilde{x},n|n-1} - K_n P_{\tilde{y}} K_n^T \tag{62}$$

There are however other ways of evaluating sigma points e.g, upon computation of some scaling parameter, λ which is a function of n_x. For this, in the place of $\frac{1}{2n_x}$ the corresponding mean and covariance weights W_i are used. Every other thing does remain the same; calculate sigma points, time update, and measurement update wherein the final state represents the desired vectors. The UKF is favored as it facilitates implementation of the algorithms as non-linear transformations of a set of deterministically chosen sigma points which replaces calculations of Jacobian matrices [21].

2.5. Map Making

Basically, the Lidar obtains coordinates of static or non-dynamic obstructions as polar co-ordinates (r, θ) representing to what degree away from a specific datum within itself that it has spun. Hence, points of the Lidar's reading that falls below or within the Lidar's view-finder's range are collected and saved as landmarks, as the robot moves. In addition, every collected point is mapped or appended to the robot's location (x_i, y_i) at the time it was collected. The robot can be controlled to all locations of interest and an array of landmarks and corresponding robot locations are generated which are then casted into bytes to which an occupancy map can be created.

More specifically, the algorithm used is a variant of the CoreSLAM [2], and is called BreezySLAM [3], developed to implement a simple SLAM that can be integrated into a filter-based localization subsystem. This algorithm however, includes the use of the Random-Mutation Hill-Climbing search where a position S is initialized by m randomly selected instances from T. For each iteration, the algorithm randomly replaces one instance in S by another randomly selected instance in $T - S$. If this replacement can improve the predictive accuracy of the instances in T, the change will be retained [22]. This process is repeated for p times, where p is the maximum search iterations possible. This helps in establishing a better robot position based on a starting position as opposed the Monte Carlo localization (MCL). While, for each obstacle detected, the algorithm does not draw a single point, but a function with a hole whose lower point is at the position of the obstacle resulting in a grey-level map with holes dug to represent obstacle likelihood [2]. Bresenham algorithm is used to draw the view finder's rays. Furthermore, the conversion from real-world meters to pixels is also afforded the user. In this study, total map size in pixels and its equivalent in meters were chosen as 1000 and $10m$ respectively. Lastly, while the map algorithm is open loop, although not always required, the image generated may be further processed for a more perfect representation, in this case, with Opencv.

2.6. A-star

The name is derived from the algorithm being an optimal form of another common graph-search algorithm known as 'A'. Three functions are of interest and are given as the heuristic; a simple calculation of the distance or cost-to-go from the current node to the nth node or end node. It is often denoted by $H(n)$ and may employ different methods in obtaining a best estimate of the measured distance, in this case, Manhattan distance was selected. The function $G(n)$; representing the estimate of the cost-to-come i.e., path cost estimate from start node to node under consideration or current node. Lastly, $F(n)$, which is the sum of the cost-to-go and cost-to-come, Eq. (63).

$$F(n) = G(n) + H(n) \tag{63}$$

The $F(n)$ helps in prioritizing which node to move next upon looking up the queue in the open-set of available nodes for the node with the smallest *F score*. The open-set is an array of tuples of available nodes to move and their corresponding *F scores*, when at any given node of interest. An item is pulled from the open-set and expanded only once it has been selected, to which an open-set is generated again. Nodes can be thought of as grid co-ordinates or map intersects. In the end the taken path is made into an array of co-ordinates representing the best shortest distance from the start point to a goal or target point. These co-ordinates are what are called waypoints. The overall low computation time and the guarantee of producing optimal paths makes for A-star's selection for this study. The algorithm is demonstrated using Figure 4 below:

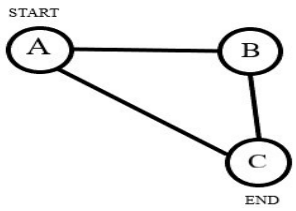

Figure 4. State transition graph illustrating the operation of the A-star.

Node A has two available nodes connected hence path AB and AC are considered and the respective $F(n)$ compared. Assuming all grid connectors have edge of 1 unit as they are equally spaced. It follows that:

- A→B: cost-to-come, $G(n)$, 1, representing the unit or edge of the path between A and B. then cost-to-go, $H(n)$, the heuristic, an estimate of the distance to the end or target point, C, also 1. Hence, from Eq. (63), $F(n) = 2$.
- A→C: cost-to-come, (n), 1, representing the unit or edge of the path between A and C. then cost-to-go, $H(n)$, the heuristic, an estimate of the distance to the end or target point, C, which is itself. Hence, 0. Therefore, $F(n) = 1$.
- The open-set will contain; $[(2, B), (1, C)]$. To which the candidate with the lowest *F score*, C, is selected and popped from the list. Hence, path chosen is A → C.

The map under consideration is an extract of the BreezySLAM generated image and processed using

Opencv in order to threshold the image. i.e., binarize; make it an image of just free spaces (white) and walls or obstructions (black). On this new map, co-ordinates become pixels to which the start point and goal point may be specified. The algorithm is run such that it checks a node or pixel's neighbor and avoids them if they are below a specified color threshold, in this case black pixels, then maps a path around it, i.e., path is strictly traced on the white spaces.

2.7. Obstacle Avoidance

The A-star algorithm generates waypoints for which already satisfies the avoidance of walls and other static obstacles. However, dynamic obstacles are not accounted and as such requires being addressed; when the Lidar or ultrasonic sensor detects an obstacle within a certain view range and angle, while the robot follows this predefined path, it first checks the distance between its current position and the next goal or target state in the waypoint, call it $p2t$ and compares it against the distance between its current position and the obstacle, $p2o$. If the distance $p2o$ is greater; it implies that the obstacle does not constitute an hinderance to its current trajectory hence, it is ignored. However, if $p2t$ is greater, a function is called that places a circle of a specified radius on the obstacle's location and draws a line tangential to the circle. The co-ordinate of the point of tangency is collected and appended to the waypoint as an intermediate stop for the robot to avoid the obstacle. The algorithm is demonstrated using Figure 5 below:

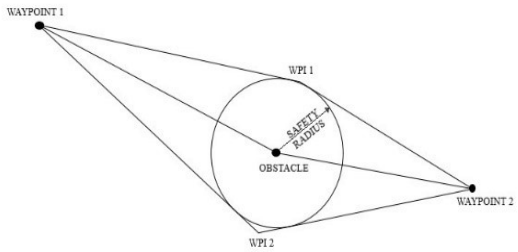

Figure 5. Obstacle avoidance illustration.

Let waypoint1 co-ordinate be (x_1, y_1) and obstacle co-ordinate be (x_o, y_o) . Then let a line *w1o* exist between these points, to which its center co-ordinate is $\left(\frac{x_1+x_o}{2}; \frac{y_1+y_o}{2}\right)$. Finally, let the distance between the center co-ordinate of *w1o* and (x_o, y_o) be *d.*

1. Iterate for x from 0 to $(h + r)$ and y from 0 to $(k + r)$, in circle equation, $(x - h)^2 + (y - k)^2 = r^2$ with center co-ordinate of *w1o* as the origin (h, k) and *d* as radius, *r.* and save all generated circle co-ordinates as tuples of an array, *B*.
2. Iterate for x from 0 to $(h + r)$ and y from 0 to $(k + r)$, in circle equation, $(x - h)^2 + (y - k)^2 = r^2$ with obstacle co-ordinate as the origin (h, k) and *safety _radius* as radius, *r.* and save all generated circle co-ordinates as tuples of an array, *C*.
3. Compare arrays *B* and *C* and extract the two common co-ordinates marking the intersection of both circles, tuple co-ordinates w1i1 and w1i2.
4. Repeat *1 – 5* for waypoint 2 as *w2o* and extract arrays *w2i1* and *w2i2* intersection points.
5. Using the line equation, $y - y_1 = m(x - x_1)$, where (x_1, y_1) is *w1o* and m is the gradient obtained between w1o and intersection w1i1.
6. Similarly, $y - y_1 = m(x - x_1)$, where (x_1, y_1) is *w2o* and m is the gradient obtained between *w2o* and intersection *w2i1*.
7. Iterate for x and y in both cases to maximum limit above the sum of radius of both circles and save in an array, *w1l* and *w2l*. Like in steps 1 and 2.
8. Compare *w1l* and *w2l* for a single point of intersection, i.e., waypoint *wpi1*.
9. Depending on the desired robot behavior; repeat steps *5, 6, 7* and *8* with the respective second point of intersection to *wpi2*. Decide which new waypoint, *wpi1* or *wpi2* to append as intermediary between waypoint1 and waypoint2.

2.8. Experimental Setup

Overall, the setup shown in Figure 6 includes:

- Raspberry pi
- Lidar
- Motor x2
- Incremental Encoder x2
- Ultrasonic sensor
- IMU
- L298n motor driver

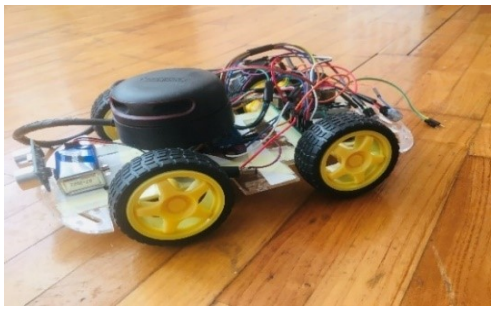

Figure 6. The assembled mobile robot.

The experiment was carried out in a room of size 5 by 4.4 m, with wooden demarcations which constituted part of the stationary obstacle, while the room's door opens to a corridor of about 6 by 1.2 m. Maximum and minimum

motor speed limits are set. The PID constants $K_p = 10$, $K_i = 0.01$, and $K_d = 0.1$, as well as the *previous error* and *cumulative error* for the calculation of the differential and integral errors respectively, were initialized at 0. The robot's dimensions (width, wheel radius etc.) were also initialized before the algorithm is run on the raspberry.

As the robot move, the encoder's feedback in combination with the IMU's data is served as input values into motion equations satisfying Eq. (41), to which state information is predicted. Furthermore, this information is then fed into the SLAM algorithm that obtains Lidar data and mapping ensues. Simultaneously, the PID is employed to correct the difference between the theoretical target state and the real-world execution. This generates a corrected w that helps in updating the robot's pose before the whole process is repeated until a satisfactory map is obtained by the user.

3. Results and Discussion

Two main use cases are foreseen; Firstly, the user controls the robot around the intended spaces and the robot, as it moves, localizes, and generates a map as depicted in Figure 7 (Map 1-6), then saves the last copy of the generated map when satisfactory. Secondly, the user specifies any target co-ordinates or points within the map, declared as start and stop pixel co-ordinates respectively. The saved image, Figure 7 (Map 7), is then binarized by the robot, Figure 7 (Map 8), and a path is generated, and subsequently traced by the robot as seen in Figure 8.

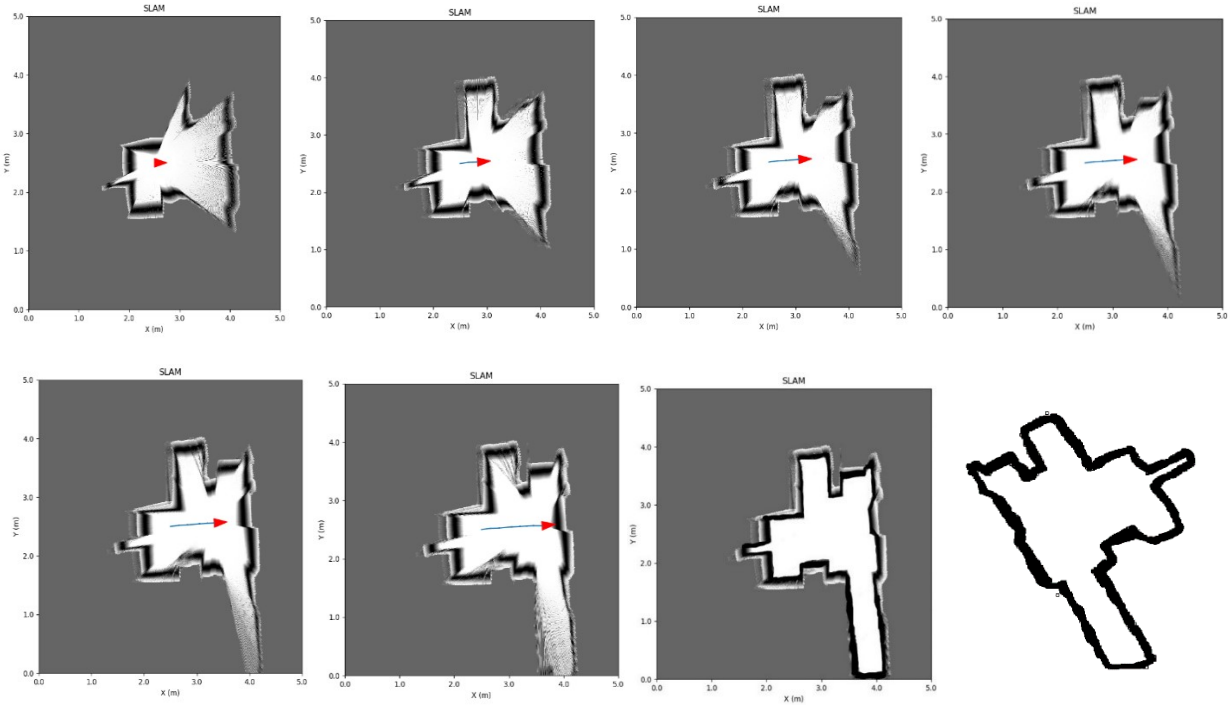

Figure 7. First use case or scenario.

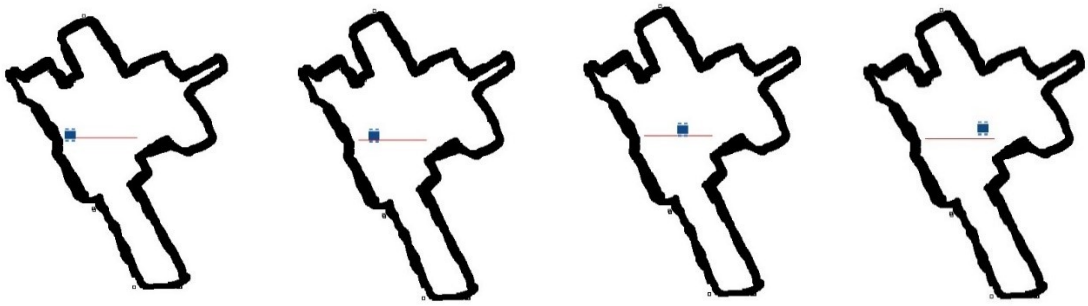

Figure 8. Second use case or scenario.

The measurement factors here would be the robot's ability to accomplish a mission or task without interaction with a human operator (i.e., navigate autonomously), as well as the closeness between the A-star algorithm generated path (AGP) in the form of waypoints and the robot's real traced path (RTP) with and without sensor fusion, implying its accuracy. Although, its ability to alter

said waypoints if only to avoid dynamic obstacles is implied.

With its real-time progress shown in Figure 8, its behavior as observed is plotted in Figure 9, and 10. Essentially, indicating a success in the intention of the study, to some degree of accuracy, also estimated.

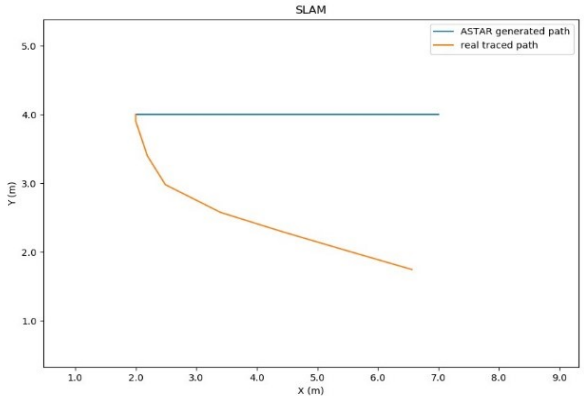

Figure 9. Performance for a straight trajectory (without sensor fusion).

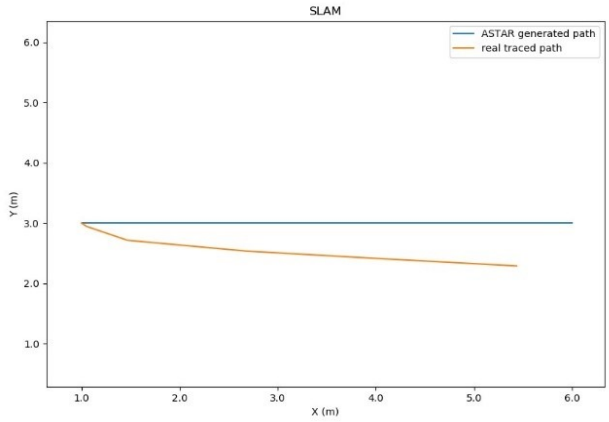

Figure 10. Performance for a straight trajectory (with sensor fusion).

In order to estimate the error, the mean deviation representing the average absolute deviation of points of the RTP from the AGP is calculated. For this, the mean Euclidean distance which is the sum of Euclidean distance at each corresponding point divided by the total number of the corresponding points, n, is employed.

$$\bar{x} = \frac{\sum_{i=0}^{n}\left(\sqrt{(x_{agp}-x_{rtp})^2+(y_{agp}-y_{rtp})^2}\right)}{n} \quad (64)$$

where (x_{agp}, y_{agp}) are A-star algorithm's generated path co-ordinates and (x_{rtp}, y_{rtp}) represents the robot's real traced path co-ordinates.

The closer the estimates are to zero, the more accurate the trace is. Using Eq. (64) with four sample corresponding waypoints for AGP and RTP from Figure 9, and 10.

Table 1. Average Euclidean Distance Error.

Implementation	Average Euclidean Distance error (m)
Without sensor fusion	1.5
With sensor fusion	0.45

Table 1 shows the estimated error of the localization implementation. With the inclusion of sensor fusion bringing the error closer to zero, offering a comparative accuracy of about 70%.

The robot's behaviour is further demonstrated in Figure 11 for a zig-zag trajectory as the rigidity of the A-star algorithm does not allow for shapes such as square, circle etc. to be naturally selected. The algorithm resorts to finding the optimal path to the goal which may not necessarily be square, circular etc.

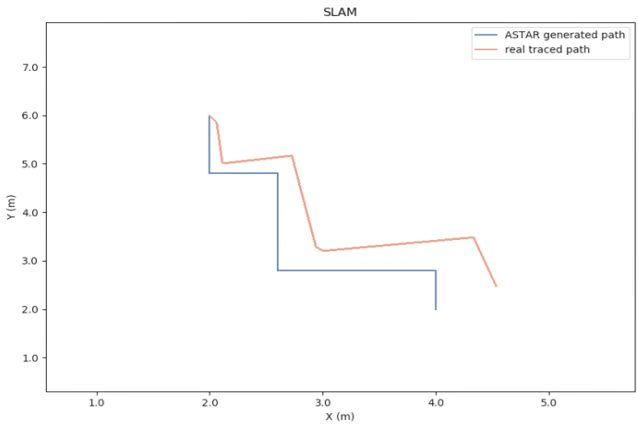

Figure 11. Performance for a zig-zag trajectory (with sensor fusion).

4. Conclusions

A differential drive robot with natural navigation was designed. The aim of which was educational, i.e., to ease intending engineer's approach on steps and calculations to be implemented for designing indoor mobile robots. The basic steps followed have been outlined and are reproducible. The code is also made available on GitHub, in python and is integrable in parts or as a whole in other differential drive robot related projects.

This study uses sensor fusion to limit the drift errors associated with encoder wheel odometry as it combines its results with the IMU. This in turn, successfully increased the accuracy of the localization and helped in map

creation. The natural navigation also included an ability to move from one specified point to another within the map while avoiding obstacle; this was achieved with the use of A-star search algorithm for the best estimated path and a tangency approach to avoiding obstacle. The approach utilized in this paper offers a not-so-simple yet not-so-complex combination of techniques that are computationally efficient and fast. However, more generally, the PID constant's values and the covariance matrix values of the sensors could be fine-tuned, with a further inclusion of the sigma update step in the UKF, for a more accurate trace. While employing a more flexible path search algorithm or technique might do even better for the robot's behavior.

Furthermore, in the future, the necessary research will be done to evaluate the best steps to analysis for stresses, feasible materials, weight considerations as well as aesthetics, also aimed at easing its integration for intending engineers designing mobile robots.

Declaration of Ethical Standards

The author(s) of this article declare that the materials and methods used in this study do not require ethical committee permission and/or legal-special permission.

Conflict of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Wei L., Cappelle C.,Ruichek Y., 2013. Camera/Laser/GPS Fusion Method for Vehicle Positioning Under Extended NIS-Based Sensor Validation. IEEE Transactions on Instrumentation and Measurement, **62**(11), pp. 3110-3122. https://doi.org/10.1109/TIM.2013.2265476.

[2] Bruno S., Oussama H., 2010. CoreSLAM: a SLAM Algorithm in less than 200 lines of C code. Mines ParisTech - Center of Robotics, Paris, FRANCE.

[3] Bajracharya S., 2014. BreezySLAM: A Simple, efficient, cross-platform Python package for Simultaneous Localization and Mapping. Student Papers, Record Group 38, Special Collections and Archives, Leyburn Library, Washington and Lee University, Lexington, VA.

[4] Tsardoulias E. G., Iliakopoulou A., Kargakos A. et al., 2016. A Review of Global Path Planning Methods for Occupancy Grid Maps Regardless of Obstacle Density. Journal of Intelligent & Robotic Systems, **84**(1), p.p. 829–858. https://doi.org/10.1007/s10846-016-0362-z.

[5] Zou Q., Sun Q., Chen L., Nie B., and Li Q., 2021. A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles. IEEE Transactions on Intelligent Transportation Systems. https://doi.org/10.1109/TITS.2021.3063477.

[6] Guran M., Fico T., Chovancova A., Duchon F., Hubinsky P., Dubravsky J., 2014. Localization of iRobot create using inertial measuring unit, 2014 23rd International Conference on Robotics in Alpe-Adria-Danube Region (RAAD), 2014, pp. 1-7, doi: 10.1109/RAAD.2014.7002261.

[7] Suriya D., Srivenkata S., Sundarrajan G., Kiran S., Ragul B., Vidhya B., 2016. A Robust Approach for Improving the Accuracy of IMU based Indoor Mobile Robot Localization. ICINCO (2) 2016: 436-445. DOI:10.5220/0005986804360445.

[8] Urrea C., Agramonte R., 2021. Kalman Filter: Historical Overview and Review of Its Use in Robotics 60 Years after Its Creation. Journal of Sensors, **2021**. https://doi.org/10.1155/2021/9674015.

[9] Hart P. E.; Nilsson N. J.; Raphael B., 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, **4**(2), pp.100–107. https://doi.org/10.1109/TSSC.1968.300136.

[10] Purnama H. S., Sutikno T., Alavandar S. and Subrata A. C., 2019. Intelligent Control Strategies for Tuning PID of Speed Control of DC Motor - A Review. 2019 IEEE Conference on Energy Conversion (CENCON), pp.24-30. https://doi.org/10.1109/CENCON47160.2019.8974782.

[11] Berntorp K., Årzén K. E., & Robertsson A., 2011. Sensor Fusion for Motion Estimation of Mobile Robots with Compensation for Out-of-Sequence Measurements. 11th International Conference on Control, Automation and Systems, pp. 211-216.

[12] Janai J., Güney F., Behl A., Geiger A., 2020. Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. Foundations and Trends® in Computer Graphics and Vision, **12**(1–3), pp 1-308.

[13] Khan M. S., Chowdhury S. S., Niloy N., Aurin F. T. Z., Ahmed T., 2018. Sonar-based SLAM Using Occupancy Grid Mapping and Dead Reckoning. TENCON 2018 - 2018 IEEE Region 10 Conference, pp. 1707-1712. doi: 10.1109/TENCON.2018.8650124.

[14] Mu X., He B., Zhang X., Song Y., Shen Y., Feng C., 2019. End-to-end navigation for Autonomous Underwater Vehicle with Hybrid Recurrent Neural Networks. Ocean Engineering, **194**. ISSN 0029-8018, https://doi.org/10.1016/j.oceaneng.2019.106602.

[15] Kang J. G., An S. Y., Kim S. and Oh S., 2009. Sonar based Simultaneous Localization and Mapping using a Neuro Evolutionary Optimization. 2009 International Joint Conference on Neural Networks, pp. 1516-1523, https://doi.org/10.1109/IJCNN.2009.5178826.

[16] Wang J., Liu J., Kato N., 2019. Networking and Communications in Autonomous Driving: A Survey. IEEE Communications Surveys & Tutorials, **21**(2), pp. 1243-1274. doi: 10.1109/COMST.2018.2888904.

[17] Chiu C.C., Hsu J.C., Leu J.S., 2016. Implementation and analysis of Hybrid Wireless Indoor Positioning with iBeacon and Wi-Fi. 2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), pp. 80-84. doi: 10.1109/ICUMT.2016.7765336.

[18] Li J. et al., 2018. PSOTrack: A RFID-Based System for Random Moving Objects Tracking in Unconstrained Indoor Environment. IEEE Internet of Things Journal, **5**(6), pp. 4632-4641. doi: 10.1109/JIOT.2018.2795893.

[19] Lin P.T., Liao C.A. and Liang S.H., 2021. Probabilistic Indoor Positioning and Navigation (PIPN) of Autonomous Ground Vehicle (AGV) Based on Wireless Measurements. IEEE Access, **9**, pp. 25200-25207. https://doi.org/10.1109/ACCESS.2021.3057415.

[20] Jensfelt P., 2001. Approaches to Mobile Robot Localization in Indoor Environments. (Doctoral thesis, Royal Institute of Technology (KTH), Stockholm, Sweden). Retrieved from http://www.diva-portal.org/smash/get/diva2:8964/FULLTEXT01.pdf

[21] Przemysławm P., Piotr K., 2020. Unscented Kalman filter application in personal navigation. Proc. SPIE 11442, Radioelectronic Systems Conference 2019, 114421C. DOI:10.1117/12.2564984

[22] Si L., Yu J., Wu W., Ma J., Wu Q., Li S., 2017. RMHC-MR: Instance selection by random mutation hill climbing algorithm with MapReduce in big data. Procedia Computer Science, **111**, pp.252-259. https://doi.org/10.1016/J.PROCS.2017.06.061.