OPTIMISING WIRELESS SENSOR NETWORK THE GEOGRAPHICAL DISTRIBUTION IN REAL TIME USING A LOW-COST MICROCONTROLLER.

Er. Parul Awasthi[1], Er. Anand Kumar Gupta[2], Prof. Ashutosh Singh[3]
[1,2]Assistant Professor ÙIET, CSJM University, Kanpur
[3]Professor Department of Electronics Engineering HBTU, Kanpur
Corresponding Author Mail id: akgietk@rediffmail.com

Abstract: This research presents a low-cost microcontroller-based system that uses pedometer measurements and communication between nodes in a wireless sensor network for localisation purposes. The proposed system performs effectively on a sparse network, unlike other methods that rely on good network connectivity.to solve nonlinear equations in real time during localisation, two optimisation algorithms have been investigated: The Gauss-Newton algorithm and particle swarm optimisation. The localisation and optimisation methods were built using a microcontroller. Experiments were conducted to evaluate efficiency.

Keywords: Microcontrollers, particle swarm optimization (PSO), wireless sensor network (WSN).

Introduction:

A Wireless Sensor Network (WSN) is a system consisting of numerous wirelessly interconnected heterogeneous sensor nodes that are spatially dispersed over a designated area of interest. Wireless Sensor Networks (WSN) have garnered significant research interest in recent years owing to their potential applications across several domains. It has been utilised in applications including ecological and natural habitat surveillance, medical instrumentation, industrial automation, and military surveillance [1], [2]. Sensor nodes should generally be cost-effective and compact for extensive deployment. Additionally, the sensor node's power consumption must be minimal to extend its operating lifespan for many years to come.

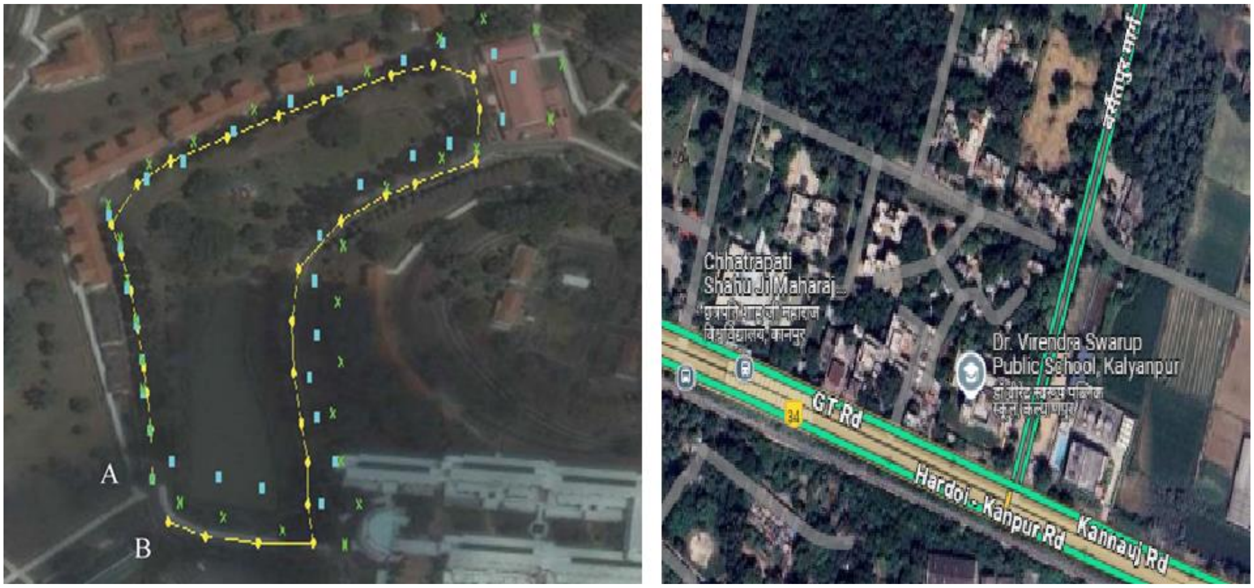

Figure: 1 Experimental findings (single-direction approach) (x = pedometer, • = true position, and □ = GNA).

Recent improvements in microelectronics have produced adaptable microcontrollers utilised in various applications, including motor driving, light dimmer control, uninterruptible power systems, and power sources. [3][9]. In Wireless Sensor Networks (WSN), the majority of systems utilise a microcontroller as the core unit to execute numerous functions, including sensor data acquisition, network protocol implementation, signal processing, and power management. A primary problem of wireless sensor networks is ascertaining the physical locations of sensor nodes. This can be accomplished by outfitting all sensor nodes with Global Positioning System (GPS) technology. Nonetheless, this technology is expensive, requires substantial power, and is constrained for outdoor use. Several GPS-independent localisation techniques have been examined for dense networks [10] [15]. They can generally be categorised as range-free and range-based algorithms. Range-free algorithms [10], [11] operate under the assumption that distance or angle information is inaccessible, utilising network connectivity to estimate node locations. Range-based algorithms [11] [15] necessitate distance measurements from anchor nodes and employ triangulation or maximum likelihood estimate methods to determine the positions of unknown nodes. Maximum likelihood estimate is used in most range-based methods due to its superior accuracy, but with increased computing demands and memory consumption. Most present works emphasise theoretical advancement while neglecting computational costs and implementation aspects. In actuality, there are significant limitations on computational power and memory. As a result, numerous advanced optimisation methods will be impractical.

This study presents the real-time Gauss–Newton algorithm (GNA) and the particle swarm optimisation (PSO) utilising the probability field technique for sensor node localisation. In the system being analysed, a deployment agent (DA), such as a pedestrian or an unmanned aerial vehicle equipped with a positioning sensor, is responsible for deploying the sensor nodes. Examine the perimeter deployment illustrated in Fig. 1 for a sparse network. A DA transitions from an initial position A, traverses an area of interest, and concludes at position B to install the sensor nodes. For the sake of the experiment, the designated agent in this study is an individual walking while utilising a pedometer and an electronic compass. The system monitors the agent's movement throughout the deployment. Following the deployment, the sensor nodes transmit beacon packets to ascertain the distances between the nodes based on the received power strength of the RF signals. Utilising both deployment and communication ranging data with the suggested method enhances localisation accuracy.

Section II indicates that solving a nonlinear equation is necessary to ascertain the optimal placement of the sensor nodes. A potential solution to the issue is the GNA. Nonetheless, the GNA is a local optimisation technique and does not ensure global convergence. An alternate method is to employ a global optimiser, such as Particle Swarm Optimisation (PSO). This work examines and implements both GNA and PSO on the same platform. Furthermore, its efficacy in identifying optimal solutions across various operating situations has been examined. Experimental findings indicate that the GNA is more efficacious when the pedometer error is less than 25%. It is shown that the PSO demonstrates greater robustness in the presence of significant pedometer errors, whereas the GNA may converge to a local minimum. Moreover, the GNA entails matrix inversion during its iterations and may infrequently exhibit instability. Consequently, GNA is a viable optimiser for this application solely if the integrated pedometer possesses high accuracy. Alternatively, the PSO is favoured.

This paper is organised as follows: Section II delineates the problem formulation employing the proposed probability-based function and the error modelling. Section III presents the sensor node architecture and the implementation of the GNA and PSO methodologies for localisation. Section IV delineates the laboratory evaluation of the system, whereas Section V elucidates the experimental system and gives several outdoor experimental outcomes. Section VI finishes this document.

2.0 PROBABILITY-BASED LOCALISATION APPROACH.

The following paragraphs introduces a localisation method that integrates data from the received signal strength indicator (RSSI) and deployment details. The agent initiates the deployment of the first sensor node from a designated site A, as illustrated in Fig. 1. Upon deployment, the locations of the sensor nodes are initially ascertained using the pedometer and compass navigation system. Thereafter, the sensor nodes interact with adjacent nodes to share the RF signal intensity that can be received. This work presents a probability-based localisation approach designed to enhance localisation accuracy by utilising deployment measurements and RSSI-based distance estimations from neighbouring nodes to develop the likelihood function for the unknown node's precise position. Utilising information from two distinct sources enables improved outcomes via data fusion.

The suggested methodology encompasses two localisation modes: unidirectional and bidirectional. In unidirectional mode, each unidentified node exclusively employs RSSI measurements from previously placed sensors. The bidirectional mode presupposes knowledge of the last sensor node's position. Furthermore, the network is capable of communicating in both forward and backward directions. This section outlines the techniques for constructing the likelihood function and formulating the optimisation for an unidentified node.

2.1 Problem Formulation

Considering a sensor node i that has been deployed with the estimated position Φdi; its actual position Φi is regarded as proximate to Φdi with a specific probability. According to probability theory, the likelihood function of the actual position Φi is the conditional probability density function of Φdi given the actual position Φi.Identify this likelihood equation as the installation probability function for the unidentified node i.

$$L_{di}(\Phi_i) = L(\Phi_i; \Phi_{di}) = P(\Phi_{di} \mid \Phi_i) \qquad (1)$$

where the diminutive "d" signifies deployment measurement and "i" represents the node's index. Given that Φdi follows a bivariate normal distribution with the real position Φi as the mean, the deployment probability function is derived as

$$L_{di}(\Phi_i) = L_{di}(x_i, y_i) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{1}{2}\left(\frac{(x_i - x_{di})^2}{\sigma_x^2} + \frac{(y_i - y_{di})^2}{\sigma_y^2}\right)\right) \quad (2)$$

The standard deviations σ_x and $\sigma_y$are estimated as $\sigma_x = x_m \cdot p$ and $\sigma_y = y_m \cdot p$, where x_m and y_m are the measured distance vector derived from the walking distance and direction. It is assumed that the error factor of the pedometer and the compass, after projection onto the x- and y-coordinates, is p.

If sensor node i can receive the beacon packet from localised node j, the distance d_{ij} between the two nodes can be determined using the RSSI measurements as d_{mij}. Nonetheless, this measurement is generally characterised by significant noise. Utilising the estimated location ˆ Φ_j of j and the estimated distance d_{mij}, the probability function of the actual position Φ_i may be calculated. Designate this function as the radio extending likelihood equation.

$$L_{rij}(\Phi_i) = P(d_{mij} \mid \Phi_i, \hat{\Phi}_j) \qquad (3)$$

The subscript "r" signifies "radio ranging," whereas the subscript "j" represents the index of the localised node.

The measured distance dm based on RSSI is often considered to follow a Gaussian distribution: dm~N(d,(d · r)²), where r represents the range error factor. The probability function of the actual distance d, given d_m and r, is

$$P(d_m \mid d) = \frac{1}{\sqrt{2\pi} d \cdot r} \exp\left(-\frac{(d - d_m)^2}{2(d \cdot r)^2}\right). \qquad (4)$$

Let

$\delta(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ be the distance between locations A and B. For an unknown node i and a localized node the likelihood function of Φi is

$$P(d_{mij} \mid \Phi_i, \hat{\Phi}_j) = P\left(d_{mij} \mid \delta(\Phi_i, \hat{\Phi}_j)\right) = \frac{1}{\sqrt{2\pi}(d_{mij}r)} \exp\left(-\frac{\left(\delta(\Phi_i, \hat{\Phi}_j) - d_{mij}\right)^2}{2(d_{mij}r)^2}\right) \quad (5)$$

Let Ji denote all the communicating localized nodes. Since the installation and RSSI range measurements are uncorrelated with one another, the overall likelihood function is derived through the addition of all the likelihood measures for the unidentified node i. Consequently, through combining (2) and (5), we acquire

$$\begin{aligned}\overline{L_i}(\Phi_i) &= L_{di}(\Phi_i) \times \prod_{j \in J_i} L_{rij}(\Phi_i) \\ &= \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{1}{2}\left(\frac{(x_i - x_{di})^2}{\sigma_x^2} + \frac{(y_i - y_{di})^2}{\sigma_y^2}\right)\right) \\ &\quad \times \prod_{j \in J_i}\left(\frac{1}{\sqrt{2\pi}(d_{mij} \cdot r)} \right. \\ &\quad\quad \left. \times \exp\left(-\frac{\left(\delta(\Phi_i, \hat{\Phi}_j) - d_{mij}\right)^2}{2(d_{mij} \cdot r)^2}\right)\right).\end{aligned} \quad (6)$$

Taking natural logarithmof(6)yields

$$\ln\left(\overline{L_i}(\Phi_i)\right) = \alpha - S(\Phi_i) \quad (7)$$

Where$\alpha = -\ln(2\pi\sigma_x\sigma_y) - j \in Ji \ln\sqrt{2\pi}\hat{}\ d_{ij}r$ is a constant andthefunctionSisdefinedas

$$S(\Phi_i) = \frac{(x_i - x_{di})^2}{\sigma_x^2} + \frac{(y_i - y_{di})^2}{\sigma_y^2} + \sum_{j \in J_i} \frac{\left(\delta(\Phi_i, \hat{\Phi}_j) - d_{mij}\right)^2}{(d_{mij} \cdot r)^2} \quad (8)$$

To determine the point Φi that maximises Li(Φi), the resulting function S must be minimised. Consequently, the localisation issue transforms into an optimisation challenge.

2.2 Discussion

The goal-setting function (8) is typically multifunctional when measurement errors from the pedometer and RSSI range are significant. For local optimisation algorithms such as GNA, it is necessary for the initial guess to be sufficiently proximate to the global minimum to ensure global convergence. In this application, the deployed position of the pedometer is utilised as the initial estimate.Consequently, a more precise pedometer measurement will result in a closer initial estimate of the global minimum.When the pedometer error factor is minimal ($p<0.25$ from evaluation studies), GNA typically converges to the global minimum, and GNA is chosen as the optimiser because to its simplicity.However, when the pedometer error is substantial ($p > 0.25$), GNA could come closer to a local minimum if the starting estimate is excessively distant from the optimal position.In such a circumstance, a global optimisation strategy is essential, and Particle Swarm Optimisation (PSO) is used. Comprehensive performance analyses of optimisation methodologies are offered in Section IV.

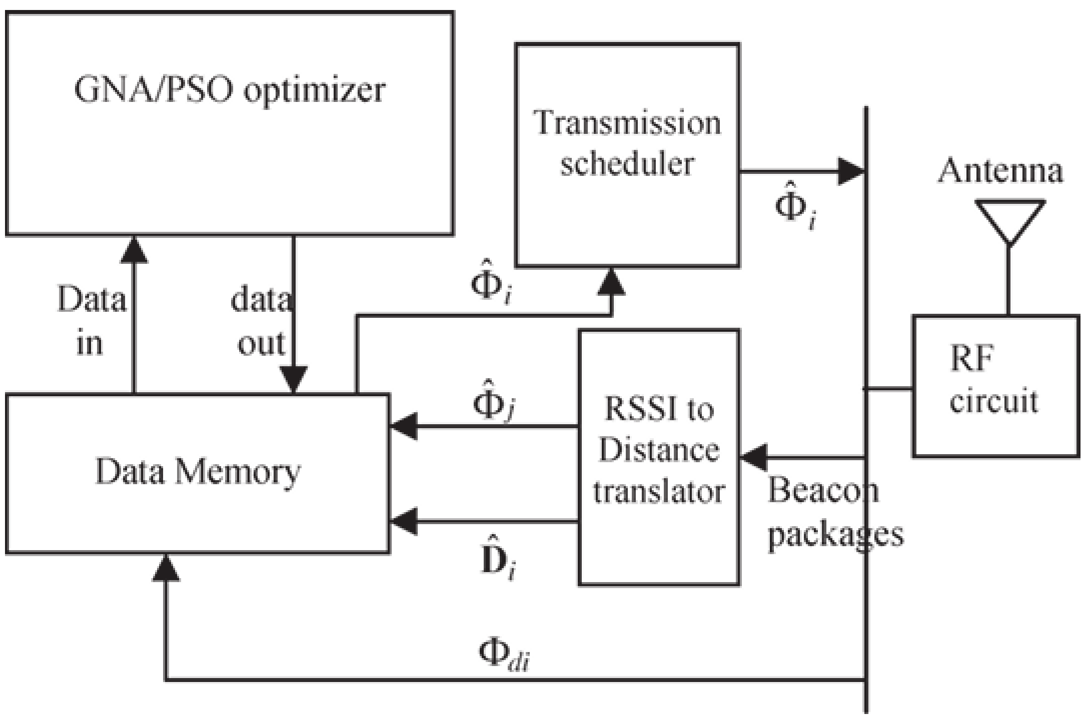

Figure 2. Block illustration of a sensor nodes.

3.0 SENSORS NODE ARCHITECTURE AND IMPLEMENTATION

Figure 2 illustrates the block diagram of the sensor node architecture created for this application.It comprises five principal components: the RF system, RSSI to distance converter, GNA/PSO optimiser, transmission scheduler, and data memory. As indicated in Section II, after the initial sensor node is deployed at a predetermined position, it commences transmitting its location to other nodes.For each consecutive unknown node i being put in place, it acquires its deployment statistics Φ_{di} through its RF system from the pedometer/compass system. Subsequently, its transmission scheduler will solicit neighbouring nodes to transmit their beacon messages including their estimated positions.In addition to receiving the beacon message, the RSSI to distance translator also measures the RSSI values of the received beacons and converts them to a distance ˆ D_i.The GNA/PSO optimiser will ascertain the estimated position of the sensor node, denoted as ˆ Φ_i, by integrating the deployment and inter-node distance information using (8).The transmission scheduler responds to requests from other nodes and broadcasts the sensor node's beacon packet.

3.1 RF System

The RF infrastructure is utilised to receive beacon packets from neighbouring sensor nodes and stride/heading data from the pedometer, as well as to emit its own beacon packets.

3.2 TransmissionScheduler

The Data Transfer Scheduler implements the media access control protocol to prevent data collisions.For the convenience of experimentation, a straightforward polling approach is employed.Upon deployment, a sensor node will acquire its pedometer data and subsequently initiate polling of adjacent sensor nodes to transmit its beacon packets.Upon successful localisation of the sensor, it will transition to listening mode and await polling from other nodes.

3.3 RSSI to Distance Translator

The RSSI for distance translators employs a function RtoD that encompasses the following equation:

$$\hat{d}_j = 10^{\frac{R_0 - R_j}{2n}} \quad (9)$$

Table 1: Pseudocode for RSSI to Proximity The interpreter

```
// RSSI to Distance Translator
// Constants (you may need to adjust these based on your environment)
RSSI_At_1m = -40  // RSSI value at 1 meter distance (this is an example value, adjust as needed)
n = 2             // Path loss exponent (can vary depending on the environment)
Function RSSI_To_Distance(RSSI):
    // Calculate the distance based on the RSSI value
    Distance = 10 ^ ((RSSI_At_1m - RSSI) / (10 * n))
    Return Distance
// Main program
Start:
    // Prompt the user for the RSSI value
    Print "Enter the RSSI value (in dBm): "
    Input RSSI

    // Call the function to calculate the distance
    Distance = RSSI_To_Distance(RSSI)
```

```
    // Display the result
    Print "The estimated distance is: ", Distance, " meters."
End
```

Beacon package contains:
j : sender's node ID
$\hat{\Phi}$: sender's estimated position
R : RSSI value of the package

Step 1: Poll a neighboring node, and extract its estimated position.
$\hat{\Phi}_j = \hat{\Phi}$;

Step 2: Poll the node 10 times and compute the average RSSI value.

$R_j = 0$;
for i =1; i<=10; i++
    poll node j;
    $R_j = R_j + R$;
end
$R_j = R_j / 10$;

Step 3: Determine the distance
$\hat{d}_j = \mathbf{RtoD}(R_j)$;

Step 4: Go to **step 1** if there are other neighboring nodes.

where Rj is the acquired RSSI value; ˆdj is the corresponding distance; Ro is the received RSSI value at a distance of 1 meter, and n is the path loss exponent. Ro and n are calibrated values acquired prior to deployment.

Typically, RSSI measurements frequently experience burst interferences from diverse noise sources. To enhance the estimation, the mean RSSI data from ten beacons are utilised to deduce distance. Table I presents the pseudocode for the RSSI to Distance Transporter.

3.4 GNA/PSO Optimizer

3.4.1 GNA: The GNA is recognised for addressing nonlinear least squares estimation issues [17], [18]. It is an iterative method that necessitates the user to supply an initial estimate of its answer.

Given m functions fi (i = 1,...,m) of n variables β = (β1, β2,...,βn), where m > n, the Generalised Newton Algorithm (GNA) can be employed to determine the minimum of the sum of squares.

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} f_i^2(\boldsymbol{\beta}). \quad (10)$$

Commencing with an initial estimate β[0], the procedure advances through iterations.

$$\boldsymbol{\beta}[k+1] = \boldsymbol{\beta}[k] + \boldsymbol{\Delta}_{\mathbf{k}} \qquad (11)$$

with the increment Δk satisfying the normal equation

$$\left(\mathbf{J_f^T J_f}\right) \boldsymbol{\Delta}_{\mathbf{k}} = -\mathbf{J_f^T f} \qquad (12)$$

where f represents the vector of functional fi, and J_f is the Jacobian matrix of J_f concerning β[k]. In the localisation problem, the predicted deployment position Φdi serves as the first approximation. Table II illustrates the

```
START
   // Step 1: Initialize population
population_size = 100
   population = GenerateRandomPopulation(population_size)
   // Step 2: Define parameters for genetic algorithm
crossover_rate = 0.8
mutation_rate = 0.1
   generations = 1000
   // Step 3: Evaluate initial population
EvaluatePopulation(population)
   // Step 4: Repeat for a specified number of generations
   FOR generation = 1 TO generations DO
      // Step 5: Select individuals for reproduction (parent selection)
selected_parents = SelectParents(population)
      // Step 6: Perform crossover (mating) to create offspring
      offspring = Crossover(selected_parents, crossover_rate)
      // Step 7: Perform mutation on offspring
mutated_offspring = Mutate(offspring, mutation_rate)
      // Step 8: Evaluate offspring's fitness
EvaluatePopulation(mutated_offspring)
      // Step 9: Select individuals for next generation (survival selection)
      population = SelectNextGeneration(population, mutated_offspring)
      // Step 10: Optionally print or track the best solution
best_solution = GetBestSolution(population)
      PRINT "Generation ", generation, " Best Solution: ", best_solution
```

```
   END FOR
   // Step 11: Final output
best_solution = GetBestSolution(population)
   PRINT "Best solution found after ", generations, " generations: ", best_solution
END
// Function to Generate a random initial population
Function GenerateRandomPopulation(population_size):
   population = []
   FOR i = 1 TO population_size DO
      individual = GenerateRandomIndividual()
      ADD individual TO population
   END FOR
   RETURN population
// Function to Evaluate the fitness of the population
Function EvaluatePopulation(population):
   FOR each individual IN population DO
individual.fitness = CalculateFitness(individual)
   END FOR
// Function to select parents for reproduction (based on fitness)
Function SelectParents(population):
selected_parents = []
   FOR i = 1 TO population_size / 2 DO
      parent1, parent2 = SelectTwoParents(population)
      ADD parent1, parent2 TO selected_parents
   END FOR
   RETURN selected_parents
// Function to perform crossover and create offspring
Function Crossover(parents, crossover_rate):
   offspring = []
   FOR each pair of parents IN parents DO
      IF random() <crossover_rate THEN
         child = PerformCrossover(parent1, parent2)
         ADD child TO offspring
      END IF
   END FOR
   RETURN offspring
// Function to mutate offspring
Function Mutate(offspring, mutation_rate):
   FOR each individual IN offspring DO
      IF random() <mutation_rate THEN
MutateIndividual(individual)
      END IF
   END FOR
```

```
    RETURN offspring
// Function to select the next generation
Function SelectNextGeneration(population, offspring):
combined_population = population + offspring
new_population = SelectBestIndividuals(combined_population)
    RETURN new_population
// Function to get the best solution
Function GetBestSolution(population):
best_individual = Individual with highest fitness in population
    RETURN best_individual
```

Step 1: Initialization

k=0;

$\Phi_i[k] = \Phi_{di}$;

Step 2: Find necessary matrixes for GNA

$$\mathbf{f}(\Phi_i) = \left[\frac{x_i - x_{di}}{\sigma_x} \quad \frac{y_i - y_{di}}{\sigma_y} \quad \frac{\delta(\Phi_i, \hat{\Phi}_{j1})}{\hat{d}_{ij1} \cdot r} - \frac{1}{r} \quad \cdots \quad \frac{\delta(\Phi_i, \hat{\Phi}_{jn})}{\hat{d}_{ijn} \cdot r} - \frac{1}{r} \right];$$

$\mathbf{J_f} = \mathbf{Jacobin}(\mathbf{f}, \Phi_i)$

$$= \begin{bmatrix} \frac{1}{\sigma_x} & 0 \\ 0 & \frac{1}{\sigma_y} \\ \frac{x_i - \hat{x}_{j1}}{\hat{d}_{ij1} \cdot r \cdot \delta(\Phi_i, \hat{\Phi}_{j1})} & \frac{y_i - \hat{y}_{j1}}{\hat{d}_{ij1} \cdot r \cdot \delta(\Phi_i, \hat{\Phi}_{j1})} \\ \vdots & \vdots \\ \frac{x_i - \hat{x}_{jn}}{\hat{d}_{ijn} \cdot r \cdot \delta(\Phi_i, \hat{\Phi}_{jn})} & \frac{y_i - \hat{y}_{jn}}{\hat{d}_{ijn} \cdot r \cdot \delta(\Phi_i, \hat{\Phi}_{jn})} \end{bmatrix};$$

Step 3: Determine the incremental direction

$\psi = (\mathbf{J}_f^T \mathbf{J}_f)^{-1}$;

$\Delta_k = -\psi \mathbf{J}_f^T \mathbf{f}$;

Step 4: Line search

$\alpha = 1$;

while $S(\Phi_i[k] + \alpha\Delta_k) \geq S(\Phi_i[k])$

$\alpha = \alpha / 2$;

end

$\alpha_k = \alpha$;

Step 5: Update

$\Phi_i[k+1] = \Phi_i[k] + \alpha_k \Delta_k$;

Step 6: Check for stopping criterion

$\delta(\Phi_i[k+1], \Phi_i[k]) < 0.01$

Update iteration index k and go to step 2 if the stopping criterion is not satisfied.

Pseudocode for the GNA, incorporating a basic line-search algorithm.

The pseudocode indicates that the approach entails matrix inversion in Step 3. The iteration will be unsuccessful if the matrix $J^T_f J_f$is singular. This matrix is theoretically non-singular in this localisation problem. Nevertheless, the matrix may approach singularity due to the microcontroller's low precision, occasionally failing to converge to the halting requirement. This matter will be addressed in a subsequent section.

3.4.2 PSO: This paper utilises the PSO method described in [19] for the global optimisation of sensor node positions. Table III presents the pseudocode utilised for the implementation. Particle Swarm Optimisation (PSO) has been applied in diverse fields, including robotics and antenna design [20]–[25]. Like other heuristic algorithms, such as the genetic algorithm [26], [27], the Particle Swarm Optimisation (PSO) method begins with a population of random solutions, referred to as particles. Each particle monitors its optimal fitness solution, referred to as pbest. Additionally, the optimiser retains the global best fitness solution, referred to as gbest. At every time step, the PSO optimiser modifies the acceleration of two members, which correspond to the x- and y-coordinates of the sensor node.

```
START
   // Step 1: Define parameters for PSO
population_size = 100          // Number of particles
max_iterations = 1000          // Maximum number of iterations
inertia_weight = 0.5           // Inertia weight (controls exploration)
cognitive_coeff = 1.5          // Cognitive coefficient (personal best influence)
social_coeff = 1.5             // Social coefficient (global best influence)
   // Step 2: Initialize particles
   particles = InitializeParticles(population_size)
   // Step 3: Initialize global best position and fitness
global_best_position = null
global_best_fitness = infinity
      // Step 4: Iterate until maximum iterations or convergence
   FOR iteration = 1 TO max_iterations DO
            // Step 5: Update each particle's velocity and position
      FOR each particle IN particles DO
         // Step 5.1: Calculate fitness of the current particle
particle.fitness = EvaluateFitness(particle.position)
                  // Step 5.2: Update particle's personal best position
         IF particle.fitness<particle.best_fitness THEN
particle.best_fitness = particle.fitness
particle.best_position = particle.position
         END IF
                  // Step 5.3: Update global best position
         IF particle.fitness<global_best_fitness THEN
```

```
global_best_fitness = particle.fitness
global_best_position = particle.position
          END IF
                    // Step 5.4: Update particle's velocity using the PSO velocity update equation
particle.velocity = inertia_weight * particle.velocity
                           + cognitive_coeff * random() * (particle.best_position - particle.position)
                           + social_coeff * random() * (global_best_position - particle.position)
                    // Step 5.5: Update particle's position
particle.position = particle.position + particle.velocity
                END FOR
       // Step 6: Optionally print or track the global best solution
       PRINT "Iteration ", iteration, " Global Best Fitness: ", global_best_fitness
   END FOR
   // Step 7: Output the global best solution
   PRINT "Best solution found after ", max_iterations, " iterations: "
   PRINT "Position: ", global_best_position
   PRINT "Fitness: ", global_best_fitness
END
// Function to Initialize particles with random positions and velocities
Function InitializeParticles(population_size):
   particles = []
   FOR i = 1 TO population_size DO
      particle = CreateRandomParticle()
      ADD particle TO particles
   END FOR
   RETURN particles
// Function to Evaluate fitness of a given particle (custom based on the problem)
Function EvaluateFitness(position):
   // This should be problem-specific: calculate the fitness value based on the position
   fitness = CalculateFitness(position)
   RETURN fitness
// Function to create a random particle (initialize position and velocity)
Function CreateRandomParticle():
   particle = {}
particle.position = GenerateRandomPosition()   // Random position in search space
particle.velocity = GenerateRandomVelocity()   // Random initial velocity
particle.best_position = particle.position     // Initial personal best is the starting position
particle.best_fitness = infinity               // Initial personal best fitness is very poor (infinity)
   RETURN particle
```

```
// Function to generate a random position within the problem's search space
Function GenerateRandomPosition():
   // This should generate a random value or vector within the valid range of the problem space
   position = random_value_in_range()
   RETURN position
// Function to generate a random velocity
Function GenerateRandomVelocity():
   // Typically, velocity is set to be a small value initially
   velocity = random_small_value()
   RETURN velocity
// Function to calculate the fitness of a position (problem-dependent)
Function CalculateFitness(position):
   // This is the function that calculates the fitness value based on the position
   // For example, it could return the value of the objective function for the given position
   fitness = some_function(position)
   RETURN fitness
```

The particle will move towards its personal best (pbest) and global best (gbest) positions with a stochastic weight. This investigation examines the composition of each particle's state.

4.0 Inspection of systems

The microcontroller in question (Microchip PIC18LF4620) has been running the suggested GNA and PSO algorithms. Microcontroller features 3968-B SRAM data memory and 64-kB Flash program memory. It runs forty MHz in a clock rate. Both algorithms have been written in C language utilising floating point structure for simplicity of development. The Microchip MPLAB C18 compiler compiles the C programs then downloads them to the Flash memory using the MPLAB ICD2 debugger.

Together, the routines for RSSI to distance translators and transmission scheduler take roughly 4-kB program memory. The codes for the GNA and PSO optimisers call for roughly 8- and 12-kB RAM, respectively.

We assess the two optimisation techniques in a laboratory environment in the next conversation. Here we bypass the RSSI for distance translators as well as the transmission scheduler. From their predicted distributions, the required data pedometer and RSSI distance measurements randomly generates themselves.

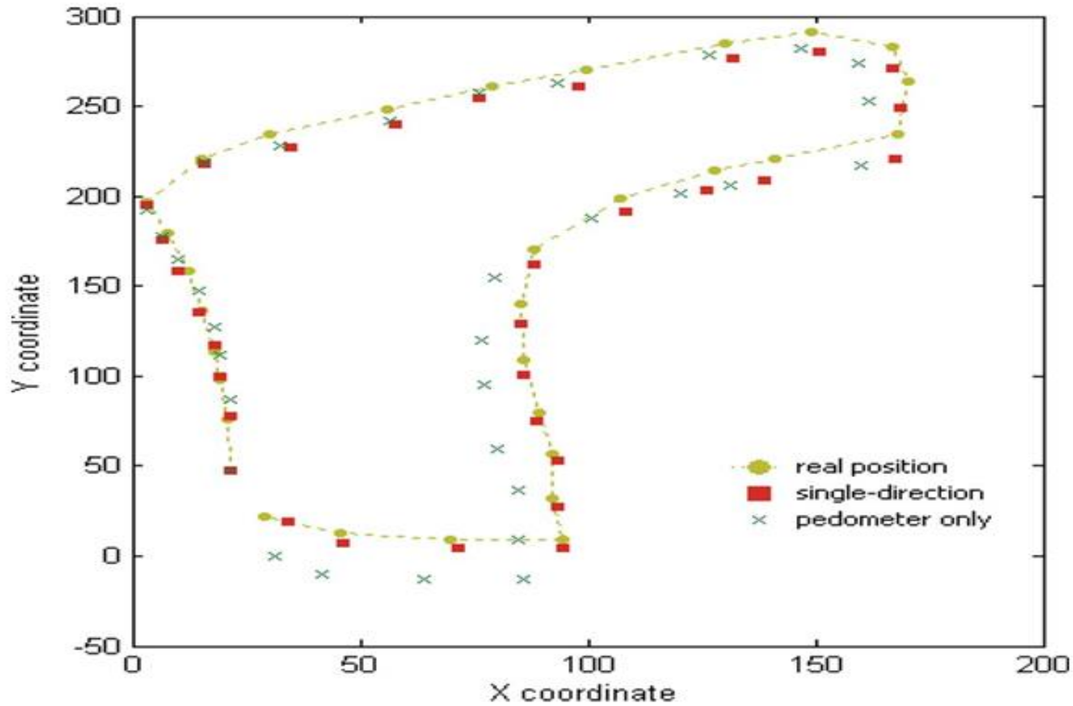

Figure 3: Single-direction system evaluation of a sparse network.

As seen in Fig. 3, it is assumed that the network to be evaluated is sparse. It comprises thirty-one nodes positioned generally along the path. utilising single-direction approach with GNA, Fig. 3 presents an example of the localised network; utilising bidirection method with GNA, Fig. 4 illustrates the localised outcome.

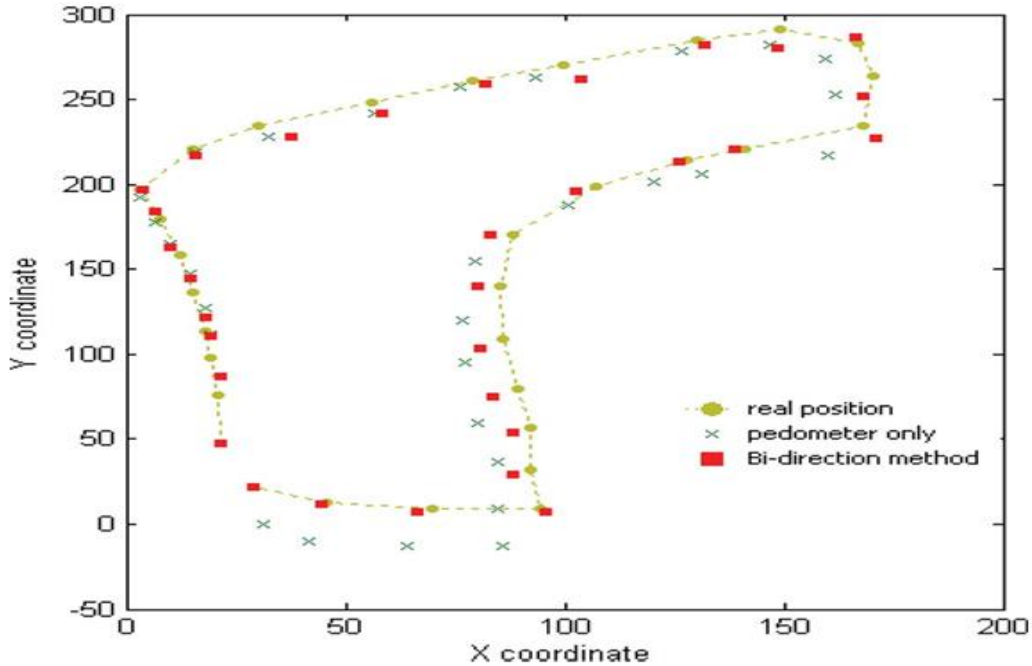

Figure 4: Bidirectional based system evaluation of a sparse network.

Not presented for brevity, the localisation result using PSO is virtually exact to the GNA. From this work, the variation between the projected placements using these two optimisation techniques for every node usually is less than 0.1 units. Here we have utilised a 0.2 pedometer and a 0.15 range error factor. Assumed to be the communication range is 60 units.

With GNA as the optimisation method, Fig. 5 displays the localisation error under several range and pedometer error factors for both single- and bidirection approaches. Once more, the PSO localisation results are rather close to the GNA. If $p = 0.3$, the difference for every data point is smaller than 0.1 unit; brevity keeps this from showing. All things considered, with the same measurements both GNA and PSO can search the same optimum. The two algorithms may converge from distinct directions, so the small variations are caused by termination at different points when stopping criterion is satisfied.

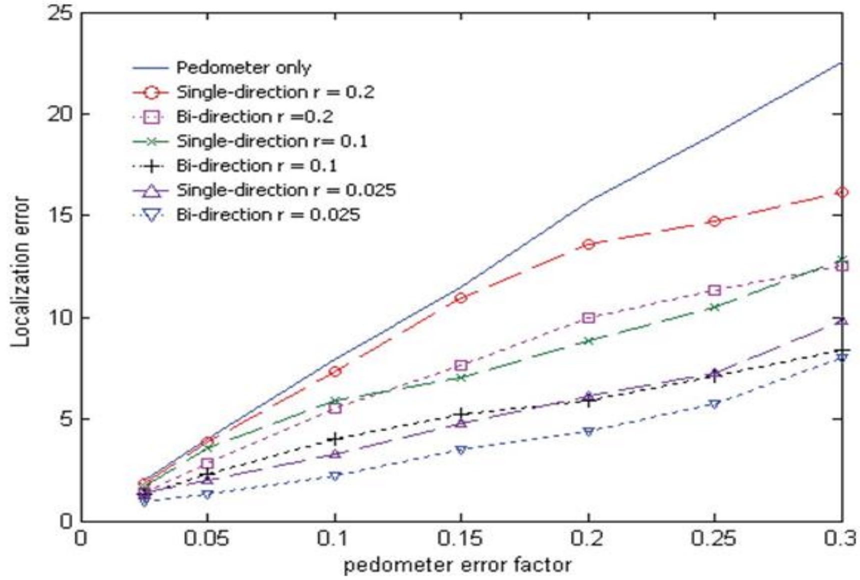

Fig. 5: Under varying measuring accuracy, average localisation error.

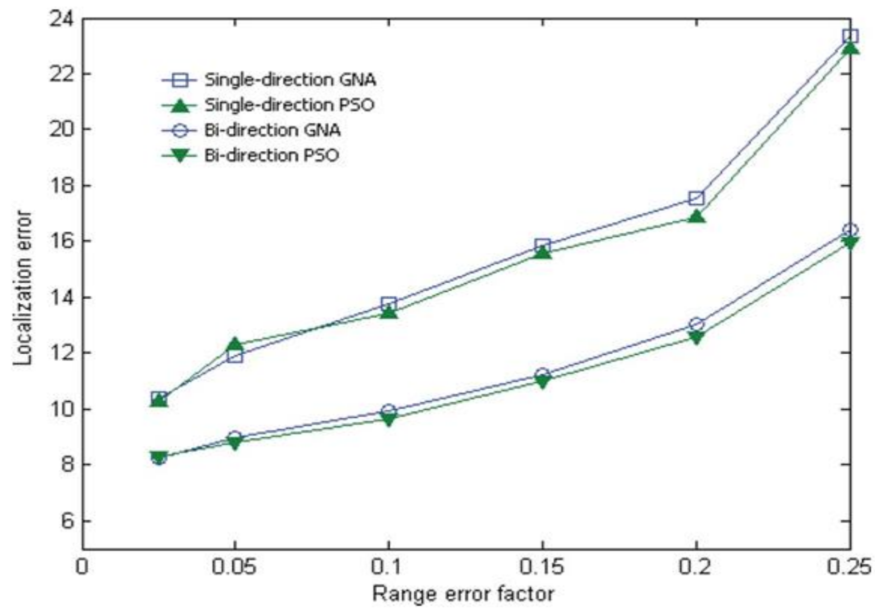

Figure 6: Average localisation mistake at p = 0.35.

Nevertheless, as demonstrated in Fig. 6, there is some variation between the localisation outcomes of these two techniques given very great pedometer error (p>0.3). PSO typically shows superior localisation. This is true because GNA's poor first predictions cause it to periodically converge to local minima.

4.1 Calculating Computational Expenses

In general, GNA needs less executive time than PSO for computational costs. From our analysis, it is also seen that the GNA computes the second derivative of the objective function most of the execution time. Objectives function evaluations occupy just roughly 15% of the execution time. Each sensor node has 2.5 average neighbours for single-direction method and 5 neighbours for bidirection method out of a communication range of 60 units.

Table IV localises a single node for various network densities by matching the average execution time required by the two techniques. Here the pedometer error factor is 0.2. Table IV makes it clear that the PSO needs double the computation time.

For modest pedometer inaccuracy, therefore, local optimisation techniques like GNA are advised. From the table, it is also noted that as the number of neighbouring nodes rises, both the techniques require more calculation time.

TABLE IV: an average execution time to localizes lender versus varying amount of neighbours

Number of neighboring nodes	GNA [s]	PSO [s]	Time ratio GNA/PSO
1	0.283	0.609	0.465
2	0.384	0.843	0.456
3	0.512	1.071	0.478
4	0.598	1.302	0.459
5	0.694	1.533	0.453
6	0.785	1.764	0.445
7	0.983	1.983	0.496

Table V: Mean Performance Time Tolerable Based on Lender versus unusual PEDO Meter Error Percentage

Pedometer error factor	GNA [s]	PSO [s]	Time ratio GNA/PSO
0.05	0.297	0.944	0.315
0.10	0.330	0.902	0.366
0.15	0.378	0.986	0.383
0.20	0.449	1.017	0.441
0.25	0.528	0.896	0.589
0.30	0.556	0.989	0.562

Table V demonstrates the way the pedometer's accuracy influences the GNA computation need. Every sensor node here averages 2.5 neighbours. The GNA calls for additional execution time to get the necessary precision as the pedometer error factor gets bigger. This is so as the first guess will be far from the ideal location. Conversely, the PSO execution time is rather constant. This attracts the PSO for a system with significant pedometer error factor.

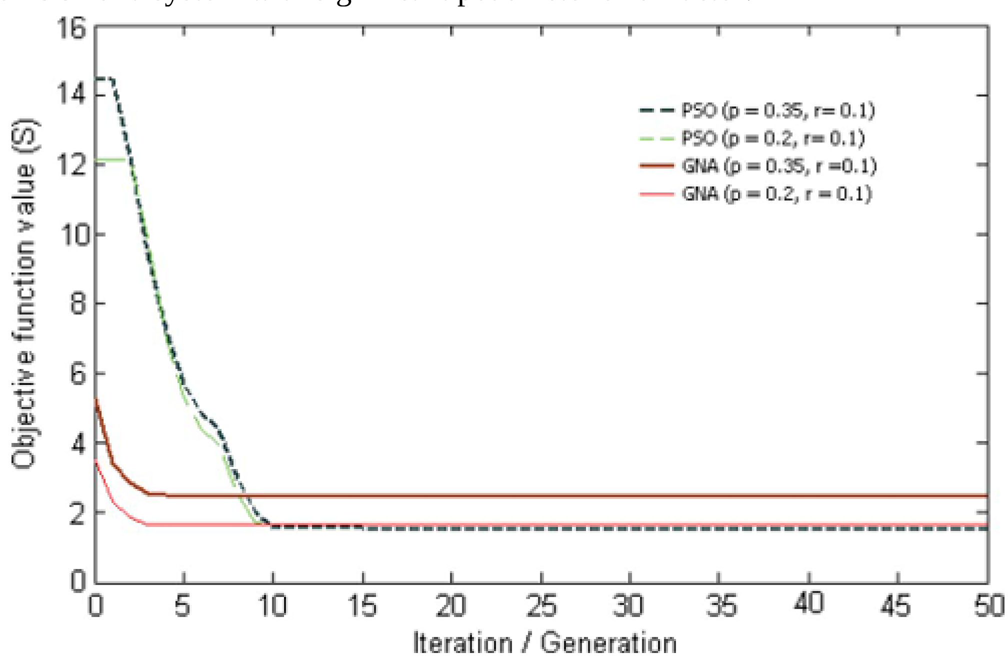

Figure 7 illustrates the average function objective value in relation to the number of iterations/generations.

Fig. 7 displays the average objective function values against the number of iterations or generations for both of these two methods, therefore displaying the performance trend. From the figure, one finds that the GNA converges faster. This outcome is expected as the GNA determines the descending direction directly by computing the second derivative of the goal function and starts from a location nearer to the minimum. Conversely, by means of comparative analysis of the objective function values, the PSO chooses a more estimate. The GNA hence usually needs fewer iteration to converge to a minimum. If the starting place is too far off the ideal location, though, the GNA could come together to a local minimum.

4.2 The evaluation of GNA Stability

GNA entails matrix inversion during its iterations. The iteration will fail if the matrix $J^{T}{}_{f}J_{f}$ is single.The determining factor of the matrices can be derived as indicated in (13), displayed at the bottom of the page.

In (13), the initial two terms are consistently positive.The third term is 0 when the current location estimation and the estimated positions of its neighbouring nodes are collinear.Consequently, JT f J_f is often non-singular; nevertheless, it may approach singularity due to finite word length when the algorithm is executed on a microcontroller. Consequently, it may not converge to a distinct place at times.This occurs when the pedometer error factor is sufficiently great to render the first two terms of (13) almost zero, and the neighbouring nodes are coincidentally collinear.

Table VI indicates that when the pedometer error factor exceeds 0.25 and likewise surpasses the range error factor, the method may become unstable. In this testing, the predicted positions of neighbouring nodes and the initial guessed position are set to be collinear.

In summation, it is seen that both GNA and PSO possess their own advantages.When the precision of the pedometer is elevated, local optimisation is adequate, and these two methods yield fairly similar localisation results. In this instance, GNA is favoured as it necessitates reduced processing and execution time.Conversely, the PSO is resilient as it consistently provides distinct position estimations. The GNA entails a matrix inversion during its iteration.Consequently, the optimal result will not be attainable.

Table VI: Possibility of In percent Stability

p \ r	0.05	0.1	0.15	0.2	0.25	0.3
0.05	0	0	0	0	0	0
0.1	0	0	0	0	0	0
0.15	0	0	0	0	0	0
0.2	0	0	0	0	0	0
0.25	0.2	0.1	0	0	0	0

when the matrix gets singular or approaches singularity due to the precision of the microcontroller.While alternative methods can be employed to circumvent the singular matrix in the GNA, this may considerably impact the convergence rate.Furthermore, in cases of significant pedometer error, GNA may converge to local minima, resulting in greater localisation error compared to PSO.For systems with significant pedometer error factors, PSO will be employed due of its resilience.
Alternatively, GNA would be a superior option due to its simplicity and minimal computational expenses.

5.0 Exterior the experiment

The experimental measurement has been conducted around a lake and a park located on the university campus.The network comprises 31 sensor nodes. Each sensor node is equipped with an XBeeZNet 2.5 OEM RF module, which is capable of measuring the RSSI.Prior to deployment, calibration was conducted to ascertain the parameters utilised in the path-loss equation by measuring the RSSI in relation to a reference distance.From the measurement, the range of error factor is 0.21.

$$\begin{aligned}\left|\mathbf{J_f^T J_f}\right| = &\underbrace{\frac{1}{\sigma_x^2}\frac{1}{\sigma_y^2}}_{\text{first term}} \underbrace{+\frac{1}{\sigma_x^2}\left(\sum_{j\in\mathbf{J}_i}\frac{(y_i-\hat{y}_j)^2}{\left(\hat{d}_{ij}\cdot r\cdot\delta(\Phi_i,\hat{\Phi}_j)\right)^2}\right)+\frac{1}{\sigma_y^2}\left(\sum_{j\in\mathbf{J}_i}\frac{(x_i-\hat{x}_j)^2}{\left(\hat{d}_{ij}\cdot r\cdot\delta(\Phi_i,\hat{\Phi}_j)\right)^2}\right)}_{\text{second term}} \\ &\underbrace{+\left(\sum_{j\in\mathbf{J}_i}\frac{(x_i-\hat{x}_j)^2}{\left(\hat{d}_{ij}\cdot r\cdot\delta(\Phi_i,\hat{\Phi}_j)\right)^2}\right)\left(\sum_{j\in\mathbf{J}_i}\frac{(y_i-\hat{y}_j)^2}{\left(\hat{d}_{ij}\cdot r\cdot\delta(\Phi_i,\hat{\Phi}_j)\right)^2}\right)-\left(\sum_{j\in\mathbf{J}_i}\frac{(x_i-\hat{x}_j)(y_i-\hat{y}_j)}{\left(\hat{d}_{ij}\cdot r\cdot\delta(\Phi_i,\hat{\Phi}_j)\right)^2}\right)^2}_{\text{third term}}\end{aligned} \quad (13)$$

A pedometer comprises of a three-axis accelerometer-based stride counter and an electronic compass.Redeployment measurements indicate that the pedometer error factor is p = 0.22. The experimental results are presented in Figures 1 and 8. In the figures, the actual positions of the sensor nodes, the estimated positions derived from the pedometer, and the single and bidirectional localisation approaches are indicated on the satellite image map. Figure 9 illustrates the errors encountered during the localisation process at each node along the deployment path. The figure indicates that the average errors for single and bidirectional modes are 16.43 m and 9.51 m, respectively. The mean error associated with the exclusive use of the pedometer, excluding RSSI, is 19.5745 meters. Consequently, the error has been minimised from a unidirectional to a bidirectional approach. The superior performance of the bidirectional approach is anticipated due to its access to a greater number of RSSI values for processing compared to the unidirectional approach.

Figure:8 shows outcomes from experiments using the bidirection approach (x-pedometer, real position, GNA).

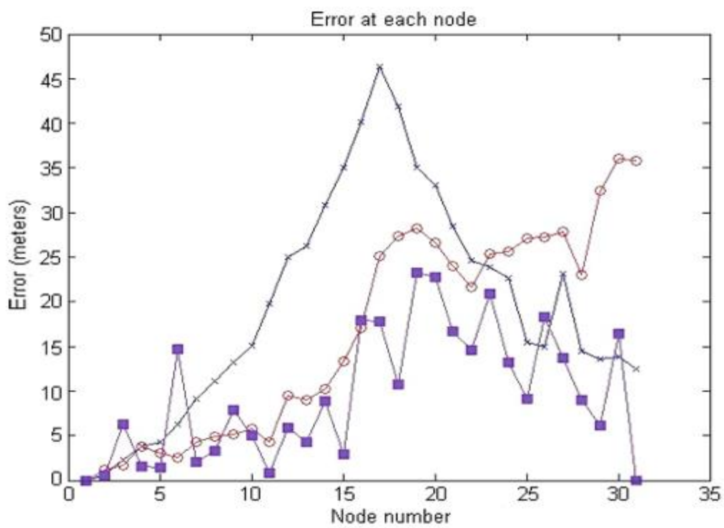

Figure:9 shows the experimental inaccuracy at each node (x-pedometer, single-direction, and bidirection).

TABLE VII PSO VERSUSGNA

	PSO under experiment	PSO under evaluation	GNA under experiment	GNA under evaluation
Single-direction localization error	16.42m	14.52m	16.43m	14.51m
Bi-direction localization error	9.55m	9.81m	9.51m	9.81m
Execution time for single-direction	0.91s	0.92s	0.51s	0.48s
execution time for bi-direction	1.53s	1.51s	0.82s	0.72s

Figures 8 and 9 present the results obtained using only the GNA. The results obtained from the application of PSO closely align with those of GNA and are omitted for brevity. Table VII presents the performance outcomes of the two methods. The table indicates that the localisation results of PSO and GNA exhibit minimal differences.

This aligns with previous findings, indicating that both methods can identify the same optimal value using the same measurements. Furthermore, the GNA is noted to require approximately half the execution time compared to the PSO.

6.0 CONCLUSION

This study employs a microcontroller to perform two optimisation methodologies: the GNA and PSO techniques, aimed at enhancing sensor node localisation in a WSN. The efficacy of the offered methodologies has been assessed and corroborated through experimental results. The results indicate that both exhibit comparable performance with enhanced precision. Furthermore, the GNA necessitates less computational and execution time compared to the PSO, especially when the pedometer's precision is elevated. Nonetheless, significant errors in the pedometer may cause the GNAm to converge to a local minimum. In such instances, the PSO is favoured for its robustness, consistently providing distinct position estimations. An alternate method could involve a Memetic Algorithm that integrates PSO and GNA. In this instance, the algorithm utilises a PSO framework and employs a GNAas as a local the investigator.

References

[1] F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: A survey,” Comput. Netw., vol. 38, no. 4, pp. 393–422, Mar. 2002.

[2] K. S. Low, W. N. N. Win, and M. J. Er, “Wireless sensor networks for industrial environments,” in Proc. Int. Conf. Comput. Intell. Model., ControlAutom., 2005, pp. 271–276.

[3] C. A. Hudson, N. S. Lobo, and R. Krishnan, “Sensorless control of sin gle switch-based switched reluctance motor drive using neural network,” IEEE Trans. Ind. Electron., vol. 55, no. 1, pp. 321–329, Jan. 2008.

[4] E. Mininno, F. Cupertino, and D. Naso, “Real-valued compact genetic al gorithms for embedded microcontroller optimization,” IEEE Trans. Evol. Comput., vol. 12, no. 2, pp. 203–219, Apr. 2008.

[5] C.-S. Wang, “Flicker-insensitive light dimmer for incandescent lamps,” IEEE Trans. Ind. Electron., vol. 55, no. 2, pp. 767–772, Feb. 2008.

[6] S. Saponara, L. Fanucci, and P. Terreni, “Architectural-level power op timization of microcontroller cores in embedded systems,” IEEE Trans. Ind. Electron., vol. 54, no. 1, pp. 680–683, Feb. 2007.

[7] F. Botteron and H. Pinheiro, “A three-phase UPS that complies with the standard IEC 62040-3,” IEEE Trans. Ind. Electron., vol. 54, no. 4, pp. 2120–2136, Aug. 2007.

[8] Z. Jiang and R. A. Dougal, “A compact digitally controlled fuel cell/battery hybrid power source,” IEEE Trans. Ind. Electron., vol. 53, no. 4, pp. 1094–1104, Jun. 2006.

[9] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, “A fast adaptive memetic algorithm for off-line and on-line control design of PMSM drives,” IEEE Trans. Syst., Man, Cybern. B, Cybern.—Special Issue Memetic Algorithms, vol. 37, no. 1, pp. 28–41, Feb. 2007.

[10] N. Bulusu, J. Heidemann, and D. Estrin, “GPS-less low-cost outdoor localization for very small devices,” IEEE Pers. Commun., vol. 7, no. 5, pp. 28–34, Oct. 2000.

[11] D. Niculescu and B. Nath, “DV based positioning in ad hoc networks,” J. Telecommun. Syst., vol. 22, no. 1–4, pp. 267–280, Jan. 2003

[12] D. Niculescu and B. Nath, “Ad hoc positioning system (APS) using AOA,” in Proc. IEEE Comput. Commun. Soc., 2003, pp. 1734–1743.

[13] C. Savarese, J. Rabaey, and K. Langendoen, “Robust positioning algorithm for distributed ad-hoc wireless sensor networks,” in Proc. USENIX Tech. Annu. Conf., Monterey, CA, 2002, pp. 317–328.

[14] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz, “Localization from mere connectivity,” in Proc. 4th ACM Int. Symp. Mobile Ad Hoc Netw. Comput., Annapolis, MD, 2003, pp. 201–212.

[15] M. L. Sichitiu and V. Ramadurai, “Localization of wireless sensor net works with a mobile beacon,” in Proc. Int. Conf. Mobile Ad-Hoc Sensor Syst., 2004, pp. 174–183.

[16] H. Guo, K. S. Low, and M. J. Er, “Localization in a sparse wireless sensor network using pedometer and communication ranging measurements,” in Proc. IECON, 2007, pp. 2627–2632.

[17] S. Y. Xue and S. X. Yang, “Power system frequency estimation using supervised Gauss–Newton algorithm,” in Proc. ISIC, 2007, pp. 3761–3766.

[18] J. De Zaeytijd, A. Franchois, C. Eyraud, and J.-M. Geffrin, “Full-wave three-dimensional microwave imaging with a regularized Gauss–Newton method—Theory and experiment,” IEEE Trans. Antennas Propag., vol. 55, no. 11, pp. 3279–3292, Nov. 2007.

[19] K. S. Low, H. A. Nguyen, and H. Guo, “A particle swarm optimization approach for the localization of a wireless sensor network,” in Proc. IEEE Int. Symp. Ind. Electron., Jul. 2008, pp. 1820–1825.

[20] A. Chatterjee, K. Pulasinghe, K. Watanabe, and K. Izumi, “A particle swarm-optimized fuzzy-neural network for voice-controlled robot sys tems,” IEEE Trans. Ind. Electron., vol. 52, no. 6, pp. 1478–1489, Dec. 2005.

[21] L. Dos Santos Coelho and B. M. Herrera, “Fuzzy identification based on a chaotic particle swarm optimization approach applied to a nonlinear yo-yo motion system,” IEEETrans. Ind.Electron., vol. 54, no. 6, pp. 3234–3245, Dec. 2007.

[22] Y. Song, Z. Chen, and Z. Yuan, “New chaotic PSO-based neural net work predictive control for nonlinear process,” IEEE Trans. Neural Netw., vol. 18, no. 2, pp. 595–601, Mar. 2007.

[23] T. Huang and A. S. Mohan, “A microparticle swarm optimizer for the reconstruction of microwave images,” IEEE Trans. Antennas Propag., vol. 55, no. 3, pp. 568–576, Mar. 2007.

[24] N.JinandY.Rahmat-Samii,“Advancesinparticle swarm optimization for antenna designs: Real-number, binary, single-objective and multiobjective implementations,” IEEE Trans. Antennas Propag., vol. 55, no. 3, pp. 556 567, Mar. 2007.

[25] L. Lizzi, F. Viani, R. Azaro, and A. Massa, “Optimization of a spline shaped UWB antenna by PSO,” IEEE Antennas Wireless Propag. Lett., vol. 6, pp. 182–185, Mar. 2007.

[26] F.-J. Lin, P.-K. Huang, and W.-D. Chou, “Recurrent-fuzzy-neural network-controlled linear induction motor servo drive using genetic al gorithms,” IEEE Trans. Ind. Electron., vol. 54, no. 3, pp. 1449–1461, Jun. 2007.

[27] K.-S. Low and T.-S. Wong, “A multiobjective genetic algorithm for optimizing the performance of hard disk drive motion control system,” IEEE Trans. Ind. Electron., vol. 54, no. 3, pp. 1716–1725, Jun. 2007.