Task and Motion Planning of multi-agent in ship hull inspired by Ant's Foraging Behavior

Gourav Khanduri

Arizona State University

MAE-598 (Multi-Robot system)

Submitted To: Professor Spring Berman

Abstract

This report investigates the behavior of multi-robot systems inspired by ant colonies, focusing on autonomous area coverage for ship hull inspection. Apart from the hull inspection solution can be used for agriculture inspection where robots have to search patches of farms. I always wanted to implement multi-robot in the ship hulls as I have seen the difficulty faced during the operation on ship hulls. The primary objective is to develop a scalable, human-independent framework to address rust detection on extensive hull surfaces of ships. The system draws inspiration from the foraging behavior of ants, utilizing Lévy flight random walk patterns to enhance search efficiency by balancing local exploration with global coverage. The A* algorithm which is a variant of Dijkstra's algorithm was implemented to optimize pathfinding and ensure effective task allocation among robots so that robots can optimally reach the affected area. Simulations were conducted using the Python Pygame library to validate the system's performance under various conditions. I tried different approaches like Brownian motion, levy walk to check which one could better fit to improve the search efficiency. Levy flight is giving promising results. So I moved forward with levy flights. This project provides a good solution for autonomous industrial inspection, mitigating the risks associated with manual operations and offering potential applications in exploration, and industrial automation. By integrating bio-inspired algorithms and scalable robotic control, this work marks a step forward in deploying efficient, collaborative multi-robot systems.

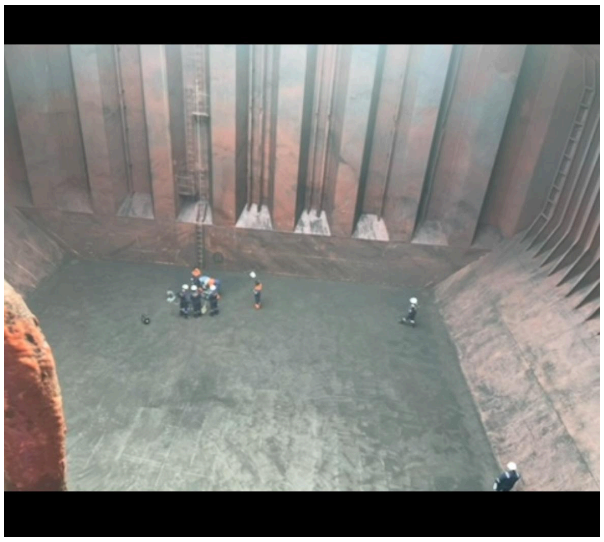


Figure 1: Ship hull -Target inspection area.

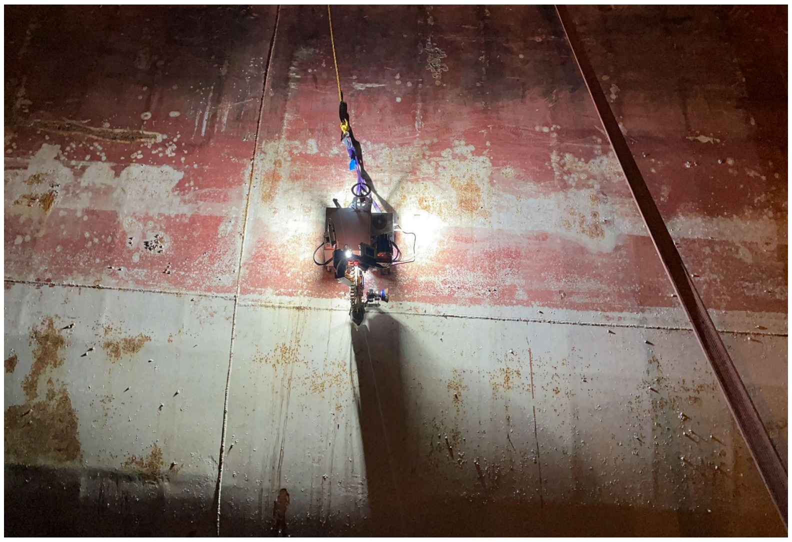


Figure 2: Realtime inspection of ship hull by one robot

Problem Introduction:

Ship hull inspection, traditionally performed by a single robot, is a time-intensive process due to the vast surface area and complexity of the structures. While single-robot systems can navigate these geometries effectively, they struggle with efficiency, often requiring extensive time to complete operations. Recognizing this limitation, I envision multi-robot systems as the future of ship hull inspection, offering significant advantages in scalability, speed, and reliability. By dividing the hull into sections and assigning robots to operate simultaneously, these systems can drastically reduce inspection times and improve task specialization, such as grinding, ultrasonic testing, or data mapping. This approach not only enhances efficiency but also addresses the growing need for quick turnarounds in maritime maintenance.

Mathematical Model:

The behavior of the multi-robot system in this project is inspired by the foraging dynamics of ant colonies where ants can switch between tasks as the requirements arise, leveraging Lévy flight for efficient exploration and rust detection on ship hulls. The real-world scenario involves deploying multiple autonomous robots to inspect a ship's hull, covering a large bounded surface area. The robots aim to detect rust patches collaboratively, after completing the rust search worker's robots have the grinding and surfacing tool. Robots can move to the affected area for cleaning by leveraging the A* algorithm to find the optimal path, minimizing redundant exploration and human intervention. This approach ensures safety, scalability, and efficiency. There may be a downside to this approach to handling the swarm of robots in a complex environment but similar robots could be used to increase the efficiency of the ship industries because people there risk their lives and it takes too much time if we are doing the same operation using human intervention or the single robot.

Environmental Properties and Experimental Constraints:

To better define the experimental conditions for this project, assumptions related to the skid-steering differential drive robot are outlined as follows:

- The number of robots is limited to 4 due to cost, exploration uses energy-efficient Levy flights in less complex areas, and communication constraints reflect realistic challenges in metallic ship environments.
- Realistic sensing and communication ranges, position tracking, speed, and marker-based navigation ensure efficient task coordination and autonomy in challenging, GPS-deprived environments.
- 3D LiDAR, extended A* path planning, task switching, and proximity sensors ensure accurate mapping, dynamic adaptability, and safe navigation in complex environments.
- It has been assumed that the robots can do holonomic and nonholonomic motions to better explore the environment.
- A bounded $x \times y,$ environment with static obstacles enables systematic quadrant-based exploration, ensuring coverage and task optimization. Decentralized marker-based collaboration is practical and scalable for constrained communication scenarios.
- Each agent can cover specific cells based on battery life and communication range.

Exploration Model:

The model presented in this work can be divided into two parts. First, random exploration in each quadrant of the defined area and then finding the optimal path of the affected area. The experiment used mean square displacement to measure the deviation of the robot's position from its reference point over time. It is being calculated for every robot's position estimation.

$$\textbf{MSD} = \left(\left|x(t) - x_0\right|^2\right) = \frac{1}{N}\sum_{i=1}^{N}\left|x_i(t) - x_i(0)\right|^2$$

Here, N represents the number of particles involved in the task, $x_i(0)$ is the initial position of the i-th particle, and $x_i(t)$is its position at time t. MSD is a widely used metric for quantifying the spatial extent of a random walk and assessing the portion of the system explored by the robots. When robot movement is treated as a continuous function of time t, MSD can be expressed as follows[1]:

The calculation of the robot's Mean Square Displacement (MSD) in the previous method assumes movements are restricted to specific directions on a grid. However, in real-world applications, a robot can move freely in any direction on a circular plane. To better reflect this unrestricted movement, the robot's direction for each step is modeled using a circular distribution. One widely used circular distribution for this purpose is the von Mises distribution, which represents the possible directions of movement more accurately in two-dimensional space.

$$f(\theta) = \frac{1}{2\pi I_0(K)} e^{K\cos(\theta - \theta_0)}$$

Where I_0 denotes the modified Bessel function of the first kind and order 0, defined by

$I_m(K) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \cos(m\theta) e^{K\cos\theta} d\theta$; θ_0 is the mean of the step direction within $[-\pi, \pi]$ and K is the inverse of the dispersion of the step direction, which can measure the concentration of the step direction θ to the mean θ_0[1].

For uncorrelated random walks:

$$E(X_t^2) = 2Dt$$

MSD in 2D with von Mises distribution:

$$E(R_n^2) \sim nE(l^2) + 2n(E(l))^2 \frac{c_t}{1-c_t}$$

Where $c_t = E(\cos\phi)$ *and* $s_t = E(\sin\phi)$

Swarm Control Law for Area Coverage implying Levy flight:

$$\frac{\partial a(r,t)}{\partial t} = \nabla.(D_a \nabla a(r,t) - Xa(r,t)\nabla b(r,t))$$

Where D_a is the diffusion coefficient for robot concentration $a(r,t)$, t *is the time*, $b(r,t)$ is the inspection marker concentration, and X represents the repulsiveness(positive value X) or attractiveness(negative value X) of the marker.

$$\frac{\partial b(r,t)}{\partial t} = \nabla.D_b \nabla b(r,t) + g(r,t) - h(r,t)$$

Where D_b is the marker diffusion coefficient,g(r,t) is the marker production rate, and h(r,t) is the marker degradation rate. For this model, the equation can be simplified by considering a constant marker deposition rate and production rate, to reach the affected area once the rust is marked it should not degrade.

$$\frac{\partial b(r,t)}{\partial t} = D_b \nabla^2 b(r,t) + qa(r,t) - \gamma b(r,t)$$

Here, the rate of change of marker concentration is a function of diffusion deposition.

Levy Flights:

Algorithm used: Pseudocode for swarm control law for area coverage.
Robots Emulating Ant foraging with Levy flights[2].

Data: LevyFlag = F; T_o; $T_c = 0$; N; $N_r = 1$; P_r; q; η; b_r; δ; v; p_l

```
while $T_c \le T_o$ do
    while $N_r \le N$ do
        if the Robot is outside the area under consideration
            then
              move within the area;
        else
          If $b_r > \eta$ then
              LevyFlag = T;
              Choose Levy destination $P_l$ with
                  equation() and ();
              Move towards $P_l$;
          else
              LevyFlag = F;
              Choose next destination with
              equation() and move towards it;
          end
        end
         if LevyFlag=T AND dis($P_l$, $P_r$) $<\delta$ then
              Reached the Levy destination $P_l$;
              LevyFlag = F;
          end
        end
        Pheromone Deposition q at $P_r$;
        $N_r = N_r + 1$;
    end
    $T_c = T_c + dt$
end
```

Consensus Controller:

After completing the coverage of all quadrants, the robots will gradually converge back to their home position, reaching consensus asymptotically.

$$\dot{x}_i = \sum_{j \in N_i} (x_j - x_i)$$

Velocity Consensus:

Robots synchronize their velocities while heading towards the home position.

Control Law:

$$\frac{dv}{dt} = k_v \sum_{j \in N_i} b_{ij}(v_j - v_i) - k_h(v_i - v_{desired})$$

Where:

$$v_i = velocity\ of\ robot\ i,$$

$$v_{desired} = Desired\ velocity\ toward\ the\ home\ position.$$

Control Barrier Function:

By implying **barrier certificates**, I can implement minimally invasive safety constraints. Ensuring robots maintain a safe distance from each other. Keeps robots constrained to defined quadrants.

$$\dot{x}_i = u_i$$
$$\dot{x}_i = f(x_i) + g(x_i)u_i$$
$$u^* = min||v - v_d||$$
$$h_{boundary}(x, u) \geq a(h_{boundary}(x))$$
$$h_{ij}(x) = ||x_i - x_j|| - \delta_{ij}$$
$$h^*_{boundary}(x) = e_i^T x - \epsilon - b^{left}$$

After completing a random search of all the quadrants, the group of robots identifies the optimal path to the target destination. Leveraging shared memory, the robots store data collaboratively, similar to how ants utilize pheromone trails for efficient navigation. Instead of pheromones, this system employs a computational approach using the A* algorithm, a variant of Dijkstra's algorithm. The A* algorithm determines the shortest path to a specific destination of the rust deposit once the optimal route is found. This enables the worker robots to follow the precomputed path and complete their tasks in the shortest possible time, ensuring efficiency and coordination across the team.

Pseudocode : A* algorithm[5]

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in came from.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path
function A_Star(start, goal, h)
    opened := {start}
  came from := an empty map

    // For node n, gScore[n] is the currently known cost of the cheapest path from start to n.
   score := map with a default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how cheap a path could be from start to finish if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        // This operation can occur in O(Log(N)) time if openSet is a min-heap or a priority queue
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to the neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := tentative_gScore + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    return failure
```

Theoretical Analysis:

The controller I examine using the following criterion:

1. Is the system stable while exploring
2. After completing the search they converge back to the home position.
3. Is the system is energy-efficient

Let me begin with how the levy walk is a suitable controller for my problem. A levy walk is a type of random walk where the step lengths follow a power-law distribution. This allows agents to perform long jumps occasionally, avoiding excessive redundancy.

Step length l follows a power-law distribution:

$$p(l) = l^{-\lambda}, 1 < \lambda \leq 3$$

Where λ controls the proportion of long to short steps.

Turning angle distribution:

The turning angle θ is described by a uniform or von Mises distribution:

$$f(\theta) = \frac{1}{2\pi I_0(K)} e^{K\cos(\theta - \theta_0)}$$

$I_0(K)$: *Modified Bessel function of the first kind.*
K: *Dispersion Parameter(control angular Concentration).*
θ_0 : Preferred turning direction.
The equilibria occur when the net force is zero:

$$\nabla U_{attr} = k_{att}.\, d \text{ drives agents towards the target}$$

$$\nabla U_{rep} = -\frac{k_{rep}}{d^2}\ prevents\ agents\ from\ colliding.$$

$$k_{att.d} - \frac{k_{rep}}{d^2} = 0$$

Equilibria distance d* computed to be:

$$d^* = (\frac{k_{rep}}{k_{att}})^{\frac{1}{3}}$$

Stable Equilibria: Points were $\nabla^2 U_{total}(x) > 0$ at rust location.

Unstable Equilibria: points where $\nabla^2 U_{total}(x) < 0$ an obstacle peaks.

Convergence

Let's check the stability of the system.
Key system dynamics:

$$\dot{x}_i = u_i$$

Robot's positions evolve based on the control input.
The stability of the system is analyzed using the Lyapunov function,

$$V(x_i) = \frac{1}{2}||x_i||^2$$

Which represents the distance of each robot from the home position. Designing a control law

$$u_i =- kx_i$$

The Lyapunov function's derivative is guaranteed to be negative, ensuring that the robots asymptotically converge to the home position at the grid's center.

$$\dot{V}(x_i) = -k||x_i||^2$$

Condition:

$$\dot{V}(x_i) < 0 \text{ for } x_i \neq 0,$$

$$\dot{V}(x_i) = 0 \text{ only at the origin}(x_i = 0).$$

Thus, $V(x_i)$ is a Lyapunov function, and the system is globally asymptotically stable at the origin.

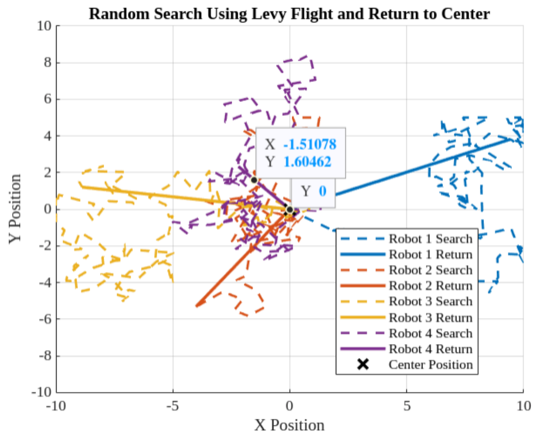


Figure 3 : Converging to Centre

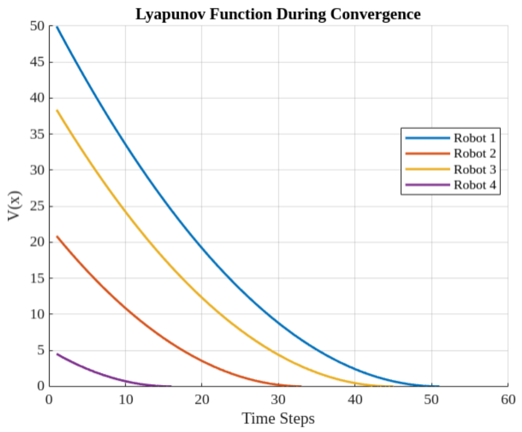


Figure 4: Velocity Consensus

Energy efficiency:

While exploring the environment, energy consumption and battery capacity are critical considerations. Robots with lower battery levels should prioritize nearby cleaning areas, ensuring that if their battery crosses a certain threshold, they can return to the home position for charging. To optimize the process, the master system can allocate tasks based on each robot's battery capacity, ensuring smooth and continuous operation.

Formulation:

$$\text{Minimise } z1 = total\ time,\ z2 = total\ energy\ consumption$$

$$\sum_i energy \leq energy_{max}$$

$$\sum_i Coverage_i = areasize(x, y)$$

Results:

The simulation of the robots was done in the Python library PyGame. There are two different simulations for the projects. The first is for the rust search by the 4 robots in their defined quadrants and the second is a simulation for removing the rust from the surface with battery-depleting features and recharging themselves after crossing a certain threshold. The robots will start from the center of the area and after completing their search they will converge back to the center. Control barrier functions are implemented inside the program so that robots do not collide with each other. Also, the boundary condition is set (-1) so that the agents should not cross the boundary and search in their defined area.

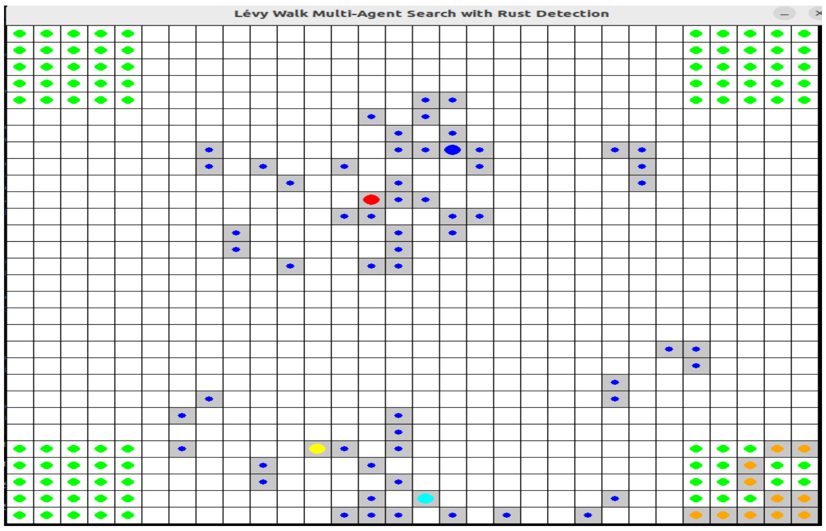
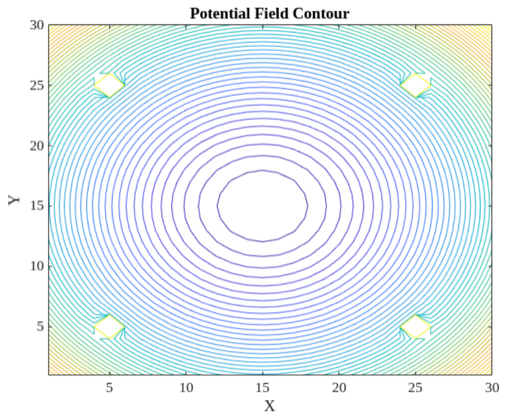


Figure 5 : Levy walk (Rust searching)

In Figure (5) it is shown that the yellow, light blue, blue, and red dots are the agents searching in their quadrants the blue dots are the marker trail and a bunch of green dots at the corner of the grid are the rusted area. Agents in their quadrants whenever they find the rust mark them as the orange color to distinguish them from the markers. As they complete the search they will converge to the center with the help of velocity consensus control.



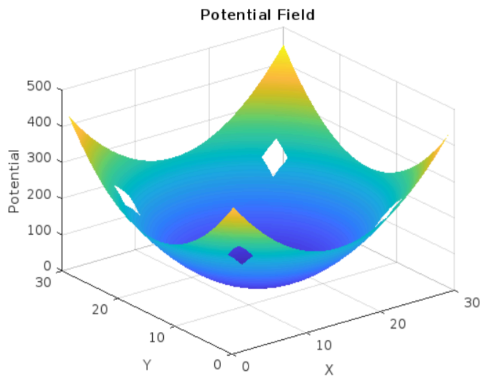


Fig 6 : Rust Allocation **Fig 7: Repulsive zone peaks**

In Figure (6) and Figure (7), I have plotted the potential field of the model. The valley's low potential indicates the paths leading to the target and the peaks (high potential) highlight repulsive zones around obstacles. It shows the forces affecting agent behavior in the grid environment. The attractive potential drives agents toward the rust locations(targets). This is important because my primary goal is to find and determine the rust efficiently. Following the repulsive potential ensures agents maintain safe distances from the obstacles and each other.

Peaks around obstacles in the graph correspond to high repulsion zones, where agents actively avoid collision. This synergy of both potentials aligns with my goal of efficient, collision-free search in a structured grid.

Attractive potential:

$$U_{attr} = \frac{1}{2}k_{att} * d^2$$

Where k_{att} $is\ the\ attractive\ constant,\ and\ d\ is\ the\ distance\ to\ the\ target.$

Repulsive potential:

$$U_{rep} = \frac{1}{2}k_{rep}(\frac{1}{d} - \frac{1}{d_{safe}})^2,\ if\ d \leq d_{safe}\ and\ 0,\ if\ d > d_{safe}$$

Where k_rep is the repulsive constant, d is the distance to obstacles, and d_{safe} $is\ the\ safety\ radius.$

$$U_{total} = U_{attr} + U_{rep}$$

The levy walk itself introduces stochasticity, ensuring that random exploration can be guided toward targets and away from obstacles.

Discussion:

In this project, I have tried to present my idea to show the feasibility of the problem statement. There are many more things that need keen attention, before diving into the real-time implementation. The big picture of the project is to train a data set of different types of rust present on the ship hulls so that during the search operation through the perception, robots can easily identify the rust according to the severity of the rusting. In the future, I am planning to carry out this project to reality. So that the idea could be implemented in the shipping industry and also in the startup I used to work in, which in a result will increase the operational efficiency in the maritime industry.

References

[1] Pang, B., Song, Y., Zhang, C. et al. Effect of random walk methods on searching efficiency in swarm robots for area exploration. Appl Intell 51, 5189–5199 (2021) Link

[2] Deshpande, A., Kumar, M. and Ramakrishnan, S., 2017, October. Robot swarm for efficient area coverage inspired by ant foraging: the case of adaptive switching between Brownian motion and lévy flight. In Dynamic Systems and Control Conference (Vol. 58288, p. V002T14A009). American Society of Mechanical Engineers. Link

[3] Kegeleirs, M., Garzón Ramos, D., Birattari, M. (2019). Random Walk Exploration for Swarm Mapping. In: Althoefer, K., Konstantinova, J., Zhang, K. (eds) Towards Autonomous Robotic Systems. TAROS 2019. Lecture Notes in Computer Science(), vol 11650. Springer, Link

[4] Berger, J. and Lo, N., 2015. An innovative multi-agent search-and-rescue path planning approach. Computers & Operations Research, 53, pp.24-31. Link

[5] A* algorithm, "Wikipedia, Wikimedia foundation, 19 November 2024" link