Social Network Analysis for Target Recognition in Swarm Robotics

Michael C. Koval, Adina E. S. Rubinoff, Mahsa Maghami and Prof. Michael Georgiopoulos

EE & CS, Rutgers University, New Brunswick, NJ; CS & Linguistics, University of Rochester, Rochester, NY; EECS, University of Central Florida, Orlando, FL

ABSTRACT

Over the past few years, swarm-based systems have emerged as an attractive and promising approach for implementing distributed autonomous systems. This is useful in different applications, such as automatic target recognition (ATR). In ATR, the most pressing concern is the accuracy of the system in detecting and recognizing the targets. To fulfill this requirement, previous approaches require more than one agent to classify a target, which decreases the number of recognition errors. Increasing the number of agents required to recognize a task will make the system more accurate; however, it delays recognition, which is not acceptable in most applications. In addition, agents are never completely identical in the real world; even two identical agents will have sensors with different error tolerances. Unfortunately, previous approaches fail to distinguish accurate robots from less accurate ones.

In this paper we propose a novel algorithm, named the NetBots algorithm, which improves accuracy in ATR by introducing a social network to the swarm system. This social network keeps track of the robots’ target detection histories and is used to estimate the robots’ accuracy. Any time two robots deciding a task agree with the majority decision, a link is formed between them. The PageRank algorithm is then used to rank the robots based on the number and ranks of the robots linking to them. Robots that are more accurate will agree with the majority more often, so they will have more links and therefore a higher rank. When deciding a task, higher ranked robots have more influence, so inaccurate robots’ votes will not negatively impact the majority decision. This approach increases the overall accuracy of the system by 1.76 percent over 30 tasks, compared to other approaches with no social network.

Keywords: Swarm robotics, Target recognition, Social network

1. INTRODUCTION AND LITERATURE REVIEW

Swarm robotics is the study of how large numbers simple agents can be designed such that a desired collective behavior emerges from their interactions. Swarm robotics takes its inspiration from the interactions of swarming animals such as ants, birds, and fish. Swarm robotics is especially appropriate for solving tasks that are distributed in nature, such as aggregation, foraging, and target recognition. Swarm systems possess a number of desirable characteristics that make them useful in a variety of applications.

The first desirable characteristic of a swarm system is robustness, which means that even if individual agents are destroyed or damaged, the swarm as a whole can continue and finish the task. This is a direct result of each robot having its own controller that dictates its actions. Thus, robots will continue to operate even if others are damaged or destroyed. When robots share a central controller, destruction of that controller prevents all the robots from working. The robustness of a swarm system makes it very useful for dangerous tasks. For example,
even if one robot clearing a minefield is blown up, the rest of the robots continue to operate and finish clearing the field.

Another beneficial characteristic of a swarm system is its flexibility, meaning that if the task changes while they are working on it, they are able to adapt and attempt to complete the modified task. This is again due to the fact that each robot has its own personal controller that can assess the current situation and adjust its actions as necessary.

Lastly, swarm systems are scalable, which means that they can operate under a wide range of robot group sizes and task sizes without substantial decrease in performance performance. This is useful for tasks that scale up or down over time. If a task starts out small but grows in size, more and more robots can be summoned to contribute to the task’s completion. Conversely, if a task is reduced in size, some robots can cease working, adjusting to the demands of the task without unnecessarily wasting resources.

Many authors have proposed taxonomies for classifying various areas of swarm robotics. It is helpful to specify where in these taxonomies our effort falls, to put it in context. This also makes it easier to find work related to ours and work to compare ours to.

Iocchi et al. present a four-level hierarchical taxonomy of swarm robotics literature that focuses on the behavior and interactions of multi-agent systems. The first level of the hierarchy distinguishes whether or not the agents work together to accomplish the given task. The next level further subdivides the literature based on if robots are aware that other agents exist. The third level determines whether the robots consider other robots’ behavior when deciding their actions. The last level distinguishes if there is central or individual control. In this taxonomy, our project falls into the category Cooperative/Aware/Strongly Coordinated/Distributed. Our robots cooperate to classify tasks; they are aware of each other; they have distinct communication protocols, and they each have an individual controller.

Another taxonomy, put forth by Bayindir and Sahin (shown in Figure 1), focuses more on general areas of study in swarm robotics. This taxonomy defines four different branches within swarm robotics, each of which have subbranches to further classify the areas of study. Some studies focus solely on one branch, while others fit into multiple or even all the branches.
The first branch of Bayindir’s taxonomy is *modeling*, which focuses on studies that try to develop models to predict robot behavior without running real-life experiments. This branch is split into four categories: *sensor-based*, *microscopic*, *macroscopic*, and *cellular automata*. Sensor-based models simulate data for each physical sensor on the agents, and then use this information to calculate the exact actions each agent would take for every tick of simulation time. Microscopic modeling is a bit more general; this category includes any type of model that specifies the state of each individual agent at every tick of simulation time. Macroscopic modeling, on the other hand, only specifies the overall state of the entire system at every tick. Lastly, cellular automata are ways of representing a system as a 2D array of cells. Each cell affects its neighbors according to rules that govern the system. Within this branch of Bayindir’s taxonomy, our project falls into the sensor-based modeling category. In order to test the NetBots algorithm we used the Stage simulator, which is an example of sensor-based modeling. Stage creates robot objects and hands them all the sensor data they would receive in reality. These robots then act on the data just as real robots would.

The second branch in this taxonomy is *behavior*, which defines the way the robots’ behavior changes over time. There are three categories in this branch: *non-adaptive*, *learning*, and *evolution*. Non-adaptive means that the algorithm governing a robot’s behavior does not change over the course of the experiment. Learning, on the other hand, means that the robot’s algorithm does change with time. As the robots perform tasks and encounter other robots, they learn and thus their algorithm changes. Evolution means that the system uses genetic algorithms to adapt the robots’ algorithms to their environment. Currently our effort falls into the non-adaptive category of Bayindir’s taxonomy branch. For now the robots’ algorithms and behavior remain constant throughout the experiment. If, later on, learning is added, they will simply exhibit a behavior that belongs to the learning category.

The third branch of Bayindir’s taxonomy is *communication*, which specifies how the robots interact with one another. This branch also has three categories: *interaction via sensing*, *interaction via communication*, and *interaction via environment*. Interaction via sensing means robots use their sensors to detect other robots’ actions and conclude information from it. Interaction via communication means the robots explicitly communicate by
sending each other signals directly. Interaction via environment means the robots change the environment in some way to alert other robots to some information. For example, in many swarm robotics experiments, the robots leave pheromones on the ground which communicate information to other robots who see them later. Our effort uses interaction via communication. The robots use direct signaling for a multitude of purposes. Hosts use it to send out a call for help and to tag a new robot as host when leaving a task. Helpers use it when informing the host of their task and what their decision is about the task. All of these situations are using interaction via communication.

The final branch of the Bayindir’s taxonomy consists of the swarm robotics problems that are studied. This branch has many categories, some of the most well-known being aggregation, foraging, self-assembly, chain formation, and hole avoidance. Our project focuses on target recognition, which is a foraging problem. The robots are not collecting objects to a “home” location; however, they are still spreading out and searching for targets, which is a foraging problem in and of itself.

In the target recognition problem, a host of objects is scattered through an environment. The robots’ job is then split into two parts. The robots must locate all the targets in the environment, and they must classify these targets into different categories based on certain characteristics. Often, multiple robots’ opinions are required to definitively classify a task. For example, Dasgupta’s work uses unmanned aerial vehicles to (UAVs) identify marks on the ground as targets. To decide the status of a target, a host of UAVs swarm and make individual decisions, which are then tallied into a final group decision.

Solving the target recognition problem is useful in many contexts. Target recognition systems could be used in bad weather to help guide pilots who are unable to see the ground. ATR systems are helpful to soldiers that need to see and identify objects in the dark. Unmanned ground vehicles depend on target recognition to detect their surroundings and avoid obstacles. Currently, unmanned vehicles are mostly used for military applications, such as scouting, but there are many non-martial applications where a remotely piloted vehicles are useful. Clearly, target recognition is extremely useful and even vital in many situations where it can save lives.

Much prior work has been devoted to target recognition. The UGV/DEMO II Program sponsored by the US Department of Defense worked to develop a system for piloting a single unmanned ground vehicle. In order to see the road they are following, the UGVs use target recognition to identify the areas on the ground as either part of the road or not. There is no group work involved, though; the robot decides based only on its sensory inputs. It can also only keep track of where it has been, should it need to save that information for later use. This system only used a single agent. A similar single-agent system was used in a study by Buluswar and Draper, which focused on using color to aid in target recognition for road following. They approximated piecewise linear non-parametric functions that describe in RGB-space the image being seen by the robot. This helped them in classifying the information and identifying the target (the road the robot was to follow).

However, target recognition is better performed by a multi-agent system. A multi-robot system can cover more area than a single agent by spreading agents throughout the environment. A single agent, no matter how powerful or perceptive, can only be in one location at once. Even if the individual agents in a multi-agent system are smaller and have limited sensing abilities, they are able to spread out and be in multiple locations at once, thus covering much more of the environment. A multi-robot system is also preferable in a dangerous situation such as a war, because if one agent is damaged or destroyed, the rest of the system can continue operating to complete the task. If only one agent were used, the completion of the entire task would depend on that one robot staying safe, which is much less likely. Out of all multi-agent systems, a swarm system is preferred, since cooperation speeds up the process of locating tasks. Dasgupta used UAVs for this purpose. His algorithm was simple and just had the robots helping each other out with tasks indiscriminately. In a situation where the robots are homogeneous, with identical physical and algorithmic capabilities, such approaches work fine. When heterogeneous robots are considered, however, a new approach is called for.

Heterogeneous robots can have different sensing abilities or different algorithms. Heterogeneous algorithms are commonly found in systems where different robots have different jobs. A system with learning will also develop heterogeneity, as the robots learn at different paces. Heterogeneous sensors are unavoidable in the real world; even if the experimenter attempts to give the robots homogeneous sensors, all sensors have an error due to tolerances in the sensors’ construction. So even sensors that are supposedly built the same will have some differences between them. This difference in sensing ability is the type of heterogeneity this effort focuses on.
Heterogeneous sensors negatively impact the overall accuracy of a swarm system, unless the system is designed to take the heterogeneity into account. Suppose three robots are deciding a task, two with bad sensors and one with good sensors. If a plain majority voting system is used, the two inaccurate robots will be able to outvote the accurate one and the task will be misclassified. Such a situation calls for an algorithm that takes the robots’ heterogeneity into account when deciding how to interact with other robots.

To address the heterogeneity of agents, we introduce social networking into our task recognition algorithm. When any two robots work together on a task and are both part of the majority decision, a link is formed between them. The PageRank algorithm is then used to calculate a rank for each robot based on its links in the graph. Robots with higher ranks are more influential when deciding tasks. In this way the overall accuracy of the system will improve.

In this paper we will define the NetBots algorithm for target recognition, so called because it uses social networking to improve the system’s target recognition accuracy. The rest of this report is organized as follows. In Section 2 the target recognition problem under consideration is described. Sections 3 and 4 describe in detail the HelpBots and NetBots algorithms, respectively. In Section 5 we introduce the simulation issues of this work. Section 6 shows and analyses the obtained results of three algorithms following with the future work in section 7. Finally, Section 8 concludes the paper.

2. TARGET RECOGNITION

The generic target recognition experiment that we focus on is as follows: We have \( n \) robots placed in an arena and their goal is to complete \( T \) tasks. The robots wander around the arena looking for tasks and avoiding obstacles. Once they arrive at the task they attempt to complete it. A certain number of robots is required to complete a task, controlled by the parameter \( Q \). Robots at a task must wait until \( Q \) robots have arrived. An amount of time after which the robots a task, controlled by the waiting time parameter, may be specified. Once \( Q \) robots arrive at the task, the task is completed and the robots continue wandering until the experiment ends.

Ijspeert et al.\(^9\) performed a similar experiment to solve the stick-pulling problem with swarm robotics. In his work, in addition to simply wandering, the robots would respond to calls for help from other robots working on tasks. When a robot was working on a task it would send out a call for help. Any wandering robot that received that call would turn and head towards the task in order to help out. This additional communication between robots increased the speed with which the robots completed all the tasks.

The NetBots algorithm includes this call for help, and adds social networking on top of that. Each robot stores a graph which represents all the links between robots that it is aware of. Every time a robot agrees with another robot about a task, a link is formed between them. Every time robots collaborate on a task, the host of the task combines their matrices and gives all collaborating robots the updated information. This way each robot updates its information about the overall state of the social network every time it works with other robots on a task.

These links are important because we use the PageRank algorithm\(^10\) on them to estimate the robots’s relative accuracy. Each robot has a rank assigned to it using the PageRank algorithm.\(^10\) This ensures that the more correct decisions a robot makes, the higher its rank will be. In other words, the more accurate a robot is, the higher its rank will be. When a task (type of food) is decided, each robot’s vote is weighted based on its rank. This way robots with higher ranks (who are therefore more accurate) will have more influence over the decision, and it will be harder for inaccurate robots to negatively impact the group’s decision on the task.

3. HELPBOOTS

In order to assess the effectiveness of the NetBots algorithm, a benchmark algorithm is needed for comparison. For this purpose we used an algorithm based on work by Ijspeert et al.\(^9\) This algorithm will be referred to as the HelpBots algorithm, since its most important aspect is the robots’ call for help when waiting at a task. A flowchart detailing this algorithm is presented in Figure 2. Pseudocode for this algorithm is shown in Algorithm 1.
Figure 2. The HelpBots algorithm as a FSM

### Algorithm 1: Overall Algorithm

Just as the algorithm presented by Ijspeert et al.’s, HelpBots begins in the Search state. In this default state, each robot retains its initial heading and travels directly forward at its maximum speed until one of its
sensors detects an object. When an object enters the robot’s sensor range, the robot classifies the object as an obstacle (i.e. wall or robot) or task. This processed sensor data is passed to the navigation system (Section 3.1) to adjust the robot’s speed and heading in response to the robot’s changing local environment.

**Algorithm 2: Deciding Role Algorithm**

```plaintext
1    this .state = host foreach robot in shortrange do
2       if robot .task == this .task and robot .state == host then
3          this .state = helper
4      end
5    end
```

**Algorithm 3: Host Algorithm**

Once the robot has successfully reached a task, it determines if it must adopt the *host* or *helper* role using Algorithm 2. If there are no robots waiting at a task, the first robot to arrive automatically becomes that task’s host. Hosts are responsible for broadcasting help calls and tallying the helpers’ opinions into a majority decision, as shown in Algorithm 3. Conversely, robots that arrive at an already occupied task promptly send the host a message containing their classification of the task and wait for the appropriate number of robots (Q) to arrive, as shown in Algorithm 5.

If the number of tasks exceeds the number of robots, it is possible that each robot will permanently remain at a separate task. More generally, n robots may be unable to identify T tasks if $T > \frac{n-1}{Q-1}$ as no task would exceed $Q-1$ robots. Implementing a maximum waiting time for which a robot remains at a task eliminates this problem and adds another parameter for optimizing the performance of the system without substantially increasing its complexity. This is inspired by and is a direct analog of the gripping time parameter discussed in the canonical still-pulling experiment by Ijspeert et al.\(^9\)

Additionally, Ijspeert et al.\(^9\) implemented a call for help signal that is received by all robots in a 60-degree cone from the front of the broadcasting robot. Unfortunately, the larger tasks used in HelpBots simulations fully obscure a frontal broadcast. Therefore, robots implementing the HelpBots algorithm broadcast their call for help to all robots within a fixed radius, regardless of the relative orientation of the host and recipient. The robot being called computes the relative angle between their current heading and the broadcasting host using the information encoded in the help message. This desired heading is then passed into the navigation system, which guides the called robot towards the help signal while continuing to avoid collisions with tasks and obstacles.

### 3.1 Navigation

Algorithm 4 shows pseudocode for the robots’ navigation algorithm. Basically, the robots wander around avoiding obstacles and each other, looking for tasks to complete. When they find a task, they become either helpers of
hosts and deal with the task accordingly. Once the task is complete, they return to the wander state. While wandering, a robot may receive a call for help from another robot that is hosting a task within its calling range. If a robot receives a call, it factors it into the potential fields calculation as if it were a very strong positive charge. This way the call for help will attract it. If it receives multiple calls for help they all get factored into the equation and they all impact the robot’s path. Finally, while following a call for help, the robot ignores any tasks within its range unless it runs directly into them. This prevents the robot from getting distracted by tasks that are not currently being worked on.

\begin{algorithm}
\begin{verbatim}
while true do
  if HelpCallRecieved() then
    CalculateHeading(posforce);
  end
  else if objectDetected() then
    if task then
      CalculateHeading(posforce);
    end
    else
      CalculateHeading(negforce);
    end
  end
  else if Collided() then
    BackUp();
  end
end
\end{verbatim}
\caption{Navigation Algorithm}
\end{algorithm}

Robots are only have knowledge of obstacles that are within the limited range of their infrared sensors (Section 5.2.1). When no obstacles, robots, or tasks are detected by the robot’s infrared sensors, it defaults to moving forward at its maximum speed. However, if there is at least one object in range, it uses its potential field navigation to avoid obstacles and tend towards tasks.

Robots treat all objects within their sensor ranges as an electrical point charges. Robots and obstacles are assigned positive charges, \( q_r, q_o > 0 \), forcing robots to be repelled from potential collisions. Conversely, tasks are assigned a negative charge \( q_t \), such that robots are attracted to tasks.
Figure 3. A robot using the potential fields method to calculate a new heading based off the forces acting on it from the nearby target and obstacle.

Let $F_{\text{net}}$ be the net electrostatic force on a robot with charge $q_r$. Define $L$ as the set of all objects detected by this robot’s infrared sensors, where $q_i \in L$ represents the charge of the object and $r_i \in L$ represents the location of the object relative to the robot. The net electrostatic force acting on this robot is defined as:

$$F_{\text{net}} = \sum_{i \in L} \frac{q_r q_i}{r_i^3} r_i$$

See Figure 3 for an illustration of the method used to calculate this net force vector. Each robot recomputes this $F_{\text{net}}$ and modifies its heading to turn towards this vector every time it has updated sensor data. While sensor updates occur in discrete ticks, the frequency of these ticks is sufficiently small that a robot can effectively respond to its environment. To further ensure that that robots do not collide, turning rate ($\omega$) is directly proportional and forward velocity ($\|v\|$) is inversely proportional to $\|F_{\text{net}}\|$:

$$\|v\| = v_{\text{max}} \tan\left(\frac{1}{\|F\|}\right)$$

$$\omega = \omega_{\text{max}} \tan(\|F\|)$$

Despite this method’s simplicity, it has a number of shortcomings. Potential field navigation causes robots to oscillate near long obstacles and in tunnels and prevents robots from passing through narrow openings. Even in non-problematic regions, the electric potential field around a robot may feature local minima. These points occur where the forces acting on a robot cancel out, causing $F_{\text{net}} = 0$. Any movement away from a local minimum will result in a force directing the robot back to the minima, causing it to get “stuck” and fail to move.
Figure 4. The potential fields of an environment plotted as a function. The goal is a global minimum and obstacles are local maxima. The robot always heads towards forces lower than its current position.

One potential solution the local minima problem, advocated by Khosla and Volpe 12, is to use *superquadric functions* to model the fields. These authors noted in their paper that circular potential fields have no local minima; however, they cover much more area than fields shaped like the obstacle they’re surrounding. In order to have the best of both worlds, the authors proposed a compromise in the form of superquadric functions. These functions are the shape of the obstacle when close to it, but become more circular the farther they get from the obstacle (See Figure 5). This way they avoid local minima while occupying a smaller area.

Figure 5. Objects surrounded by superquadric fields.

Other authors have explored ways to improve upon the potential fields method. Baxter 13 suggested sharing potential field information between robots. In his algorithm, a robot alerts other robots in its local area to obstacles it encounters. This compensates for the fact that all robots sensors have some error, which may cause them to either detect an object that is not present (false positive) or miss an object that is there (false negative). Sharing information with local neighbors allows a robot to make its decision about the presence or absence of objects based on multiple opinions; this way the decision is more likely to be accurate, since the chance of a majority of the robots in the area making the same error is smaller.
Another idea proposed by Chang et al 14 is to add gyroscopic forces to the potential fields method to solve certain problems caused by local minima. For example, three robots crossing a circle may become stuck in the center due to a local minimum in their potential field function. If gyroscopic forces are introduced, however, the robots are able to spin away from each other and continue across the circle unimpeded (see Figure 7. This way gyroscopic forces compensate for some issues unsolvable by the potential fields method on its own.

Obstacle avoidance is just one part of a robot’s navigation system - its behavior in absence of any obstacles must also be specified. In addition, it is desired that a robot moves towards any tasks it detects, so an algorithm for this must also be defined.

Fortunately, the potential fields method works just as well for driving towards a task as it does for driving away from an obstacle. By simply assigning the tasks and the robots opposite charges, Coulomb’s law will cause the robot to be drawn towards the task. 11

A variety of algorithms may be specified for the robots’ behavior in the absence of any objects in its vicinity, but the most common approach is to have the robot drive straight forward. We have chosen this method because it is simple and allows the robot to explore its environment in a steady, predictable manner. Usually the robots’ speed is constant; however, if they get close to another object they slow down in order to give them more time to react. Upon detecting an object, the robots use the object’s reflectance to determine if it is a task or an obstacle.
4. NETBOTS

An overview of the NetBots algorithm is presented as a finite state machine in Figure 8. Most of the Search and Task Completion states remain unchanged from HelpBots, described in detail in section 3. NetBots primarily distinguishes itself from the relatively simple HelpBots algorithm in the Ranking state, as depicted in the simplified finite state machine in Figure 8.

Each robot using NetBots algorithm stores an undirected graph that represents all of the interactions that the robot is aware of. Vertices in this graph represent robots and edges represent successful interactions between the joined robots. These local graphs may differ from robot to robot as each robot has a different knowledge of past interactions. Generating and updating these graphs occurs only when a task is completed and is naturally the responsibility of the host.
When a helper first arrives at a task, the helper transmits its opinion of the task and its graph to the host. After the host receives enough opinions to calculate the group decision about the task, it merges all of the robots’ graphs into a single graph that represents the combined knowledge of the group. To simplify this merge operation, NetBots considers the adjacency matrix representations of each graph.

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{SendDecisionToHost();}
\While{\textit{!task.completed}}
\State \textbf{Wait();}
\EndWhile
\end{algorithmic}
\caption{Helper Algorithm}
\end{algorithm}

Consider robots $r_1, r_2, \ldots, r_Q$ immediately after completing a task. Let $M^{(k)}$ be the adjacency matrix representation of the local graph of robot $k$ such that $M_{ij}^{(k)}$ is the number of links between robots $i$ and $j$. The merged adjacency matrix, $M'$ is defined as

$$M'_{ij} = \max(M_{ij}^{(r_1)}, M_{ij}^{(r_2)}, \ldots, M_{ij}^{(r_Q)})$$

Unlike a simple matrix addition, performing an element-wise maximum eliminates the possibility of double-counting earlier edges, a crippling problem which would eventually lead to earlier interactions being more influential than recent interactions.

Once the host has an up-to-date graph, it executes the PageRank algorithm (described in detail in Section 4.1) to compute a collective PageRank vector $(\gamma')$ for the merged graph. Each robot’s rank is added to a tally associated with its decision, effectively weighting the influence of each robot’s decision by its PageRank. Let $E$
be the set of robots that decided that the task is edible and $P$ be the set of robots that decided that the task is poisonous, such that
\[
\begin{align*}
  w_p &= \sum_{i \in P} \gamma_i \quad (4.2) \\
  w_e &= \sum_{i \in E} \gamma_i \quad (4.3)
\end{align*}
\]
where the task is identified as edible if $w_e \geq w_p$ and poisonous if $w_p < w_e$.

Once this decision is complete, the host forms links between all robots that agree with the group decision. This ensures that robots with the most accurate sensors will rapidly gain links and will become most influential in the calculation of subsequent PageRank vectors.

### 4.1 Social Networks

The main innovation of the NetBots algorithm is the social network used to rank robots based on the accuracy of their sensors. When $Q$ robots have arrived at a task, the host collects each robot’s opinion on the task. Next it uses the links in the social network to calculate ranks for each robot. It weights each robot’s decision based on its rank and then calculates a weighted majority. Lastly, links form between any robots that were a part of the majority decision. For example, say that three robots arrive at a task (item of food) and must classify it as either poisonous or edible. Robots 1 and 2 think it is edible, but robot 3 thinks it is poisonous. Also say robot 1 has rank 4, robot 2 has rank 5, and robot 3 has rank 7. When the opinions are weighted, a robot’s rank determines how many “votes” it gets. In the example, there are nine votes altogether for edible (from robots 1 and 2 together), and seven votes for poisonous (from robot 3). So the task is classified as edible and a link is formed between robots 1 and 2.

Each robot stores at all times a graph of all links in the social network that it is aware of. This is called its *partial network graph*. It uses this graph to compute its *local PageRank vector*, a vector of all the ranks of all robots it has encountered. Similarly there exists a global social network graph that contains all the links between all robots. The robots are not aware of this graph; it is just used by experimenters for comparison. This graph is used to compute a global PageRank vector that lists all the robots’ ranks.

The original PageRank algorithm, first discussed in Brin and Page’s influential paper 15, uses links to websites to rank search results. PageRank treats the internet as an unweighted, directed graph in which pages are vertices and links between pages are edges. PageRank assigns each vertex a scalar rank that can be viewed as a measure of that vertex’s “importance” relative to other vertices in the graph. More concretely, a vertex’s PageRank is defined as the probability of a web-surfer reaching a vertex after performing a random walk of the graph from a randomized starting vertex.

Considering a graph $G$ with vertices $V$ and edges $E$, define $E_i$ as the set of all edges $E_i = \{ j : (j, i) \in E \}$. With these symbols defined, the PageRank of vertex $i$ is recursively defined as:
\[
PR(i) = \frac{1 - d}{|V|} + d \sum_{j \in E_i} \frac{PR(j)}{|E_j|} \quad (4.4)
\]
where $d \in [0.0, 1.0]$ is the *damping factor* parameter that controls the probability of the web-surfer continuing to click links. Similar to prior work 16, NetBots uses a damping factor of $d = 0.85$.

This mathematical definition of PageRank holds for a single page, but is highly inefficient for computing the global PageRank vector of a graph. Mathematically, the PageRank vector ($\gamma$) of graph $G$ is defined as:
\[
\gamma = \begin{pmatrix}
  PR(1) \\
  PR(2) \\
  \vdots \\
  PR(|V|)
\end{pmatrix} \quad (4.5)
\]
where $\gamma_i$ is the PageRank of the $i$-th vertex of $G$. To facilitate the computation of $\gamma$, the probability of the web-surfacer following an edge from vertex $i$ to vertex $j$ is encoded in a *stochastic transition matrix*. The stochastic transition matrix, $P$, of a graph is traditionally defined as a *uniform distribution*, normalized such that columns sum to one:

$$P_{ij} = \frac{1}{\text{deg}(i)}$$  \hspace{1cm} (4.6)

```
input : Connectivity graph G, initial PageRank vector $x^{(0)}$, and uniform vector $v$
output: A matrix $P$ s.t. $P_{ji} = \frac{1}{\text{deg}(j)}$

1 while $\delta \geq \epsilon$ do
2     $x^{(k+1)} = cP^T x^{(k)}$;
3     $w = \|x^{(k)}\|_1 - \|x^{(k+1)}\|_1$;
4     $x^{(k+1)} = x^{(k+1)} + wv$;
5     $\delta = \|x^{(k+1)} - x^{(k)}\|$;
6     return $x^{(k+1)}$;
7 end

Algorithm 6: the PageRank algorithm
```

As seen in Algorithm 6, PageRank is iteratively executed until the $L_1$ Residual between the PageRank vectors of two consecutive iterations is less than the threshold error value, $\epsilon$. Lower values of epsilon ensure closer convergence of the computed PageRank vector to the actual PageRank vector of the graph at the cost of increased computation time. As in prior work 16, NetBots uses an error threshold of $\epsilon = 0.0001$, which achieves a good balance between convergence rate and error.

Figure 10 shows a graph of a social network between eight robots. The thicker the lines between robots are, the more links exist between them. Also, the lighter the color of the robot node, the higher the rank of that robot.

Figure 10 shows a graph of a social network between eight robots. The thicker the lines between robots are, the more links exist between them. As mentioned above, links form between robots when they work together on a task and agree on the majority decision. So robots 6 and 7 have agreed many times, while robots 2 and 3 have only agreed one or two times, and robots 1 and 2 have never agreed. The colors of the robots’ nodes indicate their ranks, computed using the PageRank algorithm. Robot 2 has a low rank because it doesn’t have
very many links. Robot 0, on the other hand, has a high rank both because it has many links and because those links are to other well-ranked robots such as robot 7.

In order to ensure NetBots’ social network is working as desired, two criteria must be met. First, each robot’s local PageRank vector must converge to the global PageRank vector. In addition, the order of the robots when sorted by the ranks in the global PageRank vector must converge to the actual order of robots when arranged by sensor error. The first requirement is quantitatively measured by the $L_1$ Residual, or the Manhattan distance, between a local PageRank vector and the global PageRank vector, as well as by Kendall’s $\tau$-Distance metric. Kendall’s $\tau$-Distance is also used to measure the difference between NetBots’ ranks and the actual sensor-error ranks.

### 4.1.1 $L_1$ Residual Metric

The $L_1$ Residual is a metric for measuring the error between two vectors that represent the same quantity. Specifically, the $L_1$ Residual of vectors $v_1$ and $v_2$ of the same length is defined as $\|v_2 - v_1\|_1$, where $\|\ldots\|_1$ is a Manhattan norm. The Manhattan norm of a vector is the sum of the absolute values of its elements, such that the Manhattan norm of $x$ of length $n$ is defined as:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$  \hspace{1cm} (4.7)

This metric is used in PageRank to test for the iterative convergence of the PageRank vector. Let $\gamma_k$ and $\gamma_{k+1}$ be the PageRank vectors of iterations $k$ and $k + 1$. As described in Section 4.1 the algorithm stops iterating when the $L_1$ residual between $\gamma_k$ and $\gamma_{k+1}$ (stored as $\delta$) falls below a fixed threshold ($\epsilon$):

$$\|\gamma_{k+1} - \gamma_k\|_1 < \epsilon$$  \hspace{1cm} (4.8)

Furthermore, the convergence of local PageRank vectors to the global PageRank vectors in the NetBots algorithm is quantitatively measured by the average $L_1$ residual between the local vectors ($\gamma_1, \gamma_2, \ldots, \gamma_n$) and the global PageRank vector ($\gamma_g$). Plots of $L_1$ residual versus an independent variable show the relationship between that value and the system’s effectiveness at sharing information between robots.

### 4.1.2 Kendall’s Distance Metric

Kendall’s $\tau$-Distance\(^{17}\) is a measure of dissimilarity between two ordered vectors. Consider two ordered vectors $p_1$ and $p_2$ of length $n$. Define the function $K_{\{i,j\}}(p_1, p_2)$ such that it returns zero if elements $i$ and $j$ are in the same pairwise order and one if the elements are out order:

$$K_{\{i,j\}}(p_1, p_2) = \begin{cases} 
1 & : i \text{ and } j \text{ in different orders} \\
0 & : i \text{ and } j \text{ in same order} 
\end{cases} \text{ in } p_1 \text{ and } p_2$$  \hspace{1cm} (4.9)

Using this mathematical definition of the pairwise order of two vector elements, the number of pairwise differences in order is intuitively defined as

$$K(p_1, p_2) = \sum_{\{i,j\} \in P_D} K_{\{i,j\}}(p_1, p_2)$$  \hspace{1cm} (4.10)

where $P_D = \{(i, j) : i \in p_1 \text{ and } j \in p_2\}$ is the set of all pair-wise combinations of the elements of $p_1$ and $p_2$. Kendall’s $\tau$-Distance divides this count of pairwise differences in order by the total number of pairwise combinations, yielding a metric normalized to fall in the range $[0, 1]$:

$$\text{KDist}(p_1, p_2) = \frac{K(p_1, p_2)}{n(n-1)/2}$$  \hspace{1cm} (4.11)

Let $R_i$ be the robots $0, \ldots, n$ by the local PageRank vector of robot $i$ ($\gamma_i$) and $R_g$ be the ranking of the robots by the global PageRank vector ($\gamma_g$). The Kendall’s $\tau$-Distance between $R_i$ and $R_g$, $\text{KDist}(R_i, R_g)$, measures the difference between local and global rankings, similar to how the $L_1$ Residual between $\gamma_i$ and $\gamma_g$ measures the difference between local and global PageRank vectors (see Section 4.1.1).

The flexibility of this metric also allows for comparison between the ranks computed by the NetBots ($R_g$) algorithm and the actual ranks of the robots ($R_{\text{act}}$, based upon their true sensor error). Low values of $\text{KDist}(R_g, R_{\text{act}})$ demonstrate a close correlation between NetBots rankings and actual robot accuracy.
5. SIMULATION

5.1 Khepera III Robot

In order to demonstrate the effectiveness and efficiency of our proposed approach to the target recognition problem, we used the Stage simulator. The robots in our swarm system are Khepera III robots simulated in Stage. We used the potential fields approach to navigation, using the robots’ IR sensors to detect objects in the environment. The tasks (food) are colored differently, to distinguish the edible ones from the poisonous ones. The robots used their built-in cameras to recognize the color (type) of food. In order to improve the robots’ collaboration, and to help them choose good partners to collaborate with, we added in social networking and ranked each robot according to their accuracy in correctly recognizing the types of food.

Despite the dichotomy between the physical production of physical robots hardware and their controller software, the two construction endeavors are intrinsically and inextricably linked. Controller algorithm developments must be tested on some sort of platform: whether it be a physically embodied agent or a virtual robot that exists only in simulation software. To quantitatively compare our algorithm against other algorithms designed to solve the same problem, we first selected a robot with physical properties and sensing abilities that suit our problem at hand.

After carefully weighing the information presented in this section, we concluded that the Khepera III robot is ideally suited for evaluating our algorithm. Designed by the K-TEAM Corporation, the Khepera III combines the familiarity of the original Khepera robot with the extensive sensing and communication capabilities that are common on modern robots. More importantly, the Khepera class of robots supports the addition of custom extension turrets, allowing us to customize the robot to better suit our task.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td>130 mm</td>
</tr>
<tr>
<td>Height</td>
<td>70 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>690 g</td>
</tr>
<tr>
<td>IR Transducers</td>
<td>9</td>
</tr>
<tr>
<td>Communication</td>
<td>Bluetooth, WiFi add-on</td>
</tr>
</tbody>
</table>

Table 1. Technical specifications of K-TEAM’s Khepera III robot.

Prior swarm robotics research efforts have used a wide range of physical agents, ranging from Hu’s custom three-wheeled robots to the ubiquitous, commercialized Khepera class of robots. Custom robots are acutely tuned for a specific task and particular experiment. For example, Hu’s robots feature extensive directional infrared communication to support his research on local inter-robot communication. Contrastingly, Dasgupta’s simulated robots, developed to evaluate his solution to the target recognition problem, are unmanned ariel vehicles (UAVs) with high-fidelity color cameras and a long-range communication system.

Selecting a widely-used agent gives us the ability to focus on our controller algorithm without considering minute implementation details such as each sensor’s pose (position and orientation) and configuration parameters. Most importantly, selecting such an agent permits us to easily compare the results of the NetBots algorithm to canonical benchmark solutions, such as the HelpBots algorithm.

Our ideal robot remains as close to canonical swarm robotic agents as possible without sacrificing the sensing and communication abilities that the NetBots algorithm demands. The Khepera class of robot ideally fulfills both requirements: It is used in a plethora of well-regarded experiments ( and can easily be customized to suite our task with the addition of a custom add-on turret.

To effectively respond to their environment through any navigation technique, our robots must detect the relative pose of nearby objects. Furthermore, each agent must differentiate between desirable objects (tasks) and obstacles (environmental obstacles and other robots) such that the agents can avoid obstacles and tend towards tasks.
Using the original Khepera’s infrared rangefinder array, Ijspeert et al. implemented a basic obstacle avoidance method in his original, influential paper on the stick-pulling problem. Specifically, the Khepera III’s array of infrared rangefinders consists of nine infrared transducers ringing the base of the robot. Four of these transducers are evenly distributed at 36° intervals across the front of the robot. Another three are distributed across the back at intervals of 45°, with the remaining two sensors lying directly on the lateral bisection of the robot. Figure 11 depicts this configuration, offering full 360° coverage with slightly a superior resolution on the front half of the robot.

Figure 11. The sensors on a Khepera III robot. Each beam represents the sensor range. The direction of each sensor is indicated by a degree amount printed on the sensor beam. The robot’s heading is indicated by a black line pointing towards the front of the robot.

While this simple solution works well for detecting if an object is present, additional information is required to differentiate tasks from obstacles and robots. Inspired by the use of infrared reflective tape to increase the visibility of objects to infrared sensors in several swarm robotics experiments (24, 25), we ringed tasks with this special reflective tape. Instead of merely reporting a distance, the sensors monitor both distance and intensity: Obstacles are relatively non-reflective and report a low intensity. Conversely, tasks are highly reflective to infrared light and will report a very high intensity of reflected light.

After the agent reaches a task, it must use sensor data to identify the task as either edible or poisonous. Despite the abundance of sensors on the Khepera, including unused ultrasonic rangefinders, none of the sensors are well-suited for high-fidelity target identification. We have decided to use the KoreUSBCam extension turret designed by K-TEAM Corporation, the creator of the Khepera III robot. Unlike the alternative KheperaIII Wireless Camera, the KoreUSBCam is low-resolution (from 160 × 120 to 640 × 480 pixels) that transmits data directly to the Khepera III’s controller without passing through an intermediate host computer.

The NetBots task identification algorithm, similar to Dasgupta’s UAV algorithm4 requires local communication between robots to transmit the result of an identification vote. Unlike Dasgupta’s voting algorithm, NetBots’ helpers must also transmit a graph to the task’s host, and the host must transmit back a vector of computed ranks and a graph. This type of information is targeted at a single robot, the host, and must be reliably delivered.

This demand for targeted and reliable communication is best achieved with a message passing algorithm. Khepera III robots support two of the most popular physical incarnations of wireless message passing: Bluetooth and 802.11 WiFi (via an optional add-on WiFi card). While both technologies have been extensively used in prior works (26, 27), we have decided to use WiFi based upon its superior range and K-TEAM’s online recommendation:

The Khepera III is also able to include Wireless Ethernet network communication. This configuration is a perfect solution for applications requiring communication between two robots.
5.2 Simulation in Stage

Due to the prohibitively high costs—considered either as a nominal cost or the opportunity cost of the time spent running the real-world trials—of testing the NetBots algorithm on actual robots, we decided to implement the NetBots algorithm on simulated agents. Specifically, we decided to implement the NetBots algorithm in the two-dimensional Stage simulator created by the Player/Stage open source project (see Figure 12) due to its extensibility, flexibility, and prevalence in swarm robotics literature. Unfortunately, mapping physical robot properties into an extremely limited selection of Stage models is not easy and requires a number of non-intuitive logical leaps. Section 5.2.1 is included to justify these decisions and clarify how we simulated a Khepera III robot in Stage.

![Figure 12. A screenshot of the stage simulator](image)

5.2.1 Infrared Proximity Sensors

As discussed earlier, a set of nine infrared proximity transducers ring the Khepera III. Each sensor’s samples contain measures of both distance (in meters) and reflectivity. After receiving one sample from each transducer, the robot’s controller aggregates the sensor information into a crude navigation model of the nearby region.

At first glance, Stage has a **ranger** model that is intended to represent an array of range finding transducers, such as the Khepera’s circular array of infrared sensors. Unfortunately, there is no method to access the strength of the signal returned by the **ranger** model, eliminating it from further consideration.

Thankfully, Stage provides a **laser** model, originally intended to model a scanning laser rangefinder. Whereas the **ranger** model lacks crucial reflectivity information, the **laser** model provides too much information. Stage’s **laser** model ray-casts a number of discrete lines (as determined by the laser’s **sample size** property) that are evenly distributed throughout the laser’s field of view. Each ray-casted line, or **sample**, returns the distance it traveled and a reflectivity constant.

To realistically model the Khepera’s inferior infrared proximity detectors, which only have a single, distributed, sample, the simulation code must aggregate a large number of laser samples into aggregate distance ($D_a$) and reflectivity ($R_a$) transducer readings. Let $D_i$ be the distance and $R_i$ be the reflectivity of the $i$-th sample of a specific transducer. The aggregate distance and reflectivity of a simulated infrared transducer is defined as

$$D_a = \min(D_1, D_2, \ldots, D_9) \quad (5.1)$$

$$R_a = \max(R_1, R_2, \ldots, R_9). \quad (5.2)$$
Each simulated infrared transducer has a maximum range of 25 cm\textsuperscript{19} with approximately a 30° field of view. We selected this field of view to greatly simplify our sensor-processing code by ensuring that the fields of view of two adjacent sensors never overlap (see Figure 11). Each transducer is simulated as a laser with a sample size of 10, an experimentally determined value that yields a good balance between accuracy and efficiency.

5.2.2 Color Digital Camera

5.2.3 Wifi Communication

6. RESULTS

6.1 Experimental Setup

To test the NetBots algorithm, tests were run on both HelpBots and NetBots. In these experiments, the parameter $Q$ (robots required to complete a task) was set equal to three. This is the smallest number of helpers we could require to complete a task and still be guaranteed to never have a tie. The tasks were split between two different categories. We refer to these categories as “edible” and “poisonous”, as if the robots were foraging for food that could be harmful. This is a canonical target recognition problem.

The robot models we used in Stage are 1.35m in diameter, a factor of ten larger than real-life Khepera III robots. Robots as small as the Khepera III are hard to simulate in Stage, and scaling everything by a factor of ten does not affect the results as long as the scaling is consistent. The tasks were represented as colored circles 1.7m in diameter, and the arena in which the experiments took place in is 32m by 20m. Within the arena, the tasks and robots were evenly distributed using a uniform distribution (see Figure 13).

In order to simulate heterogeneous sensors, we assigned each robot a randomly generated error value between 0 and 50. This value determines the robot’s percent chance of making a mistake when deciding a task (kind of food). For example, say a robot has an error value of 23. Whenever it is looking at a task, it chooses a random
number between 0 and 100. If this number is less than 23, it changes its decision about the food to an erroneous one. This way it has a 23% chance of making an error. In any given run of our experiment, an error vector contains a list of the error values of all the robots in that run.

6.2 HelpBots

The next set of experiments were run on the HelpBots algorithm, in order to have a benchmark for comparing our results. In these experiments we used the parameters shown in table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Robots</td>
<td>6, 8, 10, 12, 14</td>
</tr>
<tr>
<td>Waiting Time</td>
<td>20, 30, 40, 50, 75, 100, 125, 150, 175, 200</td>
</tr>
<tr>
<td>Number of Tasks</td>
<td>9</td>
</tr>
<tr>
<td>Task Distribution in the Environment</td>
<td>Uniform</td>
</tr>
</tbody>
</table>

Each run of the HelpBots algorithm had a setup defined by the waiting time, number of robots, and vector of error values. We tested ten different values for waiting time (20, 30, 40, 50, 75, 100, 125, 150, 175, 200), five different numbers of robots (6, 8, 10, 12, 14), and five different randomly generated error vectors. This makes for 250 different setups, each of which we ran four times, for a total of 1000 runs. Each run had nine tasks, either five edible and four poisonous or vice versa. We ended each run after ten minutes, or when all the tasks had been completed, whichever came first.

Figures 14, 15, 16, and 17 display results from our tests with the HelpBots algorithm. Each graph has data for five different numbers of robots. In addition, each data point is an average over 20 runs: five different setups with different error vectors, each run four times.

![Figure 14](image_url)

Figure 14. HelpBots results: Accuracy Rate (AR) plotted against waiting time. AR measures both the accuracy of the robots’ decisions and how fast they make them. A low rate means they were slow and inaccurate; a high rate means they were fast and accurate. Six lines are plotted, one for each number of robots (n) tested.

Figure 14 shows accuracy rate plotted against the waiting time parameter, parameterized with respect to the number of robots. A high accuracy rate means that the robots were able to identify the food correctly and quickly, so the higher the accuracy rate the better.
The first thing to note in this graph is that the more robots there are in the experiment, the better the accuracy rate is. This makes sense because if there are more robots, more will be working on tasks at any time, so the tasks will be completed faster. Another thing to note is that very low waiting times have slightly lower accuracy rates. If the robots leave very soon after they arrive at a task, there is no time for other robots to come help them, so it takes longer overall to complete all the tasks. Lastly, increasing the waiting time does not guarantee a better accuracy rate. For example, when n=6, the accuracy rate peaks around a waiting time of 40 seconds, and for n=8 it peaks around a waiting time of 80 seconds. If the robots wait at tasks for a long time, then they can all get stuck waiting at different tasks and nothing will be completed. So a longer waiting time can make the accuracy rate go down.

Figure 15. HelpBots results: Time plotted against correctly detected targets. The higher the line, the longer it took to identify all the tasks, while a low line means they were identified very quickly. Six lines are plotted per graph, one for each number of robots (n) tested.

Figure 15 shows time plotted against the number of correctly identified targets. There is a separate graph for each of four waiting times, 30 through 150. Time is on the vertical axis, which means that the faster the robots complete the tasks, the lower on the graph their line will be. So the lower a line is, the better.

One thing to note here is that the larger number of robots complete all the tasks much faster than the smaller numbers of robots. This is because, as mentioned above, when more robots are made available to work on tasks, the tasks will be completed faster overall.

Another thing to note is that towards the end of the graphs, after most of the tasks have been completed, the time the tasks have taken to be completed goes back down. This is most visible with six and eight robots. For example, in the graph with a waiting time of 50 seconds, it takes six robots eight seconds to complete eight tasks, but only six seconds to complete nine tasks. This is an artifact of averaging - each of these lines is an...
average of twenty runs. Due to the stochasticity of the runs, the nine-task average comes out lower than the eight-take average.

Figure 16. HelpBots results: Average number of correctly identified tasks plotted against waiting time. Each run contained 9 tasks for the robots to identify. For each value of waiting time, data is shown for each of six numbers of robots (n) tested. The higher the bar, the more accurate the robots were in identifying tasks.

Figure 16 shows the number of correctly identified tasks plotted against waiting time. Of note in this graph is that overall, waiting time doesn’t really affect accuracy. However, as waiting time increases, the number of tasks correctly identified by 6 robots decreases dramatically. This is because, for very high waiting times, six robots are simply unable to complete all the tasks, since they spend so much of their time waiting at tasks for other robots to come help them.

Figure 17. HelpBots results: Success rate (SR) plotted against waiting time. Data is shown for the six different numbers of robots (n) tested. SR is calculated as \( \frac{\text{successful runs}}{\text{total runs}} \), where an unsuccessful run is one that has to be stopped before all the tasks are completed. The lower the bar, the fewer times the robots succeeded in completing all the tasks. A SR of 100 means the robots always completed all the tasks.

Figure 17 shows the rate of success plotted against the waiting time. For the higher number of robots, the success rate is 100%. When there are many robots, they will be able to complete all the tasks no matter how long
they have to wait around for helpers, because there will always be robots available to help them. However, the lower numbers of robots are not always successful, because it takes them longer to complete all the tasks. Ten minutes may not be enough time. Also, for the lower numbers of robots that do have below 100% success rates, it is interesting to note that the success rate peaks at a different waiting time for each of them. For example, with eight robots, the best success rate (of 100%) is somewhere between the 50 and 100 second waiting times (probably around 75 seconds), while for 6 robots the peak is at 50 seconds. This shows that lower waiting times are better for smaller amounts of robots. The more robots there are, the longer they can afford to wait for a helper, because there’s a higher chance that a robot will be free and come along to help. This also shows that higher waiting times aren’t always better - after a certain point, increasing the waiting time reduces the success rate.

6.3 NetBots

NetBots was run with the parameters shown in Table 3. As in HelpBots, each run of NetBots had a setup defined by the waiting time, number of robots, and vector of error values. We tested four different values for waiting time (50, 100, 150, 200), four different numbers of robots (6, 8, 10, 12), and three different randomly generated error vectors. This makes for 48 different setups, each of which we ran four times, for a total of 192 runs. Each run had ten tasks, five edible and five poisonous. We ended each run after ten minutes, or when all the tasks had been completed, whichever came first.

Figures 18, 19, and 20 show data collected from the NetBots experiments. Each graph has data for four different numbers of robots. In addition, each data point is an average over 12 runs: three different setups with different error vectors, each run four times.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Robots</td>
<td>6, 8, 10, 12</td>
</tr>
<tr>
<td>Waiting Time</td>
<td>50, 100, 150, 200</td>
</tr>
<tr>
<td>Number of Tasks</td>
<td>30</td>
</tr>
<tr>
<td>Task Distribution in the Environment</td>
<td>Uniform</td>
</tr>
</tbody>
</table>

Table 3. Values used for parameters in the NetBots experiments.

Figure 18. NetBots results: Accuracy rate (AR) plotted against waiting time. AR measures both the accuracy of the robots’ decisions and how fast they make them. A low rate means they were slow and inaccurate; a high rate means they were fast and accurate. Six lines are plotted, one for each number of robots (n) tested.
Figure 18 shows accuracy rate plotted against waiting time for the NetBots algorithm. Here four different numbers of robots were plotted, six through twelve with a step size of two. As with HelpBots, accuracy rate measures the robots' accuracy in classifying the food and also the rate at which they completed all the tasks. As expected, the higher numbers of robots have better accuracy rates. If the number of tasks stays the same, increasing the number of robots makes it easier for them to complete the tasks faster. Another thing to note is that the accuracy rate peaks for twelve robots at a waiting time of 150s. This is also reflected in Figure 19 below.

![Figure 19](image_url)

Figure 19. NetBots results: Average number of correctly identified targets plotted against waiting time. The robots' goal was to complete 30 tasks each run. For each value of waiting time, data is shown for each of six numbers of robots (n) tested. The higher the bar, the more accurate the robots were in identifying tasks.

Figure 19 plots average number of correctly identified targets against waiting time. This is a measure of how accurate the robots are at recognizing tasks. Of note is the fact that six robots are significantly less accurate than higher numbers of robots, and also that the number of correctly identified tasks goes down for six robots as waiting time increases. This is because six robots are often unable to complete all the tasks, especially with longer waiting time. So the number of tasks they complete correctly goes down as well. Another thing to note is that, as seen above in Figure 18, accuracy for twelve robots peaks at a waiting time of 150 seconds.
Figure 20 plots success rate against waiting time, which, as previously mentioned, measures the percentage of times that the robots completed all the tasks. In fact, in these trials, it was only with a waiting time of 100 seconds that six robots ever completed all the tasks - with all the other waiting times, their success rate is 0%. This is a lot lower than the rates shown in the HelpBots trial (in Figure 17) because this trial has three times as many tasks, so it is three times harder for the robots to complete all the tasks.

6.4 HelpBots vs. NetBots

Figures 21 and 22 show side-by-side results comparing the HelpBots and NetBots algorithms, and Figures 23 and 24 show the values of the L1 and Kendall’s metrics for the NetBots experiments.
Figure 21. HelpBots vs. NetBots: Time (in minutes) plotted against number of correctly detected targets. Each graph shows data from four different numbers of robots (n), as well as from both the NetBots and HelpBots algorithms. The lower a line is on the graph, the faster the targets were detected.

Figure 21 shows time in minutes plotted against number of correctly detected targets. The dotted lines represent results from HelpBots, while the solid lines show results from NetBots. Each color represents results with a different number of robots. Finally, each graph shows results with a different waiting time.

As shown earlier in Figure 15, the experiments with higher numbers of robots complete in less time. On average, the NetBots algorithm does not complete tasks any faster than the HelpBots algorithm. This is to be expected because NetBots and HelpBots have the same behavior for finding tasks and cooperating to solve them; NetBots simply improves the method of voting on tasks.
Figure 22. HelpBots vs. NetBots: Correctly completed tasks plotted against waiting time (in seconds). Each graph shows results from experiments with a different number (n) of robots.

Figure 22 plots the average number of correctly identified targets against waiting time. Each graph shows data for a different number of robots. Each graph also shows data from the HelpBots and the NetBots algorithms side by side.

In the graph with n equal to six, the NetBots algorithm consistently identifies more targets correctly than the HelpBots algorithm. This is the expected result; after the robots’ PageRank vectors converge to the actual error vector, the robots successfully distinguish between accurate and inaccurate robots. When n is greater than six, however, there are too few interactions for the PageRank vectors to converge before all the tasks are completed. If we ran these trials with more tasks, the NetBots algorithm would perform better than HelpBots for larger numbers of robots. With thirty tasks, however, the graphs show that NetBots is on average as accurate as HelpBots for numbers of robots greater than six.
Figure 23. Value of the $L_1$ residual metric over number of correctly detected targets for the NetBots algorithm. Each graph represents a different waiting time, and each shows data for four different numbers of robots. Ideally the $L_1$ residual should converge to zero.
The above figures show results of evaluation metrics applied to the NetBots algorithm. Each figure has graphs for four different waiting times, and each graph has data from all four numbers of robots tested. Figure 23 plots the value of the $L_1$ residual against the number of correctly detected targets. In all situations, the $L_1$ residual decreases steadily to zero as time progresses. This shows that the error between the local and global PageRank vectors is decreasing over time; in other words, the vectors are converging, exactly as expected. Figure 24 plots the value of Kendall’s $\tau$-Distance metric against number of correctly detected targets. As with the $L_1$ Residual, the value decreases to zero over time, showing that the difference between the local and global PageRank vectors decreases as the experiments progress. This confirms that the values of the PageRank vectors are converging, as desired.

7. FUTURE WORK

In the future this work could be expanded upon and explored further in many different ways. Improvements can be made in the ranking methods, in how ranking is used, and in the applications our algorithm is applied to.

The robots’ ranks, in addition to being used when voting on a task, could be used when a robot receives multiple calls for help. Currently the robot simply treats them all as charges and responds to them all accordingly. To improve on this, however, the robots could send out their rank along with their calls for help. Then robots receiving multiple help calls at one time could use this information to prioritize the calls, giving calls from higher ranked robots priority.

Another way to improve the robots’ ranking algorithm is to give higher priority to robots in IR range. When a host calculates ranks, it can give priority to the opinions of the robots at its task over those of other robots.
by assigning them higher weights. This is helpful because robots will have the most accurate knowledge about links that directly affect them, since those will be weighted more.

The algorithm proposed in this paper was designed for heterogeneous robots, and therefore it lends itself very well to a system with learning. Even if robots start out homogeneous, there is no way to ensure they will learn the same things at the same rate. In fact, part of the advantage of learning systems is diversity; different robots learn different things, and the beneficial things persist. So systems with learning will tend toward heterogeneity, making them perfect candidates for the NetBots algorithm. In the future, this could be explored by introducing a learning element to the existing NetBots system.

8. CONCLUSIONS

The target recognition problem requires that a robot or robots locate and classify a collection of objects scattered throughout an arena. This problem is solved very well by a multi-agent system, because many robots are able to scatter and cover the area more effectively and efficiently than a single robot. In particular, a swarm system is well-equipped to solve this sort of problem. A swarm system is one where the robots' individual actions produce a cohesive group behavior. This is perfect for target recognition because the robots are able to act as a cohesive unit to complete tasks quickly and efficiently.

The goal of this paper was to introduce social networking into a swarm system to improve accuracy in solving the target recognition problem. In the past much work has been done on target recognition in different areas and with different approaches. This paper focused on improving upon the work done by Ijspeert et al.\textsuperscript{9} which used a help signal to aid in solving the problem. To test this approach, the HelpBots algorithm was designed, following the general idea laid out in the Ijspeert et al. paper. In this algorithm, the robots wander about the arena avoiding obstacles and looking for tasks. If they see a task they go to it and wait until enough robots have arrived to complete the task, or until too much time has gone past. While waiting they send out a help signal to all robots within range. If a robot receives a help call while wandering, it heads towards the task that the calling robot is working on.

To improve on this algorithm, we introduced social networking and ranked the robots based on their perceived past accuracy. Each robot stores a partial graph that contains information about all the links in the social network that it is aware of. Every time robots collaborate on tasks they combine the information in their graphs into a new, updated graph. In this way the robots strive to always have the most updated information about the social network. Links are formed between robots that agree with the majority decision at a task. Robots are then ranked using the PageRank algorithm, which ranks robots based on how many links they have and based on the rank of the robots linking to them. Robots that decide tasks correctly more often will have more links, so they will receive higher ranks. This way the ranks of the robots will eventually indicate their accuracy.

Results from experiments run on both the HelpBots and NetBots algorithms confirm expectations: for small numbers of robots, the NetBots algorithm is more accurate than the HelpBots algorithm. When the number of robots is low, the robot-to-task ratio is also low, so enough interactions occur for the robots’ PageRank vectors to converge to the actual error vector. For larger numbers of robots, this is not the case, because a higher robot-to-task ratio prevents the PageRank vectors from converging. In our work with thirty tasks, the NetBots algorithm shows no significant advantage over the HelpBots algorithm for numbers of robots greater than six. This confirms our initial belief that the NetBots algorithm is optimal for typical swarm robotics applications where the tasks far outnumber the robots.

ACKNOWLEDGMENTS

This material is based upon work/research supported in part by the National Science Foundation under Grant No. 0647120 and Grant No. 0647018. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
REFERENCES


