2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Trust Modeling in Multi-Robot Patrolling*

Charles Pippin[1] and Henrik Christensen[2]

Abstract— **On typical multi-robot teams, there is an implicit assumption that robots can be trusted to effectively perform assigned tasks. The multi-robot patrolling task is an example of a domain that is particularly sensitive to reliability and performance of robots. Yet reliable performance of team members may not always be a valid assumption even within homogeneous teams. For instance, a robot's performance may deteriorate over time or a robot may not estimate tasks correctly. Robots that can identify poorly performing team members as performance deteriorates, can dynamically adjust the task assignment strategy. This paper investigates the use of an observation based trust model for detecting unreliable robot team members. Robots can reason over this model to perform dynamic task reassignment to trusted team members. Experiments were performed in simulation and using a team of indoor robots in a patrolling task to demonstrate both centralized and decentralized approaches to task reassignment. The results clearly demonstrate that the use of a trust model can improve performance in the multi-robot patrolling task.**

I. INTRODUCTION

In multi-agent systems, the act of delegating a goal to another agent requires a mental belief, or *trust*, that the agent will reliably complete the goal [1]. However, trust is not often considered explicitly in multi-robot systems. In conventional approaches, each robot team member explicitly operates as part of a team and it may be assumed that a robot will perform according to an expected operational standard. However, fully autonomous robotic teams may have different quality levels and operational capabilities. Consider an example of search and rescue robots from multiple different organizations forming an ad-hoc team to cooperatively search for survivors after a disaster. Such teams may be able to negotiate using common standards, but would not be overseen by a single organization. Even within a group of homogeneous robots, there are differences in performance due to power levels, odometry calibration, wear and tear, and sensor noise, for instance. Therefore, these teams may need to learn which team members are trustworthy and dynamically adjust their control strategies. Robots can use observations of team member performance to build models of how well others can be trusted to perform various tasks. These observations could occur during online training or learning and through real world exploration.

*This work was supported internally by the Georgia Tech Research Institute.

[1]C. Pippin is a senior research scientist with the Georgia Tech Research Institute, Georgia Institute of Technology, Atlanta, GA, USA. pippin at gatech.edu

[2]H. Christensen is the Director of the Center for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA, USA. hic at cc.gatech.edu

The multi-robot patrolling problem is a surveillance task that uses multiple robots to repeatedly visit every important location in a known environment, with the goal of minimizing the time in-between visits. This problem is interesting from a multi-robot research perspective, because it presents challenges in optimization and task assignment, cooperation, communication and reliability. Cooperation is important in this task, as it is necessary for the robots to work together to improve the efficiency of the system as a whole. An effective multi-robot patrol team should be able to visit points more efficiently and with greater reliability than a single robot. However, reliability is also important, particularly in security applications. For instance, if robots on the team do not perform as expected, the system should degrade gracefully. Fully autonomous robot teams will require the ability to evaluate performance of team members for multiple reasons: human operators may not be able to manage large teams of robots in dynamic environments, robot teams may form in an ad-hoc fashion, and the performance metrics may not always be human observable.

The performance criterion considered in this paper is the *refresh time*, which is the time gap between any two visits to the same location[1]. The maximum refresh time reflects the bounds on the effectiveness of a robot team in detecting events in the environment [7]. If a robot fails to perform its assigned tasks or visits locations too infrequently, this will affect the performance of the team, and other robots should provide assistance.

The main contribution of this paper is the use of an observation based model for deciding when robots can be trusted to perform reliably in a multi-robot patrolling task. The probability based model is more robust to noise in observations as compared to threshold based approaches. The use of this model in the patrolling task results in improved performance when there are unreliable or untrustworthy robots on the team.

This paper is organized as follows. In Section II we present the related work for the multi-robot patrolling task. In Section III , we review the multi-robot patrolling task, and in Section IV, we introduce the use of a performance monitor and a trust model. In Sections V and VI, we present results of experiments using this approach on a team of indoor robots and in simulation. Finally, in Section VII, we conclude and present future work.

[1]In the literature, this is also referred to as the *idle time* of a node. In considering robot performance, we prefer to use *refresh time* to avoid confusion related to the idleness of a robot vs. the *idle time* of a node.

II. Related Work

Recent examples of work on the problem of cooperative patrolling by a multi-robot system are presented in [3], [4], [13], [11] and [9]. A theoretical analysis of the patrolling problem is provided by Chevaleyre [2]. The results showed that the problem could be solved with a Traveling Salesman Problem (TSP) approach. This is extended to the multi-robot case by spacing each of the robots evenly along the path [2], [3] in a cyclic TSP patrol. Chevaleyre also showed that it makes sense to partition the graph in some cases, particularly when there are long corridors or edges separating clusters of nodes. An example such case is shown in Figure 1 when $l > h$.

An approach for reassigning tasks from poorly performing team members was presented in Parker's L-ALLIANCE framework, in which a robot monitored a peer robot and took over a task from when the time for completing the task exceeded a threshold [6]. Pippin and Christensen presented an approach to monitoring robot performance as compared to the performance of the team for determining when a robot's performance could be considered out of control [8].

Once poorly performing team member have been detected, robots must decide whether to dynamically re-allocate tasks among the better performing team members. In [10] the authors presented an approach for using an auction algorithm for dynamic task reassignment. In that work, however, a threshold-based trust model was used. When the performance of a single robot exceeded the average performance of the team plus one standard deviation, the robot was marked untrusted.

The choice of a dynamic task allocation mechanism is independent of the application of a trust model. The trust model can also be applied to different dimensions for performance and reliability. In [9], a trust model was applied to a sensor detection task in a multi-UAV patrol to determine which team members can accurately perform a sensing task.

III. Multi Robot Patrolling

Many recent approaches to the patrolling task represent areas in the environment with a topological map (a graph) [7]. The nodes in a graph represent areas of interest in the environment, and edges in the graph represent traversable paths between two locations. Applying the notation from the literature, we can refer to the graph as $G(V, E)$, where $V = 1 \ldots n$ is the set of nodes and E is the set of edges. A weight is associated with each edge, $e_{i,j}$, representing the distance between each edge. The graph is assumed to be metric and undirected. Let $R = 1 \ldots r$ be the team of robots to assign the set of nodes in each of the r graph partitions. When the patrol task begins, there is an initial startup time for all robots to navigate to their assigned starting nodes in the graph and to begin patrolling. Robots patrol simultaneously and repeatedly along the graph, visiting their assigned patrol nodes, according to a given strategy [2].

Chevaleyre presents two main classes of patrolling strategies, the cyclic strategy and partition based strategies [2]. In the cyclic based strategies, a single closed path, s, is generated that visits all of the nodes in the graph at least once. In the single robot case, a robot travels this closed path indefinitely. In the worst case, the amount of time for a robot to visit a node twice while following this strategy is equal to the length of s. Calculating the closed path is known to be *NP-hard*, and this problem is closely related to the *Traveling Salesman Problem* [2].

In the multi-robot case, the simplest approach is to space the robots along the closed path such that during the patrol they maintain a constant distance between them [2], [3]. Cyclic strategies have known optimality bounds and are preferred when the graph does not contain long edges that connect clusters of nodes [2]. In addition, these strategies have a deterministic behavior and this may not be desirable for security application [11]. From a reliability perspective, when one robot malfunctions, the remaining $(r-1)$team members can simply space themselves evenly over the patrol cycle and continue patrolling. However, there are situations in which robots may have degraded performance, but continue to function. In these situations it would be desirable to allow the poorly performing robot to continue to perform a subset of its original patrol path.

IV. Approach

A. Graph Partitioning

Graph partition approaches divide the graph into subsets of nodes and assign these nodes to individual robots on the team. Pasqualetti et al. present optimality bounds for three major types of partition based patrol graphs: cycles, trees, and chains, and remark that the selection of the roadmap may not be unique for an environment and that the performance can vary based on the choice of the graph structure [7]. For the partitioning case, a cyclic graph can be transformed into an acyclic roadmap using min-max path cover approaches or a chain partition approach. For acyclic graphs, a tree based approach can be used. For the purposes of this paper, we convert a cyclic roadmap of the environment into a chain partition, using the approximation algorithm described in [7].

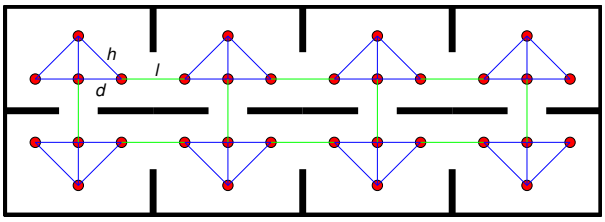

Fig. 1. The patrol graph is shown on a map of the *museum* patrol environment. Within rooms, nodes have a spacing of d for the horizontal and vertical edges, and of h for the diagonal edges. The graph can be optimally partitioned for 8 robots by cutting the long edges, l, between rooms. The optimal route within partitions is of length $2d+2h$. The partition approach results in a lower max refresh time compared to the cyclic TSP approach when $l > h$.

B. Trust Model

This work relies on the use of a probability based trust model, using the beta distribution [14], [5]. We rely espe-

cially on the trust mechanism from [14] for incorporating direct trust and reputation into a probabilistic formulation. This mechanism provides not only a trust belief about an agent, but also a confidence value. The approach can incorporate positive, α, and negative, β, histories to calculate the belief and confidence values.

The trust model maintains a set of α and β values for each robot that represent the histories of observations of that robot. For a given robot team member, if the calculated trust value is less than the trust threshold, τ, and with confidence greater than γ, it is not trusted. However, a succession of positive observations (direct or indirect) can move an untrusted agent back to being trusted again. Furthermore, this approach is tolerant of noise as it can take multiple observations to move the value above or below the trust threshold. To better explain this model, the equations from [14] for calculating the trust value τ and confidence, γ, are included below.

Given α and β updates for a dimension of trust for $robot_j$, it can calculate the Expected Value for trust using the trust model as follows.

$$E_{trust_{i,j}} = \frac{\alpha}{\alpha + \beta} \tag{1}$$

The value, $E_{trust_{i,j}}$, is the expected trust value that the trust model owned by $robot_i$ (or owned by a central trust model) has toward $robot_j$, given a set of observations, $O^{1:t}$, from the start through time t. Therefore, the trust value, τ, is

$$\tau = [E_{trust_{i,j}} | O^{1:t}] \tag{2}$$

The confidence factor, γ, represents the proportion of the beta distribution that is within ϵ of τ. This value is calculated as the integral of the PDF over the confidence interval, divided by the integral over the full interval. If there are relatively few observations, the distribution approaches the uniform distribution and the portion of the distribution that is within the confidence interval will be small, resulting in a low value for γ.

$$\gamma = \frac{\int_{\tau-\epsilon}^{\tau+\epsilon} X^{\alpha-1}(1-X)^{\beta-1}\,\mathrm{d}X}{\int_0^1 U^{\alpha-1}(1-U)^{\beta-1}\,\mathrm{d}U} \tag{3}$$

We define the set of *untrusted* robots, Υ, to include those with a trust score below the minimum trust threshold, $\tau < \theta_\tau$ and with confidence above the minimum confidence level, $\gamma > \theta_\gamma$. All other robots belong to the *trusted* set, T. The *Trust Authority* maintains the current sets T and Υ, and can be queried to determine the set membership for a robot.

Finally, the use of a trust model allows for the robot to include different dimensions into the trust calculation. Each dimension can be incorporated into the model and weighted.

C. Performance Monitor

In this work, we assume that an external monitor is available to observe robot performance. Approaches to monitoring depend on the environment, but may include human observation, observation by other robots, computer vision based techniques, and RFID tags for logging visits to locations. Here, we only consider that a monitoring technique is available for use by the system and that it can reliably report when a particular robot visits each node. In practice, we allow for each robot to broadcast results messages when a node is visited, and assume that these are reported truthfully and also that the network reliably delivers these messages.

Each robot in the patrol graph shown in Figure 1 can have 2-3 neighbors. We define the set of neighbors for a robot, r, to be N_r and the set of immediate neighbor nodes to be V_n^r. Regarding the features that describe the performance of the robot, the trust model described below can incorporate multiple trust dimensions, using a mixture of weights for each. Performance dimensions that may be considered as input to the trust model include those related to *sensors* (probability of detection, tracking accuracy), *actions* (execution time, distance, fuel consumed, trajectory accuracy) and *deliberation* (explicit cooperation, correctness of plans generated, appropriate behavior selection, etc.).

In this paper, we adopt the performance metric of *maximum refresh time*. That is, the goal of the system is to minimize the maximum refresh time for all nodes in the multi-robot patrol. When the refresh time of any robot's assigned nodes exceeds a threshold on this metric, we seek to re-assign some of that poorly performing team member's nodes to others. Each robot self reports node visits to the monitor which tracks the idle time for each node. At each time step, the monitor can calculate the node with the maximum refresh time for each robot. We set the amount of time in between performance monitoring periods to be the expected maximum refresh time for the patrol partition.

The max refresh time for a robot is the maximum refresh time for all nodes assigned to robot r. Let I_k^r be the set of the refresh times at the previous k node visits for a robot, r. Let I_n^r denote the refresh time of a node visited by robot r and being the nth visit by r to any node assigned to it. The running max refresh time for a single robot, $M_k^r = \max(I_{k..n}^r)$, is the observed maximum refresh time for a robot over the window $(n-k, \ldots, n-1, n)$, where $0 < k \leq n$, and M_k^R for all robots. The leave-one-out running max refresh average is the average running max refresh time over all other trusted robots, $\overline{M}_k^{T-r}$. Then, the threshold for the max refresh time, θ_{M_r}, is defined as the leave-one-out running max refresh average, plus ρ standard deviations (here, $\rho = 3$).

$$\theta_{M_r} = \overline{M}_k^{T-r} + \rho * \sigma; \tag{4}$$

We define a patrol period as the expected amount of time to perform a patrol of the maximum partition plus a constant factor. This factor is included to capture the additional time needed to navigate due to the nonholonomic motion of the robot and related to time spent navigating around obstacles. At the end of each patrol period, the monitor checks whether $M_k^r > \theta_{M_r}$ for each robot. In that case, a robot is considered to be performing poorly and the trust model is updated with a negative observation, β.

In addition to the max refresh time, we can consider the metric of average refresh time. The average node refresh time

for a robot, A_k^r, is the average refresh time per node for a robot in the last patrol period. The threshold for the average refresh time, θ_{A_r}, is defined as the average node refresh time for all trusted robots, plus ρ standard deviations.

$$\theta_{A_r} = \overline{A}_k^T + \rho * \sigma; \quad (5)$$

D. Task Reassignment Methods

We considered two types of task reassignment methods, for the centralized and localized task reassignment case. Both approaches rely upon a central trust authority for the maintenance and sharing of the trust model. These approaches are described further below.

1) Central Observation and Assignment: In this approach, the centralized monitor records the node visit frequency for each robot and updates the trust model with positive and negative performance observations when a robot is within or exceeds the performance thresholds, respectively. This algorithm is shown in Figure 2.

When a robot's max refresh time is observed to exceed the threshold, and this is not due to the robot assisting others, then the trust model is updated with a negative instance. In the other case, we do not automatically update the trust model with a positive result because the max refresh time could be low for a *poor performer*, if other robots have come to assist it, but the average for the remaining nodes could still be high. In this case, we also consider the average refresh time metric, and if it is below the threshold, the trust model is then updated with a positive signal.

If a robot moves from the trusted set to the untrusted set, $T \rightarrow \Upsilon$, the central monitor reassigns one of the nodes from the untrusted robot to a neighboring robot in T. Similarly, if a robot moves from $\Upsilon \rightarrow T$, the central monitor returns all of its original tasks. Once a robot assists another robot by taking a new node, it is added to the set of *Assistors*, so that it will not have its trust score penalized for the resulting increased refresh time.

2) Local Observation and Assignment: In this approach, each robot locally cooperates, but without coordination. This algorithm is shown in Figure 3. Here, each robot locally reports the performance observed for each of their neighbors, by observing the visit frequencies of the nodes in neighboring partitions, and sending positive and negative performance observations to the central trust authority when the average running refresh time for a node, A_k^v, exceeds the expected max cycle time. The expected max cycle time is defined as the expected time for a *good performer* to complete a full cycle, times a factor to allow for a small amount of motion error. In addition, robots periodically query the central trust authority to get the trust score for their neighbors. From all of the untrusted neighboring robots, the robot will select the most untrusted neighbor. If one is found, the robot will add the closest neighboring node from the most *untrusted* neighbor to its own patrol list, and send the task reassign message to the *untrusted* neighbor for that node. To prevent an assisting robot from itself becoming untrusted, a robot sends an *assisting neighbor* message to the trust authority.

```
1: loop
2:   Do Every P Seconds:
3:   for all r : Robots do
4:     if (M_k^r > θ_{M_r}) and (r ∉ Assistors) then
5:       UpdateTrustModel(r, β);
6:     else
7:       if A_k^r < θ_{A_r} then
8:         UpdateTrustModel(r, α);
9:       end if
10:    end if
11:  end for
12:  for all r : Robots do
13:    if r ∈ T → r ∈ Υ then
14:      g ← select ∈ T ∩ N_r;
15:      SendReassignTaskMessage(r, g);
16:    end if
17:    if r ∈ Υ → r ∈ T then
18:      SendReturnAllTasksMessage(r);
19:    end if
20:  end for
21: end loop
```

Fig. 2. The Central Observation and Assignment pseudocode: the central monitor observes node visits, updates the trust model and reassign tasks when a robot becomes untrusted.

Upon receipt of this message, the trust authority enters an annotation to the trust record for that robot which it uses to allow for decreased performance in the assisting robot.

V. EXPERIMENTS

A. Robot Platform

A set of experiments was performed using the TurtleBot indoor mobile robot platform.[2] The robot has a bumper sensor and a single axis gyroscope. The robot also uses a Kinect sensor, which includes an infrared laser projector and corresponding infrared camera which generate range data of the scene for indoor distances up to 6 meters. The TurtleBot carries a net-book laptop which runs Linux and the same ROS libraries and behaviors used in the simulation experiments. The open-source Robot Operating System (ROS) architecture [12] was used to implement the robot messaging, low-level control and behaviors. Each robot uses the ROS navigation stack for navigation, localization, and obstacle avoidance. Each robot also runs a custom *Patrol* behavior which implements the graph chain partition algorithm, and repeatedly navigates to the nodes in the robot's patrol path. The experimental setup also includes a central monitor node which listens for task completion messages and includes the performance monitoring and task reassignment components. Robots communicated with the central monitor by sending messages using UDP broadcast over the local network, and it is assumed that the robots honestly report task completion. Messages are paired with acknowledgements to ensure delivery.

[2]http://TurtleBot.com

```
loop
  Do Every P Seconds:
  for all v : V_n^r do
    if (A_k^v > E(MaxCycle)) then
      UpdateTrustModel(getNodeOwner(v), β);
    else
      UpdateTrustModel(getNodeOwner(v), α)
    end if
  end for
  u ← MostUntrusted(U ∩ N_r);
  if (u) then
    SendReassignTaskMessage(u);
    SendAssistingMessage(u);
  end if
end loop
```

Fig. 3. The Local Observation and Assignment pseudocode: the local monitor on each robot observes neighboring node refresh times, updates the trust model and reassign neighbor nodes to itself when a neighboring robot becomes untrusted.

In each experiment, one of the robots is explicitly marked as a *poor performer*. The performance for this type of robot is affected by randomly adjusting the maximum forward velocity of the robot after each visit to a patrol node. The robots that perform normally have a maximum speed of 0.25ms^{-1} and the maximum speed of the *poor performer* is determined by sampling from a normal distribution with $\mu = 0.15\text{ms}^{-1}$ and $\sigma = 0.10\text{ms}^{-1}$. This results in increased max refresh times for the patrol nodes assigned to the *poor performer*.

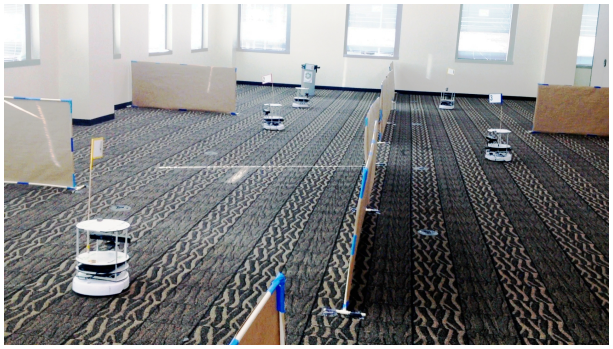

Fig. 4. Multiple TurtleBots are shown patrolling in the experimental environment, setup to resemble an art museum with multiple rooms.

B. Patrol Graph

Each robot, r, was provided with a copy of the environmental map and patrol graph, shown in Figure 1, as well as the i^{th} graph partition assigned to the r^{th} robot, which also corresponded to a single room. This graph has properties that make it easy to analyze the optimality for performing partitions. In the initial case, when it is assumed that all robots are performing equally well, it is easy to see that the optimal partition for 8 robots is to assign one robot to each room. In this environment, the partition approach results in better performance than the cyclic approach when the edge between the rooms is long. Referring again to Figure in Figure 1, this occurs when the length of the diagonal edge, h, is greater than the corridor distance, l, between the clusters of nodes in each room.

C. Experimental Setup

1) Robot Experiments: The *museum* experimental environment was designed to resemble an art museum with 8 equally sized rooms, as shown in Figure 4. The environment was approximately 10m x 30m in size. Upon startup, each robot began patrolling the nodes located in their partition. The setup also consisted of a centralized monitor node that recorded the frequency of visits to each node by robots. Upon completion of each node visit, robots broadcast a node visited message, using a UDP network broadcast, and this was recorded by the monitor. These messages were also available to the robots.

Two different types of experiments were performed using a centralized trust model, to compare the use of trust monitoring with centralized and local observation and task assignment approaches. Each experiment ran for over 30 minutes, with all 8 robots patrolling continuously.

2) Simulation Experiments: Simulations were run with eight robots on a team in the simulated *museum* environment. The experiments used the Stage multi-robot simulation environment [15], using the same ROS behaviors from the real robot experiments. Four different experiment types were performed, and each experiment was performed five times.

1) *naive strategy*: The robots patrol the set of nodes in the initial partition. One of the robots on the team was marked as a *poor performer*.
2) *central trust model strategy*: A central monitor observes the performance of the robots on the team and maintains a central trust model. When a robot becomes untrusted, the central monitor reassigns tasks optimally to a neighboring robot if providing assistance would result in an improved max refresh time, where the max refresh time for the assisting robot would be less than the current max refresh time..
3) *local trust model strategy*: A local monitor on each robot observes the performance of its neighboring robots on the team and maintains a local trust model, with local task reassignment. When a neighbor robot becomes untrusted, the local strategy will assist if no other neighbors have already assisted. The monitoring and coordination between robots is performed by sending messages.
4) *all perform*: The robots patrol the set of nodes in the initial partition as expected. None of the robots are *poor performers*.

VI. RESULTS

A. Robot Experimental Results

In the *centralized* task assignment approach, robot 3 was explicitly set as a *poor performer*, after several minutes of normal performance. The central monitor observed each

robot's performance and updated the trust model with *positve* or *negative* observations, based on the robot's performance. The trust model scores for each robot in this experiment are shown in Figure 5. It is worth noting that the trust score for robot 4 and also robot 6 dipped briefly during the experiment due to localization errors. However, the model allows for noise tolerance, and the trust scores recovered when the robot's localization recovered.

After the robot 3 was observed performing poorly, its trust score decreased until it reached the low threshold (0.5) for trust and became *untrusted*. At this point, the central monitor dynamically reassigned one of the *poor performer's* tasks to the trusted robot 1, as shown in Figure 6. The robot trajectories during the experiment reflect this task reassignment, with robot 1 picking up a patrol node from robot 3, as shown in Figure 7.

The refresh times for robot 3 and robot 1 are shown in Figure 8. The values for robot 1 are typical for all *good performers*. The refresh time for robot 1 increased after the task reassignment, because of the additional time to cover the reassigned node. However, the max refresh time across the team was improved as a result. We also plot the expected max refresh time. However, the actual max refresh time for the *good performers* is slightly higher due to the motion model for the robot.

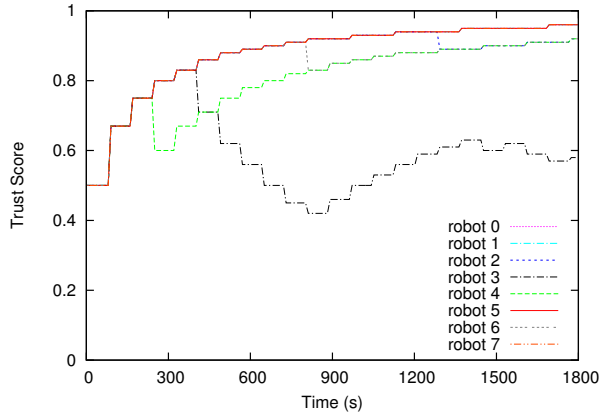

Fig. 5. The trust scores for each robot are plotted during the experiment. The trust score for robot 3 decreases as the robot begins to perform poorly. The rest of the robot's trust scores increase monotonically, with the exception of robots 4 and 6, whose scores are slightly decreased due to temporary localization errors.

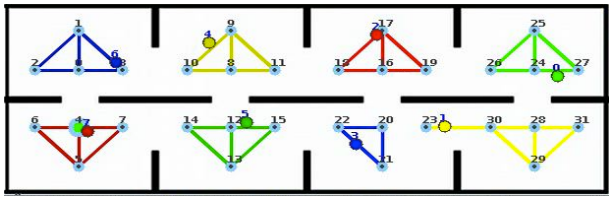

Fig. 6. The tasks assignments are shown on the task monitor's display. The centralized approach reassigned a task from poorly performing robot 3 to the trusted, neighboring robot, 1.

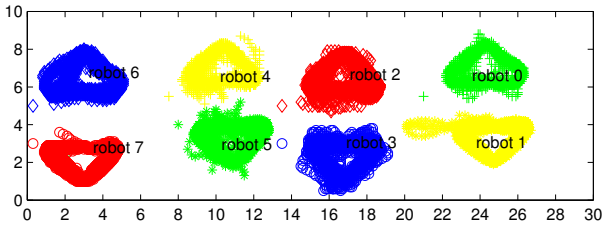

Fig. 7. The trajectories of each of the robots are shown for the entire central trust strategy experiment.

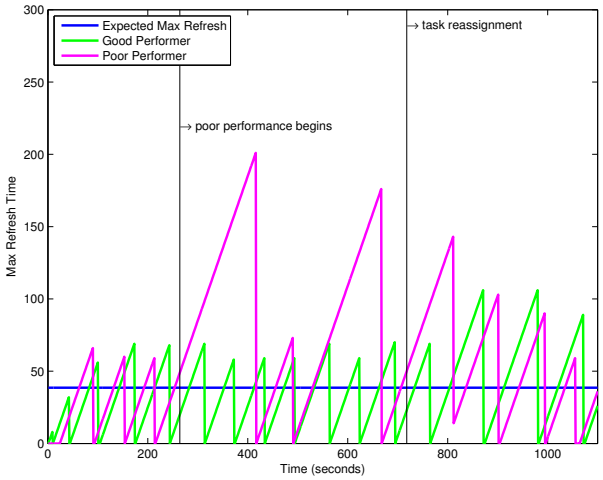

Fig. 8. Central Strategy: The refresh time for a *good performer* and a *poor performer* is shown during the period before and after the initial task reassignment. The refresh time increases due to the *poor performer* robot 3, but decreases after a task reassignment. (The performance of the *good performers* initially exceeds the expected max refresh time due to nonholonomic motion model of the robots.)

In the *local* assignment approach, each robot reported the trust of their neighbors to the central authority, which maintained the trust scores. Here, we again explicitly designated robot 3 as a poor performer, this time from the beginning of the experiment. Over time, this caused its trust score to drop below the threshold. Each of the 3 neighbors to robot 3 observed this and each took over a task, leaving robot 3 with only 1 node to patrol. The neighbors each reassigned a task to themselves and sent a *reassign task* message to robot 3. As shown in Figure 9, the experimental monitor updated the display to reflect the task reassignment after receiving these messages, but it is not necessary to the experiment. The robot trajectories during the experiment reflect this task reassignment, with robots 1, 5, and 2 picking up a patrol node from robot 3, as shown in Figure 10. An additional observation was that robot 6 performed poorly, perhaps because there were additional obstacles in its environment and this caused neighboring robots 4 and 7 to come over and assist it as well. A photo from the viewpoint of Robot 1 is shown in Figure 11, reflecting robots 1 and 2 in the partition of robot 3 to pick up tasks.

The refresh times for robot 3 and robot 1 are shown in Figure 12. The refresh time for robot 3 drops to almost zero

after it is left with only 1 node to cover. The max refresh time for robot 1 (and the other assisting robots) are similar to those for the previous experiments, in this case multiple robots are performing assistance. A benefit of this approach is that multiple robots can affect the trust score, rather than relying on a centralized observer. A possible extension would be to allow the trust reporting for a robot to be weighted by the trust level of the robot reporting the score.

The centralized assignment approach has the advantage of being able to optimally reallocate tasks in this environment; however, it may not always be possible to use a central task allocation. The local method can also be used, however, this preforms a greedy re-assignment and could result in multiple neighbors assisting the same robot unless a coordination mechanism is used.

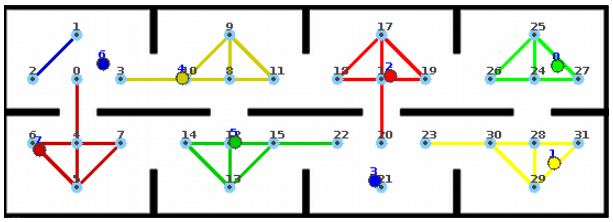

Fig. 9. The three neighbors of the poorly performing robot 3 each pick up a task. Robot 6 also performed poorly due to localization errors and also had tasks picked up by its neighbors.

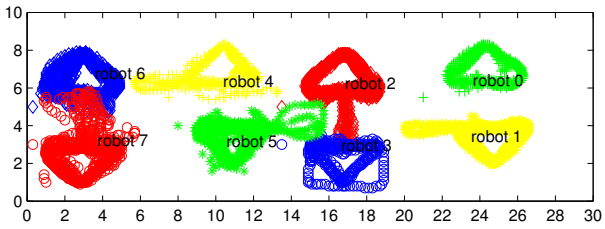

Fig. 10. The trajectories of each of the robots are shown for the entire local trust strategy experiment.

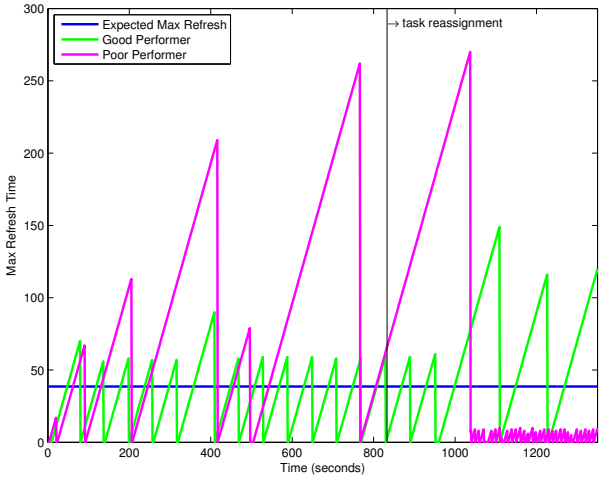

Fig. 12. Local Strategy: The refresh time for a *good performer* and a *poor performer* is shown during the period before and after the initial task reassignment. The refresh time increases due to the *poor performer* robot 3, but decreases after a task reassignment.

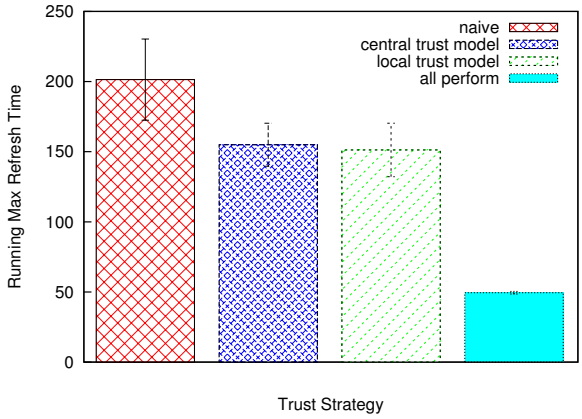

Fig. 13. Experimental results from simulations in the *museum* environment are shown. Both the centralized and local trust strategies result in improved patrol performance over the naive strategy, when a *poor performer* is present. Error bars represent one standard deviation.

B. Simulation Experimental Results

The results from the simulations are shown in Figure 13, with the running max refresh time, M_k^R, shown for each of the strategies. The simulation results are consistent with the experiments in the real environment. Both the central trust and local trust strategies resulted in an improved max refresh time over the naive approach, when a poor performer was present. In these experiments, the local trust approach used a coordination mechanism between neighboring robots to prevent multiple robots from assisting the same poor performer. While the trust model approaches resulted in improved max refresh times, there is still room for improvement. In both trust model cases, task assignment approach only reassigned a single side node to a neighboring robot, reducing the patrol distance from the poor performer by a distance of h, from $2d+2h$ to $2d+h$, while adding a distance of $2l$ to the assisting robot. Assignments from multiple neighboring nodes could reduce the max refresh time further, but with an additional resource expenditure for the assisting robots.

C. Discussion

For both the central and local assignment approaches presented here, the use of a trust model allows for more tolerance for noise in the system and for exploitation of performance history. With a threshold only approach, a single noisy observation could cause a robot to become untrusted, resulting in task reassignments. However, the use of a trust model incorporates multiple observations and in the local case, can incorporate observations from multiple observers.

While both approaches result in an improved max refresh time, the central approach can more efficiently allocate robots, but requires a central mechanism which may not always be possible. On the other hand the *local* approach performed a decentralized task allocation with no negotiation

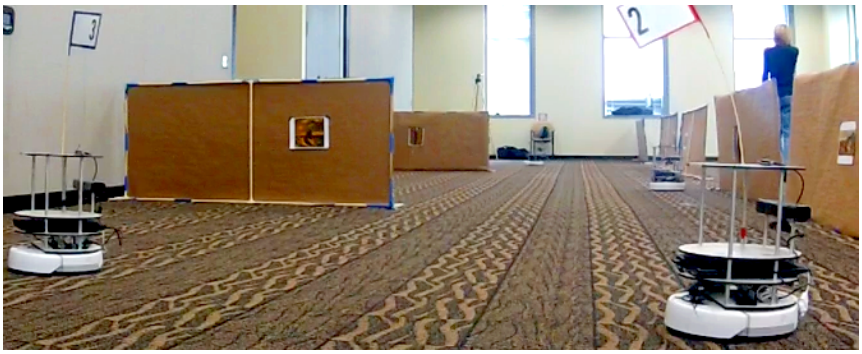

Fig. 11. Multiple robots are shown patrolling the *museum* environment from the viewpoint of a camera placed on robot 1. In this experiment, after robot 3 is observed performing poorly, its neighbors each pick up one of robot 3's tasks and send it a task reassignment message.

between robots or a central node, excluding the *reassign task* message. However, this still required a central trust authority. Additional experiments in simulation using a local trust model and local reassignment showed that with more coordination between robots a local reassignment can be used without over subscribing the assisting robots. In practice, the decision for where to place the trust authority and task reassignment mechanism is dependent on several factors of the environment, including communication, the state model, the trust monitoring approach to observation and task reassignment approach.

It is also worth noting that the design of the environment may affect the ability for a robot to assist a teammate, because there is a cost associated with traversing the corridor between patrol partitions. If it is expected that robots will need to frequently assist each other, it may be worthwhile to redesign the placement of the nodes and the size of the team Finally, it might be useful to reallocate all tasks belonging to a poorly performing robot and re-partition the tasks among the remaining $(n-1)$ robots if this would result in better max refresh times.

VII. CONCLUSIONS

This paper presents a method for using robot performance observations to build model of robot trust and apply it to dynamic task allocation in the multi-robot patrolling problem. The experimental results showed that a monitoring approach with trust modeling can be effective for detecting poorly-performing team members. In addition, a task reassignment mechanism can be effective for more efficiently re-assigning patrol tasks, when compared to the naive approach which does not monitor individual robot performance or adjust task assignments. This may prove useful in situations in which multi-robot teams are dynamically formed or when not all team members are likely to perform effectively over time.

Future work will explore the use of a multi-dimensional trust model and apply this model to other problem domains in multi-robot systems.

ACKNOWLEDGMENT

C. P. thanks Stephen Camp for his assistance with the experimental setup and execution. The authors also appreciate the helpful comments from the anonymous reviewers.

REFERENCES

[1] C. Castelfranchi and R. Falcone. Social trust: Cognitive anatomy, social importance, quantification and dynamics. In *Proceedings of the First International Workshop on Trust*, pages 35–49, 1998.

[2] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, pages 302 – 308, Sept. 2004.

[3] Y. Elmaliach, N. Agmon, and G. Kaminka. Multi-robot area patrol under frequency constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 385 –390, April 2007.

[4] K.-S. Hwang, J.-L. Lin, and H.-L. Huang. Cooperative patrol planning of multi-robot systems by a competitive auction system. In *ICCAS-SICE, 2009*, pages 4359 –4363, aug. 2009.

[5] A. Jøsang and R. Ismail. The beta reputation system. In *Proceedings of the 15th Bled Electronic Commerce Conference*, 2002.

[6] L. E. Parker. ALLIANCE: An architecture for fault tolerant multi-robot cooperation. In *IEEE Transactions on Robotics and Automation*, volume 14, pages 220–240, 1998.

[7] F. Pasqualetti, A. Franchi, and F. Bullo. On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *Robotics, IEEE Transactions on*, 28(3):592 –606, June 2012.

[8] C. Pippin and H. Christensen. Performance based monitoring using statistical control charts on multi-robot teams. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 390 –395, July 2012.

[9] C. Pippin, H. Christensen, and L. Weiss. Dynamic, cooperative multi-robot patrolling with a team of UAVs. In *SPIE. 8741, Unmanned Systems Technology XV*, number 874103, May 2013.

[10] C. Pippin, H. Christensen, and L. Weiss. Performance based task assignment in multi-robot patrolling. In *Proceedings of the 2013 ACM Symposium on Applied Computing*, SAC '13. ACM, March 2013.

[11] D. Portugal and R. Rocha. On the performance and scalability of multi-robot patrolling algorithms. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 50 –55, Nov. 2011.

[12] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. *In Proceedings of the Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA)*, 2009.

[13] E. Stump and N. Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. In *Automation Science and Engineering (CASE), 2011 IEEE Conference on*, pages 569 –575, Aug. 2011.

[14] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck. TRAVOS: Trust and reputation in the context of inaccurate information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 12, 2006.

[15] R. Vaughan. Massively multi-robot simulation in Stage. *Swarm Intelligence*, pages 189–208, 2008.