Performance Based Monitoring using Statistical Control Charts on Multi-Robot Teams

Charles Pippin
Georgia Tech Research Institute
Georgia Institute of Technology
Atlanta, Georgia 30332
Email: pippin@gatech.edu

Henrik Christensen
Center for Robotics and Intelligent Machines
Georgia Institute of Technology
Atlanta, Georgia 30332

***Abstract*—On typical multi-robot teams, there is an implicit assumption that robots can be trusted to effectively perform assigned tasks. However, reliable performance of team members may not always be a valid assumption. For instance, a robot's performance may deteriorate over time or a robot may not estimate tasks correctly. Traditional health monitoring techniques call attention to an operator or assume a binary classification of either success or failure. Robots that can identify poorly performing team members, as performance deteriorates, can adjust the task assignment process dynamically. This paper investigates the use of statistical process control charts from operations research as a tool for monitoring team member performance as part of a multi-robot task assignment framework.**

I. Introduction

The area of multi-agent systems has been an active area of research for many years, due in no small part to the ability for a team of robots to operate more efficiently and be more robust to failure than a single robot. However, there are still many challenges related to the interaction between the robots themselves. Robotic teams may have different quality and performance capabilities, costs, and owners. As such, robots may need to learn which team members reliably estimate and perform tasks as part of a team. This work describes approaches for using statistical control charts applied to the task performance of team members in a multi-robot task assignment domain.

There are two central issues[1] related to robot performance on multi-robot teams [1]: 1) Can a robot detect when other team members are not performing tasks as expected, and 2) what actions should a robot take when poor performance is detected?

In the first case, it is important to consider performance monitoring techniques from the viewpoint of the robot, rather than from that of the human operators. Future robot teams may be formed dynamically, and robots may choose team members based on observed performance. In the second case, the robot could choose to notify the operator or complete the task themselves [1], adjust the cost function based on performance characteristics [2], [3], attempt to provide aid [4], or perhaps choose to remove the poorly performing robot from the team [5]. This paper will consider the first issue and present a model for monitoring performance of bid estimates vs. actual task completion times in an auction based multi-robot task allocation problem.

[1]Parker also describes a third issue related to performance, that being how or whether to diagnose the problem.

A. Performance in Multi-Robot Auctions

While there are many mechanisms for performing multi-robot task assignment, here we will focus on a decentralized, market based task assignment approach. Market-based auction methods solve the multi-robot task allocation problem by splitting computation across multiple nodes and iteratively performing task assignments [6]. These algorithms generally do not explicitly consider individual team member performance when allocating tasks. However, there are situations in which the individual robots on the team may have varying levels of performance and task estimation accuracy. In order for tasks to be allocated efficiently, it is important to be able to reliably trust that robots will perform their assigned tasks with costs that closely approximate their estimated costs and abilities. If a robot regularly exceeds its estimated cost for performing a task, the assignment algorithm should be able to detect this condition, and adjust the approach to task assignment.

B. Control Charts

Control charts are a statistical process control tool from the field of operations research [7]. Control charts are a widely used tool to monitor process quality and to detect when a measured process deviates beyond an acceptable level of performance. In this paper, we apply control charts to the problem of detecting when a robot's performance exceeds an acceptable threshold. Control charts can be used to distinguish between acceptable noise in the process and abnormal operation. We believe that this approach could be used to allow robots to detect when team members begin to perform poorly. If poor performance can be detected early, a robot team could take steps to address the issue and improve the team score.

The rest of this paper is organized as follows. In Section II, we present the background and related work. In Section III, we discuss the use of control charts for monitoring task performance within an auction framework. In Section IV, we present results of simulated experiments using this approach. Finally, in Section V, we conclude and present future work.

II. Related Work

Control charts are commonly used to monitor industrial process performance but can be applied in many domains. Some examples include animal production systems [8], machine fault detection [9], and public health surveillance [10].

The ability to determine when a robot is not performing or functioning as expected can be used to re-assign tasks or call attention to an operator. Parker's L-Alliance framework addressed the problem of improving efficiency and fault tolerance in a multi-robot team [11]. The goal in that work was for robots to minimize the time to complete a task. Therefore, time was treated as a quality measure, wherein each robot on the team kept track of the average time, plus one standard deviation for that robot to perform a task. That approach is very similar to the use of control charts described herein; however, the robots relied on a behavioral framework as a mechanism for assigning tasks. This paper will also treat time as a quality measure, as compared to the initial task estimate. In addition, this paper further validates the running average approach by incorporating the control chart methods which have been heavily researched in other domains.

Additional approaches to robot performance based metrics are also presented by Parker in [1]. This included a discussion of qualitative and quantitative metrics, such as mean time between failure or repair. The work also included the notion of effectiveness metrics, which seek to evaluate the success or failure of a task in retrospect. Additionally, the work related the use of statistical models to detect faults when a robot is in an inconsistent state as part of a sensing task. An open challenge mentioned in the work was the need for techniques that can infer the impact of robots that have partially failed, as well as approaches for handling the partial failures.

In [12], several different approaches to a trust model representation are discussed, including the use of continuous and discrete numerical models, binary models and probabilistic models. In that work, trust is applied to the information fusion problem, by incorporating it into a Kalman filter process. Trust is describe as being multi-dimensional, based on the domain. For instance, in computer networks, trust can refer to the trustworthiness of a sensor (whether it has been compromised), the quality of data from the sensor, or the security of the link between sensors. This paper defines trust as a robot performance metric, however, the use of multiple trust dimensions is instructive.

Jones, Dias and Stentz investigated techniques for learning proper task cost estimates in oversubscribed domains, using auction algorithms [13]. In that work, each robot attempted to learn their own bid estimates, and had full knowledge of their own state vectors, including their own schedule. In this paper, we are interested in determining whether bids accurately match their estimated values, but from the viewpoint of the robot auctioneer.

III. Approach

In the basic multi-agent auction algorithm, the problem is to assign tasks to agents. In this case, the task is to visit a target location and perform an observation. In the auction framework for task exchange, each robot is a *bidder* and the items to be auctioned are the tasks. Each of the agents in the system also participates as an *auctioneer* and periodically auctions new task requests (it is assumed that the task requests are periodically provided to the agent by an external process, such as a human operator or other event). This approach can easily be used on teams with different robot characteristics: each robot knows their own location and cost function and submits cost based bids to the auctioneer. While costs and rewards use the same basis for calculation, no revenue is actually exchanged. Rather, an agent awards itself a utility value when one of its own tasks is completed.

In this work, the agents each maintain a current task list and locally compute their bid to complete the proposed task. The bid consists of the time-based cost to perform the task. A potential source of error in task estimation is in the use of an insertion heuristic for calculating the marginal cost to perform a task, in addition to those tasks already assigned. In this paper, each robot plans to visit the targets in the order in which they were assigned (using the $O1$ assignment rule from [14]). For each auction announcement received, each robot calculates its bid as the amount of time required to complete the task in addition to those on the current task list. When the winning *bidder* is assigned a new task, the task is appended to the robot's assigned task list.

A. Cost Factor Metric

At this point, we can define a *performer* as the robot that completes a task on behalf of another robot, the *originator*, who requested assistance. As described above, the robots can expect better performance if they are able to exchange tasks with other team members that can complete them more efficiently. Upon task completion, the task *performer* notifies the *originator* that the task was completed.[2] When the task is completed, the *originator* calculates the actual time to complete the task, t_{actual}, and receives a reward according to the decreasing reward function from [14]: $f(t_{actual}) = (a - t_{actual}b)$, where a is the task reward and b is the decrease factor. In the linearly decreasing reward problem setup, it becomes even more important for tasks to be completed on time.

At this point, the *originator* can calculate the cost factor, $CostFactor_{B_a}$ from the ratio of the actual task completion time, t_{actual}, to the estimated task completion time, t_{est}, for the bid, B, by the *performer*, p.

$$CostFactor_{B_p} = \frac{t_{actual}}{t_{est}} \tag{1}$$

It is worth noting that a robot could miss the original cost estimate for a number of reasons: the cost function could be using an inaccurate insertion heuristic, the robot could be low on power and moving more slowly during actual task execution, or it could have unknown knowledge of its own

[2]The task completion notification could also come from an external monitor process.

internal state, for instance. Therefore, it would be useful to be able to allow for occasional variation in the process, but to recognize when a robot is performing beyond an acceptable degree of variation in relation to the team. To address this issue, we employ control charts as a threshold mechanism to the process of robot task estimation.

B. Control Charts

Statistical process control (SPC) is used in the operations research field to improve processes through monitoring and statistical analysis. A commonly used tool in SPC is the control chart [7]. The control chart can be applied to situations when a process needs to be monitored over time against quality thresholds. Often, they are used as a graphical tool by managers of the process to detect and communicate instability. However, the time series data can be computed and used to monitor performance online. Control charts can monitor multiple process quality characteristics in the multivariate case, or a single feature in the univariate case. Here we are interested in monitoring a single feature value. An example graphical control chart is shown in Fig. 1. In general, to use the tool, samples are taken from a process over time and plotted, along with the known mean value for the process [7]. The known mean value is referred to as the center line (CL). Two other known limits are plotted, the upper control limit (UCL) and lower control limit (LCL).

The underlying concept of the control chart is that a process can have variation from two causes, common causes or assignable causes. Common causes are those that are due to small, unavoidable causes or noise in the production process. Assignable causes are those that are due to an unplanned, irregular behavior of the system. Assignable causes may be due to changes in the environment, a malfunction, or other unplanned change. When an assignable cause occurs, the system is said to be out of control [8] . The challenge, then is to separate the assignable causes from the common causes and seek to address the assignable causes before they significantly cause damage, increase cost or slow the production process.

The purpose of the control limits is to allow for the process to experience some natural variation before costly intervention occurs. Therefore, the selection of values for the control limits can be arbitrary and is dependent on the risk tolerance for the designers of the system and the tradeoff between the cost of false alarms and missed detections. Often, these limits are set to three times the standard error of the process [7]. When the measured process statistic exceeds the control limit, the process is determined to be out of control.

C. Trust Model

This work relies on the use of a a probability based trust model, using the beta distribution [15], [16] We rely especially on the trust mechanism from [15] 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.

Each agent maintains a set of α and β vectors that represent the histories of interactions with each team member. For a given 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 [15] for calculating the trust value τ and confidence, γ, are included below.

When an agent receives new α and β updates for a dimension of trust, it can calculate the Expected Value for trust using the trust model as follows.

$$E_{trust_{i,j}} = \frac{\alpha}{\alpha + \beta} \tag{2}$$

The value, $E_{trust_{i,j}}$, is the expected trust that $robot_i$ has toward $robot_j$, given a set of observations, O, from the start through time t. Therefore, the trust value, τ, is

$$\tau = [E_{trust_{i,j}} | O^{1:t}] \tag{3}$$

The confidence factor, γ, is calculated as the proportion of the beta distribution that is within ϵ of τ.

$$\gamma = \frac{\int_{\tau-\epsilon}^{\tau+\epsilon} X^{\alpha-1}(1-X)^{\beta-1}\,\mathrm{d}X}{\int_0^1 U^{\alpha-1}(1-U)^{\beta-1}\,\mathrm{d}U} \tag{4}$$

D. Shared Reputation

In addition to the observations from direct interactions with other agents, this approach allows for the agents to incorporate indirect observations from other trusted team members, known as shared reputation information. The intent for sharing reputation information among team members is to quickly spread information about trusted or untrusted agents to the rest of the team. If an agent can rely on reputation information from other agents, it might be spared from negative direct interactions with uncooperative agents. However, the shared reputation information must be combined with the locally observed trust vectors. In this framework, each agent can regularly post their trust model's α and β vectors to all other team members that are within range.

In addition, agents only incorporate those updates from other currently trusted team members. These shared, indirect, observation vectors are easily integrated into the local vectors and the scalar trust and confidence values are recalculated. An example of multiple trust observations from different robots being combined into a single model is reproduced from [15] and illustrated in Fig. 2.

Finally, the use of a trust model allows for the robot to include different dimensions into the trust calculation. Although in this paper, we only consider the cost factor for performance, other factors such as participation and task completion percentage could easily be incorporated into the model and weighted appropriately.

E. Monitoring Task Performance

At this point, we can relate how the control charts are used to inform the trust model to monitor performance in multi-robot auctions. To use a control chart, it is necessary to first define the value for the CL, which is the mean performance. On a multi-robot auction, this value could be interpreted as the average cost factor for performing a task. In the ideal case, the average is 1: each robot perfectly estimates the amount of time that it will take to complete a task. However, in practice, this value could change dynamically, based on environmental factors, the number of tasks being assigned, the path cost heuristic being used, and other factors. We will define the value for CL to be the equal to the mean cost factor for the *good performing* type robots to complete tasks under normal conditions. Initially, we define the set of *good performing* type robots to be the entire team. We will define trust in this case to be directly related to performance: a *trusted* robot is one that estimates and performs tasks efficiently. However, using the trust model described in section III-C, the set of trusted robots could change if the performance is shown to be out of control.

The pseudocode for this algorithm is shown in Fig. 3. Each robot on the team maintains a separate trust model and control chart for each team mate. At each time step, the robot samples the cost factor values for tasks that have completed, and maintains a running average over a time window, R. We incorporate a running average to allow for minor changes in the environment that would affect all robots equally (such as an increased number of tasks.) The running average is used as the value for CL on the control chart.

To calculate the running average, we include only those averages from other trusted robots, see Fig. 4. In line 2, we check whether an agent is trusted, see Fig. 5, before including that agent in the running average. We also calculate the standard deviation of the running average for all trusted robots, and add that to the CL line to get the value for the UCL line, as shown in Fig. 4, line 8. The LCL does not apply in this case as we are only concerned with robots that exceed their time estimates. To smooth variations in the performance values, we also calculate a running average for the cost factor. When the cost factor exceeds the UCL value, an out of control condition is detected.

At this point, the trust model is updated with a β signal to reflect that the robot did not perform well. If the robot performed close to their original estimate, the trust model is updated with the α signal to reflect that the robot is a *good performer*. The trust model provides an additional level of smoothing in the data and can prevent a single bad reading from causing a robot to be untrusted. The parameters of the model can be adjusted to adjust the rate at which changes affect the outcome. Additionally, the trust model allows for the incorporation of observations from other trusted robots, as described in Section III-D.

Once a robot becomes *untrusted*, their performance characteristics are no longer included in the running average calculations for the process mean. Additionally, the robot can use this information as part of the decision process for handling a poor performing team member (such as notify an operator, provide assistance, to adjust the task allocation, etc.).

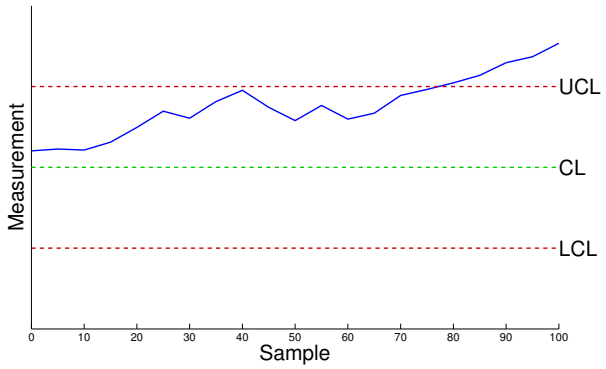

Fig. 1. An example control chart. The control chart includes a mean for the process, the center line (CL); an upper control limit (UCL); and a lower control limit (LCL). When a statistical process exceeds one of the control limits, the process is considered to be out of control.

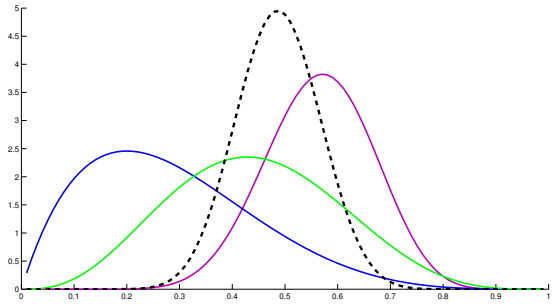

Fig. 2. The Beta Trust Model. Multiple trust models from different robots are combined into a single beta distribution (plotted with the dashed line). The model provides for a trust value, τ, and confidence factor, γ.

IV. Experiments and Results

A. Experimental Setup

A set of experiments were performed in simulation to test the cost factor learning approach in a multi-agent auction environment. In these experiments, each robot has 50 tasks that arrive at regular intervals and are sequentially auctioned by that robot's auctioneer. As part of the auction process, they also bid on their own tasks. No currency is actually exchanged as part of the auction framework.

Rewards are given for task completion to the robot that originated the task. Each robot submits bids that represent the time-based cost for completing a task. Specifically, the bid represents the number of time steps until the task could be completed. Once a robot finishes all tasks in its list, they no longer accumulate costs in the simulation. The initial locations of the robots and the tasks are randomly chosen for each iteration.

1: $t_{actual} \leftarrow (CompleteTime_{B_p} - StartTime_{B_p})$
2: $Reward_{B_p} \leftarrow f(t_{actual})$
3: $ActualCost_{B_p} \leftarrow (t_{actual})$
4: $CostFactor_{B_p} \leftarrow ActualCost_{B_p}/EstCost_{B_p}$
5: $runningAvg_p \leftarrow CalculateRunningAvg(CostFactor_{B_p})$
6: $UCL \leftarrow CalculateUCL()$
7: **if** $runningAvg_p \geq UCL$ **then**
8: $\quad UpdateTrustModel(CostFactor_{B_p}, \beta)$
9: **else**
10: $\quad$ **if** $runningAvg_p \approx CL$ **then**
11: $\quad\quad UpdateTrustModel(CostFactor_{B_p}, \alpha)$
12: $\quad$ **end if**
13: **end if**

Fig. 3. OnTaskComplete() pseudocode. The cost factor, $CostFactor_{B_p}$, is the ratio of the estimated vs. actual task completion time. The running average of the $CostFactor_{B_p}$ is monitored using a control chart and when the process is out of control the trust model is updated.

1: **for all** r in RobotTeam **do**
2: $\quad$ **if** CanTrust(r) **then**
3: $\quad\quad runningAvg_r \leftarrow CalcRunningAvg(CostFactor_{B_r})$
4: $\quad$ **end if**
5: **end for**
6: $CL \leftarrow CalcAvg(runningAvg_R)$
7: $stdDev \leftarrow CalcStdDev(runningAvg_R)$
8: $UCL \leftarrow CL + stdDev$
9: **return** UCL

Fig. 4. CalculateUCL() pseudocode. The UCL value is calculated as the mean running average value of all trusted agents, plus one standard deviation.

B. Task Estimation

In this paper, the source for estimation error is assumed to be due to *poor performing* type robots having an incorrect model of their own performance capabilities. To simulate robots that bid and execute poorly, a percentage of robots on the team are modeled as *poor performer* types and a cost factor is applied to their movements to slow their progress. However the robots themselves have no knowledge of the change to their state.

1) Consistently Poor Performance: In this experiment, 2 out of 6 robots on the team are marked as poor performers. Their performance is adjusted by randomly assigning a cost factor at the start of the experiment, using a normal distribution with $\mu = 3$ and $\sigma = 0.1$. When a *poor performer* bids on a task, the unknown cost factor is drawn from this distribution

1: **if** $\tau \leq MinTrust$ AND $\gamma \geq MinConf$ **then**
2: $\quad$ **return** FALSE
3: **else**
4: $\quad$ **return** TRUE
5: **end if**

Fig. 5. CanTrust() pseudocode. The beta trust model can be used to determine if an agent is untrusted, when a robot has a low trust value, τ, with high confidence, γ.

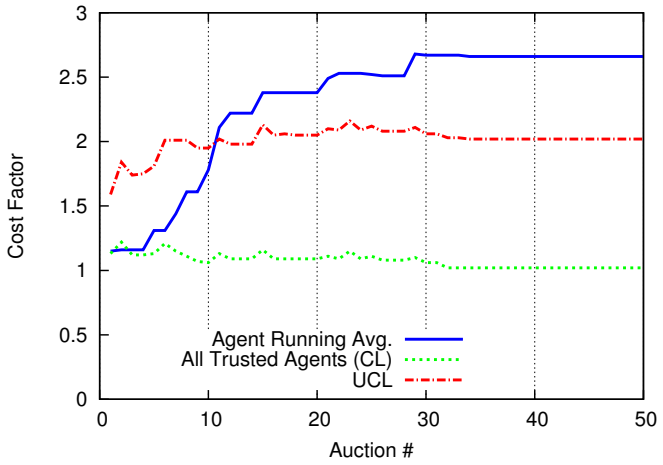

Fig. 6. Detecting performance of robots that consistently underestimate the cost for performing tasks. Here, the control chart is shown for one of the agents that is not performing well.

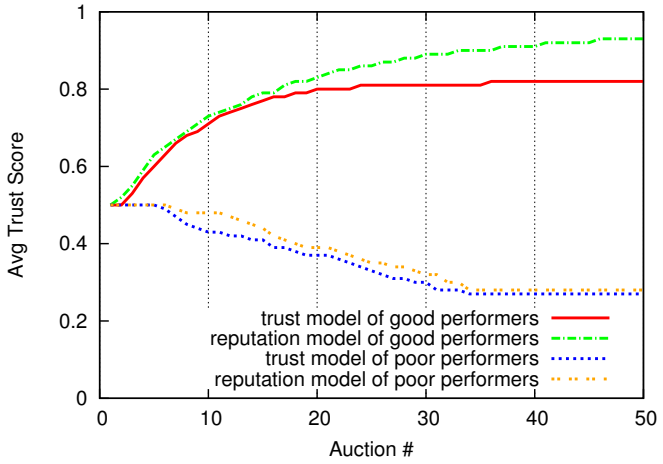

Fig. 7. Trust Model. The beta trust model can be used to determine when an agent can no longer be trusted to reliably perform a task. Negative feedback is given to the model whenever an agent exceeds the limit on the control chart.

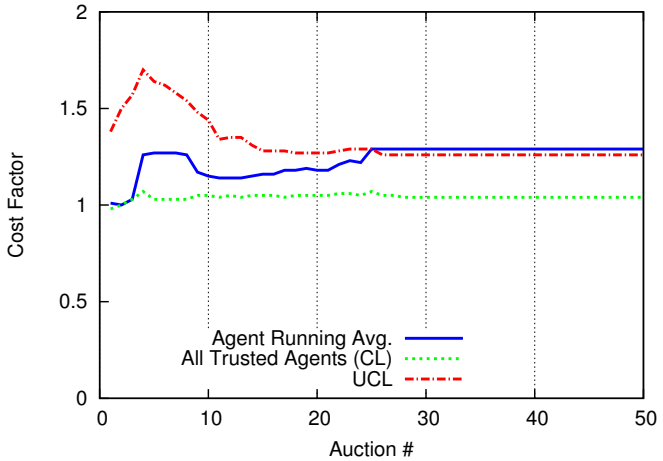

Fig. 8. Detecting when the performance of team members deteriorates over time. The control chart shown is for one of the poorly performing agents.

and is applied to the robot's task performance to simulate error in estimation and execution. As a result, the robot continually underestimates the costs for performing tasks. Each robot on the team observes the tasks completion times for tasks that others have completed on their behalf, as described in Section III-E. In this case, the cost factor for the *poor performers* exceeds the upper control limit after about 10 auctions have completed. An example of the control chart for one of the *poor performers* in the experiment is shown in Fig. 6.

The corresponding trust model values for this experiment are shown in Fig. 7. The model shows the convergence towards trusted values (above 0.5) for the *good performer* type robots and to untrusted values (below 0.5) for the *poor performer* robots, as the beta trust model is updated with feedback from the process as described in Fig. 3. This result also shows that by incorporating values from other trusted agents, as described in Section III-D, the trust values are further increased for the *good performer* robots.

2) Performance decreasing over time: In this experiment, all robots start out as *good performers*, but in 2 out of 6 robots, the performance deteriorates over time. To simulate deteriorating performance, the cost factor of the 2 deteriorating robots is drawn from the same distribution as in the previous experiment. However, while the initial values are $\mu = 1$ and $\sigma = 0.1$, the μ and σ values are increased by small values at each time step to simulate a gradual deterioration in performance. The process monitoring algorithm is able to detect the deterioration after about 25 auctions have passed when the cost factor reaches the UCL value. An example of the control chart for one of the *poor performers* in the experiment is shown in Fig. 8.

The ability for robots to detect when the performance of team members begins to deteriorate would allow for a local approach to addressing the problem. For instance, the robots themselves could include performance characteristics into their decision process or choose to only assign the most important tasks to the better performers on the team.

V. Conclusion

This paper presents a method for recognizing which robot team members are performing tasks as expected through the use of control charts. The above experiments showed that a quality control mechanism can be effective for detecting poorly-performing team members in a distributed task assignment domain. This may prove useful in situations in which multi-robot teams are dynamically formed and not all team members are likely to estimate costs correctly or when cost functions change over time. Although this paper considered auction based task allocation, this approach for using control charts to allow for robots to monitor team members could be applied more generally.

The use of a trust model in combination with control charts allows for robots to reason over the model and to share it with team members. The model can also easily be extended to include multiple dimensions of trust.

Future work will consider additional learning mechanisms relevant to task performance. This is related to the problem of determining how to recognize when tasks that were assigned to another agent were not only completed according to the initial cost estimate, but completed within stated quality parameters. In addition, we would like to explore how performance data can be used to affect either the task assignment function or to perform robot team member selection.

Acknowledgment

This work was funded internally by the Georgia Tech Research Institute. We wish to thank the anonymous reviewers for their helpful feedback.

References

[1] L. E. Parker, "Reliability and fault tolerance in collective robot systems," in *Handbook on Collective Robotics*, S. Kernbach, Ed. Pan Stanford Publishing, 2012.

[2] C. Burnett, T. J. Norman, and K. Sycara, "Trust decision-making in multi-agent systems," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*. AAAI Press, 2011, pp. 115–120.

[3] C. Pippin and H. Christensen, "A Bayesian formulation for auction-based task allocation in heterogeneous multi-agent teams," in *Proceedings of the SPIE 8047, 804710*, 2011.

[4] E. Martinson and R. Arkin, "Learning to role-switch in multi-robot systems," in *Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan,*, vol. 2, Sept. 2003, pp. 2727 – 2734.

[5] J. S. Baras, T. Jiang, and P. Purkayastha, "Constrained coalitional games and networks of autonomous agents," in *Third International Symposium on Communications, Control and Signal Processing*, March 2008, pp. 972–979.

[6] D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems: A tutorial," *Interfaces*, vol. 20, no. 4, pp. 133–149, 1990.

[7] NIST/SEMATECH e-Handbook of Statistical Methods.

[8] A. De Vries and J. K. Reneau, "Application of statistical process control charts to monitor changes in animal production systems," *Journal of Animal Science*, vol. 88, no. 13 electronic suppl, pp. E11–E24, 2010.

[9] W. Zhou, T. G. Habetler, and R. G. Harley, "Bearing fault detection via stator current noise cancellation and statistical control," in *IEEE Transactions on Industrial Electronics*, vol. 55, no. 12, Dec. 2008, pp. 4260 –4269.

[10] W. H. Woodall, "The use of control charts in health-care and public-health surveillance," *Journal of Quality Technology*, vol. 38, no. 2, pp. 89–104, 2006.

[11] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multi-robot cooperation," in *IEEE Transactions on Robotics and Automation*, vol. 14, 1998, pp. 220–240.

[12] I. Matei, J. S. Baras, and T. Jiang, "A composite trust model and its application to collaborative distributed information fusion," in *12th International Conference on Information Fusion*, July 2009, pp. 1950–1957.

[13] E. Jones, M. B. Dias, and A. Stentz, "Learning-enhanced market-based task allocation for disaster response," in *International Conference on Intelligent Robots and Systems (IROS)*, October 2007.

[14] A. Ekici, P. Keskinocak, and S. Koenig, "Multi-robot routing with linear decreasing rewards over time," in *Proceedings of the IEEE international conference on Robotics and Automation (ICRA)*, 2009, pp. 3944–3949.

[15] 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*, vol. 12, 2006.

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