Blockchain-Based Mechanism for Robotic Cooperation Through Incentives: Prototype Application in Warehouse Automation

Jonathan Grey
DePaul University
Chicago, IL, USA
Email: jgrey5302@gmail.com

Oshani Seneviratne
Rensselaer Polytechnic Institute
Troy, NY, USA
Email: senevo@rpi.edu

Isuru Godage
DePaul University
Chicago, IL, USA
Email: igodage@depaul.edu

***Abstract*—The use of blockchain in cyber-physical systems, such as robotics, is an area with immense potential to address many shortcomings in robotic coordination and control. In traditional swarm robotic applications, where homogeneous robots are utilized, it is possible to replace a robot if it malfunctions, and it can be assumed that all robots are interchangeable. However, in many real-world applications spanning from search and rescue missions to future household robotic appliances, heterogeneous robots will need to work together with the other robots and human agents to achieve specific tasks. Nevertheless, no such system exists. Therefore, we propose a system that utilizes a token economy for robotic agents that makes agents responsive to token acquisition as an incentive for collaboration in achieving a given task. The economy enables the system to self-govern, even under Byzantine and adversarial settings. We further incorporate a novel subcontracting framework within a blockchain environment to allow the robotic agents to efficiently and cost-effectively perform complex jobs requiring multiple agents with various capabilities. We conducted a thorough evaluation of the system in a prototype warehouse application scenario, and the results are promising.**

***Index Terms*—Smart contracts, Applications and services based on blockchain, Blockchain in cyber-physical systems, Multi-agent systems**

1. Introduction

Ever since Unimation Inc. produced its first industrial robot in 1956 [1], there have been innovations in the robots in warehouses and industrial robotics space. The first robots were large robotic arms that could move according to some pre-set programming. However, in the past couple of decades, robotics engineers have worked hard to combine new technological developments, like Artificial Intelligence (AI) and the Internet of Things (IoT), with automated robotic technology. Innovative companies, such as Kiva Systems that Amazon later acquired, have provided innovations in warehouses with highly effective automation [2]. Furthermore, anticipating upcoming semi- and fully- autonomous robots, it is essential to follow a scalable and secure global standard for communication agents (swarming robots and human agents) for seamless and efficient collaboration efforts. Then there is the problem of command architecture, i.e., who takes priority and command, mainly when the robots belong to mutually distrusting parties.

In this paper, we illustrate robotic coordination using smart contracts on an Ethereum sidenet. Our system is capable of using agents of heterogeneous capabilities in order to complete tasks. Autonomous agents similar to Autonomous Mobile Robots (AMRs) [3] can carry payloads from a particular location to another in a warehouse application, and smart contracts will facilitate the robotic interactions. The swarm can perform a relatively complex task without a single individual component agent capable of completing the task independently. The task is a sorting task motivated by automated warehouse which involves moving boxes out of a matrix of boxes of various colors into groups of similar colors while avoiding the black boxes. Therefore, the system's extreme flexibility for varied tasks leads to a new class of decentralized applications (dapps) for robotics with vast practical potential in many automation applications. The resulting robust movement infrastructure leverages the multi-agent architecture upon which all high-level decisions are made in real-time, such as deciding which of objects should be fetched, which of the robots should pick up the object, and which of the picking stations should be used to 'fulfill' an order modeled as the final destination for the blocks.

2. Related Work

There are several blockchain approaches to solving collective decision-making problems in robotics. Some of the notable works include decision-making programs [4], [5], [6], [7], managing Byzantine robots in a swarms' collective decision-making scenario [8], machine-machine communication in cyber-physical [9], and black box logging [10]. In all of these systems, a new blockchain system is introduced without utilizing an existing community-adopted platform like Ethereum, and no economic incentive is proposed to motivate the autonomous robotic agents. Fernandes et al. [11] discuss the integration of multiple possible interfaces with multiple data formats used in robotics with a

particular emphasis on using smart contracts and AI to improve the performance of distributed robotic systems using tezos technology for robot event management in industrial applications [12]. Blockchain technology for robot swarms on shared knowledge and reputation management system for joint estimation is introduced in [13]. Shared knowledge is a critical feature in our system, but we have utilized a more scalable method for reputation management. Work has been performed with trust systems on the blockchain as protection against Byzantine actors by Cameron et al. [14]. They use smart contracts to establish which accounts are Byzantine by re-centralizing the identification through the contract itself. Cardenas et al [15] explore an interaction model that enables a robot to engage in human-like financial transactions and enter into agreements with a human counterpart. Grey et al. introduced *Swarm Contracts* to solve limitations in centralized robotic planning applications [16] with adjudication and rewards. We have adopted a similar model in our work to compensate the robotic agents. However, in our work, we cater to heterogeneous agents with varying behaviors that provide a good approximation for real-world applications.

3. Contract Framework

We designed our system with a warehouse application in mind, which informed many of the design decisions for agents within the framework. We have outlined below the definitions and the smart contracts-based "Robotic Cooperative Environment" that is depicted in Figure 1.

3.1. Definitions

The smart contract system is based fundamentally on the operation of blockchain devices, including blockchain robots that interact with the application. These nodes can access information that is located on the blockchain. Some of these nodes interact with the physical world using robotic components and control software. These nodes interact with the contract and its data through the various function services it contains. The data defines and enables work to be performed with respect to the function services the contract offers. The nodes and services provide the basis for the business agents: *oracles*, *workers*, *chiefs*, and *chargers*. These business agents, using their nodes, interact with the contract to further their profit-seeking interest and provide services to other agents in the system. Agents act in their perceived interest by choosing a strategy, either fair or adversarial. Each strategy has the potential to earn money, but the actions of other agents determine which strategy is ideal. The specific agents in the system are as follows:

Chargers initialize requests as well as receive money as the cost of worker movement. Within the warehouse use-case, the *charger* represents a customer who desires something from the warehouse, or in the case of the simulation, two customers - one for each color of payload. There is no significance to the colors beyond adding some complexity to the system.

Workers are the agents that perform the work in the environment. Specifically, we have two types of workers: *cowboy* and *driver* workers. They have an identical purpose: moving an object from where it is currently located to somewhere else, but their difference lies where they are allowed to move (See Figure 3). Despite looking identical, the two subtypes of workers have different roles.

Cowboys accept contracts because they are uniquely able to navigate the warehouse and move the appropriate payload to an empty and reachable place. These *workers* are incapable of moving the payload to the final position, and they can only bring boxes out of their starting positions in the warehouse and move them to intermediate positions. The *driver*, on the other hand, can only put boxes in the appropriate piles from the intermediate positions and cannot enter the warehouse. This design decision has several implications: first, certain worker types have different considerations when deciding to accept or not accept contracts, and second, jobs can be made more complex to require the skills of both types of robots. Because of the blockchain, workers can be considered to work at-will and voluntarily.

Chiefs break down large requests into smart contracts a *worker* can understand and use. The *chief* represents a typical warehouse management software with knowledge of both the warehouse area itself and the orders being fulfilled. The chief interacts with the blockchain by issuing contracts which are voluntarily accepted by workers.

Oracles evaluate the completion of contracts. *Chargers*, *chiefs*, and *oracles* exist only as accounts and have no presence in the physics simulation. *Oracles*, in the context of the warehouse application, represent management and assurance.

Subcontracting is when an agent who has accepted a smart contract issues a contract to similar ends for the contract the agent accepted. The *subcontract* refers to the smart contract issued by the *cowboy worker* and accepted by the *driver worker*. The *supercontract* refers to the smart contract issued by the *chief* and accepted by the *cowboy worker*. The subcontracting process allows for the splitting of contract work, which is impossible for the accepting agent to perform, yet still possible with a swarm. The *charger-chief* relationship already constituted a facsimile of a subcontracting system in [16], but the procedure in this simulation between *cowboys* and *workers* used a smart subcontract on the blockchain to properly subcontract the work.

Blockchain Devices are a representation of a device that is connected to the blockchain, and therefore can interact with the blockchain and by extension, all other devices connected to the blockchain. Generally speaking, all agents are virtually run in blockchain devices. *Blockchain robots* are a subset of blockchain devices which have a relevant physical presence in the simulated environment. Only workers which intend to work need to have a robot.

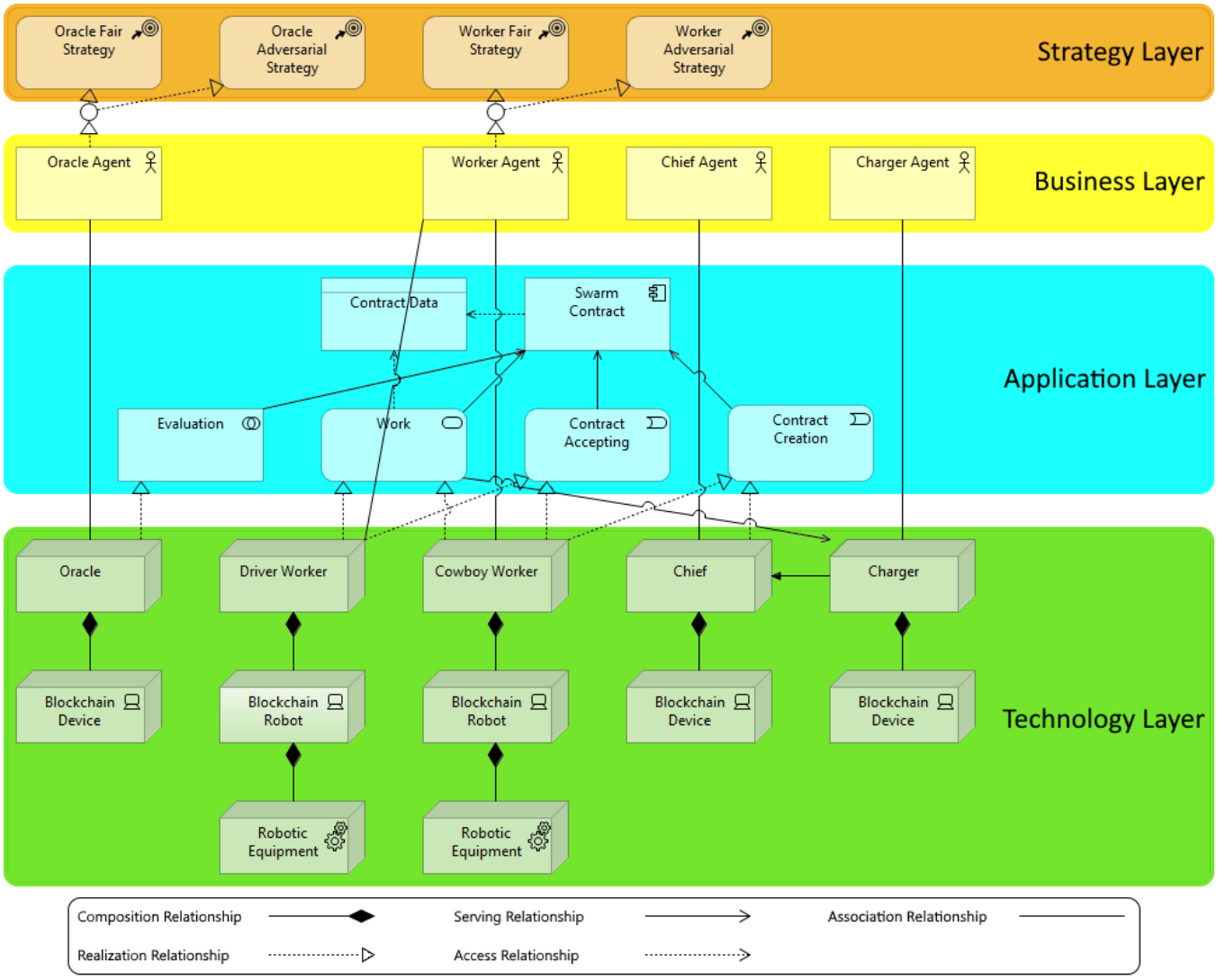

Figure 1. The Architecture of the Robotic Cooperative Environment

3.2. Process

We illustrate the process for smart contract evaluation using a warehouse use case. There is a tremendous legal and reputational burden for not correctly delivering a package, and many delivery businesses have started taking pictures of packages on doorsteps to create a body of evidence that items were indeed delivered to a location. Existing delivery businesses have internal or external contract evaluation when a dispute arises as to the delivery results. Our system provides a decentralized, mutually trusted contract evaluation service specified beforehand using smart contracts.

First, the *charger* issues a request for the final state of the simulation. This request details which payloads should be moved to specific positions. The request is comprised of ten discrete parts, equal to the number of payloads, each of which details a specific payload to be moved to a specific location. The *chief* receives this request and divides it into discrete contracts. Each contract is worth 2Ξ (Ξ is the unit used for tokens). Contracts are then available on the blockchain to be accepted by *workers*. After the *cowboy worker* has delivered the payload from the warehouse to the staging area, it issues a subcontract against the initial contract accepted; the value of the subcontract is 1Ξ. This subcontract contains nearly identical data to the original accepted contract but is slightly different. The target position is the same as the supercontract, but the remainder of the data is flagged to indicate that it can be moved by a *driver worker*. After the length of the contract has passed, the contracts are evaluated. The subcontract has a shorter length and is evaluated first in all cases because there might be potential attacks should it be evaluated after the supercontract. Such an attack would consist of making the job incomplete. Therefore, early subcontract evaluation is a valuable safety feature to ensure successful task completion by guaranteeing the incentives of the two

parties (i.e., the sub- and super-contractors) are aligned for at least the duration of the subcontract. The supercontract is then evaluated, with all directly involved parties noting the evaluation results for their trust data. Trust increases when the *oracles* act positively towards an agent. The simulation then iterates, with all parties saving their data about trust but the simulation itself resetting all payloads and *workers* to their original positions.

4. Evaluation Framework

Evaluative tests were performed in a realistic virtual environment, which allowed for ease of iterative testing autonomously while still maintaining some of the complexity of a real-world environment, such as friction and minor inaccuracies. The virtual environment was designed so that it could easily be transferred to a real-world environment with limited refactoring. The simulation was built principally in the pybullet framework [17], a python wrapper for the Bullet Physics engine, and the Ganache [18] test net, which is an Ethereum local test net and blockchain testing environment.

The simulation environment has a flat, planar space that represents the terrain. There are a specific amount of colored payloads in a grid. These payloads (represented as boxes in the simulation environment) can be black, red, or cyan, which allows for identification and sorting tasks (Figure 2-1).

The robots in the simulation take the form of wheeled vehicles. These vehicles have a box-shaped manipulator device on the front (Figure 2-2). The manipulator device is most similar to a lasso both in topology and in function. The lasso is a stable set of barriers that prevent the box from sliding in a different direction when the vehicle moves. The robots can use this manipulator device to capture a box and move it to a different location (Figure 2-3). The lasso control mechanism is similar in operation to a forklift. The lasso is held in an elevated position until needed and is dropped vertically into position. The lasso is square with a side length approximately twice the diameter of the payloads. If there were slight inaccuracies in the virtual robot's reckoning of location, it would still allow for successful payload captures. All *workers* can capture a box by lowering its lasso around it. The robots utilize a simplified dynamic path-planning algorithm. The algorithm would be best described as a greedy object avoidance algorithm, i.e., the robot will move in a direction that minimizes the distance between itself and its goal point, but only on lines in the cardinal directions at meter intervals. At each meter-interval intersection, it computes the next point it should go to. If another robot or a payload is at the best point, it chooses the next optimal point.

Within the warehouse application, the matrix of payloads is a representation of the aisles and stacks of items in a warehouse (Figure 3-1). Payloads have different colors, which represent relevance. Black items physically exist in the environment, but are not useful to the completion of a job because, within the use case, a customer would not want every item within the warehouse. Cyan and red items are useful to the completion of jobs, and are distinguishable from each other. These bright colors represent relevance, while the difference between the colors represents two distinct but coinciding jobs, which might be likened to two separate orders by separate customers from the warehouse (the final configuration shown in Figure 3-3).

The two different types of robots represent the two legs of the journey an item takes from the warehouse, first taken out of the warehouse to a staging area, then shipped the "last mile" to its final destination (Figure 3-2). In this way, the *cowboy worker* robot might be likened to warehousing robots (or even human-controlled machines) which identify and move essential items from their ordered location to a loading bay. The *driver worker* robot might be likened to a delivery robot (or again a human-controlled machine) which brings an item to the place it is wanted.

Non-physical agents have a presence within the virtual environment, but this is limited to observation without direct influence. Both *oracles* and *chiefs* may observe the environment and determine the position of payloads and *workers*. Their impact on the environment is limited to what is necessary in order to perform their jobs. They may be likened to a UAV or a remote monitoring camera network system within the use-case of the warehouse.

This system was subjected to two tests as outlined in Section 5. The first test was designed to portray an ideal or equilibrium system to ensure that the system can work without the distraction of potential adversaries. The second test was designed to determine the system's resilience and measure the system's ability to prevent attacks and recover from attacks.

5. Results

5.1. Evaluation Criteria

The goals of the experiment were to prove firstly that the contract was applicable in different situations, secondly that the contract is applicable to heterogeneous robots, and finally that the *workers* are capable of issuing and accepting subcontracts. The simulation proved these hypotheses.

The basic evaluation metric for the system was the *completion rate* of the request. The request is made up of ten sub-units, the ten payloads which need to be moved. Each payload can be evaluated independently, giving a completion rate compared to a perfect completion of the simulation by the *workers*. This measure effectively covers both the natural contracts and the subcontracts because the nature of the simulation dictates that the *workers* complete the contract by issuing and completing a subcontract and must work together to do so.

Completion rate is an appropriate metric because it adequately measures the experimental goals. The qualitative measures are evidenced by completion in general since the proper completion of a contract is evidence of the *proper usage* and *application* of the contract as well as the subcontract. However, the completion rate allows for

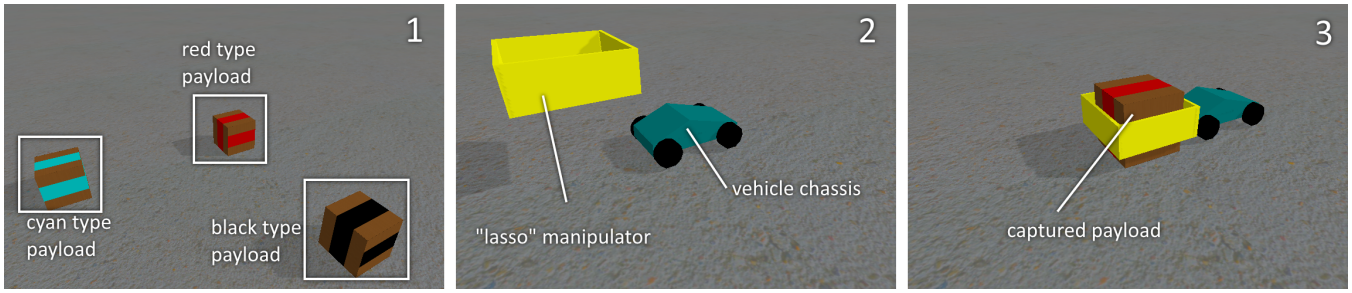

Figure 2. **1) The Three Types of Payloads in the Environment 2) The Vehicle Representing the Robotic Component of a Worker Agent 3) The Vehicle With the Red Payload Captured in Its Manipulator**

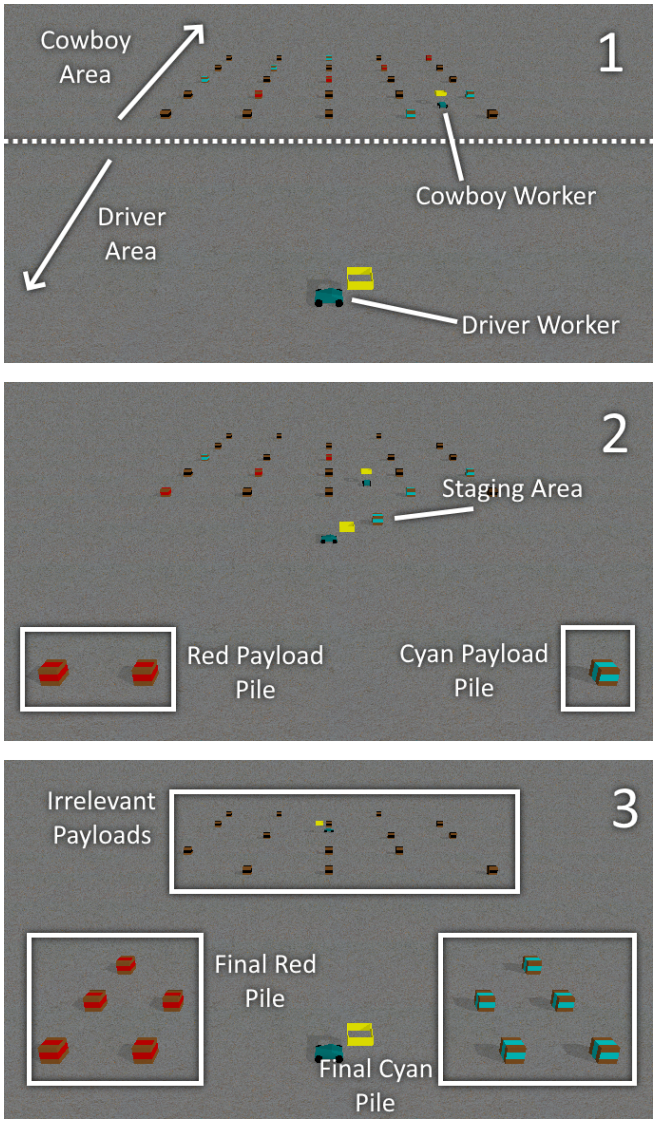

Figure 3. **1) Map of the Environment, 2) The Cowboy Stages a Payload, the Driver Brings It to Its Final Destination, 3) All Payloads Are in Their Final Position.** *(An annotated video that shows the running of the idealized system test can be found at https://bit.ly/Warehouse-Smart-Contract-Demo.)*

a quantitative measure for measuring trust over iterations. It is also an appropriate metric for the warehousing use case. The end goal of a supply chain scenario is to facilitate the transfer of goods from a producer or warehouse to the final customer. Therefore success should only be measured by the achievement of that end. In the simulation, the end goal is the delivery of the payload to the final area.

5.2. Idealized System Test

The idealized system test has no adversarial behavior in order to determine if the subcontracting scheme was appropriate. In an idealized test, the system performed well within the expectations of the design as described below. The contract was highly applicable to the experimental design, with minimal changes added to generalize the contract. The more specific x and y fields were replaced with a *data* field, which was a number that corresponded to an external data storage scheme. In essence, this change made the contract as generic as possible, widening its application to the simulation for utilizing smart contracts in robotic applications.

The subcontracting system worked properly in several ways. Subcontracts were able to be issued and evaluated properly: the generic and anonymous trust system allowed for a worker acting as a *chief* to issue contracts that were accepted without prejudice, and the *oracles* were able to evaluate the completion of each contract on its own. The latter point is especially worthy of discussion. The problem with subcontracting is that contract evaluation does not consider the means of getting the job done, only the end completion or non-completion of the job. Therefore, subcontracts have to be carefully constructed, so they are efficient while accurately reflecting the job—the scheme of constructing the subcontract with the exact target location allowed for an interference-free incentive structure. *Workers* were determined to be able to react to subcontracts and accept them in a manner commensurate with their abilities, which is significant proof of the self-governance and cooperation facilitation of the contracting system.

5.3. Adversarial Tests

Adversarial tests were also performed. An adversarial test includes adversaries which attack the system, creating

TABLE 1. TRIAL COMPOSITION

Trial	Adversarial Workers	Fair Oracles	Owner-biased Oracles	Worker-biased Oracles
1	2	5	2	2
2	2	4	4	4
3	4	5	2	2
4	4	4	4	4

non-ideal conditions which reflect the possibilities outside of a lab environment in order to determine the robustness of the system to withstand and recover from attacks. Each adversarial test was conducted with one *cowboy worker*, one *driver worker*, one fair *chief*, and other agents whose number and type varied with the test. The exact composition of each test can be found in Table 1.

Throughout iterations, two results emerged. First, there was a significant and marked increase in the objective completion percentage of the sorting task. This measure was obtained by measuring the objective position of the payloads after the contracts, observing if payloads were at the appropriate position, and dividing by the number of contracts. The measure sharply increased in all tests. Second, adversarial agents either posted losses or failed to profit like their fair peers. These measures prove that the system is both self-governing and self-improving through the power of incentives. While the system does suffer initial "growing pains" while establishing trust, it eventually becomes resilient to attacks, even though the contract issuers and acceptors do not recognize or track each other's identities, making them effectively anonymous.

The most significant results are from the worker data, shown in Table 2. Since the *workers* carry out the work in the system, and therefore their successes or failures are most relevant. Across all trials, the *cowboy* earned less than the *driver*, but the *adversarial workers* attained losses (on average, losing 5.27Ξ in Trial 1, 0.21Ξ in Trial 2, 4.38Ξ in Trial 3, and 3.69Ξ in Trial 4). The number of transactions is a proxy for the amount of interaction within the system each worker type has, and it can be seen that the broadly the fair types (*cowboy* and *driver*) have incredibly high average transactions when compared against the adversarial type (376 and 243 more, respectively, in Trial 1, 299 and 173 more in Trial 2, 334 and 213 more in Trial 3, and 900 and 588 more in Trial 4).

As for the *oracle* agents, similar results were observed for the number of average transactions as is shown in Table 3. There is a significant difference amongst the adversarial subtypes, however. The difference can be attributed to the relative number of *adversarial workers* between the trials. In trials with a lower number of *adversarial workers*, those *adversarial oracles* that had a bias towards workers performed better and interacted more with the system (43.5 compared to 12 and 28.25 compared to 12). In trials with a higher number of *adversarial workers*, *owner-biased oracles* interacted more with the system (21.5 compared to 8.5 and 17.25 compared to 10.75). The most likely mechanism of action for this correlation was the utility of the respective type of *adversarial oracle* towards the fair contract issuers. In an environment in which there are a relatively large amount of fair workers when compared to *adversarial workers*, *worker-biased oracles* often give evaluations that reflect objective reality, despite their bias, meaning they have a greater utility to fair contract issuers. Similarly, in an environment in which there are a more significant number of *adversarial workers* relative to fair workers, *owner-biased oracles* evaluate findings which are against the *adversarial workers*, again reflecting objective reality despite bias.

It is worth noting that the *cowboy* had significantly lower earnings than the *driver* in all tests (as little as a difference of 12.68Ξ but as much of a difference as 33.86Ξ). There are several reasons for this. The first reason is the nature of the contracts and prices for work. The *oracle* scheme is similar to the *adjudicators* in [16], in which *oracles* take a share of the contract price as payment for services. The *cowboy worker* accepts a larger contract, effectively agreeing to pay more in evaluation fees, but splits the contract price in half when offering a subcontract. The *driver*'s subcontract incurs evaluation fees as well, but they are half as large as the *cowboy*'s relative to the size of the expected gross profit before fees are considered. The second reason for lower earnings is that the *cowboy*, acting as the super-contractor, shoulders a much larger burden of risk. When the *driver* accepts the subcontract, the *driver* must complete the contract under penalty of losing the collateral stipulated in the contract, as is the case in [16]. However, the *cowboy* accepts the super contract, which is incumbent upon the completion of its respective task in addition to the eventual completion of the task by the *driver*. In other words, its payout is dependent on the correct functioning of both itself and another worker.

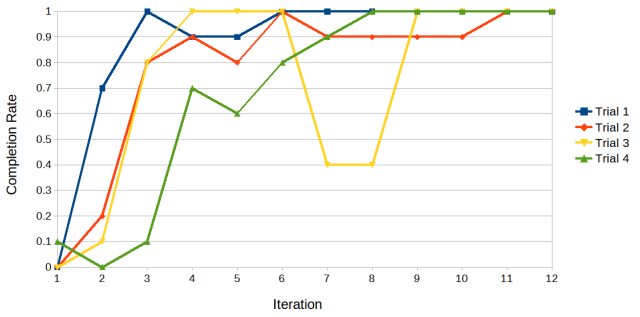

Figure 4. The Completion Rates of Each Trial at Different Iterations

The graph of completion rate versus iteration in Figure 4 shows the expected sharp increase when trust was established. There are several points in the graph which require explanation. The first iteration has a completion rate of zero. The rate is zero because adversaries have no physical form and conduct blockchain-based attacks, which gives them a full license and the opportunity to accept as many contracts as possible in the hopes of finding some that would pay. In Trial 4, it can be seen that the first iteration starts with a completion rate of 0.1, which is purely due to random

TABLE 2. PROFITS AND INTERACTION LEVELS AMONG WORKERS

Worker Type	Trial 1		Trial 2		Trial 3		Trial 4	
	Avg. Profit	Avg. Txs	Avg. Profit	Avg. Txs	Avg. Profit	Avg. Txs	Avg. Profit	Avg. Txs
Cowboy	44.78	383	9.84	312	21.1	339	67.31	908
Driver	57.26	250	43.7	186	33.58	218	99.45	586
Adversarial	-5.27	6.5	-0.21	13	-4.38	5.25	-3.69	7.75

TABLE 3. INTERACTION LEVELS AMONG ORACLES

Oracle Type	Trial 1	Trial 2	Trial 3	Trial 4
	Avg. Txs	Avg. Txs	Avg. Txs	Avg. Txs
Fair	136.8	127	133.8	441.5
Owner-biased	12	12	21.5	17.25
Worker-biased	43.5	28.25	8.5	10.75

chance; it just so happened that fair workers managed to accept the contract first and complete it.

Each of the lines in Figure 4 then shows a marked increase that is due to the lack of will on the part of the adversaries to lose more money. By these iterations, the *chief* and the *cowboy worker*, who issue contracts and select the *oracles*, have developed trust for *oracles* that rule against *adversarial workers*, even as *adversarial workers* have learned which *oracles* can be trusted through the trust development mechanisms described in [16]. The result of this is a precipitous drop in adversarial worker participation. Since there is no adversarial competition for accepting contracts, fair workers (*cowboy* and *driver*) are the only workers left willing to accept contracts, leading to an increase in the overall completion rate.

Each line experiences a drop after the initial spike, which is caused by the differing trust for *owner-biased adversarial oracles*. *Owner-biased oracles* were able to temporarily slip through trust-establishing algorithms despite their bias because their nature caused them to side with owners when evaluating contracts. In the starting iterations, when adversaries accepted contracts, this aligned with objective reality, making them equally as important as a fair oracle. However, when adversarial workers began to distrust all oracles trusted by the chief, owner-biased oracles became useless as fair workers would not accept a job evaluated by an owner-biased oracle.

As soon as these problems are dealt with, the simulation reaches a trust equilibrium. The only subsequent drops happened as a result of robot error. A noteworthy example is Trial 3, in which a stable system had a significant drop, followed by a subsequent return to normalcy, which was determined to be caused by a known but rare flaw in the simulation in which minor errors compounded and made a robot fail to complete a task. However, this trial shows the resiliency of the system as a whole. Even though an adverse environmental event prevented robots from completing the contract and subcontract, the system as a whole was able to continue developing the expected behavior in completing the tasks outlined in the smart contracts.

6. Conclusion and Future Work

IoT and AMRs have become pervasive in many industrial applications. They utilize many autonomous robots that work on command and control settings. However, they have become increasingly complex, where there are multiple types of robots, complex interactions among them, and in many cases, previously unestablished trust relationships due to aspects such as ownership of the autonomous robots. We have designed and implemented a blockchain-based system that can accept agents of varying types and enable their cooperation in complex tasks, making it a successful method for facilitating cooperation in various settings, including robotic applications. Over an iterated simulation, the system developed resilience to adversarial attacks. The contract itself did not change between the simulations except in the data it stored that had no effects on its function and is therefore consistent with the findings in [16]. Therefore, we ascertain that properly structured incentive models and voluntarism create a robotic swarm in a manner superior to a centrally mandated system. The system, furthermore, is fully capable of handling imperfect scenarios, given even simple models for establishing trust. It also creates a valuable system of self-governance among the agents enabling cooperation in the face of adversaries and insurance against their attacks.

Future work and development might include implementing the virtual system in a physical system or creating a more advanced environment for more complex cooperative interactions. Developing the physical system would determine the effectiveness of a natural environment in which novel errors can occur. A more complex system would determine the system's effectiveness against far more risk-prone contracts and prompt the further development of anti-risk measures to protect agents. More developed adversarial attacks could yield insight into the possible additional vulnerabilities and drawbacks of the system. Within the warehouse use-case specifically, future work could be done by making the environment more irregular, perhaps irregular enough to require a greater variety of robots. It would be possible to include special-purpose robots that extend the chain of subcontracts. It also would be possible to include very general-purpose robots, such as a conveyor belt robot, which could invite different economic problems such as common-carrier or free-rider problems. Therefore, we believe the system described in the paper would be a valuable addition to provide an additional trust layer for scenarios requiring dapps for heterogeneous cooperation and robotic automation.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) Grants IIS–1718755, IIS–2008797, CMMI–2048142, and CMMI–2132994.

References

[1] G. O'Regan, "Unimation," in *Pillars of Computing*. Springer, 2015, pp. 219–223.

[2] R. D'Andrea, "Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 4, pp. 638–639, 2012.

[3] A. Liaqat, W. Hutabarat, D. Tiwari, L. Tinkler, D. Harra, B. Morgan, A. Taylor, T. Lu, and A. Tiwari, "Autonomous mobile robots in manufacturing: Highway code development, simulation, and testing," *The International Journal of Advanced Manufacturing Technology*, vol. 104, no. 9, pp. 4617–4628, 2019.

[4] T. T. Nguyen, A. Hatua, and A. H. Sung, "Blockchain approach to solve collective decision making problems for swarm robotics," in *International Congress on Blockchain and Applications*. Springer, 2019, pp. 118–125.

[5] A. Mokhtar, N. Murphy, and J. Bruton, "Blockchain-based multi-robot path planning," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 2019, pp. 584–589.

[6] N. Teslya and A. Smirnov, "Blockchain-based framework for ontology-oriented robots' coalition formation in cyberphysical systems," in *MATEC Web of Conferences*, vol. 161. EDP Sciences, 2018, p. 03018.

[7] E. C. Ferrer, "The blockchain: a new framework for robotic swarm systems," in *Proceedings of the future technologies conference*. Springer, 2018, pp. 1037–1058.

[8] V. Strobel, E. Castelló Ferrer, and M. Dorigo, "Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 541–549.

[9] M. Y. Afanasev, Y. V. Fedosov, A. A. Krylova, and S. A. Shorokhov, "An application of blockchain and smart contracts for machine-to-machine communications in cyber-physical production systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, May 2018, pp. 13–19.

[10] R. White, G. Caiazza, A. Cortesi, Y. Im Cho, and H. I. Christensen, "Black block recorder: Immutable black box logging for robots via blockchain," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3812–3819, 2019.

[11] M. Fernandes and L. A. Alexandre, "Robotchain: Using tezos technology for robot event management," *Ledger*, vol. 4, 2019.

[12] L. Goodman, "Tezos—a self-amending crypto-ledger white paper," *URL: https://www. tezos. com/static/papers/white_paper. pdf*, 2014.

[13] M. Dorigo *et al.*, "Blockchain technology for robot swarms: A shared knowledge and reputation management system for collective estimation," in *Swarm Intelligence: 11th International Conference, ANTS 2018, Rome, Italy, October 29–31, 2018, Proceedings*, vol. 11172. Springer, 2018, p. 425.

[14] A. Cameron, M. Payne, and B. Prela, "Research and implementation of multiple blockchain byzantine secure consensus protocols for robot swarms," *URL: http://agnescameron.info/*, 2018.

[15] I. S. Cardenas and J. H. Kim, "Robot-human agreements and financial transactions enabled by a blockchain and smart contracts," in *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 337–338.

[16] J. Grey, I. Godage, and O. Seneviratne, "Swarm Contracts: Smart Contracts in Robotic Swarms with Varying Agent Behavior," in *Proceedings of the 2020 IEEE Blockchain Conference*. IEEE, 2020.

[17] PyBullet Developers. Bullet Real-Time Physics Simulation. [Online]. Available: https://pybullet.org

[18] ConsenSys Software Inc. 2021. Ganache – One Click Blockchain. [Online]. Available: https://www.trufflesuite.com/ganache