Swarm Contracts: Smart Contracts in Robotic Swarms with Varying Agent Behavior

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

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

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

***Abstract*—Multi-agent robotic systems are becoming pervasive in many real-world applications from search and rescue missions to future household robotic appliances that might need to work together to achieve specific tasks. We propose and implement a collaborative environment for secure communication of robotic agents in a prototype agent system that mimics the interactions between agents of varying behaviors using special-purpose smart contracts titled "Swarm Contracts." This paper describes how Swarm Contracts and blockchain technologies increase the interaction efficacy between agents by providing a more trusted information exchange to reach consensus under trustless conditions, assess agent productivity, allocate plans and tasks to deploy distributed solutions, and carry out joint missions. All these features are encapsulated in Swarm Contracts, making the decentralized applications that use them a viable alternative to centralized command and control applications that are pervasive in multi-agent robotics applications of today. We have evaluated the utility of the developed Swarm Contracts in adversarial settings and report the results that are very promising for future applications of such decentralized heterogeneous robotic agent interactions.**

1. Introduction

The ongoing technological development and globalization are paving the way for sophisticated and dexterous robots. They offer an increasingly effective, fast, and safe approach in many application areas such as search and rescue in humanitarian disasters [1]. In conventional centralized robotic planning systems, different capabilities may not conform to a uniform hardware and software architecture. A system designer could arbitrarily update the domain knowledge with a new constraint disrupting the agreed-upon communication between the agents already deployed in the system. Delayed execution of an agent may lead to a state that is different from another agent's expectation. Fuel/battery updates from robots may dictate which are the viable assets to be deployed. A robot may encounter an obstacle in the real world that the centralized planning system was not aware of, making it impossible to achieve its goal. An operator may add new goals or override the urgency of existing ones, affecting how/whether the plan will be completed by the robotic agents already in the system. Therefore, tracking all the inputs and activities, fed into and generated by the centralized robotic planning process, how they are aggregated or transformed, and pertinent facts in a dynamic execution environment are crucial to establishing trust in the robotic agents in assessing whether they can complete their mission.

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. Surrendering the control of such robots to other parties is also not an option due to security concerns. For instance, UAVs meant to assist ground vehicles could be weaponized should the UAVs' control systems are compromised. The centralized commanding hierarchy also inherits the same security vulnerabilities.

We propose *Swarm Contracts* to solve these limitations in centralized robotic planning applications. Swarm contracts are tailored smart contracts for robots to facilitate complete and uncompromising communication and collaboration to achieve tasks such as search and rescue objectives securely and in a decentralized manner among mutually distrusting and heterogeneous parties.

2. Swarm Contracts

We designed our robotic collaboration protocol using the Swarm Contracts to maximize efficiency, minimize the possibility of exploitation, and guarantee the trustworthiness of all parties involved. With these goals in mind, we borrow two important concepts from the real world in the swarm contract design. (1) We must facilitate trust between several mutually untrustworthy parties, several of which have a view into the state of the system that other parties may not be privy to. Therefore, in addition to the standard distributed ledger primitives, we introduce a "board" of mutually trustworthy other parties, i.e., the *neutral adjudicators* to decide on the completion of a given task with the promise of public accountability. (2) A standard job offer, job interview, and hiring where the employee works at the will of the employer requires the trust of each new employee in the real world.

However, in a decentralized trust-free system, there could be a significant number of new 'employees' (i.e., *workers* in our Swarm Contract) that must guarantee that they will perform the work they undertake. We have modeled this guarantee in the form of a `collateral`. The end goal of the contract is to ensure that all participating agents are motivated using rewards, not orders, unlike in traditional centralized command architectures and that all participating agents have a greater motivation to do the right thing.

2.1. Contract Variables

The smart contract stores public variables that describe the job. The contract's `price` describes the reward for completing the job described by the contract (as well as the amount of cryptocurrency required to post the contract). The contract also stores data that describes the completion conditions of the contract. In this contract, the data is defined as `x` and `y`, which have specific meaning for the smart contract simulation on the Cartesian coordinate system. However, the data definition is unimportant and could be changed if the parameters of use change. The contract also stores the `deadline`, which indicates the point in time when the contract will be evaluated. The purpose of these variables and the reason for their public declaration is to enable agents who would accept the contract to analyze the feasibility, that is, to determine if the contract is possible to be completed given the current state (i.e., physical condition) of the agent and the contract parameters and if the contract is profitable to complete given the price and parameters.

The contract also stores several public variables, as described below, so that both parties can evaluate its ultimate fairness. The contract requires a list of `adjudicators` in order to function, which is public, so that both the adjudicators' presence before the contract is accepted and their performance in the judgment of the contract can be evaluated by all agents who see the contract, possibly informing an agent's willingness to accept the contract and an agent's future decisions to trust or not to trust an adjudicator. Similarly, the contract also stores public information on the judged `status` of the contract, viewable by all parties to determine the truthfulness of adjudicators' judgment and their overall accountability. Finally, the contract `acceptor` address is also publicly viewable, primarily so that other agents can know the contract was accepted.

2.2. Adjudication

The adjudication is the process by which it is deemed that the workers have completed the work, and they are compensated appropriately. In our smart contract, we define the number of adjudicators as three, as three adjudicators retain the relative simplicity of one or two adjudicators. Three adjudicators also have, to some extent, the advantage of a significant number of adjudicators such as relative incorruptibility, and the possibility of a numeric majority and the ability to establish trustworthiness. Theoretically, as the number of adjudicators increases, the more costly it would be to corrupt all of them. As the number of adjudicators increases, the more costly for the contract owner to provide fair compensation for their services, gas prices for storing their verdicts notwithstanding. Also, depending on the cost of each adjudicator, there is significant potential for loss prevention on average.

2.3. Functions

The job described by the contract is accepted by a worker rather than assigned by a hierarchical superior. The ***constructor*** declares and sets every variable except the contract `acceptor` to facilitate this. The constructor provides various of checks to ensure that the contract is valid and harbors no inherent and apparent conflicts of interest. To this end, the contract requires that (1) the given price reflects the actual account value as paid by the contract creator, (2) the selected adjudicators are not null, (3) the selected adjudicators are not identical to the contract creator, and (4) the selected adjudicators are not identical to each other.

The ***accept function*** allows agents to accept the contract at will. In a legalistic sense, an account that calls the contract's accept function assumes responsibility for carrying out the job defined in the `data` and `deadline` variables. The function also ensures that conflicts of interest do not occur, prohibiting the already-selected adjudicators from accepting the contract. The accept function is defined as `payable` and requires a value sent to the contract, which is at least half of the price of the contract. To ensure the agent who accepts the contract has an incentive to work, the upfront value sent to the contract works as *collateral* against the completion of the contract. If the contract `acceptor` has a stake in the contract, it would be against its best interest to accept contracts which it cannot or does not intend to complete, creating a safer variant of the standard "half now, half later" form of business dealing which would be too easy to exploit in the at-will employment system. The specific proportion of one-half was selected because it ensures that (1) the worker has a proportional stake in the completion of the labor, (2) the worker has a smaller stake in the completion of the labor than the owner, which gives the worker less of a risk than the owner, (3) the worker has a significant stake which gives the worker enough risk to promote caution and security concerning the contract.

The ***revoke function*** allows the contract's creator to reclaim all of the funds contained in the contract if the deadline of the contract has expired, and the accept function of the contract has not been called. The rest of the contract's design would mean that any funds the contract possesses when the aforementioned conditions are met would be inaccessible. The selected adjudicators cannot judge in this case and, therefore, cannot drain funds from the contract as payment for their services. Assuming that the contract is created in a marketplace with a sufficient number of workers and the contract has a fair price and is possible to complete before its deadline, the revoke function will not ordinarily be called.

The adjudicate function requires that the current block time is beyond the `deadline`, and the accept function of the contract had been successfully called. Any account may successfully call the adjudicate function, but for the sake of efficiency, the check for a valid adjudicator happens

with the check for a valid adjudication. This means that only adjudicators will be able to judge the contract and receive rewards. Each adjudicator is only allowed to call the adjudicate function once. Following the setting of the result of the adjudication, the contract transfers a sum to the adjudicator equal to a preset proportion of the original price of the contract. Once the last judgment has been made, the contract ends and sends its entire value to the party that the adjudicators decided to have completed the work by the majority vote. This process will occur only on the last call of the adjudicate function, regardless of any previously established majority, to allow each adjudicator to work and claim the payment for rendering judgment.

2.4. Usage

When the constructor of the *Swarm Contract* is called, it is put on the blockchain. Since the blockchain is public knowledge, agents willing to accept contracts can detect them as they appear. After an agent has decided to accept a specific contract, the agent calls the ***accept function***. Adjudicators also detect contracts on the blockchain, since they have a financial incentive to do so. Once the deadline has been reached, relative to the block time that is theoretically uniform, the contract is adjudicated, and cryptotokens are dispensed to the determined winner. The system's fundamental requirements are that each agent knows the blockchain and the contract's design, but no more requirements are directly necessary.

The protocol uses collateral from the workers who accept the jobs to reduce the contract owner's risk. The same parameters and risks exist for the contract acceptor and are ameliorated in the same way by trusting the majority of the adjudicators. Even a small shared network of trusted adjudicators could facilitate every interaction between entirely trust-less agents. Such a feature severely reduces the future risk of losing tokens through adverse interactions with untrustworthy agents by using trial and error to determine which agents can be trusted.

2.5. Extensibility

The Swarm Contract is modifiable for the particular needs of an application. The job description parameters could be changed (even autonomously) as long as agents who would accept the contract and the adjudicators selected to judge the contract have full knowledge of the implications of the job description data. The number of adjudicators could be modified to reflect the size of the pool of available adjudicators or any specific needs of a situation, as was discussed previously. The collateral proportion and adjudicator pay could also be modified based on the monetary concerns of a particular situation.

3. Methodology

The validation of the Swarm Contracts is carried out on a virtual physics environment, implemented using PyBullet, a Python wrapper for the Bullet Physics engine (*https://pybullet.org*), with a sidenet private blockchain, implemented using Ethereum (*https://ethereum.org*) and a Ganache (*https://www.trufflesuite.com/ganache*) client. The implementation of the simulation was created entirely in Python. During its implementation, no performance or scalability issues were detected in the simulation. A physics environment was necessary because it provides a set of constraints on the system that are arbitrary but similar to the real world, particularly objects having a location in space and moving to other locations. The constraints of space and time allowed for the conceptual construction of variable costs and the possibility of task completion or failure. The simulation contained various agents, each with their specific role and behavior set. There were three types of agents: *chiefs*, *adjudicators*, and *workers*.

3.1. Agent Types

Chiefs distill a request (e.g., designate a particular geometric formation for workers to form into) into a series of contracts. Chief agents cannot control the price of contracts, nor could they control the deadline of contracts. However, they can control the selection of adjudicators, and this defined their behavioral subtypes (explained in Section 3.2).

Workers participate in a swarm physically (i.e., in the virtual physical environment). Workers are unique in their physical manifestation, and uniquely incurred costs for performing work, symbolically representing paying for energy.

Adjudicators are used specifically for the judgment of the completion of the work accepted by the workers. While it is indeed possible that workers and chiefs can fill the role of an adjudicator in contracts, for practical purposes having separate agents make more sense to reduce the complexity of each agent and to measure better the effectiveness of each agent in a monetary sense.

The charger provides energy to the worker agents (for a price), gives requests for maneuvers for chiefs to organize (for compensation), and generates energy from workers' position, which allows both the chiefs and the workers to profit as de facto employees of the charger, while the charger acts as the single sink of monetary value. The charger profits, too, but with a surplus of abstract goods produced by its de facto employees produced at the cost of the currency.

3.2. Agent Behaviors

All agent behaviors could be categorized as: **fair**, **adversarial**, and **random**. These general categories were designed to stress-test the system. The fair agents made choices that align most with what would be considered "good", "just," or perhaps "pro-social." The adversarial agents made choices that attempt to get something for nothing. The random agents made choices at random with each possible choice having the same probability. Outside of the simulation environment, there could be several reasons for agents to be adversarial and collude, such as a criminal profit motive. There are many ways agents can collude, including cryptocurrency or other currency transfers, other contracts, or even an exchange of physical goods. This transfer of value would change the system's results by increasing the profits of adversarial adjudicators at the cost of reducing the profits of adversarial workers and chiefs. However, there would be no commensurate rise in legitimate transactions because trust will be lost with respect to the adversarial adjudicators, as

Agent	Behavior	Description
Chief	Fair	Only assigns adjudicators that have in the past exclusively judged based on reflective reality.
	Adversarial	Only assigns adjudicators that have in the past exclusively judged in favor of the contract owner, i.e. itself.
	Random	Assign any adjudicator to a contract regardless of the adjudicators' past judgments.
Worker	Fair	Only accepts a contract if at least two of the three adjudicators will decide fairly, and will perform work.
	Adversarial	Only accepts a contract if at least two of the three adjudicators will decide in its favor, but will not perform work.
	Random	Accept any contract regardless of adjudicators, and will perform work.
Adjudicator	Fair	Produces judgments which reflect objective reality.
	Adversarial	Worker-biased: Produces judgments always in favor of the worker, regardless of the work done. Owner-biased: Produces judgments always in favor of the contract owner, and may disadvange the worker.
	Random	Produces a judgment of random, with a 50% chance of reflecting objective reality.

TABLE 1. DIFFERENT AGENTS AND BEHAVIORS SUPPORTED BY THE SWARM CONTRACTS

will be shown. Regardless of the extrinsic motivation for the behavior, our system will be identical to a system in which no such motivations exist. That is, the transactions or lack thereof will take place in the same way in both situations. The goal of the system is to show long-term behaviors, and the system will better show long-term behaviors through monetary interactions in a more precise manner when there only things represented are contract-related transactions. It should also be noted that while the non-random non-adjudicator agents have trust methodologies to ensure their behavior style can continue unaltered, these agents only consider trusting adjudicators. At no point does a chief make a note of any workers, although it would make no difference since the contracts are at-will from the worker's side. Similarly, at no point does a worker make a note of any chiefs. Such a lack of interaction implies that there is indeed a trust-free relationship between the workers and the chiefs. The workers behave as if each non-adjudicator agent in a contract had just interacted with the other non-adjudicator agent for the first time. Besides, the only interactions that existed were direct, meaning that workers and chiefs would only notice the behavior of adjudicators they interacted within a contract. While a promotion or denunciation or a gossip model would have hastened the development of trust between agents, it was certainly not necessary nor expected, and so was omitted. Table 1 outlines the various agent types and their behaviors.

3.3. Incentive Analysis for the Swarm

Since currency availability is limited in the system, the system is designed to be cyclical, meaning that money is paid by the charger to the chief, by the chief to the worker, and by the worker to the charger. However, there are several drains on the system, as explained below. Since the system is fully decentralized and non-coercive, at each transaction, there must be some profit incentive to spur action. Trade requires a double coincidence of want, so both parties must profit from a transaction, either in currency or goods. Also, the need for adjudicators adds a voluntary tax on every transaction that would not reenter the economy, further draining resources from the system (gas costs notwithstanding). The cooperative production of goods using hired labor would, in theory, create surplus goods for the owners who also could be sold for a profit. However, by closing the loop, inefficiencies and drains on the system are introduced. The surplus of goods generated by the system is effectively a drain on the system because the system will not measure these profits since they are measured in goods rather than currency. Compared to real-world economies, the marketplace for the work through the smart contract enables swarm behavior to emerge through self-interest. The assumption in a marketplace that every agent acts in its self-interest is equally as powerful as the assumption that every agent acts per mandates from the central authority in a classical centralized swarm. The marketplace enabled through our smart contracts are capable of creating a swarm through self-interest without mandates or pre-determined trust. While there is undoubtedly hierarchy in the system, as evidenced by the act of working in the service of others, the hierarchy is purely a result of voluntarism. It should be noted that rewards are constrained by the innate properties of a distributed system. The contract has a theoretical minimum price. When the gas cost of the adjudicate function, which is the amount of gas to run the function on the blockchain, is multiplied with the system's gas price and is higher than the payment to adjudicators, there would be no incentive to make an adjudication.

3.4. System Operation

The overarching unit of work in the system is called the *request*. These requests originate with the charger and take the form of sets of points forming a geometric shape: either a square, a line, or a triangle. This unit of work differs from the contract-level task, which only defines one point. Therefore, there is a ratio of either three (line, triangle) or four (square) contracts to each request. The request comes with several implications. First, there is a reward of four tokens (Ξ) to the chief for every point in the geometric shape. There is a worker near enough to the point of the geometric shape within an individual tolerance. Second, if there are no relevant contracts associated with a request to simplify the process, while the assigned chief is compensated for the completion of the request, the request is enforced as a mandate by the charger agent. Third, there is a predetermined deadline for the request. The deadline is known to both the assigned chief and the charger. While the request very well could have associated contracts and adjudication, this would require additional complexity for no benefit as any problems emerging from adversarial behavior between the charger(s) and chiefs would be identical to the problems between the chiefs and workers, except on a slightly larger scale and for slightly higher stakes.

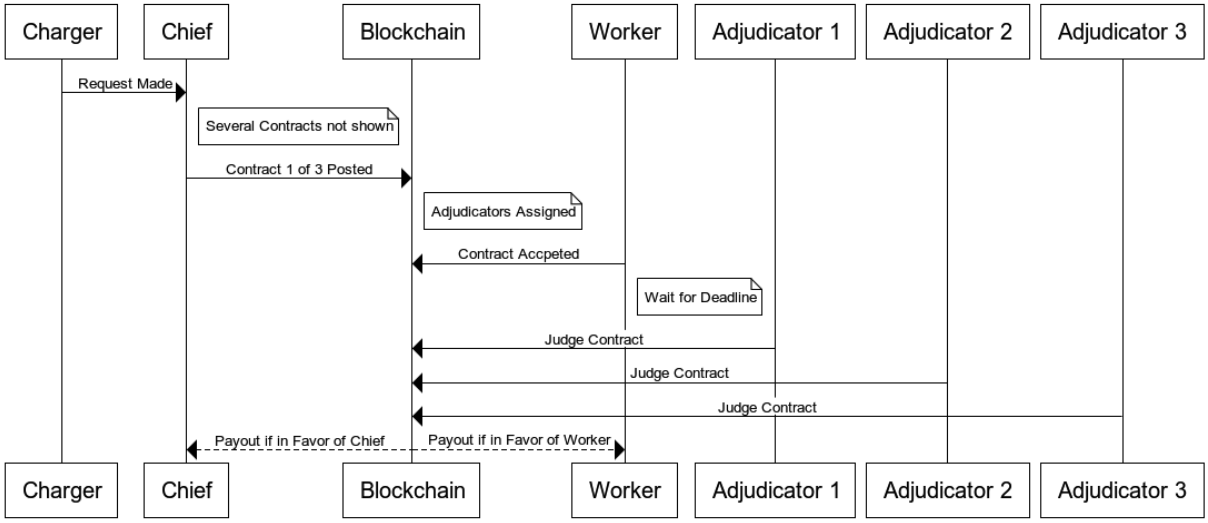

Figure 1. Interactions between the various agents in Swarm Contracts

The contracts, including assigned adjudicators, are posted to the blockchain, where they were detected by all agents involved. Workers then examine the contract and decide to attempt to accept the contract. Only one worker is successful at accepting the contract by the nature of the contract. After the workers accept the contract, they assume responsibility to complete the task within the time allotted. The contract then becomes dormant in the system until the deadline passes. Once the deadline has passed, the adjudicators detect worker positions and judge the contract, which automatically disperses to the worker or the chief based on the judgment. If a smart (non-random) worker or chief participated in the contract, they would judge the adjudicators in turn. Fair workers would remove an adjudicator from its list of acceptable adjudicators if it ruled in favor of the contract owner. Adversarial workers would remove an adjudicator from its list of adjudicators if it ruled in favor of the contract owner since this was what it decided to be in its best interest. It is worth noting that these workers perform the same action for their treatment of adjudicators, but will end up with different lists of acceptable adjudicators. The fair worker would believe that both fair adjudicators and adversarial worker-biased adjudicators are acceptable because the fair worker is ignorant of the adversarial worker-biased adjudicator's other decisions and could not tell the difference between an adversarial worker-biased adjudicator telling the truth by accident and a fair adjudicator telling the truth on purpose. However, every type of adjudicator, except the adversarial worker-biased types, would initially decide against the adversarial worker, so even with the same logic for decisions, the adversarial worker and the fair worker would create different lists of acceptable adjudicators. Chiefs operate in much the same way, except fair chiefs would remove adjudicators that did not reflect reality, even if the adjudicator decided in their favor. Because of the asymmetric nature of the contract, the fair chiefs are required by their nature to select against their short-term self-interest to select for their long-term self-interest. Figure 2 demonstrates the virtual-physical component of the simulation. The black disks represent worker robots. Contracts for a robot to go to a certain point make up requests to form a certain shape. After the contract is completed, workers idle until they accept a new contract. Idling workers can be moved out of the way by other workers. In the third panel of Figure 2, the adversarial workers can be seen in the middle, close to where they started. Adversarial workers perform no work and therefore are always idle because movement costs energy and would be against their best interest.

A video recording of the simulation is available at ***https://swarmcontract.github.io***.

4. Results

The contract was first given a thorough direct adversarial test to ensure that there are no loopholes in the contract, which would allow the draining of funds or any other improper usage. Several requirements were added as a result of this testing. Fundamentally, the behavior-based subtypes of all agents represent predators, prey, and self-defensive categories of agents. As might be assumed, with no protection for the prey, predators take full advantage of the prey.

In initial testing, the worst performance of any agents appears to be the random behavior of workers, who quickly lost money. The cause of this was apparent: adversarial chiefs, with the blessing of adversarial owner-biased adjudicators, take advantage of free labor cashing in on the contract collateral. The fair workers perform reasonably well since they could prevent losses due to trusting predatory adjudicators by not accepting contracts. The adversarial workers appear to be a non-issue, since they tend not to work

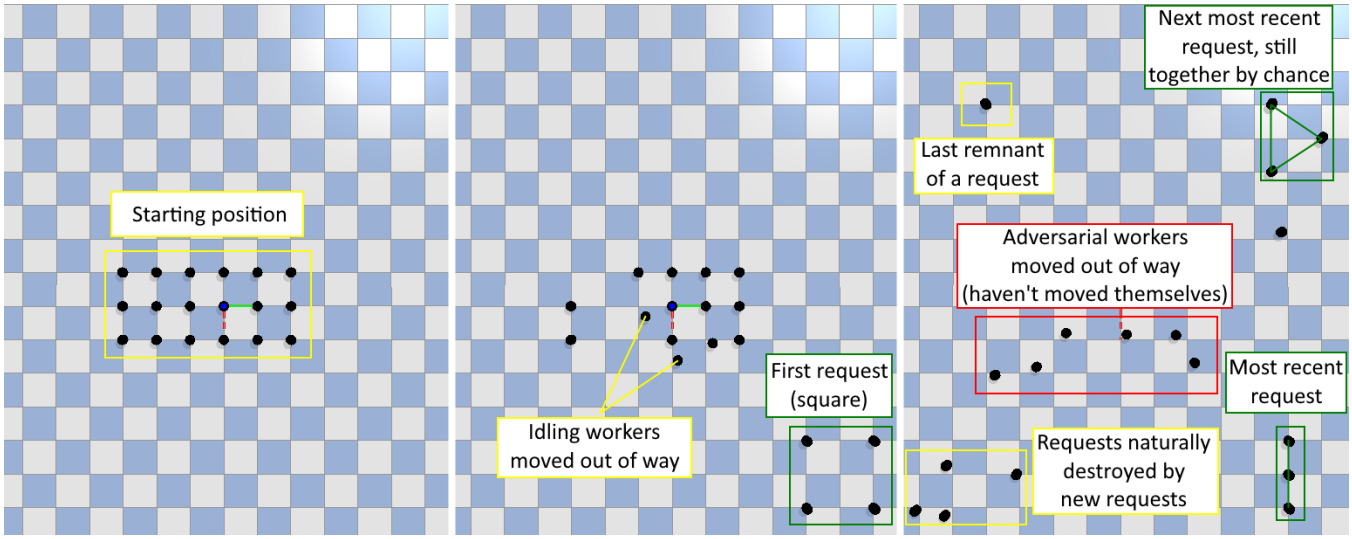

Figure 2. The Swarm Contract in action. (1) The left-most sub-figure depicts the initial state of the world, (2) the middle sub-figure depicts when the first request is received to form a square shape, and (3) the right-most sub-figure depicts several other requests to form a line and a triangle with adversarial workers bumped out the way of the requests.

at all because most contracts were not up to their exacting standards. Chiefs can fully control the adjudicators and are therefore not very susceptible to attacks of this manner, assuming they could learn. Even random chiefs who are unable to learn are able to randomly select adjudicators who may support their interest by accident, frequently enough that adversarial workers could not make money quickly or easily. Adjudicators are unable to lose money by their nature but tended to earn money at different rates. However, the rate of adjudicators earning money reflected a greater extent the decisions of chiefs and the nature of adjudicators, rather than the choices of adjudicators.

Figure 3 shows both types of results from both simulations. The transactions or adjusted transactions measure shows the number of legitimate transactions the agents made. The profit per transaction is related to the number of transactions and indicates an agent's average profitability. These results are a proxy for success within the system. Profit alone could be considered a success but could measure blatant exploitation without long-term cooperation. Transactions alone could be considered a success but could measure victimization. Therefore, success in the system is measured by a relatively high profit and a relatively high amount of transactions. Failure in the system can be characterized by a relatively low amount of profit or transactions. Low profits and a low number of transactions indicate an agent that is not trustworthy enough to work with others, and it is unsuccessful in its exploitation of other agents. High profits but a low number of transactions indicate successful exploitation, which is not a good long-term strategy. Low profits (or negative profits) with a high number of transactions indicate frequent exploitation by an adversarial agent.

4.1. Simulation with all (including random) agents

The first simulation was designed to show the interaction of every agent together. The simulation contained two of each type of chiefs, six of each type of worker, and four of each type of adjudicator. The initial testing had several implications. First, few things can be determined as to the nature and effectiveness of adjudicators. The adjudicators work at the will of the chiefs. The inability to work is far more dependent on the chiefs' design and function, rather than the adjudicators'. Second, the presence of random workers allowed a "criminal" element to proliferate, both the adversarial owner-biased adjudicators who are opportunistic and the adversarial chiefs who are exploitative by nature. Third, simple trustworthiness solutions can go a long way toward loss prevention. Establishing a system of trust and only using trustworthy adjudicators can relatively quickly lead to security with profits. Fourth, the contract's nature means that agents who issue contracts have a much higher opportunity for "criminal" activity than agents who accept contracts. Fifth, exploiting a fair worker's collateral payment is the most profitable vector of "criminality" because it allows a contract creator to profit twice – once from the compensation that incentivizes the transaction in the first place, and from claiming the worker's collateral. As a corollary, exploiting a contract creator's payment while not working is one of the least effective vectors of "criminality" because a contract creator agent can choose whichever adjudicator it prefers; even random selection of adjudicators might shut down this vector purely by accident.

4.2. Simulation with only *'smart'* (non-random) agents

The second simulation was designed to show the establishment and, ultimately, the benefits of trust development. The simulation contained two fair and two adversarial chiefs, six fair and six adversarial workers, and four of each type of adjudicator. The main difference between the second and the first simulations is the absence of random

Figure 3. The scatter plot on the left shows the results of the first simulation that includes all agents. The scatter plot on the right shows the results of the second simulation, which had no random workers or chiefs, and adjusted for the number of transaction due to the disparity caused by adversarial agents.

non-adjudicator agents in the second test. These random agents cannot learn and therefore, cannot establish a system of trust. The second simulation allows for complete trust development, preventing adversarial agents from taking advantage of naivety. The results of the simulation have several implications. First, this simulation conclusively proved that fair agents who establish trust could be the most successful in the environment. The fair agents were able to perform more transactions, which is evidence of interaction with the contracts, and were able to earn more money. Conversely, adversarial agents only were able to act a limited amount of time, which is evidence of their lack of success. As in the first simulation, adversarial workers lost the most as their lack of effectiveness in adversarial behavior was again displayed. The adversarial chiefs were able to have limited effectiveness, but only for a short time, after which their contracts go unfulfilled and have to be revoked. Such behavior causes a disparity in the number of transactions by doubling the number of transactions after worker agents begin to distrust the adjudicators the adversarial chiefs assign. Even when the quantity of transactions is adjusted to account for this fact by setting it equal to the average of the fair chiefs, the amount of money made by the adversarial chiefs remains constant over time, putting them financially behind the fair chiefs. It should also be noted that only the fair adjudicators had a high amount of transactions in the second simulation. While adversarial chiefs will assign adversarial (owner-biased) adjudicators, they will ultimately not perform their job because the contract would invalidate it.

4.3. Achieving Equilibrium

The simulation was designed to evenly distribute tasks and opportunity through random chance, when possible. Each request was given from the charger to chiefs in order, each chief selected random adjudicators from its list, and each worker is delayed by a random amount of time, which ensures a random worker has the first shot at accepting a contract. As might be surmised, the simulation would start at a disequilibrium. Both workers and chiefs of all behaviors would start by acting in a manner closest to entirely at random. As the simulation progressed, the smart agents (non-random non-adjudicators) shortened their lists of acceptable adjudicators as they interacted with them. Eventually, the simulation would reach an equilibrium in which behavior converged on the predicted behavior.

Depending on how the equilibrium is defined and the composition of the simulation, the simulation will reach an equilibrium at a different rate. The equilibrium can be defined strictly, meaning the work is performed consistently well, or the equilibrium can be defined loosely, meaning trust has been established in the system. When the equilibrium is defined strictly, it takes significantly longer for the simulation to reach equilibrium, and it does so at a much slower rate. However, the loose definition of equilibrium allows the simulation to achieve equilibrium extremely quickly as adjudicators can quickly be discovered to be favorable or unfavorable.

5. Related Work

Previous work on blockchain technologies applied to robotic systems addresses some of the challenges, approaches, benefits, and limitations in robotic systems. Several notable works include blockchain approaches to solve collective decision-making problems in swarm robotics [2], multi-robot path planning [3], detecting robotic anomalies [4], ontology-oriented robots' coalition formation in cyber-physical systems [5], workload logging in robot swarm [6], and managing byzantine robots in a swarms'

collective decision-making scenario [7]. In all of these systems, a new blockchain system is introduced, without utilizing an existing community adopted platform, and the heterogeneity and the mutually distrusting nature of the agents are not factored in. Distributed decision-making algorithms have been adopted in many robotic applications, including dynamic task allocation, collective map building, and obstacle avoidance [8]. However, there is no economic incentive in their proposed work to motivate the autonomous robotic agents. SwarmDAG [9] aims to provide an eventually consistent DAG-based distributed ledger across a swarm assuming network partitions are not permanent. That work focuses on the swarm to achieve consensus rather than utilizing the smart contract to coordinate the swarm tasks. Robot-Human agreements and financial transactions enabled by Blockchain and Smart Contracts [10] explores an interaction model that enables a robot to engage in human-like financial transactions and enter into agreements with a human counterpart. We have adopted a similar model in our work to compensate the robotic agents. However, unlike in their work, we cater to heterogeneous agents with varying behaviors that provide a good approximation for the real-world applications. Blockchain technology for robot swarms on shared knowledge and reputation management system for joint estimation is introduced in [11]. Shared knowledge is a critical feature in our system as well, but we have utilized a more scalable method for reputation management with collateral and adjudication by a subset of the agents in the system.

6. Conclusion

We have demonstrated a smart contract that models the swarm behavior of decentralized heterogeneous robotic agents. The contract is robust in the face of adversarial behavior of all the different agents involved, as demonstrated in a simulation environment. The Swarm Contract is theoretically capable of subcontracting, although it was not tested nor implemented in our simulation. Subcontracting, or creating a new contract to hire another agent to complete the job the original agent accepted, could have some applications in the case of a graceful malfunction or failure state that diminishes the agent's ability to complete the job as requested. In such a case, a subcontract could allow the agent to recoup at least some of the collateral's cost from a failed job by hiring out the job to another worker agent. Subcontracting could have several important considerations. The first and most likely common would be job scalping. If a contract creator created a contract for which the price was significantly above the going market rate for a similar job, a scalping agent could accept the contract and subcontract it at the market price to take advantage of the market inefficiency. The second consideration is the emergence of a risk mitigation market, in which chief agents offer high-price contracts for low probability work and the agents which accept offer subcontracts for less money for making attempts. We believe that the cooperation among robotic agents, as explained in this paper, will pave the way for future autonomous robotic systems that need to coordinate to achieve a standard set of tasks, which is a much different architecture to the current state of the art of centralized command and control. The ultimate goal of this work is to extend human capabilities in situations such as search and rescue by utilizing robotic agents with diverse skills and objectives. Blockchain systems may eventually provide an infrastructure for robotic swarm systems to follow specified legal and safety regulations as they become increasingly integrated into human society, which could even result in new interaction models and business models for swarm operation as proposed in this paper.

Acknowledgement

This work supported in part by the National Science Foundation grants IIS-1718755 and and IIS-2008797.

References

[1] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima *et al.*, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[2] 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.

[3] 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.

[4] V. Lopes and L. A. Alexandre, "Detecting robotic anomalies using robotchain," in *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2019, pp. 1–6.

[5] 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.

[6] I. Kalyaev, E. Melnik, and A. Klimenko, "Distributed ledger based workload logging in the robot swarm," in *International Conference on Interactive Collaborative Robotics*. Springer, 2019, pp. 119–128.

[7] 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.

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

[9] J. A. Tran, G. S. Ramachandran, P. M. Shah, C. B. Danilov, R. A. Santiago, and B. Krishnamachari, "Swarmdag: A partition tolerant distributed ledger protocol for swarm robotics," *Ledger*, vol. 4, 2019.

[10] 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.

[11] 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.