Robots Security and Blockchain Swarm

What is Swarm Robotics?
It is an approach to the coordination of multiple robots as a system which consists of large numbers of mostly simple physical robots. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment. This approach emerged in the field of artificial swarm intelligence, as well as the biological studies of insects, ants and other fields in nature, where swarm behaviour occurs.

Robot swarms were assumed to be fault-tolerant by design, due to the large number and redundancy of the robot units. In endogenous fault detection, robots detect faults in themselves; in exogenous fault detection robots detect faults in other robots. So, it could be difficult to detect certain endogenous faults, e.g., a robot might have a broken sensor but only realize it if its sensor readings are compared to its neighbor robots. Therefore, more recently swarm robotics research shifted its focus to exogenous fault detection. A disadvantage of this system is that it can only detect robots that are either completely broken or that report an endogenous error by not flashing their LED anymore: malicious robots cannot be detected nor is exogenous *partial* fault detection possible.

Potential applications for swarm robotics are many. They include tasks that demand miniaturization (nanorobotics, microbotics), like distributed sensing tasks in micro machinery or the human body. One of the most promising uses of swarm robotics is in disaster rescue missions. Swarms of robots of different sizes could be sent to places rescue workers can't reach safely, to detect the presence of life via infra-red sensors. On the other hand, swarm robotics can be suited to tasks that demand cheap designs, for instance mining or agricultural foraging tasks.

One important capability that robot swarms need to have to cooperate effectively is to be able to make collective decisions. Accordingly, collective decision-making is a well-studied subject in the field of swarm robotics. In general, to make a collective decision, robots in a swarm need to share their information and to aggregate this information using a distributed consensus protocol. The prevailing consensus protocol for averaging the values held by the individual entities in the swarm is the linear consensus protocol (LCP). However, this consensus protocol

and most other protocols used in swarm robotics make the unrealistic assumption that all the robots in the swarm work as expected.

The real-world operation will almost result in robots in the swarm that either fail (e.g., due to dust blocking their sensors) or that are malicious actions (e.g., by a hacker who gains control). These failures can damage people, nature, animals, and other robots, making the reliable detection of failures a crucial task. The term is used ‘Byzantine robot’ - It is based on Byzantine fault-tolerance and the *Byzantine Generals Problem* - as an umbrella term to describe robots that show unintended or inconsistent behavior, independent of the underlying cause. A Byzantine robot can appear well-functioning to some parts of the swarm but faulty to others and might arbitrarily change its behavior. An extension of the Linear Consensus Protocol (LCP) capable of managing these Byzantine robots is the weighted-mean-subsequence-reduced (W-MSR) algorithm. While W-MSR's outlier detection limits the influence of Byzantine robots as long as their number is low, it breaks down as soon as their number is high or an attacking robot forges pseudo-identities (Sybil attack).

To pave the way for real-world deployments, *secure* robot swarms must continue to operate effectively in the presence of Byzantine robots, potentially performing Sybil attacks. Peer-to-peer networks are particularly prone to Sybil attacks: without a trusted system, it is easy for a malicious agent to create an unlimited number of new identities and gain a disproportionate amount of power in the swarm. We contend that *blockchain technology* can be used to create such secure robot swarms due to its decentralized nature, resilience, and versatility.

A blockchain is composed of linked blocks containing data consisting of transactions. Each block is divided into two parts: a body and a header. In the body, the transactions of the participants are stored. The header contains metadata and links each block to the hash of a previous block to create a chain of blocks. A copy of the blockchain is stored by each participant in the peer-to-peer network; the peers exchange and update their blockchain information based on a consensus protocol.

Blockchain technology was originally developed for Bitcoin, the first widely successful digital peer-to-peer currency. In the context of Bitcoin, the blockchain presents a tamper-proof financial ledger in a network of mutually untrusting agents without relying on a central authority. The Ethereum framework further

demonstrated that the blockchain cannot only be used for financial transactions but can store snippets of programming code and come to an agreement regarding their outcome. These snippets of programming code are called blockchain-based smart contracts (or *smart contracts* for short). Every node (robot in this article) in the network runs a virtual machine and executes these snippets. We show how smart contracts can provide the infrastructure for implementing secure “meta-controllers” in robot swarms.

Blockchain-based meta-controller is defined as a blockchain-based meta-controller to be a controller that coordinates the swarm at a higher level than the local controllers of the individual robots. To this end, crucial information from the individual robots is securely stored, aggregated, and processed via a smart contract residing on the blockchain. This ensures that information or control commands are based on a consensus in the swarm.

Let’s take a decision-making scenario based on the blockchain approach where the robot swarm moves on a floor covered with black and white tiles and has to determine the relative frequency of the white tiles in an ARGoS environment.

The robots' task is to determine the relative frequency of white tiles in an environment in which the floor is covered with black and white tiles. For each robot, an instance of the Ethereum blockchain software is executed in a separate Docker container and the robots maintain a custom Ethereum blockchain network. Via a blockchain-based smart contract, the sensor readings of the robots are stored and aggregated. When robots are within communication range, they exchange their blockchain information. In contrast to classical approaches, the blockchain is able to mitigate the negative impact of malfunctioning or malicious robots and allows the creation of a *tamper-proof* system, in which the messages of the robots are securely stored.

To interact with a blockchain and store new data, participants create transactions and distribute them among their peers.

A transaction is digitally signed by the sender using a private key. Hence, all transactions can be unambiguously assigned to a digital address (public key) and attackers cannot create transactions under a false digital identity. In most blockchain frameworks, all data is public and can be read by every participant of the network. Still, in blockchains without an access control layer (public blockchains or permissionless blockchains), the real identities of entities (persons, organizations, robots) involved in a transaction can remain unknown since only the public keys are visible. For a transaction to become part of the blockchain, it has to be bundled into a block and added to the end of the chain of blocks. Before being part of a block, transactions are called *unconfirmed* transactions and are disseminated across nodes of the blockchain network. Bitcoin introduced a consensus protocol which allows the participants in the network to agree on which blocks to add and in what order to add them. The consensus protocol used by Bitcoin is called Proof-of-Work (PoW) and was the first protocol to effectively reach decentralized consensus preventing at the same time double-spending (i.e., a situation where the same crypto token is spent twice). PoW requires the participants to solve a computational puzzle in order to add a block to the blockchain; the puzzle consists of finding a hash value below a target value using the bundled transactions and an adjustable nonce value as input to the hash function. The nonce is a number that can be arbitrarily varied in order to change the input to the hash function and, therefore, the result of the hash function. The process of solving this puzzle is called *mining*. The number of hashes a device can compute per second is stated by its hash power. However, via the PoW-based consensus protocol, conflicting blockchain versions can be resolved: whenever a robot has to choose between possible blockchains, the blockchain that required the highest PoW (i.e., the longest blockchain) gets accepted as the true blockchain, while shorter blockchains are discarded. Transactions that were in the discarded blockchains but not in the longest blockchain become unconfirmed transactions again and can be included in later blocks.

Robot swarms could greatly benefit from shared knowledge, for example, for determining whether a consensus has been reached within the swarm, for calculating the mean value of the sensor readings of the single robots, or for determining malfunctioning units. Hence, decisions could be based on a shared view of the world. This would not only possibly simplify several swarm robotics tasks but also enlarge their field of applications facilitating decision processes.

The ARGoS robot simulator is the state-of-the-art research platform to conduct simulations in swarm robotics. Each robot acts as an Ethereum blockchain node, maintaining a custom Ethereum network. In order to connect ARGoS and Ethereum, we developed the ARGoS-Blockchain interface that provides access to the Ethereum nodes for the robots. The interface is intended to facilitate research in blockchain-based robot swarms by allowing Ethereum functions in ARGoS. Docker containers contain all the necessary dependencies to run specific applications and are more lightweight than a virtual machine. In our setup, for each robot, the Ethereum implementation *geth* is executed in a separate Docker container. The simulated robots maintain a *custom* Ethereum network, i.e., a network that is shared among the simulated robots and independent of Ethereum's main network. Different containers can communicate with each other via channels.

That's all for now.
Thank You !