DecentralBot - Final Project Report
EC544 - Spring 2021
Boston University

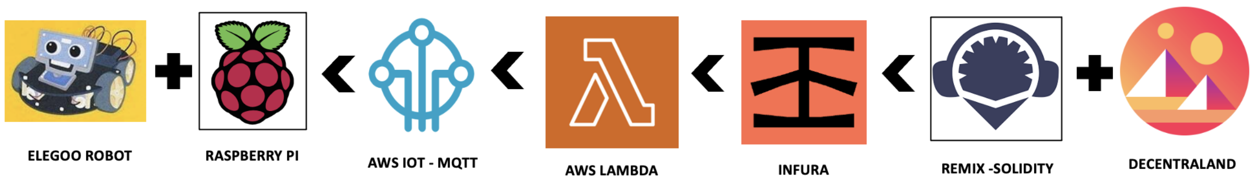
Feng Wang
wang0812@bu.edu

Joseph 'Jack' Locke
lockej@bu.edu

Introduction/Abstract

The intent of this project was to build a robot car that exists at one location in physical space and, though the internet, allow a user in another location to control the robot after paying for the 'service' through a cryptocurrency transaction. Multiple technologies were implemented in the chain of communication to enable the entire process through all seven layers of the OSI model. Our research of some of the existing protocols and services successfully allowed us to reach this goal in a simulated blockchain environment and demonstrate a proof of concept for how in the near future smart contracts will enable utilization of devices in a decentralized architecture.

Technology Stack (the technical report of success)[1]



RaspberryPi (microcontrolers)[2]

The Robot is based on the ELEGOO Project Smart RobotCar v3, originally controlled by an Arduino Uno Microcontroler. For the purposes of this project the car was modified with a 3D printed adapter to accommodate the form factor of a RaspberryPi 4, because of its increased processing ability and built in WiFi connectivity. Many inexpensive and small microcontrollers exist that could have accomplished this task, but the RaspberryPi was selected for its ubiquity and large library of existing resources. The General Purpose Input/Output (GPIO) Pins on the RasPi Control the wheel motors, and a small power converter steps the 7.5V of the ELEGOO Li-Ion battery pack that drives the motors down to 5V to run the RasPi.

A Python Script `Robocar.py` running in the native Raspbian OS controls the GPIO Pins and uses the Amazon Web services (AWS) SDK to subscribe to a MQTT topic called 'RoboCar/Command'. The script parses the messages passed in json format from the MQTT topic and moves forward or backward for 1 second, or turns right or left 90° (.25 seconds) for every single letter command it receives.

MQTT - AWS IoT[3]

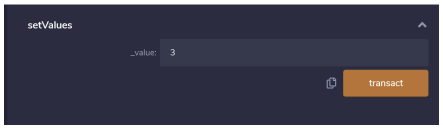
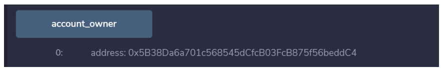
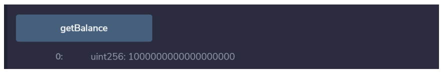
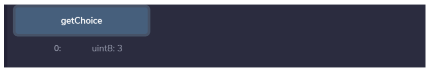
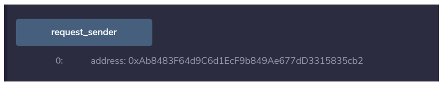
This project uses AWS MQTT service to pass commands to the robot. MQTT was selected for the ease of connectivity to AWS Lambda functions, and shadow capability that would allow an offline robot to synchronize itself with the list of commands which had been previously passed. (Note: no shadow was set up for this project and the robot car must be actively subscribed to receive commands, however shadow technology would be generally suitable for a project like this where a remote user is paying to use the service and command execution must be ensured). Any service that has a certified policy to publish to topics on this AWS endpoint can submit commands to the robot car. In this case the only two services with this authorization are the associated Lambda function described below and a test script used to verify functionality.

[1] All code files referenced and used for this project can be found on the project github. Email Jack Locke at lockej@bu.edu for access.
[2] Website - https://www.raspberrypi.org/
[3] Website - https://aws.amazon.com/console/

Smart Contract Management - Remix (Solidity IDE) - Solidity (v0.8.1)[4]

The smart contract was written using Remix IDE. Compared to other IDEs, Remix has a straightforward interface where all function calls are presented in button form with built-in test accounts, allowing an immediate response. The IDE itself identifies simple syntax errors and a built-in debugger will show individual step breakdown in the code (similar to assembly). This is very helpful in understanding logic errors in the codes. The coding language used is solidity version 0.8.1; while you can select which version of the solidity language to use in Remix, the most recent version was selected.

There are seven functions implemented in this smart contract. The specifications are listed below. This section is related to the blockchain portion of the course. While writing intelligent contracts is not taught in class specifically, the knowledge learned is helpful when understanding the return from the blockchain test net.

Name	Functionality	Restriction	Image in Remix IDE
ReceiveMoney	Allow the smart contract to receive money from any wallets	No restriction	
setValues	Allow the qualified users to set actions for the robot car	Owner of the wallets who send money to the smart contract OR the owner of wallet which deploy the smart contract	
AccountOwner	Show who is the owner of the smart contract	No restriction	 0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
withDrawMoney	Allow the owner of the smart contract	Owner of the smart contract can withdraw money	
getBalance	Show the amount of money store in the smart contract currently	No restriction	 0: uint256: 1000000000000000000
getChoice	Show the movement choice of the car currently	No restriction	 0: uint8: 3
request_sender	Show which wallet sent the current transaction	No restriction	 0: address: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

Infura Blockchain test networks[5]

Once the smart contract was written in solidity , it was deployed into testnet. However, the information was deployed in some unknown location on the testnet, and we were unable to access or retrieve the information. This is where Infura was useful.

The Infura.io API provides methods to access your own snip of Ethereum network over HTTPS and WebSocket. For this assignment, HTTPS was mainly used, but the implementation is similar for both. In order to use Infura, a project was created in the ropsten testnet, allowing the use of free simulated ether. Metamask was then used as the cryptocurrency wallet for interacting with the test Ethereum blockchain. Two test wallets were created each with some amount of simulated ether in both wallets in Metamask. One wallet would represent the owner of the robot car (deployer of the smart contract), and the other will represent the buyer who wants to move the car. In

[4] Website: https://remix.ethereum.org Documentation: https://remix-ide.readthedocs.io/en/latest/index.html https://docs.soliditylang.org/en/v0.8.1/

[5] Infura website: https://infura.io/ documentation: https://infura.io/docs/ethereum/json-rpc/eth-getBalance
Assisting tools: Postman API manager: https://www.postman.com/ Metamask: https://metamask.io/

order to connect these two wallets into our own testnet, it was added to the list of testnets with which the wallet could connect, allowing Metamask to inject the wallets into the Ropsten network when choosing the injected web3 option in remix.

Once the Infura Ropsten testnet had connected with Remix properly, Metamask was used to interact with, and deploy the smart contract. The next step was to retrieve desired information from the Ropsten test net blockchain and pass the information to the robot. Infura API calls were used with the API client Postman, allowing easy API testing. After experimenting with different Infura API json-rpc calls, the eth-getBalance call was selected, requiring the address of the wallet and keyword 'latest' and would return the latest balance of the account. From the perspective of the owner of the smart contract, if the account balance in its wallet increased, it indicated someone has successfully paid the smart contract owner. The information learned in course regarding API calls was useful to assist understanding Infura API and documentations. Knowledge regarding postman learned in the lab was very useful when retrieving information from the Infura Ropsten testnet.

AWS Lambda functions - The two halves of the project are now tied together with an AWS Lambda function. A Lambda function called WalletChecker (and implemented in the `index.js` file) accomplished two events in the chain of technology. First, it verifies the successful payment to its associated ethereum wallet as described above, and second, publishes a message to the 'RoboCar/Command' MQTT topic. The first function makes an API call to the Ropsten test blockchain hosted on the Infura network and checks the balance of the Ethereum wallet. If the value returned is higher than the initial value hard coded in the `'initialval'` variable this indicates a payment has been successfully submitted. Because this is only being implemented as a proof of concept, this mechanism is intentionally extremely simple. If implemented in a production environment more robust code would be included to check and store the initial value in the wallet and ensure the wallet value has increased by a specific amount for a desired action. After verification that the wallet value has increased, the function posts a message to the MQTT topic, `{'message' : 'f' }` indicating the robot should move forward for one second. Similar to the simplicity of the balance verification mechanism, this command is hard coded. The robot is capable of any number of possible commands and in a production environment data in the form of an integer would be queried by the API to indicate which action the robot ought to take, and a corresponding message would be passed instead.

Failures/Further Research

Two small problems were encountered with the setup of the robot itself and the implementation of the code on the Raspberry Pi. First, four successive designs and redesigns of the 3D printed Raspberry Pi adapter board were created and 3D printed and were the initial result of poor measurements, and the optimizations for port accessibility and wiring simplicity. The next problem involved investigating the most reliable and inexpensive solution for power conversion of the 7.5v power supply to the 5v required for the RaspberryPi since too little voltage could cause the RasPi to shutdown unexpectedly, but too much voltage risks burning out the RasPi. A suitable board was found on amazon and incorporated into the design of the 3D printed adapter requiring three more redesigns. Neither of these problems was significant to the technology stack implemented for the project but much practical knowledge was gained through the experience. The MQTT lab in class made the implementation of code for the board fairly straightforward, although understanding the need for different in MQTTClient names took quite a bit of time to debug.

Solidity and ethereum network was initially forign to both members of the team, thus it was easy to step into the wrong direction when investigating methods to execute for various tasks in the project. While biweekly meetings were scheduled with the professor, sometimes the intent of the professor was misunderstood, or the complexity of an approach was outmoded, resulting in many hours researching the 'wrong' topic. This was not an entire waste, since much was learned about AWS API Gateway, Ganache and Truffle, Robot Operating System (ROS) and Decentraland and other topics/technologies that weren't included in the final project. There are three main hiccups for the research regarding ethereum. The first hiccup was after the smart contract is written successfully. We thought in order to make the smart contract interactive for the users and allowing the users to use ERC20 tokens to exchange remote robot movements, we need to use the REST structure in truffle. While we were relatively comfortable with remix at that time, truffle is still fairly foreign and the progress of the project was stuck for a week. Eventually we learned, we didn't need to use truffle, instead, combining remix, Metamask and a web blockchain API call service was sufficient for the project. The second hiccup comes when we were researching web blockchain API calls. Initially, we investigated Amazon managed blockchain service, due to the lack of understanding for the Amazon AWS service structure, we made the mistake of trying to implement the blockchain service on Raspberry Pi, which is unrealistic considering that we will need to download a full node of blockchain on to a device with very limited resources. After correcting this error, we then ran into a VPC connection problem, during which we found out Amazon AWS blockchain was using Hyperledger Fabric which is not a real blockchain network. Thus we switched to infura.io. Infura is much more straightforward than Amazon AWS, but caused us one final issue. We were unable to figure out how to connect the Infura Ropsten testnet to Metamask. After a week and half research, the solution was rather simple, as we found we were just approaching the problem in the wrong direction. While we headed into the wrong direction several times when learning and experimenting with ethereum networks, a lot was learned and we had a deeper understanding regarding the whole ecosystem surrounding blockchain and ethereum.

Possible Improvements

Smart contracts

While some effort was spent when writing the smart contract regarding restricting users access in withdrawing money and setting movements, a lot more could be done to ensure the security of the smart contract. For example limiting user access to the Ropsten testnet, requiring authentication, etc. Further, the smart contract could be refined to separate robots from the contract owner. Currently, the contract owner receives all the ether and controls the robot manually. In a more advanced setting, the robot can have its own wallet and receive the money from other users and could negotiate the smart contract.

AWS Lambda Function

Like the smart contract, the execution of the Lambda function is also manually triggered when we, the designers, know a priori that the balance of a wallet has been updated. Further implementation could set a cron job to check the status of the wallet at regular intervals without the need for manual intervention. Further investigation regarding the fee structure for use of Lambda functions would be needed since every implementation charges for the time the function is processed. While minimal, a looped process calling the function frequently could accumulate expenses quickly,

Decentraland[6]

In the initial proposal we had intended to connect the robot to a UI in the virtual world decentarland. A UI would then execute the smart contract based on user input and pass the command down the technology stack to cause the robot to move. We would also cause a virtual representation of the robot to move on a piece of land for the user to view in response. In the current implementation of our project the transaction with the smart contract is controlled manually since we chose not to deploy the smart contract to decentaland. This was done because of time constraints we ran up against toward the end of the project and the potential real world costs associated with deploying the contract to a live network, especially with the recent spice in cryptocurrency prices across the board around March 2021. We did however spend some time researching the decentraland documentation, building environments in the decentraland SDK and 'Builder' and have a general idea of how we would proceed if we incorporated decentraland into a future version of this project.

Conclusion

While not entirely in the original vision we intended at the beginning of the semester, this project successfully accomplished the intended learning objectives focused on decentralizing technologies. We have demonstrated how a Robot can be paid for by a user through a smart contract and successfully executed the intended tasks within the scope of it's programming from a remote location. All aspects of the project successfully demonstrate the proof of concept behind this work and the number and variety of tasks the robot can accomplish, and the variability of fee structures/cost for executing these tasks is simply a matter of scale and improved code flow. It would not be hard to imagine a job in the future where an employee in a VR rig in their own home could, for example, 'rent' a robot crane (or fleet of robot cranes) halfway across the world, that are then used to unload a shipping container or build skyscrapers. They wouldn't have to travel, own the robot or even have a complex contract with a company and could act as a freelance 'decentralized' agent, at the time and pace of their choosing. Although this robot is a fun toy that just moves when simulated cryptocurrency is moved in a digital bucket, the technologies that enable this physically small accomplishment have quite large implications.

[6] Testnet: https://docs.decentraland.org/development-guide/second-layer/