Cyberphysical Systems with Virtual Reality and Robotics

MS ECE Project Final Paper Summer 2020

Advisor: Professor Babak Kia

August 14, 2020

Kristi Perreault

Abstract

This project explored cyberphysical systems through Virtual Reality (VR), Application Programming Interfaces (APIs), Blockchain, and robotics. To incorporate the physical world, a remote vehicle was built, and to introduce the concept of blockchain, it was “fueled” by the popular cryptocurrency Ether. Leveraging the Decentraland SDK, a basic scene was created to convert Decentraland currency, MANA, and send it to a crypto wallet by selecting a box in the virtual reality space. These components were connected with a lambda function, which monitored the account transactions on the blockchain, and published a message to the robot via the AWS Python SDK commanding it to move forward by the amount of ether it was sent. Although this was not the original approach for the project, it still explored cyberphysical systems and the topics aforementioned. This project intends to serve as the practicum requirement for the ECE Master’s program.

Problem Statement

Virtual reality (VR) is a growing area of interest in the world of engineering and technology. Its applications have spread to scientific discoveries, medicine, community development, video games, and more. Robotics, similarly, has seen huge growth in the last several years, seeing expansion in the automotive industry, hospitality, and even retail stores. Utilizing these types of technologies, this project aims to explore the possibility of a connected world with cyberphysical systems, as it relates to robotics and other industries. Cyberphysical systems are defined as the integrations of computation, networking, and physical processes [1]. It is known as the “intersection” of the physical and the virtual world - a key field of study in today’s ever-evolving technologies.

The goal was to develop a VR application with blockchain transactions that triggered a small rover-like vehicle, and facilitate intuitive interaction between humans and machines. This vehicle was fueled by ether transactions on the blockchain, facilitated by interacting with the VR environment. This project had a heavy focus on the communication stack between the VR environment and the rover and implemented several concepts from the electrical and computer engineering disciplines. These include human-machine interfaces; APIs; blockchain; cyberphysical systems with the interaction between the real-time vehicle and sensors; VR; communication protocols; authentication; and robotics. Similar work has been completed with VR and communication protocols in EC544 Computer Engineering in a Connected World, robotics in a 2013 undergraduate senior design project building out an autonomous vehicle, and working with APIs as a full time software engineer.

Prior Work

The concept of cyberphysical systems has been around for some time, and includes the integration between virtual and physical realms. This can span anywhere from robotics to

software engineering to embedded systems. This project was motivated and informed by Professor Babak Kia's 2013 senior design project, an Augmented Reality Monopoly game, which involved an iOS application, network server, and hardware tile - a cyberphysical system with a software component, communication stack, and hardware component, as is the goal here [2]. In addition, previous literature [3,4,5,6] details the importance, the applications, and the future of these cyberphysical systems, including the advances that have been made in the aforementioned industries. One of these papers [3] talks in-depth about cyberphysical systems as they relate to models, including deterministic models, timing, control, architectures, and so on, with concluding remarks on how these models presented may improve the industry. Researchers in the Electrical Engineering and Computer Sciences University of California at Berkeley outline some of the challenges and solutions present in these systems in 2008, highlighting abstraction layers in computing and timing as major issues, but cite concurrent programming and incremental improvements in technology as potential solutions [4].

The next research findings label cyberphysical systems as "The Next Computing Revolution" [5] and relate cyberphysical systems to electrical power, scientific discoveries, and disaster recovery. Finally, and perhaps most closely related to the proposed project here, a publication in the IEEE digital library discusses how cyberphysical systems will revolutionize the intelligent systems industries (think digital assistants, human interaction with machines) with improvements in cognitive ability [6]. All these articles and more discuss the relevance of cyberphysical systems in our world today, and this project aims to contribute to this growing set of literature.

Project Design and Approach

Original Approach

The original approach to this project involved thorough research, implementation, and reporting. The first step was to continue learning more about cyberphysical systems, VR, and robotics, using the Prior Work as detailed in the project proposal. Since this project had both a "physical" and a "cyber" aspect, there were two tracks of work in combination with a connectivity aspect that needed to be balanced and worked concurrently; hardware and software. The hardware component involved a rover to run the virtual maze, which was purchased, not built out. However, this rover was going to be connected to the virtual environment, so one of the early goals of this project was to set up communication. The implementation of this communication was to be explored further through research in connectivity protocols in the first weeks of the project. As a starting point, some potential ideas included using a FRDM board and MQTT protocol with AWS MQTT and lambda, or a TCP/IP socket channel connecting to a server. In addition, this project was to use Raspberry Pi for cryptocurrency, as explained in more detail below.

The second piece of this project was the software component, which was again a focus early on. The virtual reality component required a staging environment, with early thoughts being

Unity in combination with the Decentraland and other APIs, and a headset such as a Google Cardboard or Oculus Rift, which was already obtained. This was going to involve developing a scene for the maze, creating paths for the rover to follow, and establishing the correct interactions with these objects (for example, if the rover “hits” one of these obstacles, it should react accordingly and stop). Using methods from the Decentraland APIs, commands would be given to the rover to run the virtual maze. Blockchain with Ethereum were to be utilized to incentivize the rover to complete the maze in a given amount of time. If the rover completed the maze in said time window (targeting 60 seconds as an initial goal), it would be rewarded with cryptocurrency. With the introduction of this reward system, a Raspberry Pi would be used to create a crypto wallet in order to receive funds. All of these APIs, resources, and tools were explored more and finalized during the first week of the project in an attempt to verify their feasibility. Changes to technologies were made accordingly, and early on, as detailed in the Modified Approach section below.

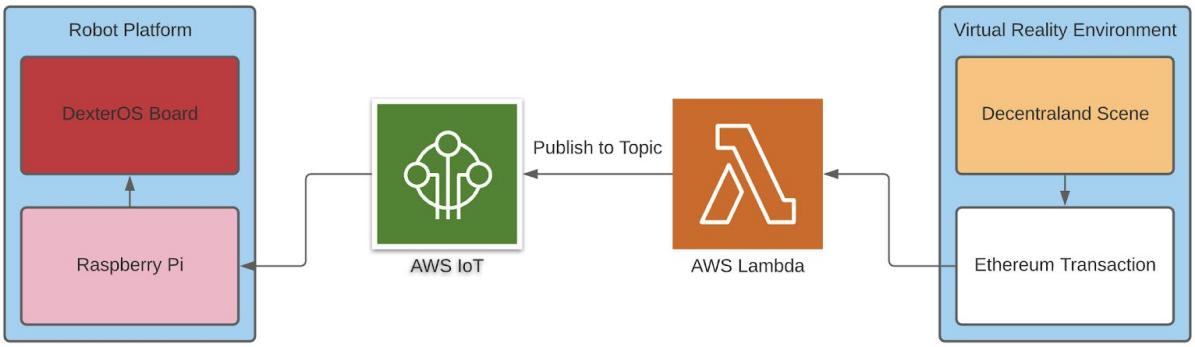
The final component of this project was going to be this report and a poster presentation detailing the resources, methods, aspects learned, project progress, future work, and references. This, along with the project itself (including rover, communication stack, and GitHub) would serve as the deliverables for the project. Some potential risks of this project included exploring the Decentraland API to ensure that it provides users with the ability to control the movement of a virtual robot, and whether or not it provided users with “boundary” information that would be required to create a maze. In addition, the ability to put a crypto wallet on a Raspberry Pi and send it currency needed to be verified.

Modified Approach

As alluded to in the section above, the approach to this project was modified during the course of the semester due to discoveries from the early research and time constraints. This project still had both a “physical” and a “cyber” aspect, and still had hardware and software tracks. Instead of having a rover to run the virtual maze, the purchased rover was connected to a lambda function in the AWS Console and the python SDK. The system onboard the rover consisted of a DexterOS board to command the motors, and a Raspberry Pi for communication with the lambda function.

The second piece of this project was the software component, which was again a focus early on. Instead of the focus being on virtual reality and building out a maze in the SDK, the project pivoted to have more of a focus on blockchain technology with ethereum transactions. The project still utilized virtual reality and Decentraland, however, instead of building out a maze, a scene with a block treated as a button was created. The lambda function waited for a transaction to occur on the blockchain, and when it saw a change to the account balance, it published a request to the rover to move forward a certain amount. In this model, the ether acts as “fuel” that powers the rover.

The final component of this project was adjusted to be this report and a 15 minute, recorded presentation, still detailing the resources, methods, aspects learned, project progress, future work, and references. The final deliverables for the project included these resources, along with a GitHub project that includes the python SDK publish/subscribe routine, the lambda function running in AWS, the Decentraland scene, and the javascript program to send ether. The system diagram below details the full stack, end to end.



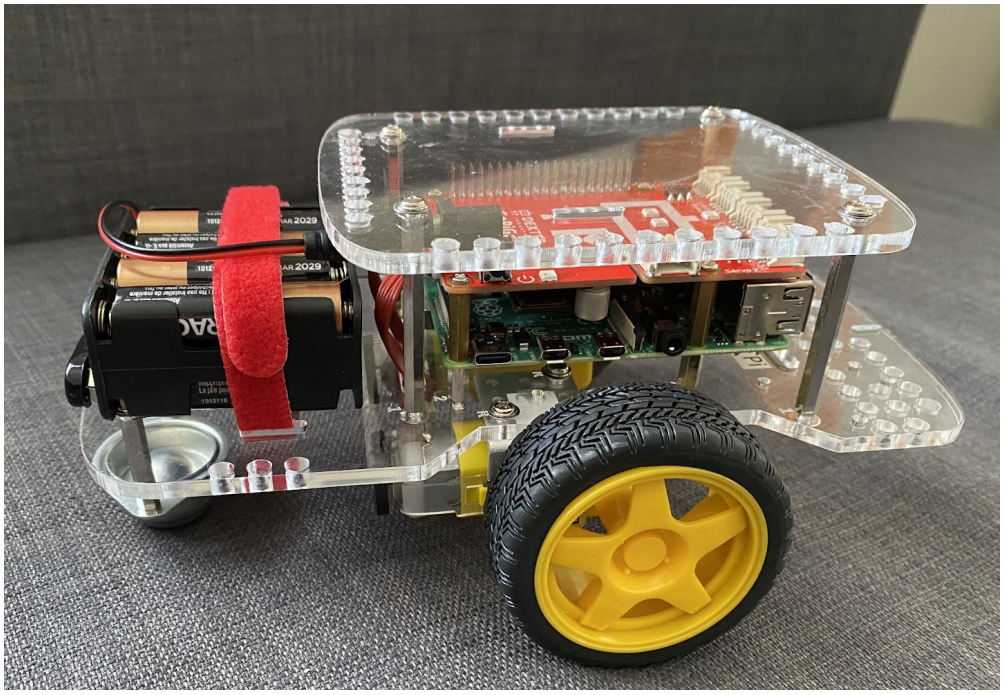
This system diagram is representative of the modified approach discussed, and serves as a high-level overview of the entire system.

Results

Overall, the project accomplished most of what it had intended to, accounting for the adjustments laid out in the Modified Approach. The following sections detail the milestones reached for each segment of the project.

Physical System

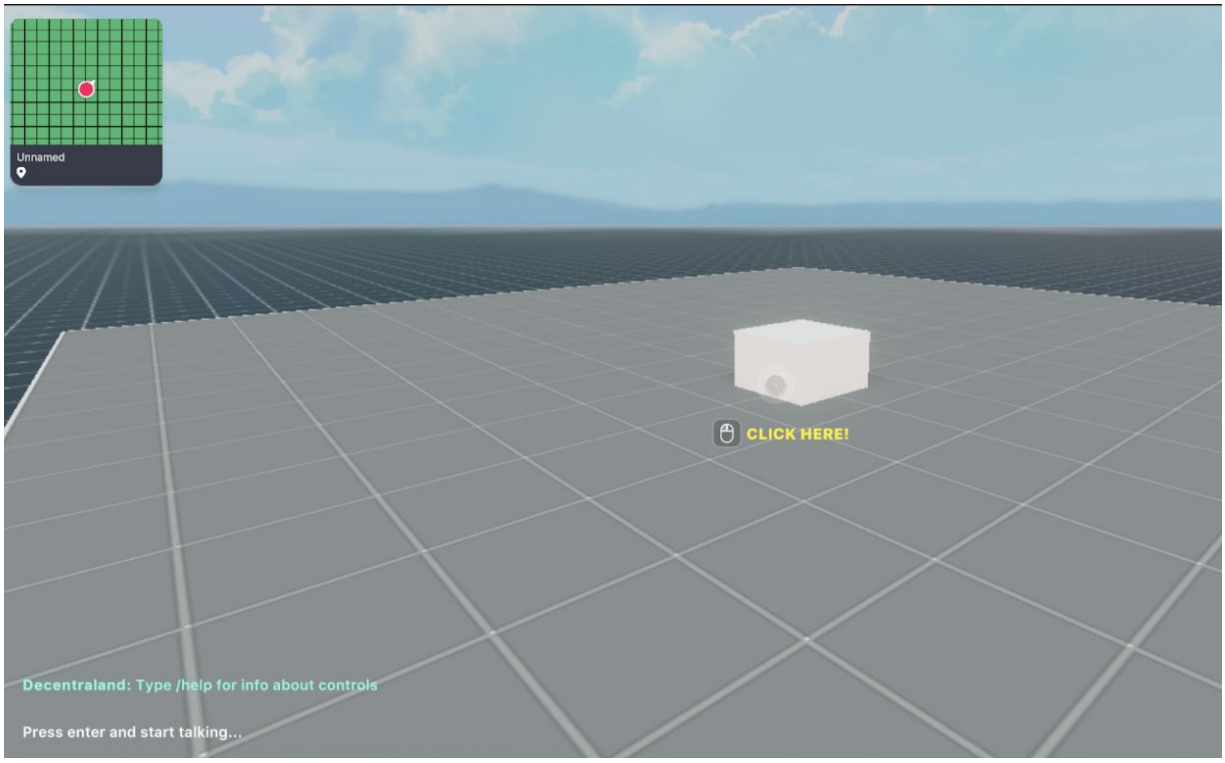
The physical system, the rover, was purchased and built successfully as anticipated. After some initial research and anticipating shipping delays due to the COVID-19 pandemic, the GoPiGo3 base kit [7] was purchased for $99, and a Raspberry Pi 4 starter kit [8] served as the onboard logic system and was ordered on Amazon for $89. The GoPiGo3 consisted of two small motors and wheels, a DexterOS board, and a battery pack, and was assembled in a few hours. The completed GoPiGo3 robot can be seen in the image below.



The Raspbian OS was installed and booted on the raspberry pi, and the DexterOS software [9] was installed, along with C, python, and the AWS IoT Device SDKs for both languages. The wireless capabilities on the Raspberry Pi 4 allowed for remote access into the Pi, which was also established with VNC Viewer and SSH. The robot was able to be commanded and controlled with a UI included with the kit and installed on the Pi, or via the DexterOS software, written in python. This involved forwards and backwards movements, left and right, speed increase and decrease, calibration, interaction with external sensors, and so on. After some trial and error, a publish and subscribe sample from the AWS IoT Device SDK for Python [10] was used to send and receive messages with AWS. The robot commands were imported into this project, and the robot received commands to move forward from this communication stack.

Virtual Reality

The Virtual Reality component of this project was developed using the Decentraland SDK [11]. The scene was created as intended, and was relatively simple; it consisted of a parcel of LAND with a square box that doubled as a button with hover text “Click Here” for sending a cryptocurrency transaction. The scene also included a map showing current location in the scene, and was able to be rendered locally using the “dcl start” command with the Decentraland CLI. A rendering of the scene can be seen in the image below.



Behind the scenes, the code was written in javascript, and built off of a MANA transaction example [12]. The code was originally generated from a Decentraland scene builder, which packages the dependencies and configuration needed for the scene to render with the CLI. The code also includes a function for the button, which contains logic for sending a transaction. The code for this section of the project, including information on how to render the scene, can be found in the public project GitHub [13].

Lambda Function

The lambda function was not included as part of the original proposal, however, after early research and the pivot of the project, it was a necessary addition. The lambda function was added as part of the communication stack. It was written in python, and serves two main purposes. First, it uses the infura API [14] to watch for transactions between two accounts on the blockchain occur. Second, when the lambda function observes a change in the account balance, it publishes a message to the robot via the AWS IoT, and the robot moves forward according to the transaction. The lambda function was created with the concept of least-privilege access, so along with the code, the appropriate roles, policies, and permissions were generated to allow communication with the topic and AWS IoT. The function was written in the AWS Console but was included in the aforementioned GitHub repository.

Cryptocurrency and Blockchain

The final deliverable was the cryptocurrency and blockchain capability of this project. Although mentioned as a component of the project in the proposal, with the modifications in the project

this played a heavier role in the project than originally intended. Instead of having a wallet on the Raspberry Pi and rewarding it with cryptocurrency for finishing the maze, the cryptocurrency instead fuels the robot. A MetaMask [15] wallet was created with two accounts, one with ether to send, and the other to receive the ether. An account on Infura was also established to interact with the accounts and blockchain. Using the MetaMask, Infura, and the Web3 JS library [16], a script was created in javascript to send a signed transaction from the first account to the second. This involved establishing the Infura account and ethereum as the providers, researching and including gas, gas price, and gas limit in the transaction, and converting between wei and ether for the request. An environment configuration file was also included, which contained the wallet addresses, wallet private key, and Infura access token to sign the transaction. This was not included in the final product since it would be a huge security concern to expose keys. The scripts attempting this transaction can be seen in the GitHub repository.

Discussion

The Results section presents the accomplishments, but behind these accomplishments was a great deal of research, tried and failed implementations, and corresponding redirects. Purchasing, assembling, and testing the physical system was straightforward, but developing code to subscribe to the topic was not. Originally, a publish/subscribe program using the AWS IoT Device SDK for Embedded Systems in C was set up and in use, and since it was in python, the DexterOS was going to be bundled as a shared library, or a DLL, and imported into the C code. This process was explored and a “practice” DLL was created before realizing that the DexterOS was not a true API and could not be used as a shared library. To work around this, the project pivoted to use the AWS IoT Device SDK for Python, which led to more time being consumed to download and install python and its proper dependencies, and set up and test the publish/subscribe program for python.

The virtual reality scene was generated as intended, even though it could be expanded upon. The scene did run into a few issues running locally with the Decentraland CLI due to some versioning conflicts, but this was quickly resolved. The lambda function coupled with the AWS IoT involved a lot of research and plenty of tried and failed attempts. Different blog posts and tutorials were found and followed in both node and python, and experimented with setting different roles and permission levels, using SNS topics, and so on. A lot of time was spent on exploring the use of AWS Greengrass, which led to a write up on the service itself, including the pros and cons of using it over the AWS SDKs. For the interested reader, the writeup is included as part of this report in Appendix A. This was determined to be an inefficient route for what the project was trying to accomplish, and thus was abandoned in favor of the AWS SDKs.

The cryptocurrency and blockchain component of the project went through some reiteration as well. Besides researching and ordering the robot, this was the first big focus of the semester. One of the big questions mentioned in the project proposal was whether or not a crypto wallet could be placed on the Raspberry Pi. Originally, the approach was to use a light node (vs a full node) on the blockchain and a container to interact with the blockchain and ethereum. After a

few weeks of research, running scripts, attempting to get the light node up and running on a computer before even moving to the Raspberry Pi, external sources were consulted, and the light node was decidedly not the best avenue for this project. This led to the use of Infura and MetaMask instead, which involved more research, and exploring API documentation, blogs, and tutorials for sending raw transactions. A few attempts were made across a few different scripts to get this working, all of which were included in the project's GitHub repository.

Challenges & Failures

As with any task, this project came with its fair share of difficulties. An obvious first callout was the challenges of working on this project during the COVID-19 pandemic, which presented numerous challenges. The entire project had to be done remotely, advisor meetings were conducted exclusively on Zoom, any resources on campus that could have typically been used were inaccessible, and there were relevant personal stressors relating to finances, health and wellbeing, living situation, family, and more that we are all facing individually. In addition, this project was being completed concurrently to working full time remote and taking condensed six week summer classes, so time management was key.

In terms of the project itself, there were a number of roadblocks as well. First, the research done up front paid off, because there were a few items that needed to be adjusted early on, as detailed in the prior sections. A lot of time was spent on getting a light node working with the Raspberry Pi and trying to set a wallet up on the device, which proved to be an inefficient way to work with the blockchain for this project. Although the robot was easy to build, it did take longer than anticipated to get the robot communicating with AWS and the lambda function. The documentation on this process was not very clear, and there were a lot of avenues explored that led to deadends. The original approach was to use the AWS IoT Device SDK for Embedded Systems in C, but there were challenges interacting with the robot software, which was written in python, so a switch was made to use the AWS IoT Device SDK for Python. Sending ether with ethereum and infura was also a bit trickier than expected, and introduced plenty of new terminology and concepts to understand, which consumed a lot of time in the last few weeks.

What I Learned

Although many of the concepts introduced in this project were familiar, most of the work was new, or explored in much more depth than before. Even though the project approach was adjusted and there were some failures and incompletes with some of the deliverables, so much was learned over the course of the semester. GoPiGo3 and the Dexter software was completely new, and interesting to research and build. Working with Raspberry Pis and the Raspbian OS has been done in the past, however, setting up the wireless capabilities was a bit of a learning curve, and working with the AWS IoT SDK for Python to publish and subscribe to a topic while controlling a robot was also a new concept. The virtual reality component was relatively new; the Decentraland SDK and scene builder were introduced in EC544 in the spring semester, so this work built off of some of the work previously done in that space.

There was some familiarity with lambda functions from work experience, so the basic concept was not new. Creating a lambda function to publish a topic and work with the AWS IoT, however, was a new application that involved lots of troubleshooting and research. In addition, the security aspect with policies, roles, and permissions was a learning curve, since it was handled differently in a work environment than it is at an individual level with these specific services. The cryptocurrency and blockchain component of this project was completely new, and arguably the largest of the learning curves. A MetaMask wallet was set up previously in the EC544 course, but not much was done beyond this previously. Ether, Infura, sending and receiving transactions, the concept of blocks and ether, and how to interact with them, gas and gas prices, and even working with Web3 JS and javascript had all never been done before. This led to a lot of research, tutorials, exploring documentation, example projects, and blogs, trial and error, and so on. This consumed most of the last few weeks of this project, but was invaluable information, and provided for a better understanding of the world of cryptocurrency and blockchain.

Future Work

With more time and the access to more resources, there are plenty of possibilities for building on or expanding this project. The rover could be upgraded, customized, or built from scratch and could expand to include an autonomous feature. For someone interested in virtual reality or gaming, the Decentraland scene could be expanded upon; to add another layer of complexity, the scene could include functionality to send and also receive MANA through ether or other types of cryptocurrency, or could revert back to the original idea for a maze. Decentraland does allow scenes to be deployed via LAND, so if time and money permitted, the scene could be deployed into Decentraland. The lambda function and communication stack could be further optimized, taking advantage of the synchronous/asynchronous feature of lambda to wait for the transactions on the blockchain in a more efficient manner. The logic could also be expanded to include more robot movements, such as turn left, turn right, move backwards, speed up, or slow down. For the cryptocurrency and blockchain piece it would be interesting to explore different forms of cryptocurrency, or introduce a dialog box for easy and customizable transactions.

Aside from expanding on the elements within this specific project, this project could aid in future works relating to cyber physical systems. The system of virtual reality, cryptocurrency, and robotics in particular would be interesting to explore at a much larger level. As development on autonomous vehicles continues, it might be relevant to think of autonomous taxis or ubers accepting payment in the form of cryptocurrency. The virtual reality and robotics component can also aid in testing autonomous vehicles in the virtual world, and mimicking in the physical world. Beyond this, there are thousands of applications for blockchain, robotics, and virtual reality; advances in transportation, medicine, economics, and so on.

Acknowledgements

Thank you to Professor Babak Kia for taking the time this summer to advise this project. Your expertise, patience, understanding, and oversight has been invaluable.

References

[1] https://ptolemy.berkeley.edu/projects/cps/
[2] Augmented Reality Monopoly EC463 2013 Senior Design Capstone Project Report & User Manual
[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4435108/
[4] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.156.9348&rep=rep1&type=pdf
[5] https://dl.acm.org/doi/pdf/10.1145/1837274.1837461
[6] https://www.computer.org/csdl/magazine/co/2017/12/mco2017120007/13rRUytF44K
[7] https://www.dexterindustries.com/store/gopigo3-base-kit/
[8]https://www.amazon.com/gp/product/B07V5JTMV9/ref=ppx_yo_dt_b_asin_title_o06_s00?ie=UTF8&psc=1
[9] https://gopigo3.readthedocs.io/en/latest/
[10] https://github.com/aws/aws-iot-device-sdk-python-v2
[11] https://docs.decentraland.org/development-guide/SDK-101/
[12] https://github.com/decentraland-scenes/MANA-Transaction
[13] https://github.com/kristiperreault/Cyberphysical-VR-Robotics
[14] https://infura.io/docs
[15] https://metamask.io/
[16] https://web3js.readthedocs.io/en/v1.2.7/getting-started.html

Appendix A

AWS Greengrass Writeup

AWS IoT Greengrass allows for IoT devices to take advantage of cloud services such as Lambda and Docker offline, and then deploy to the cloud. It offers secure communication, fast response time for local events, device validation, container support, and reduced cost. You can use any IoT Device including Raspberry Pi's with the Device SDK, which is available in python, node, c, and java. It effectively serves as a "gateway" to the cloud from IoT hardware devices.

As with anything, there are some downsides of AWS IoT Greengrass, notably the documentation is not very great. The AWS documentation is, in general, not very thorough, and AWS IoT Greengrass is rather new, so there isn't too much out there in the way of outside tutorials and blogs. The service itself is also very complex, requiring a good deal of installations and configuration to get up and running, including creating a group, creating a lambda function,

registering them with IoT via lambda definitions, defining resources to access, configuring loggers and registering devices, setting up MQTT subscriptions, and deploying to the greengrass core. Everything is also done with versioning, rendering each object immutable.

Conclusion: it does seem like a very powerful tool if you know what you are doing or have the time to figure it out. There is a helpful configuration tool called greengo that someone developed to help with the setup (linked below) that could be leveraged. It does seem like a relatively new/underused tool, since almost all of the documentation I found out there came directly from Amazon rather than other developers.

Sources:
https://aws.amazon.com/greengrass/?nc2=h_ql_prod_it_gg
https://read.acloud.guru/aws-greengrass-the-missing-manual-2ac8df2fbdf4?gi=dd8576932b69
https://github.com/dzimine/greengo