Implementing a Conceptual Teleoperation Communication System for Remote Controlled Vehicles

Matthew H. Taylor

University of Illinois at Urbana-Champaign, Department of Aerospace Engineering

This paper analyzes the feasibility of controlling a vehicle remotely with current wireless communication standards using the MQTT protocol. Two very common issues arise when creating a teleoperation communication system. The first and most common issue is message delay, while the second and more underestimated issue is failure to utilize encryption techniques. These issues can easily be resolved by creating a local broker, changing message delivery settings to use QoS 0, and applying open-source Python libraries on encryption to the communication framework.

Nomenclature

MQTT	=	Message Queuing Telemetry Transport
IoT	=	Internet of Things
QoS	=	Quality of Service
RPI	=	Raspberry Pi
GUI	=	Graphical User Interface

I. Introduction

John Deere recently donated an R-Gator Vehicle to the University of Illinois at Urbana-Champaign. The team's long-term goal is to upgrade it and transform it into a full fledge autonomous vehicle. The semester goal was to develop a method to control acceleration and steering for the autonomous vehicle remotely using open-source libraries in Python. The code written from this semester will be used as a starting point for the ultimate software stack for R-Gator's remote-control system. Two micro-computers were used for this project to accomplish these objectives (one on the ground station and one on the vehicle). This paper will explore the techniques used to create a teleoperation communication system for the vehicle in the "Work Done" section. "Work Done" will also include a technical overview of MQTT communication and discuss the advantages and disadvantages of using a local broker as a central data hub over a cloud-based broker. The discussion on implementing a local broker will then lead to a detailed solution to the common telecommunication issues of message delay and lack of security. The next section is "Future Work", which will discuss what can be done for the project going forward. Finally, the conclusion will wrap up key takeaways from the project.

II. Work Done

A. Overview

This project had many sub-goals, and the general progress followed Figure 1. The development stage took up the majority of the time during the semester and is where most of the goals were completed. Additionally, software testing was intertwined with the development stage. Tests were completed after each subsection in development wrapped up and step 2.5 had a final test after all of the code was integrated together.

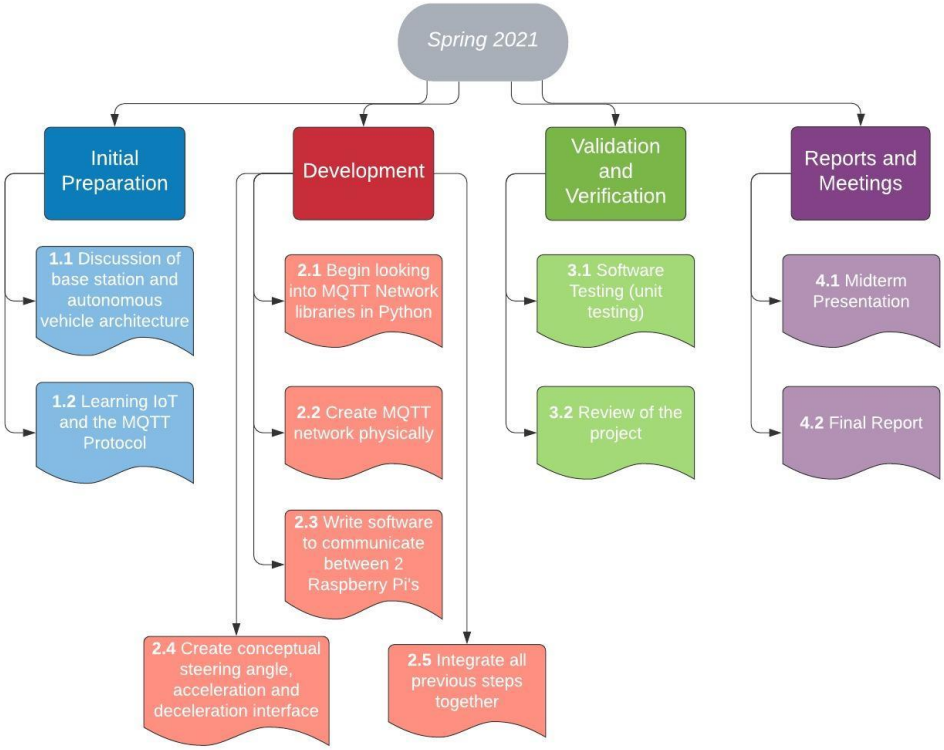

Figure 1. Block Diagram of Semester Goals.

B. MQTT Communication

The MQTT protocol is an application of IoT that allows for fast wireless communication between devices. It uses a publish/subscribe pattern where a client device can publish information to a broker (central hub for data) [1]. Another client subscribes to this information and immediately receives it from the broker when connected to the internet. One very important concept with the MQTT protocol is called quality of service, or QoS. QoS is a way for the publisher and subscriber to agree to how the message is received. There are 3 QoS levels. Level 0 is called at most once delivery, where a message is sent exactly once and there is no guarantee for the delivery of the message. Level 1 is the next step up, which is called at least once delivery. At least once delivery guarantees that a message will be delivered once or more than once, even if multiple confirmation messages must be sent between the publisher and subscriber. Finally, QoS 2 is exactly once delivery. This is by far the slowest quality of service level but is great for when internet is

extremely unstable because it always guarantees delivery of a message [2]. For this project, QoS 0 is used for enhanced speed. There are some drawbacks for this type of delivery which will be discussed in the analysis section.

This research used MQTT communication by having one Raspberry Pi client publishing steering and acceleration commands at ground station and another Raspberry Pi for onboard the vehicle subscribing to these messages. Figure 2 shows a simple diagram of how the communication will work once integrated with the R-Gator vehicle.

Figure 2. MQTT Communication Diagram.

For this project, the ground station computer acts as both the publisher and local broker, while the subscriber is still the computer onboard the vehicle. The physical setup can be seen in Figure 3. The microcomputer on the left is a Raspberry Pi 3B+, which acted as the central broker and remote controller at ground station. The microcomputer on the right is a Raspberry Pi 4 and acted as the vehicle during simulation. The code itself uses open-source libraries in Python, with the main dependency for MQTT communication being the Paho-MQTT package.

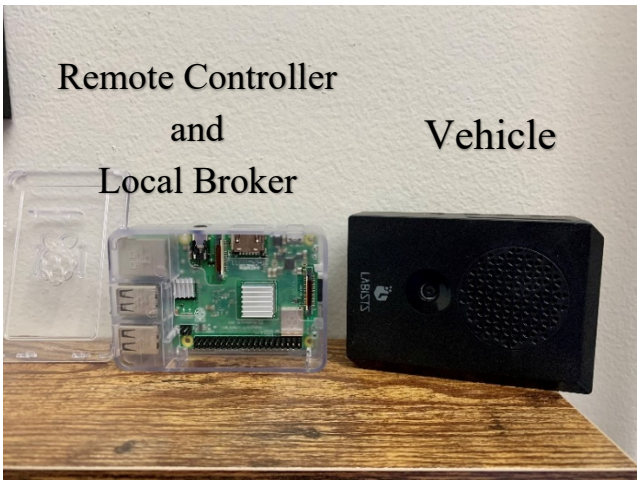

Figure 3. Experimental Setup for Remote Controller and Vehicle Computers.

C. Results

The final product for the semester was a graphical user interface that took in keyboard inputs to move a joystick. Three pieces of data were then published to the vehicle each time a key was pressed: x position, y position, and acceleration level. The x and y position refer to the location of the joystick with respect to its origin in the center. The acceleration level refers to what ring the joystick is on. As seen in Figure 4, there are three levels for acceleration and three for deceleration. The left window in Figure 4 depicts what ran on the publishing Raspberry Pi (remote controller) and the right window shows what ran on the subscribing Raspberry Pi (vehicle). The vehicle GUI did not take any keyboard inputs but received data from the remote controller and copied its movements. Furthermore, a command line interface was created for an authentication system. Before the joystick GUI starts, the user is prompted to login in order to ensure they have access to the vehicle. If the username and password is incorrect, they have no ability to take control of the vehicle.

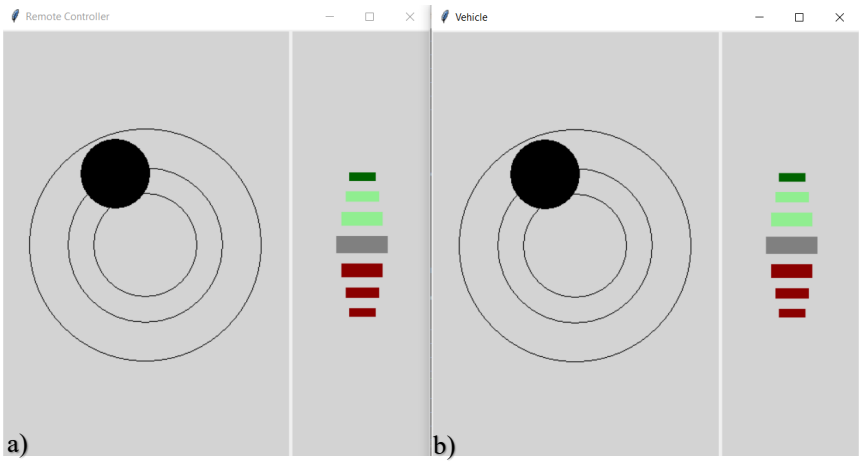

Figure 4. a) Remote Controller and b) Vehicle GUI.

D. Analysis

The data published almost immediately to the vehicle's microcomputer which is great for when the physical vehicle will be controlled. QoS 0 and the local broker implementation significantly contributed to minimizing lag in the system. First off, QoS 0 was used for at least once delivery. As stated previously, QoS 0 only sends a message to the broker once and does not guarantee delivery, but the advantage is that this form of communication is very lightweight and fast compared to QoS levels 1 and 2 [2]. One downside is that the message may not always be delivered, and when bandwidth is not at a premium, many messages could be skipped. For this research, it was assumed that connection to internet had minimal instabilities. Furthermore, it's not a huge concern if the vehicle doesn't receive every message, as long as it receives messages consistently. A legitimate problem occurs when messages are not sent consistently or when internet connection is unstable.

A local broker was also utilized over a cloud-based broker to decrease lag in the system. Initially, a cloud broker was used to send and receive information from the clients, however there was a 0.5-1 second delay each time data was published. After switching to a local broker on one of the Raspberry Pi's, the delay decreased to almost nothing. It's extremely important to have immediate response when controlling a vehicle or else many accidents could occur. Some disadvantages with having a local broker are that the publisher and subscriber must be on the same Wi-Fi network. A cloud-based broker would allow for a vehicle to be controlled from anywhere in the world when connected to separate Wi-Fi networks.

Lastly, incorporating encryption techniques was a strong priority during this project. The idea is that only authorized users should be able to access and use the joystick. In a real scenario, someone would be in the vehicle that is being controlled, and not having security measures in place to stop someone unauthorized from controlling the vehicle is too big of a risk. Paho offers great open-source libraries along with its MQTT library to help with this issue. For starters, the local broker was created with a package called Paho-Mosquitto. The Mosquitto library has protocols that can be setup to run with an authentication system, where the broker can only receive and send messages to users in its database. Additionally, the command line interface created for the joystick allows for a second layer of protection, so that passwords aren't just hard coded into the framework of the communication system.

III. Future Work

The project concluded with only the ability to simulate the vehicle being controlled with graphics that mimic the response of the remote controller. In the future, messages should be published directly to the vehicle to control steering and acceleration. The vehicle's controller area network (CAN) bus system plays a big role in taking control of an actual vehicle. CAN bus allows for any electronic control unit to communicate to the entire system in a vehicle and simplifies the communication framework for when new data needs to be received [3]. In the future, acceleration and joystick movements will feed into the CAN bus system in the vehicle to physically control acceleration and steering.

As the current design stands, there will be a ground station with a controller that communicates to the R-Gator Vehicle over the same Wi-Fi network. From the perspective of the communication system, the MQTT protocol sees no difference between a Wi-Fi network and for example, a 4G LTE network. Connecting to a 4G LTE network would allow for long range communication between the vehicle and ground station (i.e. the R-Gator Vehicle could be controlled from anywhere in the world). There are also many applications for MQTT outside of use for remote control systems. Facebook messenger, for example,

uses this protocol for low latency communication between users around the world. MQTT is a very lightweight, efficient protocol, and has encryption methods available that can keep user's information safe and secure. This protocol is a subset of internet of things, or IoT. IoT easily links data between devices in a scalable way, simplifying connections between multiple devices [1]. Hologram is a company that sells sim cards for IoT devices, allowing them to connect to a global network. There are endless opportunities with the combination of using a Hologram sim card along with any IoT device.

IV. Conclusion

General rules of thumb were discovered when creating a teleoperation communication system. Using quality of service level 0 proves to increase message delivery speed but also increases the risk that messages won't be delivered. Additionally, creating a local broker significantly decreases lag in comparison to a cloud-based broker. Finally, the use of a local broker also makes it easy to have only users in the database send and receive messages. This semester brought forth many challenges and opportunities for growth. Throughout the process, much was learned on MQTT communication and techniques for faster communication and authentication. The achievements from this semester are a great starting point for the pursuit of turning the R-Gator into a fully autonomous vehicle.

V. Code Access

All the Python libraries used for this project are open source and freely available to download. As such, the code developed for this project is also openly available for use by anyone interested in seeking more information. Visit https://github.com/mht3/Remote-Controller to access the repository and open INSTRUCTIONS.md to see how to set up your own local broker and run the remote controller and vehicle GUI.

References

[1] Liu, Xiangtao, et al. “The Method of Internet of Things Access and Network Communication Based on MQTT.” Computer Communications, vol. 153, 1 Mar. 2020, pp. 169–176.

[2] S. Lee, H. Kim, D. Hong and H. Ju, "Correlation analysis of MQTT loss and delay according to QoS level," The International Conference on Information Networking 2013 (ICOIN), 2013, pp. 714-717.

[3] J. R. Henderson, J. M. Conrad and C. Pavlich, "Using a CAN bus for control of an All-terrain Vehicle," IEEE SOUTHEASTCON 2014, 2014, pp. 1-5.