Ryerson University

**Department of Electrical, Computer, & Biomedical Engineering**
Faculty of Engineering & Architectural Science

| | |
|---|---|
| **Course Title:** | Distributed and Cloud Computing |
| **Course Number:** | COE892 |
| **Semester/Year (e.g.F2016)** | W2023 |

| | |
|---|---|
| **Instructor:** | Dr. Muhammad Jaseemuddin |

| | |
|---|---|
| *Assignment/Lab Number:* | 03 |
| *Assignment/Lab Title:* | RabbitMQ |

| | |
|---|---|
| *Submission Date:* | Wedneday March 22nd 2023 |
| *Due Date:* | Wednesday March 22nd 2023 |

| Student LAST Name | Student FIRST Name | Student Number | Section | Signature* |
|---|---|---|---|---|
| Faisal | Mahir | 500896206 | 01 | MahirF. |
| | | | | |
| | | | | |

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

# COE 892 Lab 3 Report

## Introduction

The main purpose of this lab assignment is to take the gRPC code from the lab 2 assignment and modify it to incorporate a RabbitMQ publish/subscribe architectural structure for the mine detector robot and the ground control server. More details about the program will be discussed in the section below.

## Step-by-Step Lab 3 Procedure

First and foremost, similarly with the lab 2 assignment, the ground control server that rover has to communicate with relies on three different files. These three files consist of the mines.txt file which contains the serial numbers for all of the mines on the land space, the 2D land space file, and the command list JSON data. However, the only difference with lab 3 is that the digging command is no longer be sent to the rover as a valid command. This means that the rover client should automatically disarm any mine that it comes upon contact with. The way that it does this is that upon coming into contact with a mine object, it publishes a "demining" task to a demining queue which is subscribed to by a deminer program. This deminer program then disarms the mine object from the queue and publishes the pin and other results of the disarming process to another queue called the disarmed-mines queue and those results are also written to an output log file of sorts by the gRPC server program as well.

1. The first step in my procedure is that I took the proto file that I used in my lab 2 assignment and modified it to only include the grpc protocols that involve getting the 2D land map, the command list for the rover (all commands except for the dig command 'D' of course).

```
1   syntax = "proto3";
2
3   service GroundControl {
4       rpc getMap (mapRequest) returns (mapResponse) {}
5
6       rpc getCommands (commandsRequest) returns (commandsResponse) {}
7
8       rpc getSerialNum (mineSerialNumRequest) returns (mineSerialResponse) {}
9   }
10
11  message mapRequest {
12      string mapFile = 1;
13  }
14
15  message mapResponse {
16      int32 rows = 1;
17      int32 cols = 2;
18      repeated string map = 3;
19  }
20
21  message commandsRequest{
22      int32 commandListNum = 1;
23  }
24  message commandsResponse {
25      repeated string commands = 1;
26  }
27
28  message mineSerialNumRequest {
29      int32 roverXPos = 1;
30      int32 roverYPos = 2;
```

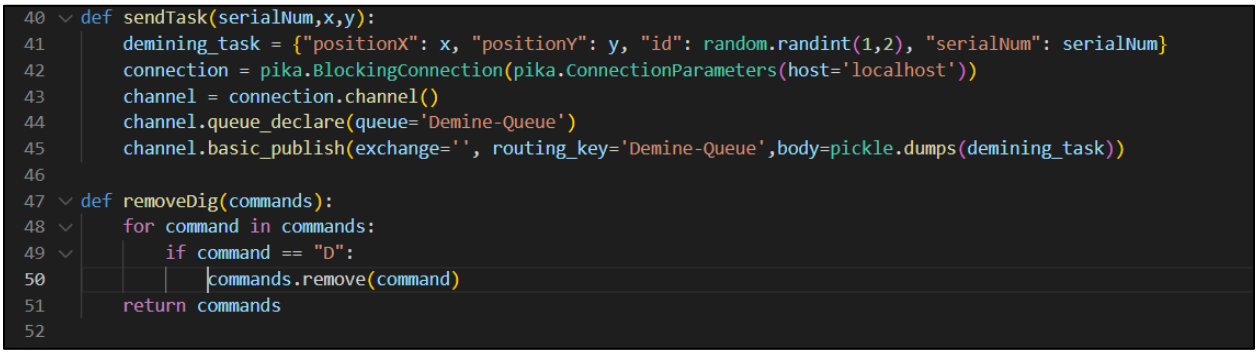
*Figure 1: Step 1 Screenshot*

2. The next step in my procedure was that I took the server.py file from my lab 2 assignment and modified it to include a new function called getPin which creates a queue called defused mines which the server uses to write the disarming results to an output file called defusedMines.txt. I also removed a bunch of gRPC methods from the lab 2 server program that were unrelated to the grpc methods that are noted in the newly modified proto file. Since the ground control server and the RabbitMQ have to be running simultaneously when the rover comes into contact with a mine object, the getPin method is invoked in the gRPC server execution method (see below)

```
def getPIN():
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='Defused-Mines')

    def callback(ch, method, properties, body):
        defusedMine = pickle.loads(body)
        print(" [x] Received PIN " + str(defusedMine["PIN"]))
        if not os.path.exists("defusedMines.txt"):
            f = open("defusedMines.txt", "x")
            f.close()
        with open("defusedMines.txt", "a") as mineLog:
            mineLog.write("Mine disarmed at x-coordinate " + str(defusedMine["positionX"]) + " and y-coordinate " + str(defusedMine["positionY"])
                          " with PIN " + str(defusedMine["PIN"]) + '\n')
    channel.basic_consume(queue='Defused-Mines', on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

def serve(): #Primary function that is used to run the server
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    groundControl_pb2_grpc.add_GroundControlServicer_to_server(GroundControl(),server)
    server.add_insecure_port('[::]:50051')
    server.start()
    getPIN()
    server.wait_for_termination()
```

*Figure 2: Step 2 Screenshot*

3. The next step in my procedure was that I took the client.py file and modified to account for the removal of the ‘D’ command from the command list. I also added a bunch of lines of code which incorporate the creation of a demine queue which the rover client uses to send demining tasks to be forwarded to the deminer program with the help of pika python 3 library and the RabbitMQ broker.



```
def sendTask(serialNum,x,y):
    demining_task = {"positionX": x, "positionY": y, "id": random.randint(1,2), "serialNum": serialNum}
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()
    channel.queue_declare(queue='Demine-Queue')
    channel.basic_publish(exchange='', routing_key='Demine-Queue',body=pickle.dumps(demining_task))

def removeDig(commands):
    for command in commands:
        if command == "D":
            commands.remove(command)
    return commands
```

*Figure 3: Step 3 Screenshot A*

```
    while rovNum.isdigit() and int(rovNum) >= 1 and int(rovNum) <= 10:
        response2 = stub.getCommands(groundControl_pb2.commandsRequest(commandListNum=int(rovNum)))
        commandMat = response2.commands
        digFreeCommands = removeDig(commandMat)
        pathMat = copy.deepcopy(landMat)
        pathRows = len(pathMat)
        pathCols = len(pathMat[0])
        currentFacing = "S"
        x = 0
        y = 0
        for command in digFreeCommands:
            if x > pathRows - 1 or y > pathCols - 1 or x < 0 or y < 0: #Check if the rover goes out of bounds for the land space
                break
            else:
                # Cases where command is M
                if currentFacing == "S" and command == "M":
                    if pathMat[x][y] == "1": #Rover is on a mine
                        print("Mine detected!")
                        pathMat[x][y] = "*"
                        response3 = stub.getSerialNum(groundControl_pb2.mineSerialNumRequest(roverXPos=x,roverYPos=y))
                        if response3.serialNum == "":
                            print("Mine cannot be disarmed. Unable to identify the mine's serial number.")
                        else:
                            pathMat[x][y] = "#"
                            sendTask(response3.serialNum,x,y)
                    elif pathMat[x][y] == "0": # you can move and the rover is not over a mine
                        pathMat[x][y] = "*"
                    if x < pathRows - 1:
                        x = x + 1
```

*Figure 4: Step 3 Screenshot B*

4. Lastly, I created a new python program for the deminers that are to be used called deminer.py. This program is a pure RabbitMQ program, and its purpose is to receive the demining tasks from the client via its demine-queue subscription take in the serial number from the task list and hash it in order to disarm the mine. Once the mine object has been disarmed, it will publish the results of the disarmed mine object to the disarmed-Mines queue which is managed by the gRPC server program. To initiate the execution of the deminer program, it simply gets input from the user by asking the user which deminer service that it wants to run. Furthermore, it is also important to note that whenever data is being sent between the three python programs (server, client, and deminer), data is sequentially being serialized and deserialized with the help of python 3's pickle library.

```
1   import pika
2   import pickle
3   import sys
4   import os
5   from hashlib import sha256
6
7   def disarmProces(id): #Mine Disarming process for rovers
8       validPin = False
9       pin = 0
10      while not validPin:
11          mineKey = str(pin) + id
12          hashedValue = sha256(mineKey.encode('utf-8')).hexdigest()
13          if hashedValue[0:6] == "000000":
14              validPin = True
15          else:
16              pin = pin + 1
17      return pin
18
```

*Figure 5: Step 4 disarmProcess Method*

```python
def publishPin(pin, mineData):
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='Defused-Mines')

    defusedMineData = {"positionX": mineData["positionX"], "positionY": mineData["positionY"], "PIN": pin}

    channel.basic_publish(exchange='', routing_key='Defused-Mines', body=pickle.dumps(defusedMineData))

    print(" [x] Sent PIN " + str(pin))
    connection.close()

```

*Figure 6: Step 4 publishPin Method*

```python

def main():
    demineNum = input("Please enter the deminer number (1 or 2): ")
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

    channel = connection.channel()

    channel.queue_declare(queue='Demine-Queue')

    def callback(ch, method, properties, body):
        demine_task = pickle.loads(body)
        print(" [x] Received %r" % demine_task)

        pin = disarmProces(demine_task["serialNum"])
        publishPin(pin,demine_task)

    channel.basic_consume(queue='Demine-Queue', on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print("Interrupted")
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

*Figure 7: Step 4 main method*

## Conclusion

After executing this lab program, I noticed that after the pin for the disarmed mine object has been sent and received by both the deminer program and the grpc server program, the deminer still continues to run in the background after the exchange. This is to be expected because you are essentially running the program with the assumption that the ground control server may still receive pins from the deminer after a single exchange because there may more than one mine object that the rover may come into contact with. Additionally, since the getPin method is also

invoked in the gRPC server execution method and the server must operate in the background even after the client program has terminated, and that the getPin method helps to hold all three queus in the RabbitMQ service together, that is another reason why the deminer program running in the background after the all mines have been disarmed is to be expected. Unfortunately, because of this, it would mean that in order to run the program with a different deminer, all 3 files have to be closed and rerun manually. Other than that primary observation, the lab 3 assignment seems to be running and behaving as expected.