


UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering
EEC 172 Spring 2023

LAB 6 Verification

Team Member 1: Bradley

Team Member 2: Thomas

Section Number/TA: A05

Date	TA Signature	Notes
6/6		All requirements met 70/70 + Used GPS module (UART) with CC3200 to get current location

Qualified
for Finals
Admits
06/09/23

EEC 172 Lab Six

Bradley Manzo
Computer Engineering B.A.
916953788
brmanzo@ucdavis.edu

Thomas Ke
Computer Engineering B.A.
917513004
thke@ucdavis.edu

1. INTRODUCTION

This lab tasked us with developing a novel application using the CC3200 to interface with peripherals, computing data, and uploading to an AWS server. Our team chose to develop an application to route the user to a random restaurant according to their choice in restaurant type.

2. GOALS AND TASKS

Beginning the lab, our first goal was to identify which parts of our previous labs could be adapted to serve our new system. We integrated the infrared receiving and multi tap decoding capabilities from lab 3 to retrieve the user input. Then we compiled a dictionary of all restaurants in Davis with their associated address into different categories of food.

Next, we researched what the best peripheral would be for ping a GPS signal, and what form of communication this would entail. Then we had to analyze and decode the data being sent from said peripheral, including parsing each NMEA sentence for those containing coordinates, parsing the coordinates from the string, and converting the coordinates into the proper format to be used in the Google Maps API.

The final task of this lab was to develop a simple webhook to listen to our inbox for the AWS message, parse the body of the email into the starting and ending location, and compose a new email to return to the sender enclosed with the directions to the restaurant.

3. CONCEPTS AND UNDERSTANDING

Since we adapted parts of previous labs, we will briefly cover the required concepts and understanding. The IR receiver is an active high sensor that pulls low when receiving an IR signal. The remote we are using to control our application encodes the particular button being pressed in the NEC format which specifies a leading code with a burst of 9 ms followed by a pause of 4.5 ms, and finally the data word. If a button is being held, rather than the word being retransmitted, only the leader code and a single bit are sent out at an interval of 104 ms. The data word consists of four bytes: Address, Inverted Address, Command, and Inverted Command. The command portion is decoded into the corresponding button when the interrupt handler has received the 32nd consecutive bit. Multitap is a technique which allows letters to be mapped to the keypad depending on how many times the key is pressed within a short duration, which we set to be 150ms.

Retrieving the coordinates of the user's current location requires interfacing with the NEO-6M GPS Module which pings the GPS network using its antennae and transmits NMEA sentences through a UART connection with the board. NMEA stands for the National Marines Electronics Association, and transmits a variety of sentence types containing GPS information. The sentence of interest begins with \$GPGGA and contains the coordinates in the DDMM.MMMM format, with D being degrees and M being minutes. We are able to convert this to DD.MMM format which is accepted by Google Maps API. The latitude is negative if in the southern hemisphere, and the longitude is

negative if in the western hemisphere. Therefore, the letters indicating the cardinal directions for each are also extracted from the sentence.

We also use AWS in a similar manner to the previous lab, Lab 5, where it is used to handle the device shadow, certificates, policies, etc. as well as send SNS messages to our emails from the CC3200.

When the message from the CC3200 is received by AWS, we also process the message using AWS Lambda through a Python interface with JSON, which allows us to extract the various data fields that we wish to handle. These data fields include things such as restaurant address and user location and are encapsulated within a JSON header and tail to be transmitted.

This email is then received by the webhook we designed using Make.com. It listens to the designated inbox and generates the directions to the provided destination address from provided current location and encloses them within a new email.

4. METHODS

The methods of decoding the IR signal from the remote in NEC format into the associated characters using multitap has been elaborated above.

```
Host Driver Version: 1.0.1.14
Build Version 2.10.0.0.31.1.5.0.2.1.0.3.37
Device is configured in default state
Device started as STATION
Attempting connection to access point: nosmoking... Connected!!!
[WLAN EVENT] STA Connected to the AP: nosmoking, BSSID: 44:1c:12:f5:fa:e0
[NETAPP EVENT] IP Acquired: IP=10.0.0.43, Gateway=10.0.0.1
Connection established w/ AP and IP is acquired
Device has connected to the website: a2tthdzizttu6-ats.iot.us-east-1.amazonaws.com
$GPGGA,054735.00,3833.58040,N,12146.74135,W,2,11,1.00,8.8,M,-27.8,M,0000*6A
Latitude: 38.330002, Longitude: -121.459999
```

Figure 1: Internet Connection and NMEA Sentence Structure

The user's current location is polled immediately after the program begins execution for the purposes of the demo due to the nature of the GPS. We are unable to ping a signal within the lab, however if we were to implement a SIM card peripheral and http_get the instructions directly to the CC3200 instead of email, we could poll the current location at regular intervals outside the range of any single Wi-Fi. The UART interrupt handler listens to the constant stream of data for the pattern of "\$GPGGA" and then records the result into a buffer. If the recorded sentence is too short, it means it is an invalid sentence and it is disregarded. Once a valid NMEA sentence is recorded, the string is passed back to the main program where the int status is deasserted and the comma separated string is parsed into the latitude, latitude sign, longitude, and longitude sign. These coordinates are then converted to the aforementioned appropriate format through a series of conversions and shifts. Once in the appropriate format, the coordinates are permanently stored and the associated UART is disabled for the rest of the program. The destination addresses are hardcoded in separate 2D arrays according to the cuisine. When the user selects a specific cuisine using the remote according to the GUI, a random restaurant address pair is retrieved from the array using rand(). When the user pushes the MUTE button, both the user's coordinates and destination address are encapsulated within a JSON packet when calling http_post(). This is achieved by appending each data value onto the header

and combining them with the appropriate syntax to be a valid JSON, and finally transmitted to the AWS.

As previously mentioned in the Concepts and Understanding section, we reused several of the AWS setup methods from the previous lab. However, this lab included a new portion of AWS known as AWS Lambda, which performs further processing on messages sent to the AWS device shadow.

AWS Lambda performs this processing on the sent messages using JSON, and in this case, we elected to use a Python interface with the JSON for the sake of simplicity and compatibility. The main purpose of this AWS Lambda script was to scan the formatted SNS message sent from the CC3200 and extract several important data values that we wished to display and use in the main overall flow of the program.

First, we experimented using a hardcoded JSON message, which we performed several operations on to fully understand how to extract the various data fields. This process involves first loading the JSON line into a variable, which we then parse the data fields out of, assigning their respective values to separate variables.

We then at first experimented with simply returning the extracted message but realized that this did not give the correct output that we were expecting. Because of this, we then changed the script to instead, directly publish a message to the AWS SNS topic and thus send an email in the format that we wished it to, rather than simply relying on the native return function.

After this, we changed the AWS Lambda function to instead capture an event message, the message sent from the CC3200 rather than the hardcoded JSON test message and dump that event message into a variable to then parse using the rest of the script.

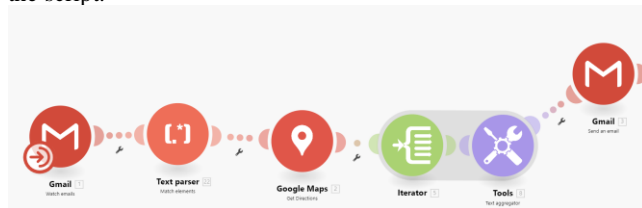


Figure 2: Webhook Structure

After the AWS sends the resulting email to our inbox, we use our make.com webhook to listen on said inbox for the message and filter for the desired subject. Once a valid message is received, the body of the email is then parsed as a key: value pair, with the key being the destination address and the value being the user's current coordinates. These are then fed to the Google Maps API for which we had to generate an API key to use. The results are output as an array of steps including the headsigns, directions, and relative distance per step. These are then formatted as English sentences as "Go in X direction towards Y street for Z distance." This array is then accumulated into a single piece of text, encapsulated in a second email and returned to the sender of the original email.

5. Discussion of Challenges

The most prominent challenge of this lab was effectively polling the NEO-6M GPS peripheral as the constant stream of data was harder to handle and account for than the predetermined data length and transmit time of Lab 3 board to board communication. It took hours of trial and error before we accurately parsed the active line of data within the interrupt handler. This was achieved by maintaining a circular buffer and

generating new strings depending on the length of the desired match relative to the looping buffer index. Another challenge was getting the coordinates into the correct format as the Google API did not accept the NMEA default format. Indeed, even after converting into a format accepted by googlemaps.com it still was invalid for the API. Finally converting to the proper format required hours of trial and error, conversions, and shifts, until it was finally accurate within about 5 meters.

Some of the challenges regarding AWS that we faced in this lab, as previously mentioned, were related to the output messages sent by the AWS. In this regard, we were getting multiple output messages with difficult to use formatting. In fact, this was one of the original reasons why we created the AWS Lambda parser, as the raw JSON message was difficult to utilize.

This issue was essentially solved by us forcing through extra messages built using the correct format and scanning for just those messages. Unfortunately, due to how AWS SNS topics work, we were unable to prevent the sending of the original raw JSON messages, but this issue was solved anyways by simply ignoring any incorrectly formatted messages.

The final challenge of this lab was parsing the data from the email on the webhook side. Parsing the data by comma would be trivial in C, however parsing both from the body of the email required a full exploration of Make.com's provided tools and features until we finally discovered the key: value pair parsing format that we were able to implement.

6. Contribution Breakdown

Thomas handled the AWS portions of the project, whereas Bradley built the circuit, adapted the code from previous labs, acquired, soldered, and communicated with the NEO-6M GPS peripheral, researched and recorded the Davis restaurants into their dictionaries, and developed the Webhook portions of the project.

7. Conclusion

Through the completion of this project for Lab 6, we learned a lot more about integrating the CC3200 with web applications and cloud platforms, as well as utilizing those services to create our own web services. We can also see how such skills utilizing those services will become more and more important in an increasingly interconnected and internet centric world. It was very rewarding and stimulating being able to develop an application of our choice and was very invigorating being able to apply the techniques we learned in this class to interface with a new peripheral completely on my own free of any lab manual.

8. Resources

You Tube Video of Demo:

<https://www.youtube.com/watch?v=EpdjqBHCHPE>

Website for Demo:

<https://daviseats.weebly.com/>

