EENG 499

Water Rocket Telemetry

Brandon Kelly
Advised by Dr. Daniel Cutshall

August 2021-May 2023

1 Introduction

This project is concerned with the creation of a water rocket telemetry system to be used in conjunction with launches at Milligan University. The telemetry system has two primary purposes:
1. Log the altitude and related data.
2. Deploy a recovery method.
This project seeks to fulfill this role through the use of a microcontroller. Primary concerns are weight, ruggedness, reliability, and safety.

2 Terminology

In order to clarify some of the language used the following definitions/ abbreviations are given:
ESP32: A type of microcontroller used to control both the receiver and transmitter. This may also be called microcontroller or µc in this documentation.

3 Background & Problem Statement

The central problem my research seeks to solve is the inability to measure quantitative data from the water rockets used by Milligan University for its summer camps and other engineering activities. The status quo consists of an angle finder which is slow, imprecise, and requires a human to track the rocket. In order to solve this problem, I was asked by Dr. Greg Harrell and Dr. Daniel Cutshall to design, build, and test a system that could log this data and survive multiple launches. This system must be able to log altitude, store it, and then initiate a recovery sequence, in order to protect both the rocket and the user.

4 Electrical Engineering

4.1 Overview and Goals

There were several specific goals in mind for the design of the electrical systems. A primary concern was power efficiency as batteries tend to be heavy and thus by minimizing power consumption we can minimize weight. A modular and relatively off-the-shelf layout was also desired as it allowed for both rapid prototyping and future upgrades. Finally, a user-friendly design focus was utilized for each "client-facing" system. This means that control and logging features are made to be intuitive.

4.2 Architecture

The telemetry system utilizes a point-to-point communication model. This means that the boards communicate directly with each other in a "peer-to-peer" mode. This topology was chosen in order to reduce complexity and to take advantage of the ESP32 features best. The ground module serves to provide the user with the input/output of the system while the module captures sensor data and actuates based on commands received from the ground control module.

4.3 Components

4.3.1 Micrcontroller

The heart of this system is a pair of ESP32 µc. These devices are made by Espressif Systems and were chosen due to their built-in communication capabilities (WiFi, Bluetooth, and Esp-Now) as well as their compatibility with the Arduino IDE and Core. The board can be powered off of either USB or with the 3.3 V/ 5 V pins. Another important feature of this board is its I^2C support, which allows for easy interfacing with the sensors that are attached. For ease of use, two ESP32 "dev boards" were used to build the prototype circuits and as code test beds. In order for the two boards to communicate with each other, the ESP-Now protocol was used. This feature, unique to the family, allows for peer-to-peer communication without typical networking overhead. All programming is done with the Arduino IDE in C. All code can be found in the appendix.

4.3.2 Sensors

Two major sensors were utilized: an MPL3115A2 altimeter and an LIS3DH accelerometer, both on breakout boards from Adafruit. These allow for the capture and logging of critical data such as the altitude reached and the acceleration throughout the flight. This data provides value both for the individual rocket designer attempting to improve their designs, and any faculty who wants a precise way to measure their students' success.

4.3.3 Servo

A servo was added to the flight microcontroller. This was then configured to accept commands from the ground allowing for the deployment of the parachute remotely. It is programmed to have two positions, in order to open and close a latch. Which state the servo is in is controlled by a switch on the ground control box that sets a single bit. This bit is then transmitted to the flight control board.

4.3.4 Ground Control Box

The user will interface with the system using a box with several switches and buttons. This allows the user to control the rocket's data logging and parachute systems. This control box will also contain an SD card writer allowing for quick and easy ingesting of the data.
A concern with the control box is that servo switch(and likely all other controls, all though this behavior was only observed on the servo switch) is very sensitive to noise. This can likely be resolved with using a pull-up resistor and by making sure every component shares a ground.

4.4 Power

The board is powered with a battery connected between a ground pin and the 3.3 V pin. The entire power draw of the system is 370 mW which allows for nearly two hours of battery life using a 3 V battery(CR123).

5 Mechanical Engineering

5.1 Goals and Overview

From a mechanical engineering standpoint, there were several points of emphasis. Similarly to the principles of electrical design used, lightweight was considered a key emphasis. Beyond that, the focus was placed on the ease of use and integration for the students. This means allowing younger students at summer camps for example to be able to quickly add this feature to the rocket, while also allowing upperclassmen to design more complex systems without interference. Finally, a strong focus on durability will be paramount for repeated uses of the system. In terms of design, there are two major components: the capsule itself and the parachute release mechanism. Each of these pieces must uphold the ideas of lightness, modularity, and durability.

5.2 Electronics Mount

One of the major challenges remaining in turning this into a full-fledged solution is the mechanical elements. One of these is the mount. An ideal mount is lightweight and capable of protecting the electronics from impact. Flexible filament might be a viable option but this would require more expertise in materials and modeling. The original prototype is described below: The rocket telemetry device is currently constructed with an all-in-one electronics tray and release mechanism. The electronics tray is 3D printed, with all wire routing done on top of the tray. Future work that could be done to refine this would be creating a routing pattern within the body of the electronics tray. A much more drastic but better solution would also be to design a PCB that contains all of the necessary modules. While durability is a paramount concern, it took a backseat to meet some of our size constraints. Some thoughts for improving the situation would be the addition of cushioning material(poly-fill or foam were both considered but neither have had testing) as well as the addition of a "crumple-zone" which would serve to absorb most of the kinetic energy upon a crash landing. While the prototype has been printed in PLA, some weight savings could be achieved if ABS was used in its place. Due to some difficulties in printing ABS, an alternative weight-saving solution would be to test what the minimum infill required would be. Currently, 20% infill is used as it is a standard value, but this could be reduced. Current weight-saving measures include several pockets being cut out around where no structural material is needed.

5.3 Release System

The rocket is designed with the use of a parachute for recovery. While this makes it easier to achieve certain durability and weight goals, it increases the engineering load quite a bit. In order to solve some of these problems, research was done into more traditional parachute releases. As this work was done, a strong candidate emerged in the use of an elastic material both as a coupling material and as a release mechanism. The basic working of the system involves one piece of elastic permanently attached to the bottle top and another attached in such a way as to be actuated by a servo. Upon release, the remaining piece of elastic will pull off the top allowing for the parachute to be deployed, as well as maintaining a connection with the the rocket. Currently, a modified washer design is being used to couple the elastic to the rocket, but "production" models could be designed such that the student building the rocket will be responsible for including the necessary mounting hardware in their fin design. Another challenge that will need to be addressed is the release mechanism and servo. The current system was designed for a slightly different parachute release, and while it may be possible to modify the system, it may be wise to redo the design altogether.

5.3.1 Parachute

The parachute we are using is one specifically designed for water rockets and will be attached to the main body of the rocket. A streamer or other "drogue" mechanism may be necessary to ensure consistent and complete deployment of the parachute.

6 Appendix

6.1 Adding New ESP32 Microcontrollers.

The first step to adding an ESP32 is to run the MACAddress finder on the new board. This will return a MAC address, which is a persistent alphanumeric code unique to each microcontroller. This code should be recorded, preferably on the board with a label and in an external log such as a spreadsheet. This MAC address will then be placed into the broadcastAddress variable of its paired board.

6.2 Code Troubleshooting

During the development of the tracking and communication code, some difficulties to diagnose appeared. This section lays out the issue and the solution as well as steps taken to find the issue.

6.2.1 'WiFi' does not name a type

In order to create the WiFi network for the ESP32s to communicate with each other a WiFi network needs to be created. An issue occurred with the WiFi library and initializing the Wi-Fi station. This was caused due to an errant brace that prematurely ended the setup function. This is unlikely to occur in everyday deployment but was difficult to diagnose utilizing the error handling in the Arduino IDE. Another possible cause, that would be more likely, is the fact that Arduino has its own Wifi library for use with certain hats but that is incompatible with the ESP32. There may be a more elegant solution if this becomes an issue, but it is usually fine to remove the Arduino library. Mine was located in the x86 Program Files folder for the Arduino but the installation location may vary.

6.3 Datasheets

Links to each datasheet: ESP32: `https://cdn-shop.adafruit.com/product-files/3269/esp32_datasheet_en_0.pdf`
Accelerometer:`https://cdn-shop.adafruit.com/datasheets/LIS3DHappnote.pdf`
Altimeter:`https://cdn-shop.adafruit.com/datasheets/1893_datasheet.pdf`

6.4 Code

6.4.1 Github Repository

The code below was current as of the upload to the Milligan Repository but may have changed since or with further development from other students. The current code base can be found at: `https://github.com/weasal/EENG499BottleRocket`. This repo also contains additional code such as the MAC Address finder code, which allows for new ESP32s to be added as ground or flight modules.

6.4.2 Ground Control Module

```
//Include statements
#include <esp_now.h>//For ESP-NOW protocol
#include <WiFi.h>//For Wifi Communication
#include <Wire.h>//I2C
//#include "SPIFFS.h"
//Sensor libraries
#include <Adafruit_LIS3DH.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_MPL3115A2.h>
#include SD
//Define I2C pins
#define I2C_SDA 21
#define I2C_SCL 22

Adafruit_LIS3DH lis = Adafruit_LIS3DH();

//Declaration of Variables
const int WAIT_TIME = 50;//Wait time between polling
//Variables to hold data from the telemetry transmitter
float pressure;
float altitude;
float acceleration;
float timeFromLaunch=0;
bool deployRec;//Create variable to see if recovery measure should be deployed b

//Creation of structure for reception of data
typedef struct struct_message {
    float pres;
    float alt;
    float acc;
} struct_message;
// Create a struct_message called telem to hold sensor readings
struct_message telem;
```

```
//ESP-NOW configuration
esp_now_peer_info_t peerInfo;
//Address of other board
uint8_t broadcastAddress[] ={0xAC,0x67,0xB2,0xCC,0x44,0x88};

//Functions to check data transmission success
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery
}

//Function to handle data reception
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&telem, incomingData, sizeof(telem));
  Serial.print("Bytes received: ");
  Serial.println(len);
  pressure = telem.pres;
  altitude = telem.alt;
  acceleration = telem.acc;
}

//Setup function
void setup() {
//Creat CSV file
File dataLog = SD.open("/data.csv", "w");
if(!dataLog)
{
  // File not found
  Serial.println("Failed to open test file");
  return;
}
 else
 {
  dataLog.println("Time,Altitude,Acceleration,Pressure");
  dataLog.close();
  }
  // Init Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed␣to␣add␣peer");
    return;
  }
    esp_now_register_recv_cb(OnDataRecv);
}

//Main loop
void loop()
{
  //File dataLog = SPIFFS.open("/data.csv", "w");
  //Check if the switch to deploy recovery measures is toggled
  deployRec = digitalRead(16);
    // Send message via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &deployRec, size
  if (result == ESP_OK)
    {
      Serial.println("Sent␣with␣success");
    }
  else
    {
      Serial.println("Error␣sending␣the␣data");
    }

  //Print incoming data to serial console
  Serial.print("pressure␣=␣"); Serial.print(pressure); Serial.println("␣hPa");
  Serial.print("altitude␣=␣"); Serial.print(altitude*3.281); Serial.println("␣ft
  Serial.print("␣\tVertical␣Acceleration:␣"); Serial.print(acceleration);Serial.
  dataLog.println(timeFromLaunch);
  dataLog.print(",");dataLog.print(altitude);
  dataLog.print(",");dataLog.print(acceleration);
  dataLog.print(",");dataLog.print(pressure);
  dataLog.close();
  timeFromLaunch=+WAIT_TIME;
  delay(WAIT_TIME);//Send data after the defined waiting period
}
```

6.5 Flight Control Module

```
//Include statements
#include <esp_now.h>//For ESP-NOW protocol
#include <WiFi.h>//For Wifi Communication
#include <Wire.h>//I2C library
//Sensor libraries
#include <Adafruit_LIS3DH.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_MPL3115A2.h>
//Define I2C pins
#define I2C_SDA 21
#define I2C_SCL 22

Adafruit_LIS3DH lis = Adafruit_LIS3DH();

Adafruit_MPL3115A2 baro;

esp_now_peer_info_t peerInfo;
const int WAIT_TIME = 50;
// REPLACE WITH THE MAC Address of your receiver
uint8_t broadcastAddress[] = {0xAC, 0x67, 0xB2, 0xCC, 0x43, 0x74};

// Define variables to store BME280 readings to be sent
float pressure;
float altitude;
float acceleration;

// Define variables to store incoming readings
bool deployRec;

// Variable to store if sending data was successful
String success;

//Structure example to send data
//Must match the receiver structure
typedef struct struct_message {
    float pres;
    float alt;
    float acc;
} struct_message;
```

```
// Create a struct_message called BME280Readings to hold sensor readings
struct_message telem;

// Create a struct_message to hold incoming sensor readings

// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast␣Packet␣Send␣Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery␣Success" : "Delivery
  if (status ==0){
    success = "Delivery␣Success␣:)";
  }
  else{
    success = "Delivery␣Fail␣:(";
  }
}

// Callback when data is received

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&deployRec, incomingData, sizeof(deployRec));
  Serial.print("Bytes␣received:␣");
  Serial.println(len);
  Serial.print("Should␣recovery␣be␣deployed(0␣for␣no;␣1␣for␣yes):␣");
  Serial.println(deployRec);
}


void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  pinMode(16,OUTPUT);

  Serial.println("Adafruit_MPL3115A2␣test!");

   Serial.println("LIS3DH␣test!");

  if (! lis.begin(0x18)) {   // change this to 0x19 for alternative i2c address
    Serial.println("Couldnt␣start");
    while (1) yield();
  }
  Serial.println("LIS3DH␣found!");

  // lis.setRange(LIS3DH_RANGE_4_G);   // 2, 4, 8 or 16 G!
  Serial.print("Range = "); Serial.print(2 << lis.getRange());
  Serial.println("G");
  if (!baro.begin()) {
    Serial.println("Could not find sensor. Check wiring.");
    while(1);
  }

  // use to set sea level pressure for current location
  // this is needed for accurate altitude measurement
  // STD SLP = 1013.26 hPa
  baro.setSeaPressure(1013.26);
    // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // Register for a callback function that will be called when data is received
  esp_now_register_recv_cb(OnDataRecv);
}

void loop() {
  if (deployRec==1)
{
  digitalWrite(16, HIGH);//Set pin 13 high to allow for recovery measures to be
  Serial.println("Deploying");
}
else
{
  digitalWrite(16,LOW);//Set pin 13 low to turn off LED
}
pressure = baro.getPressure();
altitude = baro.getAltitude();
//temperature = baro.getTemperature();
sensors_event_t event;
lis.getEvent(&event);
acceleration = event.acceleration.z;
telem.pres = pressure;
telem.alt = altitude;
telem.acc = acceleration;
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &telem, sizeof(tele
delay(WAIT_TIME);//Wait 1 ms to check again

}
```