Tilt Wing UAV with Touch Screen Telemetry Display
EE 474 Final Design Project
With Report Addendum

Team 1
Zewei Du, 1671771
Iqra Imtiaz
Brenan Lundquist, 1342508
Grant Wheatley, 1775505

Instructors: Dr. Rania Hussein

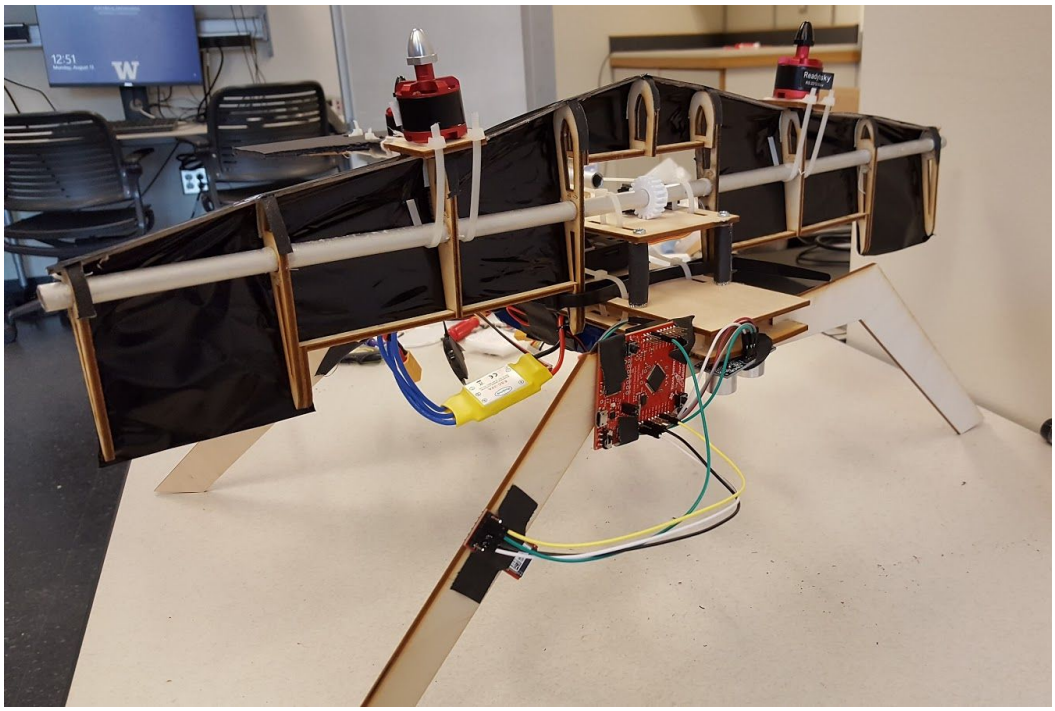

Table of Contents

Addendum Notice:

Please note that an addendum has been added to this report below the original report's references. The original report is left untouched and only information in the addendum has been added.

Abstract:

This lab aims to design an embedded system related project. Our group chose to develop and build a small scale tilt-wing UAV with wireless communication and data display with a phone app. The mechanical build will use the avionics board to control the motions of the drone and TM4C123 boards will be used to receive and transmit data to the Bluetooth, then to the app. We were unable to get data from the Mavlink avionics board so we decided to use other information. The information we are able to transmit including temperate, duration, and ultrasonic data using a ultrasound sensor. After a quarter of studying embedded system with TIVA board, we are able to put together a project with a complete embedded system.

Introduction:

To achieve six deliverables, we seperated project into four main part and each person are responsible for their own part. FreeRTOs is implemented based on previous lab 4 and with the help of freertos_demo project provided with TIVA library. As for the LCD, we replace it with an phone app in order to have the touch feature, data recording and analysing, and front-end display at the same time. Wireless communication is achieved using Bluetooth module as a peripheral with the help of UARTs. Drone counts toward our extra module. For the extra credit - a backend data processing system, it is achieved with logging the data to a text file and then showing the data on screen.

Procedure:

- ### Wireless Communication

We chose to use Bluetooth as our wireless communication. The module used is SparkFun Bluetooth Modern-BlueSMiRF Silver. As the Bluetooth module was used as a peripheral, we use UART for the communication.

Without any code, Bluetooth is connected to PC first and configured through PuTTY. For my PC, the port for Bluetooth is COM14. By opening up PuTTY and typing in $$$, I was able to see CMD command. After typing in X, the list of current configuration is shown. The baud rate is at 9600 which I don't need to change. After checking through all the configuration, I reboot it by typing in R,1. as seen in Figure 1.

Figure 1 : Bluetooth Configuration

After setting up the Bluetooth as described in the above paragraph, I was able to wire the Bluetooth module with TIVA board. I chose to use UART 7 which use PE0 as RX, and PE1 as TX. To make the communication, I wired TX of Bluetooth module to PE0, and RX of Bluetooth module to PE1. Then GND and VCC of Bluetooth module are connected to GND and 3.3V on the TIVA board respectively.

To test the Bluetooth setup, I wrote a sample code which allows the user to type in R, B, G in the PuTTY to change the onboard LED colors. UART7 is configured in the same way as UART0 which we used in lab 3. UART0 is used to take the user input and

transmit, then UART7 is used to print out the received data. The result came out as expected. It confirmed the functionality of the Bluetooth module. (Figure 2)

```
// initialize the UART7
void intialize_UART(int ibrd, int fbrd) {
  SYSCTL_RCGCUART_R |= (1 << 7); // enable uart module
  SYSCTL_RCGCGPIO_R |= (1 << 4); // enable port E
  GPIO_PORTE_AFSEL_R = (1 << 1) | (1 << 0); // enable alternate function 0-rx 1-tx
  GPIO_PORTE_PCTL_R = (1 << 0) | (1 << 4); // configure PCTL
  GPIO_PORTE_DEN_R = (1 << 1) | (1 << 0); // enable the digital function
  UART7_CTL_R &= ~(1 << 0); // disable the UART7
  UART7_IBRD_R = ibrd;
  UART7_FBRD_R = fbrd;
  UART7_LCRH_R = (0x3 << 5); // 8 data bits
  UART7_LCRH_R &= ~(1 << 3); // 1 stop bit
  UART7_CC_R = 0x0;
  UART7_CTL_R = (1 << 0) | (1 << 8) | (1 << 9); // enable UART7, RXE and TXE receive transmit enabl
}


// initiallize the UART0
void UART_Init(int IBRD, int FBRD) {
  SYSCTL_RCGCGPIO_R |= 0x1;     // enable port A
  SYSCTL_RCGCUART_R |= 0x1;     // enable UART0 and clock to appropriate GPIO

  GPIO_PORTA_DEN_R |= 0x3;    // enable the digital I/O
  GPIO_PORTA_AFSEL_R |= 0x3;  // enable analog function
  GPIO_PORTA_PCTL_R =((GPIO_PORTA_PCTL_R & 0xFFFFFF00) + 0x11); // PCTL for UART

  UART0_CTL_R &= ~0x1;      // disable the UART0
  UART0_IBRD_R = IBRD;       // set the integer portion of the baud rate
  UART0_FBRD_R = FBRD;       // set the fraction portion of the baud rate

  UART0_CC_R &= ~0x1;       // set the system clock
  UART0_LCRH_R |= 0x60;     // line control

  UART0_CTL_R = (1<<9) | (1<<8); // RXE and TXE receive transmit enable


  UART0_CTL_R |= 0x1;           // UART enable
}
```

Figure 2: UART0 & UART7 Initialization

- FreeRTOs Temperate Module

The temperature module is designed to record and report back the temperature once per second. ADC0_Init function (changed the trigger of the ADC as processor instead of timer), character function (read and print out the received character), and string function (print out string) are the same as lab 3. And two UART initialization module initializes UART0 and UART7 as talked about in the procedure section of wireless communication. (Figure 3 and Figure 4)

```
void ADC0_Init() {
  SYSCTL_RCGCADC_R |= 0x1;   // enable ADC 0
  for (int i = 0; i < 200000; i++); // to make the delay
  ADC0_ACTSS_R &= ~0x8;  //disable the sample sequencer 3

  ADC0_EMUX_R |= 0x0000; // 0b0101 0000 0000 0000 set processor 0x0 as trigger

  ADC0_SSMUX3_R  = 0x0;   // PE3 as input AIN0
  ADC0_SSCTL3_R |= 0x2;  // end of the sequence
  ADC0_SSCTL3_R |= 0x4;        // sample interrupt enable IE
  ADC0_SSCTL3_R |= 0x8;       // temperature sensor select TS
  ADC0_IM_R  |= 0x8;           // mask interrupt
  // ENABLE_EN0 |= (1<<17);    // enable interrupt
  ADC0_ACTSS_R |= 0x8;       // enable the sample sequencer 3
}
```

Figure 3 :ADC0_Init Module

```
// to display the character
void character(char input) {
  while((UART7_FR_R & 0x20) != 0) {}
  UART7_DR_R = input;
}

// to display information as string
void string(char * string) {
  while (*string != 0){
    character(*string);
    string++;
  }
  character(' ');
}
```

Figure 4:Character & String Module

Temperature task will check for the current temperature and send the data each second using vTaskDelay. And the task will check for the ADC0_RIS register for the data, and update it through UART each second. This code is implemented with reference of professor Jonathan Valvano and Ramesh Yerraballi's E-book chapter 14 [1]. (Figure 5)

```
// keep a track of temperature data
void temperature(void *pvParameters) {
  while(1) {

    ADC0_PSSI_R = 0x0008;            // 1) initiate SS3

    while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done

    double temp = 147.5 -((247.5 *(double) ADC0_SSFIFO3_R) / 4096.0);

    // data type conversion
    // able to print the temperature as string
    int number = (int) temp;
    char temperature[4];
    sprintf(temperature, "%d,", number);
    string(temperature);
/*
    ADC0_PSSI_R = 0x8; //begin sampling on sample sequencer 3
    while((ADC0_RIS_R & 0x8) == 0); //wait for sample seqiencer 3 conversion
    double temperature = (147.5 - (247.5 * 1932) / 4096.0);
    char str[80];
    sprintf(str, "%f", temperature);
    string(str);
    printf("%f", temperature); //Use View->Terminal I/O to view
*/
    ADC0_ISC_R = 0x0008;
    vTaskDelay(300); // hold and check back later, 1 sec
  }
}
```

Figure 5: Temperature Task

- ## FreeRTOs Time Module

Time task is implemented using the same idea as temperature task. The global variable count is used to keep a track of time elapsed since the system begin. And data is converted and send through UART. (Figure 6)

```
// keep a track of time elapsed
void time (void *pvParameters) {
  while(1) {
    count++;
    // data type conversion
    // able to print the duration time as string
    char duration[4];
    sprintf(duration, "%d,", count);
    string(duration);
    vTaskDelay(300); // hold and check back later
  }
}
```

Figure 6 : Time Task

- ## FreeRTOS MAVLink Module

This module was designed to communicate the TIVA board with pixhawk board over MAVLink. Uart0 was used from TIVA and internal UART of pixhawk was used for this purpose. Boards were connected via USB cable. In order to communicate, first UARTs for boards were initialized and two functions were created to request the data and receive that data. This is done because the flow of communication is not continuous. In the request function streams are requested twice per second and then the module wait until there is some data in UART. In the receive function, messages received are decoded using MAVlink protocol. The baud rate for for both boards is selected to be 57600. Unfortunately we were unable to receive any useful data from the decoded messages. This will be talked about in the result section below. The code implemented for this module was reference from Hugues, “MAVLink and Arduino: step by step,” [2] and was modified to use TIVA board in replacement of Arduino.

- ## Ultrasonic Module

In this module, TIVA board is connected to ultrasonic sensor HC-SR04. This sensor calculates the distance of the nearest obstacle. This sensor is placed on the drone so the data collected through sensor can be sent over bluetooth. For this module, interrupts for port A and timer 0 were configured. Port A2 was connected to the echo of the sensor and port A3 was connected to the trigger of the sensor. Both devices were connected by V-BUS and ground after that. The code for this was referenced from YashBansod, “YashBansod/ARM-TM4C-CCS,” [3] and was modified according to our needs.

- Phone App:

To see and record the ultrasound distance data, we wanted to create an android app to communicate with the bluetooth module. Before even attempting to create our app, we tested if our bluetooth module would communicate with a phone by using the Serial Bluetooth Terminal app [4]. Using the same demo for the bluetooth module as described in our Wireless Communication procedure section above, we confirmed that we were able to send information over bluetooth to a phone. We then used MIT's App Inventor website [5] to help us create an app called Drone_Info. We named it Drone_Info since the App Inventor does not allow for app names with a space. Figure 7 below shows a screenshot of the app receiving ultrasound distance data. With our app we are able to connect to our bluetooth module, read ultrasound distance data, and save data to a text file. We also have a map view which we were going to use to show the gps location of the drone with the Mavlink board. Since we were unable to get data from the Mavlink board we kept the map to show what the app would have been like if we were able to get the gps data working as well.

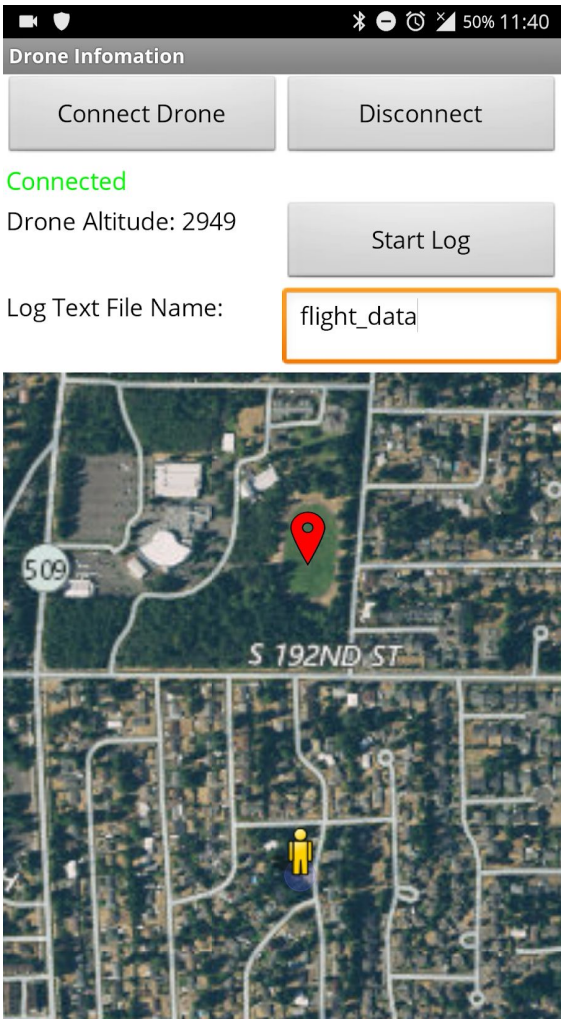

Figure 7: Drone_Info app screenshot

Table 1 below show descriptions of what each user interface component in the app does:

Connect Drone Button	Allows us to select from a list of paired bluetooth devices such as our bluetooth module.
Connection Indicator Text	Shows if a bluetooth device is “connected” in green text, if not the the text is red and reads “Disconnected” (for an example of which, see Figure #).
Disconnect Button	Disconnects the current connected bluetooth device from the app
Drone Altitude	Shows the current ultrasound distance data
Start/Stop Log Button	Allows us to start a log of the ultrasound distance data that will be saved in a text file that the user names. The button text will change to “Stop Log” when we want to stop recording data.
File Name Textbox	Allows us to enter a file name of the log text file to be saved in the Documents Directory on the phone
Map: Red Marker	The red marker was going to update with the gps data of the drone however since we were unable to get the gps data, the marker does not move from the University of Washington fountain
Map: Yellow Person	The yellow person shows the user current position on the map

Table 1: User Interface Component Descriptions

To use the app, first we have to select our bluetooth module using the Connect Drone button. Then once connected and we see the Drone Altitude updating with new values, we can enter in the name of what we want the log file to be called into the File Name textbox. To start logging data, we can press the Start Log button. When we want to stop logging, we can press the Stop Log button and go to the documents directory on the phone and view our log file.

There are a few features included in the app such as a reminder to turn on bluetooth if it is not on, to give a file name if you try to start logging without supplying a file name, and also showing the user the file path to the log file when pressing the Stop Log button.

The log file that our app creates starts out with the date that the data was recorded. Below that is two columns, the first showing the time that the data was received and the second column the actual ultrasound distance data. An example of a log file can be seen below in Figure 8

```
12/08/2018
11:40:44 AM,6
11:40:45 AM,2973
11:40:46 AM,2973
11:40:47 AM,6
11:40:48 AM,3003
11:40:49 AM,2969
11:40:50 AM,2969
11:40:52 AM,2973
11:40:53 AM,2965
11:40:54 AM,6
11:40:55 AM,2964
11:40:56 AM,2964
11:40:57 AM,6
11:40:58 AM,3003
11:40:59 AM,3003
11:41:00 AM,6
11:41:01 AM,6
11:41:02 AM,2949
11:41:03 AM,2949
11:41:05 AM,2964
11:41:06 AM,6
```

Figure 8: Example Log File

Programing an android app using App Inventor is done in two parts, the designer view and the block view. The designer view is for designing the layout of the app and adding modules that the user doesn't directly access like a timer. The designer view of our app can be seen in Figure 9 below.

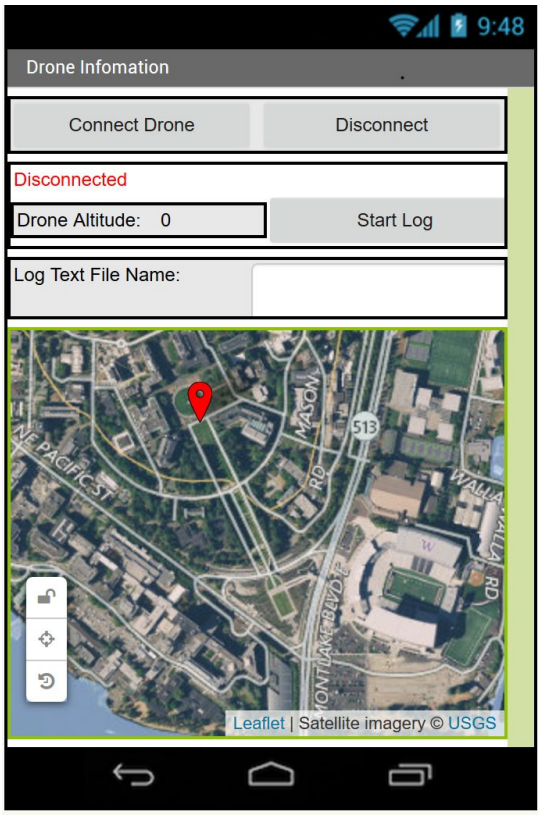

Figure 9: Designer View in MIT App Inventor

The block view is where you program what your user interface components and other modules do. It is named “block” view since all the programing is done in drag-and-drop blocks where your able to build a program as if functions were blocks of lego. To figure out what each block does, we had to read the general documentation [6] provided by App Inventor which gives the documentation of the functions provided. We used the App Inventor documentation on the bluetooth functions [7] to be able to connect to our bluetooth module and ready data when available. The block view of our app can be seen in Figure 10 below.

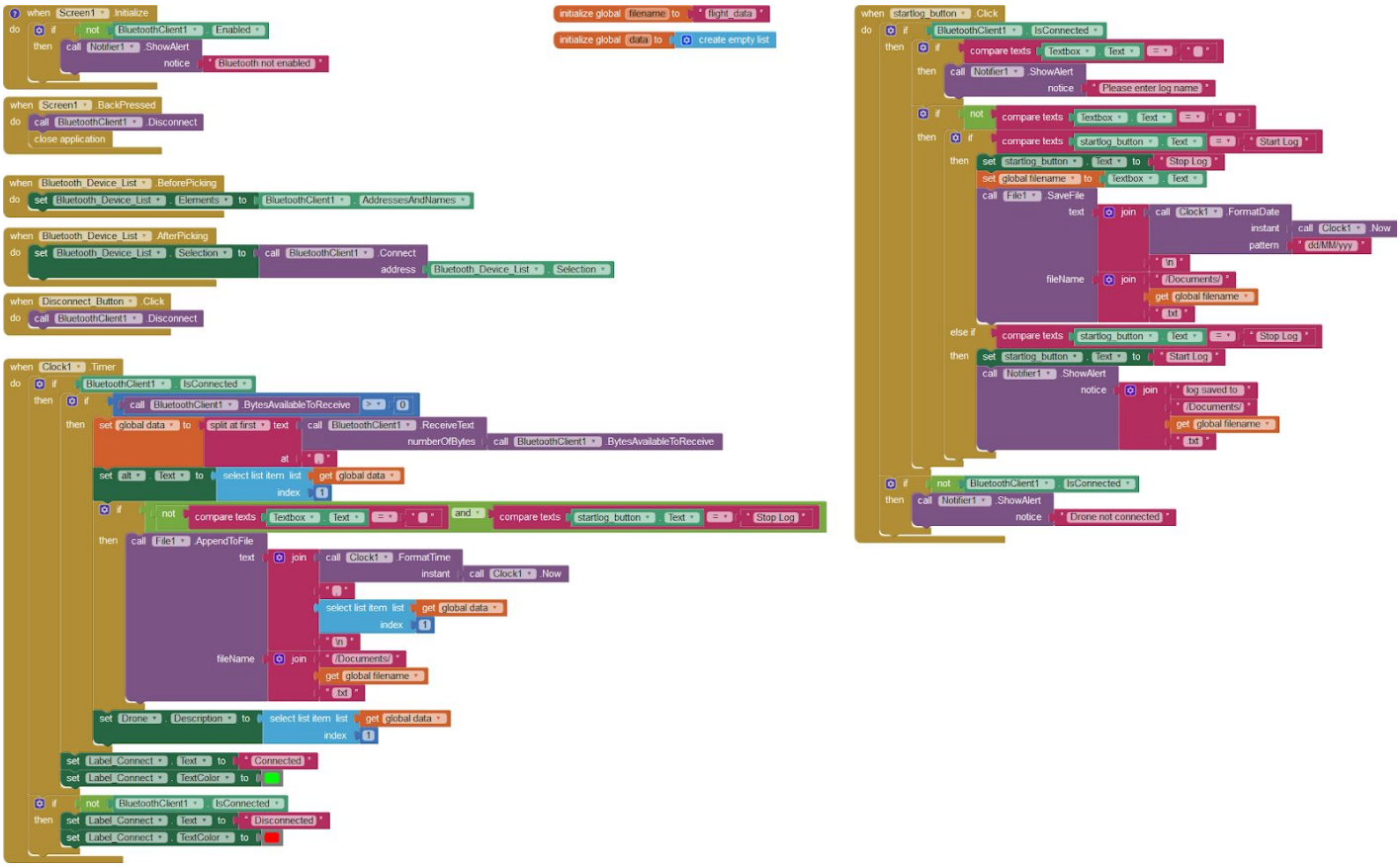

Figure 10: Block View in MIT App Inventor

To describe what is going on in block view we will try to generalize what is going on. Please note that we are going to use the phrasing such as “the startlog_button click block” to refer to everything inside the scope of “when startlog_button is clicked” function.

- The Screen1 Initialize block happens when the app is opened and if the user has not enabled bluetooth on their phone a message appears on the app saying that bluetooth is not enabled.

- The Screen1 BackPressed block happens when the user presses the back button on their phone which will disconnect the bluetooth device that is connected and close the app.

- The Bluetooth_Device_list BeforePicking and Bluetooth_Device_list AfterPicking blocks refer to when the Connect Drone button is pressed which brings up a list of paired bluetooth devices to connect to and will connect the one that the user selects.

- The Disconnect_Button Click block happens when the user presses the Disconnect Button which simply disconnects the connected bluetooth device.

- The startlog_button Click block controls creating a log file and starting/stopping logging data. It checks if the user has made a bluetooth connection before logging, if the user has entered in a file name before logging, creating a log file with the current date at the top, changing the name of the button between "Start Log" and "End Log", and showing the user the path to the log file when the logging is stopped. The file name the user entered is saved to a global variable named "filename".

- The Clock1 Timer block controls receiving the data over bluetooth, displaying the data, and actually logging the data every second. It changes the connection indicator text based on if a bluetooth device is connected, receives the data saves it a "variable" (which is actually an array) called "data", prints the data to the screen, and appends the log file with the current time and the data.
    - Since UART can only send one character per time, we use a comma after each data value as a "stop/end bit". When receiving the data over bluetooth, we split the text at the comma into two and stores both into a array which is our data variable. Then when we want to access the actual data value we just access the first value in our data array.

Our original idea for the app was to also have the gps data from the Mavlink board. As seen in Figure 7 above, the red marker would update every second with the Clock1 timer along with the user position which is given by the little yellow person. That way the user would be able to track the drone while knowing where they were as well. The gps data of the drone would have also been included with our log file. Another feature we would have had would be when the user starts logging data, a red line would follow the drone marker so that the user could quickly see the flight path of the drone while logging. We wanted to keep the map and marker to show what would have been if we had the gps data so the red marker does not move from the University of Washington fountain.

- Drone:

The drone that we chose to build for this project was a vertical take-off and landing (VTOL), tilt wing tricopter. The front wing carries two motors, one on either side. A third motor sits on the tail. The drone takes off under power of all three motors with the wing pointed upward (Fig. 11A). As the drone climbs and gains speed, the wing tilts forward, shifting the thrust vector forward and enabling the wing to generate lift. The rear motor can then power down, and the drone flies like a twin-motor, fixed wing plane (Fig. 11B).

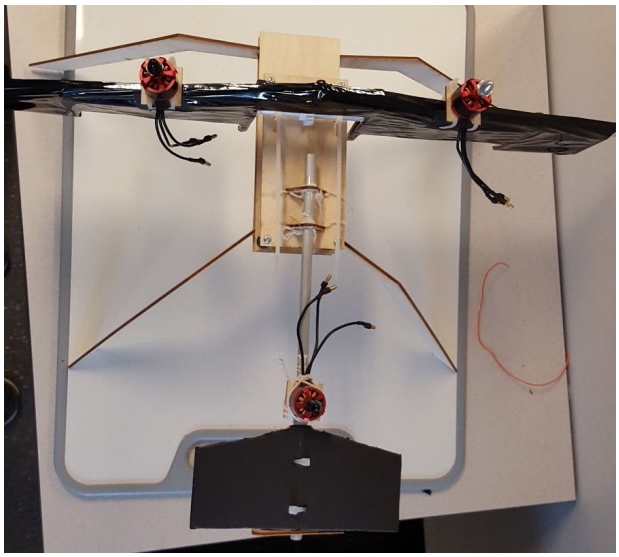

Figure 11 (A)

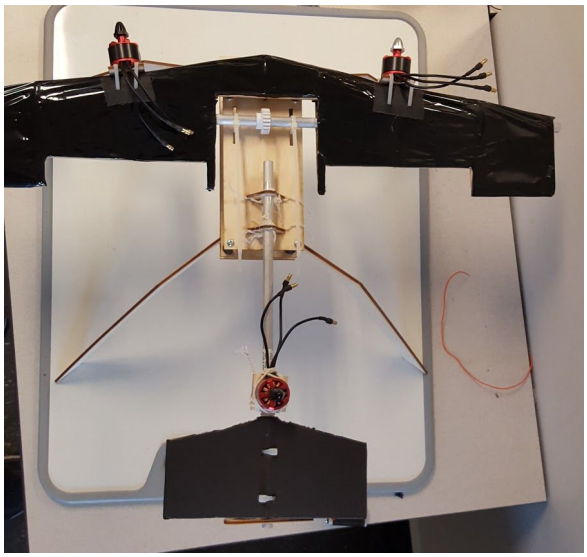

Figure 11 (B)

The drone is controlled by the open source Pixhawk autopilot. The autopilot contains sensors such as a gyroscopes, magnetometers, and accelerometers. The autopilot uses data from the on-board sensors and control inputs to control the drone's attitude, speed, and servo positions by varying the PWM outputs to each motor and servo. The servos are controlled directly by PWM signals, and the motors are controlled by electronic speed controllers (ESCs) that interpret the PWM signals provided by the autopilot. The ESC maps the provided PWM signals (typically ranging from 1000 µs to 2000 µs) to an amount of power to deliver to the motor. This power is tri-phasic, alternating the polarity of the three coils inside the stator. The ESCs are also necessary in order to isolate the 5 V control components from the high current, 12 V power provided by the lithium-polymer battery. The ESCs also operate as BECs, providing 5 volts to the autopilot. The autopilot then distributes power to each of the control components. The TIVA and other additional modules required more power than the autopilot can comfortably deliver, so a separate 5 V BEC was used. Figure 12 shows the full wiring diagram for the drone's electronics.

TILTWING WIRING DIAGRAM

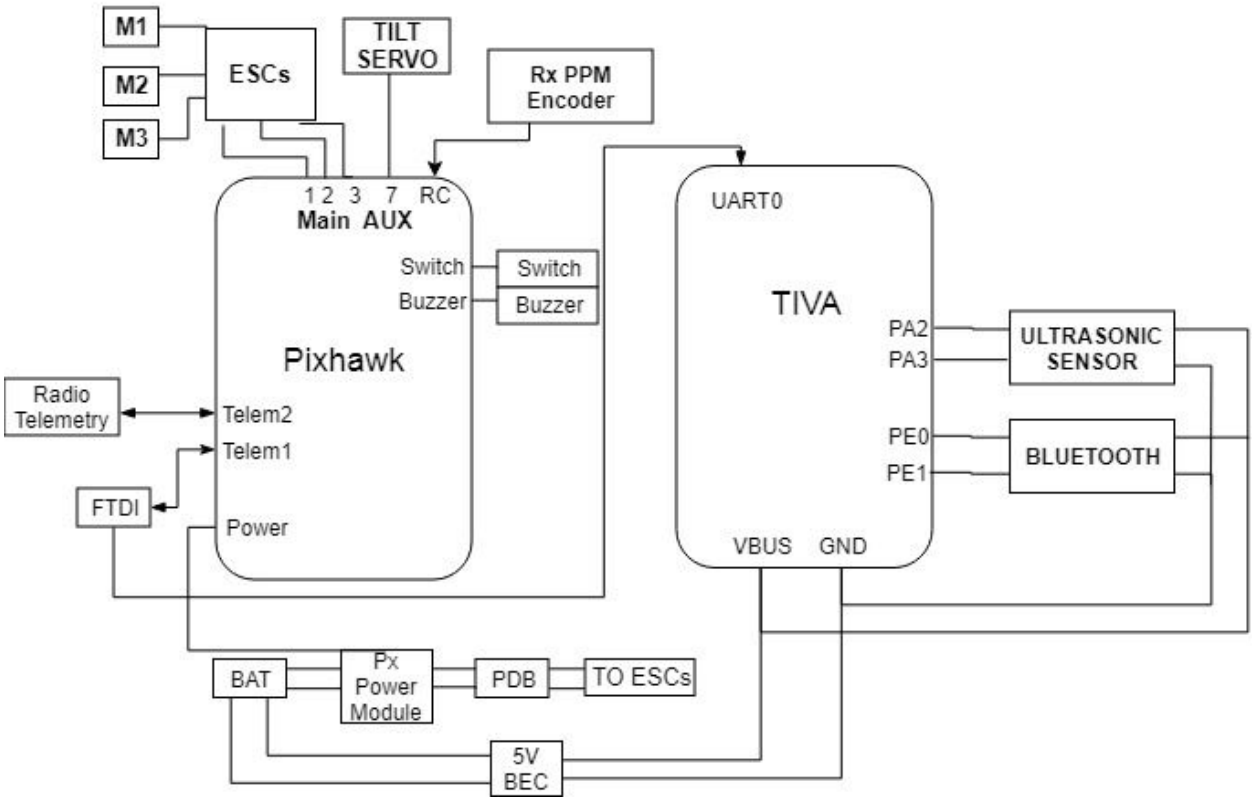

Figure 12. Drone Wiring Diagram

The firmware used was a modified version of the native tricopter firmware available for the Pixhawk. The pixhawk firmware provides support for VTOL functions and the ability to adjust control methods mid-flight. The Pixhawk can be configured for nearly all conceivable airframe types by modifying the hundreds of available parameters. It is possible to directly alter the source code to provide additional functionalities, however, this was not considered necessary for this project. Future iterations of this project could benefit from the addition of a dedicated PID to better control the motor speeds during the tilting of the wing.

To illustrate our firmware design method, some of the critical parameter changes made to enable this project are outlined in Table 2. All values were drawn from the list of parameters in the Pixhawk developer's guide datasheet [8]. Our method was to begin with the tricopter firmware package, and then import the quadplane firmware package. The quadplane firmware allows for rotors to be defined as operating in the vertical direction, and others as operating horizontally. We configured the rear motor for vertical operation, and the front two for both vertical and horizontal operation. We then defined different PWM ranges for the front motors depending on their operation direction. We then had to enable a wing tilt servo, with operation angles, PWM

constraints, and define a control feedback method. Finally, we had to make considerations for the transitions from hover to forward flight, and from forward flight back to a hover. The most crucial were to set an angle where the vehicle would rapidly transition forward, and delaying the tilt-back to allow time for the rear-rotor to spin back up.

Q_ENABLE	1	Enable quadplane support
Q_FRAME_CLASS	7	Define for tricopter
Q_TILT_MASK	3 (motor 1 + motor 2)	Applies a bitmask that defines which motors can tilt
Q_THR_MIN_PWM, Q_THR_MAX_PWM	985 1750	Define distinct PWM ranges for the tilting motors
Q_TILT_TYPE	1 - Binary 2 - Vectored	Define tilt type to binary for direct horizontal to vertical tilting. Define as vectored for steady tilt-over (requires an additional PID curve0
SERVO6_FUNCTION	41	Define for single tilting control with auxiliary channel 6
Q_TILT_MAX	30	Set maximum tilt in degrees before vehicle will transition to horizontal flight
Q_TILT_RATE_UP Q_TILT_RATE_DOWN	25 10	Max speed (in degrees per second) that the wing will tilt
UART_ESC_MOTOR1 UART_ESC_MOTOR2 UART_ESC_MOTOR3	1 2 3	Map channels for motor control
Q_R_TILT_DEL (FLOAT)	3	Set a delay to allow for the rear motor to spin up before reverse tilt

Table 2: Notable parameter settings for VTOL tilt wing

Result:

- ## Outcome:

MavLink:
Unfortunately we were unable to receive any useful data from the decoded messages from the MavLink board. While using PuTTY to receive the decoded messages over UART, we were able to get Unicode Block Elements such as ▓ and other random characters. However we were able to get a consistent pattern of the random characters which we believe that we were able to get the messages from the MavLink but the data was either being corrupted or something was wrong with our decoding. This led us to switch to using a ultrasonic sensor to receive distance data from the drone to send to our phone app over bluetooth.

Ultrasonic Module:
The ultrasound sensor is able to get distance data and send the data to the bluetooth module.

Wireless Communication:
The Bluetooth module is able to transmit and receive data such as temperature and duration with the help of UART.

Phone App:
We were able to create a phone app as seen in Phone App Procedure section above. We were able to receive data and log data coming from our bluetooth module. Although programming with MIT's App Inventor [1] is done with premade functions and lego-like blocks, we still had to use our knowledge of embedded system design to do so. Programming the app was similar to creating interrupts, for example when a button is clicked in our app is similar to our lab 2 and lab 3 when we had to make a interrupt service routine for when a button is pressed or a timer expires.

Drone:
While all initial ground tests proved favorable, an ESC suffered a burnout during the initial flight test, preventing any testing of the tilt mechanism in flight. All ESCs functioned regularly during ground testing, however, the loads experienced during ground tests are significantly lower than in flight testing. What appeared to be a manufacturing defect in the ESC caused it to short and catch fire while attempting the first hover. The process of refining the parameters took a very long time, however, the ability to dramatically alter the behavior of the vehicle by simply changing values in a table - without having to edit the source code - made for much easier configuration. While it was disappointing to not be able to fully explore the behaviors and capabilities of the tilt wing drone, we plan to replace the destroyed ESC and continue its development.

After replacing the ESC, we were able to conduct initial flight tests. Please see the report Addendum for more information.

- Videos:

  A short video about our project can be found at: https://youtu.be/BtA0IoUXWHw
  The hover test can be found at: https://youtu.be/H31C7qKiLSM
  The LCD and Android app implementation: https://youtu.be/UoTJ3Q_YSnA

Conclusion:

From this project we learned how to integrate our current embedded system knowledge with some new knowledge to create something completely new. Although we could not get data from the Mavlink avionics board, we were still able to get some useful information directly from the Tiva board. This final project does show us what it will be like when we leave this class to designing and making a project from scratch. One of our favorite things about this project was learning about the bluetooth module and making the phone app since it showed us how "smart home" appliances could function and could be made. We hope that we can continue working on projects like these with being able to send even more types of data to our phone app or focus on being able to control a drone from our app. All in all this final project reinforced the idea that although we have learned a lot about embedded systems, there is always more to learn and discover in future projects.

Project Contribution:

The project was divided up as seen in Table 3 below

Component of Project:	**Assigned to:**
Wireless Communication	Zewei Du
FreeRTOs Temperate Module	Zewei Du
FreeRTOs Time Module	Zewei Du
FreeRTOS MAVLink Module	Iqra Imtiaz
Ultrasonic Module	Iqra Imtiaz
Phone App	Grant Wheatley
Drone (Mechanical, Electrical, and Programming)	Brenan Lundquist
Video	Each component in the video was done by whomever that component was assigned.

Table 3: Project Contributions

Reference:

[1] "Chapter 14: ADC, Data Acquisition and Control," *Brian L. Evans*. [Online]. Available:http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C14_ADCdataAcquisition.htm. [Accessed: 07-Aug-2018].

[2] Hugues, "MAVLink and Arduino: step by step," ArduPilot Discourse, 06-Feb-2018. [Online]. Available: https://discuss.ardupilot.org/t/mavlink-and-arduino-step-by-step/25566. [Accessed: 14-Aug-2018].

[3] YashBansod, "YashBansod/ARM-TM4C-CCS," GitHub. [Online]. Available: https://github.com/YashBansod/ARM-TM4C-CCS/blob/master/TM4C123G LaunchPad Ultrasonic HC-SR04/main.c. [Accessed: 14-Aug-2018].

[4] Kai Morich. (2018). Serial Bluetooth Terminal (1.17) [Mobile application software]. Available: http://play.google.com/store/apps/details?id=e.kai_morich.serial_bluetooth_terminal&hl=en_US [Accessed: 07-Aug-2018]

[5] "MIT App Inventor", MIT App Inventor, 2017. [Online]. Available: http://appinventor.mit.edu/explore/. [Accessed: 07-Aug-2018].

[6] "Component Reference", MIT App Inventor, 2017. [Online]. Available: http://ai2.appinventor.mit.edu/reference/components/. [Accessed: 07-Aug-2018].

[7] "Connectivity Components", MIT App Inventor, 2017. [Online]. Available: http://ai2.appinventor.mit.edu/reference/components/connectivity.html#BluetoothClient. [Accessed: 07-Aug-2018].

[8] "Parameter Reference", Pixhawk Development Guide, 2018. [Online]. Available: https://dev.px4.io/en/advanced/parameter_reference.html[Accessed 5-Aug-2018].

Report Addendum:

After submitting the final project report there were a few changes to our project. Instead of using ultrasound data we decided to change to using temperature data from the Tiva's internal temperature sensor. This was due to having to also include freeRTOS into our system. We also added an LCD screen to enable and disable data to be sent over bluetooth. Due to the changes, the phone app also got updated to be focused on temperature data. We also replaced the burnt out ESC and were able to begin flying the drone.

- New Data and LCD Screen:

Figure 13: Bluetooth Data Disabled

Figure 13 above shows our new system but not on the drone. The blue "Bluetooth Button" enables and disables data to be sent over the bluetooth module. The status of which can be seen on the left side of the LCD where it says disabled in Figure 13 above and enabled in Figure 14 below. Below the status shows the temperature if enabled, or "00" if disabled meaning no new temperature data is being sent.

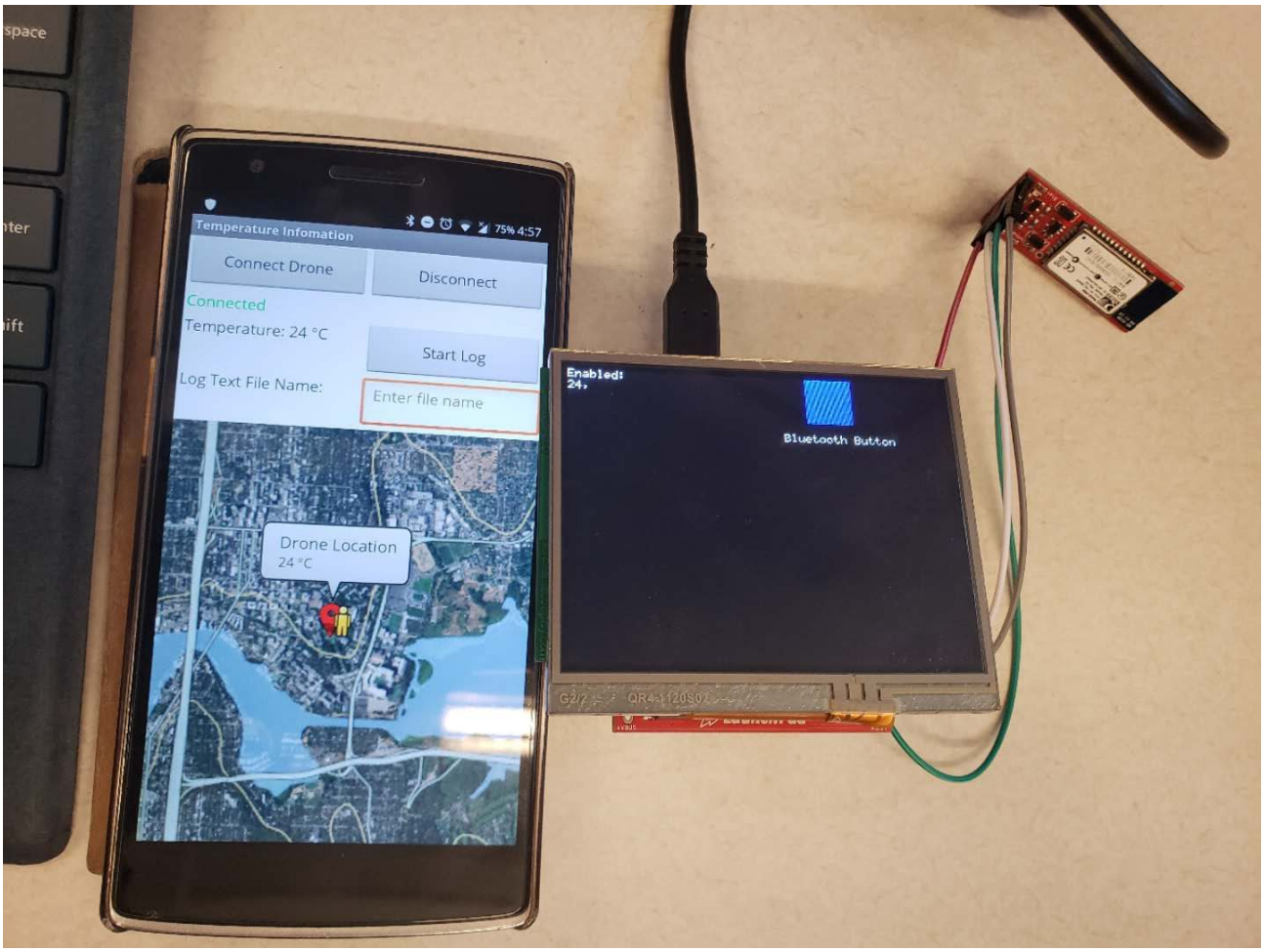

Figure 14: Bluetooth Data Enabled

On the left of Figure 13 and Figure 14 above you can see the updated phone app to be focused around temperature. It works in the same way described in Phone App section above. In Figure 13 and Figure 14 above you are now able to see the information box if the user tapped on the red drone marker. This was originally going to show the GPS coordinates of the drone but now we are showing that we are able to update the information box in real-time such that we would be able to update the GPS coordinates to the information box as well in real-time.

Zewei Du created the freeRTOS code and tasks for the LCD screen and the temperature data gathering.

- Drone Flight:

After replacing the burnt out ESC, the replacement ESC promptly caught fire and it too burnt out. However, a third ESC was installed and it functioned perfectly. The original, broken wing tilting assembly was replaced, however, it was poorly designed and broke during ground testing. In order to at least do some flight testing, the wing was fixed in the ‘up’ position. The drone was able to leave the ground and hover, as shown in the hover test video in the Videos section of this report.

The vehicle suffered from two main flaws, pointy landing gear, and manually calibrated rear rotor tilting. The pointy landing gear dug into the ground on takeoff, resulting in an unpredictable and asymmetrical takeoff; the drone would often flip before fully leaving the ground. Rear rotor tilting is necessary in order to compensate for the yaw drift inherent in tricopters. Two rotors rotate clockwise, and one rotor rotates counterclockwise. Pitched propellers work by hitting the air at an angle, forcing air that is next to blades to be pushed down. This action imparts not only a lifting force on the vehicle, but a sideways force at the propellor blade. The unbalanced rotor configuration (biased clockwise) results in a net counterclockwise torque as a reaction to more air being pushed clockwise. To compensate, the rear rotor must be tilted in the opposite direction, providing a component thrust vector to counter the yaw tendency. Our yaw compensation was passive, having to be 'calibrated' through trial and error by manually adjusting the angle of the rear rotor. The two problems compounded, and every time the drone flipped the rotor angle was disturbed and had to be re-calibrated. In order to solve these issues, the landing gear will be redesigned to utilize either skids or wheels, and a servo will be used for active rear-rotor tilt control and yaw stabilization.

While we were unable to properly test whether or not we could create a tilt wing drone, we came very close; we were hindered only by faulty parts, our own incompetence in mechanical design, and time. In our attempt, we at least managed to create something that can get off the ground, and will definitely continue this project in order to attain full, VTOL, tilt wing flight.