Autonomous Raspberry Pi Drone with Implementation of Apache Flink

Richard Chear, Robert F. Goodbar, Jaemin Kim, Khaled El Masri, Nhut-Linh Ngo, and Mohamed E. Aly
{rchear, rfgoodbar, jaeminkim, kelmasri, nhutlinhngo, mealy}@cpp.edu

Electrical and Computer Engineering Department
California State Polytechnic University, Pomona
Pomona, California

***Abstract*— Apache Flink is a framework for stateful computations over unbounded and bounded data streams. This allows for streams of data to be sent to the Flink cluster to be computed and transformed in real time. With the help of Emlid's Navio2, an autopilot HAT for Raspberry Pi, the drones are able to send real-time sensor data that are on-board to the Flink cluster to allow for monitoring the status of each drone.**

***Keywords*— Apache Flink, autonomous drone, distributed computing, Linux, Raspberry Pi, rechargeable battery**

I. Introduction

Apache Flink was initially released back in May 2011 and was developed by the Apache Software Foundation. The core engine of Apache Flink is written in Java and Scala and it is used throughout the tech industry for its high throughput, low-latency, and its ability to process data parallely and in pipelined-manner.

Raspberry Pi (RPi) is a single-board computer that utilizes Linux as an operating system and developed in the United Kingdom in association with Broadcom. Due to its ease of use, its ability to connect to a Windows machine via PuTTy, and its versatility are the reasons for using Raspberry Pi as the main component for the drones.

II. Drone

A. Components

General components needed to build a drone are as follows:

1. Frame
2. Brushless DC motors
3. Electric Speed Controllers (ESC)
4. Propellers (Props)
5. Telemetry module
6. Lithium Polymer (LiPo) battery

For the frame, it was required to get a design that will allow for all the components to be on-board. Readytosky S500 Quadcopter Drone Frame provided plenty of space throughout the frame, while maintaining its sturdiness and keeping the weight relatively low. Weighing in at just 16 ounces (28.3495g), it allowed us to put all the components needed.

As for the brushless DC motors, Crazepony's EMAX RS2205 2300KV, along with 30A RC Brushless Motor ESCs, RAYCorp's 1045 10 inch props and Turingy's Nano-Tech 4000mAh 4S 30C LiPo battery were used. The combination of these components allowed for the drone to lift off, while providing sufficient amount of current and voltage required to keep the drone in flight.

Along with the components mentioned above, there were additional components that were implemented such as XT60 plugs for connecting the LiPo battery to four ESCs, Matek's Power Distribution Board (PDB) with Battery Elimination Circuit (BEC), Raspberry Pi camera module, and GPS module.

B. Thrust to Weight Ratio

While gathering the components required for this project, it was critical to keep thrust to weight ratio in mind. During the flight, an aircraft experiences forces from four different sources: lift, weight, thrust, and drag [1]. From Newton's second law of motion,

$$F = ma$$

where F is the amount of force that is being applied, m is the mass of an object, and a is the acceleration. Weight is calculated by

$$W = mg$$

where W is the weight of an object, m is the mass of an object, and g is the gravitational constant. Substituting $m = W/g$ into the force equation results in

$$F = \frac{Wa}{g}$$

which can be simplified to

$$\frac{F}{W} = \frac{a}{g}$$

which represents the thrust to weight ratio and "it is directly proportional to the acceleration of the aircraft" [1].

The total mass for components used for this project is

Frame	28.3495
ESC (4)	128
DC Motor (4)	116
Props (4)	40
Telemetry module	13.8913
LiPo battery	392.24
Raspberry Pi (RPi)	49.895
RPi camera module	3
Navio2 w/ GPS	50
Miscellaneous	121.4032
Total (g)	942.779

Assuming that $g = 9.81\ m/s^2$, the amount of force that Earth's gravity exerts on the drone is 9.248N.

The amount of thrust that a quadcopter can generate can be calculated by

$$m = \frac{T}{g} = \sqrt[3]{\frac{\frac{\pi}{2} D^2 \rho P^2}{g}}$$

where,

$$T = thrust\ (N)$$

$$D = prop\ diameter\ (m)$$

$$\rho = density\ of\ air\ (1.225\ kg/m^3)$$

$$P = Power\ of\ rotor\ (W)$$

$$m = equivalent\ mass\ of\ thrust$$

$$g = gravitational\ constant\ (9.81\ m/s^2)$$

In order for the drone to be able to take off, the total amount of thrust provided by rotors must exceed the value.

According to the datasheet provided by the mtor manufacturer, the amount of thrust each motor exerts at 30A using 5 inch props is 1024g. Since the props that were implemented to this project are 10 inch props, the thrust will be greater than 1024g. Estimating that the amount of thrust provided by the 10 inch props to be between 1700-2000g, at its maximum of 30A drawn from each motor, the total equivalent mass of thrust provided by 4 rotors is about 7400g. Which results in

$$equivalent\ mass\ of\ thrust / total\ mass\ of\ quadcopter$$

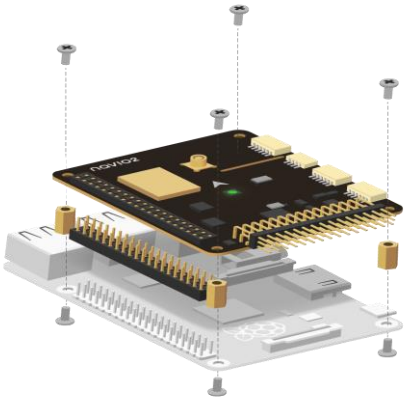
$$= 7400/942.779$$

$$= 7.85N$$

The total amount of thrust needed in order for the drone to take flight came out to be 1.428N. Since the amount of thrust provided by the rotors exceeds that amount, with the chosen components, the drone is able to achieve flight with high acceleration.

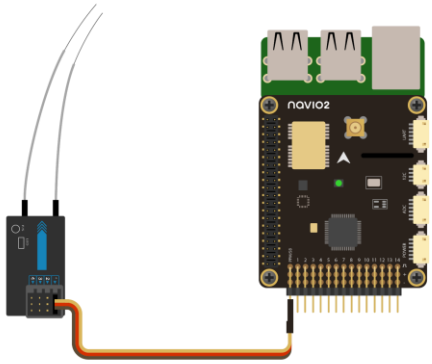
C. Wiring Diagrams

Attaching Navio2 on RPi is simple; by attaching the provided screws and header extensions on the RPi, the installation process is complete.

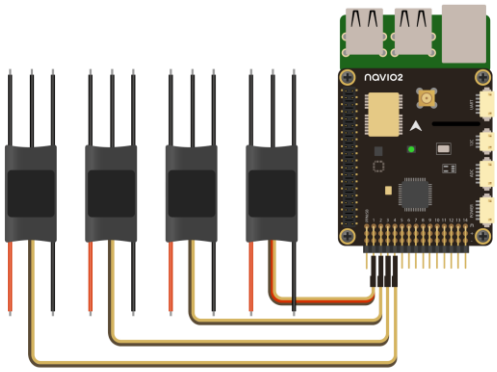


To attach the transmitter that is responsible for receiving RC input from the RC controller, the SBUS was connected with the PPM/SB rail. This was achieved by using three female-to-female

jumper wires to connect the signal, positive and negative terminals.

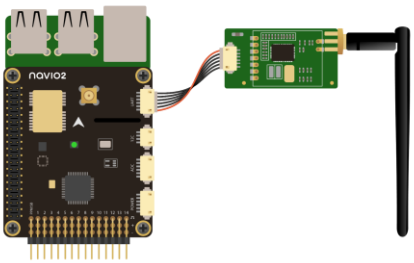


Connecting the ESCs to the Navio2 can be achieved by simply attaching the jumper wires from the ESCs to the servo rails on the Navio2.

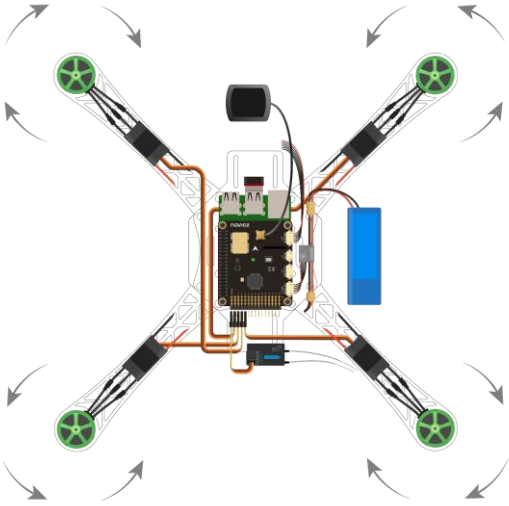


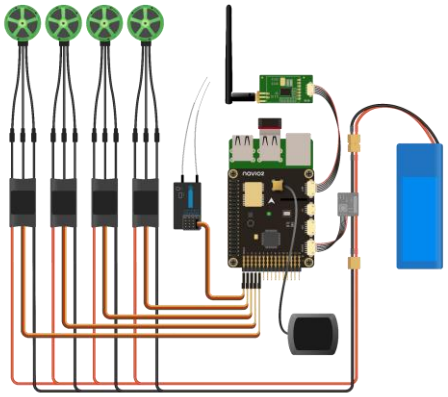
The servo rails on the Navio2 are as follows: data line, +V, and ground from top to bottom.

To connect the telemetry module, the UART port is utilized. The telemetry module is responsible for transmitting and receiving data between the drone and the ground station.



For the quadcopter setup, the final wirings should reflect the images provided by Emlid's website.





D. Setting up Emlid Raspbian OS

Navio requires preconfigured Raspbian with ROS, which Emlid provides. To install Raspbian with ROS, simply download the image provided on Emlid's website, and use Etcher to flash the microSD card with the OS.

This version of Raspbian is preconfigured to be headless, which means that Graphical User Interface (GUI) is not enabled since it is not required for drone applications. Upon successfully flashing the microSD card with custom Raspbian and powering on the RPi, configuring the network is required.

To configure the network, it is required to use a terminal emulator such as PuTTy. PuTTy allows the use of a Windows machine to connect to the RPi via Secure Shell Protocol (SSH).

After connecting to the RPi via SSH, running the command

sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

will allow the user to edit the network settings and will be able to set the WiFi configurations. The user should and need to reboot the Raspberry Pi for the settings to take effect.

After the system reboots, typing the command

sudo emlidtool ardupilot

will display a menu that allows the user to choose the type of vehicle, the version of the vehicle, enable on-boot start of the ardupilot, and manually start and stop ardupilot.

To specify launching options, assuming the vehicle is a copter, the user can run the command

sudo nano /etc/default/arducopter

From there, the user can specify the IP address of the ground station and define the telemetry modules. The telemetry modules have parameters and its definitions are provided by Emlid and are as follows:

- -A- serial 0 (always console; default baud rate 115200)
- -C- serial 1 (normally telemetry 1; default baud rate 57600)
- -D- serial 2 (normally telemetry 2; default baud rate 57600)
- -B- serial 3 (normally 1st GPS; default baud rate 38400)
- -E- serial 4 (normally 2nd GPS; default baud rate 38400)
- -F- serial 5

To ensure that the OS is up-to-date, it is recommended to run the command

sudo apt-get update && sudo apt-get upgrade

III. Apache Flink & Ardupilot

A. What is Apache Flink

Apache Flink is a framework and distributed processing engine that computes data through stateful computations over bounded and unbounded data stream. Bounded streaming has a defined start and end, it can be processed by intaking all data before performing any computation. Unbounded Streaming has a start but no end, it does the computation while receiving in new data streams[3]. The advantage of Apache Flink is that it's deployable anywhere, is scalable to any size and has real time parallel computing . This is a great platform for controlling the drones using master and slave protocol, where the drones will continuously stream data from each other and be given command by the master drone. Since it's also scalable theoretically there's no limit to how many drones that can operate with each other. Real time parallel computing is important in any large cluster environment, as it greatly reduces any bottleneck.

B. Downloading and Setting Up Apache Flink

Apache Flink can be run on any Mac, Windows and Linux operating system. The prerequisite to getting Flink running is at least having java 8 installed. To download and install flink you can go on their website at: https://flink.apache.org/downloads.html or "wget https://mirrors.advancedhosters.com/apache/flink/flink-1.13.0/flink-1.13.0-bin-scala_2.11.tgz " on a linux machine. Once installed to start up a local cluster navigate into the flink directory and use the command "./bin/start-cluster.sh" to start a local standalone cluster. From there you can use a provided example that comes when you install Flink. Inorder to do any meaningful work on flink you have to create a Java ARchive file (jar file), using the Intellij compiler is recommended. Make sure to include all the Maven dependencies as a prerequisite. To do this when creating a new project quickselect the "Maven" and click "Create from archetype", this will create a sample Flink project that will handle all dependencies for the setup. If you don't see the Archetype then click on the Add Archetype and under the following values

change “Groupid = org.apache.flink” and “ArtifactId = flink-quickstart-java”. An important note is in the “pom.xml” the Flink dependencies are specified as “provided” meaning that they are not packaged with the binaries and will use the dependencies package from the Flink cluster, this will give you an error when running in local mode. To fix this go to the “modules” in project structure and change all dependencies to Compile.

C. *Apache Flink Code*

Initially the Flink code was created to read a CSV file from Ardupilot and process the data there. The first step is to include all the Flink libraries for the program.

```
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.tuple.*;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.datastream.DataStreamSource;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.util.Collector;

import java.util.Arrays;
import java.util.List;
```

Then we want to initialize the execution environment for Flink, this sets up the environment for flink to read files rather than socket streaming.

```
final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
```

Then the code checks for input arguments with

```
final ParameterTool params = ParameterTool.fromArgs(args);
```

We create a tuple of set 4, then read the CSV file from the specified path. Ignoring the first line, then we want to parse the strings into different data class. In this case the first data type is a String, specifying if it’s the Acceleration,Gyrometer or Magnetometer Data being passed. The second through fourth are Double as they are the actual values. Lastely, print out to system comcil with “ System.out.println(); ” and execute the environment with “ env.execute(); ”.

```
DataSet<Tuple4<String,Double,Double,Double>> AccelData
        = env.readCsvFile( filePath: "/home/pi/Navio2/Python/output/AccelGyroMag.csv")

        .ignoreFirstLine()
        .parseQuotedStrings('\"')

.types(String.class,Double.class,Double.class,Double.class);
System.out.println("\n***********************************************************");
System.out.println("Accel data from file");
System.out.println("-------------------------------------------------------------");
System.out.println("\nAccel Data = " + AccelData);              //Prints out to system
env.execute( jobName: "Accel data");
```

The whole code is provided here:

```
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.tuple.*;
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.streaming.api.datastream.DataStreamSource;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.util.Collector;
```

```
public class Flink_CSV {
    public static void main(String[] args) {

        try {
            System.out.println("\n***********************************************************");
            System.out.println("Starting the Basic Transformation program....");
            System.out.println("-------------------------------------------------------------");

            /**************************************************************
             *                  Read CSV file into a DataSet
             **************************************************************/

            final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();        //Create the execution environment for Flink
            final ParameterTool params = ParameterTool.fromArgs(args);
            env.getConfig().setGlobalJobParameters(params);
            DataSet<Tuple4<String,Double,Double,Double>> AccelData                                  //Creates a tuple set of 4
                    = env.readCsvFile( filePath: "/home/pi/Navio2/Python/output/AccelGyroMag.csv")      //Reads the CSV file from path


                    .ignoreFirstLine()
                    .parseQuotedStrings('\"')

            .types(String.class,Double.class,Double.class,Double.class);
            System.out.println("\n***********************************************************");
            System.out.println("Accel data from file");
            System.out.println("-------------------------------------------------------------");
            System.out.println("\nAccel Data = " + AccelData);                //Prints out to system council
            env.execute( jobName: "Accel data");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

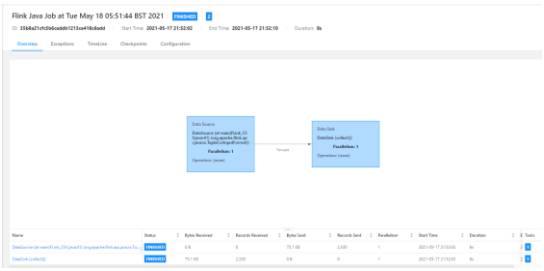
Create a jar file from the code and move it to the desired directory on the PI. We use FileZilla to handle the file transfer protocol between the computer and PIs. Run the flink program with flink run and should look something like this “ run -c com.flinklearn.FLink_CSV examples/streaming/Flink_CSV.jar”.

A snippet of the council output is provided:

```
(Accel,-0.33518823242187495,-0.0143652099609375,9.634267480468749)
(Gyro,0.00425689701897019,0.013834915311653116,0.0010642242547425475)
(Mag,0.0,0.0,0.0)
(Accel,-0.3447650390625,0.08140285644531249,9.8449572265625)
(Gyro,0.003192672764227 6422,0.011706466802168022,-0.002128448509485095)
(Mag,0.0,0.0,0.0)
(Accel,-0.32561142578124996,0.09576806640624999,9.567229833984374)
(Gyro,0.005321121273712737,0.01596336382113821,0.0010642242547425475)
(Mag,0.0,0.0,0.0)
(Accel,-0.3399766357421875,0.0143652099609375,9.696516723632811)
(Gyro,0.0063853455284552845,0.018091812330623305,0.0)
(Mag,0.0,0.0,0.0)
(Accel,-0.373495458984375,0.05746083984375,9.849745629882811)
(Gyro,0.008513794037940 38,0.012770691056910569,0.0)
(Mag,0.0,0.0,0.0)
pi@navio:~/flink-1.12.1 $
```

D. *Navigating through Apache Flink WebUI*

Apache Flink has a handy WebUi which you can see and track real time performance metrics, making it easier to troubleshoot bottlenecks with various pipelines. Using the Flink WebUI is easy and straightforward. Clicking on the Task Managers we can see the allocated memory, heap, network and various metrics each nodes has to offer. Navigating to the completed jobs tab here we can see the Data Source of the jar file which is the Tuple 4 set. There is only 1 Parallelism since we created only one thread for the code to run.



E. Future Improvements

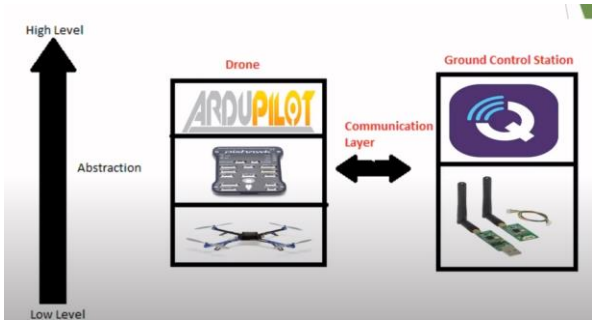
The next goal is for us to create a TCP or UDP port connection with Ardupilot with Flink so that the two programs can communicate with each other with live data. Using a TCP or UDP connection will also allow Flink to send data as commands to Ardupilot to control the drones. Ultimately, the goal is for there to be a drone sending data and controlling other drones. Flink will work as the distribution software where it will send commands to multiple drone's Ardupilot, thus controlling it. Unfortunately, our group couldn't finish debugging the TCP communication with Ardupilot. A snippet of the code being worked on is provided, it just has to be debugged to work. The idea is to create a socket stream from Ardupilot with port 9999 and stream that data to Flink. From there Flink will parse the datastream into three separate column then merge them back together to be printed on the WebUi.

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

/*************************************************************************
 *                    Doesen't work
 ************************************************************************/

final ParameterTool params = ParameterTool.fromArgs(args);  //check input parameters
env.getConfig().setGlobalJobParameters(params);             //make parameters available in the web UI

DataStreamSource<String> ACCStream = env.socketTextStream("localhost",9999);


        }
    });

DataStream<ACCValue> ACC_X_stream = env.addSource(new ACC_X("ACC_X",10));
DataStream<ACCValue> ACC_Y_stream = env.addSource(new ACC_Y("ACC_Y",20));
DataStream<ACCValue> ACC_Z_stream = env.addSource(new ACC_Z("ACC_Z",30));

DataStream<ACCValue> ACCStream = socketACCStream
        .merge(ACC_X_stream,ACC_Y_stream,ACC_Z_stream);
ACCStream.print();

env.execute("ACC Stream");
```

F. What is Ardupilot

Ardupilot is an open source, unmanned vehicle Autopilot Software Suite. It's capable of controlling autonomous multirotor drones, fixed wing aircraft, VTOL and more [4]. Ardupilot is the flight controller software in which it communicates with the Navio2, then the Navio2 will control the drone hardware.

We also need to connect Ardupilot with a ground control station. Ground control station is able to send commands to the drone such as landing or a GUI to update a parameter on a drone . The communication between the drone and control station is MAVLink which can be installed and run from their website.



G. Ardupilot Code

The ardupilot code was to get the data from mpu 9250 and create a CSV file with it's respected values. First step is to import the libraries then establish the imu test connection, initialize it, then create the a CSV file in which Flink will read it and process the data online.

```
import spidev
import time
import argparse
import sys
import navio.mpu9250
import navio.util
import csv
#include <AP_HAL/utility/Socket.h>
navio.util.check_apm()

imu = navio.mpu9250.MPU9250()


if imu.testConnection():
    print("Connection established: True")
else:
    sys.exit("Connection established: False")

imu.initialize()
time.sleep(1)

with open('/home/pi/Navio2/Python/output/AccelGyroMag.csv', mode='a', newline='') as csvfile:
    writer = csv.writer(csvfile)
    while True:
        m9a, m9g, m9m = imu.getMotion9()
        writer.writerow(['Accel', m9a[0], m9a[1], m9a[2]])
        writer.writerow(['Gyro', m9g[0], m9g[1], m9g[2]])
        writer.writerow(['Mag', m9m[0], m9m[1], m9m[2]])

        time.sleep(0.5)
```

Again we wanted to create a TCP connection with Flink and Ardupilot but unfortunately our group couldn't debug and get it to work in time. The worked on code is provided bellow. The idea is similar to the creating a CSV file, but instead create a socket connection with port 9999 with the local IP address.

```
import spidev
import time
import argparse
import sys
import navio.mpu9250
import navio.util
import csv
import jpysocket
import socket
import struct

#Installation :
#pip install jpysocket


navio.util.check_apm()
imu = navio.mpu9250.MPU9250()
if imu.testConnection():
    print("Connection established: True")
else:
    sys.exit("Connection established: False")

imu.initialize()
time.sleep(1)


while True:
    m9a, m9g, m9m = imu.getMotion9()
    print(m9a[0])
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('192.168.0.7', 9999))
    #ba = bytearray(struct.pack("f", m9[0]))
    client.send(bytes("I am CLIENT \n"))
```

IV. Drone Charging

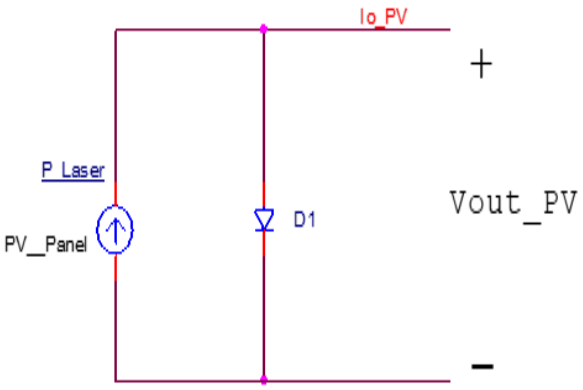
A. *Wireless Transfer Drone Charging*

Wireless charging or also known as wireless power transfer, is the technology that enables a power source to transmit energy to an electrical load across an air gap, without interconnecting cords. This power transfer concept has the potential of changing the way charging batteries is done by using a wireless power transfer technology such as laser charging. When applying laser charging to the drone swarm system, the need for docking to charge would be eliminated.

Laser light conversion requires a material to convert from one form of energy to electrical. A photovoltaic cell with a narrower wavelength concentration than the solar panel cells seems to be required. Ordinarily, a solar cell for solar energy generation uses monolithic multi-junction cells. These cells use a wafer for the conversion process of the solar spectrum into usable power. The multi- junction is connected in a series fashion which would require the same current to function. Laser light would only require a smaller wavelength for the collection and the conversion process.

It seems possible that variation in the design of the current multi junction device (PV material and structure) could prove feasible to work with laser light.

Electricity-to-Laser Power Conversion



Laser generated when stimulation current over threshold.

I(t) = stimulation current

P_Laser = Laser Beam Power

q = Charge

υ =Laser Frequency

h is the Planck constant

P_laser = hυ/q* [I(t) – Ith]

Laser-to-Electricity Power Conversion

laser beam power is converted to electrical power with a variation PV-panel.

The circuit model of the semiconductor is shown above. The PV-panel output voltage Vo and current Io can be characterized as:

Io_pv = [Isc – Is] *[e^(Vo/vt) – 1]

Isc = PV-panel short-circuit current

Is= saturation current

thermal voltage "Vt"

$Vt = nkT/q$

n = Variation PV-panel quality factor

k =Boltzmann constant

q = electron charge constant

T = absolute temperature.

The variation PV-panel converts the laser beam power P_laser to electrical current Io and voltage Vo at the receiver circuit.

The Laser Wireless Power Transfer is a relatively neglected area of research related to lasers. The use of lasers to transmit energy over to flying drones can vastly increase the time of drone flight. The current average consumer drone flying time is around 20 minutes.

B. Wired Drone Charging

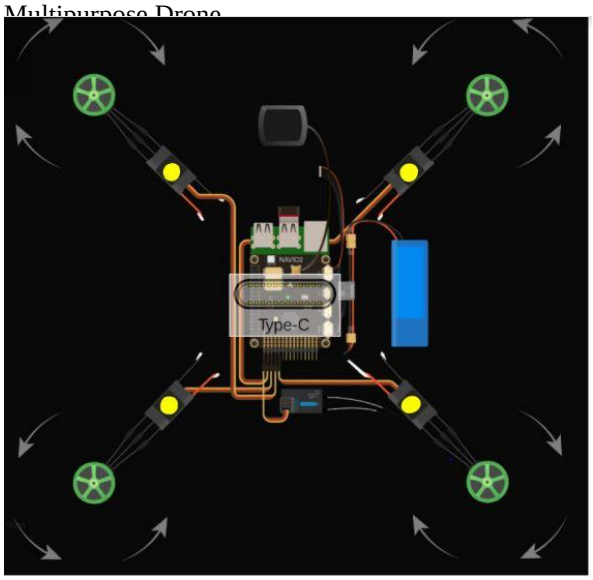
Wired Charging for the drone would place more constraint on the usage of the drone, however, it would save more power as less power is lost when charging the drone in comparison to the wireless charging.

In this design, there would be two types of drones. The first is a cargo drone, and the second is the multipurpose drone. The cargo drone would transport the drone swarm to the destination, with a high voltage battery pack inside it for charging the drones.

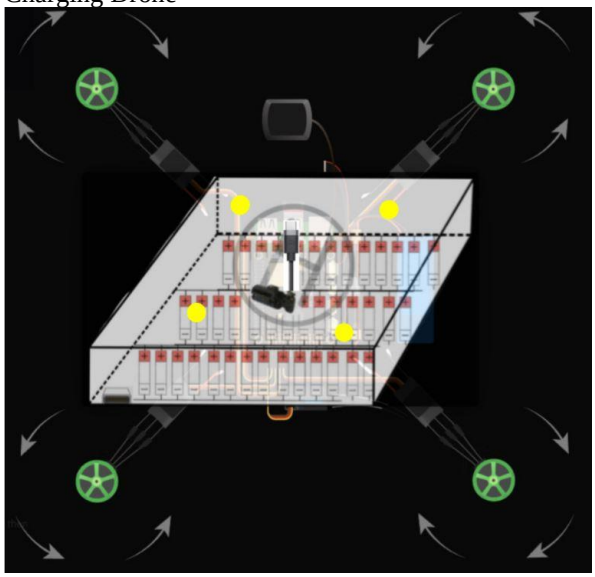
The charging drone would find a safe stationary location to land at to conserve power. The charging drone would have the ability to communicate with the drone swarm, and once one of the multipurpose drones reaches 20% of the battery life it can go to the charging drone and charge. If another drone reaches 20% while the first drone is charging the first drone will leave the charging drone if the battery is over 70%.

The charging of the drone will be done through a USB-C cable. A USB-C connector would be placed on the bottom of the multipurpose drone, and a USB-C mating connector would be placed on the top of the charging drone with an automated extending arm to insert the USB-C mating connector into the multipurpose drone. To ensure adequate placement of the USB-C connection, 4 mounting points are designed to be on the pad of the charging drone.

Multipurpose Drone



Charging Drone



CONCLUSIONS

REFERENCES

[1] Nancy Hall, *Thrust to Weight Ratio*, National Aeronautics and Space Administration (NASA), May 13, 2021. [Online] Available: https://www.grc.nasa.gov/www/k-12/airplane/fwrat.html

[2] "Hardware Setup: Emlid Docs." *Emlid Docs Blog RSS*, docs.emlid.com/navio2/hardware-setup/.Available: https://docs.emlid.com/navio2/hardware-setup

[3] "What Is Apache Flink? - Architecture." *Apache Flink*. [Online] Available:https://flink.apache.org/flink-architecture.html

[4] "Introduction: Emlid Docs," *Emlid Docs Blog RSS*. [Online]. Available: https://docs.emlid.com/navio2/

[5] "ArduPilot." *Wikipedia*, Wikimedia Foundation, 26 Apr. 2021, Available:en.wikipedia.org/wiki/ArduPilot.

[6] "High Level Perspective of a Flight Stack | Drone Programming." *YouTube*, YouTube, 1 Dec. 2018, www.youtube.com/watch?v=2iF9jp0YA8w.