Obtaining Real-Time Sensor Data from Unmanned Aerial System

Project Final Report

Vlad Dubrovenski and Han Zhou

GitHub Link:

https://github.com/vladi7/DroneVideoRecognition

Video Link:

https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=6f7b2a6a-4835-46bf-b2d6-ac840173f16a

1. Introduction

Unmanned Aerial System, or simply drone, have become an important tool in various industries. For example, in agriculture, the drones can be utilized to collect the data about the crops and even use the drones to apply various chemicals to crops in order to improve the efficiency. Another big industry is public safety, where law enforcement, fire departments, and other teams are using drones for the past several years to collect the data and improve the safety of team members. The delivery is another area where the drones can be utilized, and Amazon is currently testing drones for delivery of small-sized packages.

2. Motivation

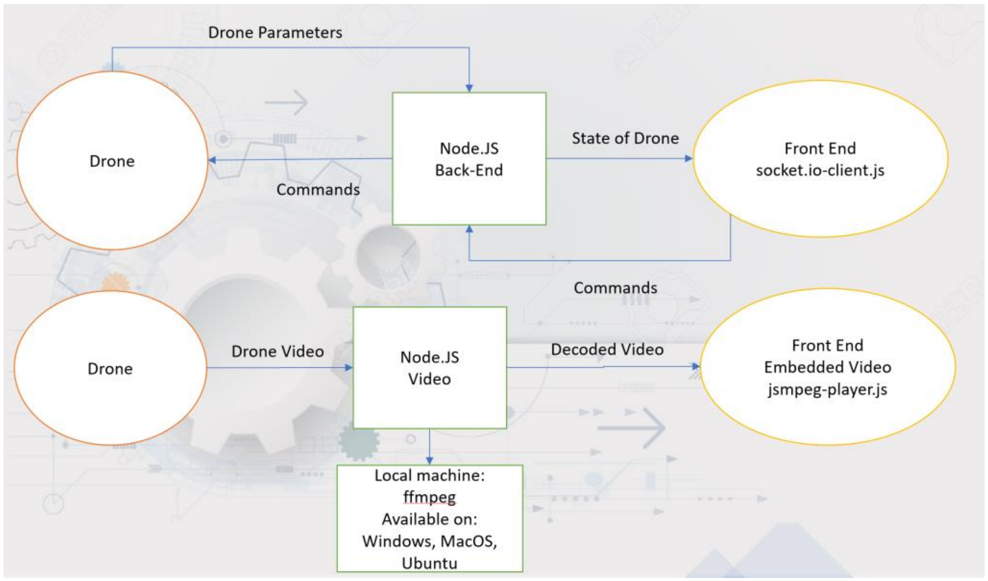
Our motivation for this project is to explore various parameters that can be collected from DJI Tello drone. This is a simple a drone that, meanwhile, has many sensors which makes this drone very robust. For example, one can land it on a hand, because the drone has the photo sensor that can detect the palm of the hand. Also, this drone has a few stabilization module which makes it extremely hard to crash it even for a novice user.

We believe this drone is ideal for testing as it is extremely robust and hard to destroy. Also, this drone has a nice programming API that we can use to both control it and retrieve the information from most of its sensors. And, of course, it is relatively inexpensive, so, we don't have to worry too much even if we crash it, which actually happens quite often since it is usually flown inside the apartment.

3. Challenges

We encountered a few issues while designing the solution for our system. First one was about the connection to computer. It was pretty simple to do. One of the authors already connected the drone with computer over wifi prior, but he used Delphi, which is a low-level language written in C and Assembly (one of the only languages that are not written exceptionally in C). We found it to be very easy to connect the drone to PC with node.js using socket_io library.
Another challenge was sending commands from front end to the drone. It was solved by issuing commands through socket to our back end Node.js server which transmitted those commands to the drone via another socket.
The last challenge for this project was how to get the data from the sensors. We found out from the documentation that the data can be obtained from another port of the drone, for which we used another socket, and transmitted it to the front end from Node.js.

4. Architecture


Drone Parameters
Drone
Node.JS Back-End
State of Drone
Front End socket.io-client.js
Commands
Commands
Drone
Drone Video
Node.JS Video
Decoded Video
Front End Embedded Video jsmpeg-player.js
Local machine: ffmpeg Available on: Windows, MacOS, Ubuntu

- The above figure is the architecture of our project. It contains two modules, one for controlling and reading the real-time parameters of the drone, and the other is for obtaining the video stream. Both schemes have three modules: Drone, Node.JS, and Front End but the video module also uses executable decoder that is installed on a server. It is an open source project under MIT license (included in this project as well). FFMPEG makes it very easy to decode the video. It uses a lot of RAM, however.

  Control Block

  - Drone

    Drone has the IP address of 192.168.10.1 and 3 different ports. Port 8889 is used for receiving the commands and replying with either “ok” or “error” with the error message included. There are various movement commands that can be issued. Also, command “streamon” and “streamoff” are used to turn on and turn off the video stream from the drone. The stream can be obtained from the port of 11111. This is our next task. The video stream will also need to be decoded. We also use the set commands mainly to set various speed of the drone (e.g. “speed 100”) which is measured in cm/s. We do not use the read commands as all the sensors’ parameters and battery level can be obtained from the port 8890. We cannot obtain any connectivity parameters (wifi etc.) from 8890 but they are irrelevant to this project. All of the ports use UDP protocol.
  - Server (Node.js + Express.js)

    We created the Node.js server using Express.js for simplicity as we did not need to fine-tune anything in this server. For dealing with sockets, we used the socket_io library. For this project, we only use 3 sockets, one to connect to the front end, another one to connect to the drone to control it, and the last one to obtain the information about the drone. For the next, we plan to add another socket for video retrieval from the drone.
  - Front End

    We decided to use Angular CLI for front end to have some practice with this great framework. We used Typescript instead of JavaScript to gain some experience with it. We connected to NodeJS with socket using “socket.io-client” library. We also designed the front end to be responsive, simple, attractive, and reliable.

Video Block

- Drone
  Drone uses the same IP address as in the control block, but the port 11111 is utilized to transmit the video via UDP. Video feed needs to be decoded. The resolution is 620x480, however, "Iron Man version" has a resolution of 720p (1280x720).
- Node.js Video
  This module is stored in videoFeed.js. As it was mentioned prior, this module utilizes FFMPEG for decoding the video. It can compress and decompress the video feed is needed.
- Front End
  This block utilized the web-socket client to communicate with the server. It also uses jsmpeg-player.js for displaying the video. The video is displayed in real-time with a lag of approximately 1 second. This lag is planned to be improved with the utilization of Bluetooth/Radio waves from Raspberry PI which will make the WiFi do less work, therefore, the lag will decrease.

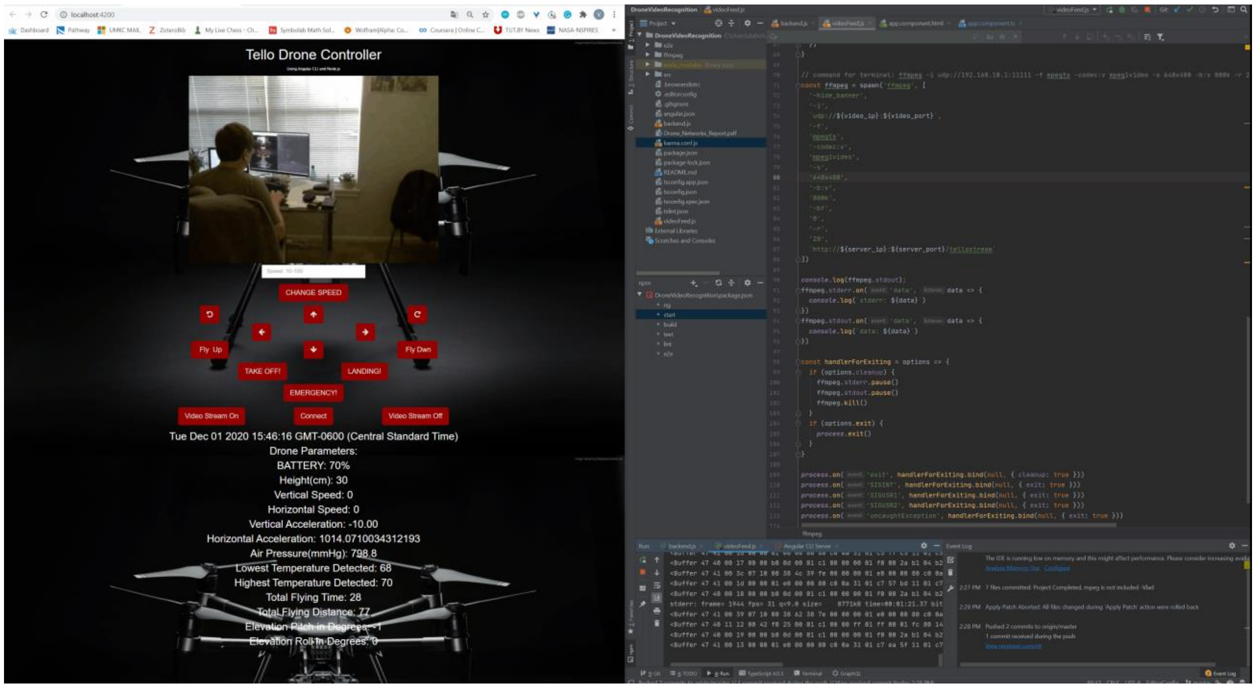
Technology Justification

- **Angular**
  Angular is ideal for single page applications. Since this application is only for 1 drone, it is a very elegant solution. TypeScript provides object-oriented style of programming, which is useful and make the code cleaner.
- **Node.js**
  Node.js is very fast in creating the web servers. Unlike servlet-based servers (e.g. Tomcat), the deployment time is small. The request processing is still faster with Tomcat, but we don't have too many requests since we only control one drone. We also used Express.js to further speed up the deployment of the servers as our settings did not have to be fine-grained.
- **FFMPEG**
  This is an open-source project under MIT License, which we also included with our project. This project makes it very easy to decode the video feed in real-time but it utilizes a lot of RAM
- **Server/Client Hardware**

The following Hardware Spec was used:

- 32GB RAM (custom installed, the fastest available for the motherboard used for both HTTP and Web Socket servers)
- SSD (Is utilized for fast write/read of video)
- Discrete GPU NVIDIA GTX 6GB (For Front-end)
- 12 Core Processor (Core i7 8750H @2.2 GHz without boost)
- Dell G7 Laptop (2 years old but still amazing ☺)

5. Experiment



Tue Dec 01 2020 15:46:16 GMT-0600 (Central Standard Time)
Drone Parameters:
BATTERY: 70%
Height(cm): 30
Vertical Speed: 0
Horizontal Speed: 0
Vertical Acceleration: -10.00
Horizontal Acceleration: 1014.0710034312193
Air Pressure(mmHg): 798.8
Lowest Temperature Detected: 68
Highest Temperature Detected: 70
Total Flying Time: 28
Total Flying Distance: 77
Elevation Pitch in Degrees: -1
Elevation Roll in Degrees: 0

```
}

// command for terminal: ffmpeg -i udp://192.168.10.1:11111 -f mpegts -codec:v mpeg1video -s 640x480 -b:v 800k -r 2
const ffmpeg = spawn('ffmpeg', [
  '-hide_banner',
  '-i',
  `udp://${video_ip}:${video_port}`,
  '-f',
  'mpegts',
  '-codec:v',
  'mpeg1video',
  '-s',
  '640x480',
  '-b:v',
  '800k',
  '-bf',
  '0',
  '-r',
  '20',
  `http://${server_ip}:${server_port}/tellostream`
])

console.log(ffmpeg.stdout);
ffmpeg.stderr.on('data', data => {
  console.log(`stderr: ${data}`)
})
ffmpeg.stdout.on('data', data => {
  console.log(`data: ${data}`)
})

const handlerForExiting = options => {
  if (options.cleanup) {
    ffmpeg.stderr.pause()
    ffmpeg.stdout.pause()
    ffmpeg.kill()
  }
  if (options.exit) {
    process.exit()
  }
}

process.on('exit', handlerForExiting.bind(null, { cleanup: true }))
process.on('SIGINT', handlerForExiting.bind(null, { exit: true }))
process.on('SIGUSR1', handlerForExiting.bind(null, { exit: true }))
process.on('SIGUSR2', handlerForExiting.bind(null, { exit: true }))
process.on('uncaughtException', handlerForExiting.bind(null, { exit: true }))
```

```
<Buffer 47 40 00 17 00 00 b0 0d 00 01 c1 00 00 00 01 f0 00 2a b1 04 b2
<Buffer 47 41 00 3c 07 10 00 38 4c 39 fe 00 00 00 01 e0 00 00 00 c0 0a
<Buffer 47 41 00 1d 00 00 01 e0 00 00 00 c0 0a 31 01 c7 57 bd 11 01 c7
<Buffer 47 40 00 18 00 00 b0 0d 00 01 c1 00 00 00 01 f0 00 2a b1 04 b2
stderr: frame= 1944 fps= 31 q=9.0 size=    8771kB time=00:01:21.37 bit
<Buffer 47 41 00 39 07 10 00 38 62 38 7e 00 00 00 01 e0 00 00 00 c0 0a
<Buffer 47 40 11 12 00 42 f0 25 00 01 c1 00 00 ff 01 ff 00 01 fc 80 14
<Buffer 47 40 00 19 00 00 b0 0d 00 01 c1 00 00 00 01 f0 00 2a b1 04 b2
<Buffer 47 41 00 13 00 00 01 e0 00 00 00 c0 0a 31 01 c7 ea 5f 11 01 c7
```

```
The IDE is running low on memory and this might affect performance. Please consider increasing avail
Analyze Memory Use   Configure
2:27 PM  7 files committed: Project Completed. mpeg is not included -Vlad
2:28 PM  Apply Patch Aborted: All files changed during 'Apply Patch' action were rolled back
2:28 PM  Pushed 2 commits to origin/master
         1 commit received during the push
```

For the experiment, we retrieved and processed all the parameters available from the drone. Some of the parameters needed further calculations, like conversion of the Air Pressure from oz/in^2 to mmHg. Horizontal velocity and acceleration where giving in x and z directions, so we used Pythagorean theorem to calculate the magnitude of the velocity and acceleration. This experiment demonstrated that even with a simple and relatively cheap drone we could retrieve a lot of useful information, like temperature and air pressure. This can be used, for example, by firefighters to check the temperature inside the building before going in. Of course, they would have more complicated drones with a lot of sensors, but the goal of this project was to explore the efficient way of obtaining real-time sensor data from the drone.

As it can be seen in the above screenshot, the video was also obtained and processed. You may consult the powerpoint (or GitHub readme) in order to play the GIF which will show the project in action.

6. Conclusion/Future Work

In this project, we obtained all possible useful parameters from the drone. We also can have the complete control of the drone from the web front end. We do skip a few irrelevant commands that we plan to add in the future work. Another goal we accomplished was obtaining the video from the drone and display it, processing it, and displaying it in our front end. Further goal is to make our back end robust and less error-prone. Another goal will be to process the video from the drone with Deep Learning Library in order to recognize the objects. And the ultimate goal is to explore the control functions with Bluetooth or radio module (both supported by Tello) maybe beneficial to reduce the lag. It does, however, present a trade-off between the distance vs response time and using Bluetooth/Radio for controlling needs to be carefully considered