

VIP Smart Cities Technology Transportation Users

Team Leader: Robe Zhang

Team Members:

Jian Nan (Andy) Huang

Mikhail Vasilevitskiy

Nakib Mashrur

Aidan Orion Llorca

Semester: Spring 2018

ABSTRACT

In this semester, the transportation users sub team worked on interacting with the 3DR Solo Drone itself as well as in a simulated environment. Much of the procedure was gathered off of research gained from reading various drone websites. The process and steps taken to interact with the 3DR Solo Drone are documented in the Software Setup Google Document^[9]. Currently, the drone is able to run python scripts in the simulation, however executing the scripts on the physical drone has its limitations. The drone simulation utilizes the following packages: DroneKit, Ardupilot, and Mission Planner. In the simulation, the script is able to observe the flight status of the drone and fly to a given GPS location. The overall goal is to deploy the drones that are capable of obtaining sensor data in various locations across the city.

INTRODUCTION

Drones technology is slowly paving its way from recreational use to commercial use. Recreational drones such as the 3DR solo are relatively small in size with limited battery life, and have limited carrying capacity. They are useful in helping the user observe the environment by reaching high elevations and inaccessible places. However, these drones are manually operated by the user. Industrial size drones have increased capabilities such as having an extended battery life, a larger payload size and customizable features. Companies such as Amazon are implementing drones to help deliver packages^[1]. These drones would autonomously fly in dedicated air zones and carry packages to consumers. Equipped with additional sensors, the drones would be able to detect other drones and consumers^[2]. Drones have also been used in observing disaster sites by utility companies to access damage and status of equipment^[3]. Although there are advancements for drone technology, its capabilities is still regulated by the Federal Aviation Administration.

The Smart Cities Technology VIP seeks to create drones that are capable of obtaining sensor data in various locations across the city. The neighborhoods of interest are Red Hook, Brownsville, Financial District and Hudson Yard. By attaching with various components and redesigning the structure of the 3DR drone, the VIP team would be able to gather data about an area. Each neighborhood would have a designated charging station. Components include a lidar sensor would aid the drone from collision, sensors, and an upgraded battery. The transportation users sub team worked on interacting with the 3DR Solo Drone via scripts in a simulated environment mimicking the goal.

Overall, having the ability to control drones interests a lot of people, including the government, researchers, and casual users. The government can use this technology and deploy their own drones to collect data or do other tasks. Researchers can also use this technology for data gathering purposes. Lastly, drone users in general can program their own drones for entertainment purposes.

LITERATURE REVIEW

The team has researched security regarding drone flight. The most notable finding is from Samy Kamkar, the ethical, grey hat hacker that disclosed vulnerabilities involving Internet of Things devices (IoT). Samy Kamkar is a grey hat hacker that was responsible for the first self propagating cross-side scripting worm released in 2005^[4]. The worm injected itself into MySpace profile pages with a payload of “but most of all, Samy is my hero.” The payload is displayed on the infected MySpace profile page. The worm self propagates by each additional user which visits an infected profile. Samy also discovered a method to remotely hijack a vehicle by hacking the the vehicle's ECU using his “remote gadget”^[5]. He was able to manipulate the core functionality of the vehicle, such as the braking and acceleration systems. These attacks are extremely dangerous for the driver, passengers, and other drivers. Samy demonstrates the trade-off many companies make when they develop products that are connected to the internet. He raises questions about the security of new products being introduced to the market. Many companies are more focused on developing a working product and are neglecting the security features to prevent malicious attackers from gaining access to the device. Many companies create these devices to be easy to use which also means that it is easy to get into for attackers. Samy also demonstrates an exploit that he has developed to autonomously overtake drones^[6]. In his youtube video, he demonstrates how his attacker drone disconnects other drones from their wifi networks and redirects their internet connection to the attacker drone. Afterwards, the attacker drone is able to command the disconnected drone. This is a major vulnerability that should be addressed before any form of autonomous drones can be deployed. A recommendation for mitigating this attack is to create a high entropy wifi password to avoid any possible wifi cracking/deauthentication attacks. Samy explains his methodology and tools to execute this attack in his website^[7].

METHODS

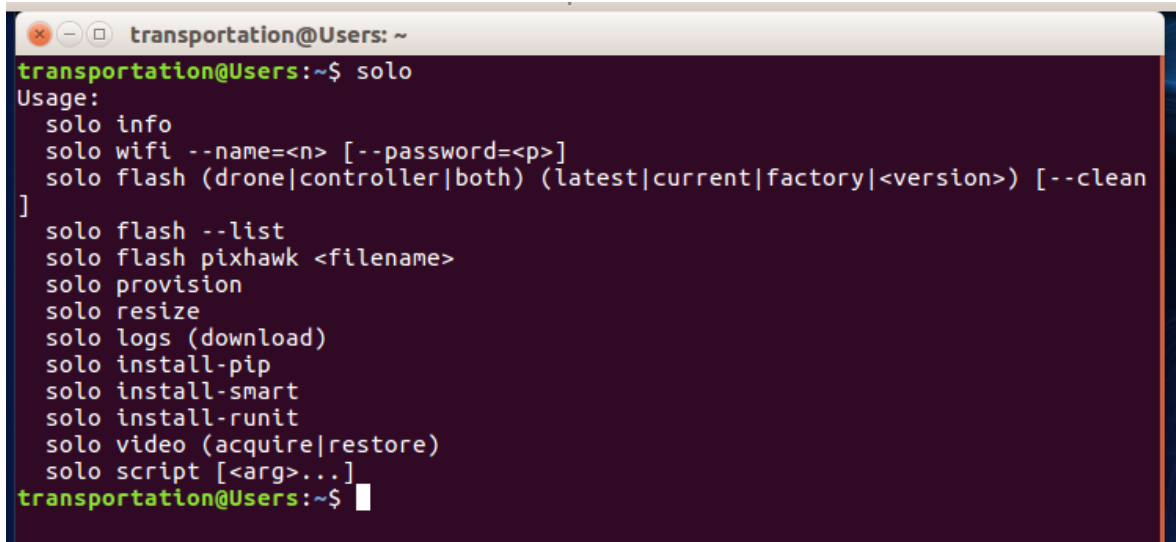
The creation of the drone simulator involved detailed steps utilizing Ardupilot, DroneKit, and Mission Planner. The operating system of choice was Linux because it allows the most versatility for the programs. Also, the majority of the programs required for the simulation was initially developed using Linux and deviating from such will require extra reconfigurations. We decided to go with the [Ubuntu 16.04.3 LTS](#) distribution of Linux in a virtual environment via a hypervisor. The hypervisor allows the host machine to run instances of virtual machines or guest machines on top of the host machine. The chosen hypervisor is [Oracle Virtual Box](#) with the extension pack. After selecting the environment, distribution, and flavor, we setup the guest machine with at least 4GB of memory, and at least 15GB of storage space. We then ensured that the guest machine was under a NAT network to share internet with the host machine. Upon booting into the guest machine, we installed pip, virtualenv, and solo cli using the following commands^[8]:

```

sudo apt-get update
sudo apt-get install python-pip
sudo pip install --upgrade pip
sudo pip install virtualenv
sudo pip install https://github.com/3drobotics/solo-cli/archive/master.zip --no-cache-dir

```

A successful installation of solo cli should display the following:



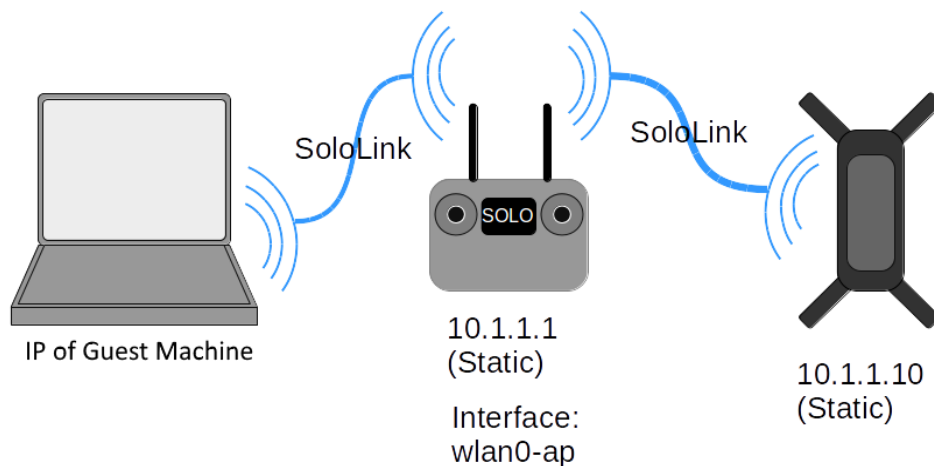
```

transportation@Users: ~
transportation@Users:~$ solo
Usage:
  solo info
  solo wifi --name=<n> [--password=<p>]
  solo flash (drone|controller|both) (latest|current|factory|<version>) [--clean]
]
  solo flash --list
  solo flash pixhawk <filename>
  solo provision
  solo resize
  solo logs (download)
  solo install-pip
  solo install-smart
  solo install-runit
  solo video (acquire|restore)
  solo script [<arg>...]
transportation@Users:~$

```

The

next step was to connect to the 3DR Drone to the guest machine. This was done by first removing the propellers off the 3DR Drone as a safety precaution and turning on both the drone and the controller. The controller acts as the flight controller that commands the drone. The network can be observed in the following diagram:



Connecting the guest machine is accomplished by simply connecting to the controller wifi network. A successful connection should display the following:

```

transportation@Users:~$ solo info
connecting to Solo and the Controller...
{
  "controller": {
    "ref": "3dr-controller-imx6solo-3dr-artoo-20160926202703",
    "version": "2.4.2"
  },
  "gimbal": {
    "connected": false
  },
  "pixhawk": {
    "apm_ref": "7e9206cc",
    "px4firmware_ref": "5e693274",
    "px4nuttx_ref": "d48fa307",
    "version": "1.3.1"
  },
  "solo": {
    "ref": "3dr-solo-imx6solo-3dr-1080p-20160926202940",
    "version": "2.4.2"
  }
}

```

We came across connection errors, but we were able to resolve them by following the forum: <https://diydrones.com/group/solo/forum/topics/3dr-solo-bli-command-line>. The only issue is by connecting to the controller network, we lost connection to the internet. This was fixed by using the *solo wifi* command. The solo wifi command allows the guest computer to be connecting to both the internet and the solo network. The NYU wifi issue was not compatible with the *solo wifi* command since it requires additional credentials to login. We bypassed this by setting up a wifi hotspot. This can be done using a mobile device or a laptop as documented in the software setup document^[9]. After setting up a valid wifi network, the following command connects the guest computer to both the internet and the solo network:

solo wifi --name={Your Network Name} --password={Your Network Password}

```

transportation@Users:~$ solo wifi --name=Wurple-X --password=11111111
connecting to the Controller...

connecting to encrypted wifi network.
(your computer may disconnect from Solo's network.)

please manually reconnect to Solo's network once it becomes available.
it may take up to 30s to a reconnect to succeed.
No handlers could be found for logger "paramiko.transport"

rm: can't remove '/mnt/rootfs.rw/lib/modules/3.10.17-rt12-*/kernel/net/ipv4/netfilter/iptables_filter.ko': No such file or directory
Stopping HOSTAP Daemon: stopped /usr/sbin/hostapd (pid 802)
hostapd.
killall: wpa_supplicant: no process killed
killall: udhcpc: no process killed
restarting DNS forwarder and DHCP server: dnsmasq...
stopping DNS forwarder and DHCP server: dnsmasq... stopped /usr/bin/dnsmasq (pid 788)
done.
starting DNS forwarder and DHCP server: dnsmasq... done.
done.
connecting to the internet...
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
udhcpc (v1.21.1) started
Sending discover...
Sending discover...
Sending discover...
Sending select for 172.20.10.5...
Lease of 172.20.10.5 obtained, lease time 85536
/etc/udhcpc.d/50default: Adding DNS 172.20.10.1
Starting HOSTAP Daemon: Configuration file: /etc/hostapd.conf
rfkill: Cannot open RFKILL control device
wlan0-ap: interface state UNINITIALIZED->COUNTRY_UPDATE
Using interface wlan0-ap with hwaddr 8a:dc:96:3f:6e:f8 and ssid "SoloLink_3F6EF8"
wlan0-ap: interface state COUNTRY_UPDATE->ENABLED
wlan0-ap: AP-ENABLED
hostapd.
setting up IP forwarding...

success: Solo is now connected to the Internet.
if your computer does not yet have Internet access, try
disconnecting and reconnecting to Solo's wifi network.
(resetting Solo's DNS... done.)

```

With a successful connection, we installed the SITL (Software In The Loop) simulator^[10]. The SITL allows for testing of DroneKit-Python scripts without a real drone. SITL was installed using the following command:

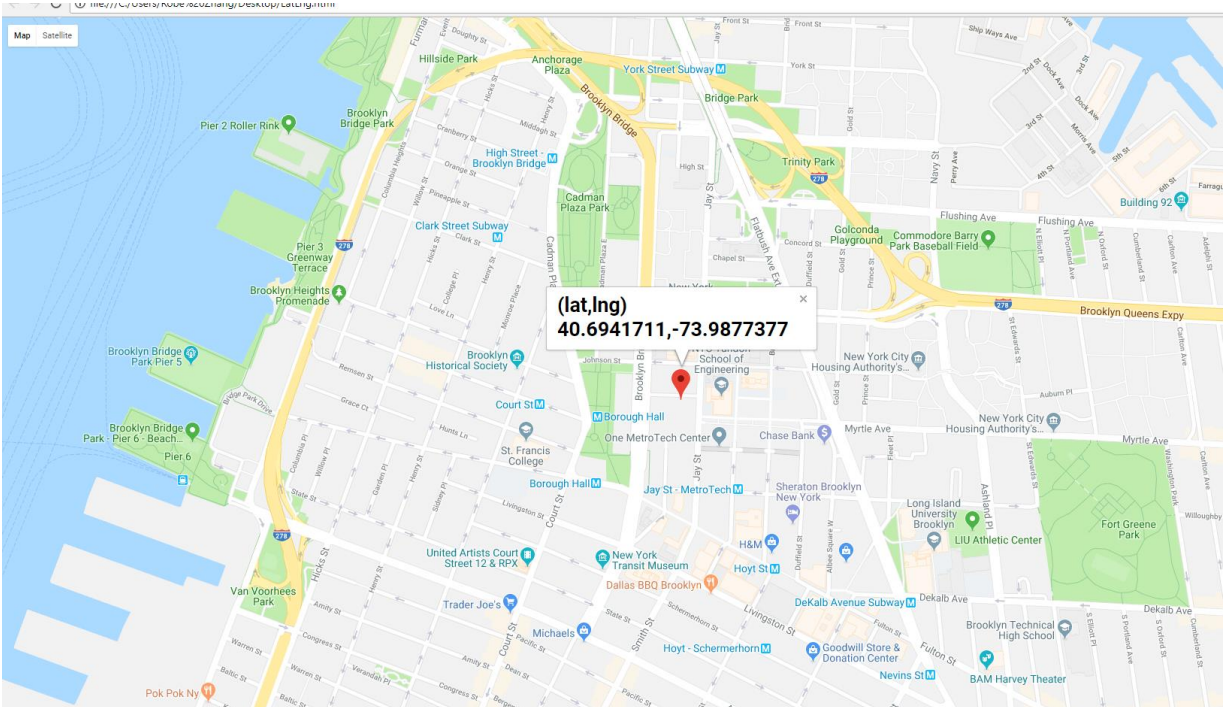
```
sudo pip install dronekit-sitl -UI
```

The simulator is launched and waiting for a TCP connection on 127.0.0.1:5760 with a starting location at Tandon using the following command:

```
dronekit-sitl solo-2.0.20 --home=40.69404728377923,-73.98643185091908
```

```
transportation@Users:~$ dronekit-sitl solo-2.0.20 --home=40.69404728377923,-73.98643185091908
os: linux, apm: solo, release: 2.0.20
SITL already Downloaded and Extracted.
Ready to boot.
Execute: /home/transportation/.dronekit/sitl/solo-2.0.20/apm --home=40.69404728377923,-73.98643185091908 -
-model=quad
Started model quad at 40.69404728377923,-73.98643185091908 at speed 1.0
bind port 5760 for 0
Starting sketch 'ArduCopter'
Serial port 0 on TCP port 5760
Starting SITL input
Waiting for connection ....
```

The GPS location was obtained using the Google Maps API and the LatLng program^[11].



We then installed Ardupilot, MAVProxy, and Mission Planner. The specific instructions for installing these packages are listed in [Software Setup Document](#). Once all three packages are successfully installed, a script can be tested.

We had to first launch the SITL in a terminal with the following command:

```
dronekit-sitl solo-2.0.20 --home=40.6940472,-73.9864318,0,0
```

We then had to launch MAVProxy in a new terminal with one listening TCP connection on *tcp:127.0.0.1:5760* and two broadcasting UDP connection on *127.0.0.1:14550* and *127.0.0.1:14551* with the following command:

```
mavproxy.py --master tcp:127.0.0.1:5760 --out udp:127.0.0.1:14550 --out udp:127.0.0.1:14551
```

MAVProxy connects with SITL with the listening connection. The two broadcasting UDP connections are for the Mission Planner Simulator and the Python Script.

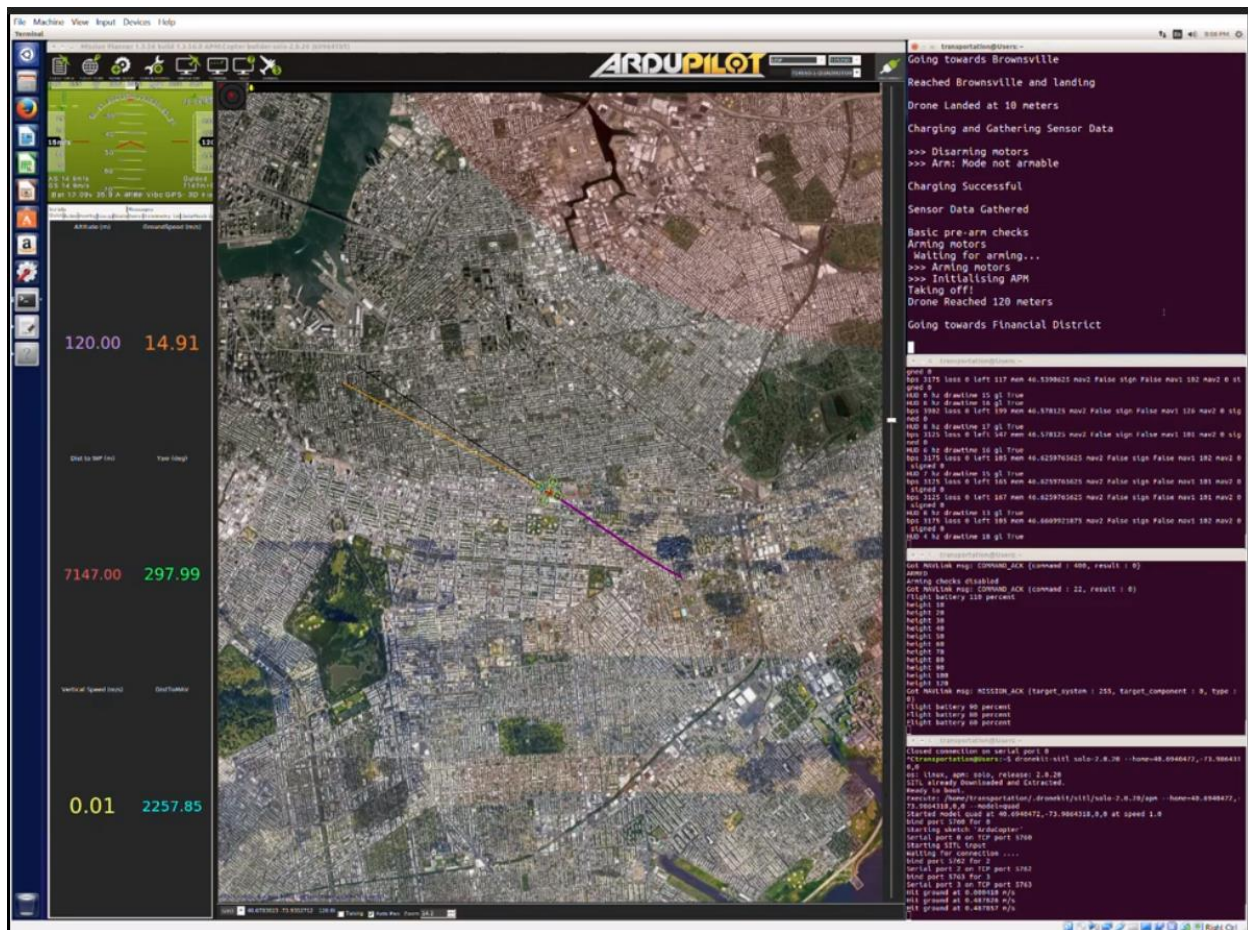
Next, we started Mission Planner in a new terminal with the following command:

```
sudo mono ~/Desktop/MissionPlanner-latest/MissionPlanner.exe
```

We manually connected Mission Planner to listen to the UDP port *127.0.0.1:14550*. Now, the simulation is waiting for commands. In a new terminal, we ran the python script with the following command:

```
sudo python ~/Desktop/ExposPointsOfInterest.py --connect udp:127.0.0.1:14551
```

The script should successfully execute on Mission Planner as shown below:



ANALYSIS

To understand the Python DroneKit package, we tested sample scripts. We downloaded the repository from github: <https://github.com/3drobotics/solodevguide> and executed the script with: `python helloworld.py tcp:127.0.0.1:5760`^[12]

The script gives information about the current status of the drone which is used developing scripts for the goto functionality. We are able to view the drone's current latitude, longitude, altitude, velocity, ground speed, battery levels, flight mode and etc. This information allows us to create functions to control the drone.

```
transportation@Users:~/Desktop/solodevguide/examples/helloworld$ python helloworld.py tcp:127.0.0.1:5760
Connecting to tcp:127.0.0.1:5760...
>>> APM:Copter builder-solo-2.0.20 (609645b5)
>>> Frame: QUAD
>>> Calibrating barometer
>>> Barometer calibration complete
>>> GPS 0: detected as u-blox at 38400 baud
Vehicle state:
Global Location: LocationGlobal:lat=0.0,lon=0.0,alt=None
Global Location (relative altitude): LocationGlobalRelative:lat=0.0,lon=0.0,alt=0.0
Local Location: LocationLocal:north=None,east=None,down=None
Attitude: Attitude:pitch=1.1079120668e-05,yaw=4.44178222097e-05,roll=-1.89987531485e-05
Velocity: [0.0, 0.0, 0.0]
Battery: Battery:voltage=0.0,current=0.0,level=100
Last Heartbeat: 1.931358506
Heading: 0
Groundspeed: 0.0
Airspeed: 0.0
Mode: STABILIZE
Is Armable?: False
Armed: False
Done.
```

We also tested the sample goto script with the command:

`python simple_goto.py tcp:127.0.0.1:5760`^[13]

The sample goto script brings the drone to a given altitude then begins to fly towards a LatLng destination for 30 seconds and attempts to return to launch. The script does not check if the vehicle has reached its destination before it attempts to return to launch. The script only flies for 30 seconds before it moves onto the next command. By understanding the basic functionality of the script, we were able to create additional functions to allow for the drone to reach its destination before executing the next task.

We developed a “landCopter” function, a “chargeAndGatherData” driver function and a “goUntil” function^[14]. The “landCopter” function changes the vehicle mode of the drone from GUIDED to

LAND. The LAND vehicle mode allows for the drone to disarm its throttle once it has landed. In the function, we monitor the relative altitude of the drone until it has almost reached 0 meters in relative altitude.

```
def landCopter(landingHeight):
    vehicle.mode = VehicleMode("LAND")
    while True:
        # print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt * 0.95 <= 0:
            print("Drone Landed at %d meters\r\n" % (landingHeight))
            break
```

The “chargeAndGatherData” driver function is currently a temporary function that will charge the drone while it gathers data. This logic will be incorporated once the data gathering sensors are attached and functional. The driver function will charge the drone and activate the data gathering sensors.

```
def chargeAndGatherData():
    print("Charging and Gathering Sensor Data\r\n")
    time.sleep(10)
    print("\r\nCharging Successful\r\n")
    print("Sensor Data Gathered\r\n")
```

```
transportation@Users:~/Desktop/dronekit-python/examples/simple_goto$ python simple_goto.py
Starting copter simulator (SITL)
SITL already Downloaded and Extracted.
Ready to boot.
Connecting to vehicle on: tcp:127.0.0.1:5760
>>> APM:Copter builder-solo-2.0.20 (609645b5)
>>> Frame: QUAD
>>> Calibrating barometer
>>> Barometer calibration complete
>>> GPS 0: detected as u-blox at 38400 baud
Basic pre-arm checks
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Waiting for vehicle to initialise...
Arming motors
Waiting for arming...
>>> Arming motors
>>> Initialising APM
Taking off!
Altitude: 0.0
Altitude: 0.0
Altitude: 0.21
Altitude: 1.49
Altitude: 3.36
Altitude: 5.98
Altitude: 8.01
Altitude: 9.25
Altitude: 9.64
Reached target altitude
Set default/target airspeed to 3
Going towards first point for 30 seconds ...
Going towards second point for 30 seconds (groundspeed set to 10 m/s) ...
Returning to Launch
Close vehicle object
Traceback (most recent call last):
  File "simple_goto.py", line 105, in <module>
    sitl.stop()
  File "/usr/local/lib/python2.7/dist-packages/dronekit_sitl/__init__.py", line 291, in stop
    kill(self.p.pid)
  File "/usr/local/lib/python2.7/dist-packages/dronekit_sitl/__init__.py", line 32, in kill
    process = psutil.Process(proc_pid)
  File "/usr/local/lib/python2.7/dist-packages/psutil/__init__.py", line 341, in __init__
    self._init(pid)
  File "/usr/local/lib/python2.7/dist-packages/psutil/__init__.py", line 381, in _init
    raise NoSuchProcess(pid, None, msg)
```


The “goUntil” function take three inputs: latitude, longitude and altitude. The drone goes to the destination based on the latitude and longitude location. Similar to the “landCopter” function, we monitor the relative GPS location of the drone until it has almost reached its destination. As we were testing the precision of the simulation, we found that the simulation was able to detect the coordinates up to four decimal places. Therefore, we are checking the latitude and longitude of the drone with precision of 4 decimal places. Once the drone has reached its destination, it has completed the function.

```
# go to lat lon until it reaches there
def goUntil(lat,lon,height):
    point1 = LocationGlobal(lat,lon, height)
    vehicle.simple_goto(point1)
    while '%.4f'%(vehicle.location.global_relative_frame.lat) != '%.4f'%(lat) and '%.4f'%(vehicle.location.global_relative_frame.lon) != '%.4f'%(lon):
#         print "Lat: %s" % vehicle.location.global_relative_frame.lat
#         print "Lon: %s" % vehicle.location.global_relative_frame.lon
        time.sleep(2)
```

DISCUSSION

We have made progress in setting up the simulation environment to test script for the drone. One thing to note about the simulation is the GPU requirement to process the images clearly. The VIP computer on the second floor has enough GPU specifications to run the simulation smoothly. Regular laptops may have trouble running the simulation smoothly. Another thing to note about the simulation is the speed limit of the drone. The speed is capped at 15 m/s. Getting the simulator to function properly and having the capability to test scripts in a simulated environment is a great first step. This would allow scripts to be tested without causing any harm to the actual drone. We have also been working on uploading scripts directly onto the drone. We are able to export our scripts onto the drone, however, there seems to be an issue with arming the drone. The script requires the vehicle mode to be GUIDED, which requires the drone to maintain a steady GPS connection. We were unable to obtain the steady GPS connection required for the drone to execute the script. From observing forums, we have determined that the drone has a built in safety check before the script can gain full control of the drone’s functions. This means we have to somehow override safety protocols. We were looking into turning off the safety checks through an option on MAVProxy, however we were not successful in doing so.

CONCLUSION

Throughout this semester a lot of progress was made in creating a drone simulator. The simulator created consisted of the following programs: Ardupilot, DroneKit, and Mission Planner. The creation of this simulator will help other subteams test the efficiency and reliability of their sensors and battery packs. Upon the completion of the testing phase, the drones will be sent out to Red Hook, Brownsville, Financial District and Hudson Yard to collect specific sets of data. All in all, this simulator shows the team is heading in a good direction; our next steps are to improve this simulator and adjust it to suit every subteam needs.

RECOMMENDATION

The team's future goals are to disable the drone's arm throttle because doing this will allow more freedom in how scripts can be written. In addition, the pre-arm safety check should also be disabled because receiving GPS signals inside a building is rather difficult, which hinders the drone from flying. Furthermore, much more complicated drone scripts will be written to incorporate the various sensors (Lidar) into the movement pattern of the drone. Since the sensors will be controlled via the Arduino microcontroller, further research in pairing Python and Arduino will need to be conducted.

REFERENCES

- [1] Amazon Prime Air <https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011>
- [2] Shaban, Hamza. "Amazon is issued patent for delivery drones that can react to screaming voices, flailing arms" The Washington Post, The Washington Post, 22 Mar. 2018 https://www.washingtonpost.com/news/the-switch/wp/2018/03/22/amazon-issued-patent-for-delivery-drones-that-can-react-to-screaming-flailing-arms/?utm_term=.c7b5815a1a70
- [3] Utility Company In Natural Disasters <https://edgylabs.com/drones-help-utilities-cope-with-the-times-and-restore-service-post-natural-disasters>
- [4] Samy Worm [https://en.wikipedia.org/wiki/Samy_\(computer_worm\)](https://en.wikipedia.org/wiki/Samy_(computer_worm))
- [5] Sydell, Laura. "This 'Gray Hat' Hacker Breaks Into Your Car - To Prove A Point." NPR, NPR, 23 Feb. 2018. <https://www.npr.org/sections/alltechconsidered/2018/02/23/583682220/this-gray-hat-hacker-breaks-into-your-car-to-prove-a-point>
- [6] SkyJack Video https://www.youtube.com/watch?v=EHKV01YQX_w
- [7] Samy's SkyJack Website <http://www.samy.pl/skyjack/>

- [8] 3DR Solo Development Guide
<https://dev.3dr.com/starting-utils.html>
<https://github.com/3drobotics/solo-cli>
http://python.dronekit.io/guide/quick_start.html
<https://github.com/dronekit/dronekit-python>
- [9] Software Setup Document
<https://docs.google.com/document/d/1hQA10KxpLFrS18YwyRy5wI48esr7vpzUuvhnAqpp-xY/edit?usp=sharing>
- [10] SITL Setup
http://python.dronekit.io/develop/sitl_setup.html
- [11] LatLng Extractor
https://drive.google.com/file/d/13_AzSAEk3CICzeDtleCdz7Qa_nD_Um9O/view?usp=sharing
- [12] Hello World Sample Script
<https://dev.3dr.com/example-helloworld.html>
- [13] GoTo Sample Script
http://python.dronekit.io/examples/simple_goto.html
- [14] Points of Interest Script
https://drive.google.com/file/d/1u7iJv-ehxv8mlsYly30_KFYPOBAiIG-A/view?usp=sharing