Software Development for Unmanned Aerial Systems

Instructor:

Jane Cleland-Huang, PhD

JaneClelandHuang@nd.edu

Department of Computer Science and Engineering

University of Notre Dame

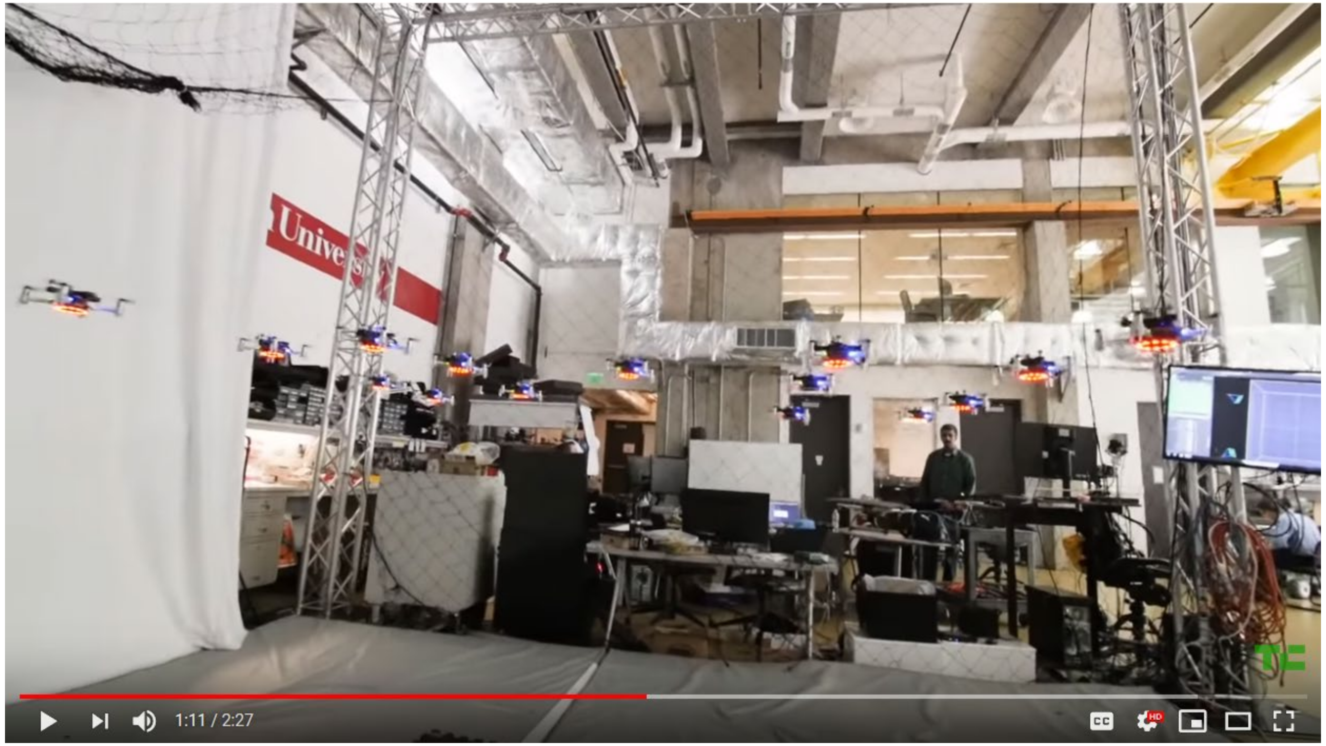
Goals for Today's Class

1. Working with multiple collaborating drones

2. Companion computers

3. Project ideas, promotions and team selection

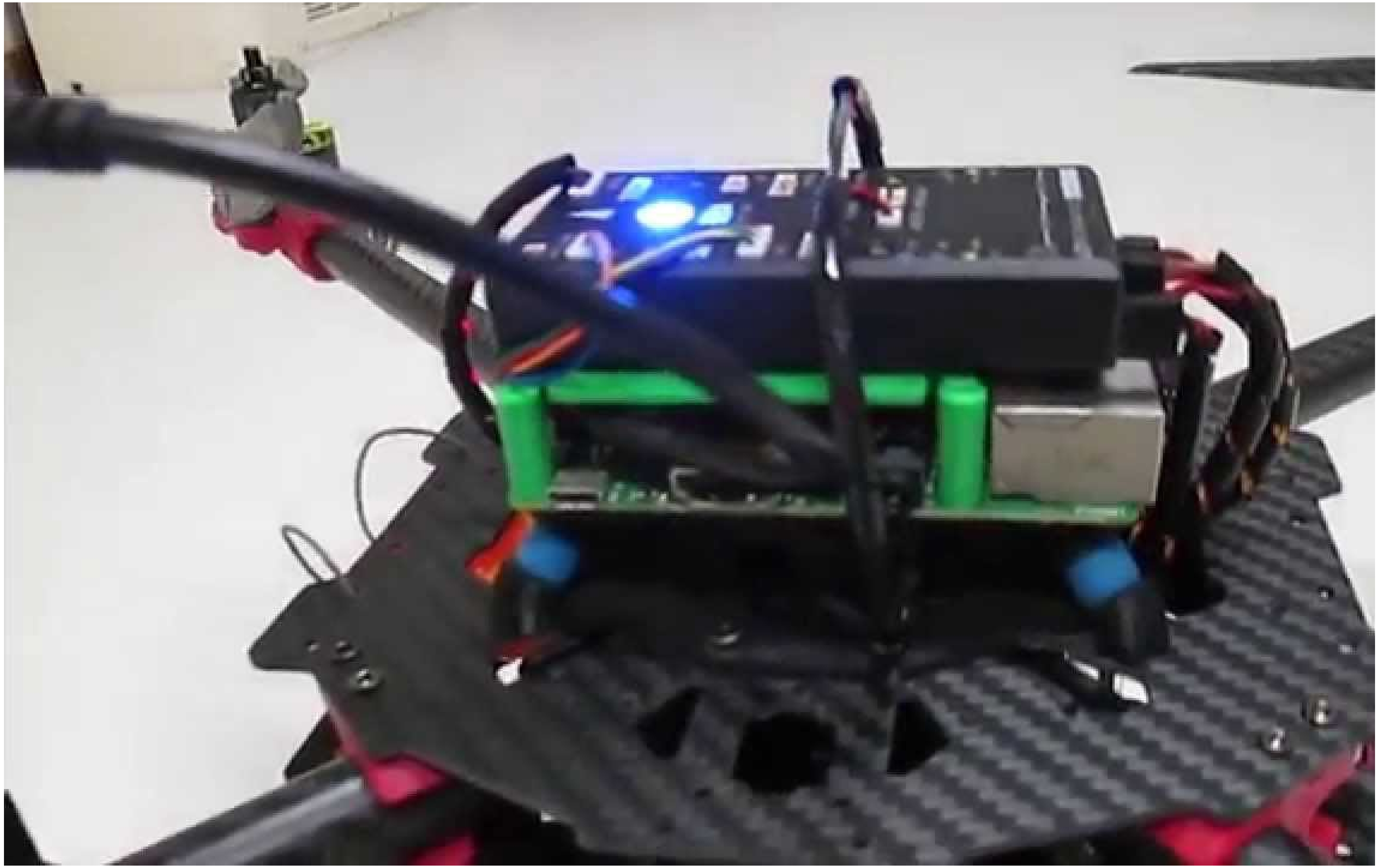
4. If time – UAV ethics ad-hoc debates.



Teaching groups of drones to fly in formation

https://www.youtube.com/watch?reload=9&v=i5mqWy-_VXY

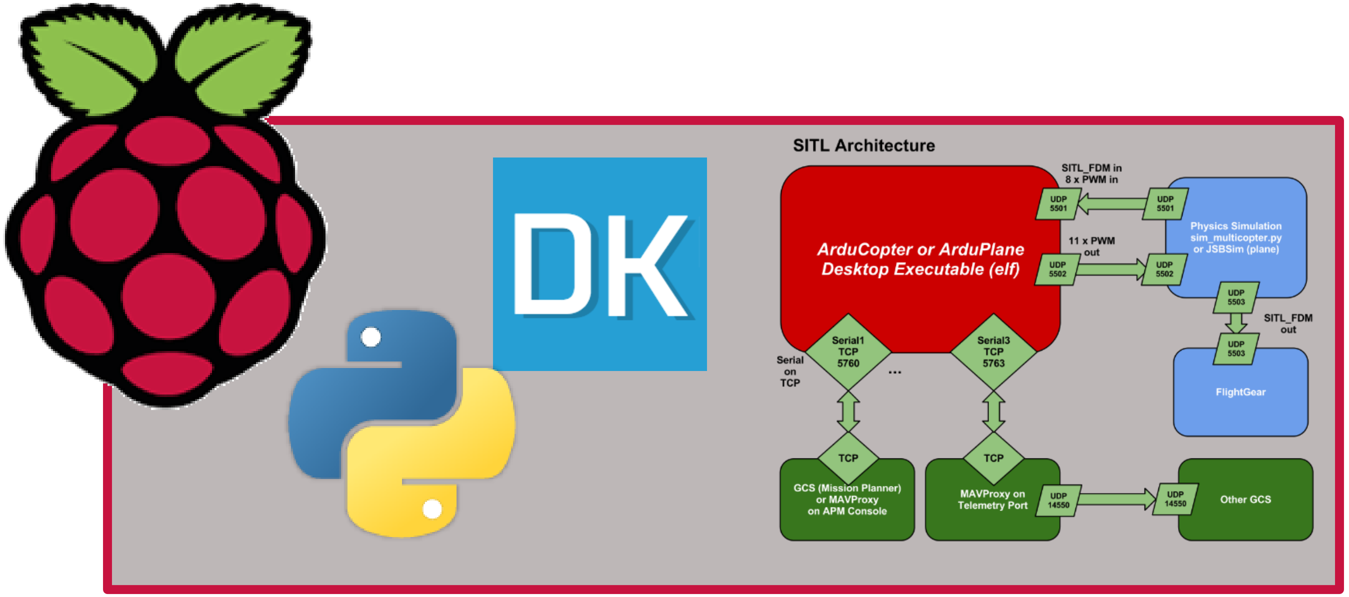
Pis on Board



We can attach a Raspberry Pi or other companion computer onto the UAV.

It can run DroneKit Python and send commands directly to the pixhawk controller in place of (or in addition to) the ground based code.

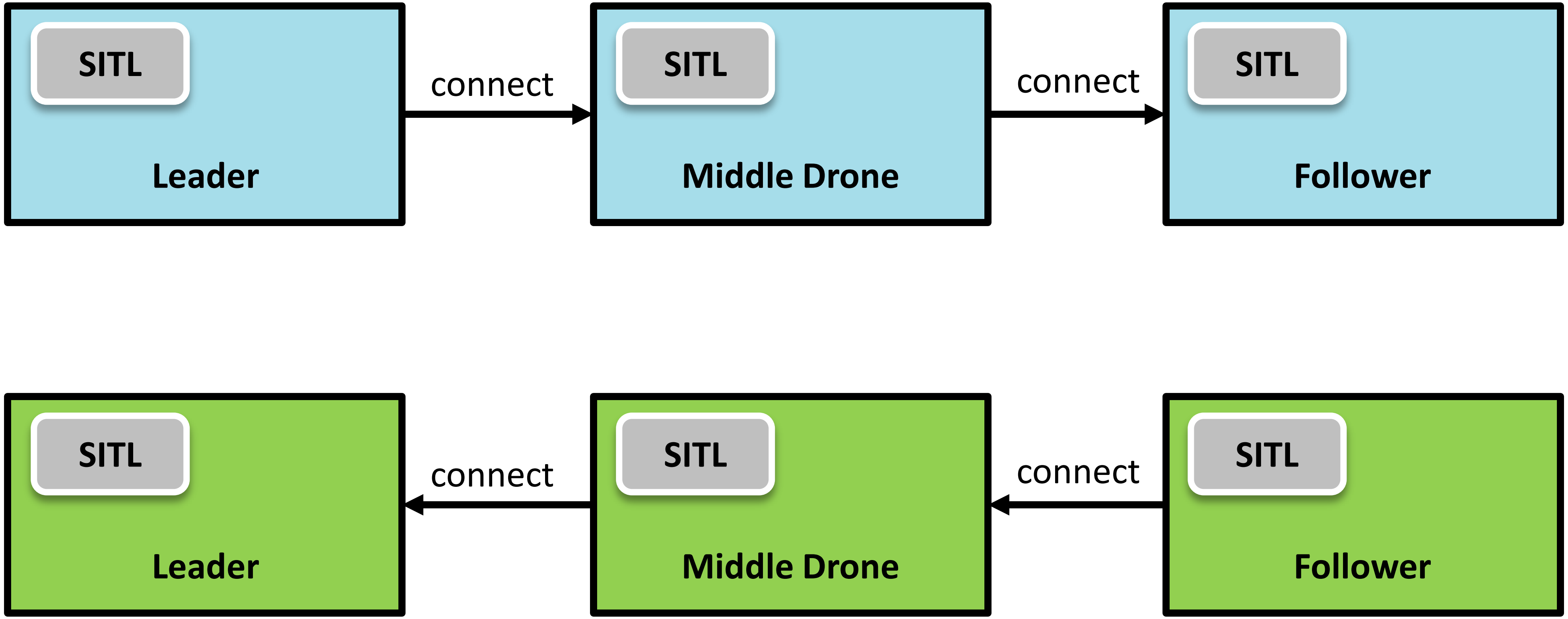
Learning to work with Pis



❶ Instead of putting the Pi on the Drone, we are going to put the Drone in the Pi.

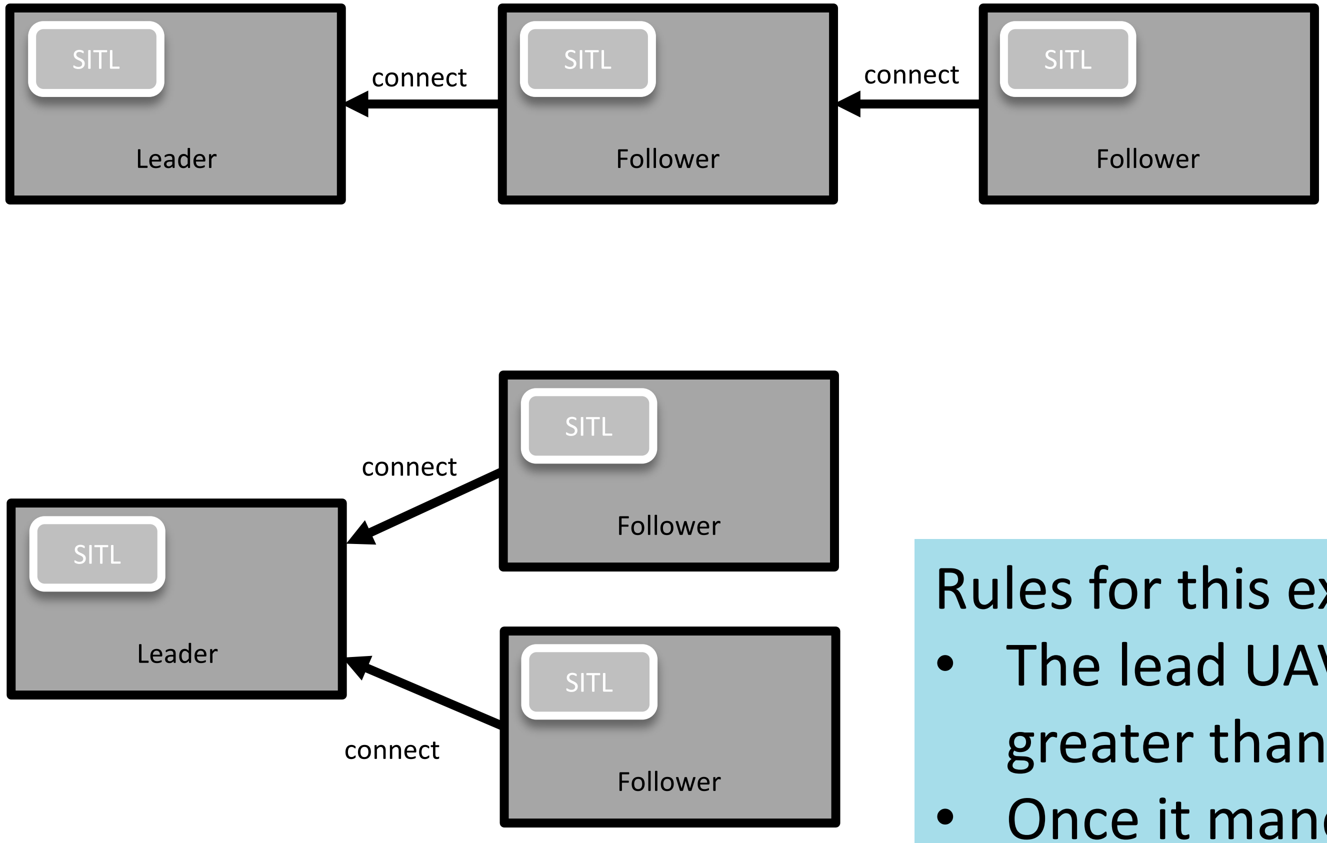
❷ We will simulate multiple UAVs flying in a chain using a string of three Pis.

Platooning



We will simulate multiple UAVs flying in a chain using a string of three Pis.

Two formations

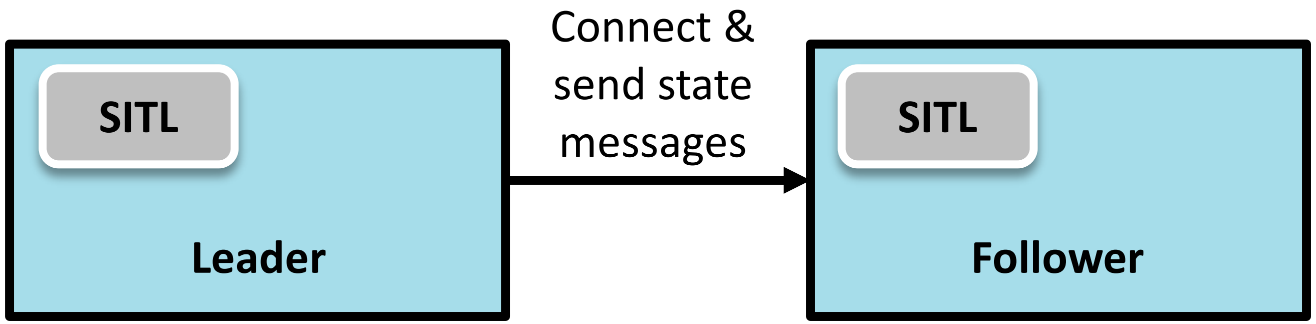


When flying in formation, UAVs need to
- Maintain correct separation distance.
- Maintain correct angles.

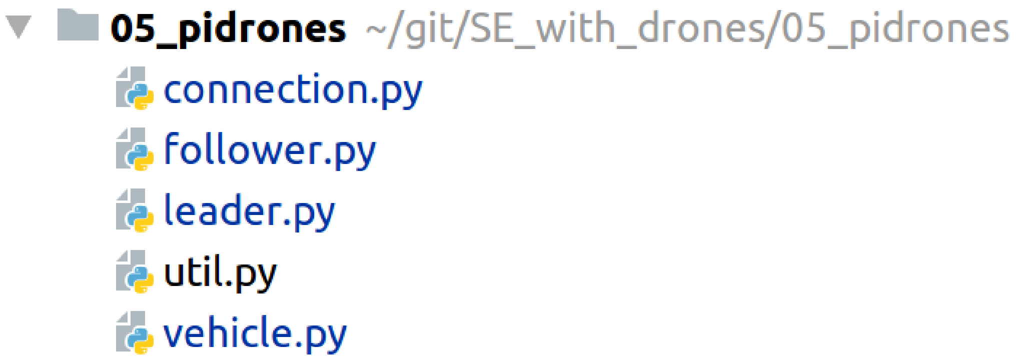
Even during maneuvers.

Rules for this exercise:
- The lead UAV will never maneuver at an angle greater than 30 degrees.
- Once it maneuvers, it will fly at least 5 meters before performing another maneuver.
- No leader will attempt to fly through its chain.
- All flights will use waypoints.

Class Exercise



We will provide a new repository (during class tonight) for each group to be used for this assignment.



We will add one team member as an admin to the repo, and that team member will need to add the other group members.

Leader

```python
import os
import time

from vehicle import FollowVehicle, LeadVehicle

ARDUPATH = os.path.join('/', 'home', 'uav', 'git', 'ardupilot')
CONNECTION_STRING = None

ADDRESS = "localhost"

if __name__ == '__main__':
    home = "41.714521, -86.241855"
    print("Starting Lead Vehicle...")
    time.sleep(5)


    leadVehicle= LeadVehicle()
    leadVehicle.start(ARDUPATH,CONNECTION_STRING,home,"Leader",ADDRESS,1234)

    leadVehicle.arm_and_takeoff(15)

    time.sleep(60)
    leadVehicle.land()
    time.sleep(60)
```

In this implementation, the leader registers with the follower.

You will need to replace the ADDRESS with the IP address of the follower.

Set home wherever you want it to be.

leader.py

LeadVehicle

NONE i.e., use SITL

Home coords

“Leader”

IP address of the follower

```
class LeadVehicle(Vehicle):

    def start(self, ardupath, connection_string, home, name, address, port):
        self.initializeVehicle(ardupath, connection_string, home, name)

        sender = MessageSender(name)
        sender.connect(address, port)
        threading.Thread(target=self.work, args=(sender,)).start()
```

❶ Establish connection to the follower using MessageSender.

```
    def work(self,sender):
        time.sleep(5)
        while True:
            state = util.message_from_vehicle(self.vehicle)
            sender.sendMessage(json.dumps(state))
            time.sleep(MESSAGE_FREQUENCY)
```

❷ Establish connection to the follower.

Vehicle.py

Leader as the message Sender

```python
class MessageSender():
    def __init__(self, id, ):
        self.id=id

    _WAITING = 1
    _CONNECTED = 2
    _DEAD = -1

    status = _WAITING

    def connect(self,address,port):
        threading.Thread(target=self.connectTo,args=(address,port,)).start()

    def connectTo(self,address,port):
        try:
            sock = socket.create_connection((address, port), timeout=5.0)
            self._sock = socketutils.BufferedSocket(sock)
            print(">>> Connected to lead vehicle "+ address + " on port  "+ str(port))
        except socket.error as e:
            print('-Socket error ({})'.format(e))
            print(e)
            time.sleep(10.0)

    def sendMessage(self,message):
        print("SEND:"+message)
        self._sock.send(message)
        self._sock.send(os.linesep)
```

Establish connection to the follower.

Open a socket for the connection.

Send a message.

connection.py

Follower Vehicle

```
import os
import time

from vehicle import FollowVehicle, LeadVehicle

ARDUPATH = os.path.join('/', 'home', 'uav', 'git', 'ardupilot')
CONNECTION_STRING = None

if __name__ == '__main__':
    home = "41.714521, -86.241855"
    print("Starting Follow Vehicle...")

    time.sleep(5)

    followVehicle = FollowVehicle()
    followVehicle.start(ARDUPATH, CONNECTION_STRING, home, "Follower", 1234)

    time.sleep(60)
```

The follower should probably be within the vicinity of the leader.

Think carefully about where you will set them up vs. how much takeoff synchronization you want to prevent crashes at startup.

follower.py

Vehicle.py

```
class FollowVehicle(Vehicle):

    def start(self, ardupath, connection_string, home, name,port):
        self.initializeVehicle(ardupath, connection_string, home, name)

        receiver = MessageReceiver("def",self)
        receiver.connect(port)

    status="ON_GROUND"

    def handleMessage(self,message):
        msg = json.loads(message)
```

Establish communication to the receiver using MessageReceiver.

Follower as MessageReceiver

```python
class MessageReceiver():
    def __init__(self, id,callback ):
        self.id = id
        self.callback=callback

    _WAITING = 1
    _CONNECTED = 2
    _DEAD = -1

    status = _WAITING

    def connect(self,port):
        threading.Thread(target=self.createConnection, args=(port,)).start()

    def createConnection(self, port):
        try:
            serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            serv.bind(('0.0.0.0', port))
            print(">>> Opening connection on port  " + str(port))
            serv.listen(5)
            conn, addr = serv.accept()
            print(">>> Connection established")
            self._sock = socketutils.BufferedSocket(conn)
            while True:
                msg = self._sock.recv_until(os.linesep, timeout=50)
                print("RECEIVED: "+msg)
                self.callback.handleMessage(msg)
        except socket.error as e:
            print('>Socket error ({})'.format(e))
            time.sleep(10.0)
```

In FollowVehicle….

```python
def handleMessage(self,message):
    msg = json.loads(message)
```

Establish socket connection

Receive incoming messages. Uses a callback to handle messages.

Wifi Routers

<u>Stinson Remmick:</u>
SSID: NDSUAS
pwd: NotreDame

Locker Combinations:
101: 28-18-04
102: 16-30-24

Please remember to return everything and lock up the locker when you finish.

<u>Area around my office (aka Bui's Hall)</u>

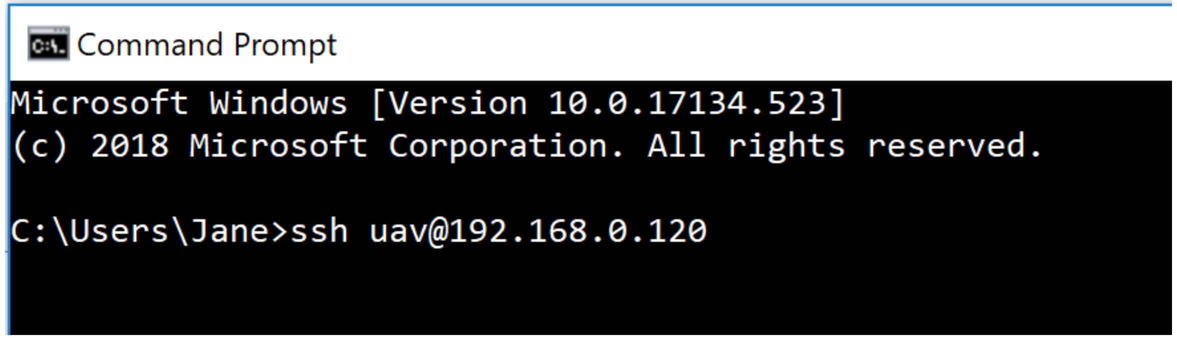
SSID: NDSUAS-2
Pwd: NotreDame

Changing the Pi Wifi Connection

https://www.raspberrypi.org/documentation/configuration/raspi-config.md

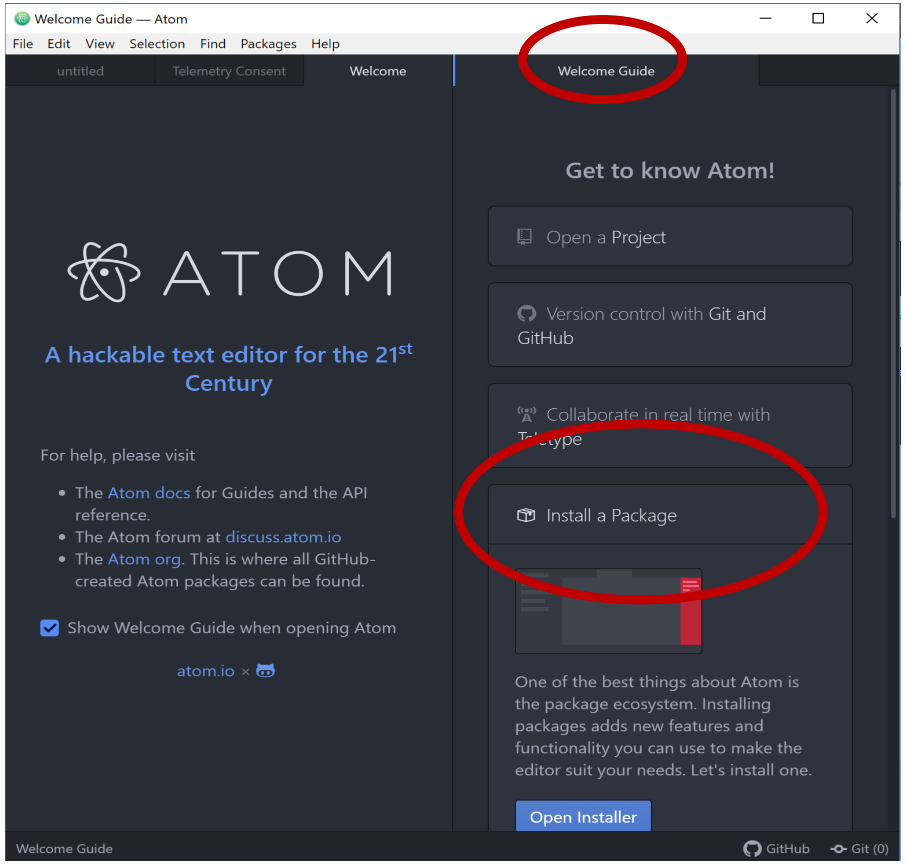
Connect to the local wifi router before trying to connect to the Pis. Note, it is tricky to connect via any ND network.

- ssh into your Pi.
  uav@HOSTNAME   or
  uav@192.168.0.1##
  (where ## = Pi Number.)
- Password = uav2019
- Create the directory that you want to work in e.g.,   /home/uav/dev/simpleconnect



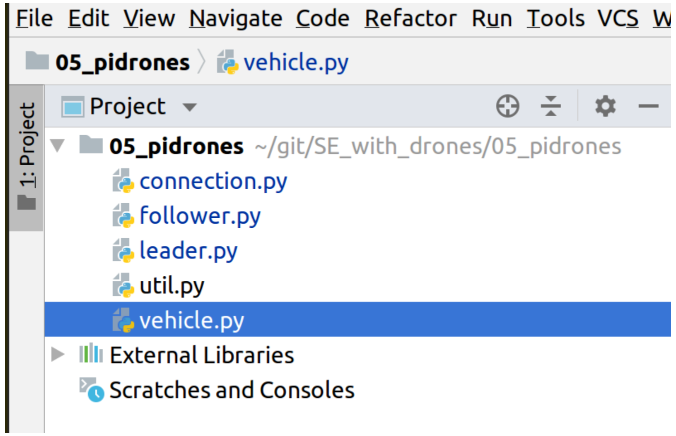
Working with Atom

https://flight-manual.atom.io/getting-started/sections/installing-atom/#platform-linux

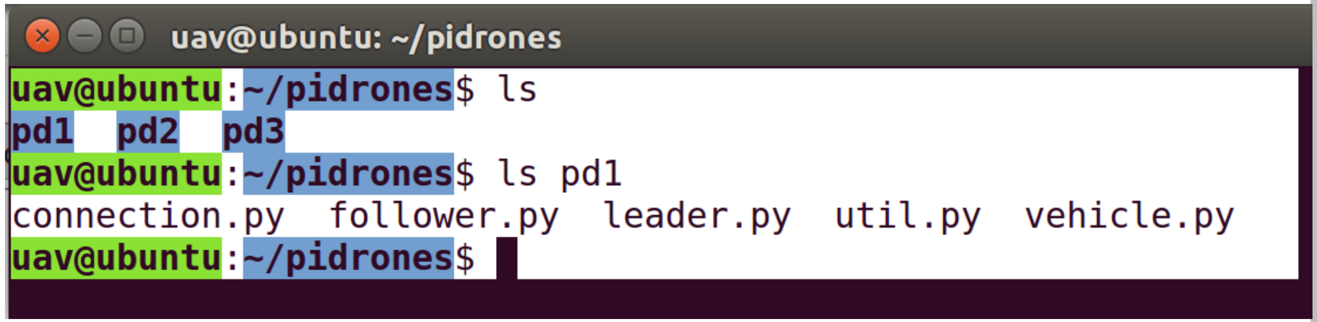


❶ Click on Welcome Guide tab.
Install package: remote-ftp plugin

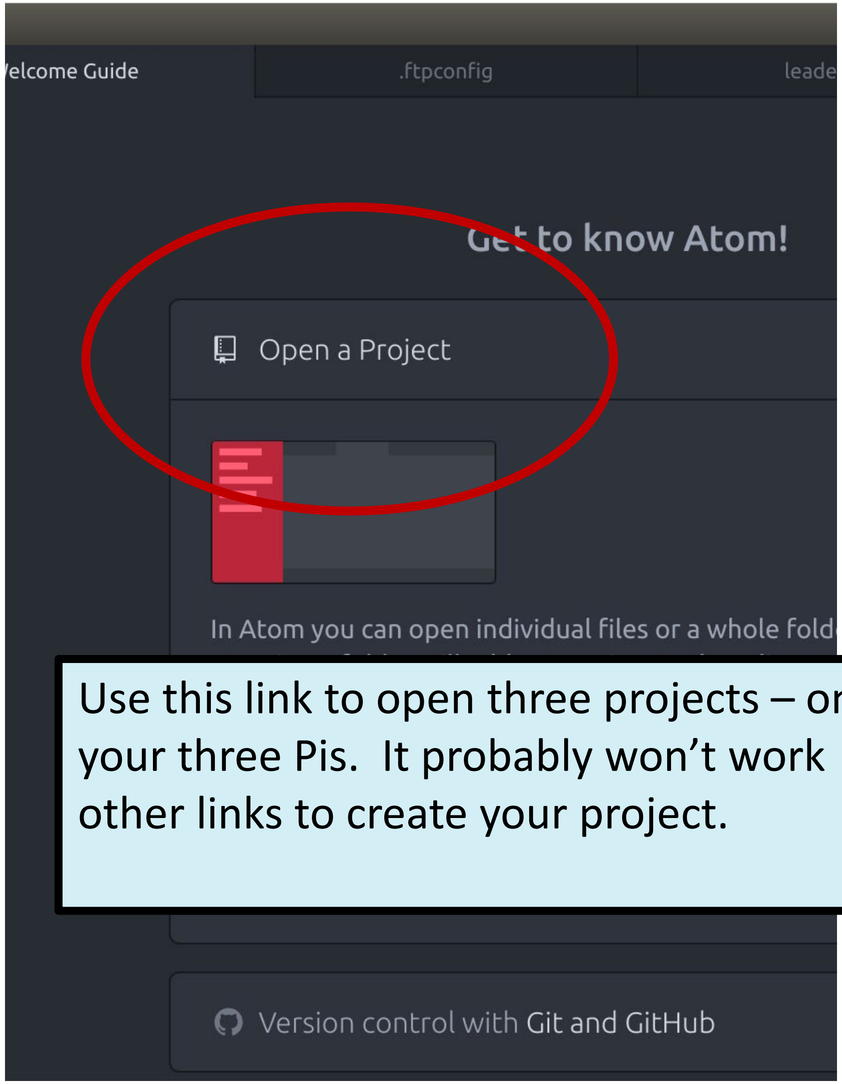
❷ Check out the SE_with_drones repository



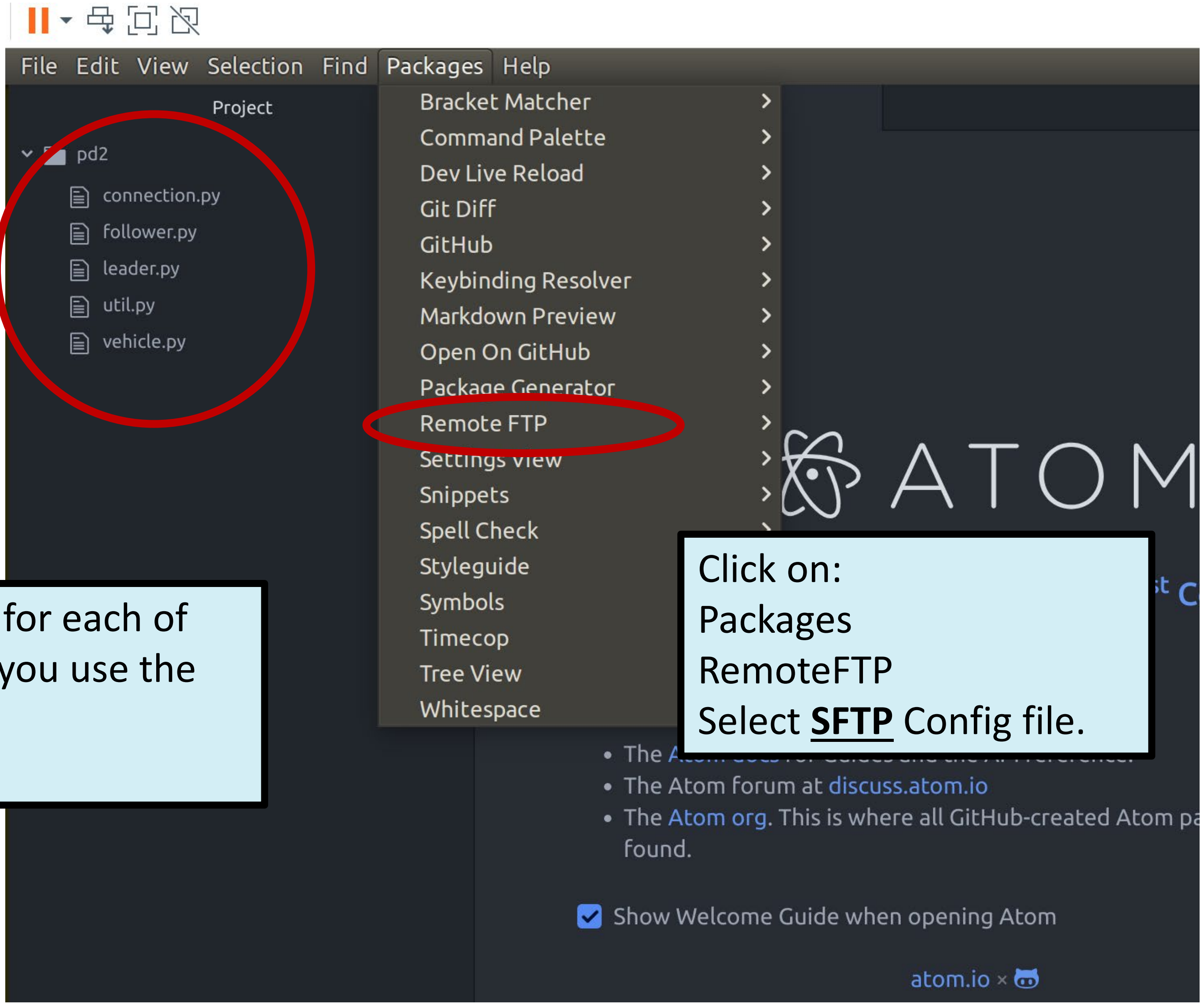
❸ Create a pidrones directory & a sub-directory for each of the 3 Pis. Copy the 05_pidrones to each dir.



Working with Atom

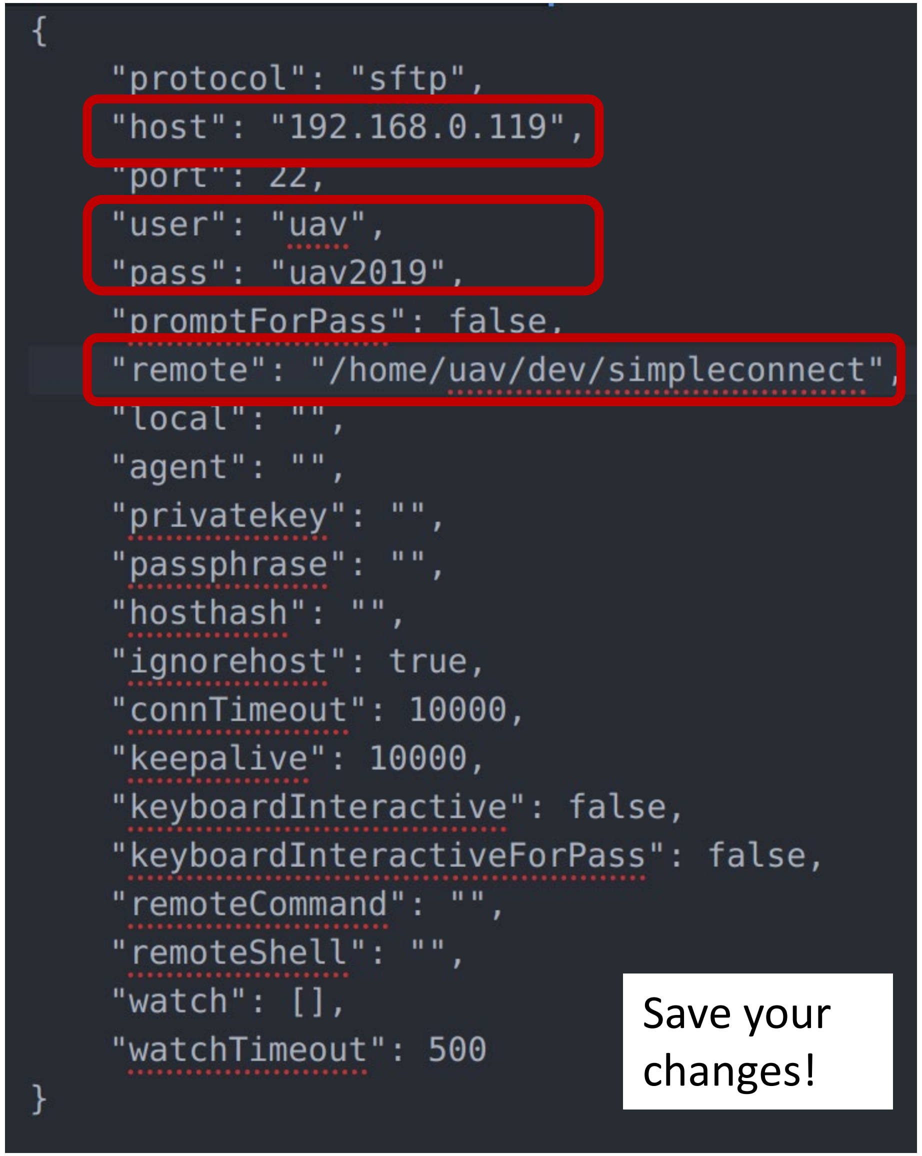


Use this link to open three projects – one for each of your three Pis. It probably won't work if you use the other links to create your project.

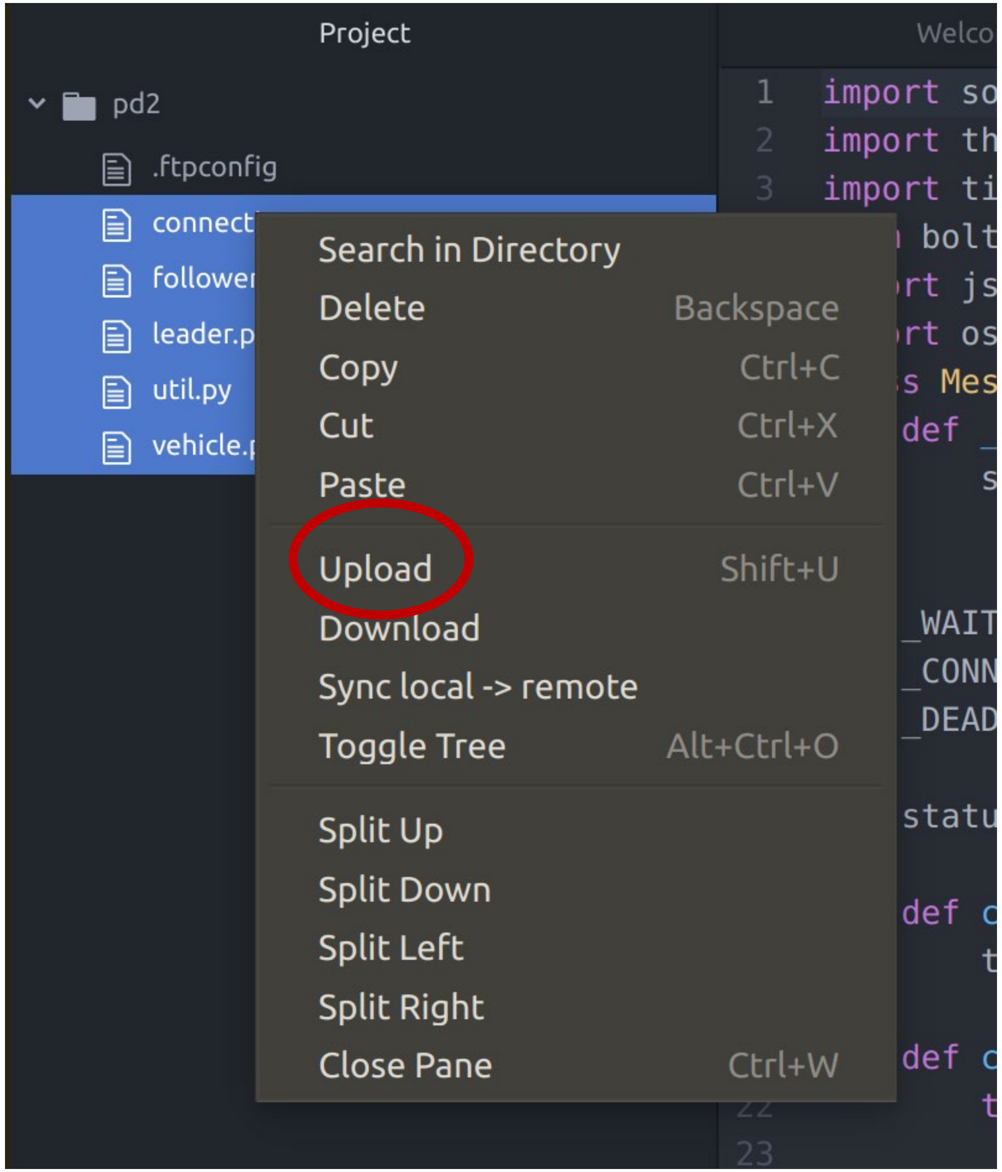


Click on:
Packages
RemoteFTP
Select **SFTP** Config file.

Working with Atom



Save your changes!



On the Pi

uav@uavclass-pi-19: ~

```
C:\Users\Jane>ssh uav@192.168.0.119
ssh: connect to host 192.168.0.119 port 22: Connection aborted

C:\Users\Jane>ssh uav@192.168.0.119
uav@192.168.0.119's password:
Linux uavclass-pi-19 4.14.79-v7+ #1159 SMP Sun Nov 4 17:50:20 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan 23 05:03:15 2019 from 192.168.0.121
uav@uavclass-pi-19:~ $
```

uav@uavclass-pi-20: ~

```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Jane>ssh uav@192.168.0.120
uav@192.168.0.120's password:
Permission denied, please try again.
uav@192.168.0.120's password:
Linux uavclass-pi-20 4.14.79-v7+ #1159 SMP Sun Nov 4 17:50:20 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan 23 07:42:19 2019 from 192.168.0.121
uav@uavclass-pi-20:~ $
```

```
Follower:
PD1: PI-19
192.168.0.119

Leader:
PD2: PI-20:
192.168.0.120
```

1. SSH into both of the Pis.
2. Keep a clear plan in your head for the role of each Pi and the directory (project) it is associated with in Atom.
3. The current implementation requires you to start the follower first.
4. Connect to Follower Pi and from the correct directory
   python follower.py
5. Repeat for the Leader Pi.
   python leader.py

Starting them both up..

Follower

uav@uavclass-pi-19: ~/dev/simpleconnect

```
uav@uavclass-pi-19:~ $ uav@uavclass-pi-19:~ $ cd dev
uav@uavclass-pi-19:~/dev $ ls
simpleconnect  simpleconnectxx  simplefollow
uav@uavclass-pi-19:~/dev $ cd simpleconnect
uav@uavclass-pi-19:~/dev/simpleconnect $ python follower.py
Starting Follow Vehicle...
Connecting to vehicle on: tcp:127.0.0.1:5760
>>> Calibrating barometer
>>> ArduCopter V3.6-dev (30b3d63e)
>>> Frame: QUAD
>>> Barometer calibration complete
>>> GPS 1: detected as u-blox at 115200 baud
>>> Opening connection on port  1234
>>> EKF2 IMU0 initial yaw alignment complete
>>> EKF2 IMU1 initial yaw alignment complete
>>> EKF2 IMU0 tilt alignment complete
>>> EKF2 IMU1 tilt alignment complete
>>> EKF2 IMU0 Origin set to GPS
>>> EKF2 IMU1 Origin set to GPS
```

Leader

uav@uavclass-pi-20: ~/dev/simpleconnect

```
uav@uavclass-pi-20:~/dev/simpleconnect $ uav@uavclass-pi-20:~/dev/simpleconnec
t $ python leader.py
Starting Lead Vehicle...
```

Note: The need to start them in the correct order makes the program overly fragile. You'll need to fix this for your group homework.

Follower starts up.
Leaders starts up, connects to the follower, and takes off. (Starts flying)
Leader sends state messages to the follower.

Shutting down the Pis

uav@uavclass-pi-19: ~/dev/simpleconnect

Follower

```
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.0080000003
", "location": {"y": -86.2418499, "x": 41.7145184, "z": 14
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.00400000189989805, "mode": "GUIDE
D", "location": {"y": -86.2418498, "x": 41.7145184, "z": 14.986}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.0020000000949949026, "mode": "GUID
ED", "location": {"y": -86.2418499, "x": 41.7145183, "z": 14.988}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.006000000052154064, "mode": "GUIDE
D", "location": {"y": -86.2418499, "x": 41.7145184, "z": 14.994}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.005000000353902578, "mode": "GUIDE
D", "location": {"y": -86.2418498, "x": 41.7145184, "z": 14.997}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.006000000052154064, "mode": "GUIDE
D", "location": {"y": -86.2418499, "x": 41.7145185, "z": 14.997}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.01100000087171793, "mode": "GUIDED
", "location": {"y": -86.2418499, "x": 41.7145184, "z": 14.996}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 0.07200000435113907, "mode": "GUIDED
", "location": {"y": -86.2418502, "x": 41.7145187, "z": 14.995}, "heading": 356}
RECEIVED: {"status": "ACTIVE", "groundspeed": 1.537000060081482, "mode": "GUIDED",
 "location": {"y": -86.2418657, "x": 41.7145351, "z": 14.989}, "heading": 321}
RECEIVED: {"status": "ACTIVE", "groundspeed": 3.205000162124634, "mode": "GUIDED",
 "location": {"y": -86.2419002, "x": 41.7145739, "z": 14.994}, "heading": 323}
RECEIVED: {"status": "ACTIVE", "groundspeed": 4.076000213623047, "mode": "GUIDED",
 "location": {"y": -86.2419509, "x": 41.7146279, "z": 14.994}, "heading": 323}
RECEIVED: {"status": "ACTIVE", "groundspeed": 4.492000102996826, "mode": "GUIDED",
 "location": {"y": -86.2420168, "x": 41.7146969, "z": 14.994}, "heading": 324}
RECEIVED: {"status": "ACTIVE", "groundspeed": 4.712000370025635, "mode": "GUIDED",
 "location": {"y": -86.2420793, "x": 41.7147621, "z": 14.995}, "heading": 325}
RECEIVED: {"status": "ACTIVE", "groundspeed": 4.843000411987305, "mode": "GUIDED",
```

uav@uavclass-pi-20: ~/dev/simpleconnect

Leader

```
SEND:{"status": "ACTIVE", "groundspeed": 0.0110000008717
, "location": {"y": -86.2418499, "x": 41.7145184, "z": 1
}
STARTING TO MOVE:
SEND:{"status": "ACTIVE", "groundspeed": 0.07200000435113907, "mode": "GUIDED"
, "location": {"y": -86.2418502, "x": 41.7145187, "z": 14.995}, "heading": 356
}
SEND:{"status": "ACTIVE", "groundspeed": 1.537000060081482, "mode": "GUIDED",
"location": {"y": -86.2418657, "x": 41.7145351, "z": 14.989}, "heading": 321}
SEND:{"status": "ACTIVE", "groundspeed": 3.205000162124634, "mode": "GUIDED",
"location": {"y": -86.2419002, "x": 41.7145739, "z": 14.994}, "heading": 323}
SEND:{"status": "ACTIVE", "groundspeed": 4.076000213623047, "mode": "GUIDED",
"location": {"y": -86.2419509, "x": 41.7146279, "z": 14.994}, "heading": 323}
SEND:{"status": "ACTIVE", "groundspeed": 4.492000102996826, "mode": "GUIDED",
"location": {"y": -86.2420168, "x": 41.7146969, "z": 14.994}, "heading": 324}
SEND:{"status": "ACTIVE", "groundspeed": 4.712000370025635, "mode": "GUIDED",
"location": {"y": -86.2420793, "x": 41.7147621, "z": 14.995}, "heading": 325}
SEND:{"status": "ACTIVE", "groundspeed": 4.843000411987305, "mode": "GUIDED",
"location": {"y": -86.2421516, "x": 41.714838, "z": 14.994}, "heading": 325}
SEND:{"status": "ACTIVE", "groundspeed": 4.878000259399414, "mode": "GUIDED",
"location": {"y": -86.2422164, "x": 41.7149067, "z": 14.994}, "heading": 325}
SEND:{"status": "ACTIVE", "groundspeed": 3.8550002574920654, "mode": "GUIDED",
 "location": {"y": -86.2422734, "x": 41.7149674, "z": 15.002}, "heading": 326}
SEND:{"status": "ACTIVE", "groundspeed": 1.0240000486373901, "mode": "GUIDED",
 "location": {"y": -86.2423056, "x": 41.7150021, "z": 15.007}, "heading": 328}
SEND:{"status": "ACTIVE", "groundspeed": 0.15300001204013824, "mode": "GUIDED"
, "location": {"y": -86.2423064, "x": 41.7150031, "z": 15.006}, "heading": 328
```

Messages are received by the follower.

The follower currently just prints them; however it needs to act on them.

- When should it take off?
- How should it fly?

Shutting down the Pis

Command Prompt

```
ast login: Wed Jan 23 01:19:58 2019 from 192.168.0.121
uav@uavclass-pi-20:~ $ ls
bootstraph.sh  bootstrap.log  dev  git
uav@uavclass-pi-20:~ $ cd dev
uav@uavclass-pi-20:~/dev $ ls
simpleconnectxx
uav@uavclass-pi-20:~/dev $ mkdir simpleconnect
uav@uavclass-pi-20:~/dev $ ls
simpleconnect  simpleconnectxx
uav@uavclass-pi-20:~/dev $ ls
simpleconnect  simpleconnectxx
uav@uavclass-pi-20:~/dev $ cd simpleconnect
uav@uavclass-pi-20:~/dev/simpleconnect $ ls
uav@uavclass-pi-20:~/dev/simpleconnect $ ls
uav@uavclass-pi-20:~/dev/simpleconnect $ cd ..
uav@uavclass-pi-20:~/dev $ cd /home/uav/dev/simpleconnect
uav@uavclass-pi-20:~/dev/simpleconnect $ ls
uav@uavclass-pi-20:~/dev/simpleconnect $ cd ../
uav@uavclass-pi-20:~/dev $ cd simpleconnect
uav@uavclass-pi-20:~/dev/simpleconnect $ ls
connection.py  follower.py  leader.py  util.py  vehicle.py
uav@uavclass-pi-20:~/dev/simpleconnect $ sudo shutdown
[sudo] password for uav:
Shutdown scheduled for Wed 2019-01-23 06:13:20 EST, use 'shutdown -c' to cance
l.
uav@uavclass-pi-20:~/dev/simpleconnect $ Connection to 192.168.0.120 closed by
 remote host.
Connection to 192.168.0.120 closed.

C:\Users\Jane>
```

Please shut down Pis properly.

SSH to Pi

Sudo shutdown

… then you can turn off the Pi.

Lab Part #1

1. Each team member should set up one Pi.
2. For this lab, one person should setup the Leader and two people should set up followers.
3. Follow instructions from the slides so that the follower receives state messages from the leader as the leader takes off.

This 2 week group assignment is **for real**. It will become part of your flying portfolio and you will be required to test this in White Field **without crashing any drones!!!**

Lab Challenge:

1. Decide when the follower should take off. Add the code to make it take off (be cognizant of potential collisions).
2. Send the leader to a waypoint.
3. Have your follower follow the leader. Think about what it means to follow.

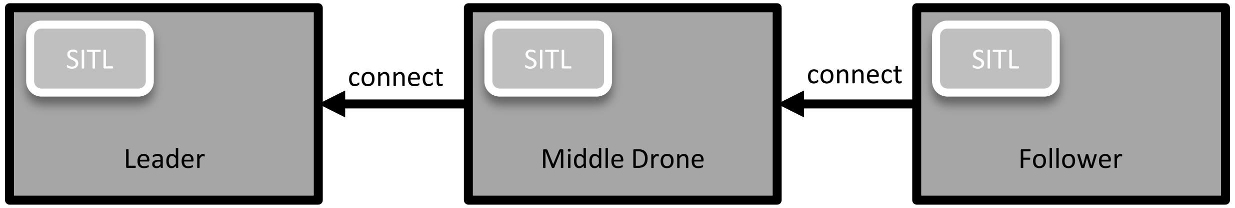
Collisions:

For purposes of the lab, we won't worry about drones on top of each other or colliding with each other. However, that will matter in the homework.

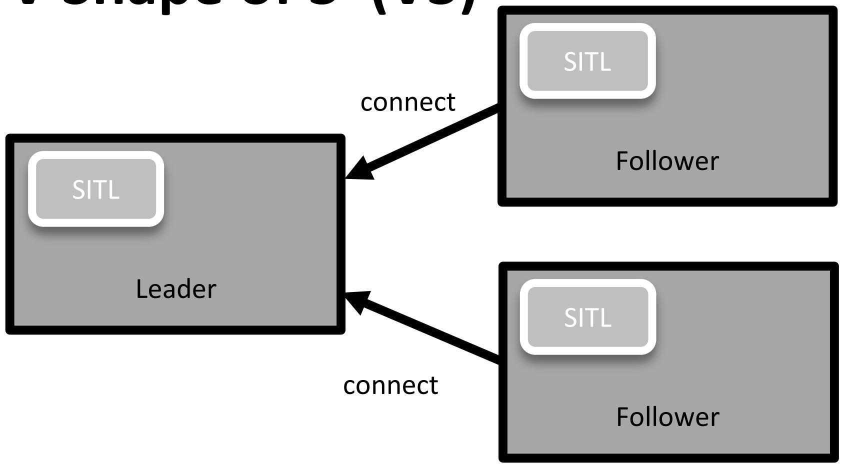
About the Homework

You will need to setup two different formations.

Platoon of 3 (P3)



V Shape of 3 (V3)



How are you going to test it?

You might need some code for this because it is not entirely simple to compute accurate lat and lon as an offset, especially if you need to accurately take into account the curvature of the earth.

Offsets

```python
import numpy as np
import nvector as nv
from math import radians, degrees

def pvec_b_to_lla(forward, right, down, roll, pitch, yaw, lat, lon, alt):
    """
    returns the lat lon and alt corresponding to a p-vec in the UAV's body frame

    Parameters
    ----------
    forward: float
        The number of meters forward of the UAV
    right: float
        The number of meters to the right of th
    down: float
        The number of meters below the UAV
    roll: float
        The UAV's roll angle in degrees
    pitch: float
        The UAV's pitch angle in degrees
    yaw: float
        The UAV's yaw angle in degrees
    lat: float
        The UAV's latitude in degrees
    lon: float
        The UAV's longitude in degrees
    alt: float
        The UAV's altitude in meters
```

```python
    Returns
    -------
    list
        This list holds three floats representing the latitude in degrees, longitude in degrees and altitude in meters (in that order).
    """

    # create a p-vector with the forward, right and down values
    p_B = np.array([forward, right, down])

    # this matrix can transform a pvec in the body frame to a pvec in the NED frame
    rot_NB = nv.zyx2R(radians(yaw), radians(pitch), radians(roll))

    # calculate the pvec in the NED frame
    p_N = rot_NB.dot(p_B)

    # create an n-vector for the UAV
    n_UAV = nv.lat_lon2n_E(radians(lat), radians(lon))

    # this creates a matrix that rotates pvecs from NED to ECEF
    rot_EN = nv.n_E2R_EN(n_UAV)

    # find the offset vector from the UAV to the point of interest in the ECEF frame
    p_delta_E = rot_EN.dot(p_N)

    # find the p-vector for the UAV in the ECEF frame
    p_EUAV_E = nv.n_EB_E2p_EB_E(n_UAV, -alt).reshape(1,3)[0]

    # find the p-vector for the point of interest. This is the UAV + the offset in the ECEF frame.
    p_E = p_EUAV_E + p_delta_E
```

```python
    # find the n-vector for the point of interest given the p-vector in the ECEF frame.
    n_result, z_result = nv.p_EB_E2n_EB_E(p_E.reshape(3,1))

    # convert the n-vector to a lat and lon
    lat_result, lon_result = nv.n_E2lat_lon(n_result)
    lat_result, lon_result = degrees(lat_result), degrees(lon_result)

    # convert depth to alt
    alt_result = -z_result[0]

    return [lat_result, lon_result, alt_result]
```

Lab Part #2

- Consider what can go wrong. You are eventually going to fly your solution as long as you can convince JCH that it is safe.
- We'll have an additional round of tests later in February; however, for now, please design, program, and perform initial tests with safety in mind.
- Create a spreadsheet list the hazards you've identified, the potential adverse outcome

Hazard	Adverse outcome	Criticality	Mitigation
Communication is lost between a lead drone and its follower.	The follower flies into the drone in front of it.	High	?
A drone plummets to the ground (or lands abruptly due to battery failure).	The follower follows its path all the way to the ground and crashes.	High	?

Lab Part #2

- Take a few minutes for each team member to design a solution that addresses your key hazards.
- Consider the following design issues:
  - The system must not require you to startup drones in any particular order.
  - The system must handle temporarily lost socket connections. (Monkey tests?)
  - Drones must *not* collide under any circumstance.
  - Your design should be able to accommodate longer chains and more interesting patterns (i.e., extensibility). Achieving this will definitely be a plus for your group.

Sketch your solution on a white board. Take a picture of it and transform it into your group design.

Make sure you understand the extent to which your design can deliver functional and quality requirements.