Software Development for Unmanned Aerial Systems

Instructor:

Jane Cleland-Huang, PhD

JaneClelandHuang@nd.edu

Department of Computer Science and Engineering

University of Notre Dame

Drones as Athletes

https://www.ted.com/talks/raffaello_d_andrea_the_astounding_athletic_power_of_quadcopters

Today's Agenda

- Drone Athletes
- Maneuverability with waypoints and velocity vectors
- Hello Physical Drone

- Air Classes
- Team Challenges

How to reference a location

Location Global

http://python.dronekit.io/automodule.html#dronekit.LocationGlobal

Three parameters: **lat**, **lon**, **alt** (relative to mean sea level).

```
LocationGlobal(-34.364114, 149.166022, 30)
```

currentLocation = vehicle.location.global_frame

Location Relative Global

http://python.dronekit.io/automodule.html#dronekit.LocationGlobalRelative

lat, lon, alt – Altitude in meters (relative to the home location).

```
LocationGlobalRelative(-34.364114, 149.166022, 30)
```

currentLocation = vehicle.location.global_relative_frame

vehicle.simple_goto(41.705078, -86.240525,30)

Why do you think my SITL drone **repeatedly crashed** when I sent it this command to fly to Duck Island on ND campus?

Two ways to fly…

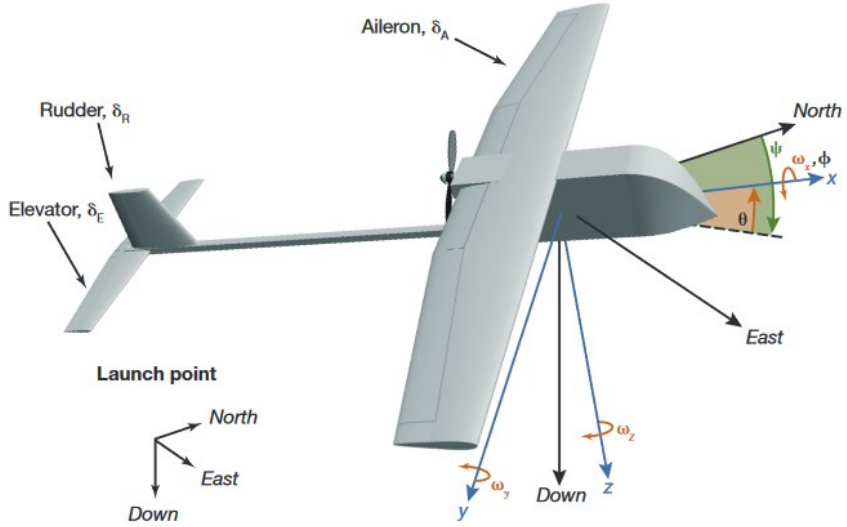
Waypoints:

Waypoints define GPS coordinates based on latitude, longitude, and altitude.

Waypoints are ideal for defining missions based on known routes with clearly defined target locations.

Drones can fly quickly between those routes.

Velocity Vectors:



Velocity vectors are defined in terms of North, East, and Down magnitudes.

Mavlink supports:

- MAV_FRAME_BODY_NED: x: north, y: east, z: down
- MAV_FRAME_LOCAL_NED: relative to current position of body. (i.e., north = forward, south = reverse, z = relative down)

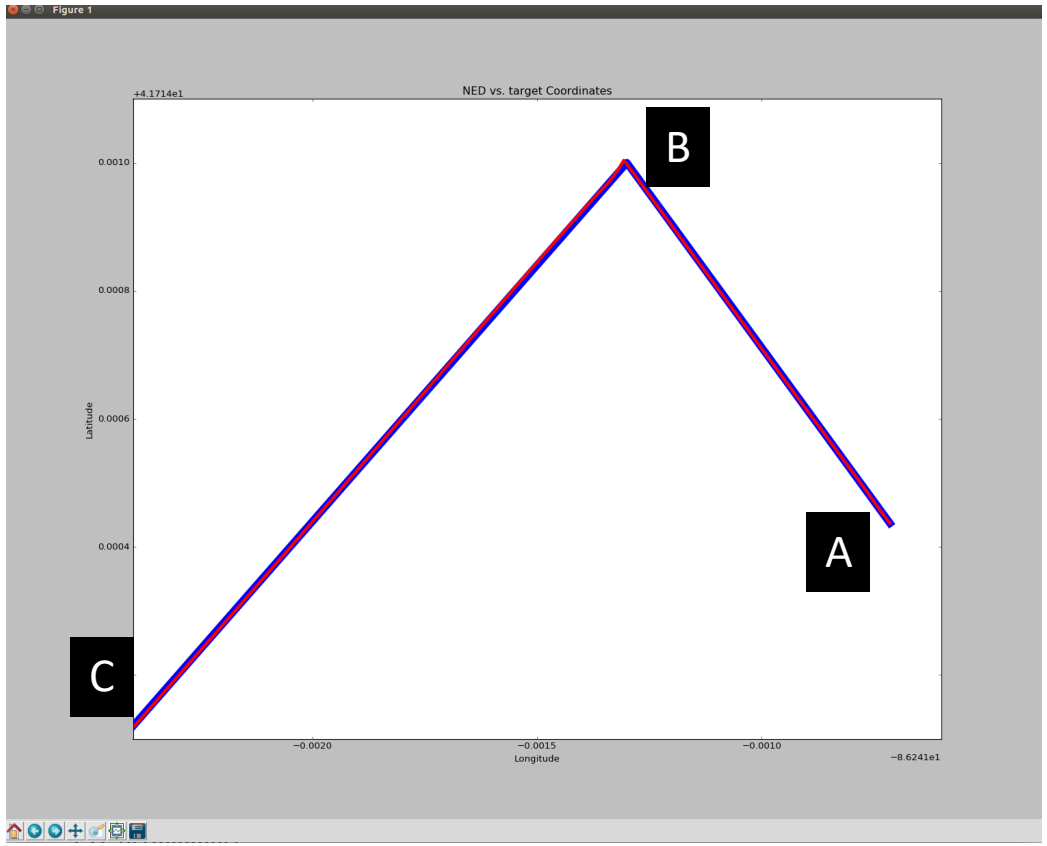
Waypoints

Fly_to(vehicle,LocationGlobalRelative(41.71500, -86.24230, 20), 10, “START_TAG”,”END_TAG”)

```
################################################################################
# Fly to
################################################################################
def fly_to(vehicle, targetLocation, groundspeed, startTag, endTag):
    print "Flying from: " + str(vehicle.location.global_relative_frame.lat) + "," + str(vehicle.location.glo
    vehicle.groundspeed = groundspeed
    currentTargetLocation = targetLocation
    vehicle.simple_goto(currentTargetLocation)
    remainingDistance=get_distance_meters(currentTargetLocation,vehicle.location.global_relative_frame)

    while vehicle.mode.name=="GUIDED":
        remainingDistance=get_distance_meters(currentTargetLocation,vehicle.location.global_relative_frame)
        print remainingDistance
        if remainingDistance< 1:
            print "Reached target"
            break
        time.sleep(1)
```

You can also establish a mission:
http://python.dronekit.io/examples/mission_basic.html

Plotting Flights



This example plots the coordinates of the UAV flying from A to B and then B to C using waypoints.

You can create your own plotting functions or use the ones provided:

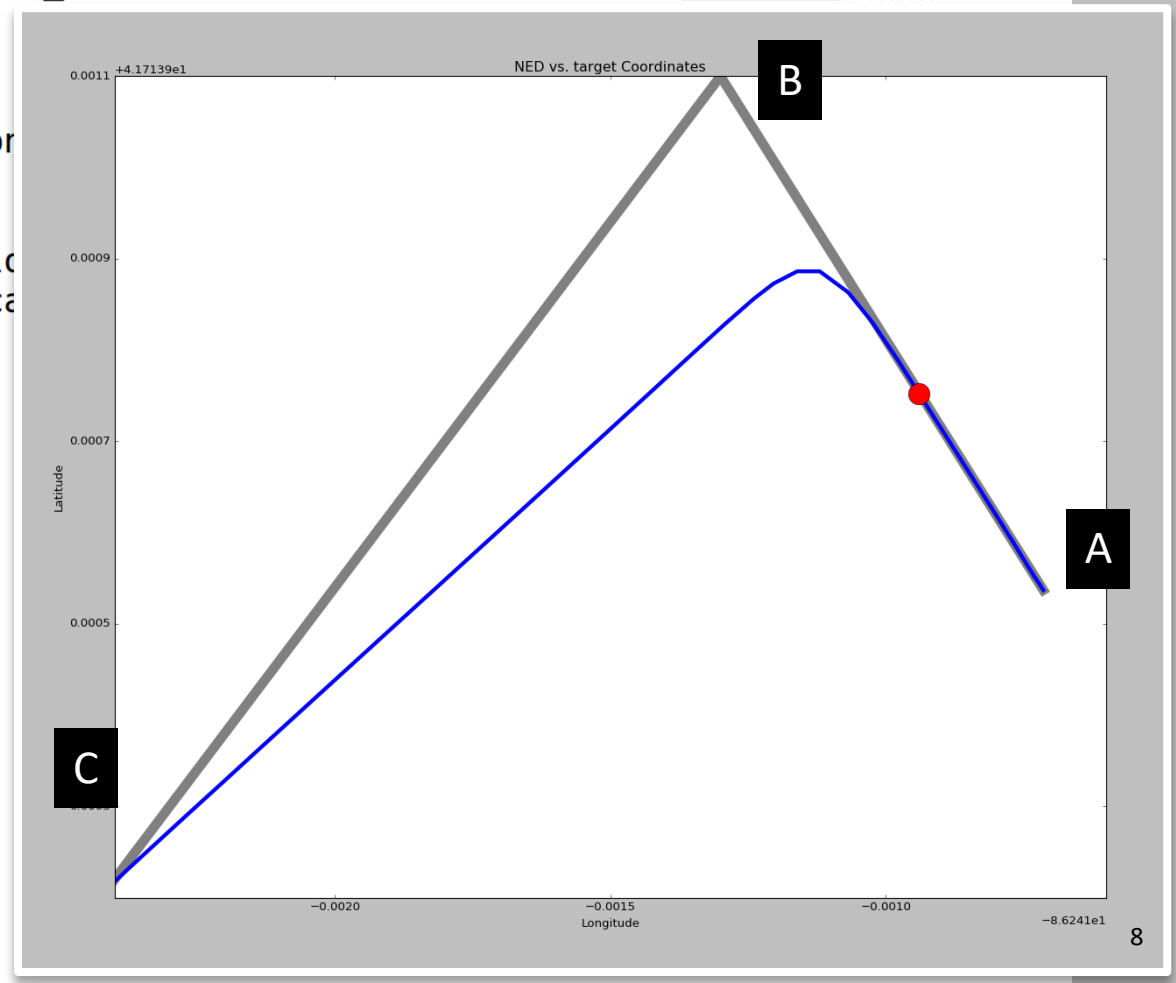
```
from ned.flight_plotter import Location,  ❶
CoordinateLogger, GraphPlotter
log1 = CoordinateLogger()
log2 = CoordinateLogger()
log1.add_data(-86.24230,41.71500)
```

```
plotter = GraphPlotter(log1.lat_array, log1.lon_array, log2.lat_array, log2.lon_array, "Longitude",
            "Latitude", "NED vs. target Coordinates")
 plotter.scatter_plot()
```

Maneuvers with WayPoints

```
##########################################################################################
# Fly to
##########################################################################################
def fly_to(vehicle, targetLocation, groundspeed, startTag, endTag,abortdistance):
    print "Flying from: " + str(vehicle.location.global_relative_frame.lat) + "," + str(vehicle.location.glob
    vehicle.groundspeed = groundspeed
    currentTargetLocation = targetLocation
    vehicle.simple_goto(currentTargetLocation)
    remainingDistance=get_distance_meters(currentTargetLocatior

    while vehicle.mode.name=="GUIDED":
        log2.add_data(vehicle.location.global_relative_frame.lo
        remainingDistance=get_distance_meters(currentTargetLoca
        print remainingDistance
        if remainingDistance< abortdistance:
            print "Aborted target waypoint"
            break
        time.sleep(1)
```

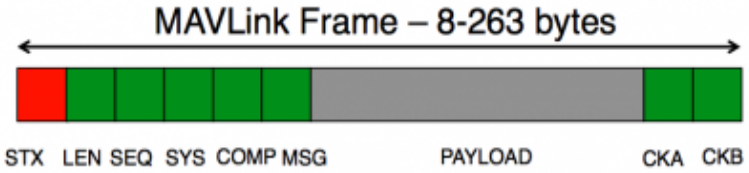
What if the UAV is suddenly issued a new waypoint?



How do Velocity Vectors work?

Field name	Index (Bytes)	Purpose
Start-of-frame	0	Denotes the start of frame transmission (v1.0: 0xFE)
Pay-load-length	1	length of payload (n)
Packet sequence	2	Each component counts up his send sequence. Allows to detect packet loss
System ID	3	Identification of the SENDING system. Allows to differentiate different systems on the same network.
Component ID	4	Identification of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
Message ID	5	Identification of the message - the id defines what the payload "means" and how it should be correctly decoded.
Payload	6 to (n+6)	The data into the message, depends on the message id.
CRC	(n+7) to (n+8)	Check-sum of the entire packet, excluding the packet start sign (LSB to MSB)

MAVLink or Micro Air Vehicle Link is a protocol for communicating with small unmanned vehicle. It is designed as a header-only message marshaling library



We need to send MAVlink commands in a message.

vehicle.send_mavlink(msg)

How to Velocity Vectors work?

```
# Sends velocity vector message to UAV vehicle
def send_ned_velocity(self, velocity_x, velocity_y, velocity_z, duration, vehicle):
    """
    Move vehicle in a direction based on specified velocity vectors.
    """
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,  # time_boot_ms (not used)
        0, 0,  # target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED,  # frame
        0b0000111111000111,  # type_mask (only speeds enabled)
        0, 0, 0,  # x, y, z positions (not used)
        velocity_x, velocity_y, velocity_z,  # x, y, z velocity in m/s
        0, 0, 0,  # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
        0, 0)  # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)

    for x in range(0, duration):
        vehicle.send_mavlink(msg)

    time.sleep(0.1)

    # Sends velocity vector message to UAV vehicle
```

Where does this come from?

Things to configure:

- duration
- sleep
- magnitude

How do we compute the NED vector?

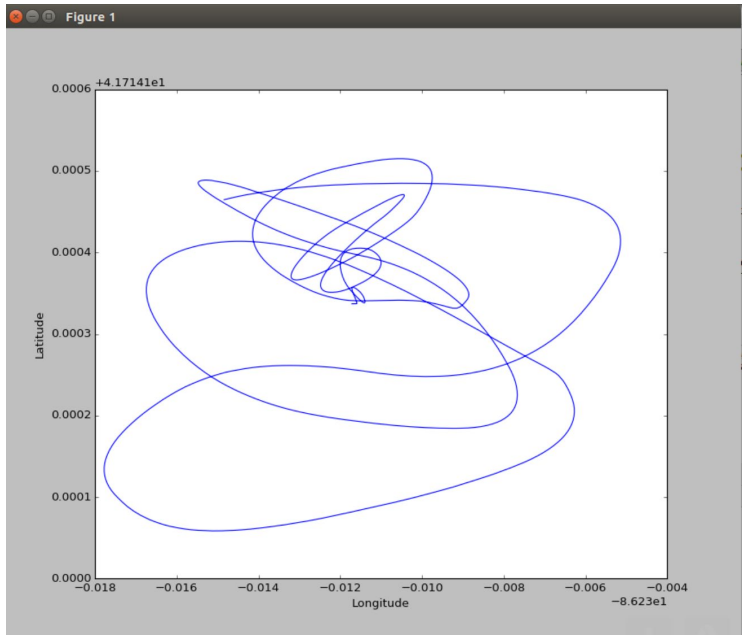
```
# Sets NED given a current and target location
def setNed(self, current, target):
    lat_C, lon_C = rad(current.lat), rad(current.lon)
    lat_T, lon_T = rad(target.lat), rad(target.lon)
    # create an n-vector for current and target
    nvecC = nv.lat_lon2n_E(lat_C, lon_C)
    nvecT = nv.lat_lon2n_E(lat_T, lon_T)
    # create a p-vec from C to T in the Earth's frame
    # the zeros are for the depth (depth = -1 * altitude)
    p_CT_E = nv.n_EA_E_and_n_EB_E2p_AB_E(nvecC, nvecT, 0, 0)
    # create a rotation matrix
    # this rotates points from the NED frame to the Earth's frame
    R_EN = nv.n_E2R_EN(nvecC)
    # rotate p_CT_E so it lines up with current's NED frame
    # we use the transpose so we can go from the Earth's frame to the NED frame
    n, e, d = np.dot(R_EN.T, p_CT_E).ravel()

    #Scale Nedvalues
    if n > e:
        scaleFactor = abs(1.00000/n)
    else:
        scaleFactor = abs(1.00000/e)
    return Nedvalues(n*scaleFactor, e*scaleFactor, d)
```

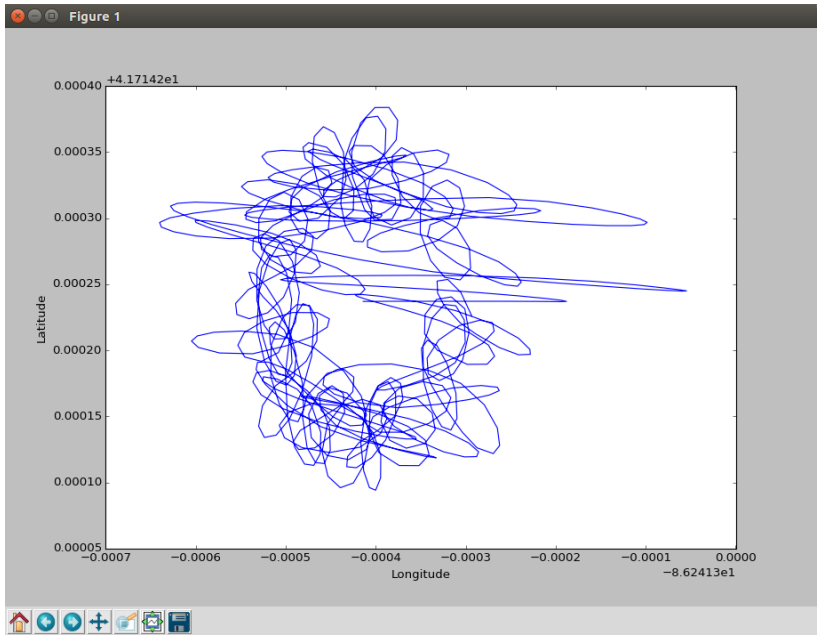
Computes the NED that will take you all the way to the final destination.

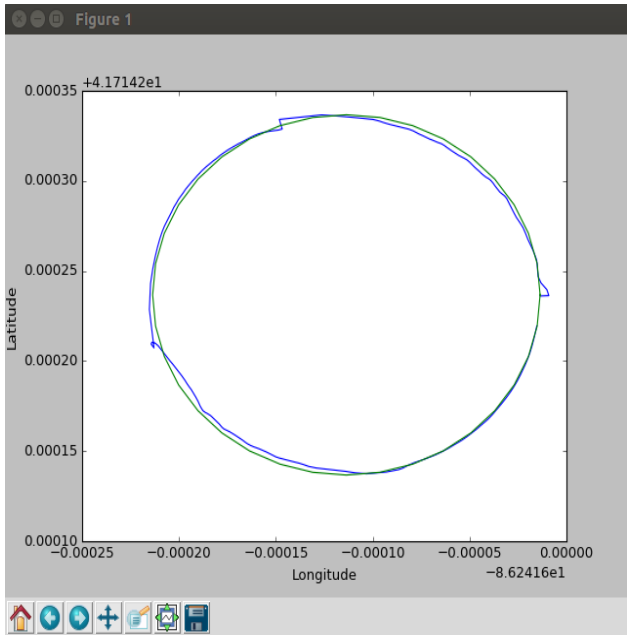
If we want our UAV to be responsive to commands before reaching its destination, we need to scale the NED and send multiple smaller directives.

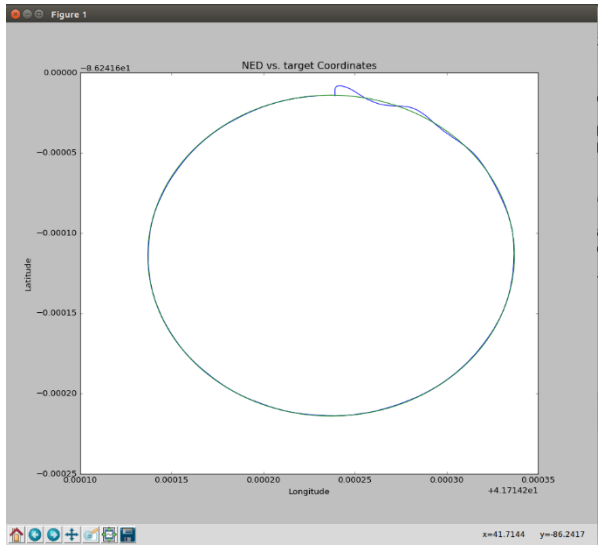
Which can fly the best circle? NED or Waypoints?

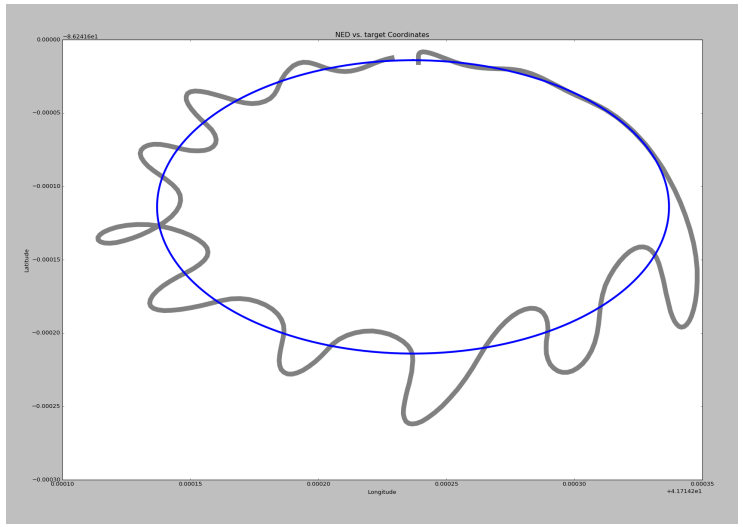
A “Circle” with NED Vectors





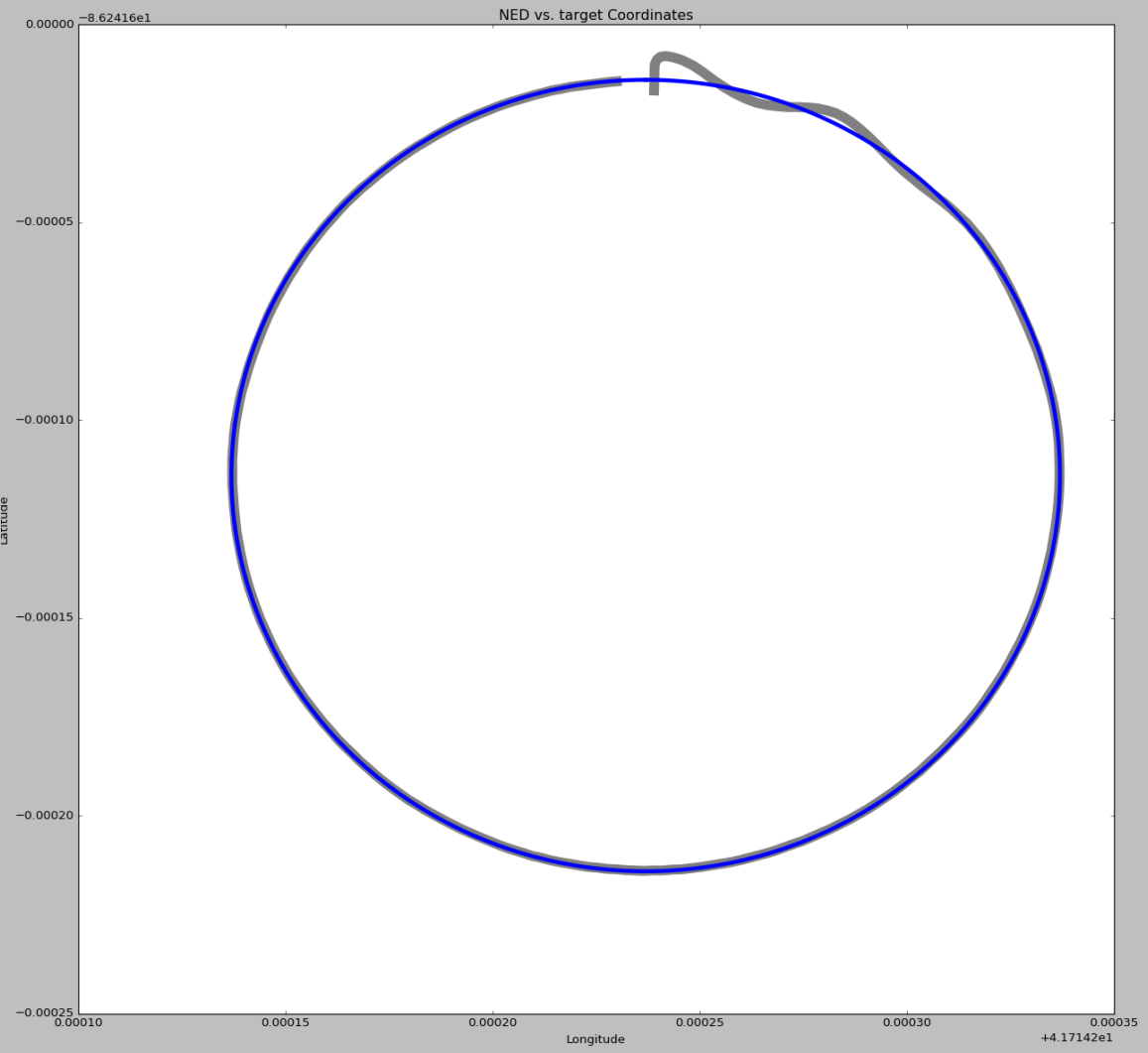







```
while distanceRemaining > 1
    ……
    nedcontroller.send_ned_velocity
    (ned.north, ned.east, ned.down, 1,
    vehicle)
    ……
```

A “circle” with NED Vectors

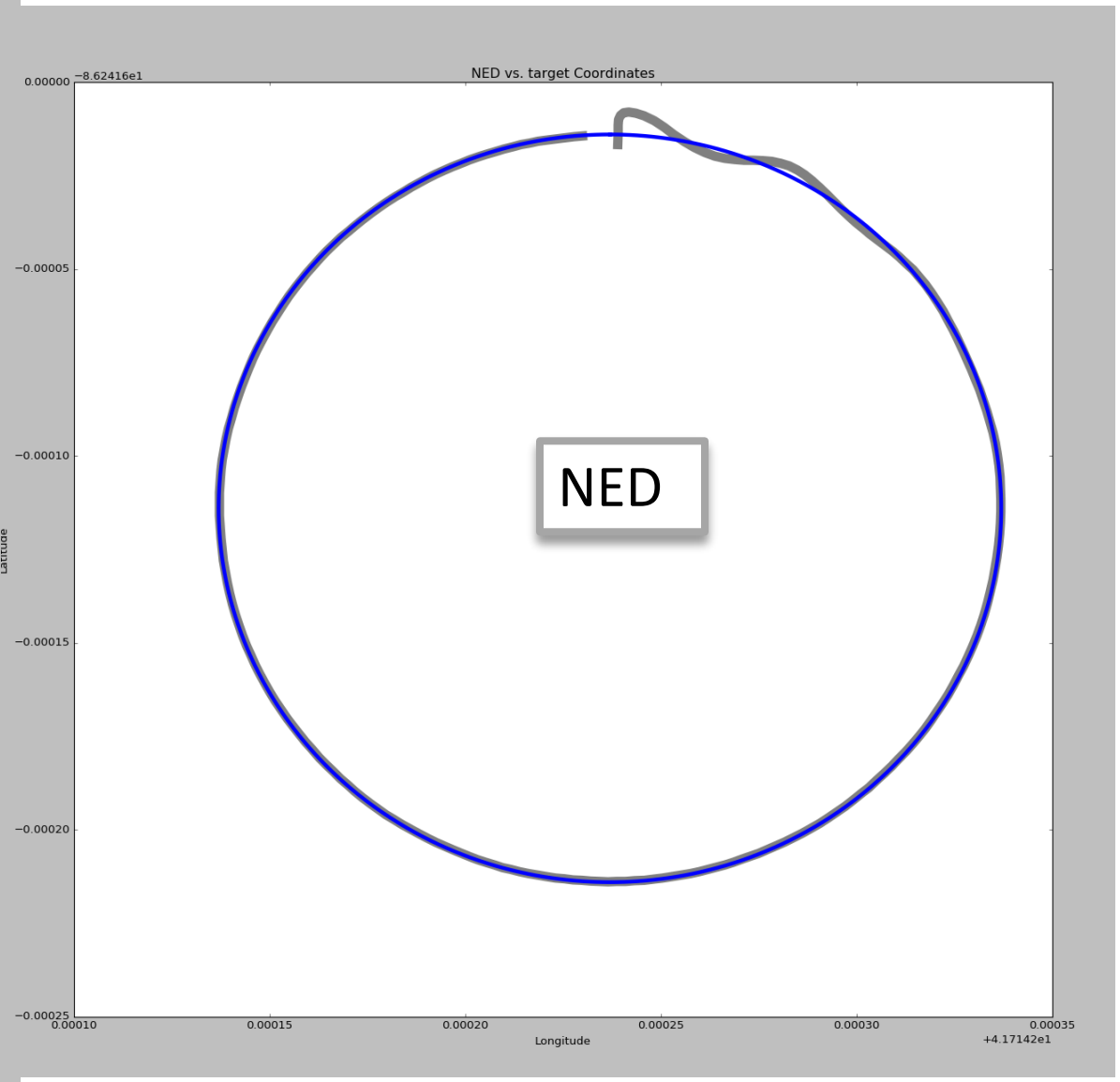


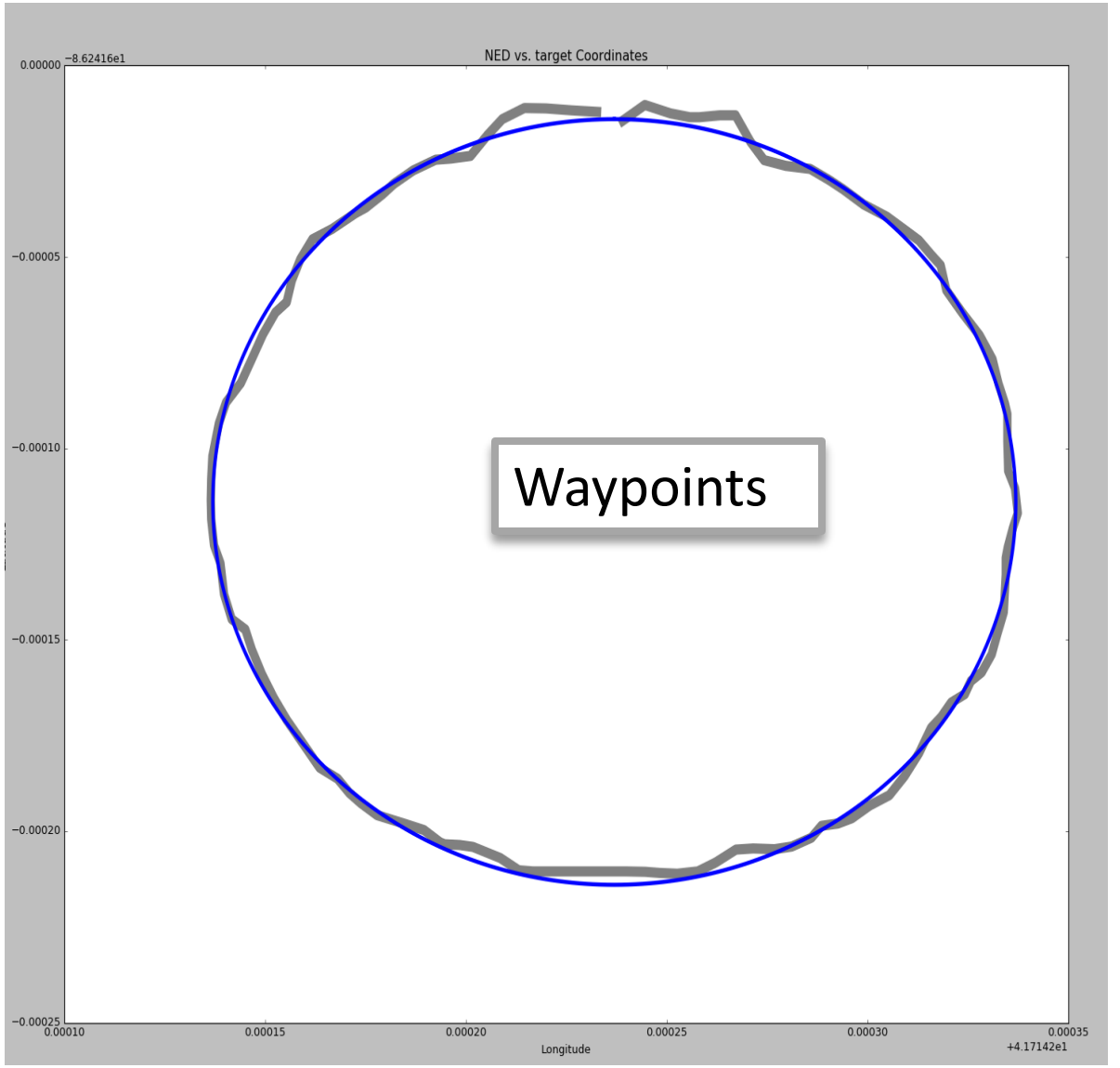

```
# Sets NED given a current and target location
def setNed(self, current, target):
    lat_C, lon_C = rad(current.lat), rad(current.lon)
    lat_T, lon_T = rad(target.lat), rad(target.lon)
    # create an n-vector for current and target
    nvecC = nv.lat_lon2n_E(lat_C, lon_C)
    nvecT = nv.lat_lon2n_E(lat_T, lon_T)
    # create a p-vec from C to T in the Earth's frame
    # the zeros are for the depth (depth = -1 * altitude)
    p_CT_E = nv.n_EA_E_and_n_EB_E2p_AB_E(nvecC, nvecT, 0, 0)
    # create a rotation matrix
    # this rotates points from the NED frame to the Earth's f
    R_EN = nv.n_E2R_EN(nvecC)
    # rotate p_CT_E so it lines up with current's NED frame
    # we use the transpose so we can go from the Earth's fram
    n, e, d = np.dot(R_EN.T, p_CT_E).ravel()

    #Scale Nedvalues
    #if n > e:
    #    scaleFactor = abs(1.00000/n)
    #else:
    #    scaleFactor = abs(1.00000/e)

    #return Nedvalues(n*scaleFactor, e*scal
    return Nedvalues(n,e,d)
```

The circle didn’t like the scaling factor.

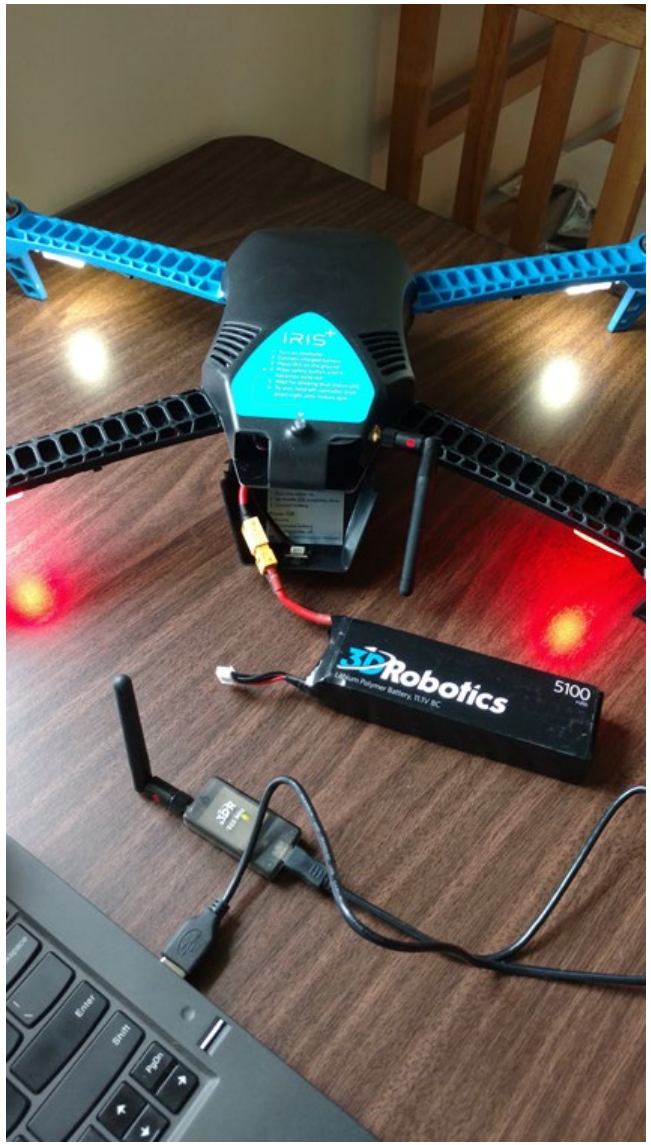
Why?

NED Vectors vs. Waypoints





Hello Physical Drone!



Connect IRIS using radio

```
jane@ubuntu: /dev
jane@ubuntu:~$ cd /dev
jane@ubuntu:/dev$ ls tty*
tty     tty17   tty26   tty35   tty44   tty53   tty62     ttyS12   ttyS21   ttyS30
tty0    tty18   tty27   tty36   tty45   tty54   tty63     ttyS13   ttyS22   ttyS31
tty1    tty19   tty28   tty37   tty46   tty55   tty7      ttyS14   ttyS23   ttyS4
tty10   tty2    tty29   tty38   tty47   tty56   tty8      ttyS15   ttyS24   ttyS5
tty11   tty20   tty3    tty39   tty48   tty57   tty9      ttyS16   ttyS25   ttyS6
tty12   tty21   tty30   tty4    tty49   tty58   ttyprintk ttyS17   ttyS26   ttyS7
tty13   tty22   tty31   tty40   tty5    tty59   ttyS0     ttyS18   ttyS27   ttyS8
tty14   tty23   tty32   tty41   tty50   tty6    ttyS1     ttyS19   ttyS28   ttyS9
tty15   tty24   tty33   tty42   tty51   tty60   ttyS10    ttyS2    ttyS29   ttyUSB0
tty16   tty25   tty34   tty43   tty52   tty61   ttyS11    ttyS20   ttyS3
jane@ubuntu:/dev$
```

Open another new terminal and type:

```
python Hello.py --connect /dev/ttyUSB0,57600
```

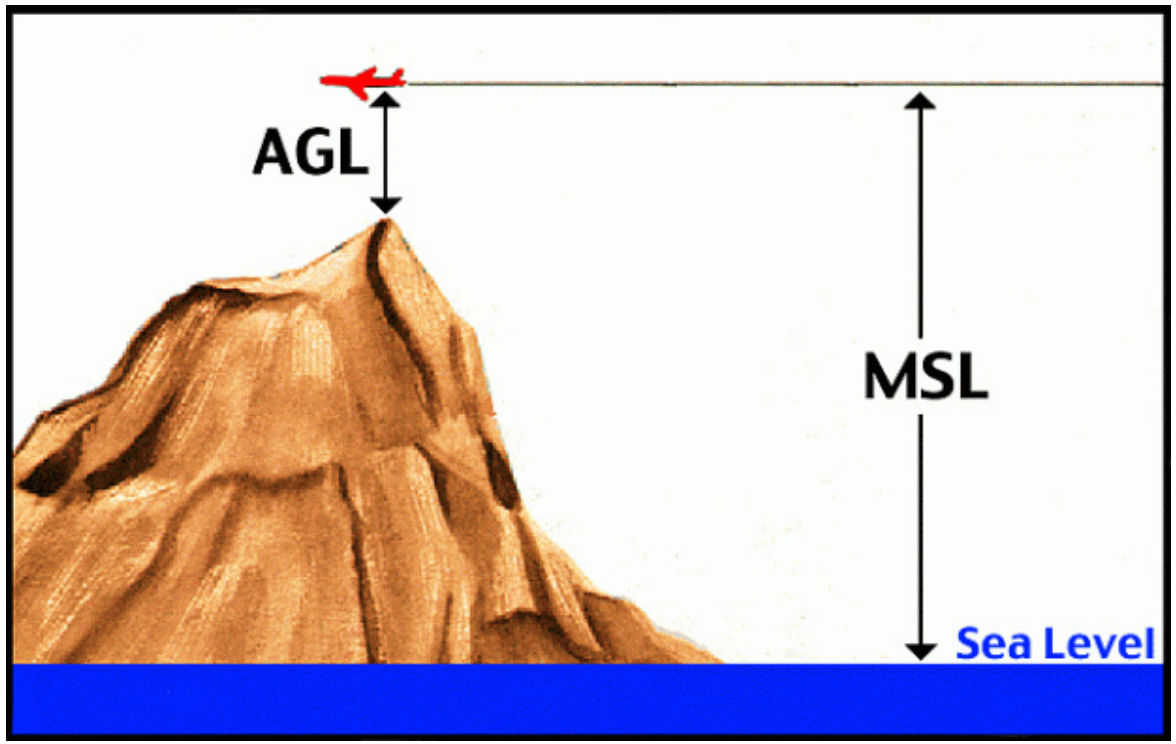
Or connect cable directly from USB on laptop into mini USB on side of IRIS. Check for the port (probably ttyACM0)

In the new terminal window type:

```
python Hello.py --connect /dev/ttyACM0,57600
```

Specify the baud rate that IRIS 3DR expects: **57600** and add to connection string.

MSL vs. AGL

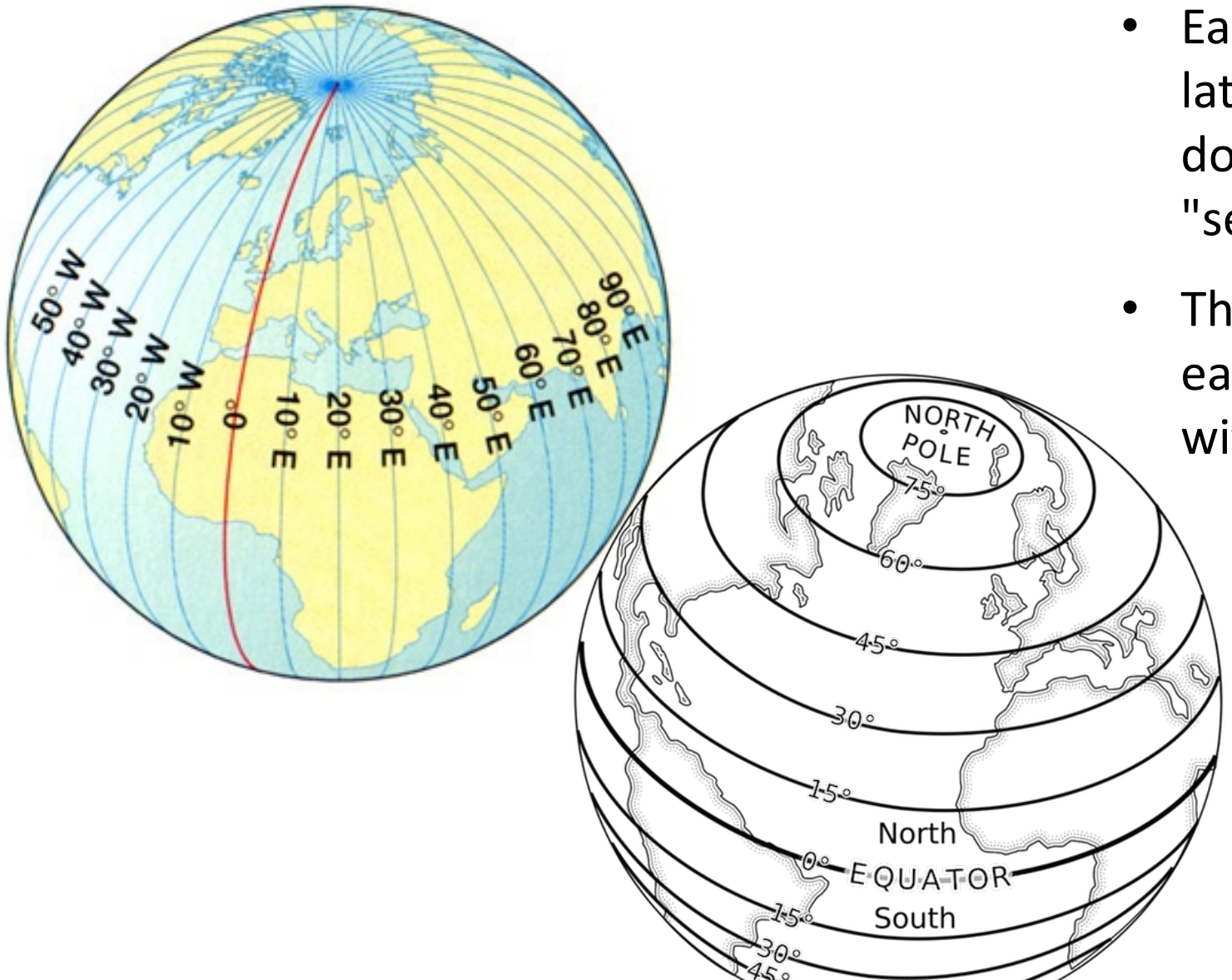


Mean Sea Level
Above Ground Level

Normally small UAVs are permitted to fly at 400 AGL; however White Field is under the flight path of South Bend airport, so we are permitted to fly only up to 100 AGL.

Furthermore, prior to flight we must notify the South Bend Air Traffic Control tower.

Longitude and Latitude



- Each degree of longitude and latitude can be further broken down into "minutes" and "seconds".
- There are 60 minutes within each degree and 60 seconds within each minute.
- The latitude coordinates are listed first, followed by the longitude coordinates.

Class A Airspace

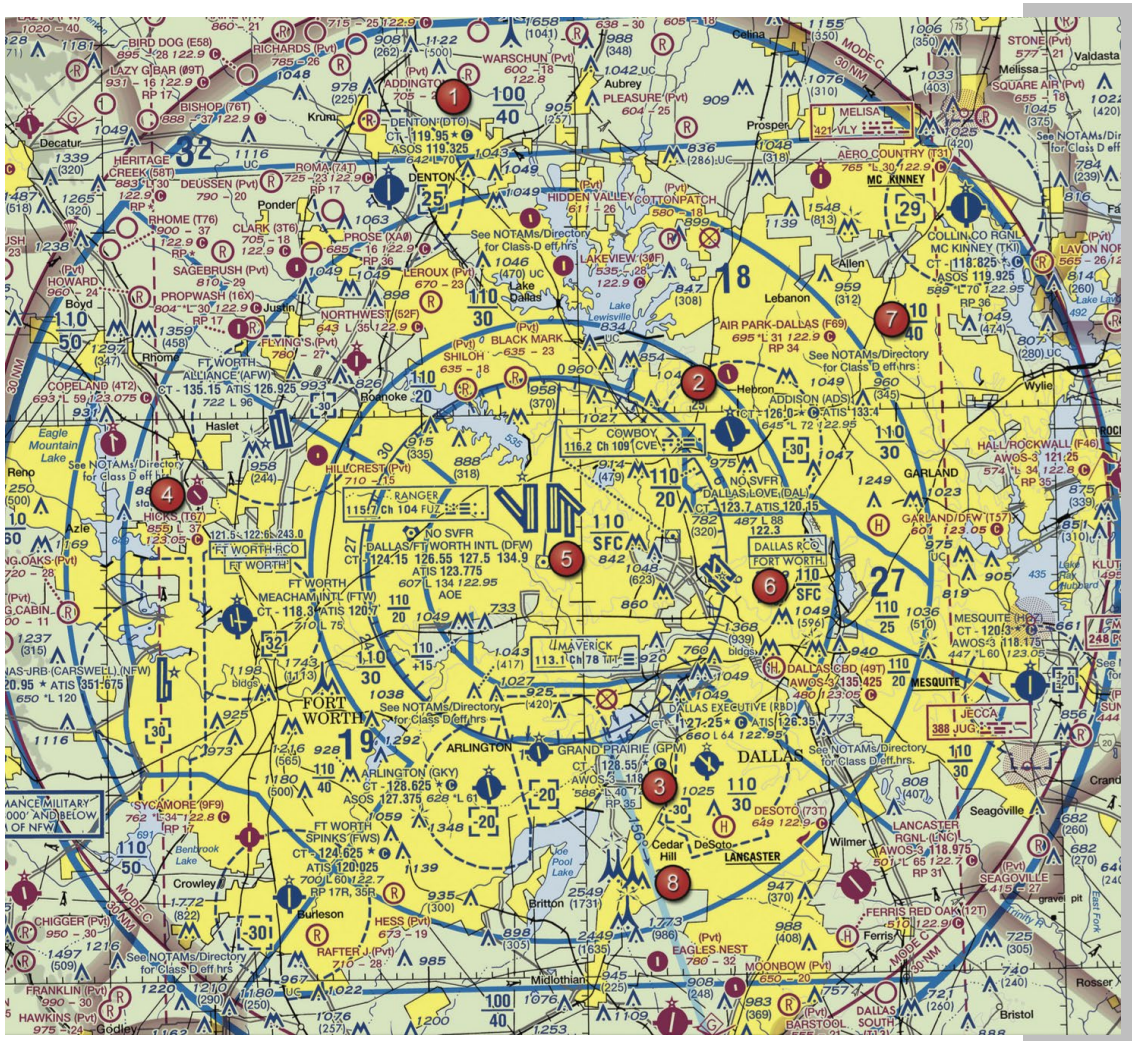
Class A Airspace is generally the airspace from **18,000 feet MSL** up to and including **60,000 feet MSL** (also referred to as "Flight Level" – or FL – 600).

All flights in this airspace must be conducted under instrument flight rules (IFR).

This area also extends out 12 NM from the coast of the contiguous 48 states and Alaska.

Class B Airspace (Big Airports)

- Layers get wider and wider as you get up in altitude.
- Customized for each airport to account for the flow of aircraft landing and taking off at the airport.
- Aircraft MUST receive prior ATC authorization.
- **Typical ceiling for Class B airspace is 10,000 feet MSL**
- Depicted on sectional charts by a solid blue line.



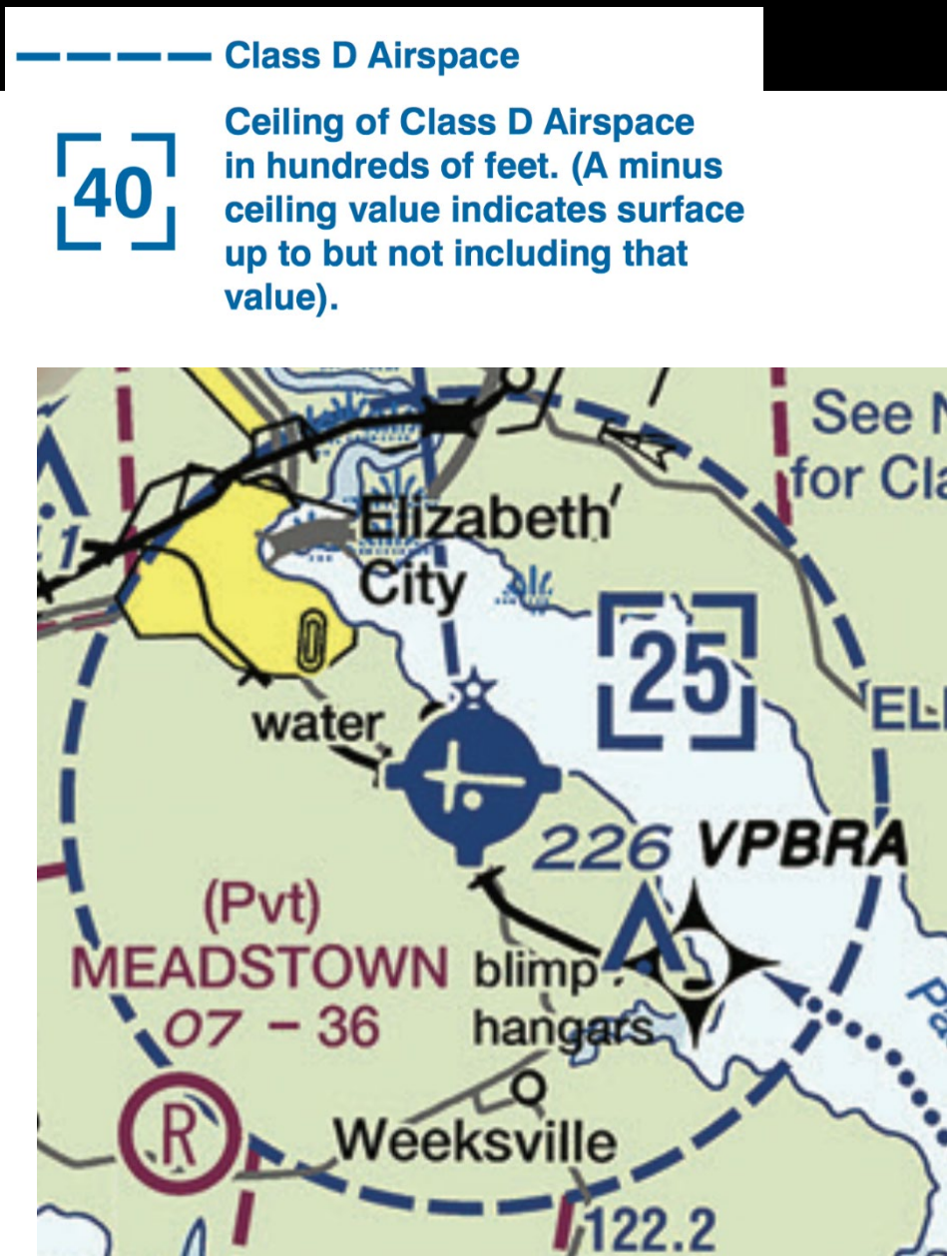
Class C Airspace

- Next sized airport
- Usually consists of two rings: a surface area and a shelf area.
- The center ring (surface area) usually extends from the surface to 4,000 feet AGL with a diameter of about 5 NM. Altitude is listed at MSL. i.e.. If airport elevation is 683, then ceiling is likely 4,700 MSL.
- The outer ring (or shelf area) usually starts at 1,200 feet AGL and goes up to 4,000 AGL. Usually has diameter of 10 NM.
- UAS remote pilots are required to obtain a clearance

Depicted on sectional charts by a solid magenta line.

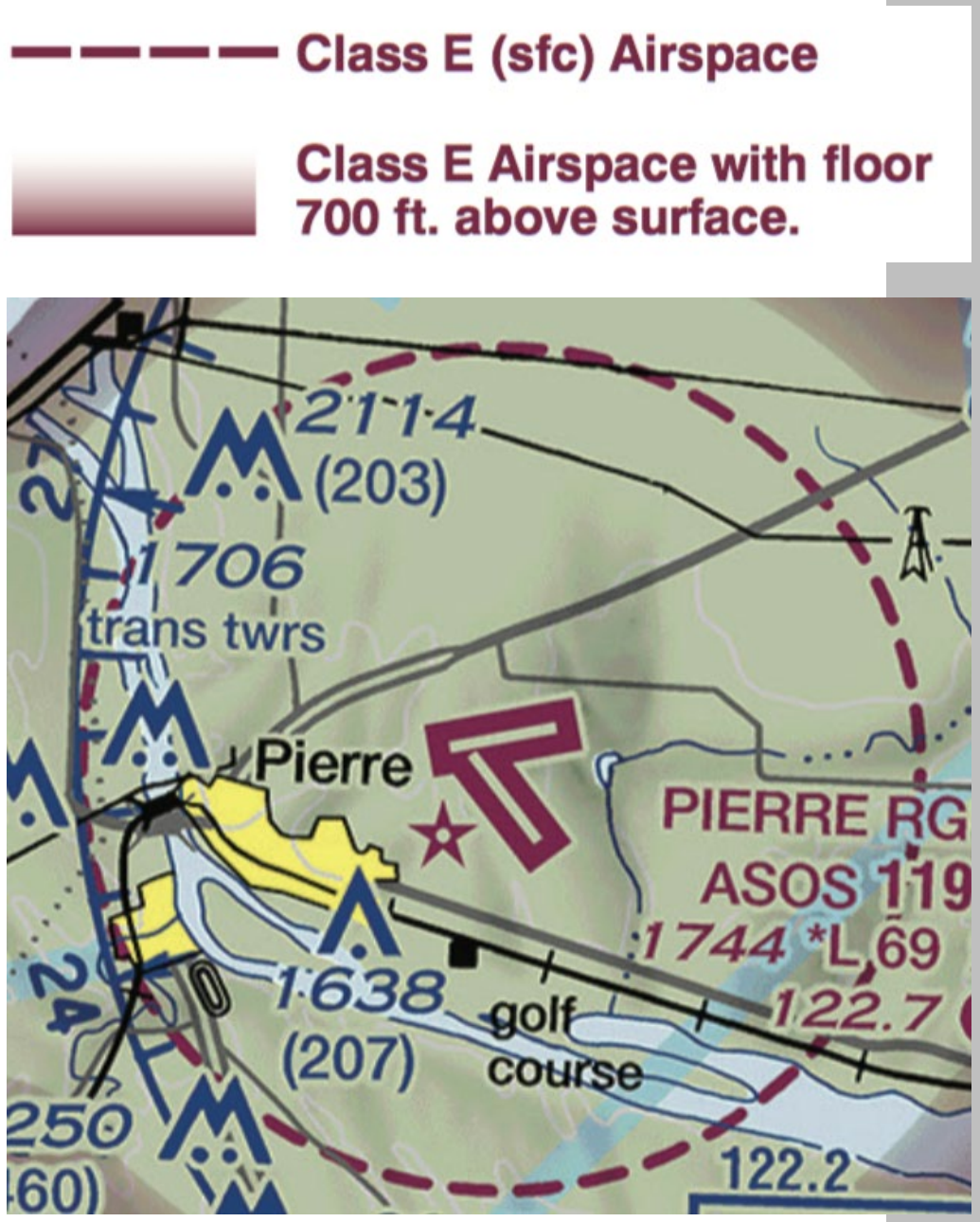
Class D Airspace

- Smaller than Class B or C but with a control tower.
- Usually one ring extends up to 2,500 feet AGL.
- Prior authorization is required before flying in class D airspace. Altitude listed on sectional chart at mean sea level (MSL). For instance, if the airport elevation is 1,200 feet MSL, the ceiling of the Class D airspace would likely be 3,700 MSL
- Class D airspace is indicated on sectional charts by a blue dashed line.
- The ceiling of Class D airspace is indicated by numbers within a blue dashed box.



Class E Airspace

- ANY controlled airspace that is NOT Class A, B, C, or D.
- Frequently adjacent to other controlled airspace to allow for instrument procedures (IFR flights) in and out of airports. Typically starts at 700 feet AGL or at the surface.
- In all other areas of the country not marked with Class B, C, or D, Class E airspace typically starts at 1,200 ft AGL and extends up to but not including 18,000 feet MSL.
- Dashed magenta line indicates floor is the surface.
- Shaded magenta line indicates floor is 700 feet above ground level



Quiz Questions:

Class A airspace begins at what altitude?

A. 18,000 feet MSL

B. 10,000 feet MSL

C. 18,000 feet AGL

Quiz Questions:

What is the typical ceiling of Class B airspace?

A. 10,000 feet above ground level (AGL)

B. 10,000 feet mean sea level (MSL)

C. 12,000 feet mean sea level (MSL)

Quiz Questions:

Refer to Figure 25 from Supplemental materials:
What Class of Airspace is overlying the Dallas Fort Worth Airport?

A. Class C

B. Class B

C. Class D

Quiz Questions:

How are the altitude boundaries of class B airspace displayed on sectional charts?

A. Milibars

B. Feet Above Ground Level

C. Feet in Mean Sea Level

Quiz Questions:

How are the altitude boundaries of class C airspace displayed on sectional charts?

A. Milibars

B. Feet Above Ground Level

C. Feet in Mean Sea Level

Quiz Questions:

What is the typical ceiling of Class C airspace?

A. 4,000 feet above ground level (AGL)

B. 4,000 feet mean sea level (MSL)

C. 4,700 feet above ground level (AGL)

Quiz Questions:

According to 14 CFR part 107 the remote pilot in command of a small unmanned aircraft planning to operate within Class C airspace…

A. Is required to file a flight plan.

B. Must use a visual observer.

C. is required to receive ATC authorization.

Quiz Questions:

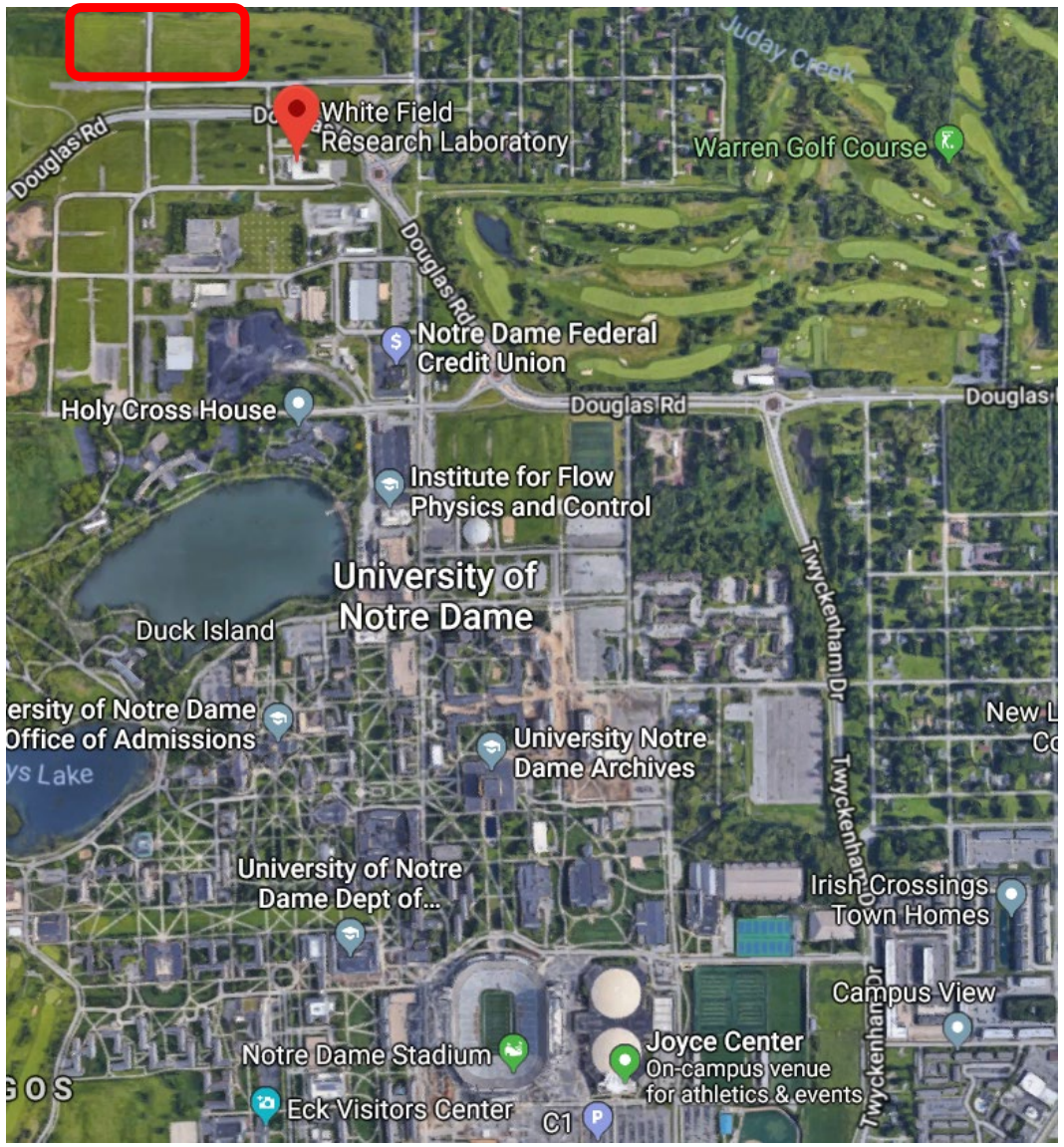
Refer to FAA-CT-8080-2H, Figure 25
What class of airspace is overlying the Addison airport?

A. Class D, from the surface up to 3,000 ft

B. Class E, starting at 700 ft AGL

C. Class E, starting at the surface

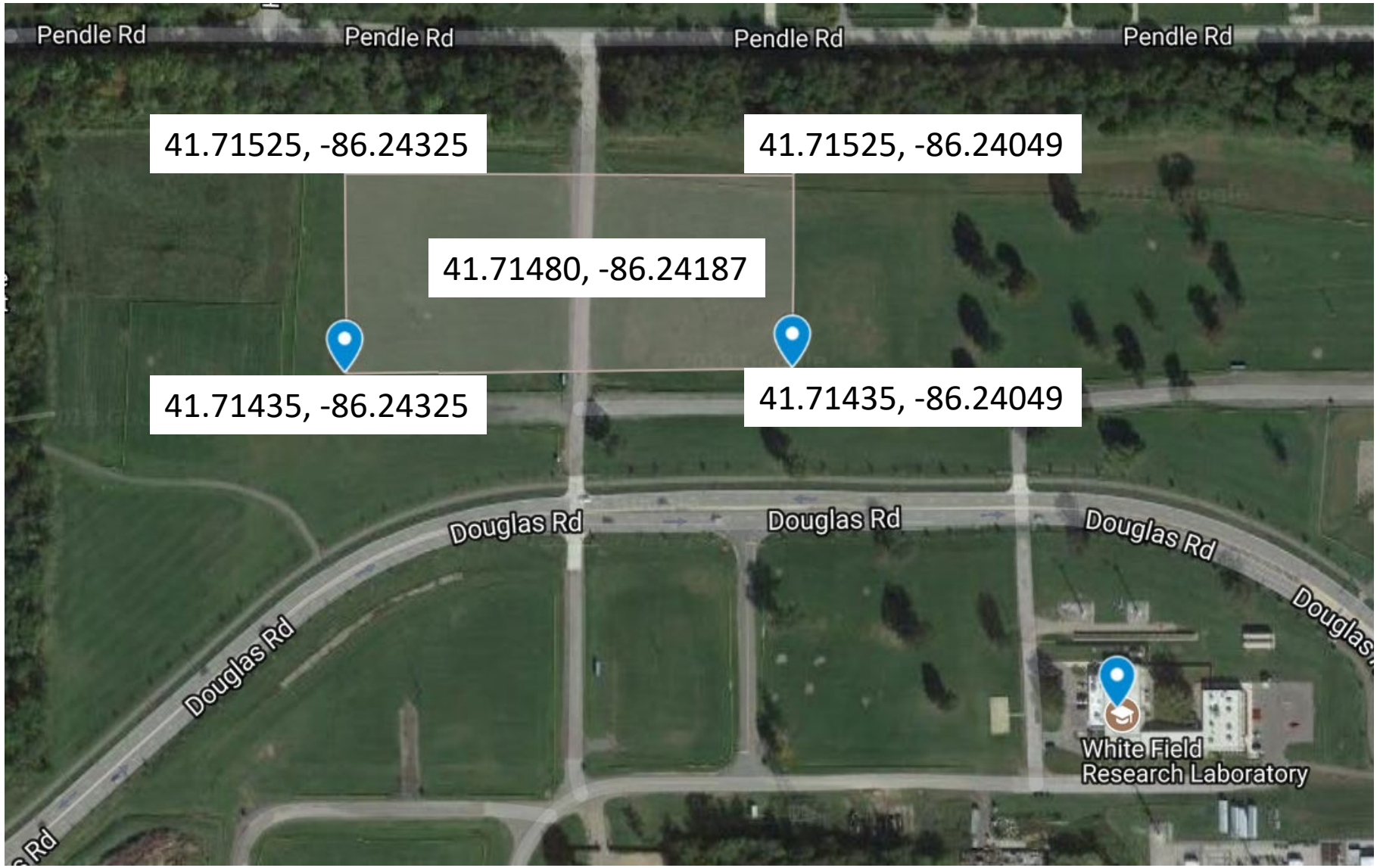
Where will we be flying?



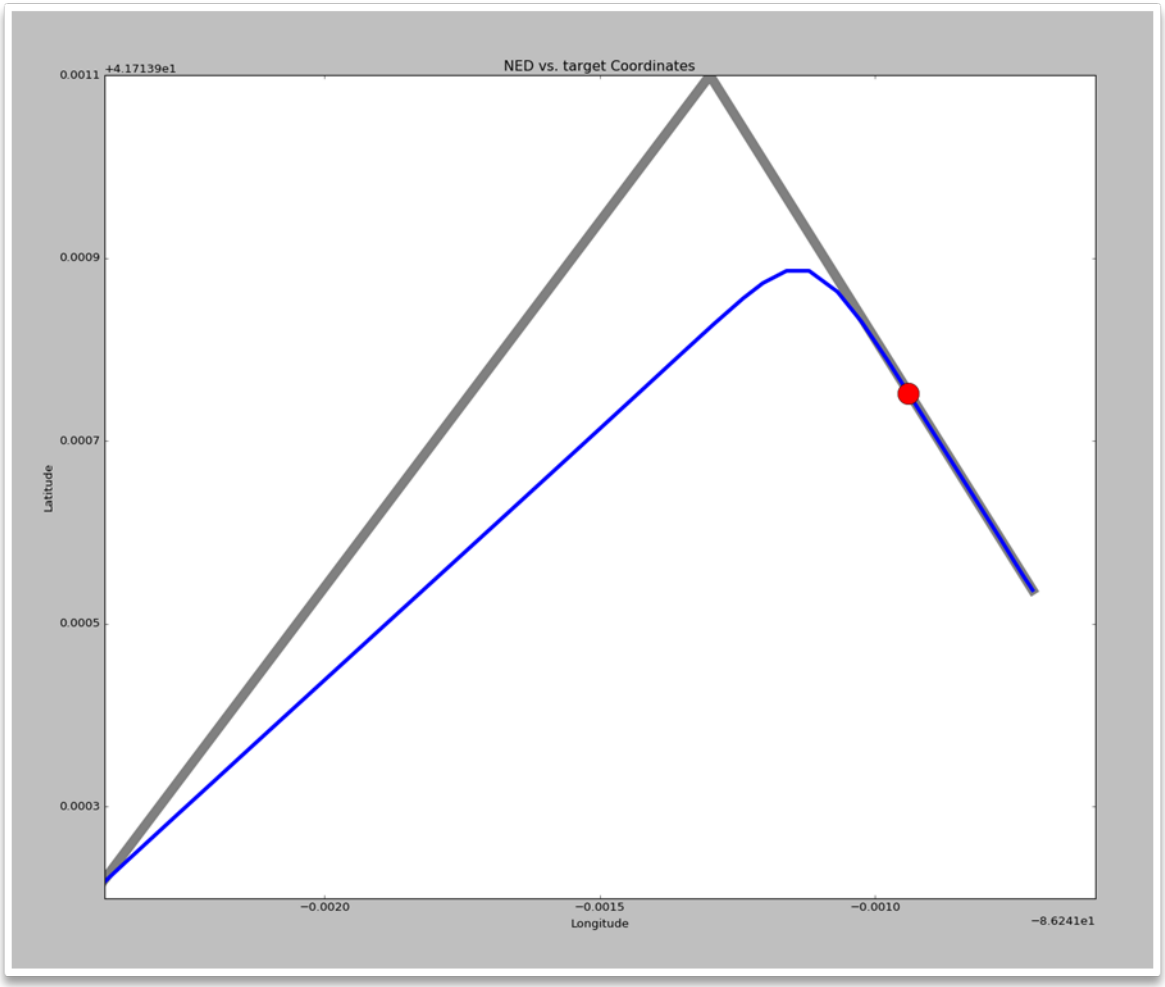
Perform your SITL homework @White Field with SITL. Later we will actually test it there.

Drive, bike, or walk!

Where will we be flying?



If time:



- Work in groups of 2-3.
- Plan an experiment using waypoints.
- Fly the drone at different speeds and midway abort the current waypoint and give it a new target location. (Try different angles).
- What did you observe as you changed speed and ‘angle’?