Software Development for Unmanned Aerial Systems

Instructor:

Jane Cleland-Huang, PhD

JaneClelandHuang@nd.edu

Department of Computer Science and Engineering

University of Notre Dame

What is the course about?

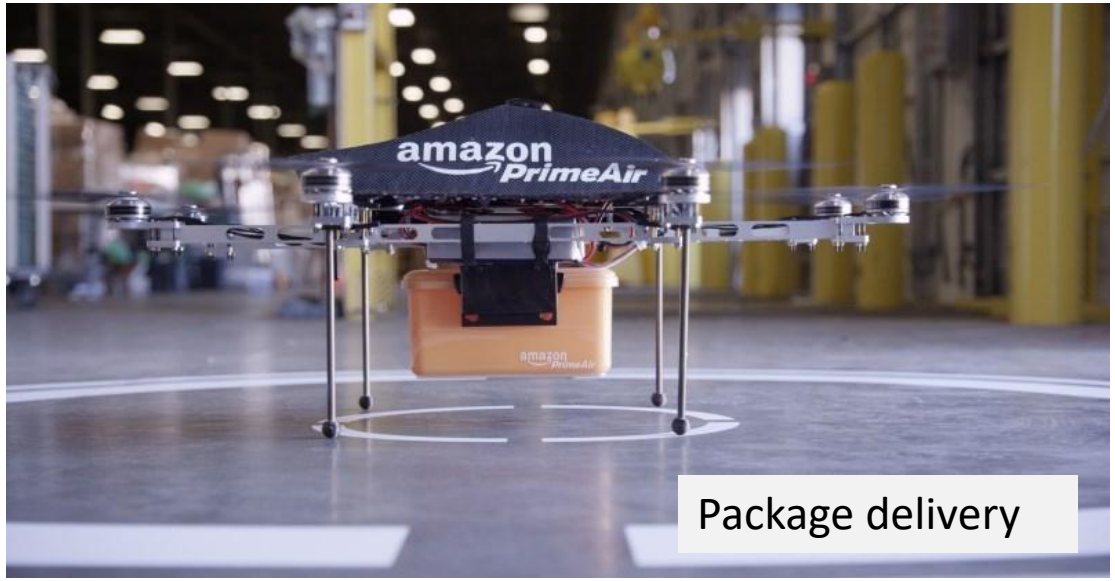
This is a hands-on development course in which we address **several challenges** related to UAV programming and will **engage in a Systems Engineering project** that involves building a project using IRIS 3DR Drones and (maybe) other UAVs.

Drones are becoming ubiquitous

UK Royal Mail

Police Departments


Hurricane tracking

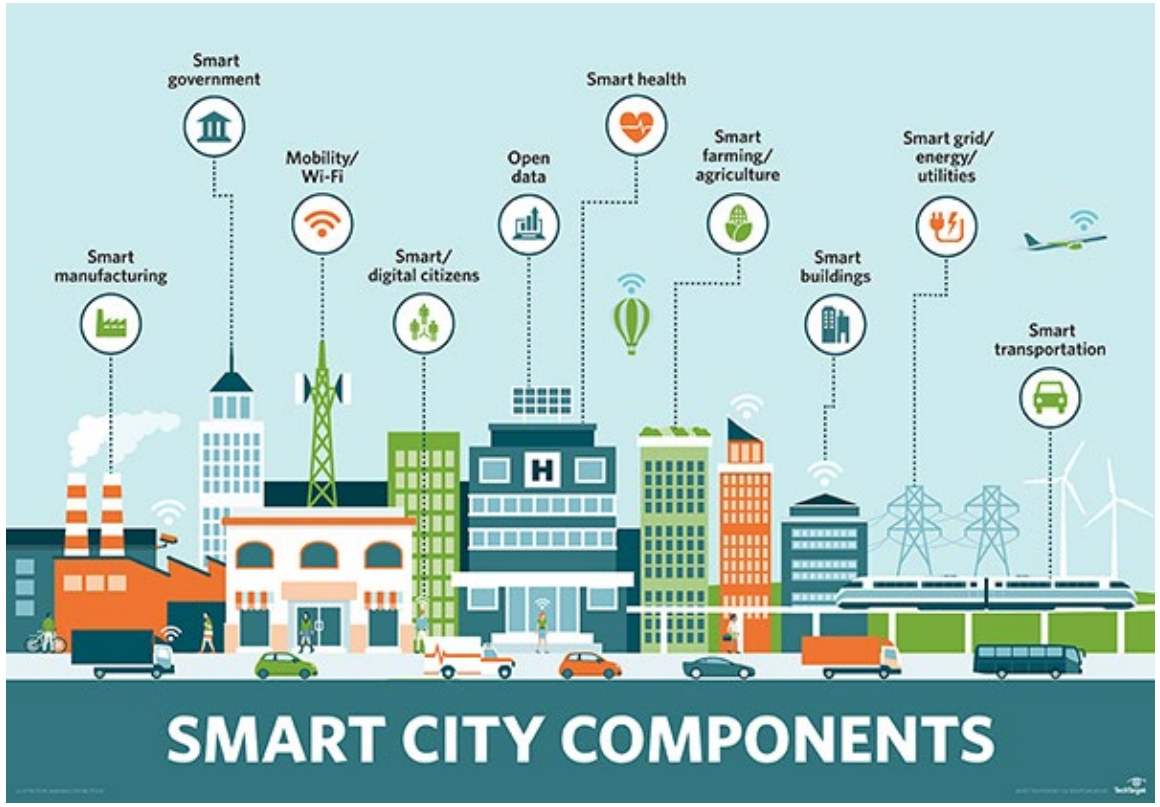


Package delivery


Wildlife management

Drones are Cyber Physical Systems

Cyber-Physical systems (CPS) incorporate, and seamlessly integrate, physical components and computational algorithms within a single system.



They are deployed in areas such as autonomous driving, smart cities, medical devices, and unmanned aerial systems.

Drones are Safety Critical

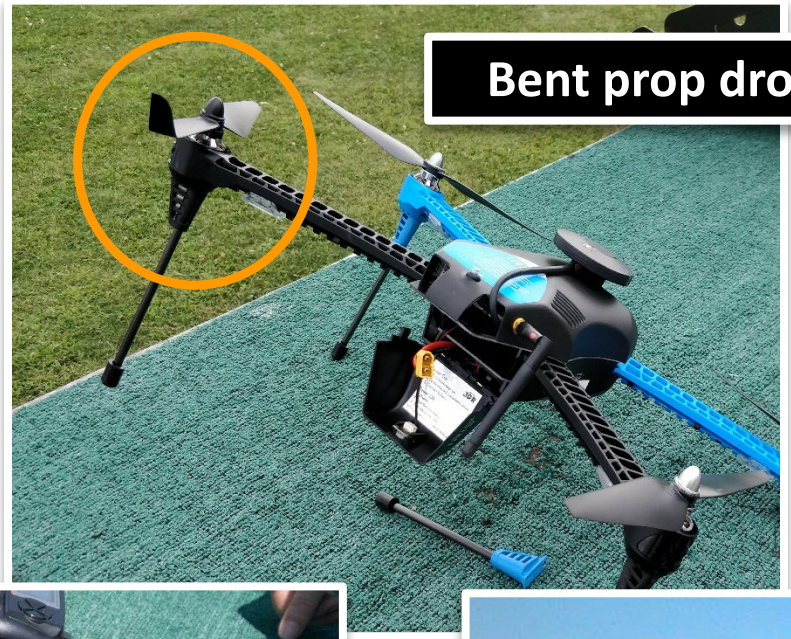
Any software written to control and fly a drone is considered Safety Critical because its failure could cause physical harm or even death.

All of our applications must therefore:

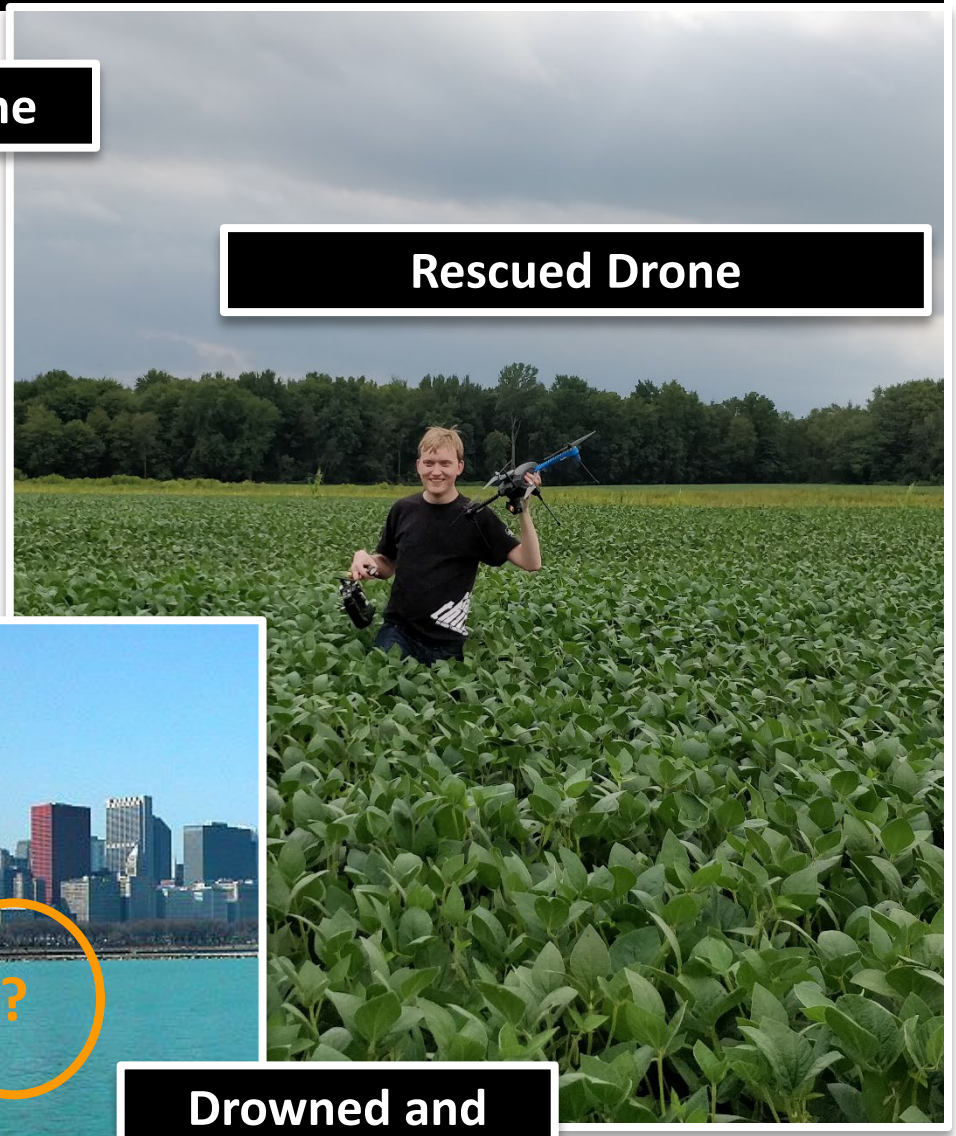
- Comply with federal FAA regulations
- Prevent hazards from occurring such as:
    1. Landing or descending in an unsafe area such as on top of a person's head, in traffic etc.
    2. Crashing into other drones during flight and causing debris to rain down.
    3. Flying into the flight path of an airplane.
    4. Flying in any other way which might cause an accident.

Drones sometimes crash

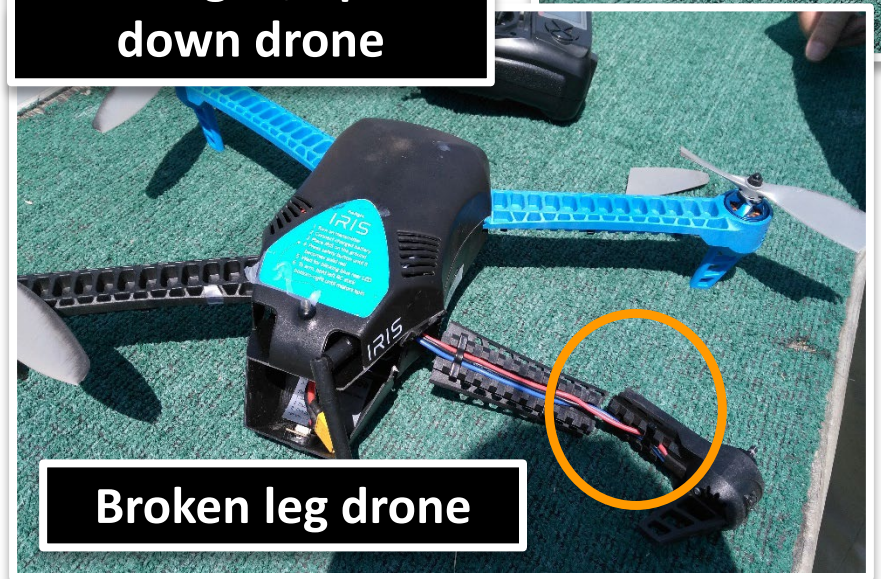
Trajectory challenged, upside down drone



Bent prop drone



Rescued Drone



Broken leg drone

Drowned and missing drone

They just keep coming…

1. UAVs are not 100% reliable because:
   - We are constantly adding new hardware configurations.
   - We are constantly adding new software and a bug can lead to a crash.
2. Whenever possible **all code must be tested in the simulator first.**
3. **Flying tests must always be in a safe place**.

Always test code in the simulator first and NEVER fly knowingly buggy code.

Be Responsible

Horror as remote-control helicopter stunt pilot, 19, partially-decapitates himself with his aircraft after he lost control

- Roman Pirozek Jr, 19, was an avid RC helicopter hobbyist
- He specialized in performing dare-devil tricks with his powerful $1,500 model helicopter
- His own father watched as the blades of the 6-pound RC flyer sliced off a piece of his head
- He was dead when police arrived

By MICHAEL ZENNIE

PUBLISHED: 16:09 EST, 5 September 2013 | **UPDATED:** 01:16 EST, 6 September 2013

FLY RESPONSIBLY

KNOW BEFORE YOU FLY

Be Safe

FAA News

Federal Aviation Administration, Washington, DC 20591

June 21, 2016

SUMMARY OF SMALL UNMANNED AIRCRAFT RULE (PART 107)

Operational Limitations	• Unmanned aircraft must weigh less than 55 lbs. (25 kg). • Visual line-of-sight (VLOS) only; the unmanned aircraft must remain within VLOS of the remote pilot in command and the person manipulating the flight controls of the small UAS. Alternatively, the unmanned aircraft must remain within VLOS of the visual observer. • At all times the small unmanned aircraft must remain close enough to the remote pilot in command and the person manipulating the flight controls of the small UAS for those people to be capable of seeing the aircraft with vision unaided by any device other than corrective lenses. • Small unmanned aircraft may not operate over any persons not directly participating in the operation, not under a covered structure, and not inside a covered stationary vehicle. • Daylight-only operations, or civil twilight (30 minutes before official sunrise to 30 minutes after official sunset, local time) with appropriate anti-collision lighting. • Must yield right of way to other aircraft. • May use visual observer (VO) but not required. • First-person view camera cannot satisfy "see-and-avoid" requirement but can be used as long as requirement is satisfied in other ways. • Maximum groundspeed of 100 mph (87 knots). • Maximum altitude of 400 feet above ground level (AGL) or, if higher than 400 feet AGL, remain within 400 feet of a structure. • Minimum weather visibility of 3 miles from control station. • Operations in Class B, C, D and E airspace are allowed with the required ATC permission. • Operations in Class G airspace are allowed without ATC permission. • No person may act as a remote pilot in command or VO for more than one unmanned aircraft operation at one time. • No operations from a moving aircraft. • No operations from a moving vehicle unless the operation is over a sparsely populated area. • No careless or reckless operations. • No carriage of hazardous materials.
	• Requires preflight inspection by the remote pilot in command. • A person may not operate a small unmanned aircraft if he or she knows or has reason to know of any physical or mental condition that would interfere with the safe operation of a small UAS. • Foreign-registered small unmanned aircraft are allowed to operate under part 107 if they satisfy the requirements of part 375. • External load operations are allowed if the object being carried by the unmanned aircraft is securely attached and does not adversely affect the flight characteristics or controllability of the aircraft. • Transportation of property for compensation or hire allowed provided that- ○ The aircraft, including its attached systems, payload and cargo weigh less than 55 pounds total; ○ The flight is conducted within visual line of sight and not from a moving vehicle or aircraft; and ○ The flight occurs wholly within the bounds of a State and does not involve transport between (1) Hawaii and another place in Hawaii through airspace outside Hawaii; (2) the District of Columbia and another place in the District of Columbia; or (3) a territory or possession of the United States and another place in the same territory or possession. • Most of the restrictions discussed above are waivable if the applicant demonstrates that his or her operation can safely be conducted under the terms of a certificate of waiver.
Remote Pilot in Command Certification and Responsibilities	• Establishes a remote pilot in command position. • A person operating a small UAS must either hold a remote pilot airman certificate with a small UAS rating or be under the direct supervision of a person who does hold a remote pilot certificate (remote pilot in command). • To qualify for a remote pilot certificate, a person must: ○ Demonstrate aeronautical knowledge by either: ▪ Passing an initial aeronautical knowledge test at an FAA-approved knowledge testing center; or ▪ Hold a part 61 pilot certificate other than student pilot, complete a flight review within the previous 24 months, and complete a small UAS online training course provided by the FAA. ○ Be vetted by the Transportation Security Administration. ○ Be at least 16 years old. • Part 61 pilot certificate holders may obtain a temporary remote pilot certificate immediately upon submission of their application for a permanent certificate. Other applicants will obtain a temporary remote pilot certificate upon successful completion of TSA security vetting. The FAA anticipates that it will be able to issue a temporary remote pilot certificate within 10 business days after receiving a completed remote pilot certificate application. • Until international standards are developed, foreign-

1. Notre Dame campus is in the South Bend flight path. Outside flight is prohibited by the FAA. We will fly at the South Bend Radio Control Club http://www.southbendrc.org/rc/learn-to-fly/
2. Don't goof around.
3. Observe all FAA regulations for non-commercial drone flight. https://www.faa.gov/uas/

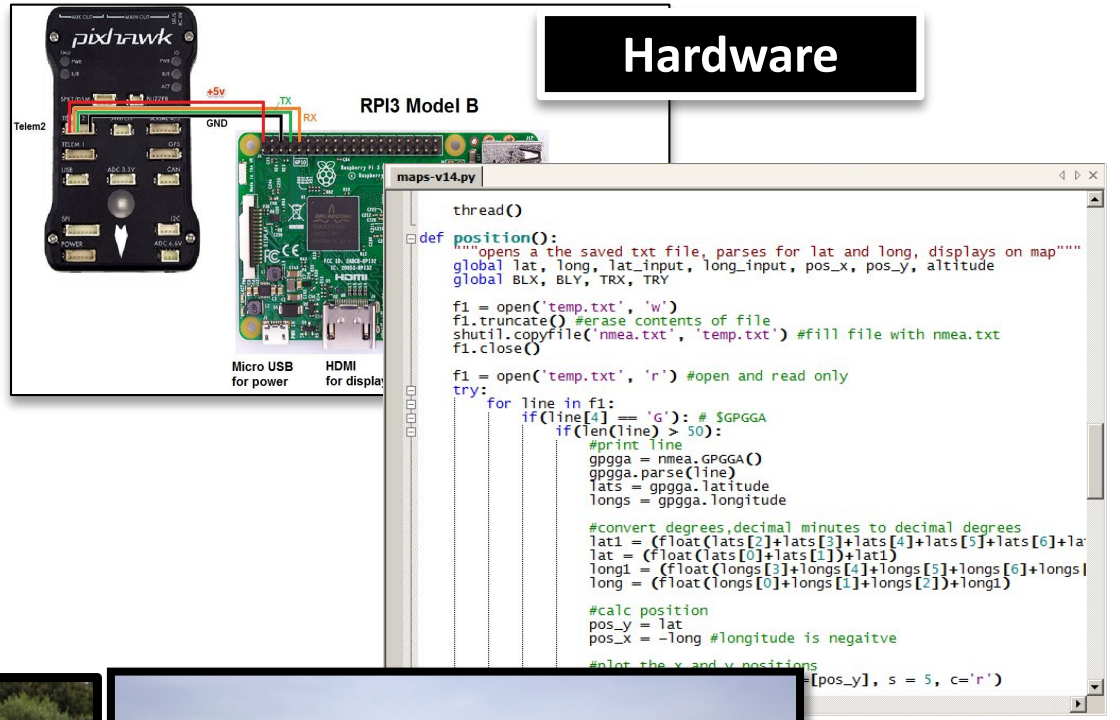
Take The Test

http://knowbeforeyoufly.org/resources/are-you-ready-to-fly-a-drone/

Rules

1. Only members of this class may operate UAVs.
2. Never, ever stand or walk underneath a UAV in flight (unless it is a specific part of your project – with safety protections established).
3. Never fly a physical UAV without testing the flight in the UAV simulator.
4. On campus ONLY fly on **white field**.
5. Always perform prechecks before flying.
6. Obey all FAA regulations (we will discuss these).

Aspects of this course



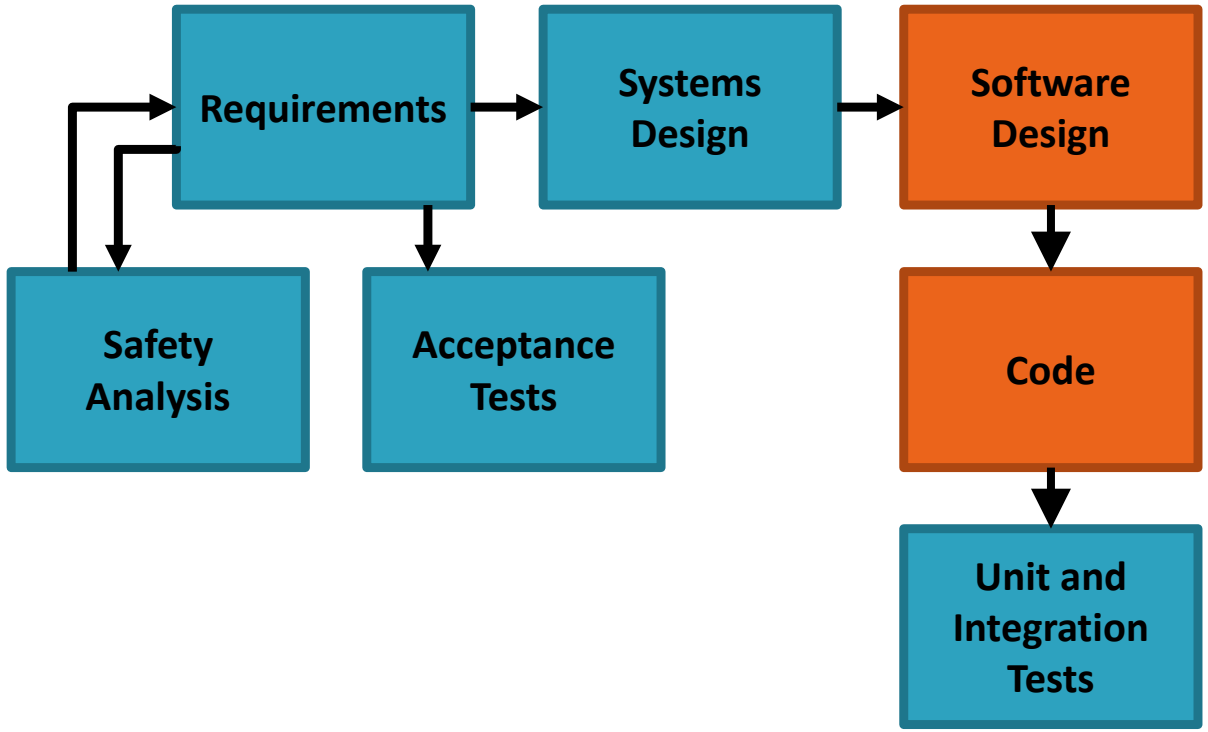
Team Projects following Software and Systems Engineering Practices

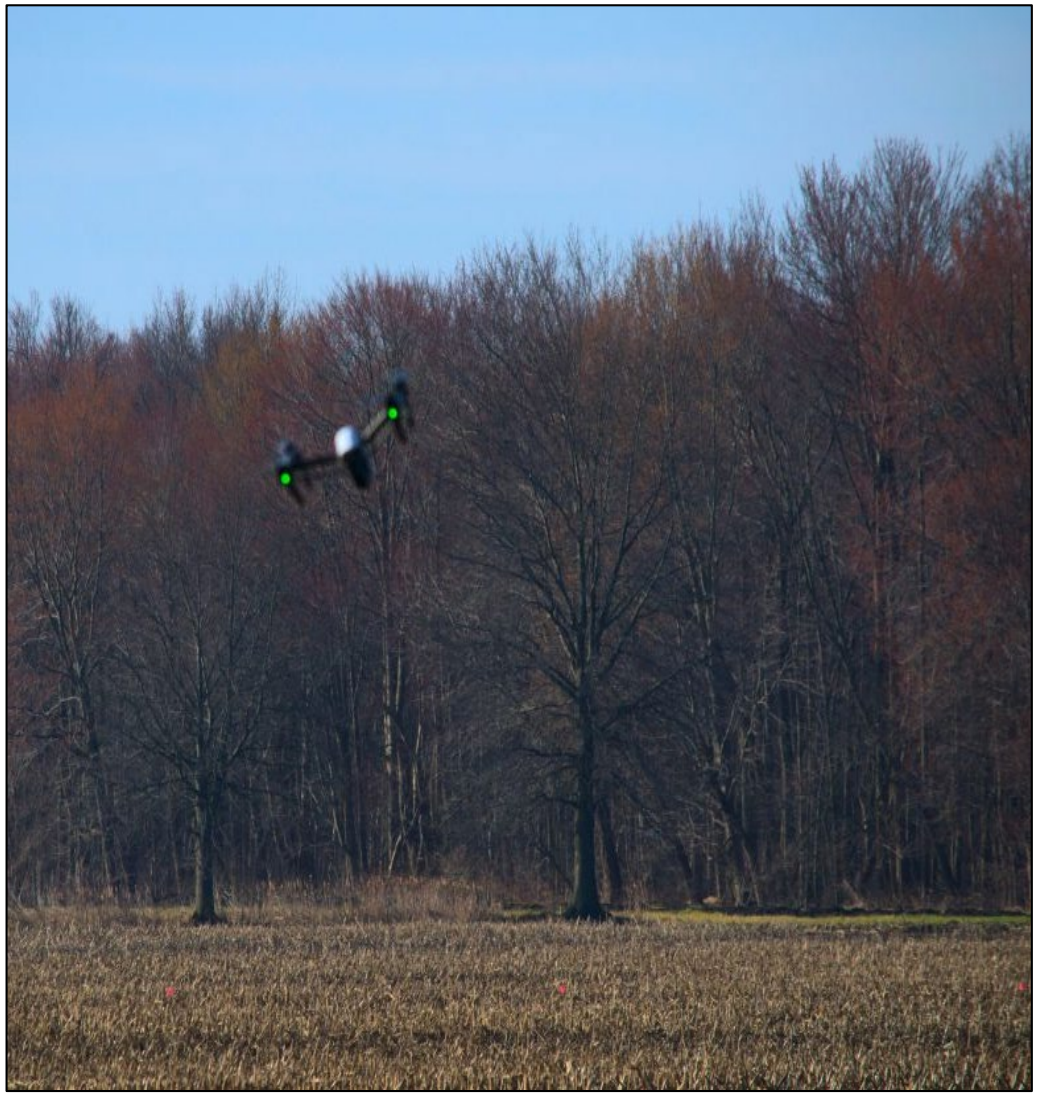
Software Programming

Learning Objectives

1. **UAV Development and Calibration:** including Python, MavProxy and DroneKit. Students will programmatically control UAVs. They will use existing tools to calibrate UAVs and to inspect their logs.
2. **FAA Regulations**: UAVs flying in a controlled airspace. Students will be familiar with governing regulations as well as safety standards.
3. **Software Engineering techniques:** Students will apply appropriate Software and Systems Engineering practices related to safety analysis, requirements engineering, architectural design, and testing to develop a non-trivial, fully functioning UAV application.
4. **Problem solving:** Students will address challenging problems such as developing and evaluating collision avoidance algorithms for multiple UAVs or correctly commanding UAVs in swarm flights.
5. **UAV Hardware:** Students will learn to work with UAV related hardware such as RTK, Raspberry Pis onboard, and wifi-in-the-sky.
6. **Ethics of Drones:** Software systems, especially ones that involve physical computing devices, impact society as a whole. Through proposing an application with Societal benefits, students will think critically about pertinent issues related to use of drones in society and be able to articulate the major issues involved.
7. **Communication and Team skills:** Students will work as part of a team to describe their technical designs and solutions verbally and in writing.

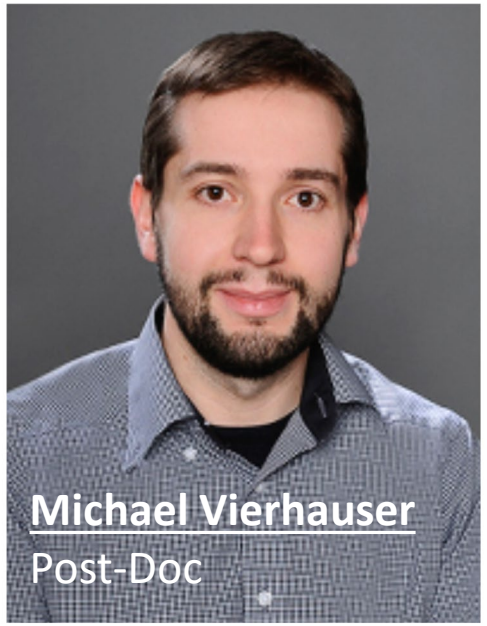
Systems Engineering with Cyber-Physical Systems

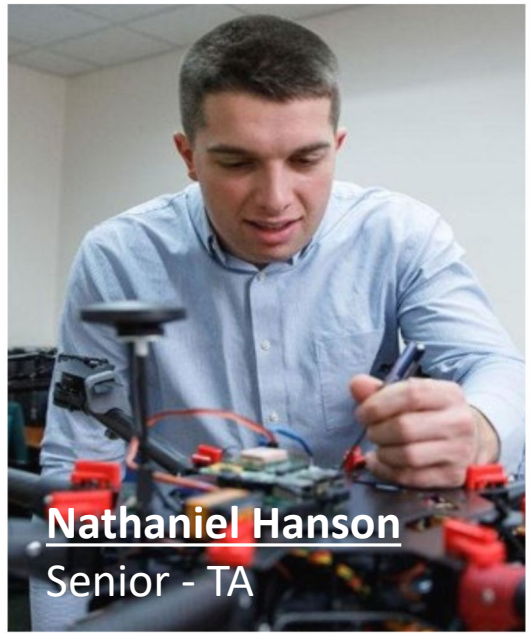




Developing Cyber-Physical Systems involves additional effort in comparison to software-only systems. We will focus on specific coding and design skills first – and then expand into other aspects of the CPS space.

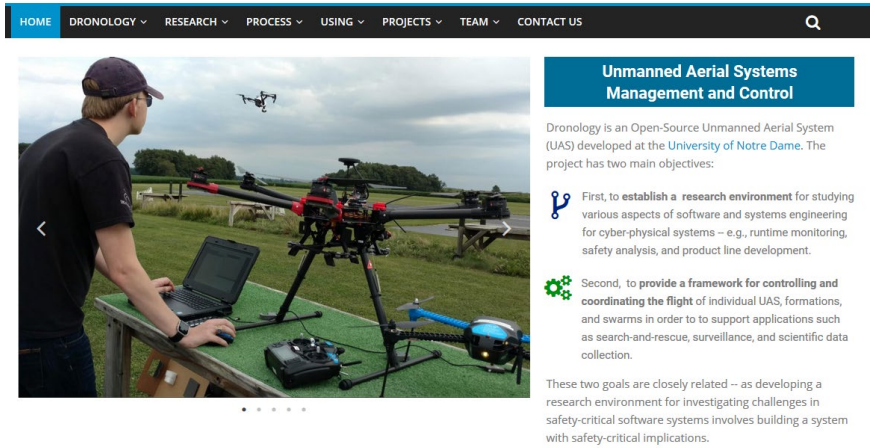
People





Dronology

A Software Engineering Research Environment using Unmanned Aerial Systems



You can see more about our ongoing project at: http://Dronology.info

We are offering paid summer internships on our Dronology project.

How the Course is structured

01/16 **Week 1** Introduction & setup	**Lab 1:** Setup Environment. Hello Drone!	**Hwk 1:** Simple SITL (5 pts)	
01/23 **Week 2** DroneKit Python basics + velocity vectors	Lab 2: Hello Physical Drone! Maps	**Hwk 2:** Experiments (5 pts)	
01/30 **Week 3** Working with multiple Drones	Lab 3: Connecting to multiple drones instances	**Hwk 3 & 4:** (5 + 5 pts) Ground Control Stations with Mapping	
02/06 **Week 4** Safety Assurance for Cyber-Physical Systems	Lab 4: Safety Cases		
02/13 **Week 5** Working with companion computers + in-air drone to drone comm.	Lab 5: SITL exercise using PIs	**Hwk 5 & 6:** (5+5 pts) Multi-drone collaboration (Teams of 3)	
02/20 **Week 6** Short presentations. Calibration. Geo-fences etc	Lab 6: Calibration activity		
02/27 **Week 7** Focus on Projects	Lab 7: Project Time	**Hwk 7:** Flying portfolio (5 pts)	Project teams finalized

Hwk 8: Take-home quiz on FAA Part 107b (5 pts)

If we get a particularly warm week on 02/19 or 03/05 we will move things around and go flying. Alternately we may offer a weekend flying opportunity.

Projects

Num	Date Assigned	Description	Points	Link
9	March 6th	Initial Team presentation + project documentation	5 points	
10	March 20th	Requirements, Architecture, and Screen Mockups due	5 points	
11	March 27th	Team presentation + initial project documentation	5 points	
12	April 3rd	Proof of concept *flying* demo due with physical UAVs	5 points	
13	April 10th	Updated project plan with specified final deliverables	5 points	
14	April 17th	Safety Analysis report due + status report	5 points	
15	April 24th	Test Case report due	5 points	
16	May 1st	Final Presentation to external panel	10 points	
17	May 9th	Final web portfolio due	10 points	
18	May 9th	Final individual project report due	5 points	

Projects

Let’s get started

Iris 3DR+

Iris+ Basics:

Flight Time: 16-22 min (depends on payload).

Charging Time: 45-60 min.Battery Type: 5100 mAh 3S

Battery Camera: Go Pro Hero 4 or other brands

Maximum Payload: 400 g. payload capacity

Motor-to-motor Dimension: 550 mm.Weight with Battery: 1282 g.

Electronics

Autopilot hardware: Next generation 32-bit Pixhawk with Cortex M4 processor

GPS: uBlox GPS with integrated magnetometer

Controller: Any PPM compatible RC unit, Preconfigured FlySky FS-TH9x RC

Telemetry: 3DR Radio 915mHz or 433mHz

Components:

Propellers: (2) 9.5 x 4.5 tiger motor multi-rotor self-tightening counterclockwise rotation,
(2) 9.5 x 4.5 tiger motor multi-rotor self-tightening clockwise rotation

Motors: 950 kV

GPS: uBlox GPS with integrated magnetometer

Mounts: Integrated GoPro camera mount with vibration dampener

Optional: Tarot brushless gimble with custom IRIS+ mounting kit

Battery: 5100 mAh 3S

Components

❶ Physical Board:

Contains sensors and processors.

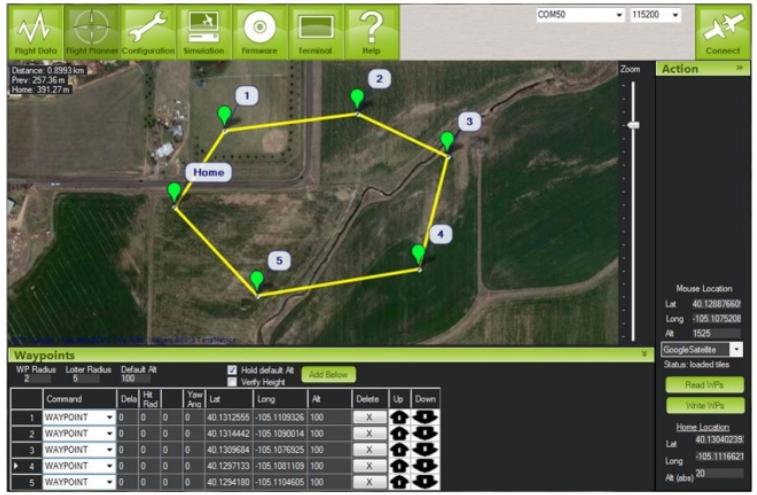
Think of it as a mini-PC that runs autopilot software.

❷ Firmware:

The firmware is the code that runs on the board.

You can update your drone's firmware use APM Planner or APM Mission Planner.

❸ Software:



Initial set-up, configuration, and testing. Mission-planning/ operation, and post-mission analysis.

Feel free to play with this for fun! We will be working more directly with SITL from the command line.

ARDUCOPTER

ARDUPLANE

Two main code branches.

http://www.ardupilot.co.uk/

Flight Controller Features

- **Gyro Stabilization** – the ability to easily keep the copter stable and level under the pilot's control.
- **Self Leveling** – the ability to let go of the pitch and roll stick on the transmitter and have the copter stay level.
- **Altitude Hold** – the ability to hover a certain distance from the ground without having to manually adjust the throttle.
- **Position Hold** – the ability to hover at a specific location.
- **Return Home** – the ability to automatically return to the point where the copter initially took off.
- **Waypoint Navigation** – the ability to set specific points on a map that copter will follow as part of a flight plan.

Pixhawk is bundled up into the 3DR Iris+.

Firmware:

If you have problems flying your drone you can update the firmware, run pre-arm tests, and recalibrate using either APM Mission Planner (Windows) or APM Planner (Windows, Linux, Mac)

Some Resources

http://diydrones.com/profiles/blogs/3dr-iris-beginners-guide-part-1-of-3
(Note: Parts 2 and 3 don't seem to exist!)

SHORT videos on recalibration

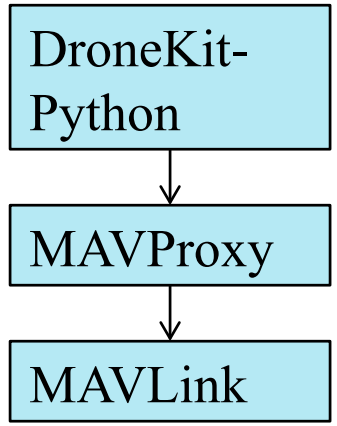
https://www.youtube.com/watch?v=oD9Z9IR2TNU
Re-*calibrate* the compass

https://www.youtube.com/watch?v=8TWjZLIATIE
Calibrating the Accelerometer.

https://www.youtube.com/watch?v=sKBVadRDcEw
Re-*Calibrating* the ESC.

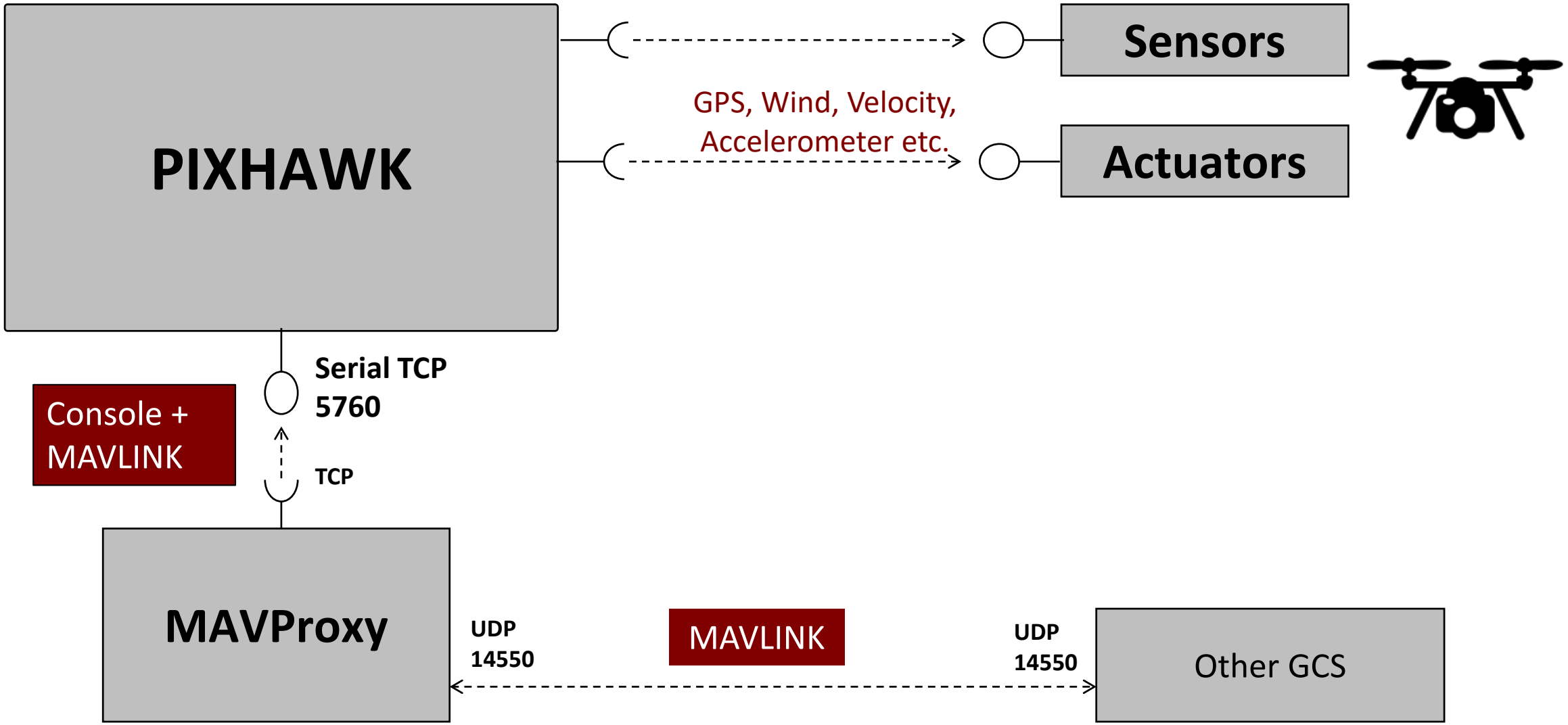
MavLink

- **MAVLink**
  Lightweight, header-only message marshalling library for micro air vehicles.
- **MAVProxy**
  Ground Control Station that communicates with any UAV using MAVLink.
- **DroneKit-Python**
  Allows developers to create Python apps that communicate with vehicles over MAVLink. It uses MAVProxy and provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations.

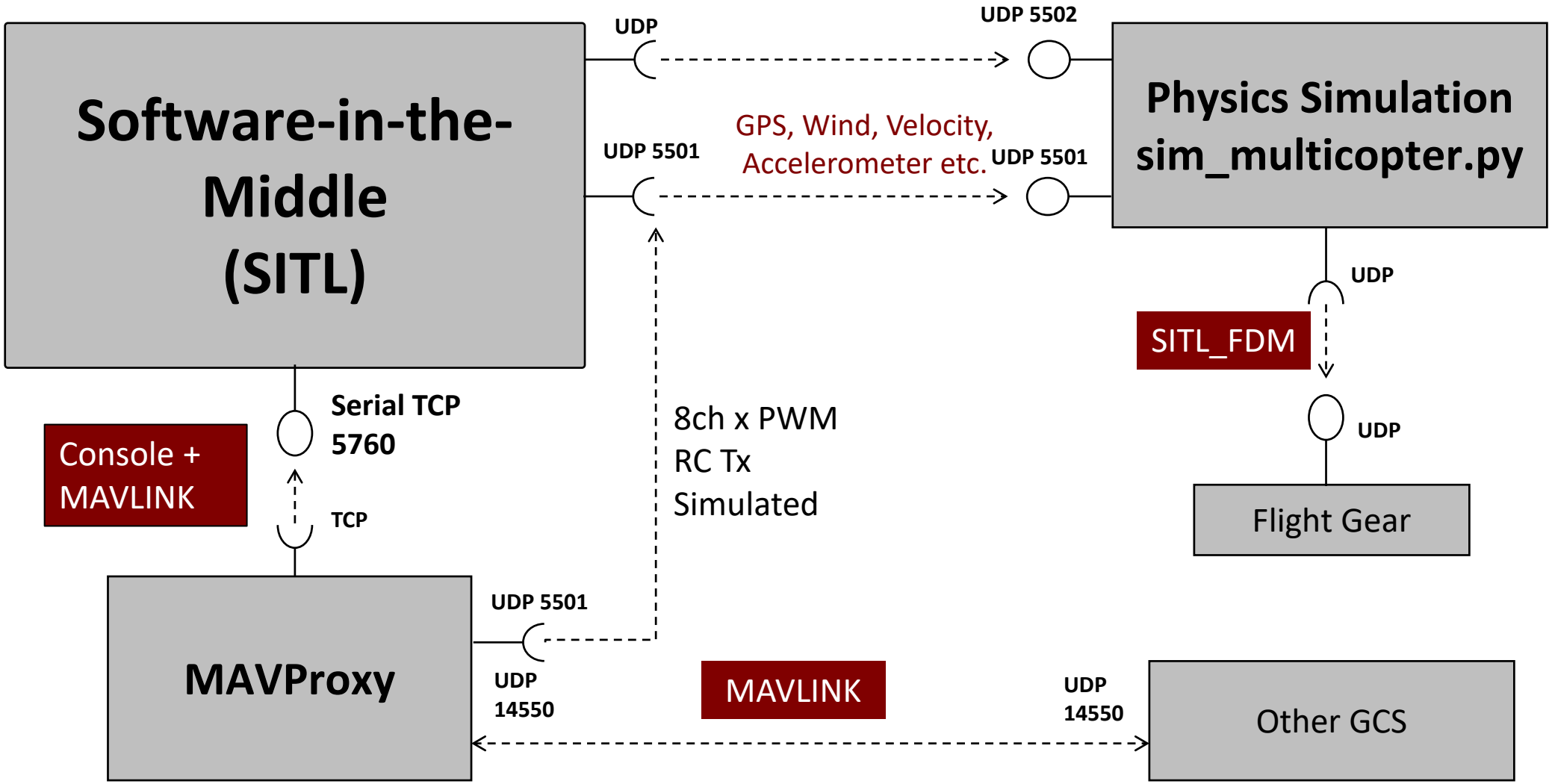


Quick Peak today… More next week.

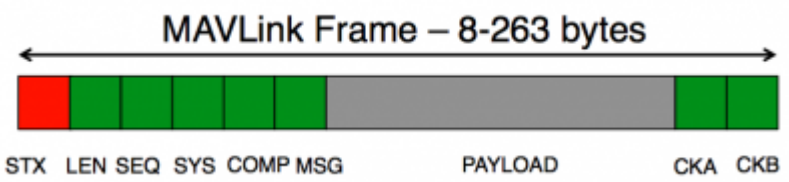
MAVLink and MAVProxy



MAVLink and MAVProxy



MAVLink Overview



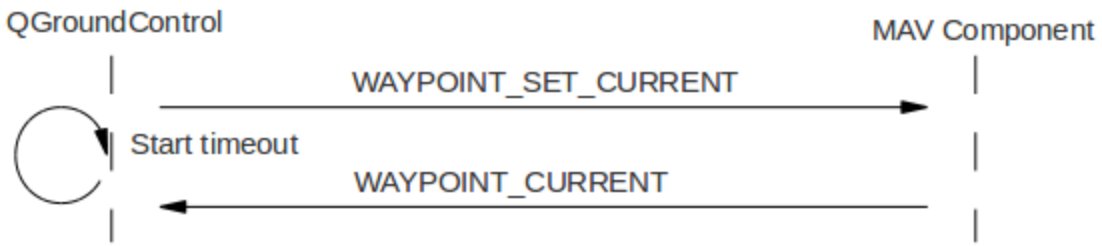
MAVLink is a lightweight, header-only message marshalling library for ***micro air vehicles***.

Packs C-structs over serial channels with high efficiency and send these packets to the ground control station.

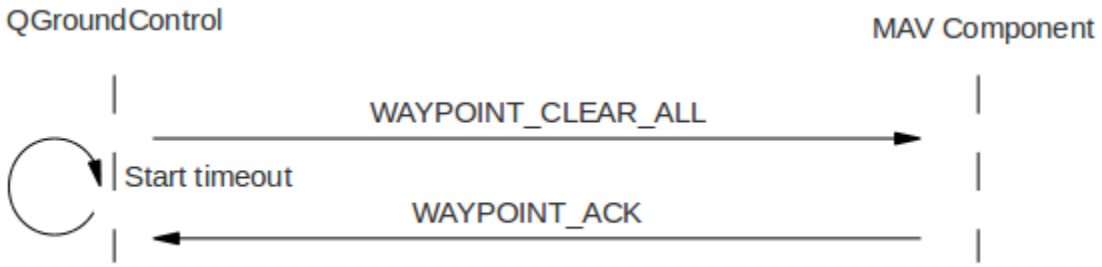
Extensively tested on the PX4, PIXHAWK, APM and Parrot AR.Drone platforms.

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 Overview



To set a new current active waypoint for a component a WAYPOINT_SET_CURRENT message is sent. After having changed the current waypoint the targeted component answers with a WAYPOINT_CURRENT message with the new current sequence number.



To clear the waypoint list of a component a WAYPOINT_CLEAR_ALL message is sent. After having cleared the waypoint list the targeted component answers with a WAYPOINT_ACK message.

Notice the timeout wait for the state to change.

MAV_AUTOPILOT

-MAV_AUTOPILOT_PX4

-MAV_AUTOPILOT_FP

-MAV_AUTOPILOT_AEROB

MAV_MODE_FLAG

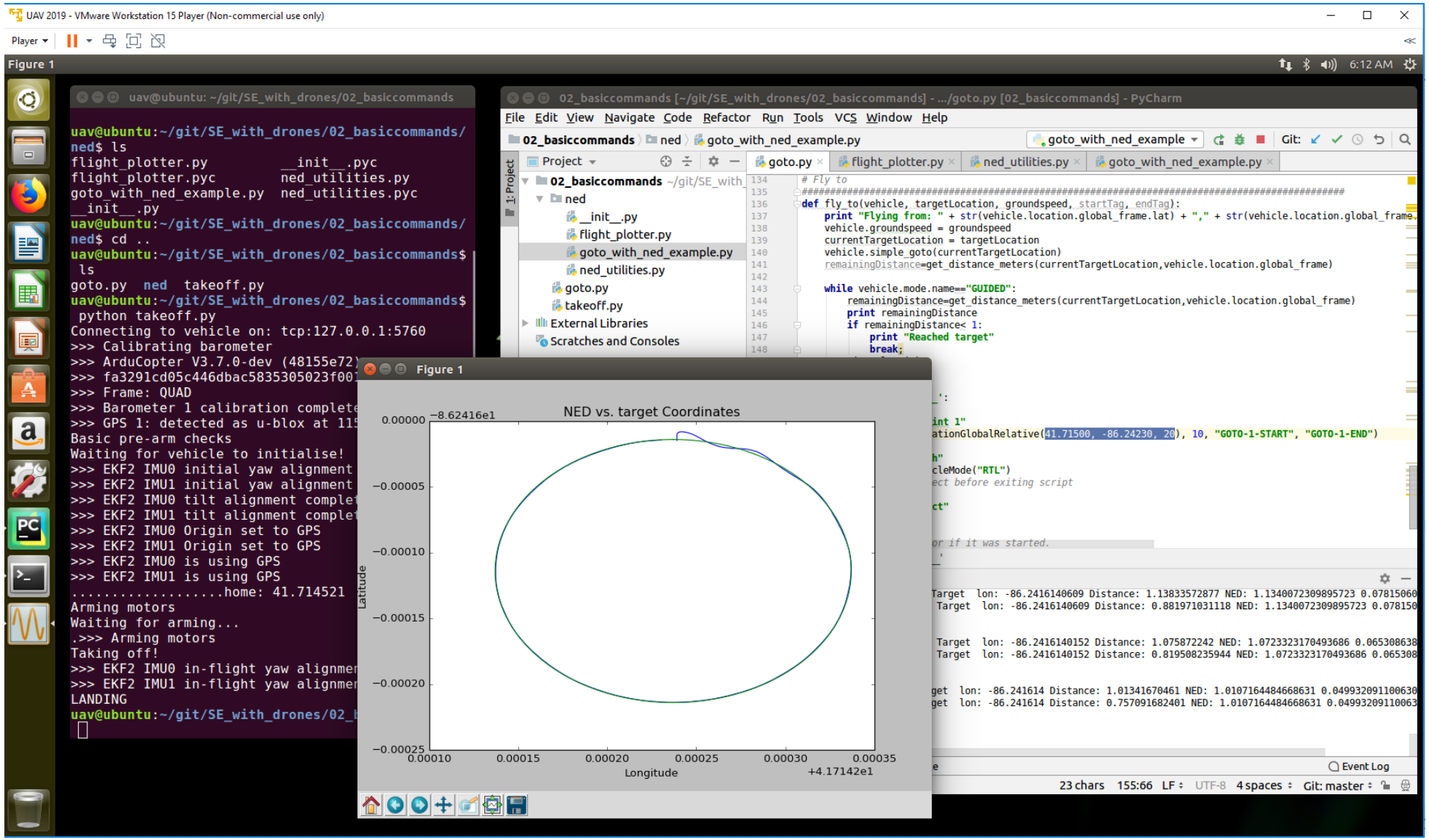
- MAV_MODE_FLAG_SAFETY_ARMED
- MAV_MODE_FLAG_GUIDED_ENABLED

MAV_TYPE

- MAV_TYPE_FIXED_WING
- MAV_TYPE_ROCKET
- MAV_TYPE_QUADROTOR

These are a SMALL sample of MAVLink enum types. For a complete list see: https://pixhawk.ethz.ch/mavlink/

In a few minutes we will start working on setup



Quick Demo

Hello Drone

❶ Python 101

```
1 # Import DroneKit-Python
2 from dronekit import connect, VehicleMode, time
3
4 #Set up option parsing to get connection string
5 import argparse
6 parser = argparse.ArgumentParser(description='Print out vehicle state information.')
7 parser.add_argument('--connect',
8                     help="vehicle connection target string.")
9 args = parser.parse_args()
10
11 connection_string = args.connect
12
13 # Connect to the Vehicle.
14 #   Set `wait_ready=True` to ensure default attributes are populated before `connect()` returns.
15 print "\nConnecting to vehicle on: %s" % connection_string
16 vehicle = connect(connection_string, wait_ready=True)
17
18 vehicle.wait_ready('autopilot_version')
19
20 # Get some vehicle attributes (state)
21 print "Get some vehicle attribute values:"
22 print " GPS: %s" % vehicle.gps_0
23 print " Battery: %s" % vehicle.battery
24 print " Last Heartbeat: %s" % vehicle.last_heartbeat
25 print " Is Armable?: %s" % vehicle.is_armable
26 print " System status: %s" % vehicle.system_status.state
27 print " Mode: %s" % vehicle.mode.name    # settable
28
29 # Close vehicle object before exiting script
30 vehicle.close()
31
32 time.sleep(5)
33
34 print("Completed")
```

❶ import

❷ argparse

❸ connect

❹ print

❺ vehicle

❻ MAVProxy bug!

❼ Working with DroneKit state changes.

① Import

```
1 # Import DroneKit-Python
2 from dronekit import connect, VehicleMode, time
3
4 #Set up option parsing to get connection string
5 import argparse
6 parser = argparse.ArgumentParser(description='Print out vehicle state information.')
```

- We'll be reusing several modules e.g. dronekit and argparse.
- Two ways to import:
  - from module import x, y
    - Allows us to access x and y without referencing module. e.g. vehicle.battery
    - Increased control over which items in the module can be used
    - Lost context (e.g. does vehicle come from dronekit?)
    - Need to import additional items from the module prior to use
  - import module
    - All items in the module are available to you
    - More tedious to type module.item (e.g. argparse.ArgumentParser)
    - Full context available.
    - Don't use import module* (why not?)

② argparse

```
1 import argparse
2 parser = argparse.ArgumentParser(description='Compute Power')
3 parser.add_argument(dest='baseNum', metavar ='N',type=int, help='the base number')
4 parser.add_argument(dest='powerNum', metavar="P", type=int, help='the power number')
5 args=parser.parse_args()
6 print (args.baseNum**args.powerNum)
```

```
jane@ubuntu:~/POPython$ python ExampleArgparse2.py -h
usage: ExampleArgparse2.py [-h] N P

Compute Power

positional arguments:
  N           the base number
  P           the power number

optional arguments:
  -h, --help  show this help message and exit
```

- Line 1: import the module
- Line 2: Create an instance of the ArgumentParser
- (Lines 3-4 & 5-6) Define two arguments. e.g.
  - dest = 'baseNum' defines a variable which can be accessed as args.baseNum
  - metavar = 'N' (or "N") and help='the base number' are both used in the help messages.
  - type=int (self-explanatory)
- Line 5: Performs the parse
- Line 6: Body of the program – accesses the arguments etc.

```
1 import argparse
2 parser = argparse.ArgumentParser(description='Compute Power')
3 parser.add_argument(dest='baseNum', metavar ='N',type=int, help='the base number')
4 parser.add_argument('--power', default='2', dest='powerNum', metavar="P", type=int, help='the power number')
5 args=parser.parse_args()
6 print (args.baseNum**args.powerNum)
```

Adds a default value of 2 to power

② argparse

```
 4 #Set up option parsing to get connection string
 5 import argparse
 6 parser = argparse.ArgumentParser(description='Print out vehicle state information.')
 7 parser.add_argument('--connect',
 8                     help="vehicle connection target string.")
 9 args = parser.parse_args()
10
11 connection_string = args.connect
```

- Very simple argument parser
  - --connect: optional argument
  - By default stores the value into a variable called args.connect

③ connect

```
from dronekit import connect

# Connect to the Vehicle (in this case a UDP endpoint)
vehicle = connect('127.0.0.1:14550', wait_ready=True)
```

Connection type	Connection string
Linux computer connected to the vehicle via USB	`/dev/ttyUSB0`
Linux computer connected to the vehicle via Serial port (RaspberryPi example)	`/dev/ttyAMA0` (also set `baud=57600`)
SITL connected to the vehicle via UDP	`127.0.0.1:14550`
SITL connected to the vehicle via TCP	`tcp:127.0.0.1:5760`
OSX computer connected to the vehicle via USB	`dev/cu.usbmodem1`
Windows computer connected to the vehicle via USB (in this case on COM14)	`com14`
Windows computer connected to the vehicle using a 3DR Telemetry Radio on COM14	`com14` (also set `baud=57600`)

wait_ready=True
-wait for default set of parameters to load.

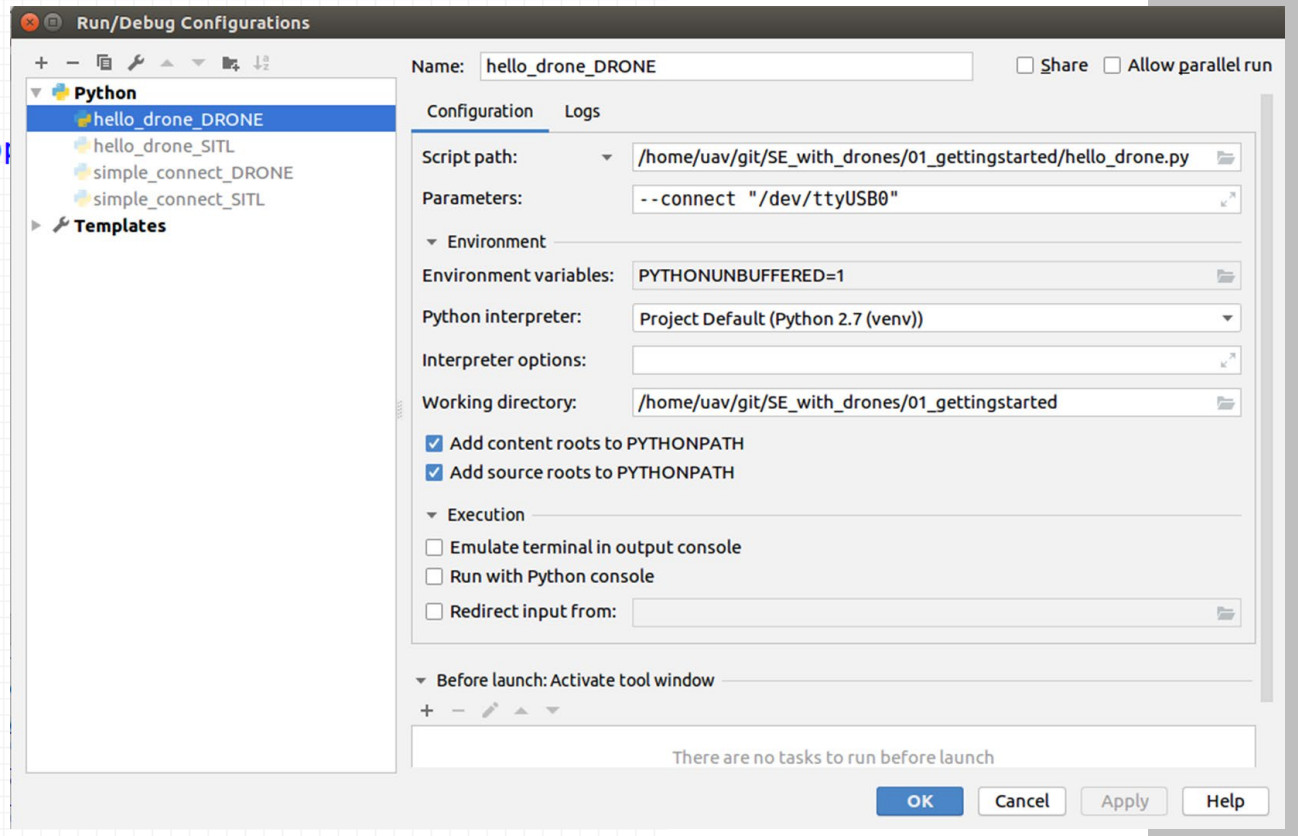
You can connect to multiple vehicles using multiple connect() calls as long as each has a unique address.

Hint: If you are connecting something, you can plug it in and then check /dev/tty* to see what has appeared.

③ connect

```python
# Import DroneKit-Python
from dronekit import connect, VehicleMode, time

#Set up option parsing to get connection string
import argparse
parser = argparse.ArgumentParser(description='Print out vehicle state information.')
parser.add_argument('--connect',
                    help="vehicle connection target string.")
args = parser.parse_args()

connection_string = args.connect

# Connect to the Vehicle.
#   Set `wait_ready=True` to ensure default attributes are pop
print "\nConnecting to vehicle on: %s" % connection_string
vehicle = connect(connection_string, wait_ready=True)

vehicle.wait_ready('autopilot_version')

# Get some vehicle attributes (state)
print "Get some vehicle attribute values:"
print " GPS: %s" % vehicle.gps_0
print " Battery: %s" % vehicle.battery
print " Last Heartbeat: %s" % vehicle.last_heartbeat
print " Is Armable?: %s" % vehicle.is_armable
print " System status: %s" % vehicle.system_status.state
print " Mode: %s" % vehicle.mode.name    # settable

# Close vehicle object before exiting script
vehicle.close()

time.sleep(5)

print("Completed")
```

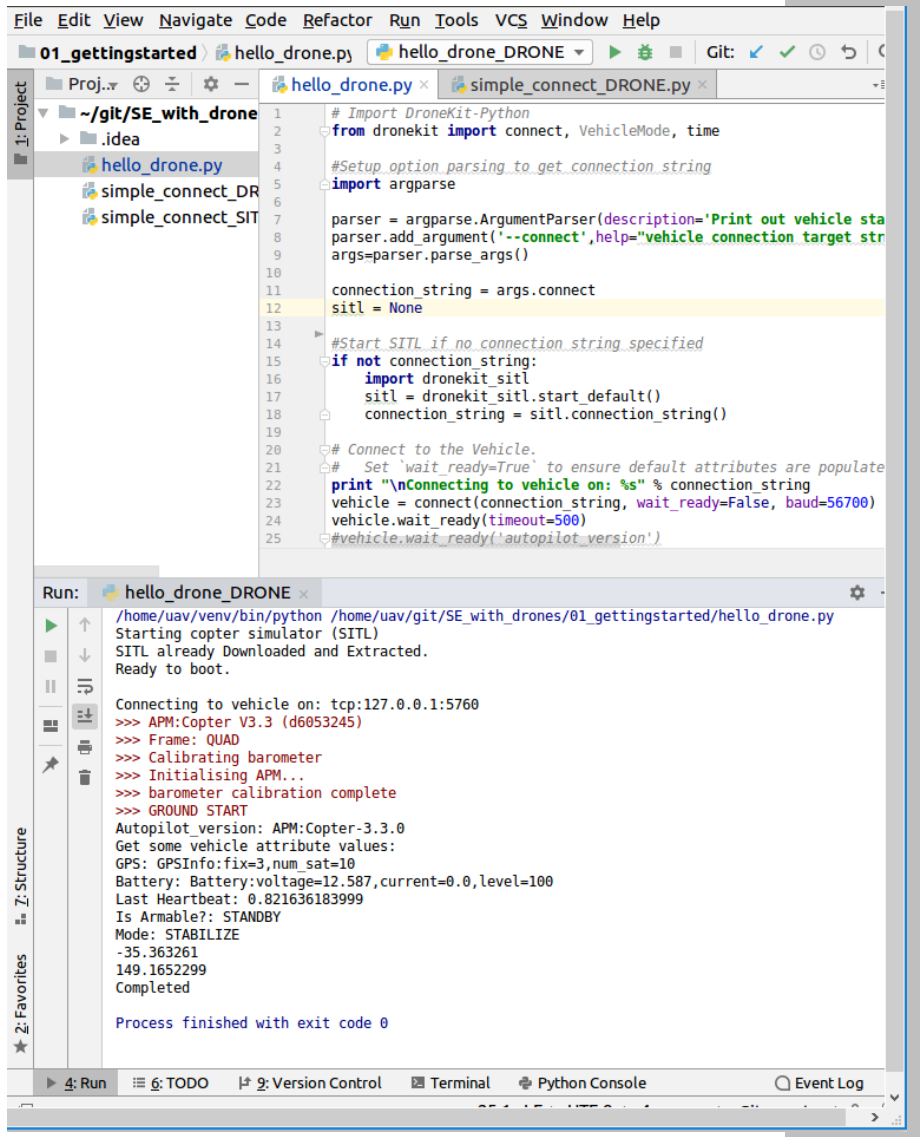
❺ vehicle



Running hello_drone.py

```python
# Import DroneKit-Python
from dronekit import connect, VehicleMode, time

#Setup option parsing to get connection string
import argparse
parser = argparse.ArgumentParser(description='Print out vehicle state information')
parser.add_argument('--connect',help="vehicle connection target string.")
args=parser.parse_args()

connection_string = args.connect
sitl = None


#Start SITL if no connection string specified
if not connection_string:
    import dronekit_sitl
    sitl = dronekit_sitl.start_default()
    connection_string = sitl.connection_string(

# Connect to the Vehicle.
#   Set `wait_ready=True` to ensure default att
print "\nConnecting to vehicle on: %s" % connec
vehicle = connect(connection_string, wait_ready

vehicle.wait_ready('autopilot_version')

#Get some vehicle attributes (state)
print "Get some vehicle attribute values:"
print "GPS: %s" % vehicle.gps_0
print "Battery: %s" % vehicle.battery
print "Last Heartbeat: %s" % vehicle.last_heart
print "Is Armable?: %s" % vehicle.system_status
print "Mode: %s" % vehicle.mode.name # settable

# Close vehicle object before exiting script
vehicle.close()

time.sleep(5)

print("Completed")
```

```
uav@ubuntu: ~/git/SE_with_drones/01_gettingstarted

uav@ubuntu:~/git/SE_with_drones/01_gettingstarted
$ python hello_drone.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 V3.3 (d6053245)
>>> Frame: QUAD
>>> Calibrating barometer
>>> Initialising APM...
>>> barometer calibration complete
>>> GROUND START
Autopilot_version: APM:Copter-3.3.0
Get some vehicle attribute values:
GPS: GPSInfo:fix=3,num_sat=10
Battery: Battery:voltage=12.587,current=0.0,level
=100
Last Heartbeat: 0.826336262
Is Armable?: STANDBY
Mode: STABILIZE
-35.363261
149.1652299
Completed
uav@ubuntu:~/git/SE_with_drones/01_gettingstarted
$
```



Flying!!

This code needs to be at the start of all drone usage. It is easiest to keep it multi-purpose for both SITL and physical drones.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
© Copyright 2015-2016, 3D Robotics.
simple_goto.py: GUIDED mode "simple goto" example (Copter Only)
Demonstrates how to arm and takeoff in Copter and how to navigate to points using Vehicle.simple_goto.
Full documentation is provided at http://python.dronekit.io/examples/simple_goto.html
"""

from dronekit import connect, VehicleMode, LocationGlobalRelative
import time


#Set up option parsing to get connection string
import argparse
parser = argparse.ArgumentParser(description='Commands vehicle using vehicle.simple_goto.')
parser.add_argument('--connect',
                    help="Vehicle connection target string. If not specified, SITL automatically started and used.")
args = parser.parse_args()

connection_string = args.connect
sitl = None


#Start SITL if no connection string specified
if not connection_string:
    import dronekit_sitl
    sitl = dronekit_sitl.start_default()
    connection_string = sitl.connection_string()
```

https://github.com/dronekit/dronekit-python/blob/master/examples/simple_goto/simple_goto.py

Connecting (again!)

```
31 # Connect to the Vehicle
32 print 'Connecting to vehicle on: %s' % connection_string
33 vehicle = connect(connection_string, wait_ready=True)
34
35
36 def arm_and_takeoff(aTargetAltitude):
37     """
38     Arms vehicle and fly to aTargetAltitude.
39     """
40
41     print "Basic pre-arm checks"
42     # Don't try to arm until autopilot is ready
43     while not vehicle.is_armable:
44         print " Waiting for vehicle to initialise..."
45         time.sleep(1)
46
47
48     print "Arming motors"
49     # Copter should arm in GUIDED mode
50     vehicle.mode = VehicleMode("GUIDED")
51     vehicle.armed = True
52
53     # Confirm vehicle armed before attempting to take off
54     while not vehicle.armed:
55         print " Waiting for arming..."
56         time.sleep(1)
57
58     print "Taking off!"
59     vehicle.simple_takeoff(aTargetAltitude) # Take off to target altitude
60
61     # Wait until the vehicle reaches a safe height before processing the goto (o
62     #  after Vehicle.simple_takeoff will execute immediately).
63     while True:
64         print " Altitude: ", vehicle.location.global_relative_frame.alt
65         #Break and return from function just below target altitude.
66         if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
67             print "Reached target altitude"
68             break
69         time.sleep(1)
70
71 arm_and_takeoff(10)
72
73 print "Set default/target airspeed to 3"
74 vehicle.airspeed = 10
```

If anyone has questions we'll open a larger version of the code. You can find all this in 02_basiccommands simple_goto.py

Note the following:

1. Take off in GUIDED or STABILIZED mode
2. Use a while loop to wait until a command has taken effect.
3. Use some print statements for debugging!

Are we there yet?

If you just issue multiple vehicle.simple_goto(...) instructions in a loop the drone will just start flying to the most recent one.

You need a way to wait for it to arrive at its destination – so you could measure distance from final location (or percentage of total distance traveled).

```
82 def get_distance_meters(location1, location2):
83         lat1 = radians(location1.lat)
84         lon1 = radians(location1.lon)
85         lat2 = radians(location2.lat)
86         lon2 = radians(location2.lon)
87 
88         dlon = radians(lon2) - radians(lon1)
89         dlat = lat2 - lat1
90 
91         a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
92         c = 2 * atan2(sqrt(a), sqrt(1 - a))
93 
94         distance = R * c * 1000
95 
96         print("Result:", distance)
97 
98         return distance
```

Are we there yet?

```
182     while vehicle.mode.name=="GUIDED": #Stop action if we are no longer in guided mode.
183         vehicle.simple_goto(currentTargetLocation)
184         remainingDistance=get_distance_metres(vehicle.location.global_frame, currentTargetLocation)
185         print "Remaining: ", remainingDistance
186         if remainingDistance<=targetDistance*0.01: #Just below target, in case of undershoot.
187             print "Reached target"
188             break;
189         time.sleep(5)
```

Note the while loop condition

Note the remaining Distance calculation

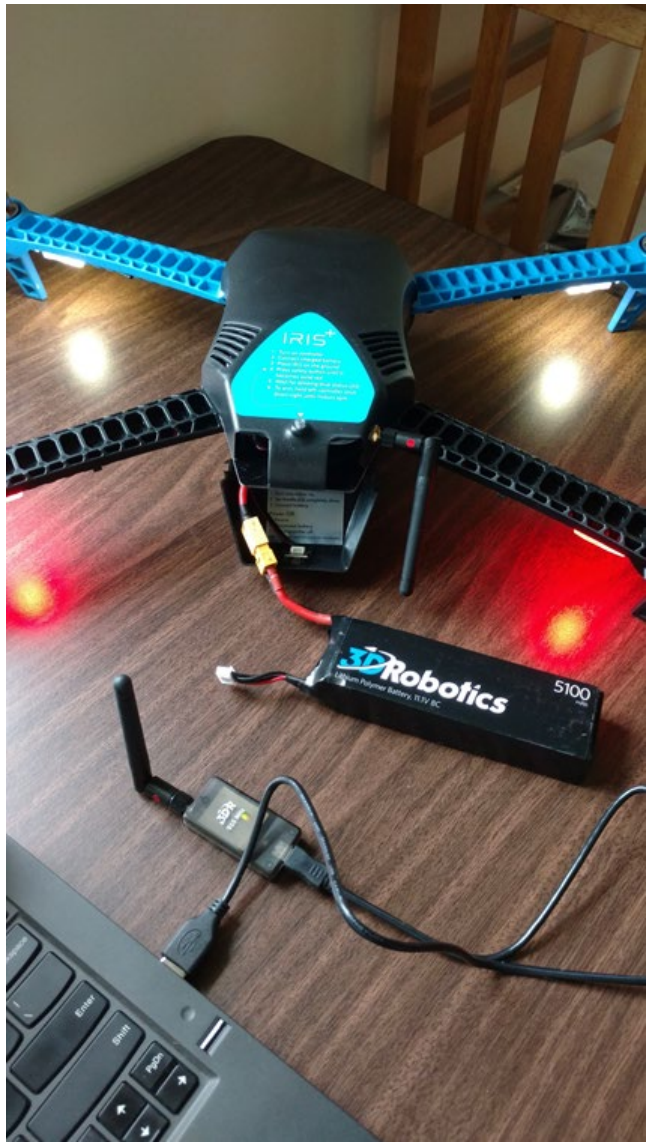
```
while True:

    nextwaypoint=vehicle.commands.next
    if nextwaypoint==0:
        print "takeoff at Home.."

    if nextwaypoint==3:
        print "going to target location.."
    if nextwaypoint==4:
        print "landing at target location.."

        break;
    time.sleep(1)
```

I decided to check MedFleet to see if this might explain the Lake Michigan crash:

Land Safely

```
91 # sleep so we can see the change in map
92 time.sleep(30)
93
94 print("Returning to Launch")
95 vehicle.mode = VehicleMode("RTL")
96
97 # Close vehicle object before exiting script
98 print("Close vehicle object")
99 vehicle.close()
100
101 # Shut down simulator if it was started.
102 if sitl:
103 sitl.stop()
```

Homework:



1. **Make sure your environment is setup**.
   If you have a problem that we can't solve today, then set an appointment with the TA

2. **Programming: Taking flight:**

   a. Choose your favorite building or area of campus. Go to google maps and identify 3-4 GPS coordinates around that building.
   b. Reusing goto.py from 02_basiccommands have the UAV take off to 20 meters, fly around the building or area (visiting the 3-4 GPS coordinates and then returns to launch and lands.
   c. Display the coordinates of the trip on the console and also print them to file. Show the remaining distance to the currently targeted waypoint. Print approximately every second. Show when each waypoint has been reached. Your logfile must be called "TripLog.log".

3. **See Assignment#1 for reading and response**