drone_Guided.py

Importing header files:

In [ ]:

```
from __future__ import print_function
from dronekit import connect, Command, VehicleMode, LocationGlobalRelative, LocationGlobal
from pymavlink import mavutil
import os
import json, urllib, math
import time
import logging , logging.handlers
```

Logging configuration:

In [ ]:

```
logging.basicConfig(filename = "Master.log" , level = logging.DEBUG , format = "%(levelname)s: %(filename)s: %(module)s: %(funcName)s: %(lineno)d:                      %(message)s")
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logFile_handler = logging.FileHandler("drone_seed_GUIDED.log")
logFile_handler.setLevel(logging.DEBUG)
logFile_streamHandler = logging.StreamHandler()
logFile_streamHandler.setLevel(logging.ERROR)
logging_formatter = logging.Formatter("%(levelname)s: %(filename)s: %(module)s: %(funcName)s: %(lineno)d:                       %(message)s")
logFile_handler.setFormatter(logging_formatter)
logFile_streamHandler.setFormatter(logging_formatter)
logger.addHandler(logFile_handler)
logger.addHandler(logFile_streamHandler)
```

Custom Class for taking lat/lon and alt points

LAT_LON_ALT class takes three parameters x,y and z save them as lon,lat and alt variables of the class object.

In [ ]:

```
class LAT_LON_ALT:
        def __init__(self,x,y,z):
                self.lon = x
                self.lat = y
                self.alt = z
```

Functions Used:

1. get_location_metres(original_location, dNorth, dEast):

Returns a LAT_LON_ALT object containing the latitude/longitude and altitude `dNorth` and `dEast` metres from the specified `original_location`. The function is useful when you want to move the vehicle around specifying locations relative to the current vehicle position. This function is relatively accurate over small distances (10m within 1km) except close to the poles.

The function does not change the altitude value

Reference: http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longitude-by-some-amount-of-meters (http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longitude-by-some-amount-of-meters)

In [ ]:

```
def get_location_metres(original_location, dNorth, dEast):
        #Radius of "spherical" earth
        earth_radius=6378137.0

        #Coordinate offsets in radians
        dLat = dNorth/earth_radius
        dLon = dEast/(earth_radius*math.cos(math.pi*original_location.la
t/180))

        #New position in decimal degrees
        newlat = original_location.lat + (dLat * 180/math.pi)
        newlon = original_location.lon + (dLon * 180/math.pi)
        new_location = LAT_LON_ALT(newlon,newlat,original_location.alt)
        return new_location
```

2. get_distance_metres(aLocation1, aLocation2):

Returns the ground distance in metres between two LAT_LON_ALT objects.

This method is an approximation, and will not be accurate over large distances and close to the earth's poles.

Reference: https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/common.py (https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/common.py)

In [ ]:

```
def get_distance_metres(aLocation1, aLocation2):
    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
```

3. arm_and_takeoff(aTargetAltitude):

Arms vehicle and fly to a target altitude.

```
In [ ]: def arm_and_takeoff(aTargetAltitude):
            # Don't try to arm until autopilot is ready
                while not vehicle.is_armable:
                    logger.warning(" Waiting for vehicle to initialise...")
                    time.sleep(1)
                # Set mode to GUIDED for arming and takeoff:
                while (vehicle.mode.name != "GUIDED"):
                    vehicle.mode = VehicleMode("GUIDED")
                    time.sleep(0.1)
                # Confirm vehicle armed before attempting to take off
                while not vehicle.armed:
                    vehicle.armed = True
                    logger.warning(" Waiting for arming...")
                    time.sleep(1)
                print("Taking off!")
                logger.info("Taking off!")
                vehicle.simple_takeoff(aTargetAltitude) # Take off to target alt
        itude

                # Wait until the vehicle reaches a safe height
                # before allowing next command to process.
                while True:
                    requiredAlt = aTargetAltitude*0.95
                    #Break and return from function just below target altitude.
                    if vehicle.location.global_relative_frame.alt>=requiredAlt:
                        print("Reached target altitude of %f" % (aTargetAltitud
        e))
                        logger.info("Reached target altitude of %f" % (aTargetAl
        titude))
                        break
                    logger.info("Altitude: %f < %f" % (vehicle.location.global_r
        elative_frame.alt,requiredAlt))
                    time.sleep(1)
```

4. goto(targetLocation):

Send SET_POSITION_TARGET_GLOBAL_INT command to request the vehicle fly to a specified LocationGlobal. At time of writing, acceleration and yaw bits are ignored.

```
msg = vehicle.message_factory.set_position_target_global_int_encode(
    0,       # time_boot_ms (not used)
    0, 0,    # target system, target component
    mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT, # frame
    0b0000111111111000, # type_mask (only speeds enabled)
    aLocation.lat, # lat_int - X Position in WGS84 frame in 1e7 * meters
    aLocation.lon, # lon_int - Y Position in WGS84 frame in 1e7 * meters
    aLocation.alt, # alt - Altitude in meters in AMSL altitude, not WGS84 if ab
solute or relative, above  terrain if GLOBAL_TERRAIN_ALT_INT
    0, # X velocity in NED frame in m/s
    0, # Y velocity in NED frame in m/s
    0, # Z velocity in NED frame in m/s
    0, 0, 0, # afx, afy, afz acceleration (not supported yet, ignored in GCS_Ma
vlink)
    0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)
```

At first we store the target location in Vehicle Global Relative Frame object and calculate the target distance.

Then using the mavlink and vehicle message factory send command to drone to move to next target location.

There is another way to send command to drone by using the inbuild function vehicle_simple_goto(). (But we are using our custom command)

Note: Take care of the coordinate system followed by different msg command (like WGS84 or Lat/lon system etc)

In [ ]:

```
def goto(targetLocation):
        # send command to vehicle
        logger.debug("Target location lat: %f , lon: %f , alt: %f" % (targetLocation.lat,targetLocation.lon,targetLocation.alt))
        vc_in_loc = vehicle.location.global_relative_frame
        vehicle_initialLocation = LAT_LON_ALT(vc_in_loc.lon,vc_in_loc.lat,vc_in_loc.alt)
        targetDistance = get_distance_metres(vehicle_initialLocation, targetLocation)
        msg = vehicle.message_factory.set_position_target_global_int_encode( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT, 0b0000111111111000, targetLocation.lat*1e7, targetLocation.lon*1e7, targetLocation.alt, 0, 0, 0, 0, 0, 0, 0, 0)
        vehicle.send_mavlink(msg)
        logger.debug("Send Command Message to drone")
        # target = LocationGlobal(targetLocation.lat,targetLocation.lon,targetLocation.alt)
        # vehicle.airspeed=15
        # vehicle.simple_goto(target)
```

CRITICAL :

There may be case that our msg command send above is dropped due to network failure. In that case drone will be stuck at a point.

To handle this we monitor the drone for 5 sec after sending the command. If drone doesnot move (i.e. targetDistance is still greater than 90% of that of at first sec) then we resend the msg command to drone.

Following code even handle the drone Mode change if any.

In [ ]:

```
        fiveSecondCheck = targetDistance
        fiveCounter = 1
        logger.debug("fiveSecondCheck distance: %f " % (fiveSecondCheck))
        logger.debug("fiveCounter value: %d " % (fiveCounter))
        while True:
            logger.debug("mode: %s" % vehicle.mode.name) #Stop action if we are no longer in guided mode.
            while (vehicle.mode.name != "GUIDED"):
                vehicle.mode = VehicleMode("GUIDED")
                time.sleep(0.1)

            if fiveCounter == 1:
                vc_loc = vehicle.location.global_relative_frame
                vehicle_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_loc.lat,vc_loc.alt)
                fiveSecondCheck = get_distance_metres(vehicle_currentLocation, targetLocation)
                logger.debug("fiveSecondCheck distance: %f " % (fiveSecondCheck))
                logger.debug("fiveCounter value: %d " % (fiveCounter))

            if fiveCounter >=5:
                logger.debug("fiveSecondCheck distance: %f " % (fiveSecondCheck))
                logger.debug("fiveCounter value: %d " % (fiveCounter))
                fiveCounter = 1
                vc_loc = vehicle.location.global_relative_frame
                vehicle_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_loc.lat,vc_loc.alt)
                currentDistanceToTarget = get_distance_metres(vehicle_currentLocation, targetLocation)
                logger.debug("fiveSecondCheck currentDistanceToTarget distance: %f " % (currentDistanceToTarget))
                if currentDistanceToTarget >= 0.9* fiveSecondCheck:
                    #resend the msg command to drone
                    vehicle.send_mavlink(msg)
                    logger.critical("Last command message dropped. Resending the command message to drone")
                    logger.debug("Resend the command message to drone.")

            vc_loc = vehicle.location.global_relative_frame
            vehicle_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_loc.lat,vc_loc.alt)
            remainingDistance=get_distance_metres(vehicle_currentLocation, targetLocation)
            logger.info("Distance to target: %f" % (remainingDistance))
            print("Distance to target: %f" % (remainingDistance))
            if remainingDistance <= 1: #Just below target, in case of undershoot.
                logger.info("Reached target")
                break
            fiveCounter += 1
            time.sleep(1)
```

5. print_vehicle_attributes():

This function list all the attributes of the vehicle and stores it in log file:

In [ ]:

```python
def print_vehicle_attributes():
        logger.info("Autopilot Firmware version: %s" % vehicle.version)
        logger.info("Autopilot capabilities (supports ftp): %s" % vehicle.capabilities.ftp)
        logger.info("Global Location:%s" % vehicle.location.global_frame)
        logger.info("Global Location (relative altitude): %s" % vehicle.location.global_relative_frame)
        logger.info("Local Location: %s" % vehicle.location.local_frame)
        logger.info("Attitude: %s" % vehicle.attitude)
        logger.info("Velocity: %s" % vehicle.velocity)
        logger.info("GPS: %s" % vehicle.gps_0)
        logger.info("Groundspeed: %s" % vehicle.groundspeed)
        logger.info("Airspeed: %s" % vehicle.airspeed)
        logger.info("Gimbal status: %s" % vehicle.gimbal)
        logger.info("Battery: %s" % vehicle.battery)
        logger.info("EKF OK?: %s" % vehicle.ekf_ok)
        logger.info("Last Heartbeat: %s" % vehicle.last_heartbeat)
        logger.info("Rangefinder: %s" % vehicle.rangefinder)
        logger.info("Rangefinder distance: %s" % vehicle.rangefinder.distance)
        logger.info("Rangefinder voltage: %s" % vehicle.rangefinder.voltage)
        logger.info("Heading: %s" % vehicle.heading)
        logger.info("Is Armable?: %s" % vehicle.is_armable)
        logger.info("System status: %s" % vehicle.system_status.state)
        logger.info("Mode: %s" % vehicle.mode.name)
        logger.info("Armed: %s" % vehicle.armed)
```

6. print_vehicle_parameters():

This function list all the parameters of the vehicle and stores it in log file.

In [ ]:

```python
def print_vehicle_parameters():
        logger.info ("Print all parameters (`vehicle.parameters`):")
        for key, value in vehicle.parameters.iteritems():
            logger.info (" Key:%s Value:%s" % (key,value))
```

7. startMission(startingLocation):

This function controls the planned mission of drone. Collect all the waypoints from the file and use goto() function to give commands to drone.

Once the drone reaches the required location we can drop the seed.

In [ ]:

```python
def startMission(startingLocation):
        with open(waypoint_file,"r") as waypointFile:
            for pt in waypointFile:
                current_line = pt.split(",")
                nextLocation = LAT_LON_ALT(float(current_line[1]),float(current_line[0]),startingLocation.alt)
                logger.debug("Next location lat: %f , lon: %f , alt: %f",nextLocation.lat,nextLocation.lon,nextLocation.alt)
                goto(nextLocation)
                print("Dropping Seed")
                logger.info("Dropping Seed")
        waypointFile.close()
```

Main Body :

In [ ]:
```
startingLocation = LAT_LON_ALT(0.0,0.0,0.0) #startingLocation variable
waypoint_file = ""      #stores the waypoint file name
```

Takes the lat lon and alt value from USER

In [ ]:
```
while True:
        try:
            startingLocation.lat = float(input("Please enter the latitute of starting point:\n"))
            logger.debug("USER entered latitute value: %s",str(startingLocation.lat))
            if(startingLocation.lat<0 or startingLocation.lat>90):
                print("Latitude value must be between 0 and 90")
                continue
            startingLocation.lon = float(input("Please enter the longitude of starting point:\n"))
            logger.debug("USER entered longitude value: %s",str(startingLocation.lon))
            if(startingLocation.lon<0 or startingLocation.lon>180):
                print("Langitude value must be between 0 and 180")
                continue
            startingLocation.alt = float(input("Please enter the altitude for the drone:\n"))
            logger.debug("USER entered altitude value: %s",str(startingLocation.alt))
            if(startingLocation.alt<0):
                print("Altitude value must be positive")
                continue
            break
        except:
            logger.error("Oops!  That was no valid lat/lon or altitude. Try again...")
```

Takes the waypoint file name from USER

In [ ]:
```
while True:
        waypoint_file = raw_input("Enter the waypoint file name with extension:\n")
        if os.path.exists(waypoint_file):
            break
        else:
            print("Enter file does not exists. Please re enter correct file")
            logger.error("Enter file does not exists.")
            continue
```

Set up option parsing to get connection string

In [ ]:
```
import argparse
parser = argparse.ArgumentParser(description='Demonstrates Seed Plantation Mission in GUIDED mode.')
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

```
In [ ]: if not connection_string:
                import dronekit_sitl
                sitl = dronekit_sitl.start_default(lat=startingLocation.lat,lon=
        startingLocation.lon)
                connection_string = sitl.connection_string()
```

Connect to the Vehicle

```
In [ ]: print('Connecting to vehicle on: %s' % connection_string)
        logger.info('Connecting to vehicle on: %s' % connection_string)
        vehicle = connect(connection_string, wait_ready=True)
```

Log vehicle attributes:

```
In [ ]: print_vehicle_attributes()
```

Log vehicle parameters:

```
In [ ]: print_vehicle_parameters()
```

```
In [ ]: print("Arm and Takeoff")
        logger.info("Arm and Takeoff")
        arm_and_takeoff(startingLocation.alt)
```

Start the mission by calling the startMission() function

After completion of mission RTL (Return to Launch)

```
In [ ]: print("Starting mission")
        logger.info("Starting mission")

        startMission(startingLocation)


        print('Return to launch')
        logger.critical("Return to launch")
        while (vehicle.mode.name != "RTL"):
                vehicle.mode = VehicleMode("RTL")
                time.sleep(0.1)
```

Close vehicle object before exiting script

```
In [ ]: print("Close vehicle object")
        logger.info("Close vehicle object")
        vehicle.close()
```

Shut down simulator.

In [ ]:

```
if sitl is not None:
    sitl.stop()
print("Completed...")
logger.info("Completed...")
```