drone_Auto.py

Importing header files:

```
In [ ]: from __future__ import print_function
        from dronekit import connect, Command, VehicleMode, LocationGlobalRelati
        ve
        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 , fo
        rmat = "%(levelname)s: %(filename)s: %(funcName)s: %(lineno)d:
                %(message)s")
        logger = logging.getLogger(__name__)
        logger.setLevel(logging.DEBUG)
        logFile_handler = logging.FileHandler("drone_seed_AUTO.log")
        logFile_handler.setLevel(logging.DEBUG)
        logFile_streamHandler = logging.StreamHandler()
        logFile_streamHandler.setLevel(logging.ERROR)
        logging_formatter = logging.Formatter("%(levelname)s: %(filename)s: %(fu
        ncName)s: %(lineno)d:                       %(message)s")
        logFile_handler.setFormatter(logging_formatter)
        logFile_streamHandler.setFormatter(logging_formatter)
        logger.addHandler(logFile_handler)
        logger.addHandler(logFile_streamHandler)
```

Functions Used:

1. get_distance_metres(aLocation1, aLocation2):

Returns the ground distance in metres between two LocationGlobal 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
```

2. distance_to_current_waypoint():

Gets distance in metres to the current waypoint. It returns "None" for the first waypoint (Home location).

```
In [ ]: def distance_to_current_waypoint():
                nextwaypoint = vehicle.commands.next
                if nextwaypoint==0:
                        return None
                missionitem=vehicle.commands[nextwaypoint-1] #commands are zero
        indexed
                lat = missionitem.x
                lon = missionitem.y
                alt = missionitem.z
                targetWaypointLocation = LocationGlobalRelative(lat,lon,alt)
                distancetopoint = get_distance_metres(vehicle.location.global_fr
        ame, targetWaypointLocation)
                return distancetopoint
```

3. arm_and_takeoff(aTargetAltitude):

Arms vehicle and fly to a target altitude. Don't try to arm until autopilot is ready.

```
In [ ]: def arm_and_takeoff(aTargetAltitude):
                while not vehicle.is_armable:
                        logger.warning("Waiting for vehicle to initialise...")
                        time.sleep(1)
```

Set mode to GUIDED for arming and takeoff:

```
In [ ]:         while (vehicle.mode.name != "GUIDED"):
                        vehicle.mode = VehicleMode("GUIDED")
                        time.sleep(0.1)
```

Confirm vehicle armed before attempting to take off:

```
In [ ]:         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:

```
In [ ]:         while True:
                        requiredAlt = aTargetAltitude*0.95
                        #Break and return from function just below target altitu
        de.
                        if vehicle.location.global_relative_frame.alt>=requiredA
        lt:
                                print("Reached target altitude of %f" % (aTarget
        Altitude))
                                logger.info("Reached target altitude of %f" % (a
        TargetAltitude))
                                break
                        logger.info("Altitude: %f < %f" % (vehicle.location.glob
        al_relative_frame.alt,requiredAlt))
                        time.sleep(1)
```

4. print_vechicle_attributes():

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

```
In [ ]: def print_vehicle_attributes():
                logger.info("Autopilot Firmware version: %s" % vehicle.version)
                logger.info("Autopilot capabilities (supports ftp): %s" % vehicl
        e.capabilities.ftp)
                logger.info("Global Location:INFO:__main__: Key:BATT_CURR_PIN Va
        lue:12.0 %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: %sINFO:__main__:Distance to waypoint (2):
        50.5458561177" % 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.dis
        tance)
                logger.info("Rangefinder voltage: %s" % vehicle.rangefinder.volt
        age)
                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)
```

5. print_vechicle_parameters():

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

```
In [ ]: 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))
```

Main Body :

```
In [ ]: start_lat = 0.0         #latitute variable
        start_lon = 0.0         #longitude variable
        start_alt = 0.0         #altitude variable
        waypoint_file = ""      #stores the waypoint file name
```

Takes the lattitude, longitude and altitude value from USER and check if USER enters the correct values or not.

```
In [ ]: while True:
                try:
                        start_lat = float(input("Please enter the latitute of st
        arting point:\n"))
                        logger.debug("USER entered latitute value: %s",str(start
        _lat))
                        if(start_lat<0 or start_lat>90):
                                print("Latitude value must be between 0 and 90")
                                continue
                        start_lon = float(input("Please enter the longitude of s
        tarting point:\n"))
                        logger.debug("USER entered longitude value: %s",str(star
        t_lon))
                        if(start_lon<0 or start_lon>180):
                                print("Langitude value must be between 0 and 18
        0")
                                continue
                        start_alt = float(input("Please enter the altitude for t
        he drone:\n"))
                        logger.debug("USER entered altitude value: %s",str(start
        _alt))
                        if(start_alt<0):
                                print("Altitude value must be positive")
                                continue
                        break
                except:
                        logger.error("Oops!  That was no valid lat/lon or altitu
        de.  Try again...")
```

Takes the waypoint file name from USER

```
In [ ]: while True:
                waypoint_file = raw_input("Enter the waypoint file name with ext
        ension:\n")
                if os.path.exists(waypoint_file):
                        break
                else:
                        print("Enter file does not exists. Please re enter corre
        ct 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 Plantati
        on Mission.')
        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=start_lat,lon=start_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()
```

Now download the vehicle waypoints

```
In [ ]: cmds = vehicle.commands
        cmds.wait_ready()
        cmds = vehicle.commands
        cmds.clear()
        line_count = 0    #Variable that keep track of total commands
```

Add command for starting location:

```
In [ ]: cmd = Command( 0,0,0,mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavut
        il.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 0, 0, 0, 0,start_lat, start_lon,st
        art_alt)
        cmds.add(cmd)
```

Open the waypoint file and add command for all waypoints:

```
In [ ]: with open(waypoint_file,"r") as way_p:
                for pt in way_p:
                        current_line = pt.split(",")
                        line_count +=1
                        lat = float(current_line[0])
                        lon = float(current_line[1])
                        logger.debug ("Point: %f %f" %(lat, lon))
                        cmd = Command( 0,0,0,mavutil.mavlink.MAV_FRAME_GLOBAL_RE
        LATIVE_ALT,mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 5, 0, 0, 0,lat, lo
        n, start_alt)
                        cmds.add(cmd)
                """"
                        Add the codes/ mechanism for dropping seed here. Depends
        on hardware

                """
        way_p.close()
```

Add command for last location i.e launch location:

In [ ]:
```
cmd = Command( 0,0,0,mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavut
il.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 0, 0, 0, 0,start_lat, start_lon,st
art_alt)
cmds.add(cmd)
```

Upload clear message and command messages to vehicle.

In [ ]:
```
print("Uploading waypoints to vehicle..." )
logger.info("Uploading waypoints to vehicle...")
cmds.upload()
print("Arm and Takeoff")
logger.info("Arm and Takeoff")
arm_and_takeoff(start_alt)

print("Starting mission")
logger.info("Starting mission")
```

Reset mission set to first (0) waypoint

In [ ]:
```
vehicle.commands.next=0
```

Set mode to AUTO to start mission:

In [ ]:
```
while (vehicle.mode.name != "AUTO"):
        vehicle.mode = VehicleMode("AUTO")
        time.sleep(0.1)
```

Monitor mission

Calculate the distance to next waypoint at regular interval (here 1 sec) and if distance is < 1.5m we assume that drone has reached the point where it has to drop the seed and Seed Dropping is going on. (Thats why we see multiple dropping seed print statement in terminal)

When we reach the last point, RTL (Return to launch) command is executed by changing the drone mode to RTL.

In [ ]:
```
while True:
        nextwaypoint=vehicle.commands.next
        print('Distance to waypoint (%s): %s' % (nextwaypoint, distance_
to_current_waypoint()))
        logger.info('Distance to waypoint (%s): %s' % (nextwaypoint, dis
tance_to_current_waypoint()))
        if distance_to_current_waypoint()<1.5:
                print("Dropping Seed")
                logger.critical("Dropping Seed")
        if nextwaypoint==line_count+1:
                print("Exit 'standard' mission when start heading to fin
al waypoint or start location")
                logger.info("Exit 'standard' mission when start heading
to final waypoint or start location")
                break;
        time.sleep(1)

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

Finally 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...")
```