```python
1    # Stock modules
2    import os
3    import sys
4    import logging
5    import time
6    import dronekit
7    import threading
8    import numpy
9
10   # Custom modules
11   from connect import Connect   # For connecting to the vehicle
12   from observe import Observe   # For observing the state of the vehicle
13   from threads import thrd  # For multithreading capabilities
14   import sound  # For playing sounds on the background (without affecting main thread
     )
15   import angle  # For operations with angles (to avoid discontinuities)
16   from control import Control  # For taking and giving control to the pilot, checking
      if it was successful
17   from auto import Auto  # For controling autonomous flight
18   from sonar import Sonar # For sonar sensors operation
19
20   ## Set-up logging ##
21   logFilename=os.path.dirname(os.path.realpath(__file__))+"/logs/"+str(time.strftime(
     "%Y%m%d-%H%M%S"))+".txt"
22
23   fid=open(logFilename,"w")    # Open and then close to create a new file
24   fid.close()
25
26   logging.basicConfig(filename=logFilename,level=logging.DEBUG,format='%(asctime)s %(
     name)s:%(levelname)s %(message)s')
27
28
29   #### Step 1: Connect to vehicle ####
30
31   logStr = "\nStart of script"
32   print logStr
33   logging.info(logStr)
34
35   vehicle = Connect()
36
37   logStr = "Vehicle connected"
38   print logStr
39   logging.info(logStr)
40
41   #### Step 2: Observe state until "take control" condition is met ####
42
43   sonars=[Sonar(3,4),Sonar(14,15),Sonar(17,18)]
44
45   """
46   for c in range(10): # Measure several times to have data on velocity
47       for s in range(3):
48           sonars[s].measureDistance()
49           sonars[s].computeVelocity()
50   """
51
52   for c in range(10):      # Pre-populate arrays
53       print ""
54
55       for s in range(3):
56           sonars[s].measureDistance()
57           sonars[s].computeVelocity()
58           sonars[s].calculateCollision()
59
```

```python
60          logStr = "S%d>> Distance: %.3f [m] Velocity: %.2f [m/s]  Tcollision: %.2f [
    s]  Tsafe: %.2f [s]" % (s,sonars[s].avgDistance,sonars[s].avgVelocity,sonars[s].Tco
    llision,sonars[s].Tsafe)
61          print logStr
62          logging.info(logStr)
63
64
65  logStr = "Starting measurements"
66  print logStr
67  logging.info(logStr)
68
69  while not (sonars[s].Tsafe < 0 and sonars[s].Tcollision > 0):
70  #while not avgDistance < 1:
71
72      for s in range(3):
73
74          sonars[s].measureDistance()
75          sonars[s].computeVelocity()
76          sonars[s].calculateCollision()
77
78          logStr = "S%d>> Distance: %.3f [m] Velocity: %.2f [m/s]  Tcollision: %.2f [
    s]  Tsafe: %.2f [s]" % (s,sonars[s].avgDistance,sonars[s].avgVelocity,sonars[s].Tco
    llision,sonars[s].Tsafe)
79          print logStr
80          logging.info(logStr)
81
82      logStr = ""
83      print logStr
84      logging.info(logStr)
85
86
87  logStr = "Condition met"
88  print logStr
89  logging.info(logStr)
90
91  sound.beep(440, 200)
92
93
94  #### Step 3: Take control ####
95
96  def changeMode(mode):
97      vehicle.mode = dronekit.VehicleMode(mode)
98
99
100 def checkMode(mode):
101     return vehicle.mode.name==mode
102
103
104 ctrl = Control(takeFun=changeMode, checkTakeFun=checkMode, giveFun=changeMode, chec
    kGiveFun=checkMode,
105             takeArgs="GUIDED", checkTakeArgs="GUIDED", giveArgs="LOITER", checkG
    iveArgs="LOITER")
106
107 logStr = "Taking control"
108 print logStr
109 logging.info(logStr)
110
111 ctrl.take()
112 ctrl.checkTake()
113
114 while not threading.activeCount() <= 3:
115     time.sleep(0.02)
116
```

```python
117   logStr = "Control taken"
118   print logStr
119   logging.info(logStr)
120
121
122   #### Step 4: Autonomous flight ####
123
124   def do_move(distance,tMove,direction=[1,0,0]):
125
126       # def goto_position_target_local_ned(north, east, down):
127       #    """
128       #    Send SET_POSITION_TARGET_LOCAL_NED command to request the vehicle fly to a
   specified
129       #    location in the North, East, Down frame.
130
131       #    It is important to remember that in this frame, positive altitudes are ente
   red as negative
132       #    "Down" values. So if down is "10", this will be 10 metres below the home al
   titude.
133
134       #    At time of writing, acceleration and yaw bits are ignored.
135
136       #    """
137       #    msg = vehicle.message_factory.set_position_target_local_ned_encode(
138       #        0,        # time_boot_ms (not used)
139       #        0, 0,     # target system, target component
140       #        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
141       #        0b0000111111111000, # type_mask (only positions enabled)
142       #        north, east, down, # x, y, z positions (or North, East, Down in the MAV
   _FRAME_BODY_NED frame
143       #        0, 0, 0, # x, y, z velocity in m/s  (not used)
144       #        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS_Mavl
   ink)
145       #        0, 0)     # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)
146       #    # send command to vehicle
147       #    vehicle.send_mavlink(msg)
148
149
150       def body2ned(frontBody,leftBody,upBody=-vehicle.location.global_relative_frame.
   alt):
151
152           yaw=vehicle.attitude.yaw
153           yawCorrected=yaw+40/180/math.pi # Weird offset. Don't know why, but it work
   s
154           north=frontBody*math.cos(yawCorrected)+leftBody*math.sin(yawCorrected)
155           east=frontBody*math.sin(yawCorrected)-leftBody*math.cos(yawCorrected)
156           down=-upBody
157           return [north,east,down]
158
159       def ned2global(original_location, dNorth, dEast, dDown=0):
160           """
161           Returns a LocationGlobal object containing the latitude/longitude `dNorth`
   and `dEast` metres from the
162           specified `original_location`. The returned LocationGlobal has the same `al
   t` value
163           as `original_location`.
164
165           The function is useful when you want to move the vehicle around specifying
   locations relative to
166           the current vehicle position.
167
168           The algorithm is relatively accurate over small distances (10m within 1km)
   except close to the poles.
```

```python
169
170            For more information see:
171            http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-lati
       tude-longitude-by-some-amount-of-meters
172            """
173            earth_radius=6378137.0 #Radius of "spherical" earth
174            #Coordinate offsets in radians
175            dLat = dNorth/earth_radius
176            dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))
177            dAlt = -dDown
178
179            #New position in decimal degrees
180            newlat = original_location.lat + (dLat * 180/math.pi)
181            newlon = original_location.lon + (dLon * 180/math.pi)
182            newalt = original_location.alt + dAlt
183            if type(original_location) is dronekit.LocationGlobal:
184                targetlocation=dronekit.LocationGlobal(newlat, newlon, newalt)
185            elif type(original_location) is dronekit.LocationGlobalRelative:
186                targetlocation=dronekit.LocationGlobalRelative(newlat, newlon, dAlt)
187            else:
188                raise Exception("Invalid Location object passed")
189
190            return targetlocation;
191
192
193        vehicle.simple_goto(ned2global(vehicle.location.global_frame,body2ned(distance*
       direction[0],distance*direction[1],distance*direction[2])[0],body2ned(distance*dire
       ction[0],distance*direction[1],distance*direction[2])[1],body2ned(distance*directio
       n[0],distance*direction[1],distance*direction[2])[2]))
194        # goto_position_target_local_ned(*body2ned(distance*direction[0],distance*direc
       tion[1],distance*direction[2]))
195        print "Moving"
196
197        time.sleep(tMove+1)
198
199
200    def wait(seconds):
201        time.sleep(seconds)
202        return True
203
204
205    autoMove = Auto(do_move, wait, [5,10,[0,0,1]], 10)
206
207
208    print "Starting autonomous flight"
209    autoMove.fly()
210    autoMove.stop()
211
212    while not threading.activeCount() <= 3:
213        time.sleep(0.02)
214    print "Mission finished"
215
216
217    #### Step 5: Return control to the pilot ####
218
219    logStr = "Returning control"
220    print logStr
221    logging.info(logStr)
222
223    # Recovering ctrl class instance that was created in step 3
224    ctrl.give()
225    ctrl.checkGive()
226
```

```python
227   while not threading.activeCount() <= 3:
228       time.sleep(0.02)
229
230   logStr = "Control returned"
231   print logStr
232   logging.info(logStr)
233
234   sound.tripleBeep(700, 150, 600, 150, 500, 300)
235
236   logStr = "\nTerminating script\n"
237   print logStr
238   logging.info(logStr)
239   vehicle.close()
240
241
```