Appendix “A”

```
clear all;
close all;

% Measurement data
% 1045 propeller
% RobbeRoxxy Motor (1100 kV, data collected in 2010)
data = [ 45, 7.4;...
         38, 5.6;...
         33, 4.3;...
         26, 3.0;...
         18, 2.0;...
         10, 1.0 ];

% Normalize the data, as we're operating later
% anyways in normalized units
data(:,1) = data(:,1) ./ max(data(:,1));
data(:,2) = data(:,2) ./ max(data(:,2));

% Fit a 2nd degree polygon to the data and
% print the x2, x1, x0 coefficients
p = polyfit(data(:,2), data(:,1),2)

% Override the first coffefficient for testing
% purposes
pf = 0.62;

% Generate plotting data
px1 = linspace(0, max(data(:,2)));
py1 = polyval(p, px1);

pyt = zeros(size(data, 1), 1);
corr = zeros(size(data, 1), 1);

% Actual code test
% the two lines below are the ones needed to be ported to
C:
%   pf: Power factor parameter.
%   px1(i): The current normalized motor command (-1..1)
%   corr(i): The required correction. The motor speed is:
%            px1(i)
fori=1:size(px1, 2)

% The actual output throttle
pyt(i) = -pf * (px1(i) * px1(i)) + (1 + pf) * px1(i);

% Solve for input throttle
% y = -p * x^2 + (1+p) * x;
```

```
%
end

plot(data(:,2), data(:,1), '*r');
hold on;
plot(px1, py1, '*b');
hold on;
plot([0 px1(end)], [0 py1(end)], '-k');
hold on;
plot(px1, pyt, '-b');
hold on;
plot(px1, corr, '-m');
```

Appendix”B”

```
close all;
clear all;
M = importdata('px4io_v1.3.csv');
voltage = M.data(:, 1);
counts = M.data(:, 2);
plot(counts, voltage, 'b*-', 'LineWidth', 2, 'MarkerSize', 15);
coeffs = polyfit(counts, voltage, 1);
fittedC = linspace(min(counts), max(counts), 500);
fittedV = polyval(coeffs, fittedC);
hold on
plot(fittedC, fittedV, 'r-', 'LineWidth', 3);

slope = coeffs(1)
y_intersection = coeffs(2)
```

Appendix”C”

```
#!/usr/bin/env python
"""
Autonomous drone Mission control.
University of sudan for science and technology 2017.
"""
import time
from dronekit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal, Command
import math
import pymavlink
from pymavlink import mavutil
import getch
from RPIO import PWM
import sys
defarm_and_takeoff(vehicle,aTargetAltitude):
   """
   Arms vehicle and fly to a Target Altitude.
   """
   print "Arming motors"
   # Copter should arm in GUIDED mode
vehicle.mode = VehicleMode("GUIDED")
```

```
vehicle.armed = True
vehicle.flush()

   # Confirm vehicle armed before attempting to take off
   while not vehicle.armed:
      print "Waiting for arming..."
vehicle.mode = VehicleMode("GUIDED")
         vehicle.armed = True
         vehicle.flush()
time.sleep(1)

   print "Taking off!"
vehicle.simple_takeoff(aTargetAltitude) # Take off to target altitude
vehicle.flush()
   # Wait until the vehicle reaches a safe height before processing the goto
(otherwise the command
   #  after Vehicle.simple_takeoff will execute immediately).
   while True:
      print " Altitude: ", vehicle.location.global_relative_frame.alt
      #Break and return from function just below target altitude.
      if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
         print "Reached target altitude"
```

```
        break
time.sleep(1)


defshows_data(vehicle):
    """
    show vehicle data.
    """
    print "Get some vehicle attribute values:"
    print "Autopilot Firmware version: %s" % vehicle.version
    print "Autopilot capabilities (supports ftp): %s" % vehicle.capabilities.ftp
    print "Global Location: %s" % vehicle.location.global_frame
    print "Global Location (relative altitude): %s" %
vehicle.location.global_relative_frame
    print "Local Location: %s" % vehicle.location.local_frame
    print "Heading: %s" % vehicle.heading
    print "attitude: %s" % vehicle.attitude
    print "velocity: %s" % vehicle.velocity
    print "channels: %s" % vehicle.channels
    print "Altitude (global frame): %s" % vehicle.location.global_frame.alt
    print "Altitude (global relative frame): %s" %
vehicle.location.global_relative_frame.alt
    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 "armed: %s" % vehicle.armed
    print "Mode: %s" % vehicle.mode.name
    print "groundspeed: %s" % vehicle.groundspeed
    print "airspeed: %s" % vehicle.airspeed
time.sleep(0.01)


defmanual_control(vehicle):
    """
    Function that makes the vehicle be controlled with keyboard.
    """
arm_and_takeoff(vehicle,2)
    #changing vehicle mode to stabilize
    print "\nSet Vehicle mode = STABILIZE (currently: %s)" %
vehicle.mode.name
    while not vehicle.mode=='STABILIZE':
        print 'waiting to change Mode to STABILIZE...'
vehicle.mode = VehicleMode('STABILIZE')
vehicle.flush()
```

```
    print "vehicle mode: %s" % vehicle.mode
    # initialize servo objects with PWM function
    roll = PWM.Servo()
    pitch = PWM.Servo()
    throttle = PWM.Servo()
    yaw = PWM.Servo()
    # start PWM on servo specific GPIO no, this is not the pin no but it is the
GPIO no
roll.set_servo(17,1520)# pin 11
pitch.set_servo(18,1520)# pin 12
throttle.set_servo(27,1100)# pin 13, pin 14 is Ground
yaw.set_servo(22,1520)# pin 15
    # assign global min and max values
th_min = 1100
th_max = 2000
th =1100
    print "control drone from keyboard"
    try:
        while True:
            # waiting for key strokes
            key = getch.getch()
            if key == 'w':
```

```
th = th + 10
        if (th<th_min):
th = 1100
throttle.set_servo(27,th)
time.sleep(0.3)
elif (th>th_max):
th = 2000
throttle.set_servo(27,th)
time.sleep(0.3)
        else:
throttle.set_servo(27,th)
time.sleep(0.3)
        print 'th :' + str(th)
elif key == 's':
th = th - 10
        if (th<th_min):
th = 1100
throttle.set_servo(27,th)
time.sleep(0.3)
elif (th>th_max):
th = 2000
throttle.set_servo(27,th)
```

```
time.sleep(0.3)
            else:
throttle.set_servo(27,th)
time.sleep(0.3)
            print 'th :' + str(th)
        #yaw left
elif key == 'a':
yaw.set_servo(22,1350)
            print "yaw left"
time.sleep(0.3)
yaw.set_servo(22,1500)
    #yaw right
elif key == 'd':
yaw.set_servo(22,1650)
            print "yaw right"
time.sleep(0.3)
yaw.set_servo(22,1500)
    #roll left
elif key == '4':
roll.set_servo(17,1350)
            print "roll left"
time.sleep(0.3)
```

```
roll.set_servo(17,1500)
        #roll right
elif key == '6':
roll.set_servo(17,1650)
                print "roll right"
time.sleep(0.3)
roll.set_servo(17,1500)
        #pitch forward
elif key == '8':
pitch.set_servo(18,1650)
                print "pitch forward"
time.sleep(0.3)
pitch.set_servo(18,1500)
        #pitch back
elif key == '2':
pitch.set_servo(18,1350)
                print "pitch back"
time.sleep(0.3)
pitch.set_servo(18,1500)
        #atlitude hold
elif key == 'h':
vehicle.mode = VehicleMode("ALT_HOLD")
```

```
time.sleep(0.5)
            print " mode is %s" % vehicle.mode.name
        #land mode
elif key == 'l':
vehicle.mode = VehicleMode("LAND")
time.sleep(5)
            print " mode is %s " % vehicle.mode.name
        #stabilize mode
elif key =='5' and th<1200:
vehicle.mode = VehicleMode("STABILIZE")
time.sleep(0.5)
            print "mode is %s" % vehicle.mode.name
elif key == 'q':
time.sleep(0.1)
            break
          else:
            print "wrong input...."
time.sleep(0.1)

   except KeyboardInterrupt:
vehicle.mode = VehicleMode("LAND")
time.sleep(5)
```

```
        print "keyboard interrupt- Landed"
    finally:
roll.stop_servo(17)
pitch.stop_servo(18)
throttle.stop_servo(27)
yaw.stop_servo(22)


defsend_ned_velocity(vehicle, velocity_x, velocity_y, velocity_z, duration):
    """
    Move vehicle in 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)
    # send command to vehicle on 1 Hz cycle
```

```
    for x in range(0,duration):
vehicle.send_mavlink(msg)
def diamond(vehicle):
    """
    Move vehicle in direction based on diamond points.
    """
    NORTH=2
    SOUTH=-2
    EAST=2
    WEST=-2
    UP=-0.5
    DOWN=0.5
    DURATION=10
    # Shape shape
    print "Making a diamond!"
condition_yaw(0)
send_ned_velocity(vehicle,NORTH,0,0,DURATION)
    print "Flying for 20 seconds direction NORTH!"
    #send_ned_velocity(vehicle,0,0,0,5)
condition_yaw(90)
send_ned_velocity(vehicle,0,EAST,0,DURATION)
    print "Flying for 20 seconds direction EAST!"
```

```
    #send_ned_velocity(vehicle,0,0,0,5)
condition_yaw(180)
send_ned_velocity(vehicle,SOUTH,0,0,DURATION)
    print "Flying for 20 seconds direction SOUTH!"
    #send_ned_velocity(vehicle,0,0,0,5)
condition_yaw(270)
send_ned_velocity(vehicle,0,WEST,0,DURATION)
    print "Flying for 20 seconds direction WEST!"
    #send_ned_velocity(vehicle,0,0,0,5)
    print("Going North, East and up")
condition_yaw(90)
send_ned_velocity(vehicle,NORTH,EAST,UP,DURATION)
    print("Going South, East and down")
condition_yaw(90)
send_ned_velocity(vehicle,SOUTH,EAST,DOWN,DURATION)
    print("Going South and West")
condition_yaw(90)
send_ned_velocity(vehicle,SOUTH,WEST,0,DURATION)
    print("Going North and West")
condition_yaw(90)
send_ned_velocity(vehicle,NORTH,WEST,0,DURATION)
    print "Returning to Launch"
```

```
vehicle.mode    = VehicleMode("RTL")
   print "Waiting 10 seconds RTL"
time.sleep(10)
   print "Landing the Aircraft"
vehicle.mode    = VehicleMode("LAND")
defbuild_loop_mission(vehicle, loop_center, loop_radius, altitude):
   """
   Adds a takeoff command and 12 waypoint commands to the current mission.
   The waypoints are positioned to form a dodecagon with vertices at loop_radius around the specified
LocationGlobal (loop_center).
   The function assumes vehicle.commands matches the vehicle mission state
   (you must have called download at least once in the session and after clearing the mission)
   Modified from Dronekit-python
   """
cmds = vehicle.commands
   print " Clearing any existing commands"
cmds.clear()
   print " Building loop waypoints."
```

```
    # Add new commands. The meaning/order of the parameters is
documented in the Command class.

    # Add MAV_CMD_NAV_TAKEOFF command. This is ignored if the vehicle
is already in the air.

cmds.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF,

                0, 0, 0, 0, 0, 0, 0, 0, 10))

    # Define the twelve MAV_CMD_NAV_WAYPOINT locations and add the
commands

    for n in range(0, 11, 1):

d_north = math.sin(math.radians(n*30))*loop_radius

d_east = math.cos(math.radians(n*30))*loop_radius

       point = get_location_metres(loop_center, d_north, d_east)

cmds.add(Command(0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,

                    0, 0, 0, 0, 0, 0, point.lat, point.lon, altitude))

    print " Upload new commands to vehicle"

cmds.upload()

defget_location_metres(original_location, dNorth, dEast):

    """

    Returns a LocationGlobal object containing the latitude/longitude
`dNorth` and `dEast` metres from the
```

specified `original_location`. The returned Location has the same `alt` value

as `original_location`.

The function is useful when you want to move the vehicle around specifying locations relative to

the current vehicle position.

The algorithm is relatively accurate over small distances (10m within 1km) except close to the poles.

"""

earth_radius=6378137.0 #Radius of "spherical" earth

#Coordinate offsets in radians

dLat = dNorth/earth_radius

dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

#New position in decimal degrees

newlat = original_location.lat + (dLat * 180/math.pi)

newlon = original_location.lon + (dLon * 180/math.pi)

return LocationGlobal(newlat, newlon,original_location.alt)

defgetSSMeters(aLocation, alpha, dLat, dLon, turn):

"""

Returns a LocationGlobal object containing the latitude/longitude values of the next position in the sector search

"""

earth_radius=6378137.0 #Radius of "spherical" earth

```
    if turn == 0:
        #Coordinate offsets in radians
        bearing = math.radians(alpha - 180 + 60)
        if bearing >=360:
            bearing += 360
Lat = (dLat * math.sin(bearing)) / earth_radius
        Lon = (dLat * math.cos(bearing)) / earth_radius
    #
elif turn == 1:
        #Coordinate offsets in radians
        bearing = math.radians(alpha + 180 - 60)
        if bearing >=360:
            bearing += 360
Lat = (dLat * math.sin(bearing)) / earth_radius
        Lon = (dLat * math.cos(bearing)) / earth_radius
    #New position in decimal degrees
newlat = aLocation.lat + (Lat * 180/math.pi)
newlon = aLocation.lon + (Lon * 180/math.pi)
    return LocationGlobal(newlat, newlon,aLocation.alt)
defcondition_yaw(heading, relative=False):
    if relative:
is_relative=1 #yaw relative to direction of travel
```

```
    else:
is_relative=0 #yaw is an absolute angle
    # create the CONDITION_YAW command using command_long_encode()
msg = vehicle.message_factory.command_long_encode(
        0, 0,    # target system, target component
mavutil.mavlink.MAV_CMD_CONDITION_YAW, #command
        0, #confirmation
        heading,    # param 1, yaw in degrees
        0,          # param 2, yaw speed deg/s
        1,          # param 3, direction -1 ccw, 1 cw
is_relative, # param 4, relative offset 1, absolute angle 0
        0, 0, 0)    # param 5 ~ 7 not used
    # send command to vehicle
vehicle.send_mavlink(msg)
defget_bearing(aLocation1, aLocation2):
    """
    Returns the bearing between the two LocationGlobal objects passed as parameters.
    This method is an approximation, and may not be accurate over large distances and close to the
    earth's poles.
    """
```

```
off_x = aLocation2.lon - aLocation1.lon
off_y = aLocation2.lat - aLocation1.lat
    bearing = math.degrees(math.atan2(off_y, off_x))
    if bearing < 0:
        bearing += 360.00
    return bearing;
defget_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.
    """
dlat = aLocation2.lat - aLocation1.lat
dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
defdownload_mission(vehicle):
    """
    Download the current mission from the vehicle.
    """
cmds = vehicle.commands
```

```
cmds.download()

cmds.wait_ready() # wait until download is complete.

defclear_mission(vehicle):
    """
    Clear the current mission.
    """
cmds = vehicle.commands

vehicle.commands.clear()

vehicle.flush()

download_mission(vehicle)

defadds_square_mission(vehicle,aLocation, aSize):
    """
    Adds a takeoff command and four waypoint commands to the current mission.
    The waypoints are positioned to form a square of side length 2*aSize around the specified LocationGlobal (aLocation).
    The function assumes vehicle.commands matches the vehicle mission state
    (you must have called download at least once in the session and after clearing the mission)
    """
cmds = vehicle.commands

    print " Clear any existing commands"
```

```
cmds.clear()

    print " Define/add new commands."

    # Add new commands. The meaning/order of the parameters is
documented in the Command class.

    #Add MAV_CMD_NAV_TAKEOFF command. This is ignored if the vehicle
is already in the air.

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, 10))

    #Define the four MAV_CMD_NAV_WAYPOINT locations and add the
commands

    point1 = get_location_metres(aLocation, aSize, -aSize)

    point2 = get_location_metres(aLocation, aSize, aSize)

    point3 = get_location_metres(aLocation, -aSize, aSize)

    point4 = get_location_metres(aLocation, -aSize, -aSize)

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, point1.lat,
point1.lon, 11))

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, point2.lat,
point2.lon, 12))

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
```

```
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, point3.lat,
point3.lon, 13))

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, point4.lat,
point4.lon, 14))

   #add dummy waypoint "5" at point 4 (lets us know when have reached
destination)

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, point4.lat,
point4.lon, 14))

   print " Upload new commands to vehicle"

cmds.upload()

####################

# SEARCH PATTERNS #

####################

defaddsParallelTrack(Area, initPoint, alt):

   """

   Adds mission to perform Parallel Track across specified area.

   """

cmds = vehicle.commands

   print " Clear any existing commands"

cmds.clear()
```

```
    print " Define/add new commands."

    # Add new commands. The meaning/order of the parameters is
documented in the Command class.

    #Calculate track properties

trackLength = 2 * math.sqrt(dFSA/math.pi)

legConst = 5                                    #Arbitrary ratio of leg length to
track length

legLength = legConst * trackLength                      #Leg length function of
dFSA radius

numLegs = Area / (trackLength  * legLength)

    #Add MAV_CMD_NAV_TAKEOFF command. This is ignored if the vehicle
is already in the air.

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, alt))

    #Go to initial point as specified by user

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, initPoint.lat,
initPoint.lon, alt))

    #Define waypoint pattern - point(lat, long, alt)

i = 1;

    waypoint = initPoint

    while i<= numLegs :

        # Strafe
```

```
        waypoint = get_location_metres(waypoint, 0, (legLength*(-1)**i))
cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))
        # Advance
        waypoint = get_location_metres(waypoint, trackLength, 0)
cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))
i += 1
    # Return to Launch
cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, initPoint.lat,
initPoint.lon, alt))
    print " Upload new commands to vehicle"
cmds.upload()
defaddsSectorSearch(Area, initPoint, alt):
    """
    Adds mission to perform sector search across specified area.
        """
cmds = vehicle.commands
    print " Clear any existing commands"
```

```
cmds.clear()

    print " Define/add new commands."

    # Add new commands. The meaning/order of the parameters is
documented in the Command class.

    #Add MAV_CMD_NAV_TAKEOFF command. This is ignored if the vehicle
is already in the air.

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, alt))

    #Define waypoint pattern - point(lat, long, alt)

    #Initial waypoint

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, initPoint.lat,
initPoint.lon, alt))

currLocation = initPoint

    #Initial direction

    alpha = 90

    #Area radius

    Radius = math.sqrt(Area/math.pi)

    # 1st waypoint

    waypoint = get_location_metres(currLocation, Radius*math.cos(alpha),
Radius*math.sin(alpha))

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
```

```
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))

   alpha = get_bearing(currLocation, waypoint)

currLocation = waypoint

   tri = 1

   # Sector Search

   while tri <= 3 :

triCnr = 1

      while triCnr< 3:

         waypoint = getSSMeters(currLocation, alpha, Radius, 1)

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))

         alpha = get_bearing(currLocation, waypoint)

currLocation = waypoint

triCnr += 1

      tri += 1

      if tri != 4:

         waypoint = getSSMeters(currLocation, alpha,Radius , 0)

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))
```

```
        alpha = get_bearing(currLocation, waypoint)
currLocation = waypoint
   # Return to Launch
cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, initPoint.lat,
initPoint.lon, alt))
   print " Upload new commands to vehicle"
cmds.upload()
defaddsExpandSquare(initPoint, Area, alt):
   """
    Adds mission to expanding square search search across specified area.
      """
cmds = vehicle.commands
   print " Clear any existing commands"
cmds.clear()
   print " Define/add new commands."
   # Add new commands. The meaning/order of the parameters is
documented in the Command class.
   #Determine number of loops - Square search area
   Radius = math.sqrt(dFSA / math.pi)
numLoops = math.sqrt(Area) /  Radius
```

```
    #Add MAV_CMD_NAV_TAKEOFF command. This is ignored if the vehicle
is already in the air.

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_TAKEOFF, 0, 0, 0, 0, 0, 0, 0, 0, alt))

    #Initial Waypoint - Centre of search area

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, initPoint.lat,
initPoint.lon, alt))

    #Define waypoint pattern - point(lat, long, alt)

i = 1;

dist = Radius

advanceToggle = 1

strafeToggle = 1

    waypoint = initPoint

    while i<= numLoops :

        if i % 2 == 0:

        # Strafe

            waypoint = get_location_metres(waypoint, 0, (dist*(-
1)**strafeToggle))

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))
```

```
strafeToggle ^= 1

dist += Radius

      else:

      # Advance

         waypoint = get_location_metres(waypoint, (dist*(-
1)**advanceToggle), 0)

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, waypoint.lat,
waypoint.lon, alt))

advanceToggle ^= 1

i += 1

   # Return to Launch

cmds.add(Command( 0, 0, 0,
mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, 0, 0, 0, initPoint.lat,
initPoint.lon, alt))

   print " Upload new commands to vehicle"

cmds.upload()

############################################

#

# CONNECTION

#

############################################
```

```
# Connection to the vehicle
print "Connecting to drone...."
vehicle = connect('/dev/ttyAMA0', baud = 57600, wait_ready= True)
print "Connected..."
clear_mission(vehicle)
print "mission cleared..."
print "\n\nMission start...!!\n\n"
while True:
        print "Basic pre-arm checks"
   # Don't let the user try to fly autopilot is booting
   if vehicle.mode.name == "INITIALISING":
      print "Waiting for vehicle to initialise"
time.sleep(1)
   while vehicle.gps_0.fix_type < 2:
      print "Waiting for GPS...:", vehicle.gps_0.fix_type
time.sleep(1)


   # wait for user input
   print "Commands:\n 'takeoff' = arm and takeoff for 2 meter \n 'fly
manual' = controll vehicle with keyboard  \n 'fly auto' = arm and takeoff for
2 meter and change mode to AUTO  \n 'fly square' = fly to form a square  \n
'fly diamond' = fly to form a diamond  \n 'fly dodecagon' = fly to form a
dodecagon  \n 'show' = show vehicle data  \n 'search' = mission to search
```

```
across specified area \n 'land' = change mode to LAND!  \n 'loiter' = change mode to LOITER!  \n 'guided' = change mode to GUIDED!  \n 'circle' = change mode to CIRCLE!  \n 'althold' = change mode to ALT_HOLD!  \n 'stabilize' = change mode to STABILIZE!  \n 'acro' = change mode to ACRO!  \n 'sport' = change mode to SPORT!  \n 'poshold' = change mode to POSHOLD!  \n 'simple' = change mode to SIMPLE!  \n 'supersimple' = change mode to SUPER_SIMPLE!  \n 'return' = change mode to RTL!  \n 'auto' = change mode to AUTO!  \n  "

   string = raw_input ('Enter Command: ')

   word = string.split()

   word1 = word[0]

            # take off the drone for a scpecific Altitude

   if word1 == 'takeoff':

        arm_and_takeoff(vehicle,2)

time.sleep(1)


elif word1 == 'fly':

      # mapping for MANUAL MODE

      if word[1] == 'manual':

         print 'Flight control: MANUAL'

manual_control(vehicle)

         # mapping for AUTO MODE

elif word[1] == 'auto':

arm_and_takeoff(vehicle,2)
```

```
time.sleep(10)
vehicle.mode = VehicleMode("AUTO")
        # fly to shape a square
elif word[1] == 'square':
            print 'Flight control: square'
            size = 4
add_square_mission(vehicle,vehicle.location.global_frame,size)
time.sleep(2)
arm_and_takeoff(vehicle,2)
            print "Starting mission"
vehicle.commands.next=0
            # Set mode to AUTO to start mission
vehicle.mode = VehicleMode("AUTO")
vehicle.flush()


        # fly to shape a diamond
elif word[1] == 'diamond':
            print 'Flight control: diamond'
            diamond(vehicle)
time.sleep(2)
arm_and_takeoff(vehicle,2)
            print "Starting mission"
```

```
vehicle.commands.next=0
            # Set mode to AUTO to start mission
vehicle.mode = VehicleMode("AUTO")
vehicle.flush()

        # fly to shape a dodecagon
elif word[1] == 'dodecagon':
            print 'Flight control: dodecagon'
loopcenter = vehicle.location.global_frame
build_loop_mission(vehicle,loopcenter,4,2)
time.sleep(2)
arm_and_takeoff(vehicle,2)
            print "Starting mission"
vehicle.commands.next=0
            # Set mode to AUTO to start mission
vehicle.mode = VehicleMode("AUTO")
vehicle.flush()

      else:
arm_and_takeoff(vehicle,2)
time.sleep(1)
elif word1 == 'show':
```

```
shows_data(vehicle)


elif word1 == 'search':
    Pattern = raw_input("Enter 'SS' = Sector Search \n 'PT' = Parallel Track \n 'ES' = Expanding Square\nSpecify desired search pattern: ")
Lat = raw_input("Enter Latitude: ")
    Lon = raw_input("Enter Longitude: ")
    Area = raw_input("Enter Specify search area (m2): ")
    Alt = raw_input("Enter Specify search altitude (m): ")
    point = LocationGlobal(float(Lat), float(Lon), float(Alt))
    if Pattern == 'SS':
      addSectorSearch(float(Area), point, float(Alt))
elif Pattern == 'PT':
addsParallelTrack(float(Area), point, float(Alt))
elif Pattern == 'ES':
addsExpandSquare(point, float(Area), float(Alt))
arm_and_takeoff(vehicle, 2)
vehicle.commands.next=0
vehicle.mode = VehicleMode("AUTO")
vehicle.flush()
    print 'Drone is armed'
elif word1 == 'land':
```

```
        print "\n\nLanding!\n\n"
        #changing vehicle mode to LAND
            print "\nSet Vehicle mode = LAND (currently: %s)" %
vehicle.mode.name
            while not vehicle.mode=='LAND':
                    print 'waiting to change Mode to LAND...'
            vehicle.mode = VehicleMode('LAND')
            vehicle.flush()
            print "vehicle mode: %s" % vehicle.mode
time.sleep(5)
        print 'The drone has landed!'
vehicle.flush()
elif word1 == 'loiter':
        print "\n\nsetting mode to loiter!\n\n"
        #changing vehicle mode to LOITER
            print "\nSet Vehicle mode = LOITER (currently: %s)" %
vehicle.mode.name
            while not vehicle.mode=='LOITER':
                    print 'waiting to change Mode to LOITER...'
            vehicle.mode = VehicleMode('LOITER')
            vehicle.flush()
            print "vehicle mode: %s" % vehicle.mode
```

```
time.sleep(1)
        print 'loiter mode!'
vehicle.flush()
elif word1 == 'auto':
        print "\n\nsetting mode to AUTO!\n\n"
        #changing vehicle mode to AUTO
          print "\nSet Vehicle mode = AUTO (currently: %s)" %
vehicle.mode.name
          while not vehicle.mode=='AUTO':
                  print 'waiting to change Mode to AUTO...'
          vehicle.mode = VehicleMode('AUTO')
          vehicle.flush()
          print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
        print 'AUTO mode!'
vehicle.flush()
elif word1 == 'guided':
        print "\n\nsetting mode to GUIDED!\n\n"
        #changing vehicle mode to GUIDED
          print "\nSet Vehicle mode = GUIDED (currently: %s)" %
vehicle.mode.name
          while not vehicle.mode=='GUIDED':
```

```
                print 'waiting to change Mode to GUIDED...'
        vehicle.mode = VehicleMode('GUIDED')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
    print 'GUIDED mode!'
vehicle.flush()
elif word1 == 'circle':
    print "\n\nsetting mode to CIRCLE!\n\n"
    #changing vehicle mode to CIRCLE
        print "\nSet Vehicle mode = CIRCLE (currently: %s)" %
vehicle.mode.name
        while not vehicle.mode=='CIRCLE':
                print 'waiting to change Mode to CIRCLE...'
        vehicle.mode = VehicleMode('CIRCLE')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
    print 'CIRCLE mode!'
vehicle.flush()
elif word1 == 'althold':
    print "\n\nsetting mode to ALT HOLD!\n\n"
```

```
    #changing vehicle mode to ALT_HOLD
        print "\nSet Vehicle mode = ALT_HOLD (currently: %s)" %
vehicle.mode.name
        while not vehicle.mode=='ALT_HOLD':
            print 'waiting to change Mode to ALT_HOLD...'
        vehicle.mode = VehicleMode('ALT_HOLD')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
    print 'ALT HOLD mode!'
vehicle.flush()
elif word1 == 'stabilize':
    print "\n\nsetting mode to STABILIZE!\n\n"
    #changing vehicle mode to stabilize
        print "\nSet Vehicle mode = STABILIZE (currently: %s)" %
vehicle.mode.name
        while not vehicle.mode=='STABILIZE':
            print 'waiting to change Mode to STABILIZE...'
        vehicle.mode = VehicleMode('STABILIZE')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
vehicle.mode = VehicleMode("STABILIZE")
```

```
time.sleep(1)
        print 'STABILIZE mode!'
vehicle.flush()
elif word1 == 'acro':
        print "\n\nsetting mode to ACRO!\n\n"
        #changing vehicle mode to ACRO
          print "\nSet Vehicle mode = ACRO (currently: %s)" %
vehicle.mode.name
          while not vehicle.mode=='ACRO':
                  print 'waiting to change Mode to ACRO...'
          vehicle.mode = VehicleMode('ACRO')
          vehicle.flush()
          print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
        print 'ACRO mode!'
vehicle.flush()
elif word1 == 'sport':
        print "\n\nsetting mode to SPORT!\n\n"
        #changing vehicle mode to SPORT
          print "\nSet Vehicle mode = SPORT (currently: %s)" %
vehicle.mode.name
          while not vehicle.mode=='SPORT':
```

```
                print 'waiting to change Mode to SPORT...'
        vehicle.mode = VehicleMode('SPORT')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
    print 'SPORT mode!'
vehicle.flush()
elif word1 == 'poshold':
    print "\n\nsetting mode to POSHOLD!\n\n"
    #changing vehicle mode to POSHOLD
        print "\nSet Vehicle mode = POSHOLD (currently: %s)" %
vehicle.mode.name
        while not vehicle.mode=='POSHOLD':
                print 'waiting to change Mode to POSHOLD...'
        vehicle.mode = VehicleMode('POSHOLD')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
    print 'POSHOLD mode!'
vehicle.flush()
elif word1 == 'simple':
    print "\n\nsetting mode to SIMPLE!\n\n"
```

```
    #changing vehicle mode to stabilize
        print "\nSet Vehicle mode = SIMPLE (currently: %s)" %
vehicle.mode.name
        while not vehicle.mode=='SIMPLE':
            print 'waiting to change Mode to SIMPLE...'
        vehicle.mode = VehicleMode('SIMPLE')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
    print 'SIMPLE mode!'
vehicle.flush()
elif word1 == 'supersimple':
    print "\n\nsetting mode to SUPER SIMPLE!\n\n"
    #changing vehicle mode to SUPER_SIMPLE
        print "\nSet Vehicle mode = SUPER_SIMPLE (currently: %s)" %
vehicle.mode.name
        while not vehicle.mode=='SUPER_SIMPLE':
            print 'waiting to change Mode to SUPER_SIMPLE...'
        vehicle.mode = VehicleMode('SUPER_SIMPLE')
        vehicle.flush()
        print "vehicle mode: %s" % vehicle.mode
time.sleep(1)
```

```
        print 'SUPER SIMPLE mode!'
vehicle.flush()
elif word1 == 'return':
        print "\n\nReturning to Launch!\n\n"
        #changing vehicle mode to Return To Launch
          print "\nSet Vehicle mode = RTL (currently: %s)" %
vehicle.mode.name
          while not vehicle.mode=='RTL':
                  print 'waiting to change Mode to RTL...'
          vehicle.mode = VehicleMode('RTL')
          vehicle.flush()
          print "vehicle mode: %s" % vehicle.mode
time.sleep(10)
        print 'The drone has returned!'
vehicle.flush()
   else:
        print 'Wrong Input....'
time.sleep(0.1)

while vehicle.armed:
          vehicle.armed = False
          print "disarming..."
```

```
        time.sleep(1)
        vehicle.flush()
print "DISARMED"
print "closing vechicle object..."
vehicle.close()
print "\n\nMission complete\n\n"
```