Drone Applications, Components and Assembly

Lab 6

Meher Shrishti Nigam

20BRS1193

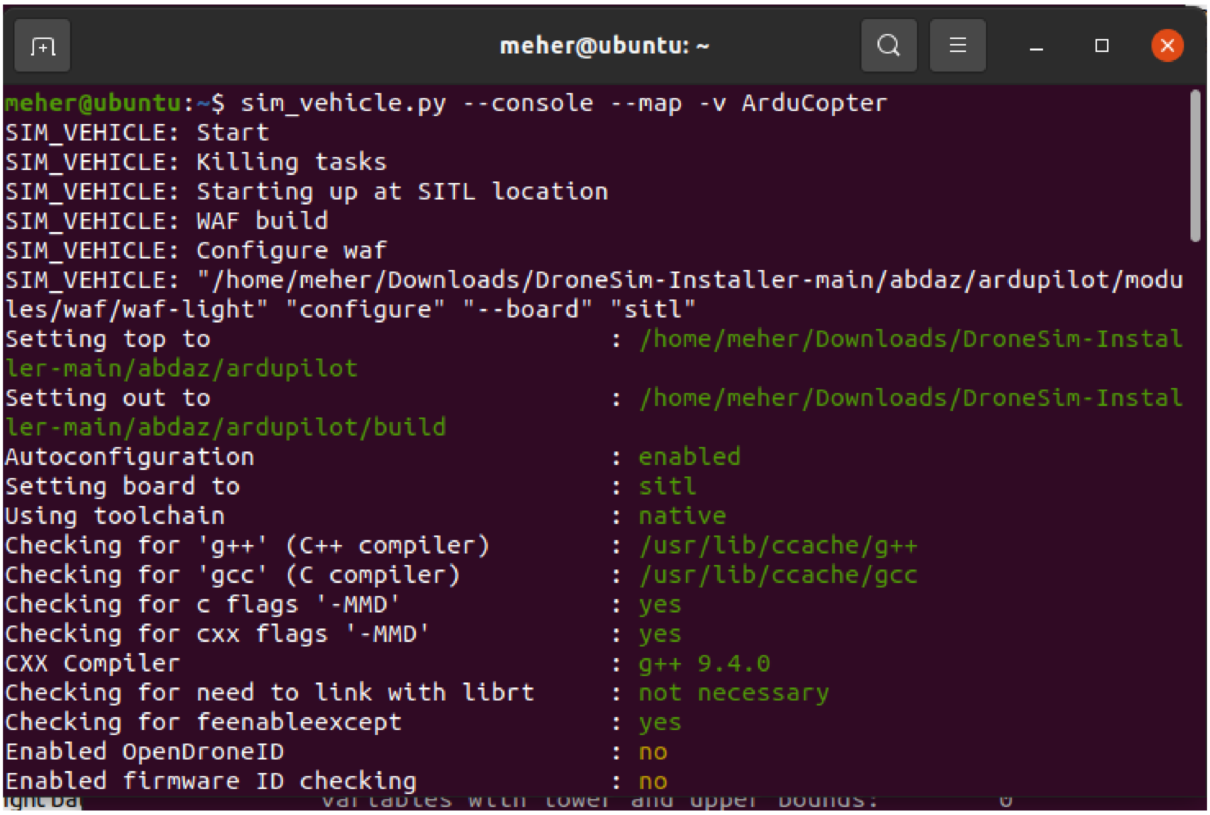
Aim:

To demonstrate RTL (Return to Launch) capabilities in drones, Model Predictive Control (MPC) utilizing the CasADi library, and establishing a different height for each drone waypoint.

Procedure:

1. Install required SITL programs – ArduPilot-SITL
2. Install mavproxy to connect with SITL
3. Install the dronekit Python package
4. In a terminal, run the SITL startup command for a copter and in another terminal, run mavproxy:
   python sim_vehicle.py --map --console -v ArduCopter

   mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 127.0.0.1:14550 --out 127.0.0.1:14551



```
meher@ubuntu: ~
meher@ubuntu:~$ sim_vehicle.py --console --map -v ArduCopter
SIM_VEHICLE: Start
SIM_VEHICLE: Killing tasks
SIM_VEHICLE: Starting up at SITL location
SIM_VEHICLE: WAF build
SIM_VEHICLE: Configure waf
SIM_VEHICLE: "/home/meher/Downloads/DroneSim-Installer-main/abdaz/ardupilot/modules/waf/waf-light" "configure" "--board" "sitl"
Setting top to                           : /home/meher/Downloads/DroneSim-Installer-main/abdaz/ardupilot
Setting out to                           : /home/meher/Downloads/DroneSim-Installer-main/abdaz/ardupilot/build
Autoconfiguration                        : enabled
Setting board to                         : sitl
Using toolchain                          : native
Checking for 'g++' (C++ compiler)        : /usr/lib/ccache/g++
Checking for 'gcc' (C compiler)          : /usr/lib/ccache/gcc
Checking for c flags '-MMD'              : yes
Checking for cxx flags '-MMD'            : yes
CXX Compiler                             : g++ 9.4.0
Checking for need to link with librt     : not necessary
Checking for feenableexcept              : yes
Enabled OpenDroneID                      : no
Enabled firmware ID checking             : no
```

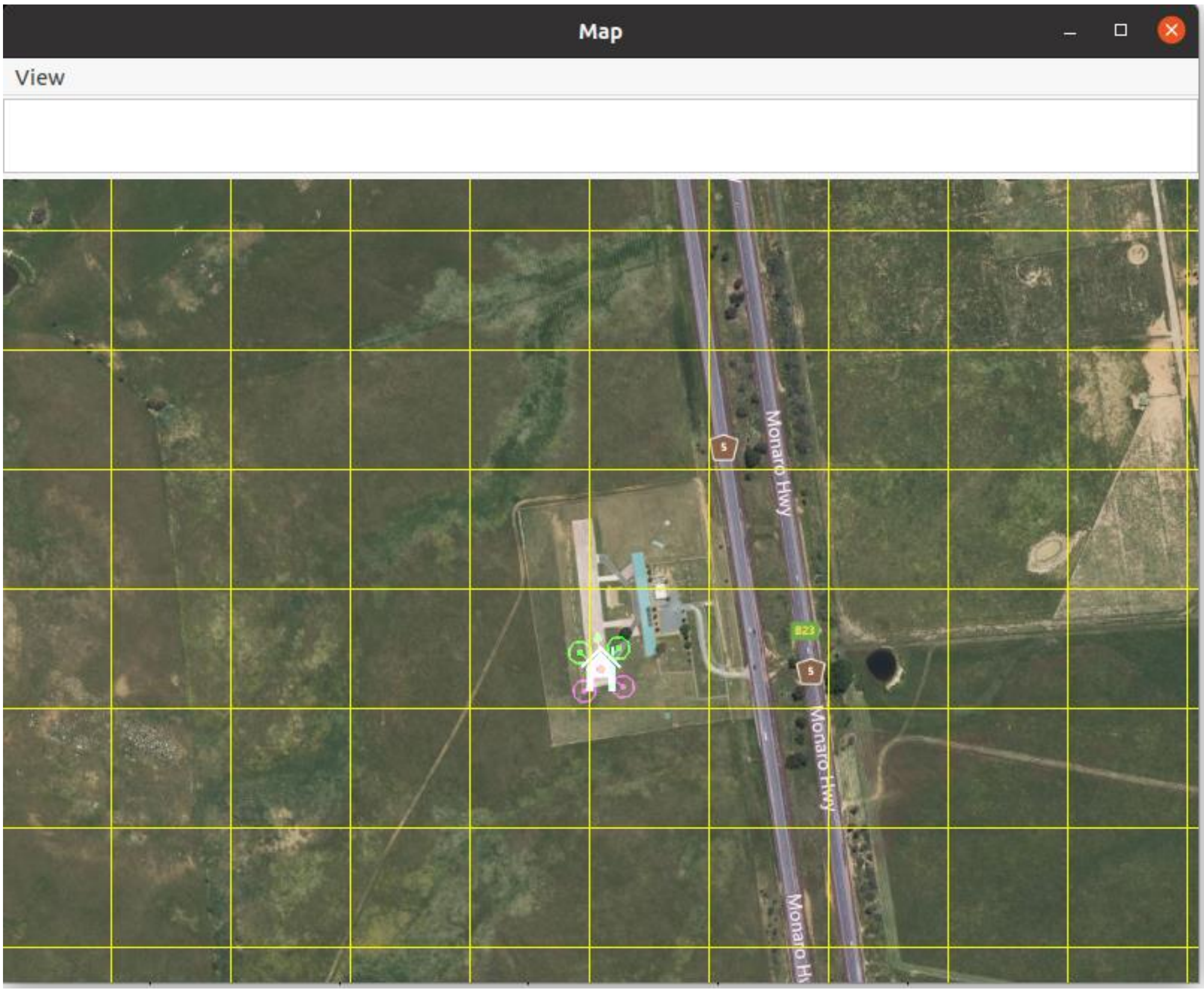
meher@ubuntu: ~

```
meher@ubuntu:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --
out 127.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> link 1 down
```

ArduCopter

```
Setting SIM_SPEEDUP=1.000000
Suggested EK3_DRAG_BCOEF_* = 16.288, EK3_DRAG_MCOEF = 0.209
Starting sketch 'ArduCopter'
Starting SITL input
Using Irlock at port : 9005
bind port 5760 for 0
Serial port 0 on TCP port 5760
Waiting for connection ....
Connection on serial port 5760
Loaded defaults from Downloads/DroneSim-Installer-main/abdaz/ardupilot/Tools/aut
otest/default_params/copter.parm
bind port 5762 for 2
Serial port 2 on TCP port 5762
bind port 5763 for 3
Serial port 3 on TCP port 5763
Home: -35.363262 149.165237 alt=584.000000m hdg=353.000000
Smoothing reset at 0.001
validate_structures:489: Validating structures
Loaded defaults from Downloads/DroneSim-Installer-main/abdaz/ardupilot/Tools/aut
otest/default_params/copter.parm
```

```
Console
MAVProxy  Vehicle  Link  Mission  Rally  Fence  Parameter
STABILIZE   ARM   GPS: OK6 (10)   Vcc: --   Radio: --   INS   MAG   AS   RNG   AHRS   EKF   LOG   FEN   RC   TERI
Batt1: 100%/12.59V 0.0A   Link 1 OK 100.0% (75675 pkts, 0 lost, 0.00s delay)
Hdg 353/ 0   Alt 0m   AGL 0m/0m   AirSpeed 0m/s   GPSSpeed 0m/s   Thr 0   Roll 0   Pitch 0   Wind -180/0m/s
WP 0   Distance 0m   Bearing 0   AltError 0m(L)   AspdError 0m/s(H)   FlightTime --   ETR 0:00   Param 1335/1335   Mission
online system 1
Mode STABILIZE
Init Gyro**
AP: ArduPilot Ready
AP: AHRS: DCM active
fence present
AP: RC7: SaveWaypoint LOW
AP: ArduCopter V4.4.0-dev (465e8839)
AP: ea14ec8d37cb48e097cc478b381d3de0
AP: Frame: QUAD/PLUS
Flight battery 100 percent
AP: EKF3 IMU0 initialised
AP: EKF3 IMU1 initialised
AP: AHRS: EKF3 active
AP: EKF3 IMU0 tilt alignment complete
AP: EKF3 IMU1 tilt alignment complete
AP: EKF3 IMU0 MAG0 initial yaw alignment complete
AP: EKF3 IMU1 MAG0 initial yaw alignment complete
AP: GPS 1: detected as u-blox at 230400 baud
AP: EKF3 IMU1 origin set
AP: EKF3 IMU0 origin set
AP: Field Elevation Set: 584m
pre-arm good
AP: EKF3 IMU1 is using GPS
AP: EKF3 IMU0 is using GPS
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
```



5. In a third terminal, run the required python file. Below are the codes, outputs and terminals of the three python files.

MPC.py

<u>Python code demonstrate Model Predictive Control (MPC) using CasADi library:</u>

```
import casadi as cs

import numpy as np

import matplotlib.pyplot as plt


# Define the system dynamics

A = np.array([[1, 1], [0, 1]])

B = np.array([[0], [1]])

C = np.array([[1, 0], [0, 1]])

D = np.array([[0], [0]])
```

```
# Define the MPC parameters
N = 5
dt = 0.1
Q = np.diag([1, 1])
R = np.array([[1]])

# Define the optimization problem
opti = cs.Opti()

# Define the state variables
x = opti.variable(2, N+1)
x0 = opti.parameter(2, 1)

# Define the control variables
u = opti.variable(1, N)

# Define the reference trajectory
x_ref = opti.parameter(2, N+1)
u_ref = opti.parameter(1, N)

# Define the initial state constraint
opti.subject_to(x[:, 0] == x0)

# Define the dynamic constraints
for k in range(N):
    x_next = cs.mtimes(A, x[:, k]) + cs.mtimes(B, u[:, k])
    opti.subject_to(x[:, k+1] == x_next)
```

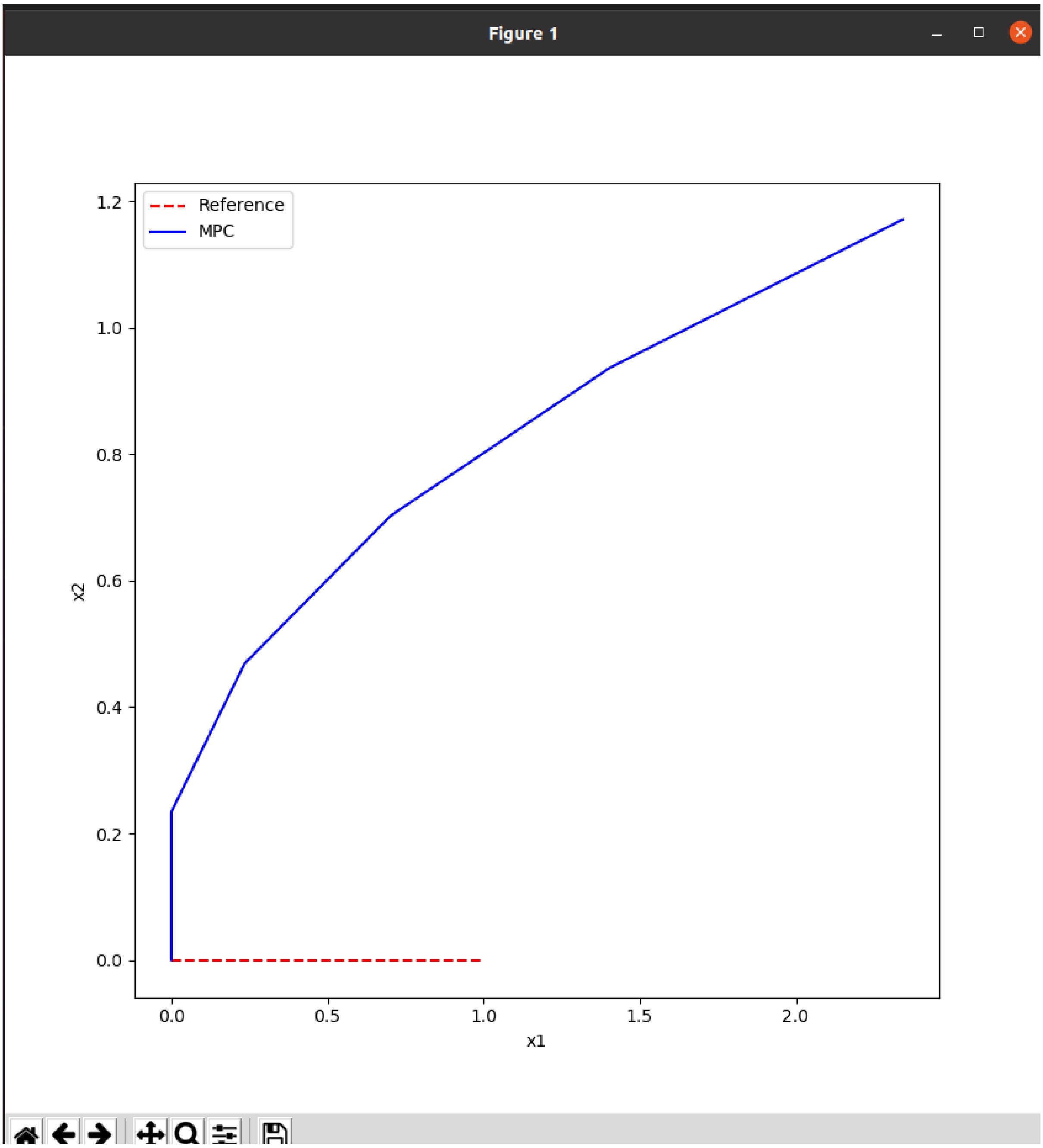
```
# Define the cost function
J = 0
for k in range(N):
    J += cs.mtimes([(x[:, k] - x_ref[:, k]).T, Q, (x[:, k] - x_ref[:, k])])
    J += cs.mtimes([(u[:, k] - u_ref[:, k]).T, R, (u[:, k] - u_ref[:, k])])
opti.minimize(J)

# Define the control constraints
opti.subject_to(u <= 1)
opti.subject_to(u >= -1)

# Set the initial state parameter
x0_val = np.array([[0], [0]])
opti.set_value(x0, x0_val)

# Define the reference trajectory and control inputs
x_ref_val = np.zeros((2, N+1))
x_ref_val[0, :] = np.linspace(0, 1, N+1)
u_ref_val = np.zeros((1, N))
opti.set_value(x_ref, x_ref_val)
opti.set_value(u_ref, u_ref_val)

# Simulate the system and plot the results
x_val = np.zeros((2, N+1))
u_val = np.zeros((1, N))

for i in range(N):
    # Update the optimization problem with the current state
    opti.set_initial(u, u_val)
```

```
    opti.set_initial(x, x_val)

    # Solve the optimization problem
    sol = opti.solve()

    # Extract the control input
    u_val = opti.value(u[:, 0])

    # Update the system state
    x_val[:, i+1] = np.squeeze(cs.mtimes(A, x_val[:, i]) + cs.mtimes(B, u_val))

# Plot the results
plt.plot(x_ref_val[0, :], x_ref_val[1, :], 'r--', label='Reference')
plt.plot(x_val[0, :], x_val[1, :], 'b', label='MPC')
plt.legend()
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

```
meher@ubuntu:~/Downloads$ python MPC.py

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************

This is Ipopt version 3.14.11, running with linear solver MUMPS 5.4.1.

Number of nonzeros in equality constraint Jacobian...:       42
Number of nonzeros in inequality constraint Jacobian.:       10
Number of nonzeros in Lagrangian Hessian.............:       20

Total number of variables............................:       17
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:       12
Total number of inequality constraints...............:       10
        inequality constraints with only lower bounds:        5
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        5

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.2000000e+00 0.00e+00 7.37e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  3.3091670e-01 1.11e-16 1.93e-01  -1.7 7.22e-01    -  8.37e-01 1.00e+00f  1
   2  3.2274534e-01 1.11e-16 2.00e-07  -1.7 3.68e-02    -  1.00e+00 1.00e+00f  1
   3  3.2269431e-01 1.11e-16 1.50e-09  -3.8 3.14e-03    -  1.00e+00 1.00e+00f  1
   4  3.2269430e-01 2.78e-17 1.84e-11  -5.7 4.15e-05    -  1.00e+00 1.00e+00h  1
   5  3.2269430e-01 5.55e-17 2.51e-14  -8.6 9.66e-08    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:   3.2269430051813469e-01    3.2269430051813469e-01
Dual infeasibility......:   2.5059035640133008e-14    2.5059035640133008e-14
Constraint violation....:   5.5511151231257827e-17    5.5511151231257827e-17
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5061070880374509e-09    2.5061070880374509e-09
Overall NLP error.......:   2.5061070880374509e-09    2.5061070880374509e-09
```

meher@ubuntu: ~/Downloads

```
Number of objective function evaluations             = 6
Number of objective gradient evaluations             = 6
Number of equality constraint evaluations            = 6
Number of inequality constraint evaluations          = 6
Number of equality constraint Jacobian evaluations   = 6
Number of inequality constraint Jacobian evaluations = 6
Number of Lagrangian Hessian evaluations             = 5
Total seconds in IPOPT                               = 0.022

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  | 108.00us ( 18.00us)  98.41us ( 16.40us)         6
       nlp_g  |  66.00us ( 11.00us)  63.03us ( 10.50us)         6
  nlp_grad_f  | 101.00us ( 14.43us)  99.46us ( 14.21us)         7
  nlp_hess_l  |  90.00us ( 18.00us)  72.69us ( 14.54us)         5
   nlp_jac_g  | 120.00us ( 17.14us) 111.74us ( 15.96us)         7
       total  |  14.17ms ( 14.17ms)  32.66ms ( 32.66ms)         1
This is Ipopt version 3.14.11, running with linear solver MUMPS 5.4.1.

Number of nonzeros in equality constraint Jacobian...:       42
Number of nonzeros in inequality constraint Jacobian.:       10
Number of nonzeros in Lagrangian Hessian.............:       20

Total number of variables............................:       17
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:       12
Total number of inequality constraints...............:       10
        inequality constraints with only lower bounds:        5
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        5

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.5290891e+00 4.68e-01 7.02e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  3.7148939e-01 5.55e-17 2.10e-01  -1.0 8.52e-01    -  8.25e-01 1.00e+00f  1
   2  3.2324261e-01 1.11e-16 2.00e-07  -1.7 2.14e-01    -  1.00e+00 1.00e+00f  1
   3  3.2269470e-01 1.11e-16 2.83e-08  -2.5 2.70e-02    -  1.00e+00 1.00e+00f  1
   4  3.2269430e-01 1.11e-16 1.50e-09  -3.8 6.59e-04    -  1.00e+00 1.00e+00f  1
   5  3.2269430e-01 1.11e-16 1.84e-11  -5.7 9.17e-06    -  1.00e+00 1.00e+00h  1
   6  3.2269430e-01 1.11e-16 2.51e-14  -8.6 9.54e-08    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 6

                                   (scaled)                 (unscaled)
Objective...............:   3.2269430051813475e-01    3.2269430051813475e-01
Dual infeasibility......:   2.5059035640133008e-14    2.5059035640133008e-14
Constraint violation....:   1.1102230246251565e-16    1.1102230246251565e-16
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
```



RTL.py

Python code to simulate RTL (Return to Launch) functionality in drones:

import time

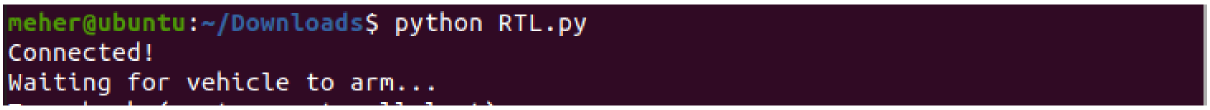
from dronekit import connect, VehicleMode, LocationGlobalRelative

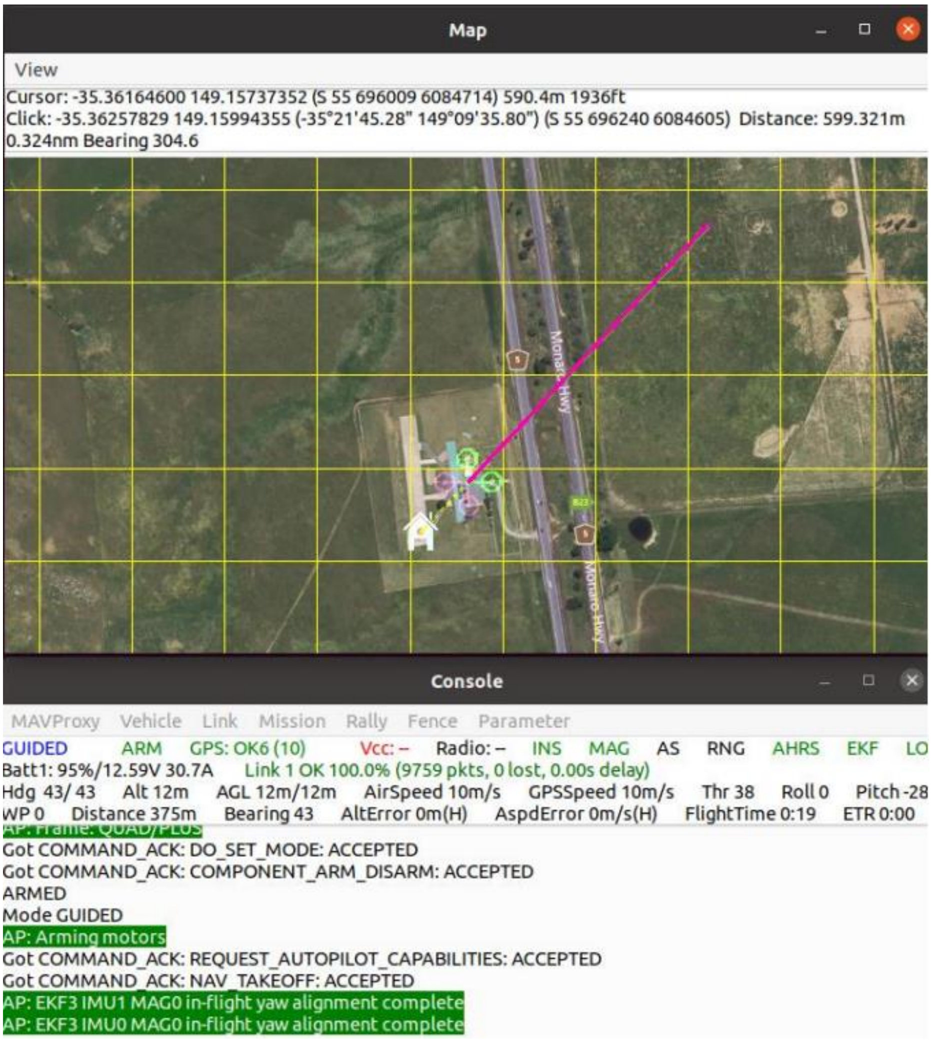
import math

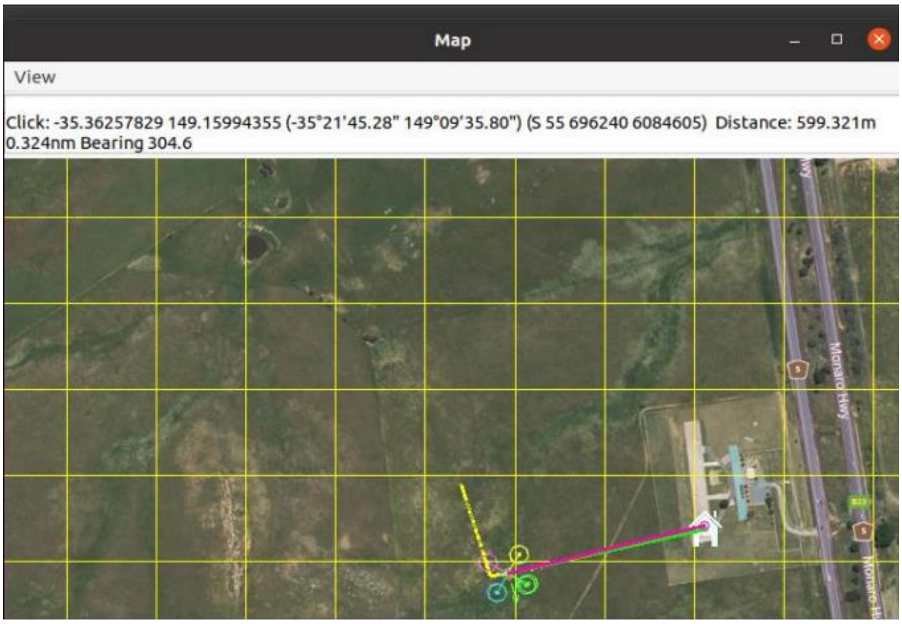
```
def get_distance_metres(aLocation1, aLocation2):
    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lonreturn math.sqrt((dlat*dlat) + (dlong*dlong)) *
1.113195e5


def distance_to_current_waypoint(awp):
    distancetopoint = get_distance_metres(vehicle.location.global_frame, awp)
    return distancetopoint


# Connect to the PX4 vehicle
#connection_string = 'udp:127.0.0.1:14550'
#vehicle = connect(connection_string, wait_ready=True)
vehicle = connect('udp:127.0.0.1:14550')
print("Connected!")


# Set the vehicle mode to GUIDED
vehicle.mode = VehicleMode("GUIDED")


# Arm the vehicle
vehicle.armed = True
while not vehicle.armed:
    print("Waiting for vehicle to arm...")
    time.sleep(1)
vehicle.simple_takeoff(10)


# Wait for the drone to reach a certain altitude
while True:
    altitude = vehicle.location.global_relative_frame.alt
    if altitude >= 9.5:
        break
```

```
    time.sleep(1)

# Define the mission waypoints
waypoints = [
    LocationGlobalRelative(-35.363261, 149.165230, 10),
    LocationGlobalRelative(-35.362933, 149.164652, 10),
    LocationGlobalRelative(-35.363275, 149.164340, 10),
    LocationGlobalRelative(-35.363700, 149.164889, 10)
]

# Move to each waypoint in turn with a fixed altitude of 20 meters
for waypoint in waypoints:
    # Set the target waypoint with a fixed altitude of 20 meters
    target_altitude = 20
    target_location = LocationGlobalRelative(
        waypoint.lat, waypoint.lon, target_altitude)
    vehicle.simple_goto(target_location)

    # Wait for the vehicle to reach the waypoint
    while True:
        current_pos = vehicle.location.global_relative_frame
        dist = current_pos.distance_to(target_location)
        if dist < 1:
            break
        time.sleep(1)

# Set the vehicle mode to RTL (Return to Launch)
vehicle.mode = VehicleMode("RTL")
```

```
# Wait for the vehicle to return to the launch point and land
while vehicle.armed:
    print("Waiting for vehicle to land...")
    time.sleep(1)

# Disconnect from the vehicle
vehicle.close()
```







WAYP.py

<u>Python code to simulate varying altitude of waypoints in drones:</u>

```
import time

from dronekit import connect, VehicleMode, LocationGlobalRelative

import math


def get_distance_metres(aLocation1, aLocation2):

    dlat = aLocation2.lat - aLocation1.lat

    dlong = aLocation2.lon - aLocation1.lonreturn math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5


def distance_to_current_waypoint(awp):

    distancetopoint = get_distance_metres(vehicle.location.global_frame, awp)

    return distancetopoint
```

```
# Connect to the PX4 vehicle
#connection_string = 'udp:127.0.0.1:14550'
#vehicle = connect(connection_string, wait_ready=True)
vehicle = connect('udp:127.0.0.1:14550')
print("Connected!")

# Set the vehicle mode to GUIDED
vehicle.mode = VehicleMode("GUIDED")

# Arm the vehicle
vehicle.armed = True
while not vehicle.armed:
    print("Waiting for vehicle to arm...")
    time.sleep(1)
vehicle.simple_takeoff(10)

# Wait for the drone to reach a certain altitude
while True:
    altitude = vehicle.location.global_relative_frame.alt
    if altitude >= 9.5:
        break
    time.sleep(1)

# Define the mission waypoints
waypoints = [
    LocationGlobalRelative(-35.363261, 149.165230, 10),
    LocationGlobalRelative(-35.362933, 149.164652, 10),
    LocationGlobalRelative(-35.363275, 149.164340, 10),
    LocationGlobalRelative(-35.363700, 149.164889, 10)
]
```

```
# Set the target waypoint with a varying altitude
target_altitude = waypoints.index(waypoint) * 5 + 10
target_location = LocationGlobalRelative(waypoint.lat, waypoint.lon, target_altitude)
vehicle.simple_goto(target_location)

# Wait for the vehicle to reach the waypoint
while True:
    current_pos = vehicle.location.global_relative_frame
    dist = current_pos.distance_to(target_location)
    if dist < 1:
        break
    time.sleep(1)

# Set the vehicle mode to LAND
vehicle.mode = VehicleMode("LAND")

# Wait for the vehicle to land
while vehicle.armed:
    print("Waiting for vehicle to land...")
    time.sleep(1)

# Disconnect from the vehicle
vehicle.close()
```

```
meher@ubuntu:~/Downloads$ python WAYP.py
Connected!
```


Map
View
Monaro Hwy
Monaro Hwy
Console
MAVProxy Vehicle Link Mission Rally Fence Parameter
GUIDED ARM GPS: OK6 (10) Vcc: -- Radio: -- INS MAG AS RNG AHRS EKF LO
Batt1: 99%/12.59V 31.5A Link 1 OK 100.0% (4919 pkts, 0 lost, 0.00s delay)
Hdg 352/315 Alt 5m AGL 5m/3m AirSpeed 2m/s GPSSpeed 0m/s Thr 34 Roll 0 Pitch 0 Wi
WP 0 Distance 0m Bearing 0 AltError 0m(H) AspdError 0m/s(H) FlightTime 0:06 ETR 0:00 Pa
Got COMMAND_ACK: DO_SET_MODE: ACCEPTED
Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
ARMED
Mode GUIDED
AP: Arming motors
Got COMMAND_ACK: REQUEST_AUTOPILOT_CAPABILITIES: ACCEPTED
Got COMMAND_ACK: NAV_TAKEOFF: ACCEPTED
AP: EKF3 IMU0 MAG0 in-flight yaw alignment complete
AP: EKF3 IMU1 MAG0 in-flight yaw alignment complete

Conclusion:

We develop and execute the corresponding Python programmes to demonstrate Model Predictive Control (MPC) with the CasADi library, RTL (Return to Launch) functionality in drones, and setting a variable height for each waypoint of a drone.