NAME : VISHAL KUMAR MAHATHA

REG NO : 20BRS1168

COURSE : Drone Applications and Assembly

LAB : 6

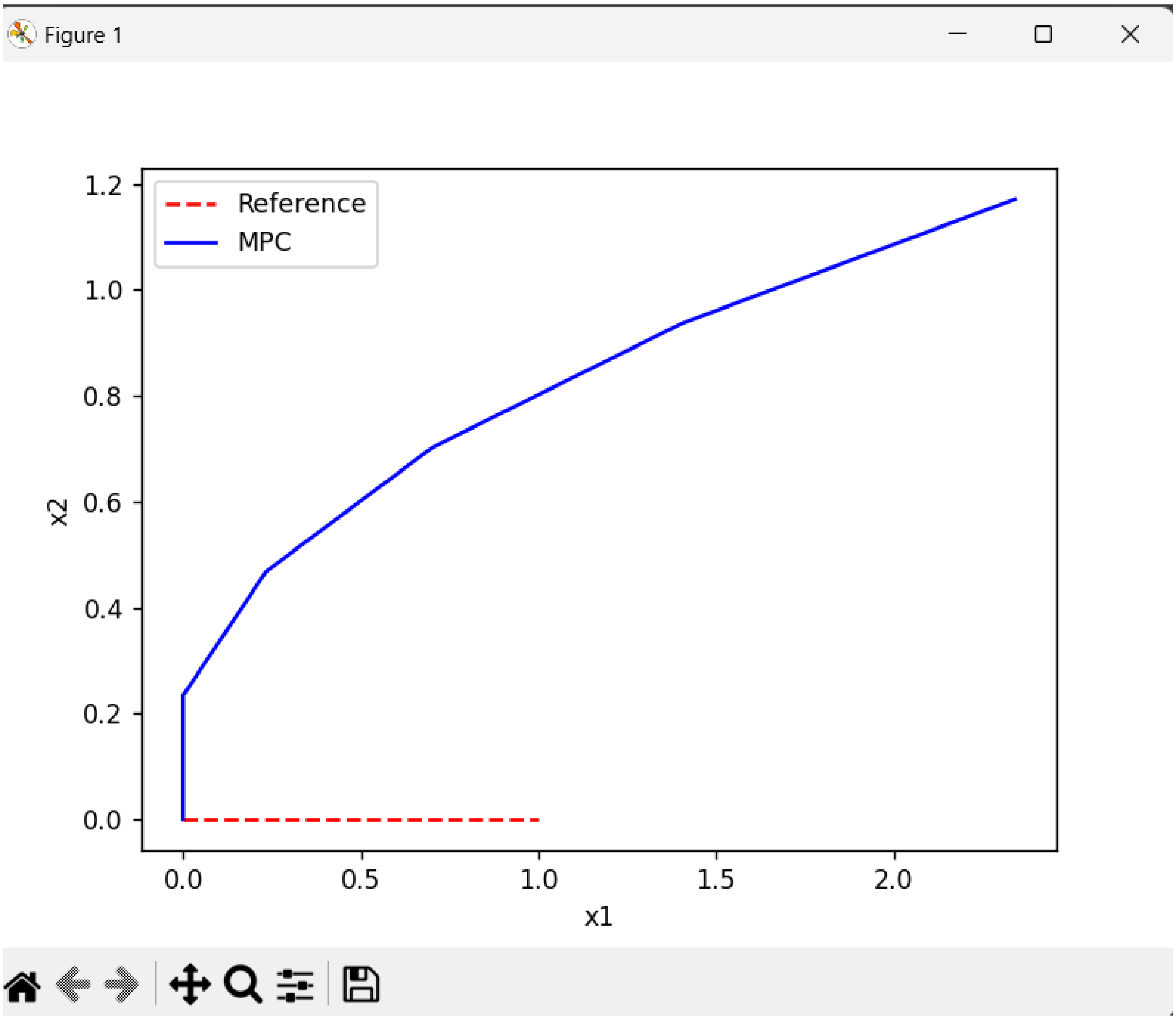
1) CODE :

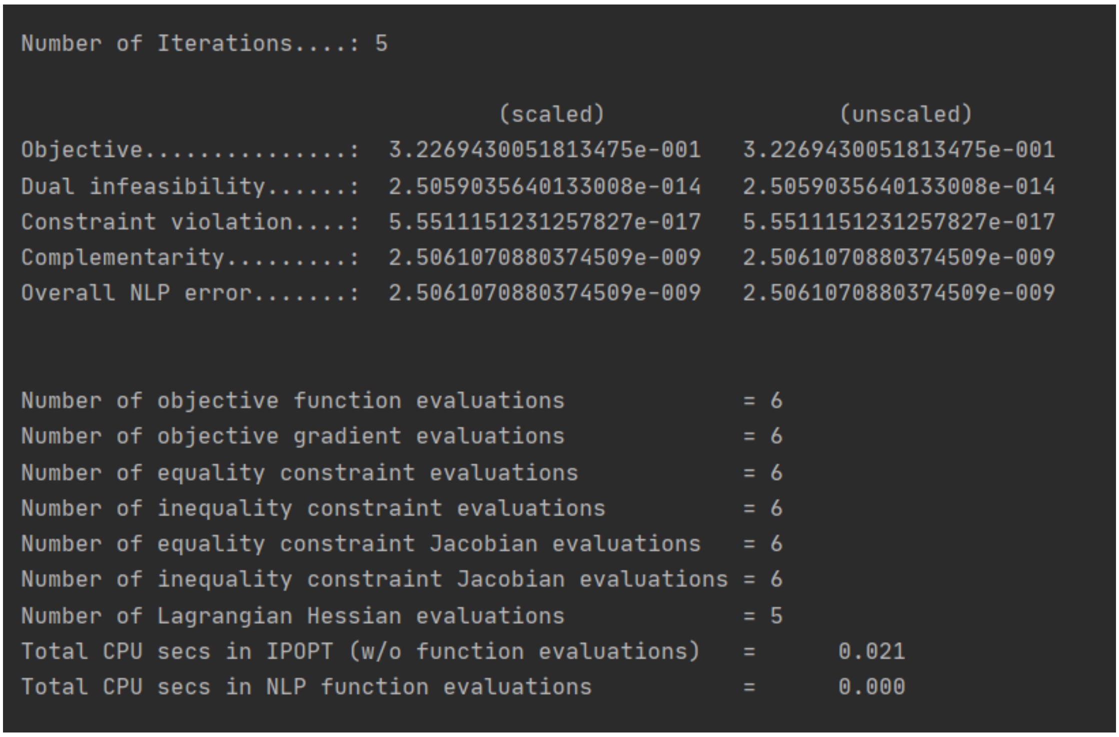
```
import casadi as cs
import numpy as np
import matplotlib.pyplot as plt

# 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)
```

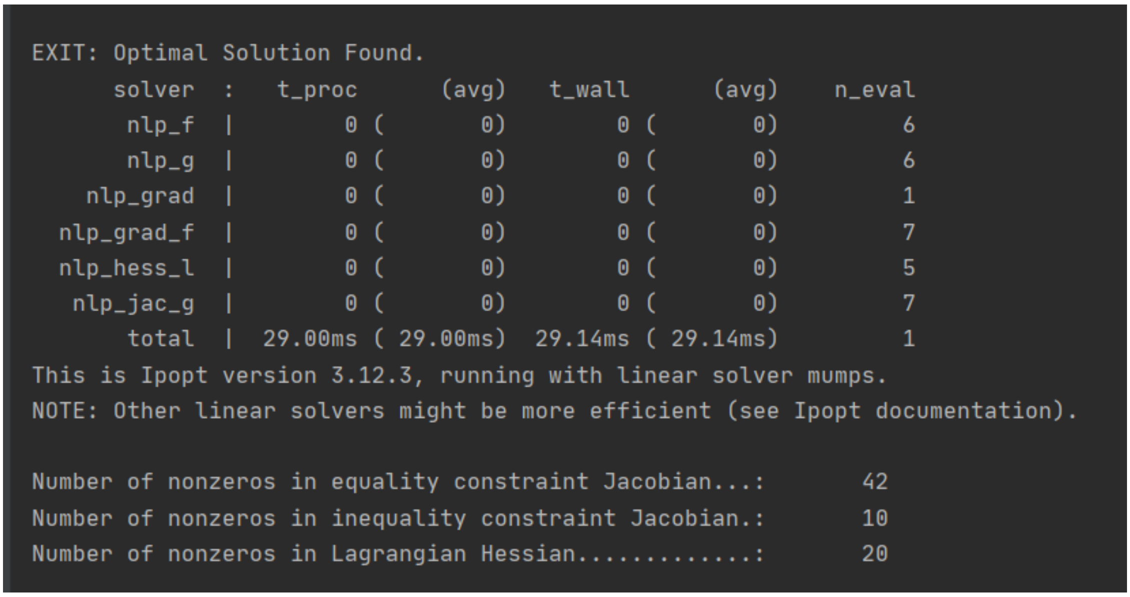
```
u_ref_val = np.zeros((1, N))
opti.set_value(x_ref, x_ref_val)
opti.set_value(u_ref, u_ref_val)

# 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)
    opti.solver(‘ipopt’)
    # 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()
```

OUTPUT:

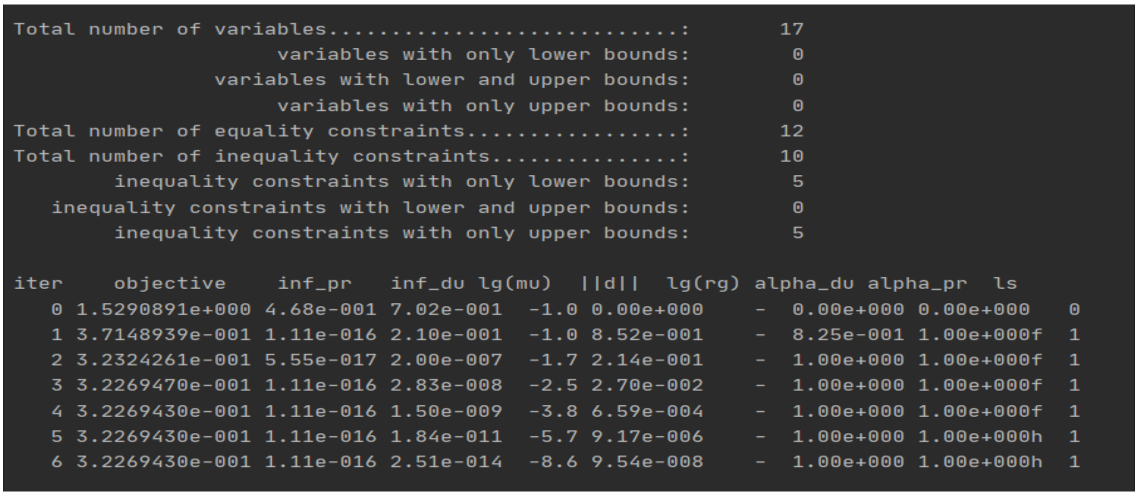




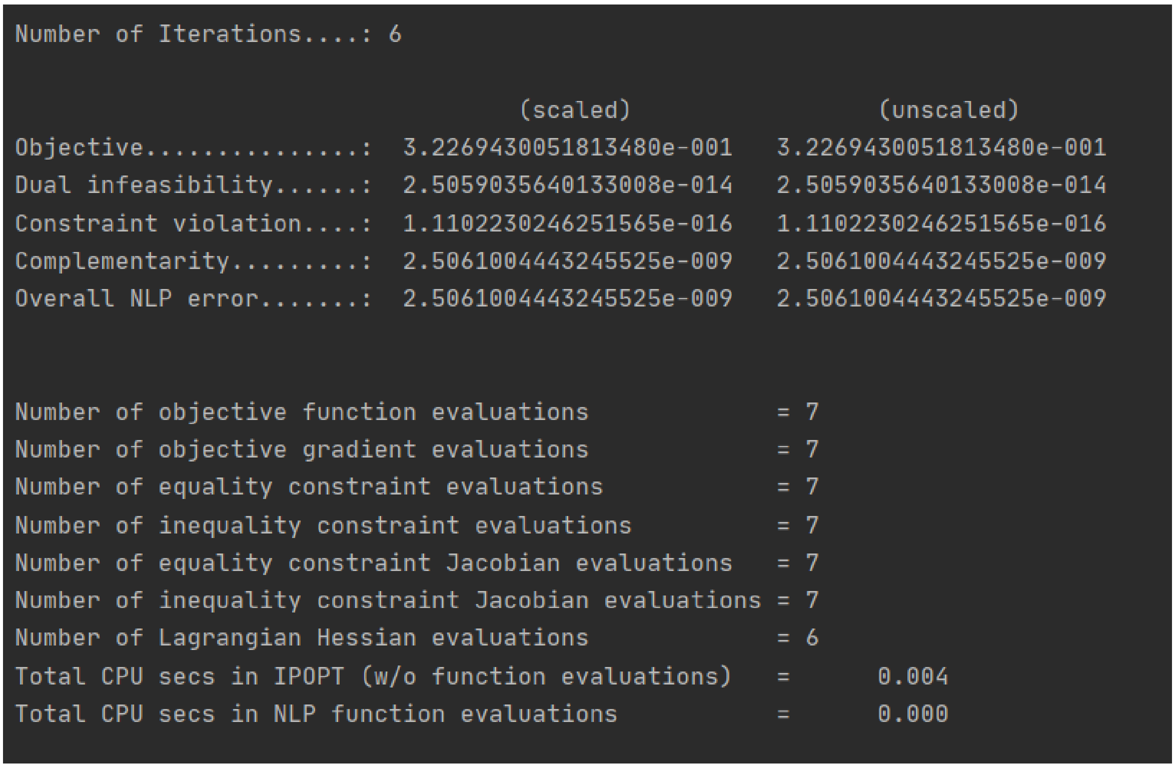
```
Number of Iterations....: 5

                                   (scaled)                 (unscaled)
Objective...............:   3.2269430051813475e-001    3.2269430051813475e-001
Dual infeasibility......:   2.5059035640133008e-014    2.5059035640133008e-014
Constraint violation....:   5.5511151231257827e-017    5.5511151231257827e-017
Complementarity.........:   2.5061070880374509e-009    2.5061070880374509e-009
Overall NLP error.......:   2.5061070880374509e-009    2.5061070880374509e-009


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 CPU secs in IPOPT (w/o function evaluations)   =      0.021
Total CPU secs in NLP function evaluations           =      0.000
```



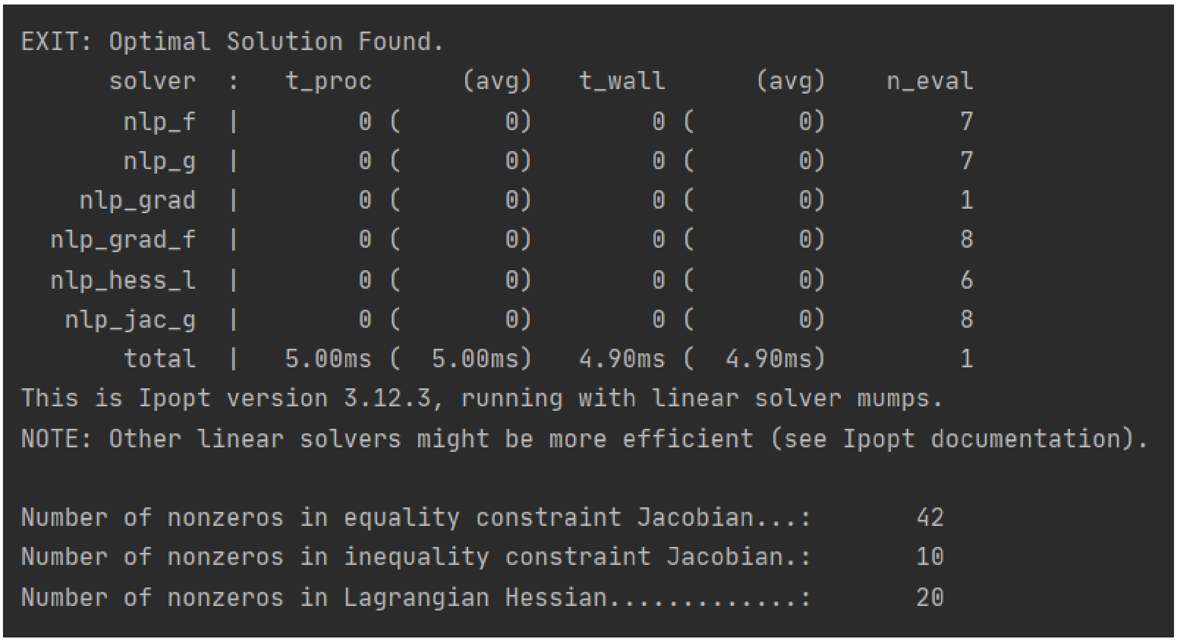
```
EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |        0 (       0)        0 (       0)         6
       nlp_g  |        0 (       0)        0 (       0)         6
    nlp_grad  |        0 (       0)        0 (       0)         1
  nlp_grad_f  |        0 (       0)        0 (       0)         7
  nlp_hess_l  |        0 (       0)        0 (       0)         5
   nlp_jac_g  |        0 (       0)        0 (       0)         7
       total  |  29.00ms ( 29.00ms)  29.14ms ( 29.14ms)         1
This is Ipopt version 3.12.3, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

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



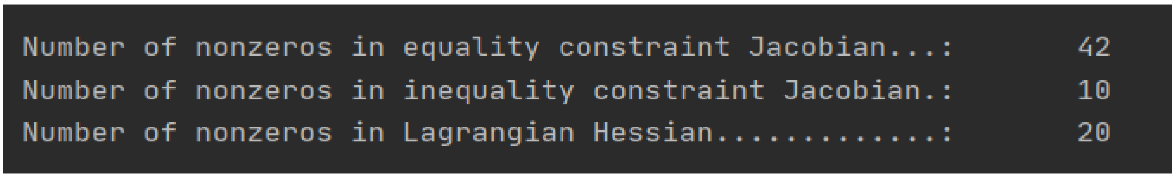
```
Total number of variables............................:       17
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:       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+000 4.68e-001 7.02e-001  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 3.7148939e-001 1.11e-016 2.10e-001  -1.0 8.52e-001    -  8.25e-001 1.00e+000f  1
   2 3.2324261e-001 5.55e-017 2.00e-007  -1.7 2.14e-001    -  1.00e+000 1.00e+000f  1
   3 3.2269470e-001 1.11e-016 2.83e-008  -2.5 2.70e-002    -  1.00e+000 1.00e+000f  1
   4 3.2269430e-001 1.11e-016 1.50e-009  -3.8 6.59e-004    -  1.00e+000 1.00e+000f  1
   5 3.2269430e-001 1.11e-016 1.84e-011  -5.7 9.17e-006    -  1.00e+000 1.00e+000h  1
   6 3.2269430e-001 1.11e-016 2.51e-014  -8.6 9.54e-008    -  1.00e+000 1.00e+000h  1
```



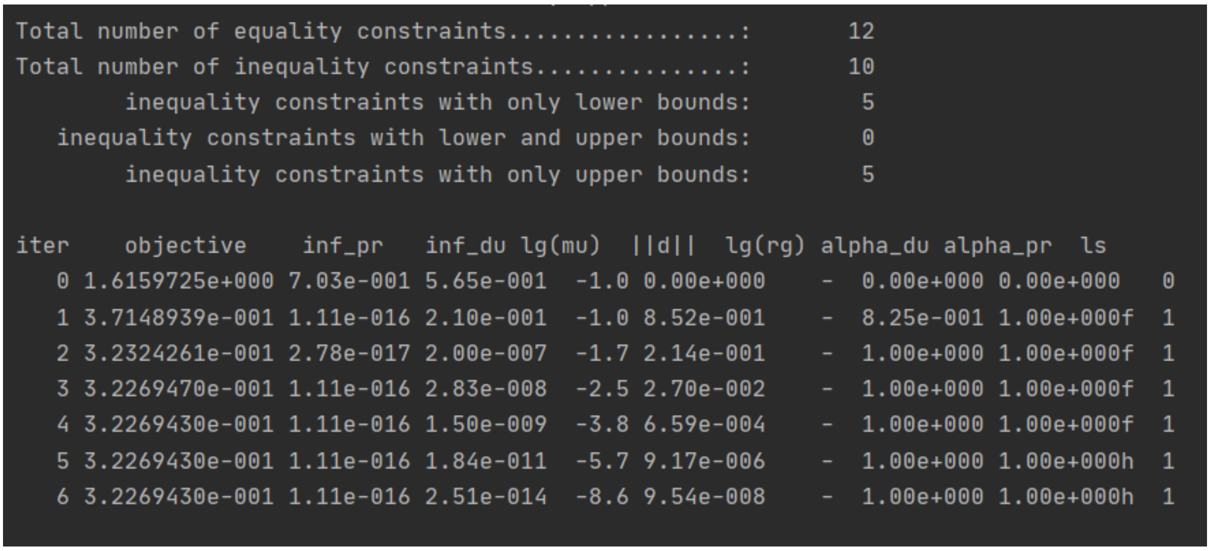
```
Number of Iterations....: 6

                                   (scaled)                 (unscaled)
Objective...............:   3.2269430051813480e-001    3.2269430051813480e-001
Dual infeasibility......:   2.5059035640133008e-014    2.5059035640133008e-014
Constraint violation....:   1.1102230246251565e-016    1.1102230246251565e-016
Complementarity.........:   2.5061004443245525e-009    2.5061004443245525e-009
Overall NLP error.......:   2.5061004443245525e-009    2.5061004443245525e-009


Number of objective function evaluations             = 7
Number of objective gradient evaluations             = 7
Number of equality constraint evaluations            = 7
Number of inequality constraint evaluations          = 7
Number of equality constraint Jacobian evaluations   = 7
Number of inequality constraint Jacobian evaluations = 7
Number of Lagrangian Hessian evaluations             = 6
Total CPU secs in IPOPT (w/o function evaluations)   =      0.004
Total CPU secs in NLP function evaluations           =      0.000
```



```
EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |        0 (        0)        0 (        0)         7
       nlp_g  |        0 (        0)        0 (        0)         7
    nlp_grad  |        0 (        0)        0 (        0)         1
  nlp_grad_f  |        0 (        0)        0 (        0)         8
  nlp_hess_l  |        0 (        0)        0 (        0)         6
   nlp_jac_g  |        0 (        0)        0 (        0)         8
       total  |   5.00ms (  5.00ms)   4.90ms (  4.90ms)         1
This is Ipopt version 3.12.3, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

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



```
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 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 1.6159725e+000 7.03e-001 5.65e-001  -1.0 0.00e+000    -  0.00e+000 0.00e+000   0
   1 3.7148939e-001 1.11e-016 2.10e-001  -1.0 8.52e-001    -  8.25e-001 1.00e+000f  1
   2 3.2324261e-001 2.78e-017 2.00e-007  -1.7 2.14e-001    -  1.00e+000 1.00e+000f  1
   3 3.2269470e-001 1.11e-016 2.83e-008  -2.5 2.70e-002    -  1.00e+000 1.00e+000f  1
   4 3.2269430e-001 1.11e-016 1.50e-009  -3.8 6.59e-004    -  1.00e+000 1.00e+000f  1
   5 3.2269430e-001 1.11e-016 1.84e-011  -5.7 9.17e-006    -  1.00e+000 1.00e+000h  1
   6 3.2269430e-001 1.11e-016 2.51e-014  -8.6 9.54e-008    -  1.00e+000 1.00e+000h  1
```

2)CODE:

```
import time
from dronekit import connect, VehicleMode, LocationGlobalRelative

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

# 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)

# 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)
]
```

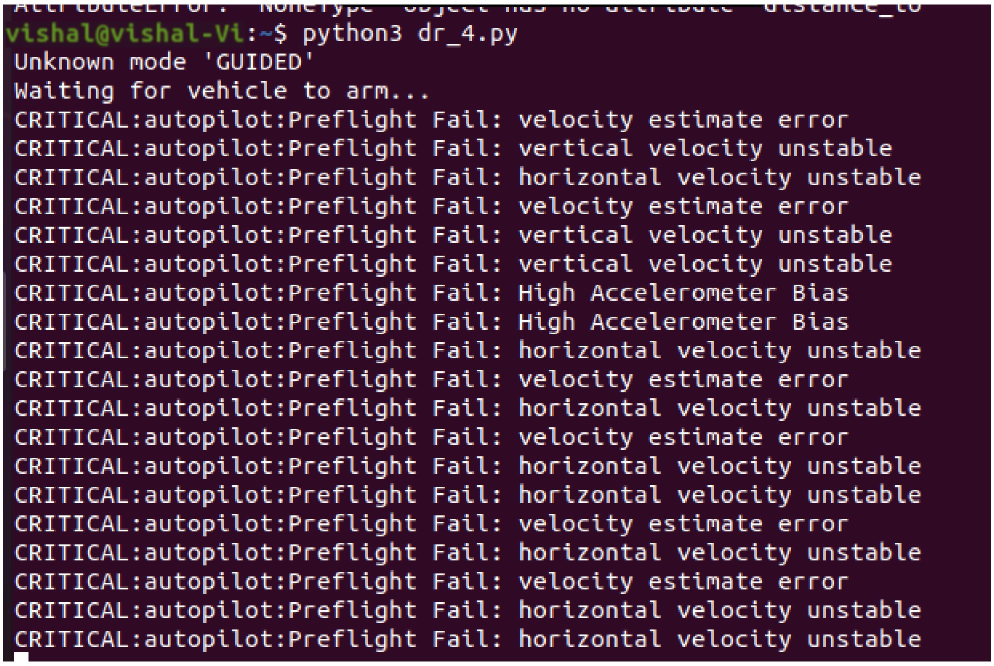
This program sets a fixed altitude of 20 meters for all waypoints. The program also sets the vehicle mode to RTL (Return to Launch) instead of LAND, which will cause the vehicle to automatically return to its launch point and land.

```
# 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)
```

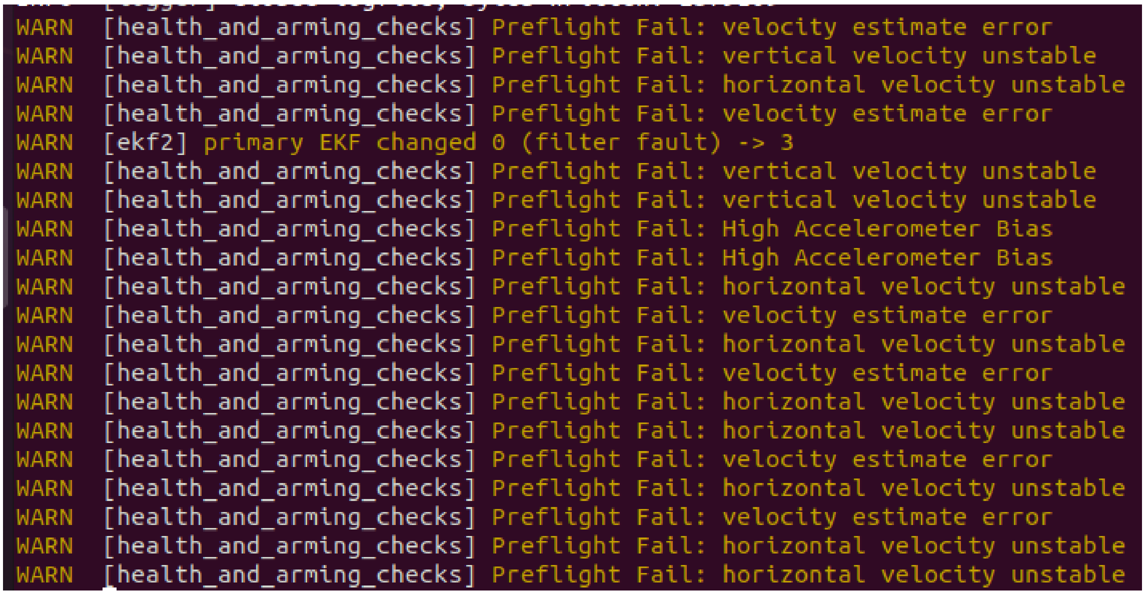
Disconnect from the vehicle

vehicle.close()

OUTPUT :



```
AttributeError: 'NoneType' object has no attribute 'distance_to'
vishal@vishal-Vi:~$ python3 dr_4.py
 Unknown mode 'GUIDED'
 Waiting for vehicle to arm...
 CRITICAL:autopilot:Preflight Fail: velocity estimate error
 CRITICAL:autopilot:Preflight Fail: vertical velocity unstable
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: velocity estimate error
 CRITICAL:autopilot:Preflight Fail: vertical velocity unstable
 CRITICAL:autopilot:Preflight Fail: vertical velocity unstable
 CRITICAL:autopilot:Preflight Fail: High Accelerometer Bias
 CRITICAL:autopilot:Preflight Fail: High Accelerometer Bias
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: velocity estimate error
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: velocity estimate error
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: velocity estimate error
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: velocity estimate error
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
 CRITICAL:autopilot:Preflight Fail: horizontal velocity unstable
```



```
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
WARN  [health_and_arming_checks] Preflight Fail: vertical velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
WARN  [ekf2] primary EKF changed 0 (filter fault) -> 3
WARN  [health_and_arming_checks] Preflight Fail: vertical velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: vertical velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: High Accelerometer Bias
WARN  [health_and_arming_checks] Preflight Fail: High Accelerometer Bias
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
WARN  [health_and_arming_checks] Preflight Fail: horizontal velocity unstable
```

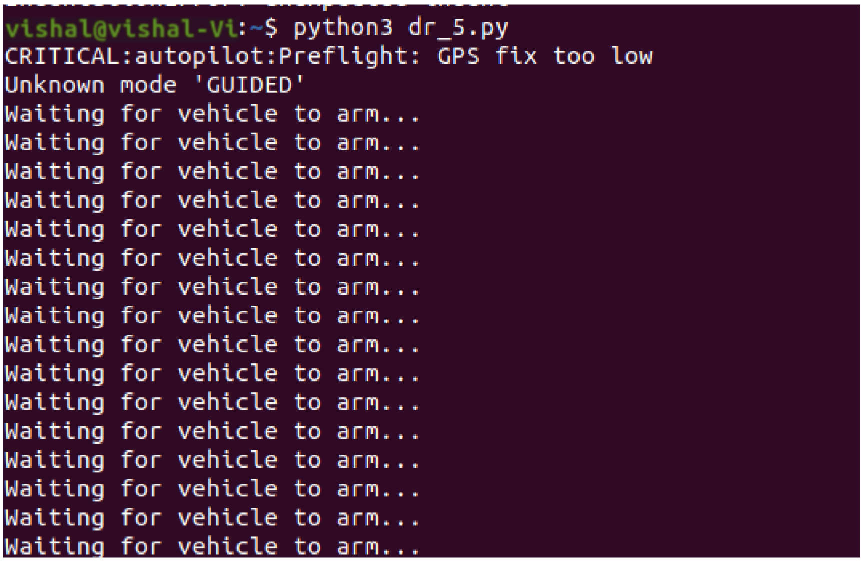
3)CODE:

import time

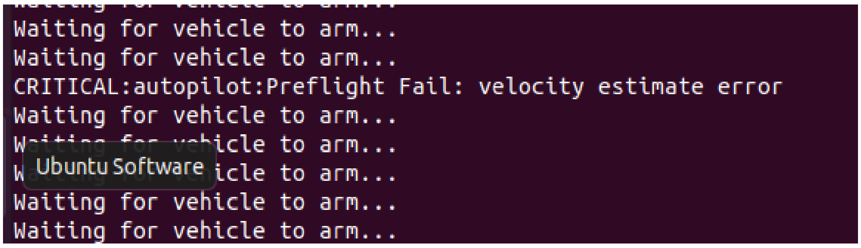
from dronekit import connect, VehicleMode, LocationGlobalRelative

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

# 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)

# Define the mission waypoints
waypoints = [
    LocationGlobalRelative(-35.363261, 149.165230, 10),
    LocationGlobalRelative(-35.362933, 149.164652, 15),
    LocationGlobalRelative(-35.363275, 149.164340, 20),
    LocationGlobalRelative(-35.363700, 149.164889, 10)
]

# Move to each waypoint in turn with a varying altitude
for waypoint in waypoints:
    # 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
```

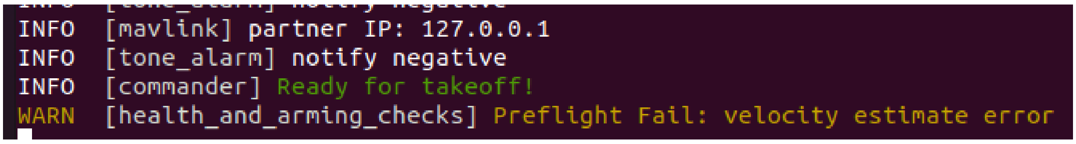
OUTPUT :



```
vishal@vishal-Vi:~$ python3 dr_5.py
CRITICAL:autopilot:Preflight: GPS fix too low
Unknown mode 'GUIDED'
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
```



```
Waiting for vehicle to arm...
Waiting for vehicle to arm...
CRITICAL:autopilot:Preflight Fail: velocity estimate error
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
Waiting for vehicle to arm...
```



```
INFO  [tone_alarm] notify negative
INFO  [mavlink] partner IP: 127.0.0.1
INFO  [tone_alarm] notify negative
INFO  [commander] Ready for takeoff!
WARN  [health_and_arming_checks] Preflight Fail: velocity estimate error
```