Natalia Sempere Sánchez
IR2136 - Robots Aeris i Submarins (2024-25)

LAB02 : CONTROL SITL PyMavLink

1) OBJECTIVE OF PYMAVLINK

Control a simulated drone in SITL step by step using PyMavLink.

Students will learn to :

- Connect to SITL.
- Read basic sensors (position and altitude)
- Send the drone to a specific position
- Arm the drone
- Perform a takeoff
- Monitor the battery status
- Implement an automatic safety landing based on battery charge.

2) STEP 1 : CONNECT TO SITL

First, establish a connection to the SITL simulator on the appropriate port

```
Unset
from pymavlink import mavutil

# Connect to SITL

connection = mavutil.mavlink_connection('udp:127.0.0.1:14550')
connection.wait_heartbeat() # Wait for the first heartbeat to confirm
connection
print("Connected to SITL.")
```

3) STEP 2 : READ BASIC SENSORS

Retrieve basic information from the drone’s sensors, such as global position and relative altitude.

```
Unset
def read_sensors(connection):
    msg = connection.recv_match(type='GLOBAL_POSITION_INT', blocking=True)
    if msg:
        latitude = msg.lat / 1e7  # Convert to degrees
        longitude = msg.lon / 1e7  # Convert to degrees
        altitude = msg.relative_alt / 1000.0  # Convert to meters
        print(f"Position: Lat={latitude}, Lon={longitude}, Alt={altitude}
m")
    else:
        print("Failed to read sensor data.")

# Call the function
read_sensors(connection)
```

4) STEP 3 : ARM THE DRONE

The drone must be armed before flight.

```
Unset
def arm_vehicle(connection):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
        0,  # Confirmation
        1,  # Arm
        0, 0, 0, 0, 0, 0
    )
    print("Drone armed.")

# Arm the drone
arm_vehicle(connection)
```

5) STEP 4 : PERFORM A TAKEOFF

Command the drone to take off to a safe altitude

```
Unset
def takeoff(connection, altitude):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_NAV_TAKEOFF,
        0,  # Confirmation
        0, 0, 0, 0, 0, 0, altitude  # Desired altitude
    )
    print(f"Taking off to {altitude} meters.")

# Take off to 10 meters
takeoff(connection, 10)
time.sleep(15)
```

6) STEP 5 : SEND THE DRONE TO A SPECIFIC POSITION

Define a target position with latitude, longitude, and altitude. The drone must be in GUIDED mode to accept this command.

```
Unset
def set_mode(connection, mode):
    connection.set_mode(mode)
    print(f"Mode changed to {mode}.")

def go_to_position(connection, lat, lon, alt):
    connection.mav.set_position_target_global_int_send(
        0,  # Timestamp
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
        0b110111111000,  # Ignore velocity and acceleration
        int(lat * 1e7),  # Latitude as integer
        int(lon * 1e7),  # Longitude as integer
        alt,  # Desired altitude
        0, 0, 0,  # Velocities
        0, 0, 0 ,  # Accelerations
        0, 0 # Yaw and Yaw rate
    )
    print(f"Moving drone to Lat? {lat}, Long={lon}, Alt={alt} m")

set_mode(connection, 'GUIDED')

# Take off to 10 meters
takeoff(connection, 10)
time.sleep(15)

# Change to GUIDED mode and move the drone

go_to_position(connection, 37.7749, -122.4194, 10) #Example coordinates
```

7) STEP 6 : MONITOR BATTERY STATUS

Define a target position with latitude, longitude, and altitude. The drone must be in GUIDED mode to accept this command.

```
Unset
def read_battery_status(connection):
    msg = connection.recv_match(type='BATTERY_STATUS', blocking=True,
timeout=5)
    if msg :
        voltage = msg.voltages[0] / 1000.0 # Convert mV to V
        current = msg.current_battery / 100.0  # Convert cA to A
        remaining = msg.battery_remaining  # Remaining battery percentage
        print(f"Battery: {voltage:.2f} V, Current: {current:.2f} A,
Remaining: {remaining}%")
        return remaining
    else:
        print("Failed to get battery status.")
        return None

# Read battery status
read_battery_status(connection)
```

8) STEP 7 : IMPLEMENT AUTOMATIC SAFETY LANDING

If the battery falls below 20% (in my case 50), perform an emergency landing.

```
Unset
def land_vehicle(connection):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_NAV_LAND,
        0,
        0, 0, 0, 0, 0, 0, 0
    )
    print("Landing command sent.")

# Monitor battery and land if necessary
while True:
    battery_remaining = read_battery_status(connection)
    if battery_remaining is not None and battery_remaining <= 50:
        print("Low battery! Initiating emergency landing.")
        land_vehicle(connection)
        break
    time.sleep(5)  # Check every 5 seconds
```

9) CÓDIGO COMPLETO

Unset
```
from pymavlink import mavutil
import time
def read_sensors(connection):
    msg = connection.recv_match(type='GLOBAL_POSITION_INT', blocking=True)
    if msg:
        latitude = msg.lat / 1e7  # Convert to degrees
        longitude = msg.lon / 1e7  # Convert to degrees
        altitude = msg.relative_alt / 1000.0  # Convert to meters
        print(f"Position: Lat={latitude}, Lon={longitude}, Alt={altitude}
m")
    else:
        print("Failed to read sensor data.")


def arm_vehicle(connection):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
        0,  # Confirmation
        1,  # Arm
        0, 0, 0, 0, 0, 0
    )
    print("Drone armed.")

def takeoff(connection, altitude):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_NAV_TAKEOFF,
        0,  # Confirmation
        0, 0, 0, 0, 0, 0, altitude  # Desired altitude
    )
    print(f"Taking off to {altitude} meters.")


def set_mode(connection, mode):
    connection.set_mode(mode)
    print(f"Mode changed to {mode}.")

def go_to_position(connection, lat, lon, alt):
    connection.mav.set_position_target_global_int_send(
        0,  # Timestamp
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
        0b110111111000,  # Ignore velocity and acceleration
        int(lat * 1e7),  # Latitude as integer
        int(lon * 1e7),  # Longitude as integer
        alt,  # Desired altitude
        0, 0, 0,  # Velocities
        0, 0, 0 ,  # Accelerations
        0, 0 # Yaw and Yaw rate
    )
    print(f"Moving drone to Lat? {lat}, Long={lon}, Alt={alt} m")

def read_battery_status(connection):
    msg = connection.recv_match(type='BATTERY_STATUS', blocking=True,
timeout=5)
    if msg :
        voltage = msg.voltages[0] / 1000.0 # Convert mV to V
        current = msg.current_battery / 100.0  # Convert cA to A
        remaining = msg.battery_remaining  # Remaining battery percentage
        print(f"Battery: {voltage:.2f} V, Current: {current:.2f} A,
Remaining: {remaining}%")
        return remaining
    else:
        print("Failed to get battery status.")
        return None

def land_vehicle(connection):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_NAV_LAND,
        0,
        0, 0, 0, 0, 0, 0, 0
    )
    print("Landing command sent.")

# Connect to SITL
connection = mavutil.mavlink_connection('udp:127.0.0.1:14550')
connection.wait_heartbeat() # Wait for the first heartbeat to confirm
connection
print("Connected to SITL.")

# Call the function
read_sensors(connection)

# Arm the drone
arm_vehicle(connection)

set_mode(connection, 'GUIDED')
```

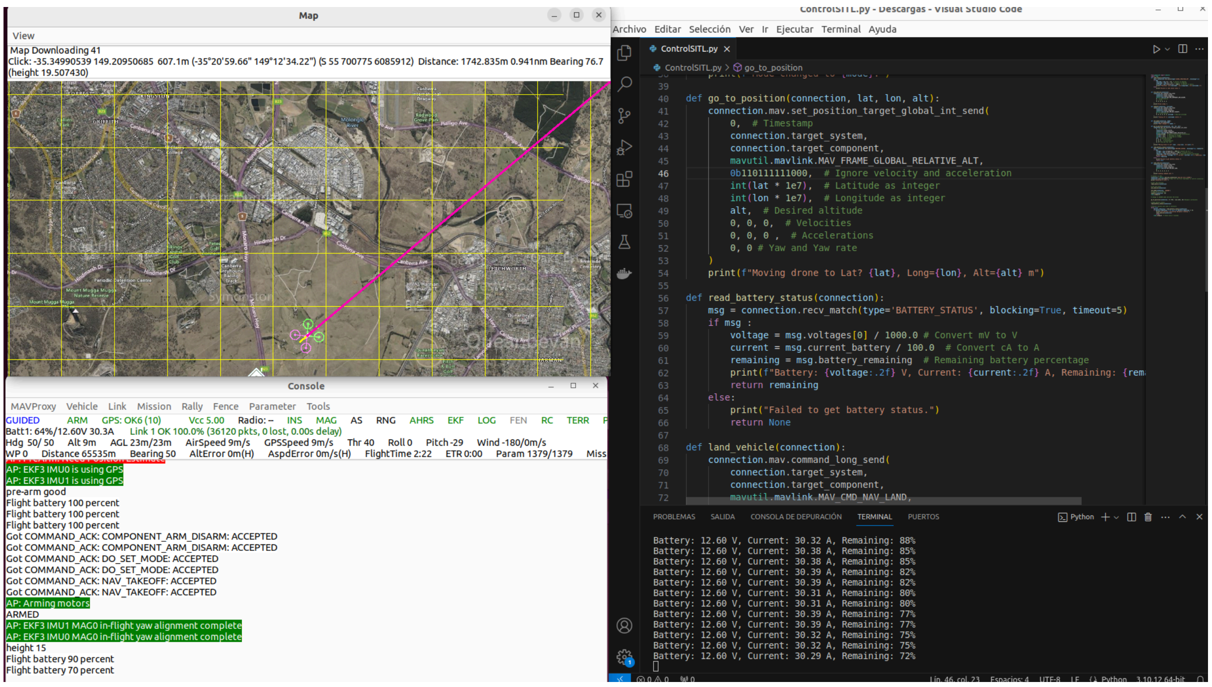
```
# Take off to 10 meters
takeoff(connection, 10)
time.sleep(15)

# Change to GUIDED mode and move the drone

go_to_position(connection, 37.7749, -122.4194, 10) #Example coordinates

# Read battery status
read_battery_status(connection)

# Monitor battery and land if necessary
while True:
    battery_remaining = read_battery_status(connection)
    if battery_remaining is not None and battery_remaining <= 50:
        print("Low battery! Initiating emergency landing.")
        land_vehicle(connection)
        break
    time.sleep(5)  # Check every 5 seconds
```


Map
View
Console
MAVProxy Vehicle Link Mission Rally Fence Parameter Tools
ControlSITL.py - Descargas - Visual Studio Code
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
ControlSITL.py
ControlSITL.py > go_to_position
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
Python
Lín. 46, col. 23 Espacios: 4 UTF-8 LF Python 3.10.12 64-bit

```
Map Downloading 41
Click: -35.34990539 149.20950685  607.1m (-35°20'59.66" 149°12'34.22") (S 55 700775 6085912)  Distance: 1742.835m 0.941nm Bearing 76.7
(height 19.507430)
```

```
GUIDED     ARM    GPS: OK6 (10)     Vcc 5.00    Radio: --   INS   MAG   AS   RNG   AHRS   EKF   LOG   FEN   RC   TERR   P
Batt1: 64%/12.60V 30.3A     Link 1 OK 100.0% (36120 pkts, 0 lost, 0.00s delay)
Hdg  50/ 50    Alt 9m    AGL 23m/23m    AirSpeed 9m/s    GPSSpeed 9m/s    Thr 40    Roll 0    Pitch -29    Wind -180/0m/s
WP 0    Distance 65535m    Bearing 50    AltError 0m(H)    AspdError 0m/s(H)    FlightTime 2:22    ETR 0:00    Param 1379/1379    Miss
AP: EKF3 IMU0 is using GPS
AP: EKF3 IMU1 is using GPS
pre-arm good
Flight battery 100 percent
Flight battery 100 percent
Flight battery 100 percent
Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
Got COMMAND_ACK: DO_SET_MODE: ACCEPTED
Got COMMAND_ACK: DO_SET_MODE: ACCEPTED
Got COMMAND_ACK: NAV_TAKEOFF: ACCEPTED
Got COMMAND_ACK: NAV_TAKEOFF: ACCEPTED
AP: Arming motors
ARMED
AP: EKF3 IMU1 MAG0 in-flight yaw alignment complete
AP: EKF3 IMU0 MAG0 in-flight yaw alignment complete
height 15
Flight battery 90 percent
Flight battery 70 percent
```

```python

def go_to_position(connection, lat, lon, alt):
    connection.mav.set_position_target_global_int_send(
        0,  # Timestamp
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,
        0b110111111000,  # Ignore velocity and acceleration
        int(lat * 1e7),  # Latitude as integer
        int(lon * 1e7),  # Longitude as integer
        alt,  # Desired altitude
        0, 0, 0,  # Velocities
        0, 0, 0 ,  # Accelerations
        0, 0 # Yaw and Yaw rate
    )
    print(f"Moving drone to Lat? {lat}, Long={lon}, Alt={alt} m")

def read_battery_status(connection):
    msg = connection.recv_match(type='BATTERY_STATUS', blocking=True, timeout=5)
    if msg :
        voltage = msg.voltages[0] / 1000.0 # Convert mV to V
        current = msg.current_battery / 100.0  # Convert cA to A
        remaining = msg.battery_remaining  # Remaining battery percentage
        print(f"Battery: {voltage:.2f} V, Current: {current:.2f} A, Remaining: {rem
        return remaining
    else:
        print("Failed to get battery status.")
        return None

def land_vehicle(connection):
    connection.mav.command_long_send(
        connection.target_system,
        connection.target_component,
        mavutil.mavlink.MAV_CMD_NAV_LAND,
```

```
Battery: 12.60 V, Current: 30.32 A, Remaining: 88%
Battery: 12.60 V, Current: 30.38 A, Remaining: 85%
Battery: 12.60 V, Current: 30.38 A, Remaining: 85%
Battery: 12.60 V, Current: 30.39 A, Remaining: 82%
Battery: 12.60 V, Current: 30.39 A, Remaining: 82%
Battery: 12.60 V, Current: 30.31 A, Remaining: 80%
Battery: 12.60 V, Current: 30.31 A, Remaining: 80%
Battery: 12.60 V, Current: 30.39 A, Remaining: 77%
Battery: 12.60 V, Current: 30.39 A, Remaining: 77%
Battery: 12.60 V, Current: 30.32 A, Remaining: 75%
Battery: 12.60 V, Current: 30.32 A, Remaining: 75%
Battery: 12.60 V, Current: 30.29 A, Remaining: 72%
```