**Motor Arming Using Python Scripting**

**Step 1: Verify Key Parameters in Mission Planner**

Since EKF3 still requires GPS, you must force the Cube+ to use LiDAR and IMU for altitude estimation.

Go to Mission Planner → Config → Full Parameter List and Set:

Parameter Value Description

EK3\_ENABLE 1 Enable EKF3

EK3\_SRC1\_POSZ 1 Use Rangefinder (LiDAR) for altitude

EK3\_SRC1\_VELZ 3 Use Barometer for vertical velocity

EK3\_SRC1\_POSXY 3 Use Optical Flow for position (Set to 5 if Optical Flow is unavailable)

EK3\_SRC1\_VELXY 5 Use IMU (Accelerometer) for velocity

EK3\_SRC1\_YAW 2 Use Compass for yaw estimation

EK3\_RNG\_USE\_HGT 70 Use rangefinder below 7m

GPS\_TYPE 0 Disable GPS completely

AHRS\_GPS\_USE 0 Prevent AHRS from using GPS

EK3\_ALT\_M\_NSE 0.05 Set altitude noise value

EK3\_RNG\_M\_NSE 0.1 Set LiDAR noise value

ARMING\_CHECK 0 Disable all pre-arm checks

Set Rangefinder (LiDAR) Settings

Parameter Value Description

SERIAL2\_PROTOCOL 9 Enables rangefinder (LiDAR)

SERIAL2\_BAUD 115200 TF Mini LiDAR baud rate

RNGFND1\_TYPE 20 TF Mini LiDAR

RNGFND1\_MIN\_CM 30 Minimum range (30 cm)

RNGFND1\_MAX\_CM 600 Maximum range (6m)

RNGFND1\_GNDCLEAR 10 LiDAR height from ground

RNGFND1\_ORIENT 25 Set downward-facing orientation

Write Parameters & Reboot Cube+

**Step 2: Check LiDAR Data in Mission Planner**

Go to Mission Planner → Status Tab

Look for:

rangefinder1

sonarrange

sonarvoltage

If values are 0 or missing, the LiDAR is not working.

Check LiDAR wiring.

Ensure SERIAL2\_BAUD = 115200.

Reboot Cube+ and check again.

**Step-3 Python Scripting for Arming**

from dronekit import connect, VehicleMode

import time

# ✅ Fast connection function (No parameter updates)

def connect\_vehicle():

print("🔄 Connecting to Cube+ on /dev/ttyUSB0...")

vehicle = connect('/dev/ttyUSB0', baud=57600, wait\_ready=False) # ✅ Fastest connection

print("✅ Connection successful!")

return vehicle

# ✅ Fast arming function with non-blocking checks

def arm\_motors(vehicle):

print("🔄 Switching to STABILIZE mode...")

vehicle.mode = VehicleMode("STABILIZE") # ✅ Switch mode first

print("🔄 Attempting to arm motors...")

vehicle.armed = True

# ✅ Use non-blocking checks for faster execution

timeout = time.time() + 5 # Allow max 5 seconds for arming

while not vehicle.armed and time.time() < timeout:

print("⏳ Waiting for motors to arm...")

time.sleep(0.5)

if vehicle.armed:

print("✅ Motors are armed and spinning.")

else:

print("❌ Failed to arm motors. Check pre-arm status.")

time.sleep(3) # ✅ Keep motors spinning for 3 seconds

# Main execution block

if \_name\_ == "\_main\_":

vehicle = connect\_vehicle() # Connect to Cube+

arm\_motors(vehicle) # Arm motors only

vehicle.close() # Disconnect after test

print("🔌 Disconnected from Cube+.")

**RUN CODE:**

**Python arming.py**

**OUTPUT:**Motors Arming for 5seconds

**2 Meter Hovering And Landing**

**Python code:**

from dronekit import connect, VehicleMode

from pymavlink import mavutil

import time

# ✅ Connect to Cube+

def connect\_vehicle():

print("🔄 Connecting to Cube+ on /dev/ttyUSB0...")

vehicle = connect('/dev/ttyUSB0', baud=57600, wait\_ready=True)

print("✅ Connection successful!")

return vehicle

# ✅ Send throttle command (override RC channel 3)

def send\_throttle(vehicle, throttle):

vehicle.channels.overrides['3'] = throttle

print(f"⚡ Throttle set to {throttle} (RC Channel 3)")

# ✅ Get altitude from LiDAR

def get\_lidar\_altitude(vehicle):

if vehicle.rangefinder and vehicle.rangefinder.distance:

return vehicle.rangefinder.distance

else:

print("⚠ Warning: LiDAR altitude unavailable. Using 0m.")

return 0

# ✅ Arm motors and hold at idle

def arm\_motors(vehicle):

print("🔄 Switching to STABILIZE mode...")

vehicle.mode = VehicleMode("STABILIZE")

time.sleep(1)

print("🔄 Attempting to arm motors...")

vehicle.armed = True

timeout = time.time() + 5

while not vehicle.armed and time.time() < timeout:

print("⏳ Waiting for motors to arm...")

time.sleep(0.5)

if vehicle.armed:

print("✅ Motors armed successfully!")

else:

print("❌ Failed to arm motors.")

# ✅ Gradual Takeoff with Proper Throttle

def takeoff(vehicle, target\_altitude):

print("🚀 Switching to GUIDED mode for takeoff...")

vehicle.mode = VehicleMode("GUIDED")

time.sleep(1)

print("🔄 Arming motors for takeoff...")

arm\_motors(vehicle)

print("🛫 Gradual takeoff to 2m...")

throttle = 1100 # Start with low throttle (RC values: 1000-2000)

while True:

altitude = get\_lidar\_altitude(vehicle)

if altitude >= target\_altitude \* 0.95:

print(f"✅ Reached target altitude: {altitude:.2f}m")

break

throttle += 10 # Gradually increase throttle

if throttle > 1600:

throttle = 1600 # Prevent excessive throttle

send\_throttle(vehicle, throttle) # Apply throttle

time.sleep(0.1)

# ✅ Stable Hover at 2m

def hover(vehicle, duration):

print(f"🛸 Hovering at 2m for {duration} seconds...")

start\_time = time.time()

throttle = 1500 # Maintain throttle for hovering

while time.time() - start\_time < duration:

altitude = get\_lidar\_altitude(vehicle)

error = 2.0 - altitude

# Adjust throttle based on altitude error

if error > 0.1:

throttle += 10 # Increase if altitude is too low

elif error < -0.1:

throttle -= 10 # Decrease if altitude is too high

throttle = max(1200, min(throttle, 1700)) # Keep within safe range

send\_throttle(vehicle, throttle)

time.sleep(0.1)

# ✅ Controlled Landing

def land(vehicle):

print("🛬 Initiating landing...")

vehicle.mode = VehicleMode("LAND")

time.sleep(1)

throttle = 1300 # Reduce throttle for descent

while vehicle.armed:

altitude = get\_lidar\_altitude(vehicle)

if altitude <= 0.05:

break

throttle -= 10 # Gradually reduce throttle

send\_throttle(vehicle, max(throttle, 1000)) # Prevent sudden drop

print(f"⬇ Landing... Altitude: {altitude:.2f}m")

time.sleep(0.5)

print("✅ Landed safely. Motors disarmed.")

# ✅ Main Execution

if \_name\_ == "\_main\_":

vehicle = connect\_vehicle()

takeoff(vehicle, 2.0)

hover(vehicle, 5)

land(vehicle)

vehicle.close()

print("🔌 Disconnected from Cube+.")

**Output:**

Drone TakeOff to 2m   
but failed in expected Landing