Motor Control with Thorlabs Kinesis

For driving a motorized stage from Python. Assumes you can write a function and work with arrays — see Python Basics.

This page covers driving a motorized translation stage from Python — connecting, reading position, moving, homing, and running a position scan. Examples use a **Thorlabs KST101 controller** with a **ZST225 linear stage** (25 mm travel) as a concrete example, controlled through Thorlabs' Kinesis SDK via the `pythonnet` bridge. The overall pattern (connect → configure → move → read → disconnect) transfers to other motorized stages.

Prerequisites

The Thorlabs Kinesis SDK (which installs the .NET libraries) and the Python package (`pip install pythonnet`). After installing Kinesis, the DLLs live under `C:\Program Files\Thorlabs\Kinesis\`.

One-time hardware setup

The KST101 must be told which stage is attached before Python can talk to it correctly. Using the controller's front-panel menu (with USB unplugged), select the actuator type, then power-cycle the controller to save it. If positions later don't match the front-panel display, this configuration is usually the cause.

1 Finding the Serial Number

Every controller is addressed by its serial number. It's printed on the case and shown on the front panel, but the reliable way — especially with several controllers on one machine — is to ask:

```
import clr

base = "C:\\Program Files\\Thorlabs\\Kinesis\\"
clr.AddReference(base + "Thorlabs.MotionControl.DeviceManagerCLI.dll")
from Thorlabs.MotionControl.DeviceManagerCLI import DeviceManagerCLI

DeviceManagerCLI.BuildDeviceList()
for serial in DeviceManagerCLI.GetDeviceList():
    print(serial)
```

`BuildDeviceList()` scans the USB bus and must run before anything else — it's what populates the list the rest of the API reads from. An empty result means the controller isn't connected, isn't powered, or is held open by another program.

The first two digits identify the controller family (`26` for KST101, `27` for KDC101, and so on), which is a quick way to confirm you're looking at the device you think you are.

2 Connecting

Run the connection code once at the start of a session; keep the `device` object alive until you're completely done (every later operation needs the connection).

```
import clr
import time

base = "C:\\Program Files\\Thorlabs\\Kinesis\\"
clr.AddReference(base + "Thorlabs.MotionControl.DeviceManagerCLI.dll")
clr.AddReference(base + "Thorlabs.MotionControl.GenericMotorCLI.dll")
clr.AddReference(base + "Thorlabs.MotionControl.KCube.StepperMotorCLI.dll")

from Thorlabs.MotionControl.DeviceManagerCLI import DeviceManagerCLI
from Thorlabs.MotionControl.KCube.StepperMotorCLI import KCubeStepper
from System import Decimal

serial_no = "26004813"   # from the controller's front-panel display — replace with
yours

DeviceManagerCLI.BuildDeviceList()
device = KCubeStepper.CreateKCubeStepper(serial_no)
device.Connect(serial_no)
time.sleep(0.5)

if not device.IsSettingsInitialized():
    device.WaitForSettingsInitialized(5000)

device.LoadMotorConfiguration(serial_no)   # required before move/position commands
device.StartPolling(250)                    # required for position updates
time.sleep(0.25)
device.EnableDevice()
time.sleep(0.5)

info = device.GetDeviceInfo()
print(f"Connected to {info.Description} (serial {info.SerialNumber})")
```

The serial number is the 8-digit number on the controller's display (not the number on the motor connector).

2.1 Disconnecting

```
device.StopPolling()
device.Disconnect()
```

3 Reading Position

```
def get_position(device):
    """Current position in mm."""
    try:
        return float(str(device.Position))
    except Exception:
```

```
        return float(device.Position.ToDouble(None))

print(f"Position: {get_position(device):.4f} mm")
```

4 Moving

4.1 Absolute and Relative

```
from System import Decimal
import time

def move_to(device, position_mm, timeout_ms=60000):
    device.MoveTo(Decimal(position_mm), timeout_ms)
    while device.Status.IsMoving:
        time.sleep(0.01)

def move_relative(device, distance_mm, timeout_ms=60000):
    move_to(device, get_position(device) + distance_mm, timeout_ms)

move_to(device, 5.0)        # to absolute 5 mm
move_relative(device, 1.0)  # forward 1 mm
move_relative(device, -0.5) # back 0.5 mm
```

4.2 Homing

Homing sends the stage to its reference position (usually one end of travel). Do this once after connecting if you need absolute positions to be meaningful.

```
import time

def home(device, timeout_ms=60000):
    device.Home(timeout_ms)
    while device.Status.IsHoming:
        time.sleep(0.1)

home(device)
```

4.3 Setting Velocity

```
from System import Decimal

def set_velocity(device, velocity_mm_s=1.0, acceleration_mm_s2=1.0):
    p = device.GetVelocityParams()
    p.MaxVelocity = Decimal(velocity_mm_s)
    p.Acceleration = Decimal(acceleration_mm_s2)
    device.SetVelocityParams(p)

set_velocity(device, 2.0)
```

5 Complete Example: Position Scan

A self-contained scan that steps through positions — the skeleton of an automated measurement. Pair it with a DAQ or VISA read inside the loop to record data at each position (e.g., an automated beam profile or any scan-and-measure experiment).

```
import clr
import time
import numpy as np

base = "C:\\Program Files\\Thorlabs\\Kinesis\\"
clr.AddReference(base + "Thorlabs.MotionControl.DeviceManagerCLI.dll")
clr.AddReference(base + "Thorlabs.MotionControl.GenericMotorCLI.dll")
clr.AddReference(base + "Thorlabs.MotionControl.KCube.StepperMotorCLI.dll")

from Thorlabs.MotionControl.DeviceManagerCLI import DeviceManagerCLI
from Thorlabs.MotionControl.KCube.StepperMotorCLI import KCubeStepper
from System import Decimal

def run_position_scan(serial_no, start_mm, end_mm, step_mm):
    DeviceManagerCLI.BuildDeviceList()
    device = KCubeStepper.CreateKCubeStepper(serial_no)
    device.Connect(serial_no)
    time.sleep(0.5)
    device.WaitForSettingsInitialized(5000)
    device.LoadMotorConfiguration(serial_no)
    device.StartPolling(250)
    time.sleep(0.25)
    device.EnableDevice()
    time.sleep(0.5)

    targets = np.arange(start_mm, end_mm + step_mm, step_mm)
    positions = []
    try:
        for i, target in enumerate(targets):
            device.MoveTo(Decimal(float(target)), 60000)
            while device.Status.IsMoving:
                time.sleep(0.01)
            time.sleep(0.5)   # let vibrations settle before measuring
            actual = float(str(device.Position))
            positions.append(actual)
            # --- read a sensor here and store it alongside `actual` ---
            print(f"Step {i+1}/{len(targets)}: {actual:.4f} mm")
    finally:
        device.StopPolling()
        device.Disconnect()
    return positions

positions = run_position_scan("26004813", 0, 10, 0.5)
```

The `time.sleep(0.5)` after each move lets mechanical vibration settle before you take a reading — important for clean data.

6 Troubleshooting

`Device {serial} not found` — check USB; verify the serial matches the controller display exactly; make sure no other software (Kinesis, APT) holds the controller.

`Object reference not set to an instance of an object` — the stage type isn't configured on the controller; set it via the front-panel menu (see one-time setup above).

`Device settings not initialized` — add `device.LoadMotorConfiguration(serial_no)` after `WaitForSettingsInitialized`.

Positions don't match the display — stage-type mismatch; reconfigure and power-cycle the controller.

Motor won't move — confirm it's enabled (`EnableDevice()`), polling is started (`StartPolling(250)`), the target is within travel limits, and nothing else controls the device.

`ModuleNotFoundError: No module named 'clr'` — `pip install pythonnet`. The package is named `pythonnet` but imports as `clr`, which makes this error look unrelated to what you installed.

`FileNotFoundException` on `clr.AddReference` — the DLL path is wrong. Check that Kinesis is installed and that the path in your script matches where it landed; a non-default install location has to be reflected here.

Positions drift over repeated scans — the stage has backlash. Approach every target from the same direction, or use `SetBacklash` to let the controller compensate. This is the most common cause of a scan that doesn't repeat.

Back to Python resources