THE DIGITAL TWIN · VOLUME 1
BOOK 3 · CODE

The Digital Twin
Source Code & Resources

Complete, Runnable Source for Every Project, with Datasets and a Hardware-Free Simulation Track

Dr. Ahsan Rahman
University of Prince Mugrin · Al Madinah

How to Use This Book

This is the code companion to The Digital Twin: Hands-On Projects (Book 2 of this suite). Every program that appears in that book — forty-five tested listings across Arduino, MATLAB, Python, and PLC Structured Text — is collected here in one place, organised by project, with the file name, the run command, and the dataset each one expects.

You do not need Book 2 to use this one. Each project section opens with a short description of what the code does and how its files fit together, so the listings stand on their own. If you want the full explanation of why a model takes the form it does, that lives in Book 2 (Practice) and Book 1 (Theory); here, the focus is simply on getting the code running.

How this book is organised

The code is grouped into five project folders, exactly mirroring the parts of Book 2:

`part2-dc-motor` the DC-motor twin, Builds A1–A3 (MATLAB, Python, Hybrid).

`part3-led-station` the LED-monitoring twin, Builds B1–B3.

`part4-advanced` IoT telemetry, machine-learning anomaly detection, and remaining-useful-life.

`part5-plc` the PLC twins, Builds A4–B4, in IEC 61131-3 Structured Text plus their Python hosts.

`datasets` five sample CSV files so every analysis and validation script runs without hardware.

 |

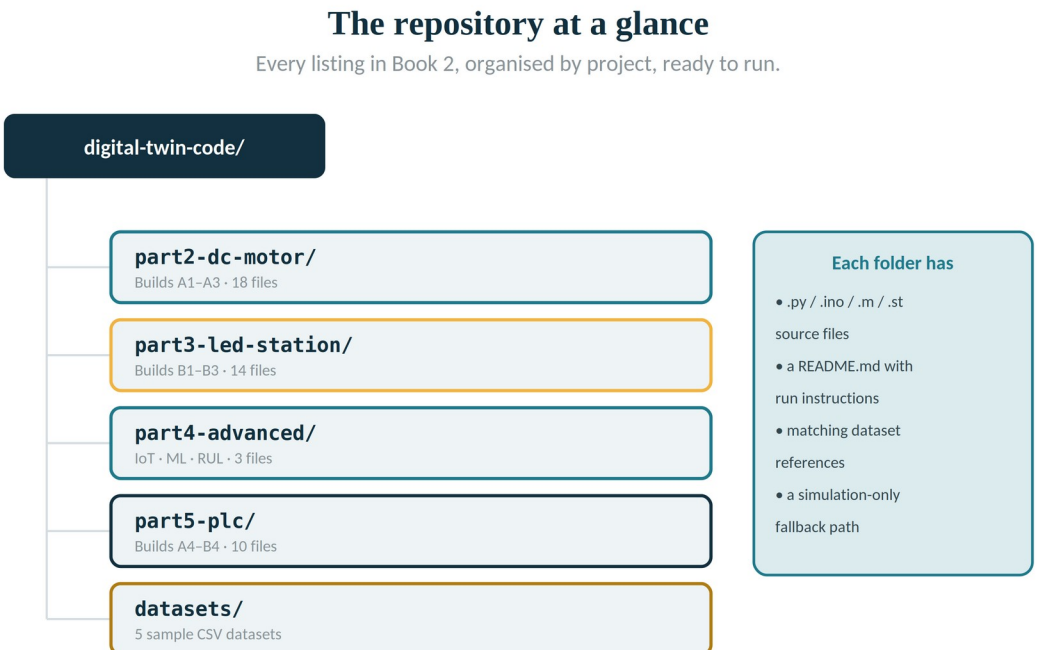

Figure R.1. The repository at a glance. *Every listing in Book 2, organised by project folder and ready to run.*

Getting set up

Most of the Python projects need only a few free packages. Install them once and every Python listing in the book will run:

Install the Python dependencies

```
pip install pyserial numpy scipy matplotlib
pip install pymodbus asyncua        # only for the PLC projects (Part 5)
```

The Arduino sketches (.ino) load in the free Arduino IDE; the MATLAB scripts (.m) run in MATLAB or, with minor changes, GNU Octave; and the PLC Structured Text (.st) runs in the free CODESYS or OpenPLC runtime on a PC or Raspberry Pi. No paid software and no physical PLC are required to work through this book.

Figure R.2. Three ways to run every project. *Choose full hardware, a partial mix, or the bundled-dataset simulation track.*

The simulation-only track

> Every analysis, calibration, identification, and validation script in this book can run against the bundled datasets in the datasets/ folder, with no hardware attached. Where a script normally reads from a live serial port or a PLC, a one-line change points it at the matching CSV instead — each project section says exactly which file to use. This means the entire book can be taught in a classroom, or worked through at home, with nothing but a laptop.

A note on the listings

The code in this book is reproduced verbatim from the validated projects in Book 2. The numeric Python — the motor model, the PID controller, the identification and validation routines, the anomaly detector, and the remaining-useful-life estimator — was executed and checked against the bundled datasets before printing. The Arduino, MATLAB, and PLC listings are written to load and run on their respective free toolchains. Where a listing depends on hardware that not every reader will own, the simulation-only path lets you run an equivalent exercise against recorded data.

Table of Contents

SOURCE CODE · PART TWO

Project 1 — The DC Motor Twin

The DC-motor twin is built three ways in Book 2: with MATLAB and Simulink (Build A1), with Python (Build A2), and as a hybrid of both (Build A3). All three share one Arduino firmware, motor_bench.ino, which exposes a simple text serial contract: the host sends a duty command, the board replies with the measured speed. The files below are everything needed to reproduce all three builds.

Files in part2-dc-motor/

Firmware (Arduino): `bench_check.ino, motor_bench.ino`

Build A1 (MATLAB): `capture_step.m, identify.m, tune.m, run_loop.m, validate.m`

Build A2 (Python): `bench.py, model.py, identify.py, pid.py, run.py, dashboard.py, validate.py`

Build A3 (Hybrid): `orchestrator.py, replay.m, detect.py`

Run it without hardware

To run the Python build against recorded data instead of a live bench, point `identify.py` at `datasets/motor_step_response.csv`, and the validation against `datasets/motor_run_log.csv`. Both files ship with the book.

Firmware

▸ **bench_check.ino**

Listing 7.1 — Arduino smoke test (bench_check.ino)

```
// Minimal bench check: ramp the motor and report encoder counts.
const int PWM_PIN = 5, DIR_PIN = 4, ENC_A = 2, ENC_B = 3;
volatile long encCount = 0;

void onEncoderA() {            // called on every A-channel edge
  if (digitalRead(ENC_A) == digitalRead(ENC_B)) encCount++;
  else                                          encCount--;
}

void setup() {
  Serial.begin(115200);
  pinMode(PWM_PIN, OUTPUT); pinMode(DIR_PIN, OUTPUT);
  pinMode(ENC_A, INPUT_PULLUP); pinMode(ENC_B, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(ENC_A), onEncoderA, CHANGE);
  digitalWrite(DIR_PIN, HIGH); // one fixed direction for the test
}
```

```
void loop() {
  for (int duty = 0; duty <= 255; duty += 5) {  // ramp up
    analogWrite(PWM_PIN, duty); delay(40);
  }
  for (int duty = 255; duty >= 0; duty -= 5) {  // ramp down
    analogWrite(PWM_PIN, duty); delay(40);
  }
  Serial.print("encoder count = "); Serial.println(encCount);
}
```

▸ **motor_bench.ino**

Listing 8.1 — Arduino firmware (motor_bench.ino), shared by A1 and A2

```
// Serial contract:  host sends "u <duty>\n"  (duty 0..255)
//                   board replies "<speed_rad_s>\n" every SAMPLE_MS
const int PWM_PIN = 5, DIR_PIN = 4, ENC_A = 2, ENC_B = 3;
const unsigned long SAMPLE_MS = 20;       // 50 Hz reporting
const float COUNTS_PER_REV = 360.0;       // encoder resolution

volatile long encCount = 0;
long lastCount = 0;
unsigned long lastReport = 0;
int duty = 0;

void onEncoderA() {
  if (digitalRead(ENC_A) == digitalRead(ENC_B)) encCount++;
  else                                          encCount--;
}

void setup() {
  Serial.begin(115200);
  pinMode(PWM_PIN, OUTPUT); pinMode(DIR_PIN, OUTPUT);
  pinMode(ENC_A, INPUT_PULLUP); pinMode(ENC_B, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(ENC_A), onEncoderA, CHANGE);
  digitalWrite(DIR_PIN, HIGH);
}

void loop() {
  if (Serial.available()) {                 // read a command line
    String line = Serial.readStringUntil('\n');
    if (line.startsWith("u ")) {
      duty = constrain(line.substring(2).toInt(), 0, 255);
      analogWrite(PWM_PIN, duty);
    }
  }
  unsigned long now = millis();
  if (now - lastReport >= SAMPLE_MS) {      // report speed
    noInterrupts();
    long c = encCount; interrupts();
    long delta = c - lastCount; lastCount = c;
    float dt = (now - lastReport) / 1000.0;
    float revs = delta / COUNTS_PER_REV;
    float speed = (revs / dt) * 2.0 * 3.14159265; // rad/s
    Serial.println(speed, 4);
    lastReport = now;
  }
}
```