Dyson School of Design Engineering

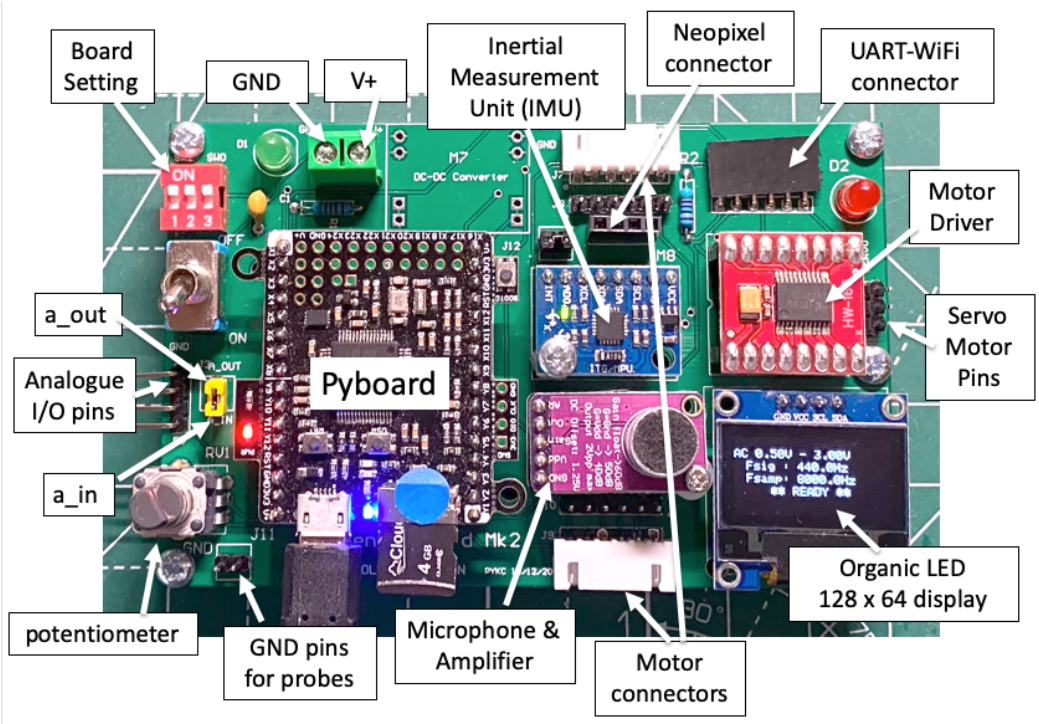
Imperial College London

DE2 Electronics 2

PyBench User's Manual

Overview

PyBench is a specially designed board to support the Electronics 2 module in the second year of our MEng degree programme. The board consist of various module and functions as shown here:

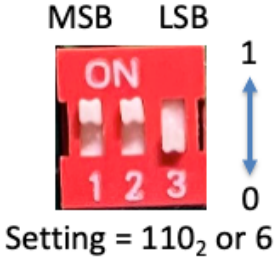


Power Supply

PyBench can be powered either via the MicroUSB connect on the Pyboard, or through the screw terminal (green). V+ is between 3.6V and 16V. For Electronics 2, you will be provided with a 9V battery and clip. This will be used when PyBench is used in an untethered, standalone unit later in the term and will require slight modifications.

Board Configuration Settings

The 3-ways dual-in-package (DIP) switches (red) are used to set the configuration of PyBench as shown below as a 3-bit binary number.



SW[2:0]	Function
000	Run user.py
001	Not used
010	Not used
011	Wifi module Test
100	Spectrum of mic signal
101	Bulb board test
110	Pybench board Test
111	Run pybench.py

When all switches are in the '0' position (down), the system runs the user program with a file name 'user.py'. Settings 1 (001) to 2(010) are not used and can be programmed by user by changing 'main.py'.

Setting 3 is reserved for testing the WiFi module which is currently not installed. Setting 4 (which you used in Lab 1) perform DFT on the microphone signal and displays a live spectrum. Setting 5 (101) is used to test the bulb board (small PCB used for Lab 3 – Understanding Systems). It is used to calibrate the sensitivity of the light detecting diode to the bulb intensity.

Setting 6 (110) is used for self-test. Under this setting, PyBench will enter a self-test mode after pressing the RESET button (left). Thereafter, pressing the USER button will scan through four tests in sequence according to the following:

1. Microphone and amplifier test
2. Inertia Measurement Unit (IMU) test
3. Motor test
4. ADC and DAC test

Setting 7 (111) is the normal setting when conducting Lab Experiments. It runs the PyBench programme 'PyBench_main.py' stored on the microSD card and allow the PyBench board to be controlled via the serial USB interface.

The Pyboard microcontroller module

The Pyboard is a standalone microcontroller system using the ARM microprocessor. It has been preprogrammed to run MicroPython (uPy) natively. Details on the Pyboard can be found on: https://docs.micropython.org/en/latest/pyboard/general.html.

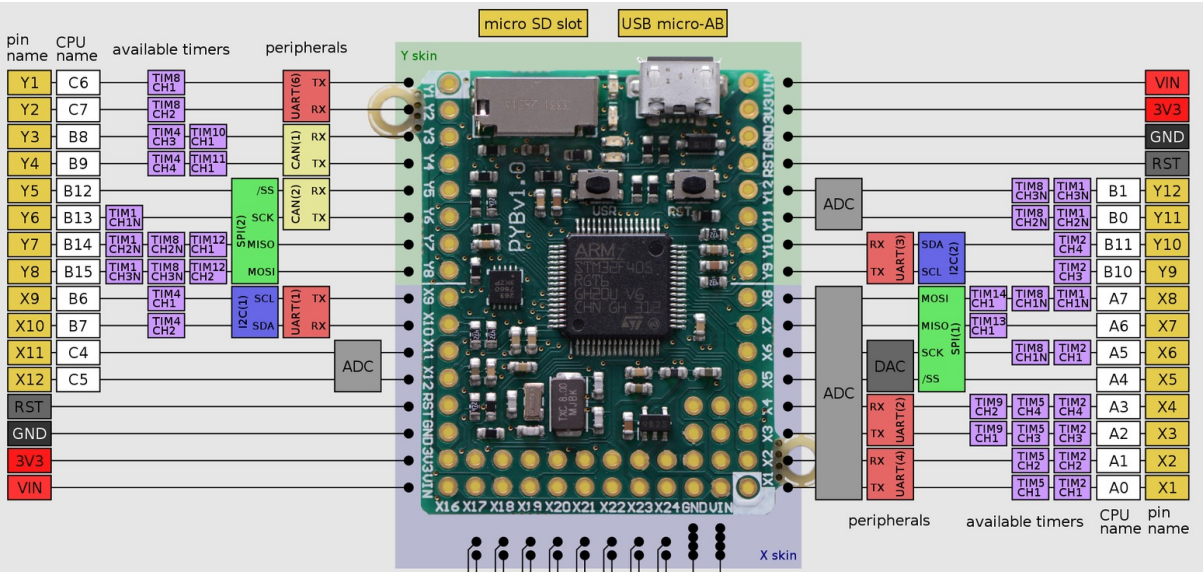
The way that we are using the Pyboard is via the MicroSD card, which contains the PyBench software environment as explained later in this document.

Pyboard brings out many I/O pins, named as X1 to X12 and Y1 to Y12. They are connect to various components on the PyBench PCB according to the following table:

PIN	FUNCTION
X1	Motor PWM_A/Servo 1
X2	Motor PWM_B/Servo 2
X3	Motor control AIN1/Servo 3
X4	Motor control AIN2/Servo 4
X5	Analogue OUTPUT
X6	SW2
X7	Motor control BIN1
X8	Motor control BIN2
X9	IMU-I2C SCL
X10	IMU-I2C SDA
X11	POT10K
X12	Analogue INPUT

PIN	FUNCTION
Y1	DT-06 WIFI Tx
Y2	DT-06 WIFI Rx
Y3	SW1
Y4	Motor sensor A_A
Y5	Motor sensor A_B
Y6	Motor sensor B_A
Y7	Motor sensor B_B
Y8	SW0
Y9	OLED-I2C SCL
Y10	OLED-I2C SDA
Y11	Microphone amplifier
Y12	NEOPIXEL

The physical pin layout of the PYB v1.1 is summarised in the diagram below:



The MicroSD card and PyBench software version 3.0

When you first power up the PyBench system, it tests whether a SD card is present. If yes, it executes the file '**boot.py**' which in turn runs the file '**main.py**'.

'**main.py**' examines the 3-ways switch setting and acts accordingly.

Using PyBench with a computer

You can connect the PyBench system to a computer via the USB port of the PYB microcontroller. The MicroSD card will automatically appear as a **disk drive** on your computer running either Windows 10 or Mac OSX. Not driver installation is required. You can read and write files to the SD card at will.

WARNING: DURING DISK READ AND WRITE, THE RED LED ON THE PYB BOARD WILL FLASH. DO NOT REMOVE THE USB CABLE WHILE THE LED IS FLASHING. DOING SO WILL CORRUPT YOUR SD CARD.

You can examine the content of the SD card installed. It contains the PYBENCH_MK2 environment and the following MicroPython (uPy) files:

Program	Purpose
boot.py	First to run, the boot file specifies which is the main program.
bulb_test.py	Test and calibration program for the small plug in board for Lab3. Run if SW = 101.
echo.py	Test program for wifi module
main.py	Test the DIP switch setting and execute the corresponding .py file.
mic.py	Microphone class library to acquire signals from microphone module.
motor.py	Driver class library for the motor driver chip TB6612 to drive motors.
mpu6050.py	IMU driver class library – to communicate with the accelerometer and gyroscope.
neopixel.py	Neopixel LED strip driver class library.
oled_938.py	OLED display driver class library.
plotspec.py	Plot spectrum of microphone signal on OLED in dB. Fs = 10kHz.
pybench_main.py	The controlling program for PyBench to interpret commands. Run if SW = 111.
pybench.py	The PyBench class library. Can be used in your own application programs later.
pybench_test.py	Self-test program for the PyBench board to verify the hardware. Run if SW = 110.
font.py	Character fonts used by oled_938.py.

DFT package by Peter Hinch:

Program	Purpose
dft.py	Fast DFT algorithm written in ARM assembly instructions.
dftclass.py	MicroPython class wrapper for dft.py.
polar.py	Complex coefficient to magnitude/phase translation.
window.py	Application of a window function to input signal.

To develop your own uPy programmes for PyBench, you would first use something like VSCode or PyCharm to edit y our Python script. You can then directly transfer your uPy programs onto the SD card, which just appears as a disk drive.

Using PyBench with Matlab via a USB cable

PyBench, under the switch setting of 111 (all DIP switches up), can be remotely controlled via a serial USB cable. Coupled with another purposed written PyBench.m Matlab program, the PyBench board becomes a programmable electronics workbench for Matlab. To initialize the system under this setup, the following is required:

1. PyBench.m must be on Matlab's search path.
2. Once the USB cable is installed, the following Matlab script is required:

```
ports = serialportlist;        % find the list of communication ports
pb = PyBench (ports(end));     % create pb as a PyBench object
```

3. Thereafter, the following PyBench methods are available to control the hardware:

Methods	Purpose
pb.set_sig_freq (f)	Set signal frequency to f. 0.1 Hz ≤ f ≤ 3000 Hz
pb.set_samp_freq (f)	Set sampling frequency to f. 1 Hz ≤ f ≤ 30,000 Hz
pb.set_max_v (v)	Set maximum amplitude to v. 0 ≤ v ≤ 3.3
pb.set_min_v (v)	Set minimum amplitude to v. 0 ≤ v ≤ 3.3
pb.set_duty_cycle (d)	Set duty cycle of a square signal to d. 0 ≤ d ≤ 100
pb.dc (v)	Output a dc voltage v. 0 ≤ v ≤ 3.3
pb.sine ( )	Output a sinusoidal signal at set signal frequency between max_v and min_v.
pb.triangle ( )	Output a triangular signal at set signal frequency between max_v and min_v.
pb.square ( )	Output a square signal at set signal frequency between max_v and min_v, with the set duty cycle.
v = pb.get_one ()	Capture one sample v from analogue input. 0 ≤ v ≤ 3.3
data = pb.get_block (n)	Capture n samples from analogue input. 0 ≤ data ≤ 3.3
data = pb.get_mic (n)	Capture n samples from microphone. 0 ≤ data ≤ 3.3
[p, r] = pb.get_accel ( )	Get pitch angle p and roll angle r from the IMU. -90 ≤ p, r ≤ +90
[dx, dy, dz] = pb.get_gyro ( )	Get accelerations (dx, dy, dz) in three axes from the IMU in degrees/sec.

Note that when using PyBench with Matlab, you may find that serial communication may be lost or, in a tight Matlab infinite loop, you cannot regain control of Matlab. If this happens, you can do the following:

1. Type CTRL+C in Command Window. If this still does not work, exit Matlab and unplug PyBench cable. Re-start Matlab and connect the USB cable again.
2. Type CTRL+C in Command Window. Enter `fclose(instrfind());` or `fclose(pb.usb);` will close the serial port.