Verificare il Green Pass europeo

Da una settimana, il Green Pass (che dimostra l'immunità o comunque il non contagio dal Covid-19) è obbligatorio per entrare in buona parte degli edifici in tutta Italia. Molti gestori di attività che si svolgono al chiuso sono preoccupati di non riuscire a controllare i Green Pass del pubblico. Ma è davvero così difficile? In realtà, no. L'EU Digital COVID Certificate è infatti un certificato digitale, le cui specifiche sono pubblicamente disponibili proprio per consentire a chiunque di verificare la validità di uno di questi QR code.

Esistono delle app per smartphone che svolgono questa operazione, ma richiederebbero comunque un operatore che scansioni i vari QR code degli avventori di un locale, e non sempre questo è praticabile. Esiste, però, la possibilità di automatizzare tutto il procedimento, gestendo l'accesso all'edificio tramite un meccanismo come un tornello, o una porta azionabile elettronicamente, e un semplice programma in Python che possiamo scrivere in breve tempo. Utilizzando un computer dotato di pin GPIO come il RaspberryPi è possibile realizzare un sistema completamente automatico: si può usare una webcam per riconoscere il QRcode, un lettore di smartcard per confrontare i dati con quelli della Tessera Sanitaria, e un relay per attivare il tornello (o la porta) soltanto nel caso in cui il Green Pass risulti valido. Il ricorso alla Tessera Sanitaria è fondamentale perché altrimenti un utente potrebbe presentarsi alla porta d'ingresso con un QRcode appartenente a un'altra persona, magari fotografato e condiviso tra tanti utenti. La Tessera Sanitaria è invece una sola per ogni cittadino, quindi permette di identificare automaticamente le persone e assicurarsi che ogni accesso sia legittimo. Naturalmente si potrebbero utilizzare altri

meccanismi, come la CIE o la Firma Digitale, ma la Tessera Sanitaria è l'unico ID digitale a disposizione di tutti i cittadini italiani. Il progetto che proponiamo si basa su un RaspberryPi2 o superiore, con RasperryOS Buster, una webcam USB, un lettore di smartcard USB, un altoparlante passivo con jack audio, e un eventuale modulo relay per far scattare l'apertura della porta.

Table of Contents

Installare i requisiti

I requisiti di questo software sono parecchi, ma possiamo installarli con una serie di comandi. Da notare che ci serve lo script https://github.com/panzi/verify-ehc, e quindi dovremo anche installare tutti i suoi requisiti. Possiamo farlo con questa serie di comandi:

In poche parole, prima di tutto installiamo le varie librerie necessarie per leggere le smartcard, poi quelle necessarie per prelevare immagini dalla webcam e manipolare le immagini (utilizzeremo OpenCV e PIL). Poi serviranno anche le librerie per gestire i pin GPIO del Raspberry, in modo da attivare un relay, e quelle per decodificare i QRCode. Infine, la libreria per emettere suoni (beepy) e i vari requisiti di Verify EHC.

Riconoscere il QRCode

Per prima cosa, scriveremo il codice che ci serve per riconoscere il QRcode, cioè per ottenere il testo (che poi tradurremo in JSON). Qui è necessario un piccolo hack: al momento, la versione di Debian disponibile come RaspberryOs è Buster. Purtroppo questa versione è ormai molto vecchia, quindi non è possibile avere le ultime versioni dei pacchetti. E la libreria qrtool è disponibile solo per Python2 (si trova su apt come **python-qrtools**). Esistono ovviamente divese altre librerie, ma questa è quella che abbiamo notato essere più efficiente e semplice da utilizzare. Quindi per ora metteremo le sue poche righe di codice in uno script a parte, che verrà interpretato da Python2 invece che da Python3: in futuro, usando la nuova versione di Debian, sarà possibile usare questa libreria nella versione per Python3. Il codice è questo:

Il codice è estremamente semplice: lo script si aspetta in argomento il percorso di un file contenente l'immagine di un QRcode da interpretare. Con questo file si può costruire un oggetto QR, decodificabile con la funzione **decode()**. A questo punto la variabile **.data** contiene il testo che è stato riconosciuto nel QRCode, che possiamo restituire sullo standard output. E che andremo a leggere dal programma principale.

Il Green Pass è un testo

firmato con le informazioni del paziente memorizzate in un JSON

Le funzioni di servizio

Ora iniziamo a scrivere lo script principale, quello che si occuperà di svolgere tutto il processo di verifica sia del Green Pass che della Tessera Sanitaria. Cominciamo dall'importazione delle librerie:

Librerie extra sono sostanzialmente quelle a cui accennavamo per lo script di installazione delle dipendenze, poi servono alcune librerie standard di Python per la gestione del JSON, dei sottoprocessi e dell'orario.

Continuiamo definendo alcuni oggetti che saranno utili per tutto lo script, e che quindi avranno valore globale. Per esempio, il percorso in cui trovare il file di configurazione, oppure quello in cui scrivere i log. Poi proviamo a importare le librerie per gestire i pin GPIO del RaspberryPi: saranno necessarie per attivare il relay, e quindi aprire automaticamente la porta o il tornello nel caso il Green Pass risulti valido. In realtà possiamo anche utilizzare lo script su un computer diverso dal Raspberry, e in quel caso non riusciremmo a importare le librerie dei GPIO. In questo caso impostiamo la variabile **rpi** a False, così sapremo che non ci troviamo su un Raspberry. Creiamo un dizionario vuoto per memorizzare la configurazione, e poi l'oggetto **reader**: questo rappresenterà il nostro punto di accesso al lettore di smartcard. Siccome dovrebbe essere possibile procedere anche senza il lettore, perché l'utente potrebbe decidere di usare solo la webcam per il riconoscimento del QR code e poi lasciare a un operatore l'identificazione della persona, se non riusciamo a trovare il lettore di Smart Card catturiamo

l’eccezione e andiamo avanti comunque.

Definiamo due funzioni “di servizio”, non fondamentali ma utili per definire due procedure e non preoccuparsene più. La prima si occupa di leggere il file di configurazione, che sarà nel formato JSON, e memorizzare il contenuto in un dizionario. La seconda è quella che apre la porta facendo scattare il relay: ci serve il numero del pin GPIO da attivare, ma vogliamo anche assicurarci di essere davvero su un Raspberry, perché altrimenti non ci sono i GPIO e non dobbiamo fare nulla.

Decodificare il Green Pass

Iniziamo con le cose “serie”: lo script **verify-ehc.py** si occupa di decodificare la stringa del Green Pass (che è un testo codificato in Base45).

Lo chiamiamo direttamente con os.system, scrivendo l’output in un file. Poi leggiamo il file, memorizzandolo come testo in una variabile. Utilizziamo os.system perché il modulo subprocess ha difficoltà a leggere tutte le righe, dal momento che lo script scrive l’output a intervalli non regolari.

L’output è diviso su più righe, e in realtà a noi interessano solo alcune. Nello specifico, ci interessa la riga **Is Expired** che, se presente, indica che il certificato era valido, ma ora è scaduto. E poi la riga **Signature Valid**, che è presente solo se la firma del certificato risulta corretta: questa indica che il certificato è stato generato da uno dei ministeri della salute dell’Unione Europea, e quindi possiamo considerarlo non contraffatto. Infine, cerchiamo la riga **Payload**, perché dopo di essa viene riportato l’intero contenuto del Green Pass vero e proprio, con i dati personali della persona. Questo payload è in formato JSON, quindi possiamo tranquillamente caricarlo

in un dizionario, assicurandoci di prendere il testo e rimuovere gli invii a capo per evitare che il modulo json di Python possa avere difficoltà a interpretarlo.

Leggere la Tessera Sanitaria

Purtroppo non esiste una documentazione pratica per l'utilizzo delle informazioni presenti nella Tessera Sanitaria italiana, solo delle specifiche tecniche. Noi ci siamo basati sul lavoro di decodifica fatto alcuni anni fa da MMXForge.

I dati su una tessera sanitaria sono memorizzati in un particolare filesystem, ed è possibile selezionare i file inviando una serie di comandi binari (che codifichiamo in esadecimale per leggiblità). La funzione per la lettura dei dati personali dalla tessera sanitaria deve quindi iniziare stabilendo una connessione con la smartcard e poi utilizzando quella connessione per inviare una serie di comandi.

La Tessera Sanitaria italiana contiene un microchip leggibile con un comune lettore di smartcard

Otteniamo come risposta una tupla di tre oggetti: il primo rappresenta i dati restituiti dalla smartcard, gli altri due eventuali codici per identificare errori. Se tutto va bene, sw1 e sw2 dovrebbero sempre contenere i valori 0x90 e 0x00 rispettivamente. Nel nostro caso non c'è bisogno di

verificarli, perché siamo solo interessati a estrarre i dati dal file corretto, qualsiasi cosa vada storta indica che la tessera inserita non era corretta e possiamo considerare nulla l'identificazione.

A questo punto, la variabile data contiene tutti i dati dell'utente, ma come byte. Dobbiamo convertirla in stringa e poi estrarre i singoli dati. I dati sono codificati in modo abbastanza semplice: i primi due caratteri contengono il numero di byte che costituiscono il successivo dato, così sappiamo sempre esattamente quando leggere. Quindi dobbiamo leggere i primi due caratteri, trasformarli in un numero intero, e leggere quel numero di byte per estrapolare il codice dell'emettitore della tessera. Poi leggiamo i due caratteri successivi per conoscere il numero di byte da leggere per avere il cognome. Segue il nome dell'intestatario della tessera.

Con la stessa logica possiamo continuare a leggere i dati personali dell'utente. Sono, in sequenza, sesso, statura, codice fiscale, cittadinanza, comune di nascita e stato di nascita (nel caso la persona non sia nata in Italia). Memorizziamo tutti questi dati in un dizionario, così sarà più facile accedere a quello che ci interessa.

Verificare se il certificato è valido

Iniziamo ora la funzione che ci permetterà di stabilire se il Green Pass sia valido.

I due oggetti che dobbiamo ricevere in argomento sono il dizionario con i dati del green pass e quello con i dati della tessera sanitaria. Possiamo considerare il green pass immediatamente non valido se dai suoi stessi dati risulta che

sia scaduto (expired) o se la sua firma non risulti correttamente apposta da uno dei ministeri della salute europei (in questo caso **signature_valid** sarebbe **False**).

Se è stata fornita una Tessera Sanitaria valida, possiamo confrontare i suoi dati con quelli del Green Pass. Dobbiamo solo fare una piccola conversione sulla data di nascita, perché nella TS è memorizzata nel formato GG/MM/YYYY, mentre nel GP è memorizzata come AAAA-MM-GG. Poi possiamo confrontare data di nascita, nome, e cognome: li confrontiamo in minuscolo, per evitare problemi con eventuali lettere mauscole non corrispondenti.

Se non è stata fornita una tessera sanitaria, per esempio perché la persona non è un cittadino italiano, e la configurazione consente comunque all'operatore di verificare l'identità della persona, facciamo apparire un semplice prompt per chiedere proprio all'operatore se il Green Pass appartenga davvero alla persona che si è presentata. Se l'operatore preme i tasti **y** oppure **s**, il Green Pass è considerato legittimo, ma segnaliamo comunque che non era stata fornita una tessera sanitaria. Così nell'eventuale log viene indicato che l'identificazione è stata manuale.

Per leggere un QR code basta una comune webcam, anche non FullHD

Catturare il QRcode dalla webcam

Per catturare il QRcode creiamo una funzione che utilizza OpenCV, così è facile scattare foto in tempo reale dalla webcam.

L'immagine verrà inserita nella variabile **img**.

Ora utilizziamo OpenCV per scrivere l'immagine su un file temporaneo (sempre lo stesso, tanto possiamo gestire un solo ingresso alla volta). Poi cerchiamo di tradurre questa immagine nel testo del GreenPass usando lo script qrcodereader. Non utilizziamo direttamente a funzione di lettura del QR code di OpenCV perché non funziona bene con webcam a bassa definizione. Se abbiamo ottenuto qualcosa, lo scriviamo nella variabile **data**.

Se è disponibile una sessione di Xorg, il server grafico di GNU-Linux, mostriamo una finestra con l'anteprima della foto scattata dalla webcam, così l'utente può capire se ha allineato correttamente il QR code. Chiaramente non possiamo farlo quando non c'è uno schermo. La funzione fa un ciclo continuo finché non viene riconosciuto un QRcode valido.

Mettere tutto assieme

Nella routine principale del programma possiamo riunire le varie funzioni che abbiamo scritto seguendo il filo logico della verifica del Green Pass: lettura del QRcode, lettura della tessera, confronto dei dati, responso all'utente sotto forma di segnale audio, apertura della porta, e eventuale messaggio sullo schermo.

Nella routine principale del programma prima di tutto leggiamo la configurazione dall'apposito file. Poi attiviamo un ciclo continuo, che svolgerà le varie operazioni in sequenza: prima di tutto si riproduce un suono, per segnalare che siamo pronti a leggere un nuovo QRcode. Poi procediamo a avviare la funzione per la lettura delle immagini dalla webcam: quando questa avrà identificato e decodificato un QR code, potremo procedere a verificarne il contenuto. Fatto questo, andiamo a leggere l'eventuale Tessera Sanitaria presente nel lettore (se la tessera non è stata inserita, il dizionario risultante sarà vuoto).

Ora abbiamo tutto quello che potrebbe servirci, quindi possiamo procedere alla verifica delle credenziali. Come abbiamo visto, la funzione **isCertValid** ci restituisce una tupla di due oggetti. Il primo è un semplice booleano, chiamato **val**, che sarà True se il Green Pass è valido e corrispondente alla Tessera Sanitaria e False negli altri casi. Mentre **err** è una stringa che contiene l'eventuale codice di errore ottenuto. Se il Green Pass è valido non soltanto lo segnaliamo con un suono, così è subito palese se l'accesso sia consentito oppure no, ma inneschiamo anche l'apertura della porta o del tornello con la funzione **open_door**.

Per finire, gestiamo il caso in cui la configurazione preveda di loggare i dati, per esempio per identificare. Ovviamente questa è una eventualità che richiede una certa cautela, perché si tratta di memorizzare dati privati sensibili delle persone, quindi non è detto che qualcuno voglia attivarla. Se il log è attivo, quindi, generiamo una riga di log costituita dal timestamp, lo stato della validità del green pass (**OK** oppure **ERROR**), l'eventuale codice fiscale (che però è una stringa vuota se non è stata fornita una Tessera Sanitaria), e l'eventuale messaggio di errore.
Prima di ripetere il ciclo attendiamo un secondo, per dare all'utente il tempo di togliere la propria tessera sanitaria e il QRcode dai lettori. Poi siamo pronti per un altro ciclo,

verificando le credenziali di un’altro avventore..

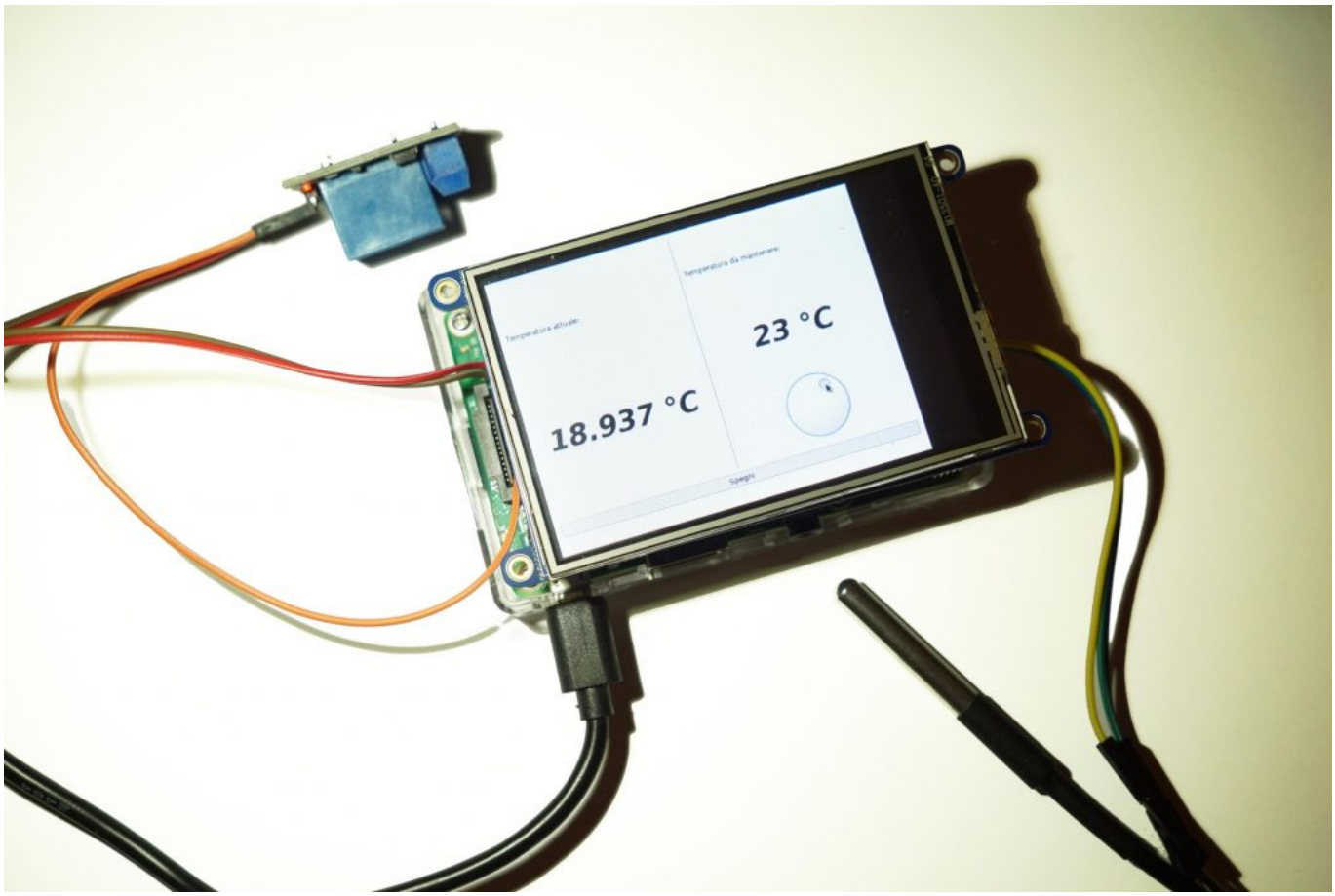
Il codice sorgente
Potete trovare il codice sorgente di questo strumento su GitHub:
https://github.com/zorbaproject/greenpass-turnstile
Il repository integra già la dipendenza https://github.com/panzi/verify-ehc, lo script che esegue la decodifica del Green Pass.

Un termostato touchscreen con RaspberryPi

I RaspberryPi sono una ottima piattaforma su cui costruire oggetti basati sull’idea dell’Internet of Things: dispositivi per uso più o meno domestico che siano connessi a reti locali o ad internet e possano essere facilmente controllati da altri dispositivi. Il vantaggio di un RaspberryPi rispetto a un Arduino è che è più potente, e permette quindi una maggiore libertà. E non solo in termini di collegamento al web, ma anche per quanto riguarda le interfacce grafiche. È un dettaglio del quale non si parla molto, perché tutti danno per scontato che un Raspberry venga usato solo come server, controllato da altri dispositivi con una interfaccia web, e non collegato a uno schermo. Ma non è sempre così. Per esempio, possiamo utilizzare un RaspberryPi per realizzare un termostato moderno. In questo caso non abbiamo più di tanto bisogno di accedervi dallo smartphone: sarebbe utile, ma di sicuro non è l’utilizzo principale che si fa di un termostato. Basterebbe avere uno schermo touchscreen, con una bella

grafica, che si possa fissare al muro per regolare la temperatura. Il punto su cui molti ideatori dell'IoT perdono parte del proprio pubblico è il mantenere le cose "con i piedi per terra". La gente, infatti, è abituata a regolare la temperatura della propria casa da un semplice pannello attaccato al muro, ed è questo che vuole. Magari un po' più futuristico e gradevole alla vista, ma comunque non troppo diverso da quello a cui si è già abituati. Non tutto ha bisogno di essere connesso al web, soprattutto considerando che è un pericolo: se il nostro termostato è raggiungibile da internet, prima o poi un pirata russo si divertirà ad abbassare la temperatura di casa nostra fino a farci raggiungere un congelamento siberiano. La strada più semplice e immediata, a volte, è la migliore. La domanda a questo punto è: come si realizza una interfaccia grafica per Raspberry? Esistono diverse opzioni, ovviamente, ma quella che abbiamo scelto è PySide2. Si tratta della versione Python delle librerie grafiche Qt5, le più comuni librerie grafiche cross platform. È l'opzione migliore semplicemente perché sono disponibili per la maggioranza delle piattaforme e dei sistemi operativi esistenti, quindi i progetti che realizziamo con queste librerie possono funzionare anche su dispositivi differenti dal Raspberry, e ciascuno può adattare il codice alle propri esigenze con poche modifiche.

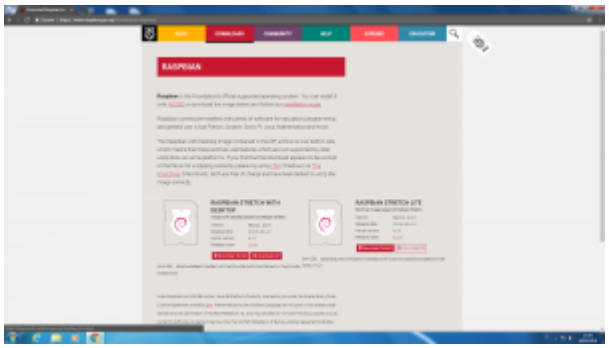
Il nostro dispositivo di riferimento sarà un RaspberryPi 3 B+, dal costo di 50 euro, con uno schermo TFT di Adafruit da 3.5 pollici. Come sistema operativo, si può utilizzare la versione di **Raspbian Buster con PySide2** preinstallato realizzata per Codice Sorgente (https://www.codice-sorgente.it/raspbian-buster-pyside2-lxqt/).



Ma le procedure presentate possono funzionare anche per il Raspberry Pi Zero W, con lo stesso schermo, bisogna solo aspettare che venga pubblicata la versione Buster di Raspbian per questo dispositivo (prevista tra qualche mese).

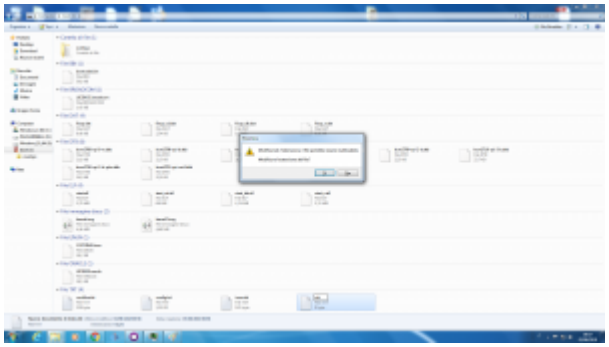
Preparare la scheda sd

Per prima cosa si scarica l'immagine di Raspbian: al momento si può recuperare dal sito ufficiale la versione Raspbian Stretch per qualsiasi tipo di Raspberry. Se invece avete un Raspberry Pi 3 o 3B+ (o anche un Raspberry Pi 2) potete utilizzare **la versione che ho realizzato personalmente di Raspbian Buster, con le librerie Qt già installate: https://www.codice-sorgente.it/raspbian-buster-pyside2-lxqt/**. Questa è la strada consigliabile, visto che Raspbian Stretch è molto datato e non è possibile installare le librerie Qt più recenti, su cui si basa il progetto di questo articolo.



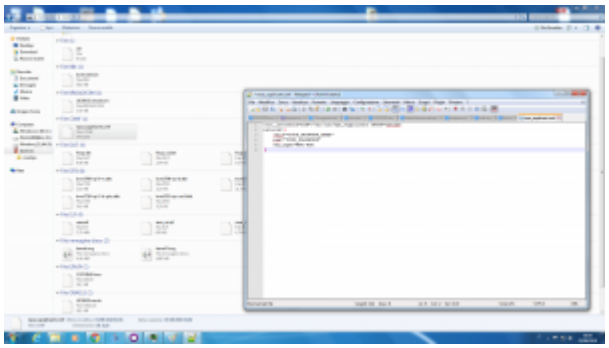
Ottenuta l'immagine del sistema operativo, la si scrive su una scheda microSD con il programma Win32Disk Imager:

Dopo il termine della scrittura, si può inserire nella scheda SD il file **ssh**, un semplice file vuoto senza estensione (quindi ssh.txt non funzionerebbe):

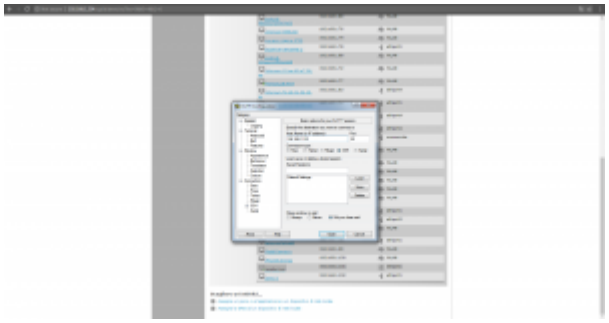


Sempre nella scheda SD si può creare il file di testo **wpa_supplicant.conf**, inserendo un testo del tipo

Bisogna però assicurarsi che il file abbia il formato Unix per il fine riga (LF, invece del CRLF di Windows). Lo si può stabilire con Notepad++ o Kate, degli editor di testo decisamente più avanzati di Blocco Note:



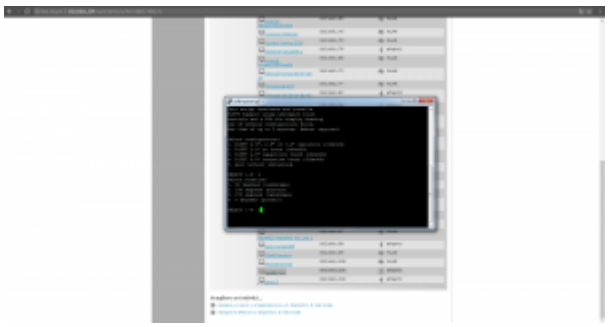
Dopo avere collegato il proprio Raspberry all'alimentazione (e eventualmente alla rete, se si sta usando l'ethernet), si può accedere al terminale remoto conoscendo il suo indirizzo IP. L'indirizzo Il programma che simula un terminale remoto SSH su Windows si chiama Putty, basta indicare l'indirizzo e premere **Open**. Quando viene richiesto il login e la password, basta indicare rispettivamente **pi** e **raspberry**. Per chi non lo sapesse, la password non compare mentre le si scrive, come da tradizione dei sistemi Unix.



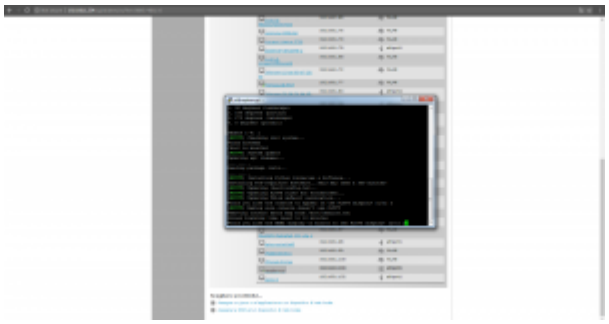
Se il login è corretto si accede al terminale del Raspberry:

Per configurare il TFT di Adafruit bisogna dare i seguenti comandi:

Quando la configurazione inizia, vengono richieste due informazioni: una sul modello di schermo che si sta usando (nell'esempio il numero 4, quello da 3.5 pollici), e una sull'orientamento con cui è fissato sul Raspberry (tipicamente la 1, classico formato orizzontale).



Alla fine dell'installazione dei vari software necessari, viene anche richiesto come usare lo schermo: alla prima richiesta, quella sulla console, conviene rispondere **n**, mentre alla seconda (quella sul mirroring HDMI) si deve rispondere **y**. In questo modo sullo schermo touch verrà presentata la stessa immagine che si può vedere dall'uscita HDMI.



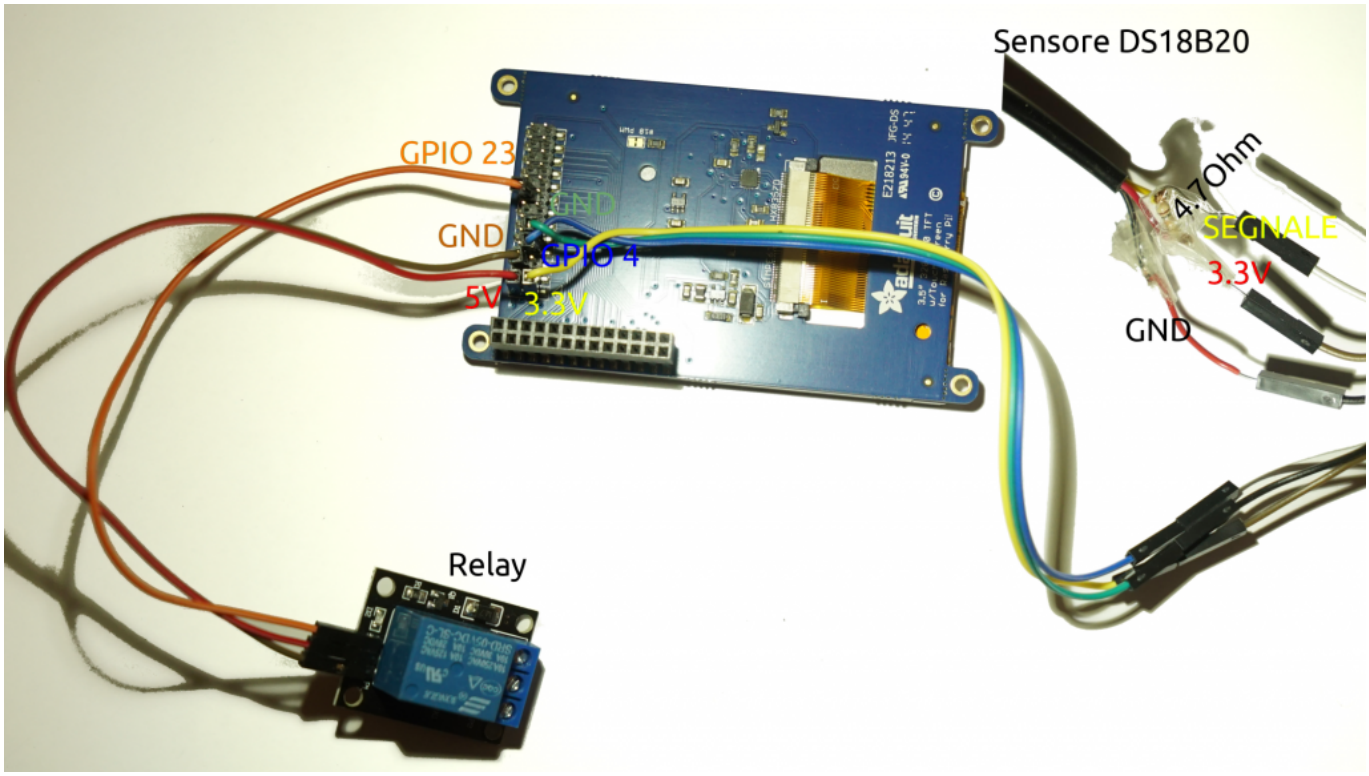
Un appunto (temporaneo) su Raspbian Buster
Se state usando l'immagine che abbiamo fornito all'inizio dell'articolo, basata su Raspbian Stretch, vi sarete accorti che lo script di Adafruit non funziona. Questo perché al momento il pacchetto tslib, necessario per lo script, non è disponibile per Buster, visto che si tratta di una versione non stabile. Per aggirare il problema, si può utilizzare

questa versione modificata dello script: https://codice-sorgente.it/cgit/termostato-raspberry.git/tree/adafruit-pitft.sh. Basta scaricarlo con il comando

Le altre istruzioni non cambiano. In futuro, quando Raspbian Buster verrà rilasciato ufficialmente, non sarà necessario usare questa versione modificata e si potrà fare riferimento all'originale.

I collegamenti del relè e del termometro

Quando si monta lo schermo sul Rasberry, i vari pin della scheda vengono coperti. È tuttavia possibile continuare a usarli perché lo schermo ce li ha doppi. Il sensore di temperatura e il relay potranno quindi essere collegati direttamente ai pin maschi che si trovano sullo schermo, seguendo lo stesso ordine dei pin sul Raspberry.



Ovviamente, un Raspberry offre molti pin, per collegare sensori e dispositivi, ma bisogna stare attenti a non usare lo stesso pin per due cose diverse. Per esempio, se stiamo usando il TFT da 3.5 pollici, possiamo verificare sul sito pinout che i contatti che usa sono il **18,24,25,7,8,9,10,11**. Rimane quindi perfettamente libero sul lato da 5Volt il pin 23, mentre sul lato a 3Volt è libero il pin 4. Il primo verrà usato come segnale di output per il relay, mentre il secondo come segnale di input del termometro. Il relay va collegato anche al pin 5V e GND, mentre il sensore va collegato al pin 3V e al GND.

La libreria per accedere ai pin GPIO dovrebbe già essere presente su Raspbian, mentre per installare quella relativa al sensore di temperatura si può dare il comando

Bisogna anche abilitare il modulo nel sistema operativo:

Dopo il riavvio diventa possibile utilizzare la libreria w1 per l'accesso al sensore di temperatura.

Un codice di test dal terminale

Possiamo verificare il funzionamento del relay e del termometro con un programma molto semplice da eseguire sul terminale:

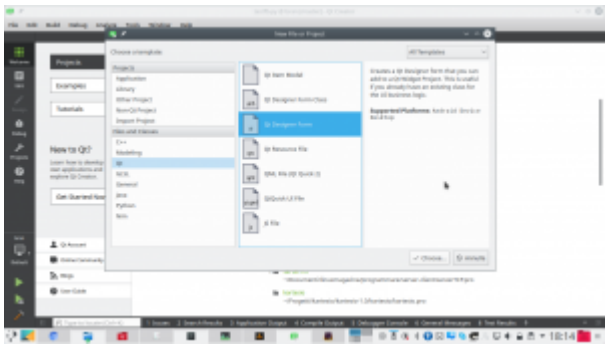
Per provare il programma basta dare i seguenti comandi:

Che cosa fa questo programma? Prima di tutto si assicura che siano caricati i moduli necessari per accedere ai pin GPIO e al sensore di temperatura. Poi esegue un ciclo infinito accendendo il relay, leggendo la temperatura attuale, aspettando 5 secondi, spegnendo il relay, leggendo ancora la

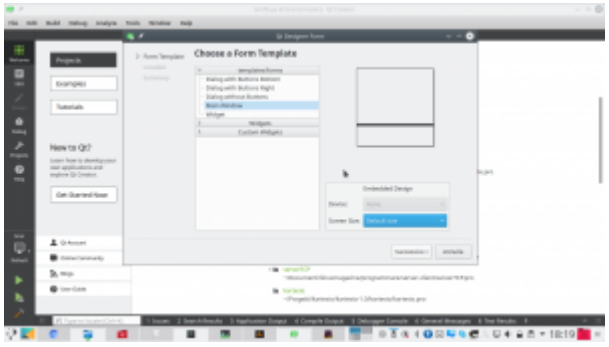
temperatura, e attendendo altri 5 secondi. Per terminare il programma, basta premere **Ctrl+C**.

L’interfaccia grafica

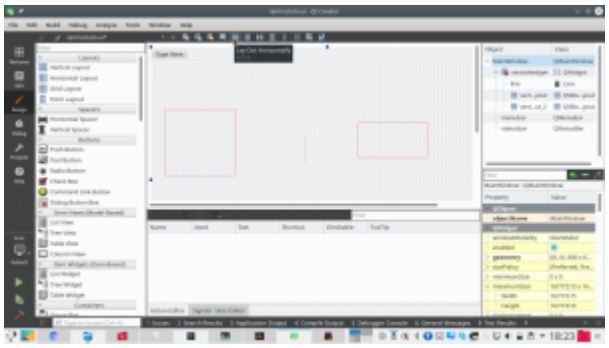
L’interfaccia grafica del nostro termostato è disponibile, assieme al resto del codice, nel repository Git: https://codice-sorgente.it/cgit/termostato-raspberry.git/tree/termostato.ui. Tuttavia, per chi volesse disegnarla da se, ecco i passi fondamentali. Prima di tutto si apre l’IDE QtCreator, creando un nuovo file di tipo Qt Designer Form.



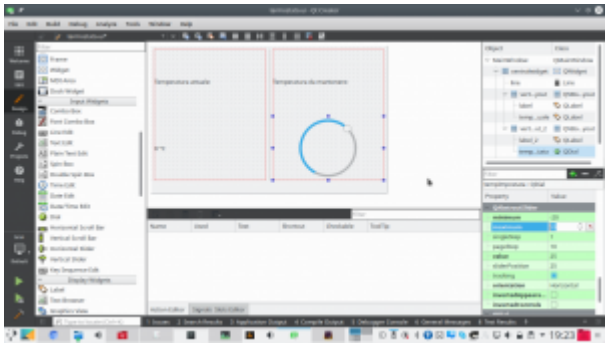
Il template da utilizzare è Main Window, perché quella che andiamo a realizzare è la classica finestra principale di un programma. Per il resto, la procedura guidata chiede solo dove salvare il file che verrà creato.



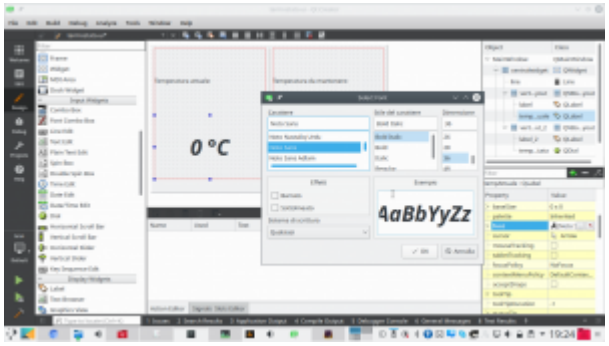
Quando si disegna una finestra, la prima cosa da fare è dividere il contenuto in layout. Considerando il nostro progetto, possiamo aggiungere due oggetti **Vertical Layout**, affiancandoli. Poi, con la barra degli strumenti in alto, impostiamo il form con un **Layout Horizontally**. I due layout verticali sono ora dei contenitori in cui possiamo cominciare ad aggiungere oggetti.



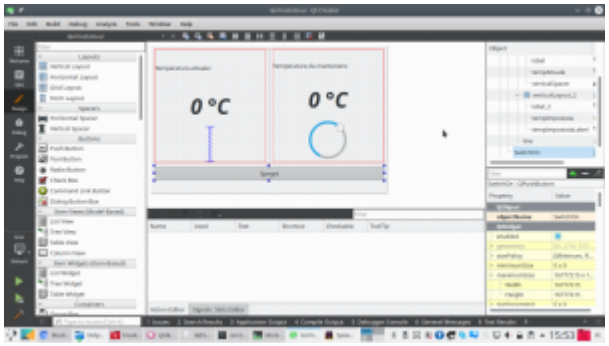
Inseriamo un paio di **label**: in particolare, una dovrà essere chiamata **tempAttuale**, e un'altra **tempImpostataLabel**. Queste etichette conterranno rispettivamente il valore della temperatura attualmente registrata dal termometro e quello che si è deciso di raggiungere (cioè la temperatura da termostatare). Nello stesso layout di **tempImpostataLabel** inseriamo un oggetto **Dial**, che chiamiamo **tempImpostata**. Si tratta di una classica rotella, proprio come quelle normalmente presenti sui termostati fisici. Tra le proprietà di questa dial, indichiamo i valori che desideriamo come minimo (**minimum**), massimo (**maximum**), e predefinito **(value).** Poi possiamo cliccare col tasto destro sulla **menubar** del form e scegliere di rimuoverla, così da lasciare spazio ai vari oggetti dell'interfaccia, tanto non useremo i menù.



Cliccando col tasto destro sui vari oggetti, è possibile personalizzarne l'aspetto. Per esempio, può essere una buona idea dare alle etichette che conterranno le due temperature una formattazione del testo facilmente riconoscibile, con una dimensione del carattere molto grande.



Infine, si può aggiungere, dove si preferisce ma sempre usando dei layout, un **Push Button** chiamato **SwitchOn**. Questo pulsante servirà per spegnere il termostato manualmente, in modo che il relay venga disattivato a prescindere dalla temperatura. Questa è una buona idea, perché se si sta via da casa per molto tempo non ha senso che il termostato scatti per tenere la casa riscaldata sprecando energia. Il pulsante che abbiamo inserito deve avere le proprietà **checkable** e **checked** attivate (basta cliccare sulla spunta), in modo da farlo funzionare non come un pulsante ma come un interruttore, che mantiene il proprio stato acceso/spento.



L'interfaccia è ora pronta, tutti i componenti fondamentali sono presenti. Per il resto, è sempre possibile personalizzarla aggiungendo altri oggetti o ridisegnando i layout.

Il codice del termostato

Cominciamo ora a scrivere il codice Python che permetterà il funzionamento del termostato:

All'inizio del file si inserisce la classica shebang, il

cancelletto con il punto esclamativo, per indicare l'interprete da usare per avviare automaticamente questo script Python3 (che non va quindi confuso col vecchio Python2). Si indica anche la codifica del file come utf-8, utile se si vogliono usare lettere accentate nei testi. Poi, si importano le varie librerie che abbiamo già visto essere utili per accedere ai pin GPIO del Raspberry, al termometro, e alle funzioni di sistema per misurare il tempo.

Ora dobbiamo aggiungere le librerie PySide2, in particolare quella per la creazione di una applicazione (**QApplication**). In teoria potremmo farlo con una sola riga di codice. In realtà, è preferibile usare questo sistema di blocchi try-except, perché lo utilizziamo per installare automaticamente la libreria usando il sistema di pacchetti **pip**. In poche parole, prima di tutto si prova (try) a importare la libreria **QApplication**. Se non è possibile (except), significa che la libreria non è installata, quindi si utilizza l'apposita funzione di pip per installare automaticamente la libreria. E siccome ci vorrà del tempo è bene far apparire un messaggio che avvisi l'utente di aspettare. È necessario usare due formule differenti perché, anche se al momento nella versione 3.6 di Python il primo codice funziona, in futuro sarà necessario utilizzare la seconda forma. Visto che la transizione potrebbe richiedere ancora qualche anno, con sistemi che usano ancora la versione 3.6 e altri con la 3.7, è bene avere entrambe le opzioni. Il vero vantaggio di questo approccio è che se si sta realizzando un programma realizzato con alcune librerie non è necessario preoccuparsi di distribuirle col programma: basta lasciare che sia Python stesso a installarla al primo avvio del programma. L'installazione è comunque possibile solo per le piattaforme supportate dai rilasci ufficiali su pip della libreria, e nel caso di Raspbian conviene sempre installare le librerie a parte.

Se l’importazione della libreria QApplication è andata a buon fine, significa che PySide2 è installato correttamente. Quindi possiamo importare tutte le altre librerie di PySide che ci torneranno utili nel programma.

Definiamo una variabile globale da usare per tutte le prossime classi: questa variabile, chiamata **toSleep**, contiene l’intervallo (in secondi) ogni cui controllare la temperatura.

Per prima cosa creiamo una classe di tipo QThread. Ovviamente, i programmi Python vengono sempre eseguiti in un unico thread, quindi se il processore è impegnato a svolgere una serie di calcoli e operazioni in background (come il raggiungimento della temperatura) non può occuparsi anche dell’interfaccia grafica. Il risultato è che l’interfaccia risulterebbe bloccata, un effetto sgradevole per l’utente e poco pratico. La soluzione consiste nel dividere le operazioni in più thread: il thread principale sarà sempre assegnato all’interfaccia grafica, e creeremo dei thread a parte che possano occuparsi delle altre operazioni. Creare dei thread in Python non è semplicissimo, ma per fortuna usando i QThread diventa molto facile. Il QThread che stiamo preparando ora si chiama TurnOn, e servirà per accendere e spegnere il relay in modo da raggiungere la temperatura impostata. All’inizio della classe si definisce un **segnale** con valore booleano (**True** oppure **False**) chiamato **TempReached**. Potremo emettere questo segnale per far sapere al thread principale (quello dell’interfaccia grafica) che la temperatura fissata è stata raggiunta e il termostato ha fatto il suo lavoro. Nella funzione che costruisce il thread (cioè **__init__**), ci sono le istruzioni necessarie per accedere ai pin GPIO del relay e e al sensore. Il pin cui è collegato il relay viene memorizzato nella variabile **self.relayPin**, mentre il sensore sarà raggiungibile tramite l’oggetto **self.sensor**. La funzione prende in argomento **w**, un oggetto che rappresenta la finestra del programma (che passeremo al thread dell’interfaccia

grafica stessa). In questo modo le funzione del thread potranno accedere in tempo reale all'interfaccia e interagire.

La funzione **run** viene eseguita automaticamente quando avviamo il thread: potremmo inserire le varie operazioni al suo interno, ma per tenere il codice pulito ci limitiamo a usarla per chiamare a sua volta la funzione **reachTemp**. Sarà questa a svolgere le operazioni vere e proprie. Prima di vederla, definiamo un'altra funzione: **readTemp**. Questa funzione non fa altro che leggere la temperatura attuale, scriverla sul terminale per debug, e restituirla. Ci tornerà utile per leggere facilmente la temperatura e controllare se è stata raggiunta quella prefissata. Nella funzione **reachTemp** per prima cosa si dichiara di avere bisogno della variabile globale **toSleep**. Poi si inserisce un blocco try-except: in questo modo, se per qualche motivo l'attivazione del relay dovesse fallire, verrà lanciato il segnale **TempReached** con valore **False** e nell'interfaccia grafica potremo tenerne conto per avvisare l'utente che qualcosa è andato storto. Se invece va tutto bene, si inizia un ciclo while, che rimarrà attivo finché il pulsante presente nell'interfaccia grafica (che abbiamo chiamato SwitchOn) è premuto (cioè nello stato **checked**). Ciò significa che appena l'utente cliccherà sul pulsante per farlo passare allo stato "spento" (cioè "non checked") il ciclo si fermerà. Nel ciclo, si controlla se la temperatura attuale (ottenuta grazie alla funzione **readTemp**) sia inferiore alla temperatura impostata per il termostato. In caso positivo bisogna assicurarsi che il relay sia acceso, quindi il suo pin si imposta con valore **HIGH**. E poi si attende il tempo prefissato prima di fare un altro ciclo e quindi controllare di nuovo la temperatura. Se, invece, la temperatura attuale è maggiore di quella da raggiungere, vuol dire che possiamo spegnere il relay impostando il suo pin al valore **LOW**, ed emettere il segnale **TempReached** col valore **True**, visto che è andato tutto bene. Abbiamo quindi un ciclo che si ripete, per esempio, ogni 10 secondi finché viene

raggiunta la temperatura impostata, e a quel punto si interrompe.

Poi creiamo un'altra classe di tipo QThread, stavolta chiamata **ShowTemp**. In questa classe non facciamo altro che leggere la temperatura attuale, dal sensore, e inserirla nella **label** che abbiamo dedicato proprio a presentare questo valore all'utente. Avremmo potuto inserire questa funzione nello stesso QThread già creato per regolare la temperatura, visto che poi lo possiamo lanciare più volte e con argomenti diversi. Quindi avremmo potuto usare un QThread unico con le due funzioni da attivare separatamente. Tuttavia, quando si fanno operazioni diverse è una buona idea tenere il codice pulito e creare QThread separati. Esattamente come per il QThread precedente, la funzione di costruzione della classe (la solita **__init__**) richiede come argomento **w**, l'oggetto che rappresenta la finestra dell'interfaccia grafica. Viene anche creato l'oggetto **sensor**, per accedere al sensore di temperatura. Anche in questa classe inseriamo la funzione **readTemp**, uguale a quella del QThread precedente, e per semplificare stavolta scriviamo le istruzioni direttamente nella funzione **run**. Anche in questo caso si accede alla variabile globale **toSleep**. Il codice che svolge davvero le operazioni è un semplice ciclo infinito (**while True**) che imposta un nuovo valore come testo della label presente nell'interfaccia grafica (cioè **self.w**). L'etichetta in questione si chiama **tempAttuale**, e possiamo assegnarle un testo usando la funzione **setText**. Il testo viene creato prendendo la temperatura (ottenuta dalla funzione **readTemp** e traducendo il numero in una stringa di testo) e aggiungendo il simbolo dei gradi Celsius. Poi, si aspetta il tempo preventivato prima di procedere a ripetere il ciclo.

Il risultato che otterremo, con l'interfaccia grafica interattiva

La classe **MainWidow**, di tipo QMainWindow, conterrà il codice necessario a far apparire e funzionare la nostra interfaccia grafica.

La prima cosa da fare, nella funzione che costruisce la classe (la solita **__init__**) è caricare, in lettura (**QFile.ReadOnly**) il file che contiene l'interfaccia grafica stessa. Il file in questione si trova nella stessa cartella dello script attuale, e si chiama **termostato.ui**, quindi possiamo scoprire il suo percorso completo estraendo dal nome dello script (**sys.argv[0]**) la cartella in cui si trova (con la funzione **os.path.dirname**) e risalire al percorso assoluto con la funzione **os.path.abspath**. L'interfaccia grafica viene interpretata con la libreria **QUiLoader**, e memorizzata nell'oggetto **self.w**. Da questo momento sarà quindi possibile accedere ai vari componenti dell'interfaccia grafica usando questo oggetto. Affinché l'interfaccia grafica venga utilizzata per la finestra che stiamo costruendo, bisogna impostare l'oggetto **self.w** come **CentralWidget**. Possiamo impostare un titolo per la finestra con la funzione **self.setWindowTitle**, tipica di ogni QMainWindow. Ora possiamo cominciare a rendere interattiva l'interfaccia grafica: per farlo dobbiamo collegare i segnali degli oggetti dell'interfaccia alle funzioni che si occuperanno di gestirli. Per esempio, dobbiamo collegare il segnale **clicked** del pulsante **SwitchOn** a una funzione che chiameremo **self.StopThis**. Lo facciamo usando la funzione **connect** del segnale di questo

pulsante. La scrittura è molto semplice: si tratta semplicemente di un sistema a scatole cinesi. Per esempio, anche quando colleghiamo il movimento della rotella (la QDial per impostare la temperatura) alla funzione **setTempImp** non facciamo altro che prendere l'oggetto che rappresenta l'interfaccia grafica, cioè **self.w**, e puntare sulla QDial al suo interno, che avevamo chiamato **tempImpostata**. All'interno di questo oggetto, andiamo a recuperare il segnale **valueChanged**, che viene emesso dalla QDial stessa nel momento in cui l'utente modifica il suo valore spostando la rotella, e per questo segnale chiamiamo la funzione **connect**. Alla funzione bisogna soltanto assegnare il nome (completo di **self.**, non dimentichiamo che è un membro della classe Python che stiamo scrivendo) della funzione che dovrà essere chiamata. Va indicato solo il nome, senza le parentesi, quindi si scrive **self.setTempImp** e non **self.setTempImp()**.

Sempre nella funzione **__init__** si provvede a dare i comandi necessari per attivare i moduli di sistema che forniscono il controllo del sensore e dei pin GPIO. Poi impostiamo la variabile **self.alreadyOn** a False: si tratta di un semplice flag che useremo per capire se il relay sia già stato attivato, o se sia necessario attivarlo, quindi ovviamente all'avvio del programma è False perché il relay è ancora spento. Ora si può accedere alla QDial e impostare manualmente il suo valore iniziale, con la funzione **setValue**, per esempio a 25 gradi. Poi va chiamata manualmente la funzione **setTempImp**, con lo stesso valore in argomento, per essere sicuri che il programma controlli se sia necessario accendere o spegnere il relay per raggiungere la temperatura in questione. La variabile **self.stoponreached** verrà utilizzata soltanto per decidere se disattivare il termostato una volta raggiunta la temperatura: di norma non è necessario, anzi, si preferisce che il termostato rimanga vigile per riaccendere il relay qualora la temperatura dovesse scendere nuovamente. Ma per funzioni di test o casi di abitazioni con un isolamento

davvero buono può avere senso impostare questa variabile a True. L'ultima cosa da fare prima di concludere la funzione di costruzione dell'interfaccia grafica è creare il thread che si occupa di leggere la temepratura attuale dal sensore e scriverla nell'apposita label. Basta creare un oggetto di tipo **ShowTemp**, perché questo è il nome che abbiamo scelto per la classe di questo QThread, indicando l'oggetto **self.w** come argomento, così le funzioni del thread potranno accedere all'interfaccia grafica di questa finestra. Il thread viene avviato usando la funzione **start**. È importante non confondersi: quando si scrive la classe del QThread il codice va messo nella funzione **run**, ma quando lo si avvia si chiama la funzione **start**, perché così vengono eseguiti una serie di controlli prima dell'effettivo inizio delle operazioni.

Definiamo due funzioni: una è **reached**, e l'altra **itIsOff**. La funzione **reached** verrà chiamata automaticamente quando la temperatura impostata per il termostato viene raggiunta. A questo punto possiamo decidere cosa fare: se la variabile **stoponreached** è impostata a True, chiameremo la funzione **itIsOff**, così da disattivare il termostato. In caso contrario, non è necessario fare nulla, ma volendo si potrebbe modificare l'aspetto dell'interfaccia grafica (per esempio colorando di verde l'etichetta con la temperatura) per segnalare che la temperatura è stata raggiunta. La funzione **itIsOff**, come abbiamo già suggerito, si occupa di disattivare il termostato. Per farlo, imposta come False il pulsante presente nell'interfaccia grafica: siccome si comporta come un interruttore, se il suo stato **checked** è falso il pulsante è disattivato. E, come avevamo visto nella classe del thread TurnOn, il ciclo che si occupa di controllare se sia necessario tenere il relay rimane attivo solo se il pulsante è **checked**. Poi viene chiamata la funzione StopThis, che si occupa di modificare il pulsante (che da "Spegni" deve diventare "Accendi").

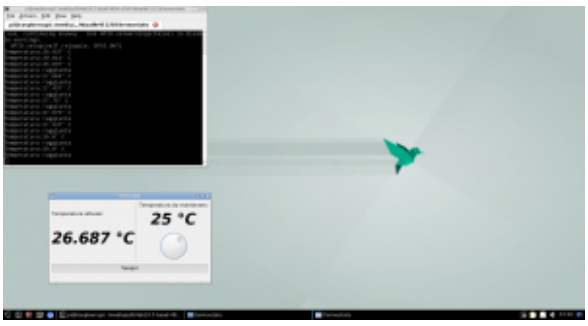
La funzione **dostuff** è quella che si occupa effettivamente di creare il thread dedicato al controllo del relay. Prima di tutto, si controlla che il thread non sia già stato avviato, usando il flag **alreadyOn** che avevamo creato all'inizio del codice di questa classe. Se il thread non è già attivo, lo si crea passandogli l'oggetto **self.w** così da permettere al thread l'accesso all'interfaccia grafica. Poi colleghiamo il segnale **TempReached**, che avevamo creato per il thread **TurnOn**, alla funzione **self.reached**. Si collega anche il segnale **finished**, dello stesso thread, alla funzione **itItOff**, così se per un motivo o l'altro il thread dovesse terminare la funzione adeguerebbe lo stato del pulsante presente nell'interfaccia grafica. Alla fine, si avvia il thread e si imposta il flag **alreadyOn** come True.

La funzione **StopThis** controlla lo stato attuale del pulsante: se è attivato, il suo testo deve essere impostato a "Spegni", e bisogna ovviamente chiamare la funzione **dostuff** in modo che venga lanciato il thread, se necessario. Viceversa, se il pulsante è disattivato, il suo testo deve essere "Accendi", così l'utente capirà che premendo il pulsante può attivare il sistema, e il flag **alreadyOn** va impostato a False, per segnalare che al momento il thread è disattivato (ricordiamo che se il pulsante è disattivato, anche il thread **TurnOn**, inserito in questa finestra col nome **self.myThread**, si disattiva automaticamente).

L'ultima funzione che inseriamo nella classe della finestra è **setTempImp**, ed è la funzione che abbiamo collegato al segnale **valueChanged** della QDial. Quando l'utente sposta la rotella, verrà chiamata questa funzione. Si può notare che questa funzione è leggermente diversa dalle altre che abbiamo scritto finora per reagire ai segnali dell'interfaccia grafica: ha un argomento, chiamato **arg1**. Questo perché il segnale **valueChanged** offre alla funzione chiamata il valore attuale della QDial. Nel nostro caso, quindi, **arg1** contiene la

temperatura che l’utente vuole impostare, quindi possiamo direttamente inserirla nell’etichetta **tempImpostataLabel**, che abbiamo creato nell’interfaccia grafica proprio per presentare la temperatura selezionata. Poi, per sicurezza, chiamiamo la funzione **dostuff** in modo da attivare il thread che fa funzionare il relay se è necessario.

Terminate le classi, possiamo scrivere il codice principale del programma. Per prima cosa, il programma crea una **QApplication**, necessaria per le librerie Qt. Poi possiamo creare una istanza della finestra principale, memorizzandola nell’oggetto **w**. Ora possiamo impostare alcune caratteristiche della finestra. Per esempio, la dimensione: se stiamo usando lo schermo PiTFT3.5, la risoluzione ideale è 640×480: naturalmente, si possono modificare i parametri sulla base delle proprie esigenze. Infine, si fa apparire la finestra con la funzione **show**. E quando questa verrà chiusa (cosa che nel nostro caso non dovrebbe succedere, ma è un buona idea tenere in considerazione l’ipotesi), si chiude normalmente il programma, fornendo alla riga di comando il risultato dell’esecuzione dell’applicazione (quindi eventuali codici d’errore se qualcosa dovesse non funzionare correttamente).



L’applicazione eseguita dal desktop del Raspberry, per provare il suo funzionamento

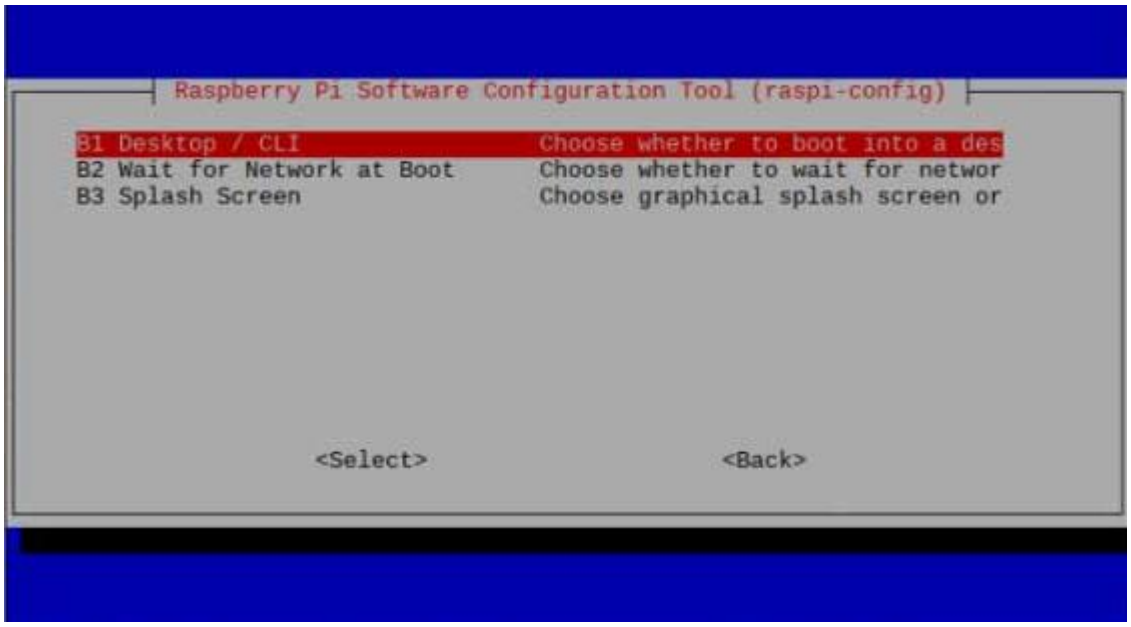
Avvio automatico

Siccome la nostra applicazione è pensata per apparire sullo schermo all'avvio del sistema, dobbiamo preparare un piccolo script per automatizzare la sua installazione:

Prenderemo come riferimento l'utente pi, che è sempre disponibile su Raspbian. Con queste prime righe di codice creiamo un servizio di sistema che esegue il login automatico per l'utente pi sul terminale tty1. Così, all'avvio del sistema non sarà necessario digitare la password, si avrà subito un terminale funzionante.

Si modificano due file di configurazione: con la modifica a xinit aggiungiamo il comando cat all'avvio del server grafico: questo permette di tenerlo in stallo e evitare eventuali procedure automatiche. La modifica a bashrc ci permette di fare un controllo appena viene aperto un terminale per l'utente **pi**. Se il nome del terminale è tty1 (su GNU/Linux ci sono diversi terminali disponibili) lanciamo sia lo script di avvio del nostro programma, sia il server grafico Xorg. Aver fatto questo controllo è importante, perché così il programma termostato verrà avviato automaticamente solo sul terminale tty1, quello che ha l'autologin, e non su tutti gli altri.

Infine, si scrive lo script di avvio del programma termostato: inizialmente, lo script attende un secondo, in modo da essere sicuro che il server grafico sia pronto a funzionare. Poi, lancia Python3 con il nostro programma.



C’è un dettaglio: con l’immagine di Raspbian Buster fornita all’inizio dell’articolo l’autologin potrebbe non funzionare correttamente, e questo perché Raspbian usa carica una immagine di splash per il boot che impedisce il corretto avvio del server grafico per come lo abbiamo configurato. La soluzione è disabilitare il boot con splash grafica, visto che comunque non ci serve, usando il comando **sudo raspi-config** nella sezione **Boot**, selezionando l’opzione **Splash Screen** e impostandola come disabilitata.

Il codice completo
Potete trovare il codice completo nel repository git https://codice-sorgente.it/cgit/termostato-raspberry.git/tree/
Per scaricarlo potete dare il comando

da un terminale GNU/Linux come quello del Raspberry, oppure usare le varie interfacce grafiche di Git disponibili. O, anche, scaricare i file singolarmente dalla pagina https://codice-sorgente.it/cgit/termostato-raspberry.git/plain/. Nel repository è presente un README con i comandi da eseguire per installare correttamente il programma sulla

versione di **Raspbian Buster** che proponiamo. C’è da dire che il codice che abbiamo presentato è pensato per un uso accademico, non professionale: una applicazione per un termostato può essere più semplice, il codice è prolisso in diversi punti per facilitare la comprensione a chi non sia pratico delle librerie Qt.