Con un soffio (Keyboard Emulator)

Un progetto OpenSource di Pierpaolo Martinello

Il progetto prevede di emulare una tastiera in modo “particolare”.

Si tratta di eseguire delle azioni tipiche di una tastiera sia meccanicamente, con normali pulsanti pulsanti / pedaliera, sia con azione pneumatica.

In pratica agisce simulando una tastiera USB in modalità di comando Su o Giù, selezionabile tra i tasti freccia e quelli di pagina usando una selezione con pulsante normalmente aperto (default come Pagina).

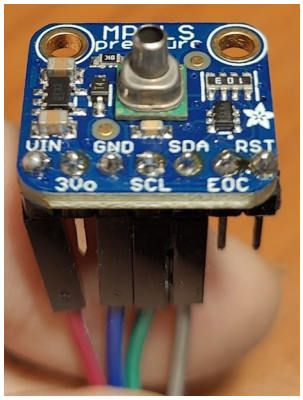
Ho implementato un Led RGB come monitor di stato tra le due modalità e azione eseguita: Se viene azionato il pulsante il colore del led lampeggia tra Rosso (indicante la modalità pagina) e Verde che invece coinvolge i tasti freccia.

Materiale occorrente

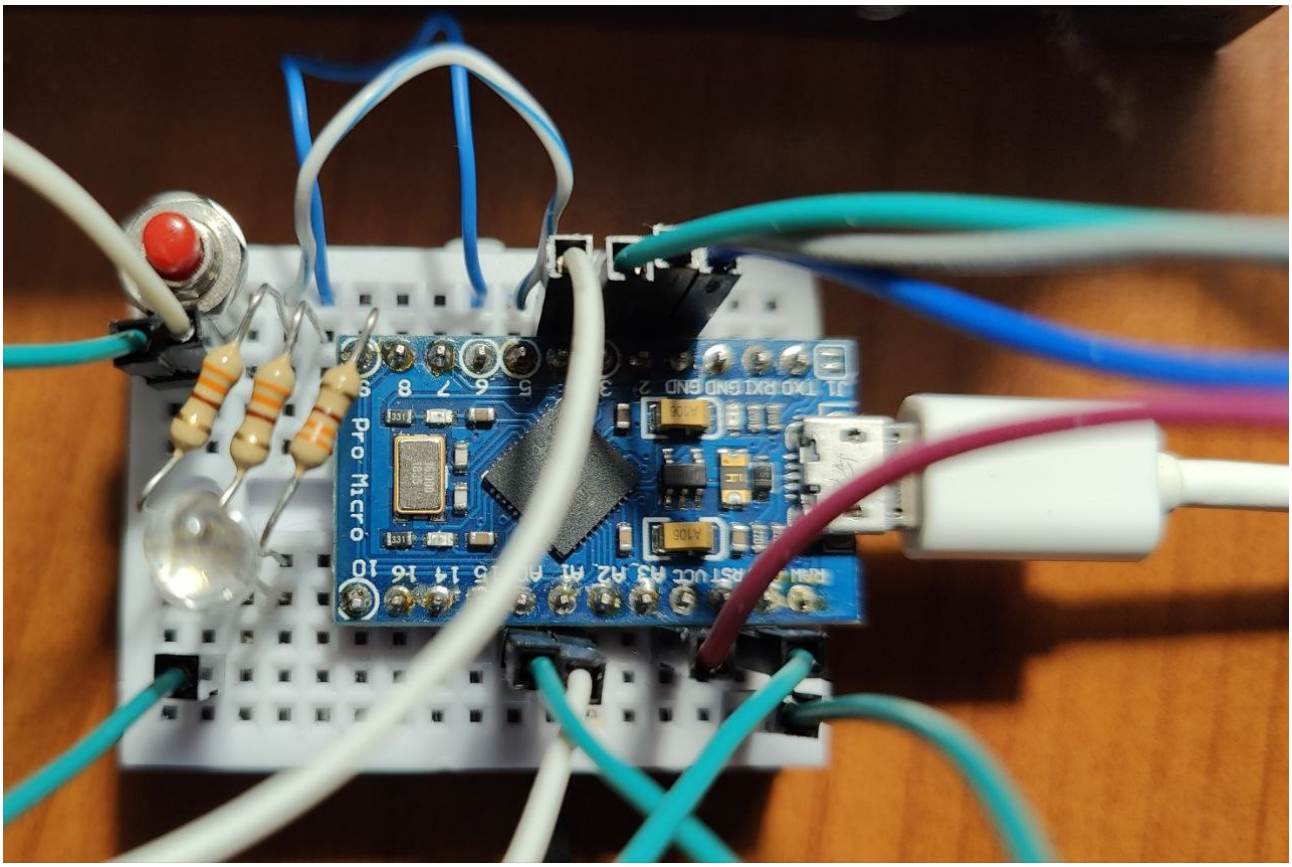
1 Arduino Leonardo (io ho usato la versione Pro Micro).
1 Sensore di pressione MPRLS Adafruit.
1 Led Rgb 6,3 mm.
3 Led Infrarosso.
3 resistenze da 330 Ohm.
1 resistenza da 47000 Ohm.
1 Transistor BC 377
1 Presa jack 3,5 stereo.
2 Porta Led in plastica.
1 Porta Led in metallo diametro a scelta.
1 Pulsante normalmente aperto.
1 Basetta sperimentale mille fori di misura 620 x 620 mm.
1 Tubicino in gomma.
1 Bocchino per Fumatori diametro 6 mm.
2 Connettori a giunzione maschio e femmina.
Guaina termo restringente.
Tubo plastica diametro 20 mm.
Foglio di plastica 100x100x2 mm.
1 Vite grande.
2 viti mignon.
4 supporti in gomma adesivi.
1 Scatola plastica Larga 104 mm, profonda 68 mm ed alta 22 mm.

L’alimentazione a 5V è prelevata dalla presa Usb del pc o da un alimentatore usb dedicato.

Il Sensore Adafruit



Il Prototipo su Breadboard



Per il codice qui sotto riportato i puristi non mi vogliano linciare: ho aggiunto parecchi spazi a vantaggio di una maggiore leggibilità.

```
/*!
 * @file Legge_UpDown.ino
 *
 * A basic test of the sensor with default settings
 *
 * Designed specifically to work with the MPRLS sensor from Adafruit
 * ----> https://www.adafruit.com/products/3965
 *
 * These sensors use I2C to communicate, 2 pins (SCL+SDA) are required
 * to interface with the breakout.
 *
 * Adafruit invests time and resources providing this open source code,
 * please support Adafruit and open-source hardware by purchasing
 * products from Adafruit!
 *
 * Written by Limor Fried/Ladyada for Adafruit Industries.
```

```
 *
 * MIT license, all text here must be included in any redistribution.
 *
 */
 /*
 * SimpleSender.cpp
 *
 *  Demonstrates sending IR codes in standard format with address and command
 *  An extended example for sending can be found as SendDemo.
 *
 *  Copyright (C) 2020-2022  Armin Joachimsmeyer
 *  armin.joachimsmeyer@gmail.com
 *
 *  This file is part of Arduino-IRremote https://github.com/Arduino-IRremote/Arduino-IRremote.

************************************************************************************
 *  MIT License
 *
 * Copyright (c) 2020-2023 Armin Joachimsmeyer
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is furnished
 * to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 ************************************************************************************
 */
 /*
 @ Copyright Pierpaolo Martinello 16/08/2023
 */

#include <Wire.h>                       // Include la libreria di comunicazione seriale
#include "Adafruit_MPRLS.h"             // Include la libreria del sensore
#include <Keyboard.h>                   // Include la libreria della tastiera
#include <Arduino.h>

#define DISABLE_CODE_FOR_RECEIVER // Disables restarting receiver after each send. Saves 450 bytes program memory and 269 bytes RAM if receiving functions are not used.
//#define SEND_PWM_BY_TIMER         // Disable carrier PWM generation in software and use (restricted) hardware PWM.
//#define USE_NO_SEND_PWM           // Use no carrier PWM, just simulate an active low receiver signal. Overrides SEND_PWM_BY_TIMER definition

// #define _DBG_          // Se abilitato mostra sulla seriale tutti i messaggi dei codici trasmessi
```

```
/*
 * This include defines the actual pin number for pins like IR_RECEIVE_PIN, IR_SEND_PIN for many
different boards and architectures
 */
#include "PinDefinitionsAndMore.h"
#include <IRremote.hpp> // include the library

// You dont *need* a reset and EOC pin for most uses, so we set to -1 and don't connect
#define RESET_PIN -1  // set to any GPIO pin # to hard-reset on begin()
#define EOC_PIN   -1  // set to any GPIO pin to read end-of-conversion by pin

int c_press;      // Dichiarazione delle variabili per la lettura della pressione
int RXLED  = 17;  // Il LED RX ha un pin Arduino interno definito
int TXLED  = 30;  // Il LED RX ha un pin Arduino interno definito

int Sw_A6  =  4;  // Dichiarazione delle variabili per i pin
int Sw_Up  = 18;  // Dichiarazione delle variabili per i pin
int Sw_Dn  = 19;  // Dichiarazione delle variabili per i pin
int Pagina =  1;  // Parte in modalità Pagina

/******************** definizione delle costanti per i led *****************/
int Sw_A9  =  9;  // porta  9 da collegare all'anodo "rosso" del modulo RBG
int PVerde =  5;  // porta  5 da collegare all'anodo "verde" del modulo RGB
int Sw_A10 = 10;  // porta 10 da collegare all'anodo "blu"   del modulo RGB

/* Inizializza il sensore di pressione*/
Adafruit_MPRLS mpr = Adafruit_MPRLS(RESET_PIN, EOC_PIN) ;

/* Funzione per gestire il colore sul led RGB */
void Colore( int Rosso, int Verde, int Blu ) {
     analogWrite( Sw_A9,  Rosso ) ;
     analogWrite( PVerde, Verde ) ;
     analogWrite( Sw_A10, Blu ) ;
}
//*****************************************************
void setup() {
  pinMode( Sw_A6, INPUT_PULLUP ) ; // Abilita la resistenza interna di PULLUP
  pinMode( Sw_Up, INPUT_PULLUP ) ; // Abilita la resistenza interna di PULLUP
  pinMode( Sw_Dn, INPUT_PULLUP ) ; // Abilita la resistenza interna di PULLUP

  // inizializza i pinn digitale LED_BUILTIN come output.
  pinMode( LED_BUILTIN, OUTPUT ) ;
  pinMode( RXLED, OUTPUT ) ;   // Set RX LED as an output
  pinMode( TXLED, OUTPUT ) ;   // Set TX LED as an output

  pinMode( Sw_A9,  OUTPUT ) ;  // dichiara la porta  9 come porta di output
  pinMode( PVerde, OUTPUT ) ;  // dichiara la porta  5 come porta di output
  pinMode( Sw_A10, OUTPUT ) ;  // dichiara la porta 10 come porta di output

  /* Attivo il monitor seriale */
  Serial.begin( 115200 ) ;

  pinMode(LED_BUILTIN, OUTPUT) ;
  // Just to know which program is running on my Arduino
  //Serial.println(F("START " __FILE__ " from " __DATE__ "\r\nUsing library version "
VERSION_IRREMOTE)) ;
```

```
  //Serial.print(F("Send IR signals at pin ")) ;
  Serial.println(IR_SEND_PIN) ;
  /*
   * The IR library setup. That's all!
  */
  //  IrSender.begin() ; // Start with IR_SEND_PIN as send pin and if NO_LED_FEEDBACK_CODE is NOT
defined, enable feedback LED at default feedback LED pin
  IrSender.begin(DISABLE_LED_FEEDBACK) ; // Start with IR_SEND_PIN as send pin and disable
feedback LED at default feedback LED pin

  Serial.flush() ;
  Serial.println( "Abilitazione Sensore MPRLS" ) ;

  /* Apro la comunicazione con il sensore */
  if ( ! mpr.begin() ) {
    Serial.println( "Impossibile comunicare con il sensore MPRLS, controllare il cablaggio!?" ) ;
    while ( 1 ) {
      delay( 10 ) ;
    }
  }
  Serial.println("Rilevato sensore MPRLS") ;

  /* Leggo la pressione */
  c_press = mpr.readPressure() ;

  if ( digitalRead(Sw_A6) == LOW)   //Checking if the first switch has been pressed
  {
    Serial.print( "Funzione PageUP/DOWN := " ) ;Serial.println( Pagina ) ;
    Pagina = 0 ;
  }
  Rxlamp(1 ,200, -1 ) ;
}
uint8_t sCommand = 0x20 ;
uint8_t sRepeats = 2 ;

void loop() {
     digitalWrite( RXLED, HIGH ) ;    // set the LED off
     digitalWrite( TXLED, HIGH ) ;    // set the LED off
     int pressure_hPa = mpr.readPressure() ;

     if ( digitalRead( Sw_A6 ) == LOW ) {  // controllo il pulsante di commutazione
        switch (Pagina)
               {
               case 0 :                // Lo devo mandare in modo Pagina
                    Pagina ++ ;
                    Serial.print( "Modo Pagina con pressione pari a
" ) ;Serial.println( pressure_hPa ) ;
                    Rxlamp( 0 , 200, -1 ) ;
                    break;

               case 1 :                // Lo devo mandare in modo Freccia
                    Pagina -- ;
                    Serial.print( "Modo Freccia con pressione pari a
" ) ;Serial.println( pressure_hPa ) ;
                    Rxlamp( 1, 200, -1 ) ;
                    break;
               }
```

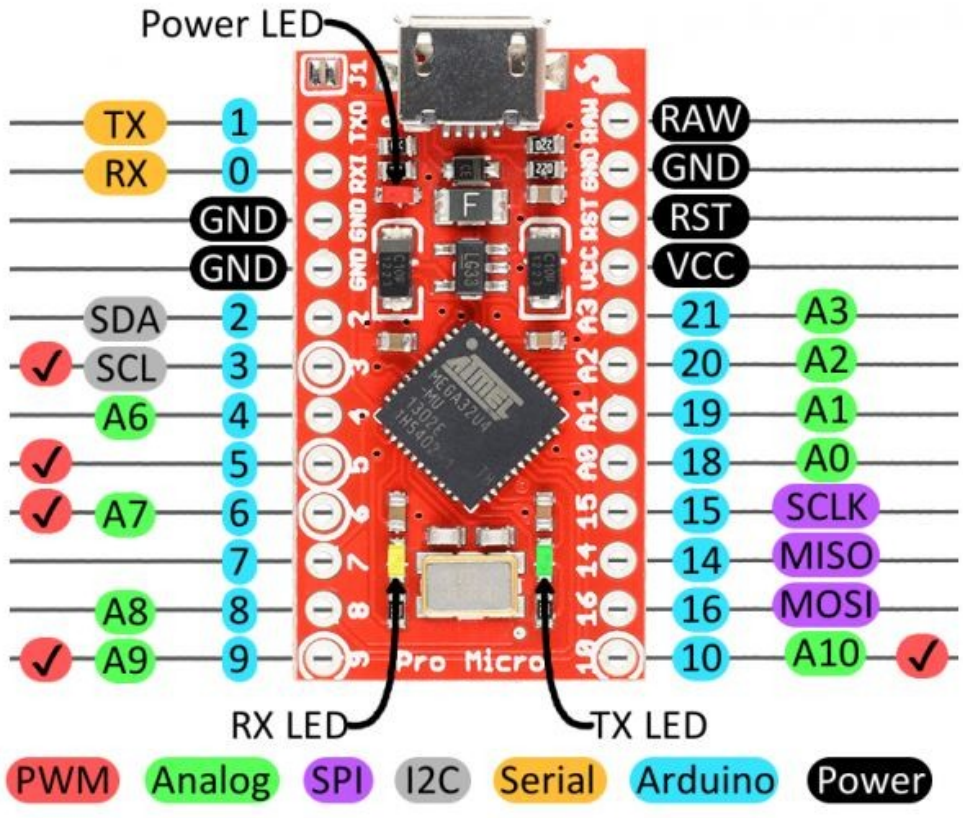
```
     }
     if ( pressure_hPa > c_press+5 || digitalRead( Sw_Dn ) == LOW )
     // Pressione in hPa => Pressure_hPa / 68.947572932) ;
     {
        Serial.print( "SOFFIO : + " ) ;Serial.print(Pagina) ;Serial.print( " ->
" ) ;Serial.println( pressure_hPa ) ;
        Rxlamp( 2, 100, 1 ) ;   //  Accendo il led ed eseguo il comando tastiera
     } else if ( pressure_hPa < c_press-5  || digitalRead(Sw_Up) == LOW )
     {
       Serial.print( "ASPIRO : - " ) ;Serial.print( Pagina ) ;Serial.print( " <-
" ) ;Serial.println( pressure_hPa ) ;
       Rxlamp( 2, 100, 0 ) ;
     }
     delay( 200 ) ;
}

/*  Accendo il led ed eseguo il comando tastiera */
void Rxlamp( int c , int t, int act ) { //  Parametri: ( Colore, Tempo di accensione, Azione [0
Indietro, 1 Avanti] )
     switch ( c ) {                     // Scelgo il colore di lampeggio
            case 0 :     // Luce rossa
                 Colore( 10, 0, 0 ) ;
                 break;

            case 1 :     // Luce Verde
                 Colore( 0, 10, 0 ) ;
                 break;

            case 2 :    // Luce Blu
                 Colore( 0, 0, 10 ) ;
                 break;
     }

     delay( t*2 ) ;
     Colore( 0, 0, 0 ) ;                // Spengo i led

     switch ( Pagina ) {
            case 0 :
                 Serial.println( "Modalità Freccia" ) ;
                 Colore( 0, 10, 0 ) ;  //Verde
              break;

            case 1 :
                 Serial.println( "Modalità Pagina" ) ;
              Colore( 10, 0, 0 ) ;  // Rosso
                 break;
     }

   GoTasto( act ) ;
   //delay( t*1.5 ) ;
   Colore( 0, 0, 0 ) ;

// FINE PROGRAMMA
}
// Funzioni Tastiera
void GoTasto ( int act ) {
     switch ( act ) {
```

```
            case 0 :     // Indietro
                 switch ( Pagina ) {
                        case 0 :                // Opero come freccia indietro
                             SendIR( 0x50 ) ;
                             Keyboard.press( KEY_UP_ARROW ) ;
                            break ;

                        case 1 :                // Opero come Pagina Su
                             SendIR( 0x20 ) ;
                             Keyboard.press( KEY_PAGE_UP ) ;
                             break ;
                 }
                 break;

            case 1 :     // Avanti
                 switch ( Pagina ) {
                        case 0 :               // Opero come freccia avanti
                             SendIR( 0x51 ) ;
                             Keyboard.press( KEY_DOWN_ARROW ) ;
                             break ;

                        case 1:                // Opero come Pagina Avanti
                             SendIR( 0x21 ) ;
                             Keyboard.press(KEY_PAGE_DOWN) ;
                             break ;
                 }
                 break ;
     }

     delay( 100 ) ;
     // Sarebbe più indicato resettare il singolo comando dato ma così il codice è più compatto
     Keyboard.releaseAll() ;   // Resetta tutti i comandi tastiera
     delay( 120 ) ;  // Attendi prima di inviare un altro comando
     }

void SendIR( uint8_t sCommand ) {
     /*
     * Print current send values
     */
     #if defined( _DBG_ )
         Serial.println() ;
         Serial.print(F("Send now: address=0x20, command=0x")) ;
         Serial.print(sCommand, HEX) ;
         Serial.print(F(", repeats=")) ;
         Serial.print(sRepeats) ;  Serial.print(F(" Pin Out=")) ;Serial.println(IR_SEND_PIN) ;
         Serial.println() ;

         //Serial.println(F("Send standard Rc5 with 8 bit address")) ;
         Serial.flush() ;
     #endif
     // Transmitter output
     IrSender.sendRC5(0x00, sCommand, sRepeats) ;

     delay(100) ;  //delay must be greater than 5 ms (RECORD_GAP_MICROS), otherwise the receiver
sees it as one long signal
}
// EOF
```

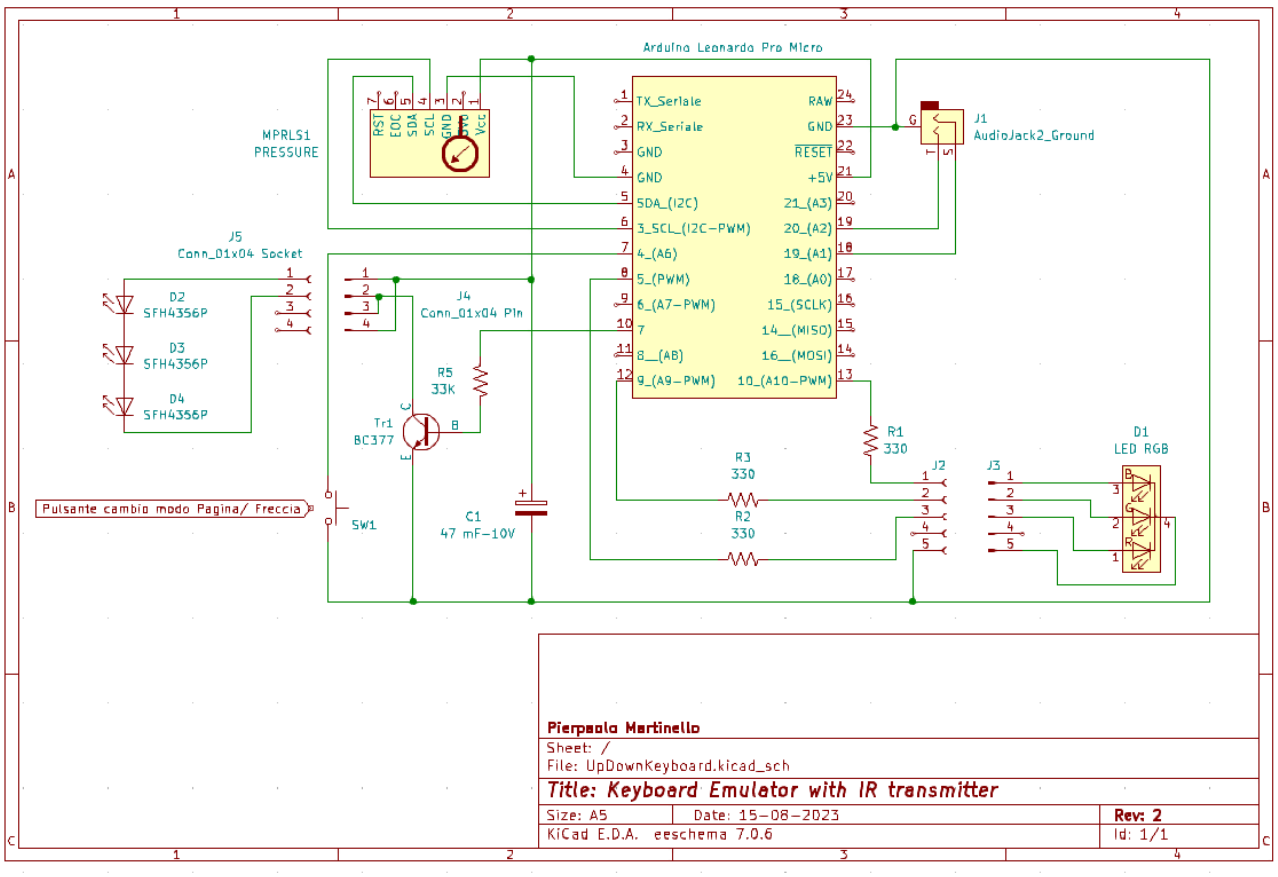
Nota:

Questo sorgente necessita di un file aggiuntivo denominato "PinDefinitionsAndMore.h" ma un solo piccolo particolare lo differenzia nei dall'originale della libreria IRemote, quello da me modificato è contenuto nella cartella Legge_UpDown.

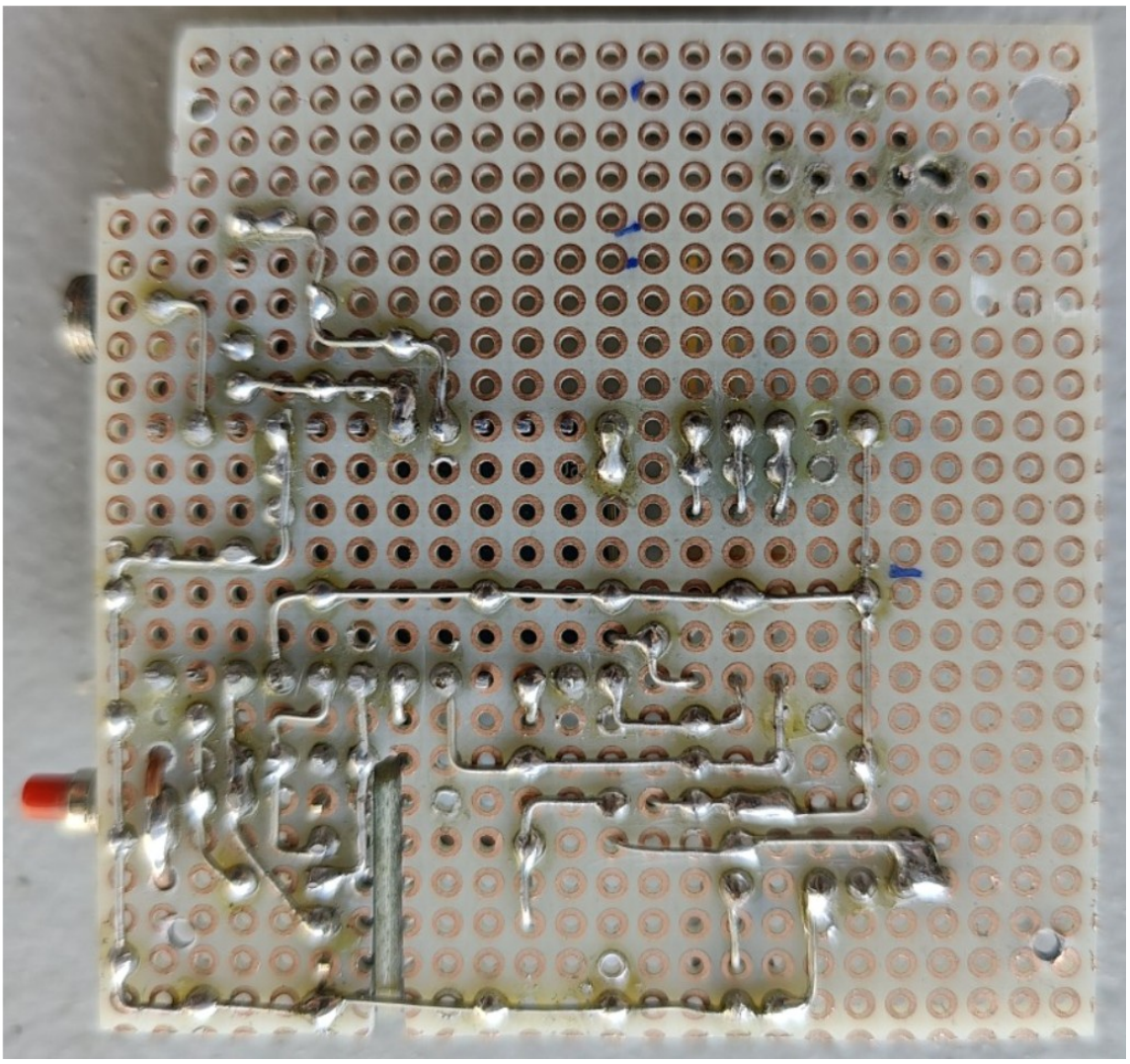
Questa la piedinatura dell'Arduino Leonardo Pro Micro



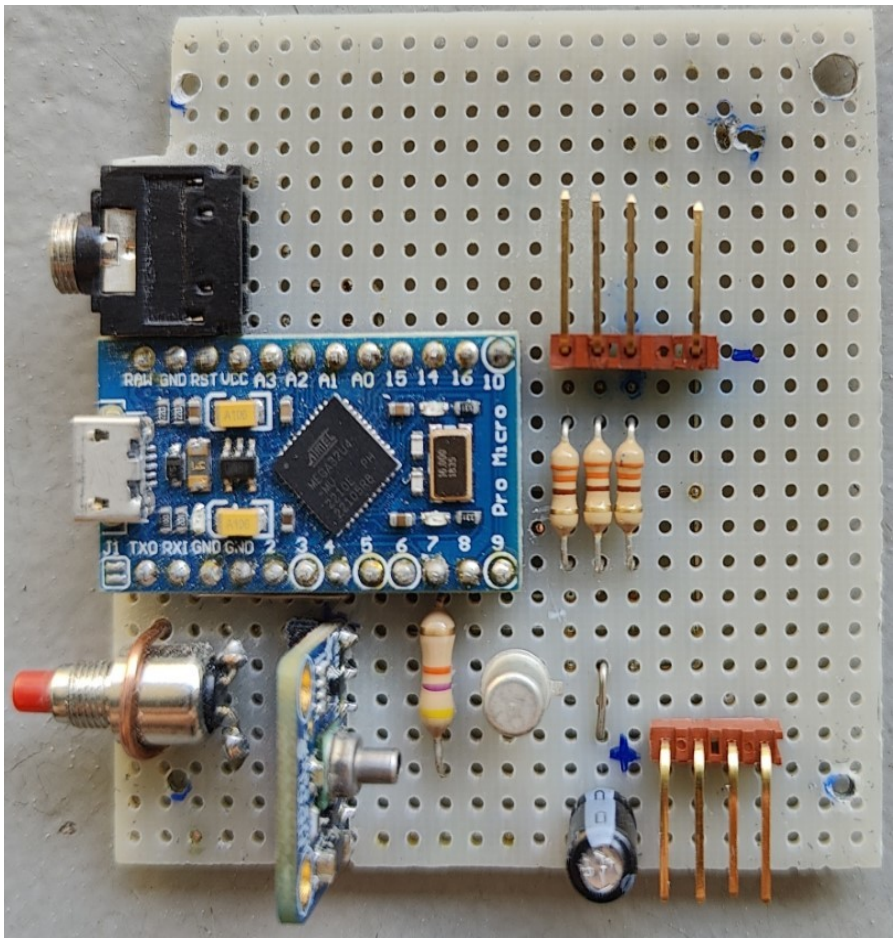
Lo schema è dunque dipendente dal codice, ho cercato una soluzione semplice e ordinata.



Questa è la disposizione delle piste sulla basetta millefori:



E questo il lato frontale



Progetto ultimato ed inscatolato.

Ora se soffiamo otteniamo un Pagina Giù, viceversa se aspiriamo otteniamo un Pagina Su.

Il led col colore **blu** ci avviserà che sta eseguendo un azione, ma in seguito si colorerà di **rosso** in modalità pagina, **verde** in modalità freccia per poi spegnersi.

Premendo il pulsante potremo variarne la modalità operativa ed avremo un lampeggio indicante la modalità d'uso impostata (**rosso** per la modalità pagina, **verde** per la modalità freccia).

L'eventuale pedaliera si può collegare alla presa jack usando il polo esterno come comune ai due contatti.

La testata rotante superiore alloggia i 3 led disposti in maniera tale da fornire un ampio angolo di trasmissione anche da piccole distanze.

Ora occupiamoci della parte remota:

Occorrente:

1 Arduino Leonardo (io ho usato la versione Pro Micro).

1 Rilevatore infarossi dedicato (Es. TSOP1838)

1 Led verde 3,5 mm.

3 Led Infrarosso.

1 resistenze da 330 Ohm.

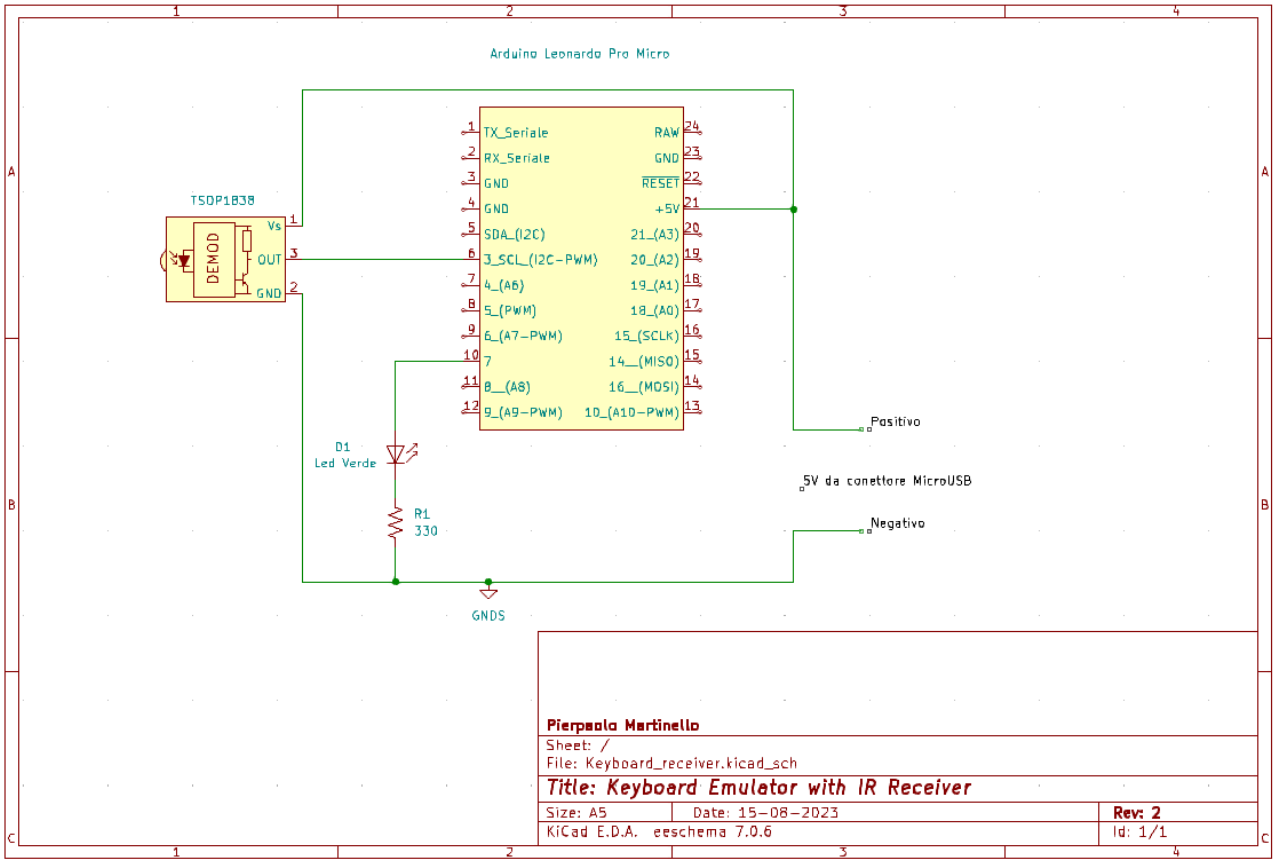
1 Basetta sperimentale mille fori di misura 54 x 30 mm.

1 Contenitore plastico.

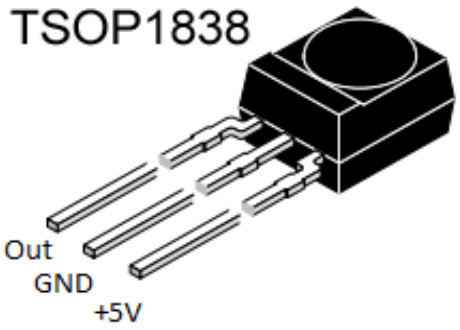
4 supporti in gomma adesivi.

1 Scatola plastica Larga 60 mm, profonda 35 mm ed alta 18 mm.

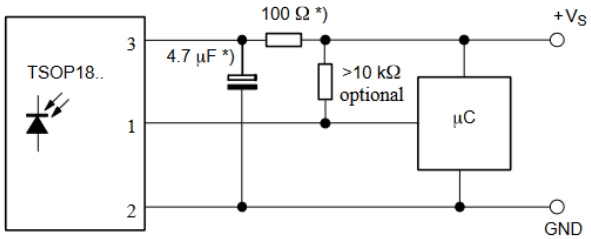
L'alimentazione 5V come al solito è prelevata dalla presa Usb.

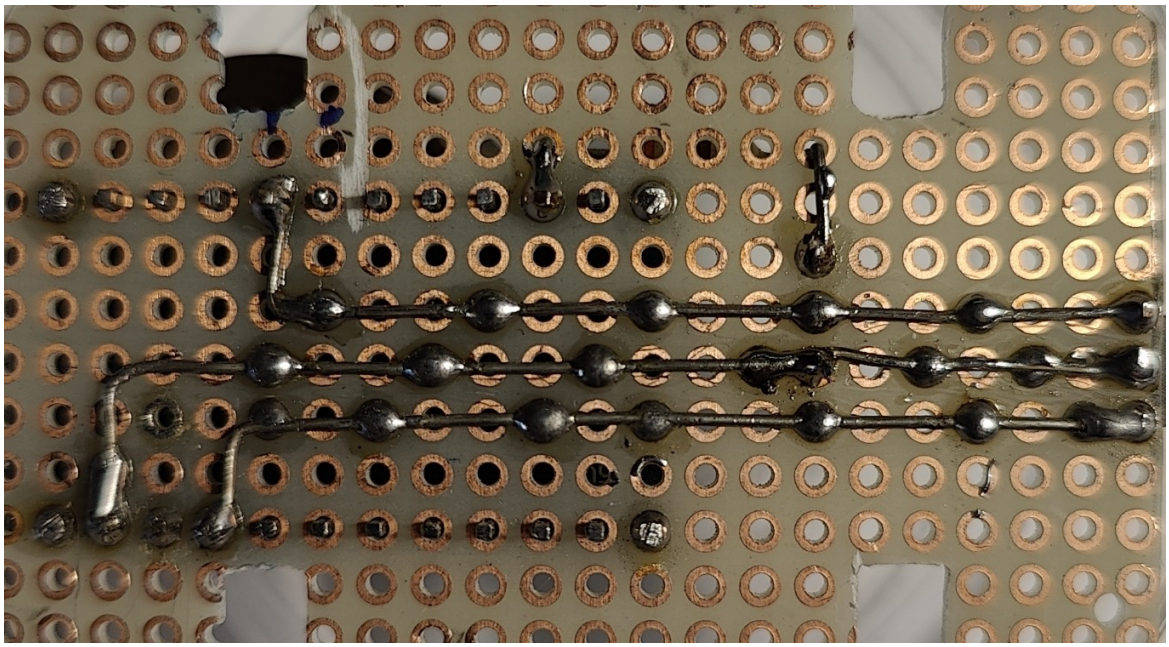


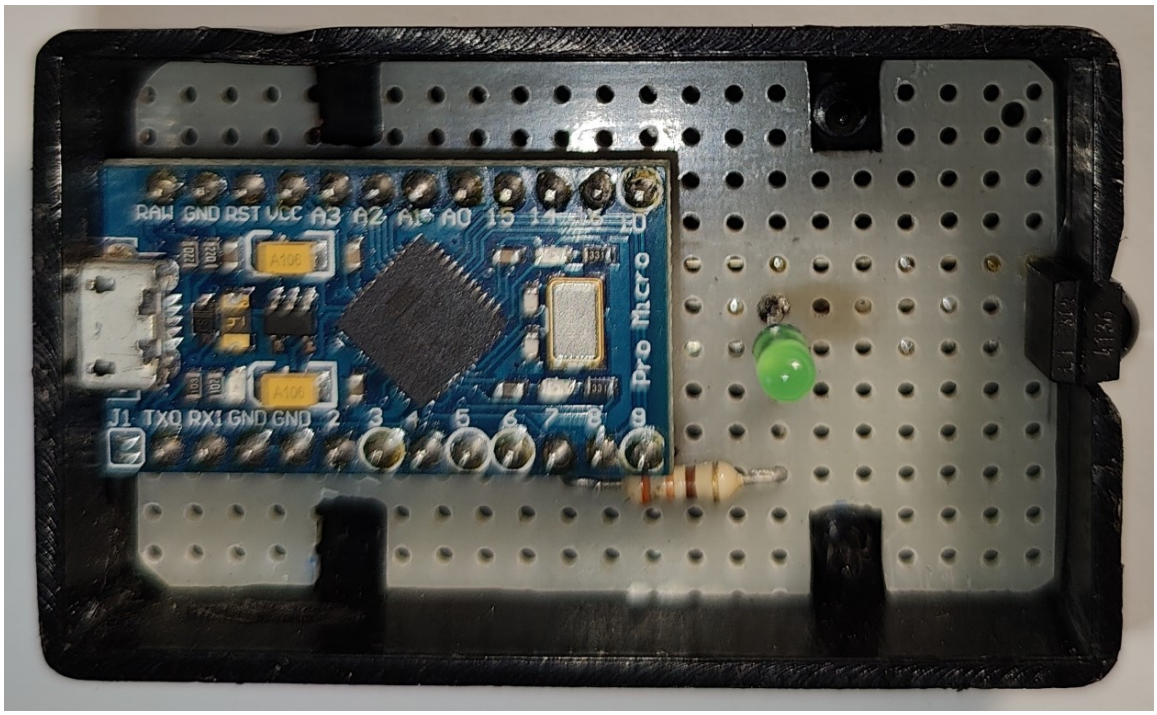
Le foto



In caso di disturbi si potrebbe inserire un filtro Rc come consigliato sul datasheet. Occorre precisare che nella prova pratica non ne ho riscontrato la necessità.







Il codice

```
/*
 * @File IrRx.ino
 *
 * Demonstrates receiving IR codes with IRremote
 *
 *  This file is part of Arduino-IRremote https://github.com/Arduino-IRremote/Arduino-IRremote.
 *
 ************************************************************************************
 * MIT License
 *
 * Copyright (c) 2020-2023 Armin Joachimsmeyer
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is furnished
 * to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 ************************************************************************************
 */

/*
 @ Copyright Pierpaolo Martinello 16/08/2023
 */
```

```
/*
 * Specify which protocol(s) should be used for decoding.
 * If no protocol is defined, all protocols (except Bang&Olufsen) are active.
 * This must be done before the #include <IRremote.hpp>
 */
//#define DECODE_DENON        // Includes Sharp
//#define DECODE_JVC
//#define DECODE_KASEIKYO
//#define DECODE_PANASONIC    // alias for DECODE_KASEIKYO
//#define DECODE_LG
//#define DECODE_NEC          // Includes Apple and Onkyo
//#define DECODE_SAMSUNG
//#define DECODE_SONY
#define DECODE_RC5
//#define DECODE_RC6

//#define DECODE_BOSEWAVE
//#define DECODE_LEGO_PF
//#define DECODE_MAGIQUEST
//#define DECODE_WHYNTER
//#define DECODE_FAST

//#define DECODE_DISTANCE_WIDTH // Universal decoder for pulse distance width protocols
//#define DECODE_HASH         // special decoder for all protocols

//#define DECODE_BEO          // This protocol must always be enabled manually, i.e. it is NOT enabled if no protocol is defined. It prevents decoding of SONY!

//#define DEBUG               // Activate this for lots of lovely debug output from the decoders.

//#define RAW_BUFFER_LENGTH  180  // Default is 112 if DECODE_MAGIQUEST is enabled, otherwise 100.

#include <Arduino.h>
#include <Keyboard.h>

/*
 * This include defines the actual pin number for pins like IR_RECEIVE_PIN, IR_SEND_PIN for many different boards and architectures
 */
#include "PinDefinitionsAndMore.h"
#include <IRremote.hpp> // include the library
#define LED_PIN   7

// #define _DBG_          // Se abilitato mostra sulla seriale tutti i messaggi dei codici ricevuti

int nact =  0 ;
String lbl[15] ;

void setup() {
    Serial.begin( 115200 );
    // Just to know which program is running on my Arduino
    // Serial.println(F("START " __FILE__ " from " __DATE__ "\r\nUsing library version " VERSION_IRREMOTE));
```

```
    // Start the receiver and if not 3. parameter specified, take LED_BUILTIN pin from the
internal boards definition as default feedback LED
    IrReceiver.begin( IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK ) ;

    Serial.print( F( "Ready to receive IR signals of protocols: " ) ) ;
    printActiveIRProtocols( &Serial ) ;
    Serial.println( F( "Ricezione sul Pin " STR(IR_RECEIVE_PIN) ) ) ;
    pinMode( LED_PIN, OUTPUT ) ;
}

void loop() {
    /*
     * Check if received data is available and if yes, try to decode it.
     * Decoded result is in the IrReceiver.decodedIRData structure.
     *
     * E.g. command is in IrReceiver.decodedIRData.command
     * address is in command is in IrReceiver.decodedIRData.address
     * and up to 32 bit raw data in IrReceiver.decodedIRData.decodedRawData
     */
    if ( IrReceiver.decode() ) {

       /*
       * Print a short summary of received data
       */
       #if defined( _DBG_ )
           IrReceiver.printIRResultShort( &Serial ) ;

           IrReceiver.printIRSendUsage( &Serial ) ;

           if ( IrReceiver.decodedIRData.protocol == UNKNOWN ) {
              Serial.println( F( "Received noise or an unknown (or not yet enabled) protocol" ) );
              // We have an unknown protocol here, print more info
              IrReceiver.printIRResultRawFormatted( &Serial, true ) ;
           }

       #endif

       /*
       * !!!Important!!! Enable receiving of the next value,
       * since receiving has stopped after the end of the current received data packet.
       */
       IrReceiver.resume() ; // Enable receiving of the next value

       /*
       * Finally, check the received data and perform actions according to the received command
       */
       if ( IrReceiver.decodedIRData.command ==0x20 ) {
          // "Pagina Su"
          GoTasto ( 0 , 1 , "Pagina Su" ) ;

       } else if ( IrReceiver.decodedIRData.command ==0x21 ) {
                 // "Pagina giù"
                 GoTasto ( 1 , 1, "Pagina Giù" ) ;

       } else if ( IrReceiver.decodedIRData.command ==0x50 ) {
                 // "Freccia Su"
                 GoTasto ( 0, 0, "Freccia Su" ) ;
```

```
      } else if ( IrReceiver.decodedIRData.command ==0x51 ) {
                // "Freccia Giù"
                GoTasto ( 1 , 0 , "Freccia Giù" ) ;
      }
   }
}
// Funzioni Tastiera
void GoTasto ( int act , int Pagina , String lbl ) {
     nact ++ ;
     if ( nact < 2 ) {       // Riduce i comandi doppi
       digitalWrite( LED_PIN , HIGH ); //accende il Led spia di esecuzione comandi
       switch ( act ) {
            case 0 :     // Indietro
                switch ( Pagina ) {
                      case 0 :                // Opero come freccia indietro
                           Keyboard.press( KEY_UP_ARROW ) ;
                           break ;

                      case 1 :                // Opero come Pagina Su
                           Keyboard.press( KEY_PAGE_UP ) ;
                           break ;
                }
                break;

            case 1 :     // Avanti
                switch ( Pagina ) {
                      case 0 :               // Opero come freccia avanti
                           Keyboard.press( KEY_DOWN_ARROW ) ;
                           break ;

                      case 1:                // Opero come Pagina Avanti
                           Keyboard.press(KEY_PAGE_DOWN) ;
                           break ;
                }
                break ;
     }

     delay( 100 ) ;
     // Sarebbe più indicato resettare il singolo comando dato ma così il codice è più compatto
     Keyboard.releaseAll() ;   // Resetta tutti i comandi tastiera
     delay( 120 ) ;  // Attendi prima di inviare un altro comando
     digitalWrite( LED_PIN , LOW ) ; //Spegni il Led spia di esecuzione comandi
     Serial.print( F( "Ricevuto sul pin " STR(IR_RECEIVE_PIN) " il comando " ) ) ;
     Serial.println( lbl ) ;
     Serial.println( "" ) ;
     }
     if ( nact > 1 ) {  // in caso di doppio comando azzera il contatore
        nact = 0 ;
     }
}
```

Nota:

Questo sorgente necessita di un file aggiuntivo denominato "PinDefinitionsAndMore.h" ma per questo sketch uso l'originale della libreria IRemote.

Ora abbiamo anche una ricevente che esegue i comandi eseguiti sia dal nostro progetto base che da un telecomando che usa la codifica RC5 (Esempio Tv Philips).

I tasti adoperati sono P+ / P- per Pagina Giù/ Pagina Su e ovviamente i tasti direzionali Freccia su e Freccia giù.

L'aggiunta di una parte attuatrice telecomandata è interessante in quanto modificandone il codice è possibile disporre di comandi wireless personalizzati a basso costo, pensate quante azioni potete eseguire sul pc/tablet associandole ad un numero digitato su di un qualsiasi telecomando TV economico, magari di recupero.

Mi auguro che sia di aiuto per chi non può usare le mani, pensiamo a dei malati di SLA o a chi deve scorrere delle istruzioni avendo le mani occupate, fino ad un uso più ludico quale scorrere spartiti musicali, cosa impossibile da farsi autonomamente specialmente se fa uso anche della pedaliera, ma ora "con un soffio" si risolve il problema.

Concludo ringraziando tutti quanti mi hanno aiutato in questa avventura, in special modo al gruppo **Bilug** di cui faccio parte e che da sempre si occupa non solo di Linux e di open-source ma anche di quanto ne gravita attorno.

Biella 18/08/2023 Pierpaolo Martinello