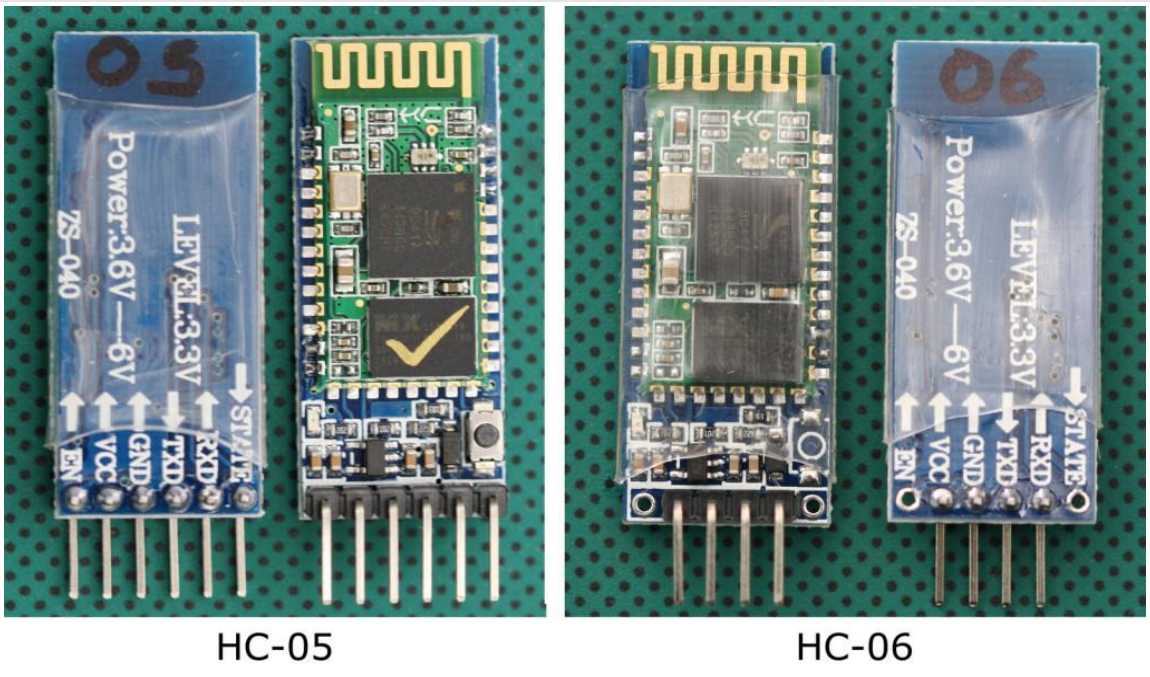
Moduli Bluetooth HC-05 e HC-06 zs-040

Se modulo con LED blu nell'angolo in alto a sinistra, e' un modello più recente con un firmware leggermete diverso.

HC05s e HC-06 sono moduli Bluetooth sono standard alcuni siglati zs-040.

Le schede zs-040 differiscono da alcune di altri moduli perchè hanno un pin EN anzichè un pin KEY.



HC-05 HC-06

L'HC-05 zs-040 e il HC-06 zs-040 sono diversi ed hanno inoltre un firmware differente:

-L'HC-06 non ha un pulsante ;

-L'HC-06 ha 4 pin header;

-L'HC-06 non ha collegati i pin 31-34.

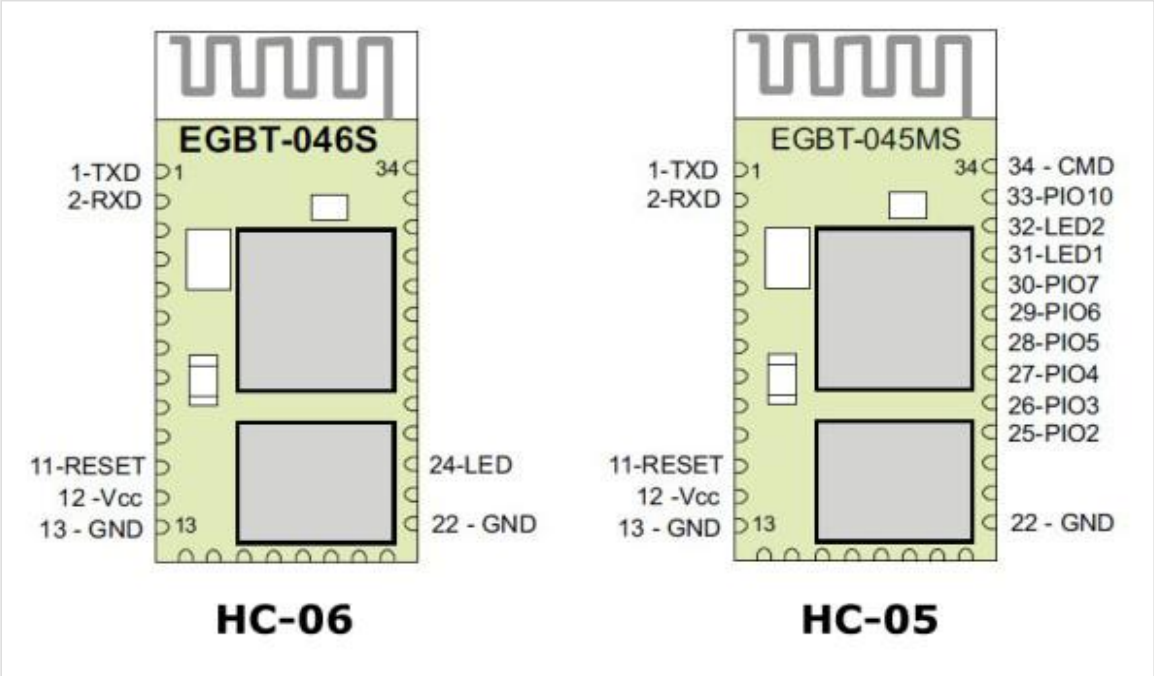
HC-05 può essere master o slave mentre l'HC-06 può essere solo slave.
Questo significa che l'HC-05 può richiedere la connessione ad altri dispositivi mentre l'HC-06 può solo accettarla e non inviarla.

Si basano su moduli Bluetooth EGBT-045MS/EGBT-046S e vengono caricati con firmware SPP con comunicazione UART.

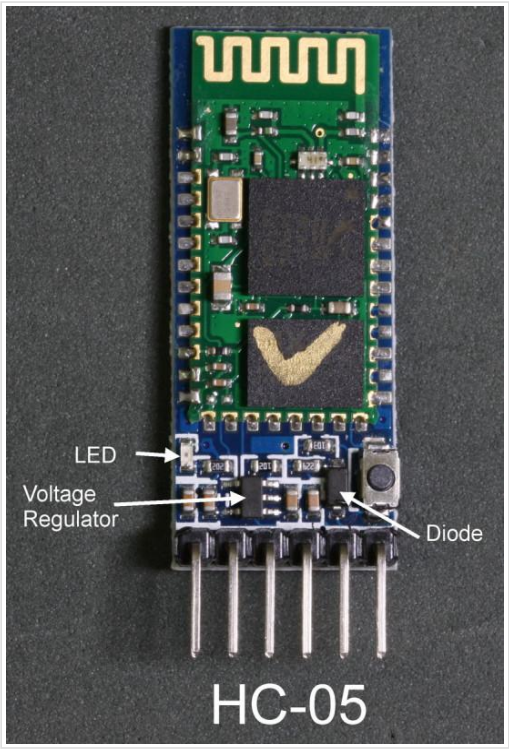
Specifiche dei moduli Bluetooth EGBT-045MS/EGBT-046S

Radio Chip: CSR BC417 con Memoria : Esterna 8Mbit Flash
Output Power: -4 to +6dbm Classe 2;
Sensibilità: -80dbm , Bit Rate: EDR, fino a 3Mbps, Interfaccia: UART, Antenna: Built-in;
Dimensioni: 27W x 13H mm , Tensione: 3.1 - 4.2VDC , Corrente: 40mA max

Per quanto riguarda i moduli EGBT-045MS e EGBT-046S hanno lo stesso hardware e l'unica differenza è il firmware.

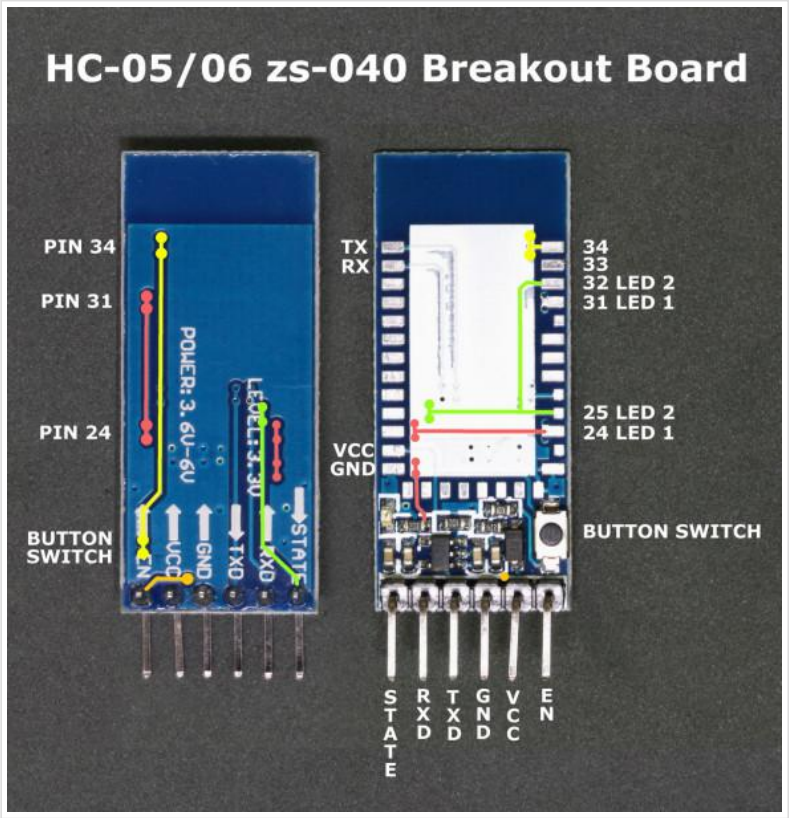


Le schede zs-040 includono un regolatore di tensione 3.3V e questo gli permette di accettare una VCC di 3.6V a 6V sul pin principale Vcc, tuttavia, i pin RX e TX sono ancora a 3,3V.
Arduino accetterà 3.3V come segnale ALTO così il HC-05 / 06 TX pin(out) può essere collegato direttamente ad un 5V Arduino.
L' HC-05/06 RX pin (in) non può accettare 5V , usare un partitore di tensione composto da 2 resistenze, da 1k e 2.2k.



Connessioni esterne della scheda:

L'immagine seguente mostra le connessioni sulla scheda zs-040 e come ono posizionati i pin.



L’HC-06 non ha nè un pin EN nè uno STATE.

LED

La traccia tra il pin 31 e il pin 24 sulla scheda collega il pin di stato LED sul EGBT-045MS al pin di stato LED sulla EGBT-046S.
Il collegamento va quindi al LED sulla scheda.
Questo collegamento può essere usato allo stesso modo la EGBT-045MS e la EGBT-046S.

Pulsanti

Lo switch collega VCC (3.3V) al pin 34 e viene utilizzato per mettere l'EGBT-045MS / HC-05 in modalità AT.
Ne esistono di 2 tipi, “mini mode” e “full mode”. attivando lo switch si attiva la “mini mode” che disattiva tutti i comandi, premendolo nuovamente l’iterrutore si attiverà la “full mode” che riattiverà tutti i comandi.
Rilasciandolo si pone il pin 34 basso.

Per comandi AT visita >> http://www.martyncurrey.com/arduino-with-hc-05-bluetooth-module-at-mode/.

Molti moduli HC-06 non hanno lo switch (lo spazio è vuoto) e di conseguenza il pin 34 è settato alto, quindi non esegue nulla.

Pin di Stato

Dalla foto sopra si può vedere che lo STATE è collegato al pin 32 ed al piedino 25 dei piccoli moduli Bluetooth.
Pin 32 è LED2 è sul EGBT-045MS / HC-05 .
Il pin 32 è basso quando il modulo non è collegato e alto quando invece lo è.
Cio’ determina se l'HC-05 sia effettivamente collegato o meno.
Collegare il pin STATE ad un pin digitale di Arduino e se digitalRead () è alto si conosce se il modulo ha una connessione attiva o meno.
È possibile, ovviamente, collegare un LED al pin STATE come un indicatore visivo di una connessione.

Pin En Mettendo il pin EN su HC-05 in modalita’basso si disabilita il modulo.

Pannello di controllo bluetooth

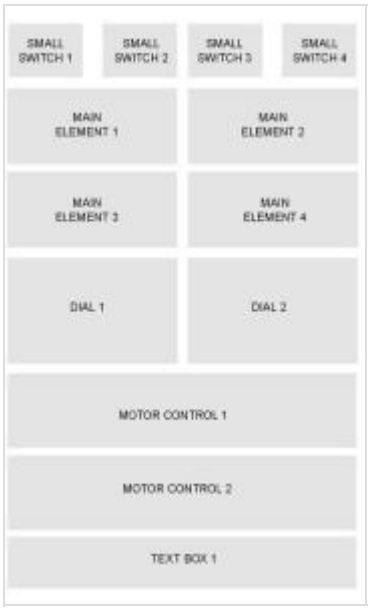
04.03.2017- Aggiornata l'applicazione alla versione 3. Vedere i cambiamenti sotto riportati per i dettagli.

L’applicazione di controllo Bluetooth può essere utilizzata con una vasta gamma di microprocessori ed anche con Arduino.
Questo usa un bluetooth 2.1 SSP perchè molto diffuso e poco costoso, HC-05s e HC-06s.
La versione BLE/Bluetooth 4 è in lavorazione..
L’inizializzazione viene eseguita lato microprocessore.
Il microprocessore invia comandi per impostare i controlli e una volta che i controlli sono stati attivati I dati possono essere inviati dal microprocessore all'applicazione e viceversa.
L'app Android e può essere scaricata da Google Play.

I controlli della APP

Il pannello di controllo ha un layout fisso, lungo la parte superiore ci sono 4 piccoli pulsanti, accanto ai quali ci sono i 4 elementi principali.
Ciascuno di questi può contenere diversi tipi di controllo differenti.
Ci sono 2 quadranti, controlli 2 velocità / direzione ed infine una casella di testo, gli elementi sono visibili solo dopo che sono stati inizializzati.
Cio’ significa che non c'è bisogno di avere controlli inutilizzati sullo schermo.

Nella griglia principale, se solo controllo viene inizializzato su una riga di controllo si espanderà per riempire l'intera riga.



Ogni elemento all'interno del suo gruppo ha un numero di posizione.
La posizione viene utilizzata sia per il comando di inizializzazione che per il comando dei dati

Ci sono 10 elementi di controllo:
- piccolo interruttore
- Grande Interruttore
- Valore
- Pulsante
- Pulsante di commutazione
- Slider
- Barra di avanzamento
- Dial

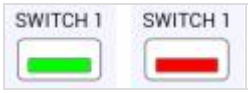
- Velocità / Direzione cursore
- Casella di testo.

Gli elementi della griglia principale possono contenere uno qualsiasi dei seguenti comandi:

- Grande Interruttore
- Valore
- Pulsante
- Pulsante di commutazione
- Slider
- Barra di avanzamento.

Si noti che ogni posizione può contenere un solo controllo.

Piccolo interruttore:



Il piccolo interruttore è un semplice interruttore on / off. Lo stato dell' interruttore commuta facendo click sul rosso o sul verde.

Ci sono 4 piccoli interruttori nella parte superiore del pannello di controllo.
Il titolo e il colore dello sfondo possono essere impostati dall'utente.la posizione degli elementi può esse da 1 a 4.

Inizializzazione comandi	<I,SS,posizione elemento,descrizione(colore)> <I,SS,1,LED 1,24524e255> Questo inizializza il piccolo interruttore in posizione 1 con il titolo di LED 1 e un colore di sfondo 245.245.255.
Comando di invio dati al microprocessore	<SS[posizione][1/0]> <SS11> = interruttore in posizione 1, on <SS10> = interruttore in posizione 1, off
Comando di ricezione dati dal microprocessore	<D,SS,[posizione],[1/0]> <D,SS,1,1> = piccolo interruttore in posizione 1, on <D,SS,1,0> = piccolo interruttore in posizione 1, off

Interruttore grande:

E' simile al piccolo interruttore e commuta allo stesso modo, il suo funzionamento è uguale all'altro.

Inizializzazione comandi	<I,SS, posizione elemento ,descrizione(colore)> <I,SS,1,LED 1,245245255> questo commando seta il piccolo interruttore nella posizione 1 con il nome di LED 1 e schermo di colore 245245255.
Comando di invio dati al microprocessore	<SS[posizione][1/0]> <SS11> = interruttore in posizione 1, on <SS10> = interruttore in posizione 1, off
Comando di ricezione dati dal microprocessore	<D,SW,[posizione],[1/0]> <D,SW,1,1> = interruttore in posizione 1, on <D,SW,1,0> = interruttore in posizione 1, off

Valore:

Il controllo del valore può visualizzare qualsiasi valore alfanumerico.
La lunghezza massima del valore dipende sullo schermo del dispositivo; il più piccolo è di 10 caratteri, il più grande è di 20 caratteri; i valori superiori a questo verranno troncati.
Non ci sono controlli per i caratteri non stampabili e l'applicazione cercherà di visualizzare tutto ciò che le viene inviato.

Il controllo visualizzerà un valore che riceve dal microprocessore,
Ci sono 4 elementi di controllo valore nella griglia principale, uno per ogni posizione.
Il nome e il colore possono essere cambiati dall'utente.
La posizione degli elementi varia da 1-4.

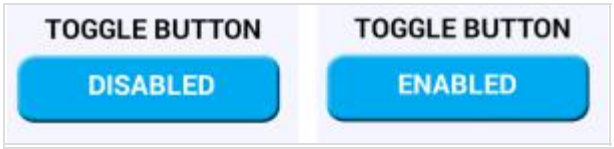
Inizializzazione comandi	<I, VL, numero dell'elemento, descrizione (colore)><I, VL, 2, temperatura, 245.245.245>Inizializza il controllo in posizione 2 con un valore di controllo con il titolo di temperatura e un colore di sfondo 245.245.245 (grigio chiaro)
Comandi inviati dal microprocessore	<D, VL, posizione dell'elemento, valore><D, VL, 2,22.4 °> = Valore alla posizione 2, display 22.4 °

Pulsante:

Questo è un pulsante che quando viene premuto invia i dati al microprocessorei.
Ci sono 4 elementi di controllo pulsante nella griglia principale, uno in ogni posizione.
Il nome, il colore e il testo del pulsante possono essere impostati dall'utente. il numero di elementi varia da 1 a 4.

Inizializzazione comandi	<I, BTposizione dell'elemento, descrizione, il il nome del pulsante (colore)> <I, BT, 3, My Button, Press Me, 255.255.128> Inizializza il controllo pulsante in posizione 3 con titolo My Button e un colore di sfondo 255.255.128 (giallo)
Comando di invio dati al microprocessore	<BT [posizione] 1> <BT31> = posizione tasto 3 cliccato

Interruttore

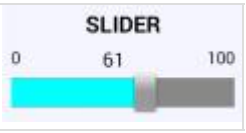


Questo è un' interruttore a 2 stati che si azione alla pressione o di uno o dell'altro tasto.

Ci sono 4 elementi di controllo pulsante di commutazione nella griglia principale, uno in ogni posizione. Il nome e il colore, possono essere impostati dall'utente. Il numero di elementi varia da 1 a 4.

Inizializzazione comandi	<I, TB, numero di elementi, descrizione, pulsante etichetta 1, pulsante etichetta 2 (colore)> <I, TB, 1, My Button, attivato, disattivato 255.255.128>Inizializza il controllo in posizione 1 di un interruttore. Il nome del pulsante nello stato 1 è "Abilitato", il nome nello stato 2 è "Disabilitato", colore di sfondo 255.255.128
Comandi inviati al microprocessore	<TB [posizione] [1/0]> <TB20> = Tasto nella posizione 2, stato 1, <TB21> = Tasto nella posizione 2, stato 2
Comandi inviati dal microprocessore	<D, TB, [posizione], [1/0]><D, TB, 2,0> = Tasto in posizione 2, stato 1,<D, TB, 2,1> = Tasto nello stato 2

Slider (cursore):



Si tratta di un cursore standard nel quale l'utente può impostare il valore minimo (partendo da 0) e il valore massimo (fino a 9999).
Nota, un cursore con un valore massimo ha una risoluzione inferiore e incrementerà a gradini, ad esempio un cursore con un minimo di 1 e un massimo di 1024 incrementi a passi di 10 (o 5 se utilizzando solo un cursore sulla riga) .
A causa di questo, l'impostazione della dei dati non può essere esatta.
Ci sono 4 elementi del cursore nella griglia principale, uno in ogni posizione.
Gli Sliders possono inviare e ricevere dati.
I cursori inviano i dati non appena vengono spostati e questo può causare comandi di avviso di invio.
L'utente può come sempre impostare nome e colore dello Slider.

Inizializzazione	<I, SL, numero di elemento, descrizione, min, max (colore)> <I, SL, 4, PWM, 0,255,128255255> Inizializza il controllo in posizione 4 ad un cursore, il nome è PWM. Il valore minimo è 0, il massimo è 255 e il colore di sfondo è 128.255.255 (luce blu / acqua)
Comandi inviati dal microprocessore	<D, SL, [posizione dell'elemento], [valore]>< D, SL, 4.100> = cursore nella posizione dell'elemento 4, posizione de lcursore a 100. Massimo per il valore dei dati è 9999 a meno che un valore minore non venga impostato dall'utente.
Comandi inviati al microprocessore	<SL [posizione elemento] [valore]> <SL40100> = cursore nella posizione dell'elemento 4, impostare la posizione del cursore a 100. Il valore dei dati è a 4 cifre.

Barra di progresso:

La barra di avanzamento è una barra di avanzamento standard.
L'utente può impostare il valore minimo (partendo da 0) e il valore massimo (fino a 9999).

Ci sono 4 elementi di controllo per la barra di avanzamento nella griglia principale, uno in ogni posizione. La bara di avanzamento può solamente ricevere i dati, non inviarli. Il numero di elementi varia da 1 a 4.

L'utente può impostare nome e colore.

Inizializzazione comandi	<I, PB, numero di elemento, descrizione, min, max (colore)> <I, PB, 2, Tempo rimasto, 0,100,128255255> Inizializza il controllo in posizione 2 per una barra di avanzamento, il titolo è Tempo rimasto, il valore minimo è 0, il valore massimo è 100 e il colore di sfondo è 128.255.255
Comandi invia dal microprocessore	<D, PB, [posizione dell'elemento], [valore]> <D, PB, 4,10> = Progress Bar nella posizione dell'elemento 4, posizione impostata a 10

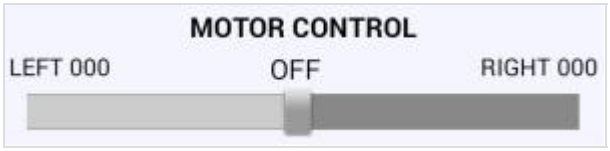
Quadrante:

Direttamente sotto la griglia ci sono 2 quadranti.
L'utente può impostare il valore minimo (partendo da 0) e il valore massimo (fino a 9999).
L'utente piò impostare nome e colore a piacimento.
Può solamente ricevere i dati, non inviarli.
Il numero di elementi varia da 1 a 4.

Inizializzazione comandi	<I, DI, numero di elemento, descrizione, min, max (colore)> <I, DI, 1, temperatura, 0,75,128255255> Inizializza il selettore in posizione 1 con il nome di temperatura. Valore minimo di 0, il valore massimo di 75 e un colore di sfondo 128.255.255
Comandi inviati dal microprocessore	<D, DI, [posizione dell'elemento], [valore]> <D, DI, 1,25> = Quadrante 1, posizione n ° 25

Nota: Utilizzare quadranti può rallentare l'applicazione, soprattutto se ci sono molti altri controlli in uso o l'invio di dati molto frequenti.

Controllo bidirezionale:



Questo controllo è stato progettato per consentire un facile controllo sui motori, ma può essere usato per altre cose. agisce come un cursore normale tranne che per il valore che va da Left100 a Right100 anziché da 0 a 200.
C'è una posizione OFF al centro ed i valori sono fissi.
Ci sono 2 controller motore; 1 e 2.
Questi controlli possono solamente inviare dati ai motori e si attivano non appena vengono spostati, generano un flusso di dati consistente.
L'utente può cambiare nome e colore, la posizione varia da 1 a 2.

Inizializzazione comandi	<I, MT, numero di elemento, descrizione (colore)> <I, MT, 1, Direzione, 255.255.255> Inizializza il controller a 2 vie in posizione 1 con il nome di direzione e un colore di 255.255.255 (bianco) di fondo

Comandi inviati al microprocessore	<DMT [posizione dell'elemento] [valore]> <DMT1L025> = regolatore del motore 1, posizione sinistra 25

Casella di testo:

Questo tasto permette di inviare al microprocessore qualsiasi tipo di testo tramite la pressione del tasto INVIO.
Il testo deve essere compreso tra <>.
Dopo l'invio del testo la casella viene pulita, nel testo si possono aggiungete anche tag inserendo TX sempre compresi fra <>.
L'utente può cambiare nome e colore del pulsante.

Inizializzazione comandi	<I, TX, numero di elemento, descrizione (colore)> <I, TX, 1, Casella di testo, 255.255.255> Inizializza la Casella di testo con il nome di casella di testo e un colore di sfondo 255255255
Comandi inviati al microprocessore: con aggiunta del tag Tx	<DTX [posizione] [Testo]> <DTX1HELLO> - Text = "CIAO"
Comandi inviati al microprocessore: con aggiunta del tag Tx	<[Testo]> <CIAO> - Testo = "CIAO"

Colore di sfondo:

Altri comandi di inizializzazione:

titolo

<T, titolo principale> Imposta il titolo principale per il pannello di controllo. Il valore predefinito è il mio controller

Frequenza di aggiornamento

<R, valore> Imposta la frequenza di aggiornamento app. Deve essere un valore da 1 a 10. 1 è veloce, 10 è lento.
1 = 10ms. 10 = 100 ms. Il valore predefinito è 2.

EOI

Indica all'applicazione quando inizializzare i comandi.

Messaggi

Il comando messaggio viene utilizzato per creare un pop-up finestra di messaggio sul dispositivo Android. Il comando viene inviato dal microprocessore, può essere utilizzato in qualsiasi momento; durante l'inizializzazione o come un comando di dati. Non ci sono controlli e l'applicazione cercherà di visualizzare che le si inivia.
<M, testo del messaggio> Visualizzazione di un pop-up finestra di messaggio in app

Connessioni

Quando l'applicazione stabilisce una connessione invia un messaggio con scritto <CONNECT>. Può stabilire connessione anche prima dell'inizializzazione dei comandi.

Reset

L'applicazione può essere azzerata facendo clic sul tasto RESET in app o con l'invio di un comando di RESET dal microprocessore. In entrambi i casi i comandi di inizializzazione devono essere reinseriti.
Quando il tasto RESET in app viene cliccato un messaggio con scritto <RESET> viene trasmesso al microprocessore. <Ripristina> Ripristina da remoto l'applicazione.

Disconnessione

Se l'utente chiude il collegamento con l'applicazione l'applicazione invia un messaggio <DISCONNECT>.

Sei qui?

Quando si utilizza la base HC-06 e HC-05s è difficile per l'Arduino sapere se la connessione Bluetooth è in realtà attiva. Uno dei modi per confermare che c'è una connessione attiva è quella di ricevere i dati e questo è dove il Are You There, <AYT>, il comando entra in gioco. Quando l'applicazione riceve il comando <AYT> risponde con <YIA> (Sì ci sono).
Per verificare la presenza di una connessione attiva continua a inviare il comando <AYT>, una volta al secondo funziona bene. Se si ottiene la risposta, allora tutto funziona correttamente.
Nessuna risposta e non si sa la connessione è stata persa.
Se si vuole fare questo attraverso l'hardware utilizzare un HC-05 con un pin di STATO.
Quando viene stabilita una connessione pin STATE sul HC-05 va alto.
Si può semplicemente collegare questo al Arduino e mantenere il controllo sul suo stato.

Esempi:

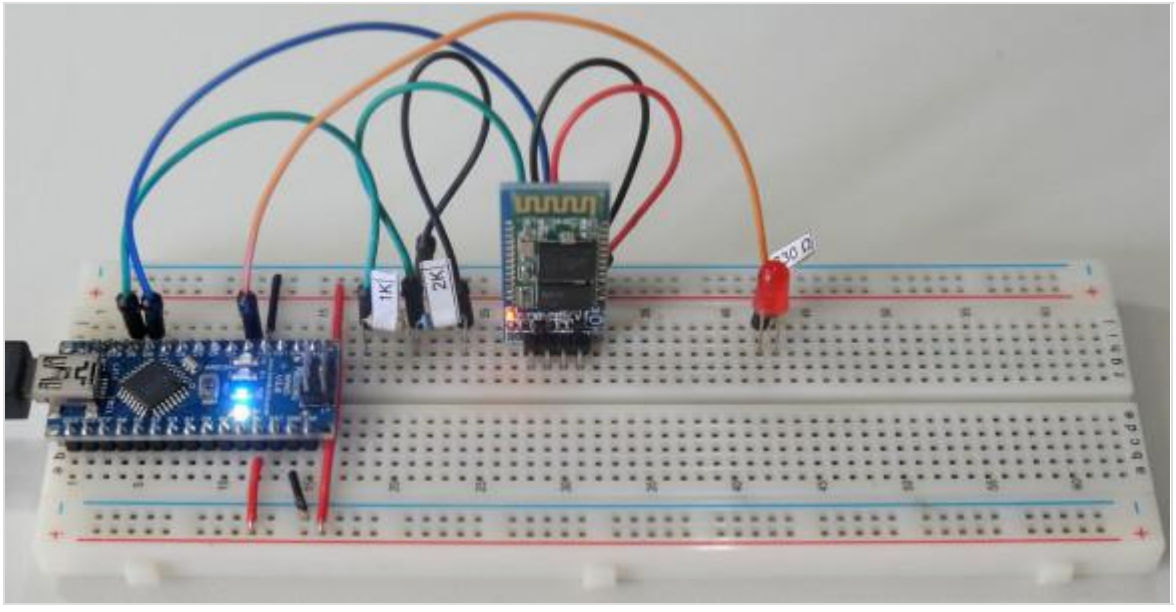
Tutti gli esempi di seguito sono per l'Arduino e utilizzano un normale HC-06 con una velocità di trasmissione di 9600.
Diverse velocità di trasmissione possono essere utilizzate solo abbinandola con la velocità di trasmissione utilizzato per aprire la connessione seriale al modulo Bluetooth nel disegno Arduino.
L'HC-05 può essere utilizzato anche in slave.
Tutti gli esempi presuppongono che il modulo Bluetooth sia già accoppiato con il dispositivo Android.

Esempio 1a: Controllare un LED usando un piccolo interruttore:

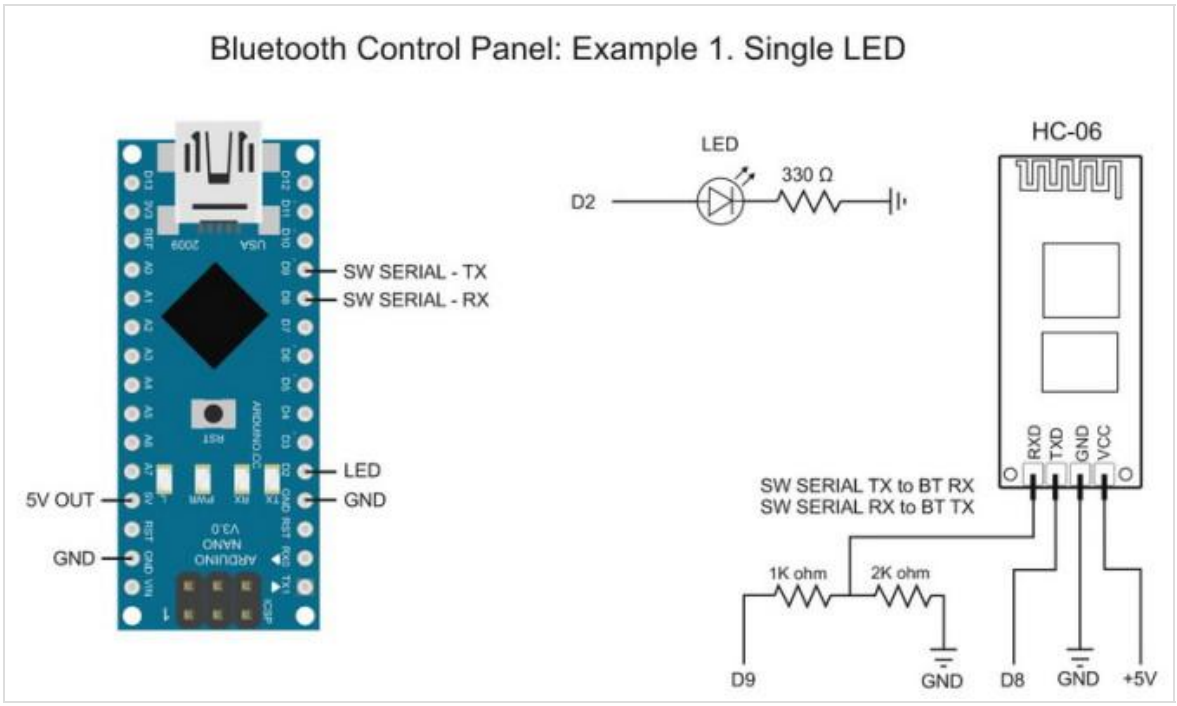
Un esempio di base per iniziare. In questo esempio si usa un piccolo interruttore per controllare un LED. Il controllo è unidirezionale, da App all' Arduino e Arduino semplicemente reagisce ai comandi che riceve.
Arduino invia i comandi di inizializzazione come parte della funzione di impostazione. Ciò significa che se l'applicazione è resettata Arduino dovrà anche essere ripristinato e viceversa.

Circuito

Il circuito ha un HC-06 collegato ai pin D8 e D9 sul Arduino e un LED sul pin D2 (inclusa resistenza adeguata).
Si usa un LED rosso e una resistenza da 330 ohm.



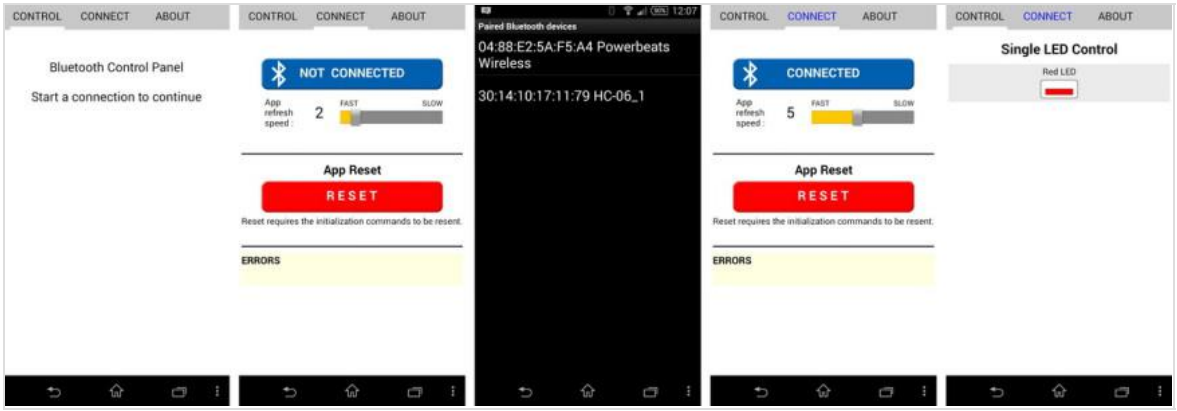
Traduzione di simone proetto 5^B 2016/2017 I.T.T. B.Pasqual Cesena



Controllare due volte le connessioni.
Assicurarsi che:
– pin Arduino RX sia connesso al pin BT TX
– pin Arduino TX sia connesso al pin BT RX attraverso il partitore di tensione.
– i resistori siano messi correttamente .I resistori da 1K vanno connessi ad Arduino

Usare l’ applicazione

Aprire l’ applicazione



– La pagina principale richiederà di iniziare la connessione
– Vai sulla pagina CONNECT.
– Cliccare il bottone bluetooth.Questo attualmente visualizzerra’ NOT CONNECTED.
– Quando clicchi il tasto BT viene mostrata una lista di dispositivi Bluetooth accessibili .
– Selezionare la HC-06 (o un qualunque altro modulo che si stia usando).
– L Applicazione si connetterà all’HC-06.
– Se la connessione è avvenuta con successo il tasto cambierà a CONNECTED e il colore della barra CONNECT cambierà e diventerà blu.
– Se i comandi di inizializzazione vengono ricevuti da Arduino, i controlli verranno inizializzati e l’applicazione tornerà alla pagina principale.
– il LED di Arduino è usato per mostrare quando l’ inizializzazione dei comandi è stata spedita

Cliccando sul pulsante esso cambierà da rosso a verde e ricomincerà.Allo stesso tempo i comandi saranno spediti a Arduino per controllare il LED.

Arduino Sketch

Dopo SETUP iniziale attendere fino al messaggio di CONNECT.
A messaggio ricevuto verranno inviati i comandi di inizializzazione.
Dopo dichè controlla ripetutamente i dati ricevuti e controlla se il data è un comando o no.

Main Parts

Nel settaggio delle funzioni lo sketch aspetta fino al messaggio CONNECT.
Quando riceve questo messaggio lui manda i comandi di inizializzazione.

```
// aspetta il messaggio CONNECT
boolean connected = false;
while(!connected)
{
        recvWithStartEndMarkers();
        if (strcmp ("CONNECT",receivedChars) == 0)
        {
                connected = true;
                if (DEBUG) { Serial.println("Connected");   }
        }
}
```

Dalla ricezione del messaggio CONNECT partono i comandi di inizializzazione.

```
BTserial.print("<T,Single LED Control>");      // titolo del pannello di controllo
BTserial.print("<I,SS,1,Red LED>");            //inizializzazione pulsante 1. Switch description = Red LED
BTserial.print("<R,5>");                       // imposta la velocità di update dell'applizazione a 50ms
BTserial.print("<EOI>");                       // fine del comando di inizializzazione

if (DEBUG) { Serial.println("Init commands sent"); }
digitalWrite(13,HIGH);                        //   accendi il LED della scheda mostrando di essere connesso
```

Dopodichè, l'applicazione mantiene il controllo per i dati che arrivano e successivamente processa ogni dato che riceve.

```
void loop()   {
      recvWithStartEndMarkers();              // controlla se sono stati ricevuti nuovi dati
      if (newData)   { processCommand(); }    // se è presente un nuovo dato guardare se è di comando
```

La funzione processCommand() controlla i dati ricevuti. A comando di ON o OFF il LED allora reagisce di conseguenza.

I due comandi usati sono SS10 e SS11.
– SS10 è il pulsante 1, setta il LED off
– SS11 è il pulsante 1, setta il LED on.

```
void processCommand(){
    if (DEBUG)
    {
            Serial.print("receivedChars = ");
            Serial.println(receivedChars);
    }

    if (strcmp ("SS10",receivedChars) == 0)
    {
```

```
        digitalWrite(LED_RED_PIN,LOW);
        if (DEBUG) { Serial.println("LED LOW");   }
    }

    if (strcmp ("SS11",receivedChars) == 0)
    {
        digitalWrite(LED_RED_PIN,HIGH);
        if (DEBUG) { Serial.println("LED HIGH");   }
    }

    receivedChars[0] = '\0';
    newData = false;}
```

Full Arduino Sketch

```
/*
* Sketch: BCP_Example_01a_Single_LED
* By Martyn Currey
* 17.07.2016
*
* Requires the Bluetooth Control Pabrl Android App. Can be downloaded from Google Play.
* See http://www.martyncurrey.com/bluetooth-control-panel
*
* setta il LED on e off da un' applicazione android
* Usa i seguenti Pin
*
* D8 - AltsoftSerial RX
* D9 - AltsoftSerial TX
* D2 - LED
*/
 // AltSoftSerial uses D9 for TX and D8 for RX. While using AltSoftSerial D10 cannot be used for PWM.
// Remember to use a voltage divider on the Arduino TX pin / Bluetooth RX pin
// Download AltSoftSerial from https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html
 #include <AltSoftSerial.h>
AltSoftSerial BTserial;
 // Change DEBUG to true to output information to the serial monitor
boolean DEBUG = true;
 // variabili usate per i dati ricevuti
const byte maxDataLength = 20;char receivedChars[21] ;
boolean newData = false;
 // variabili generali
const byte LED_RED_PIN = 2;
 void setup()   {
    //   setta il pin LED sulla basetta come uscita
    pinMode(13, OUTPUT);
    digitalWrite(13,LOW);

    pinMode(LED_RED_PIN, OUTPUT);
```

```
    digitalWrite(LED_RED_PIN,LOW);

    if (DEBUG)
    {
        //   apri la comunicazione seriale per il debugging
        Serial.begin(9600);
        Serial.print("Sketch:     ");    Serial.println(__FILE__);
        Serial.print("Uploaded: ");     Serial.println(__DATE__);
        Serial.println(" ");
    }
    // Apri un software di connessione seriale pe il modulo Bluetooth
    BTserial.begin(9600);
    if (DEBUG)   {    Serial.println("AltSoftSerial started at 9600");    Serial.println(" ");   }
     newData = false;
     // aspetta il messaggio CONNECT
    boolean connected = false;
    while(!connected)
    {
          recvWithStartEndMarkers();
          if (strcmp ("CONNECT",receivedChars) == 0)
          {
                connected = true;
                if (DEBUG) { Serial.println("Connected");   }
          }
    }

    receivedChars[0] = '\0';
    newData = false;

    BTserial.print("<T,Single LED Control>");      // titolo pannelo di controllo
    BTserial.print("<I,SS,1,Red LED>");            // inizializzazione pulsante 1. Switch description = Red LED

    BTserial.print("<R,5>");                       // setta la velocita di update a 50ms
    BTserial.print("<EOI>");                       // fine del comando di inizializzazione

    if (DEBUG) { Serial.println("Init commands sent"); }
    digitalWrite(13,HIGH);                         //   accendi il LED sulla basetta e mostra se è connessa

    delay(100);                                    // piccolo ritardo per lasciare settare l'applicazione
     } // void setup()

void loop()   {
     recvWithStartEndMarkers();                   //controlla se sono stati ricevuti nuovi dati
     if (newData)   { processCommand(); }         // se ci sono nuovi dati guardare se sono comandi


/**************************************
* Function processCommand
```

```
* parses data commands contained in receivedChars[]
* receivedChars[] has not been checked for errors
*
* passed:
*
* global:
*          receivedChars[]
*          newData
*
* Returns:
*
* Sets:
*          receivedChars[]
*          newData
*
*/void processCommand(){
    if (DEBUG)
    {
          Serial.print("receivedChars = ");
          Serial.println(receivedChars);
    }

    if (strcmp ("SS10",receivedChars) == 0)
    {
          digitalWrite(LED_RED_PIN,LOW);
          if (DEBUG) { Serial.println("LED LOW");   }
    }

    if (strcmp ("SS11",receivedChars) == 0)
    {
          digitalWrite(LED_RED_PIN,HIGH);
          if (DEBUG) { Serial.println("LED HIGH");   }
    }

    receivedChars[0] = '\0';
    newData = false;}

 // function recvWithStartEndMarkers by Robin2 of the Arduino forums
// See   http://forum.arduino.cc/index.php?topic=288234.0
/*******
* Function recvWithStartEndMarkers
* reads serial data and returns the content between a start marker and a end marker.
*
* passed:
*
* global:
*          receivedChars[]
*          newData
```

```
*
* Returns:
*
* Sets:
*          newData
*          receivedChars
*
*/
void recvWithStartEndMarkers(){
      static boolean recvInProgress = false;
      static byte ndx = 0;
      char startMarker = '<';
      char endMarker = '>';
      char rc;
      if (BTserial.available() > 0)
      {
            rc = BTserial.read();
            if (recvInProgress == true)
            {
                  if (rc != endMarker)
                  {
                        receivedChars[ndx] = rc;
                        ndx++;
                        if (ndx > maxDataLength) { ndx = maxDataLength; }
                  }
                  else
                  {
                        receivedChars[ndx] = '\0'; // terminazione della riga
                        recvInProgress = false;
                        ndx = 0;
                        newData = true;
                  }
            }
            else if (rc == startMarker) { recvInProgress = true; }
      }}
```

Esempio 1b: controllare il LED usando un pulsante.
Sketch con il settaggio iniziale e resetta

Esempio con un singolo LED ma introduce uno stato di inizializzazione ed il reset dell'applicazione .
Come gli esempi precedenti il controllo è univoco dall'applicazione.
Esempio simile al precedente con sotware differente.

I comandi di inizializzazione con funzioni.
Chiamare la funzione ogni volta ed inviare i comandi di inizializzazione ogniqualvolta che vogliamo.

```
void sendInitCommands(){
    BTserial.print("<T,Single LED Control>");    // titolo pannello di controllo
    BTserial.print("<I,SS,1,Red LED>");          // Inizializzazione pulsante1. Switch description = Red LED
    BTserial.print("<R,5>");                     // Setta la velocità di update a 50ms
```

```
    BTserial.print("<EOI>");                    // fine del comando di inizializzazione
        delay(100);                             // piccolo ritardo per dare tempo all'applicazione di settarsi.

    initialized = true;

    if (DEBUG) { Serial.println("Init commands sent"); }
    digitalWrite(13,HIGH);}
```

Funzione di reset , semplicemente fa si che il LED si spenga quando attivo il reset.

```
void reset(){
    digitalWrite(LED_RED_PIN,LOW);}
```

Il comando che accende o spegne il LED rosso , ma ci sono due nuovi comandi nel processCommand() CONNECT e RESET.

Piuttosto che controllare solo per il messaggio di CONNECT nella funzione di setup noi stiamo attualente controllando nella funzione main del processCommand().

Ciò significa che noi controlliamo sempre per questo e l'applicazione può essere resettata senza dover resettare Arduino.

Quando il pulsante reset dell'applicazione è cliccato un messaggio di RESET è inviato ad Arduino.

Ciò permette di resettare i comandi di inizializzazione.
In questo esempio noi semplicemente rispediamo gli stessi comandi ma puoi inviarne di differenti se lo desideri.

Se resetti Arduino è necessario cliccare il pulsante RESET nell'applicazione ma non è necessario far ripartire l'applicazione.

```
    if (strcmp ("CONNECT",receivedChars) == 0)
    {
        sendInitCommands();
        if (DEBUG) { Serial.println("CONNECT message received"); }
    }

    else if (strcmp ("RESET",receivedChars) == 0)
    {
        reset();
        sendInitCommands();
        if (DEBUG) { Serial.println("RESET received"); }
    }
```

Il loop principale è stato espanso e adesso usa una variabile inizializzata.
Se l'inizializzato è falso noi accendiamo il LED .
Quando il messaggio di CONNECT viene ricevuto l'inizializzato viene impostato vero e la seconda parte del loop è eseguita.

La seconda parte del loop semplicemente controlla i dati che stanno arrivando e se ne riceve qualcuno lo controlla per vedere se è di comando.

Questo esempio è abbastanza simile al primo , noi aspettiamo il messaggio CONNECT e successivamente spedisce i comandi di inizializzazione.
Dopodichè aspettiamo i comandi per impostare il LED acceso e spento.

C'è comunque una piccola differenza.
Aspettando il messaggio CONNECT nel loop principale e questo , se desiderato, permette di fare altre cose mentre si aspetta.

```
void loop()   {
    if (!initialized)
    {
        // illumina il LED per mostrare che si sta aspettando il messaggio CONNECT
        if (   millis()-flashStartTime > flashFreqTime )
```

```
        {
            flashStartTime = millis();
            if (digitalRead(13) == HIGH) {digitalWrite(13,LOW);}
            else                         {digitalWrite(13,HIGH);}
        }

         // continua a controllare il emssaggio di CONNECT
         recvWithStartEndMarkers();                    // controlla per vedere se noi abbiamo ricevuto un nuovo comando
         if (newData)  {   processCommand();  }     //   se noi abbiamo un nuovo comando, allora fa qualcosa
    }


    if (initialized)
    {
        recvWithStartEndMarkers();                   // controlla per vedere se abbiamo ricevuto un nuovo dato
        if (newData)  {   processCommand();  }     // se abbiamo un nuovo dato contorlla se è un comando
    }


    // un codice può essere inserito qui per fare ciò che desideri. Ciò significa che lo sketch puo fare altro
    //anche se le connessioni non sono applicate all'applicazione
} // void loop()
```

Full Sketch

Controlliamo i dati ricevuti e questo potrebbe portare a problemi ma con inizializzazione possiamo decidere quando e quale comando processare.
Per esempio, se non é stato inizializzato e si riceve un dato di comando potremmo ignorarlo, o mostrare un errore da qualche parte , o salvarlo più tardi.

```
/*
* Sketch: BCP_Example_01b_Single_LED_INIT_STATE
* By Martyn Currey
* 17.07.2016
* Written in Arduino IDE 1.6.3
*
* Requires the Bluetooth Control Pabrl Android App. Can be downloaded from Google Play.
* See http://www.martyncurrey.com/bluetooth-control-panel
*
* Turn an LED on and off from an Android app
* Uses the following pins
*
* D8 - AltsoftSerial RX
* D9 - AltsoftSerial TX
* D2 - LED
*
*/
 // AltSoftSerial uses D9 for TX and D8 for RX. While using AltSoftSerial D10 cannot be used for PWM.
// Remember to use a voltage divider on the Arduino TX pin / Bluetooth RX pin
// Download from https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html
```

```
#include <AltSoftSerial.h>
AltSoftSerial BTserial;
// Change DEBUG to true to output debug information to the serial monitor
boolean DEBUG = true;
// Variables used for incoming dataconst byte maxDataLength = 20;char receivedChars[21] ;
boolean newData = false;
// Vaiables const byte LED_RED_PIN = 2;
boolean initialized = false;unsigned long flashStartTime = 0;unsigned long flashFreqTime = 250;
void setup()  {
    // Set the onboard LED pin for output
    pinMode(13, OUTPUT);
    digitalWrite(13,LOW);

    pinMode(LED_RED_PIN, OUTPUT);
    digitalWrite(LED_RED_PIN,LOW);

    if (DEBUG)
    {
        // open serial communication for debugging
        Serial.begin(9600);
        Serial.print("Sketch:    ");    Serial.println(__FILE__);
        Serial.print("Uploaded: ");    Serial.println(__DATE__);
        Serial.println(" ");
    }
    //   open software serial connection to the Bluetooth module.
    BTserial.begin(9600);
    if (DEBUG)  {    Serial.println("AltSoftSerial started at 9600");      }

    newData = false;
    initialized = false;
    flashStartTime = millis();
} // void setup()
void loop()  {
    if (!initialized)
    {
        // Flash the built in LED to show the Arduino is waiting for the initialization commands
        if (  millis()-flashStartTime > flashFreqTime )
        {
            flashStartTime = millis();
            if (digitalRead(13) == HIGH) {digitalWrite(13,LOW);}
            else                         {digitalWrite(13,HIGH);}
        }
         // Keep checking for the CONNECT message
         recvWithStartEndMarkers();                  // check to see if we have received any new commands
         if (newData)  {    processCommand();  }     // if we have a new command do something
    }
     if (initialized)
    {
```

```
        recvWithStartEndMarkers();                  // check to see if we have received any new data
        if (newData)  {    processCommand();  }     // if we have new data see if it is a command
    }
} // void loop()
 void sendInitCommands(){
    BTserial.print("<T,Single LED Control>");      // Control panel title
    BTserial.print("<I,SS,1,Red LED>");            // Initialize small switch 1. Switch description = Red LED
    BTserial.print("<R,5>");                       // Set the app update speed to 50ms
    BTserial.print("<EOI>");                       // End Of Initialization commands
    delay(100);                                    // Small delay to give the app time to set up.
     initialized = true;
    digitalWrite(13,HIGH);
     if (DEBUG) { Serial.println("Init commands sent"); }}
 void reset(){
    digitalWrite(LED_RED_PIN,LOW);}
/*****************************************
* Function processCommand
* parses data commands contained in receivedChars[]
* receivedChars[] has not been checked for errors
* * passed:
* * global:
*    receivedChars[]
*   newData
** Returns:
*
* Sets:
*          receivedChars[]
*          newData*
*/
void processCommand(){
    if (DEBUG)
    {
        Serial.print("receivedChars = ");
        Serial.println(receivedChars);
    }

    if (strcmp ("CONNECT",receivedChars) == 0)
    {
        sendInitCommands();
        if (DEBUG) { Serial.println("CONNECT message received"); }
    }

    else if (strcmp ("RESET",receivedChars) == 0)
    {
        reset();
        sendInitCommands();
        if (DEBUG) { Serial.println("RESET received"); }
    }
```

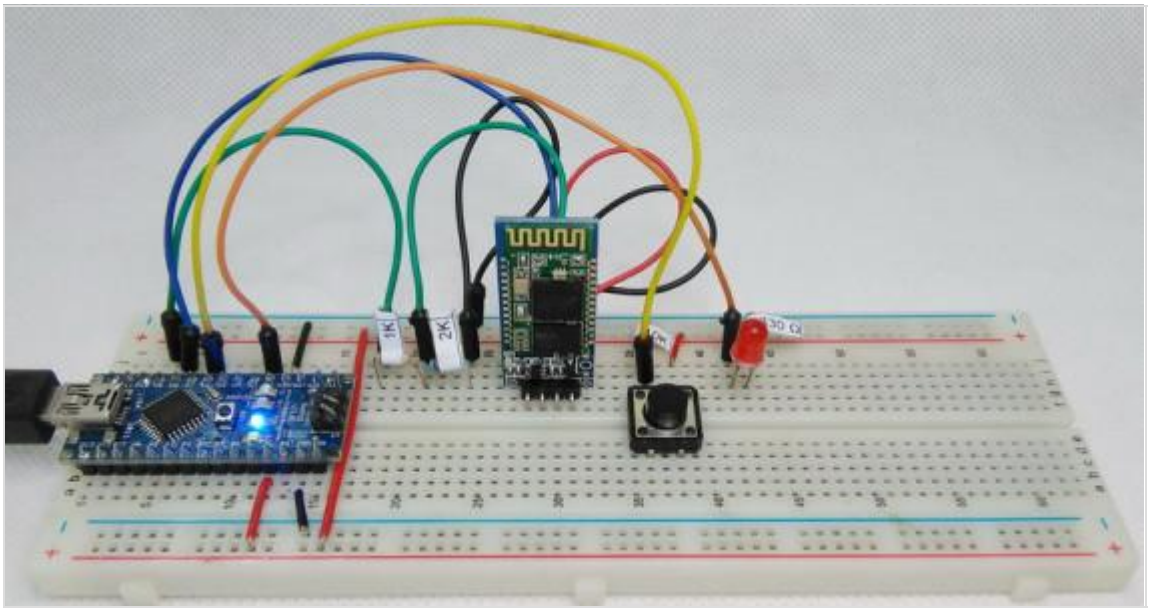
```
    else if (strcmp ("SS10",receivedChars) == 0)
    {
        digitalWrite(LED_RED_PIN,LOW);
        if (DEBUG) { Serial.println("LED LOW");   }
    }

    else if (strcmp ("SS11",receivedChars) == 0)
    {
        digitalWrite(LED_RED_PIN,HIGH);
        if (DEBUG) { Serial.println("LED HIGH");   }
    }

    receivedChars[0] = '\0';
    newData = false;}

 // function recvWithStartEndMarkers by Robin2 of the Arduino forums
// See   http://forum.arduino.cc/index.php?topic=288234.0
/*****************************************
* Function recvWithStartEndMarkers
* reads serial data and returns the content between a start marker and an end marker.
* * passed:
* * global:
*   receivedChars[]
*   newData
** Returns:
*
* Sets:
*         newData
*         receivedChars*
*/
void recvWithStartEndMarkers(){
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;
    if (BTserial.available() > 0)
    {
         rc = BTserial.read();
         if (recvInProgress == true)
         {
              if (rc != endMarker)
              {
                   receivedChars[ndx] = rc;
                   ndx++;
                   if (ndx > maxDataLength) { ndx = maxDataLength; }
              }
```

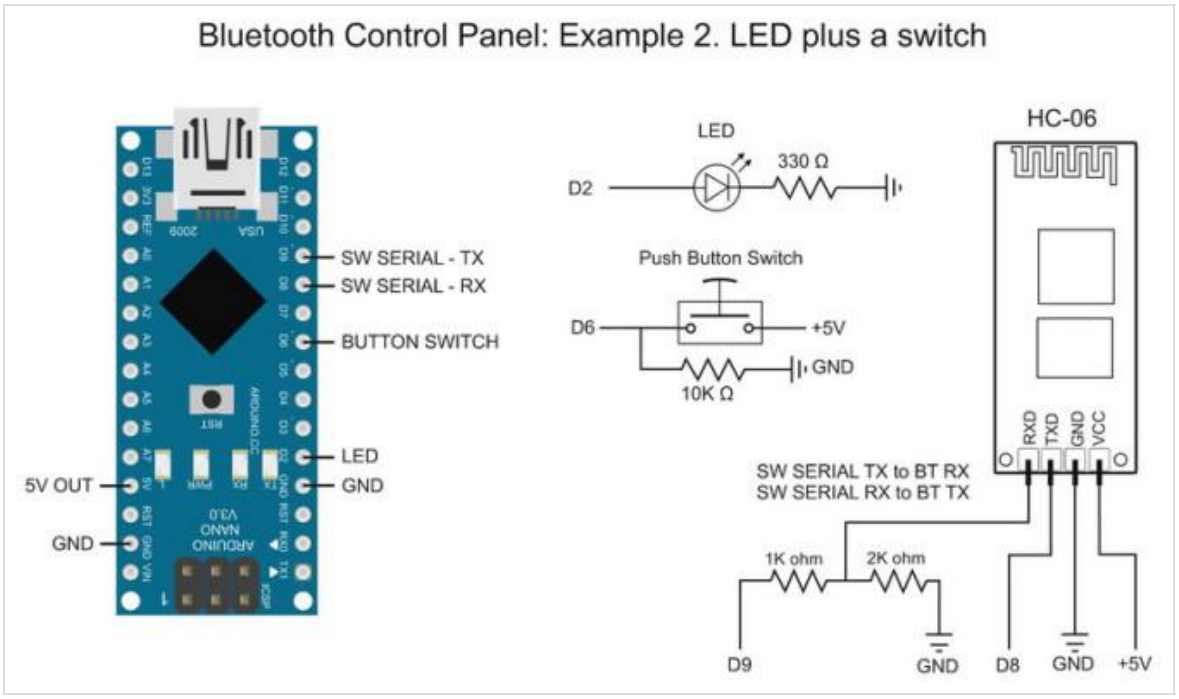
```
            else
            {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }
        else if (rc == startMarker) { recvInProgress = true; }
    }}
```

Esempio 2: controllare un LED usando un grande interruttore nell'applicazione e un interruttore fisico

Spedire i dati da Arduino all'appllicazione ed usare l'interruttore per controllare il LED .
Continuiamo ad usare un singolo LED ma ora stiamo aggiungendo anche un interruttore fisico all'interno di Arduino.
Quando l'interruttore è premuto i comandi sono inviati per settare l'interruttore nell'applicazione per trovare il nuovo stato del LED.
Abbiamo anche cambiato il piccolo interruttore interno all'applicazione in uno piu grande.

Il LED è ancora connesso al D2 e l'interruttore al D6





Interruttore fisico di cui controllare lo stato,nel loop principale useremo la funzione checkSwitch().

```
if (initialized)
{
    checkSwitch();
    recvWithStartEndMarkers();                    // check to see if we have received any new commands
    if (newData)  {    processCommand();  }      // if we have a new command do something about it
}
```

La funzione checkSwitch() legge lo stato dell'interruttore,elabora se l'interruttore è stato premuto e se lo stato del LED necessita di essere cambiato.
Teniamo conto dello stato del LED usando 2 variabili; switchState e LED_RED_State.

Ogniqualvolta l'interruttore viene premuto l'interruttore va da LOW a HIGH) lo stato del LED cambia sia da off a on che da on a off.Quando si usano variabili booleane, le istruzioni

```
LED_RED_State = ! LED_RED_State;
```

Inverte il valore; sia da LOW a HIGH che da HIGHa LOW e ha lo stesso utilizzo

```
if (LED_RED_State == HIGH) { LED_RED_State == LOW; }   else
{ LED_RED_State == HIGH; }
void checkSwitch(){
    // Simple toggle switch function with very simple debouce.
    boolean state1 = digitalRead(SWITCH_PIN);
    boolean state2 = digitalRead(SWITCH_PIN);
    boolean state3 = digitalRead(SWITCH_PIN);
    if ((state1 == state2) && (state1==state3))
    {
        switchState = state1;
        if ( (switchState == HIGH) && (oldSwitchState == LOW) )
        {
            LED_RED_State = ! LED_RED_State;
            if ( LED_RED_State == HIGH) {
                                        BTserial.print("<D,SW,1,1>" );
                                        digitalWrite(LED_RED_PIN,HIGH);
```

```
                                        if (DEBUG) { Serial.println("Sent - <D,SW,1,1>");    }
                                    }

                else                {
                                        BTserial.print("<D,SW,1,0>");
                                        digitalWrite(LED_RED_PIN,LOW);
                                        if (DEBUG) { Serial.println("Sent - <D,SW,1,0>");    }
                                    }
            }
            oldSwitchState = switchState;
        }}
```

In quanto abbiamo introdotto uno stato di inizializzazione possiamo ora usare il comando <RESET>.

Comunque, abbiamo bisogno di abilitare per l utente la possibilità di resettare l'applicazione quando il LED è on, in quanto quando lo sketch riceve il comando <RESET> richiama la funzione reset() la quale assicura di spegnre il LED.

```
else if (strcmp ("RESET",receivedChars) == 0) {
    if (DEBUG) { Serial.println("RESET message received"); }
    reset();
    sendInitCommands();}
void reset(){
    digitalWrite(LED_RED_PIN,LOW);}
```

In quanto abbiamo aggiunto il comando < RESEt> potremmo avere aggiunto il comando <DISCONNECT>.
Se l'utilizzatore chiude la connessione Bluetooth dall'applicazione questa spedisce un comando di <DISCONNECT>.
Ciò può essere usato per permettere agli sketch di Arduino di tenere traccia degli stati di connessione.
Per esempio, quando viene ricevuto un comando lo sketch semplicemente ritorna sullo stato non inizializzato

```
else if (strcmp ("DISCONNECT",receivedChars) == 0) {
     if (DEBUG) { Serial.println("DISCONNECT message received"); }
     reset();
     initialized = false;}
```


Esempio3: mostare una temperatura usando un controllo Value

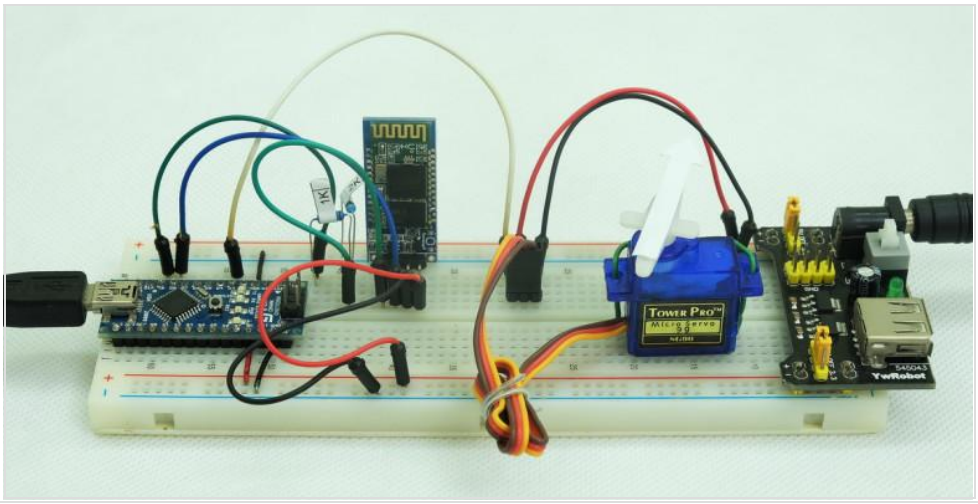
Esempio 4a: controllare un Servo

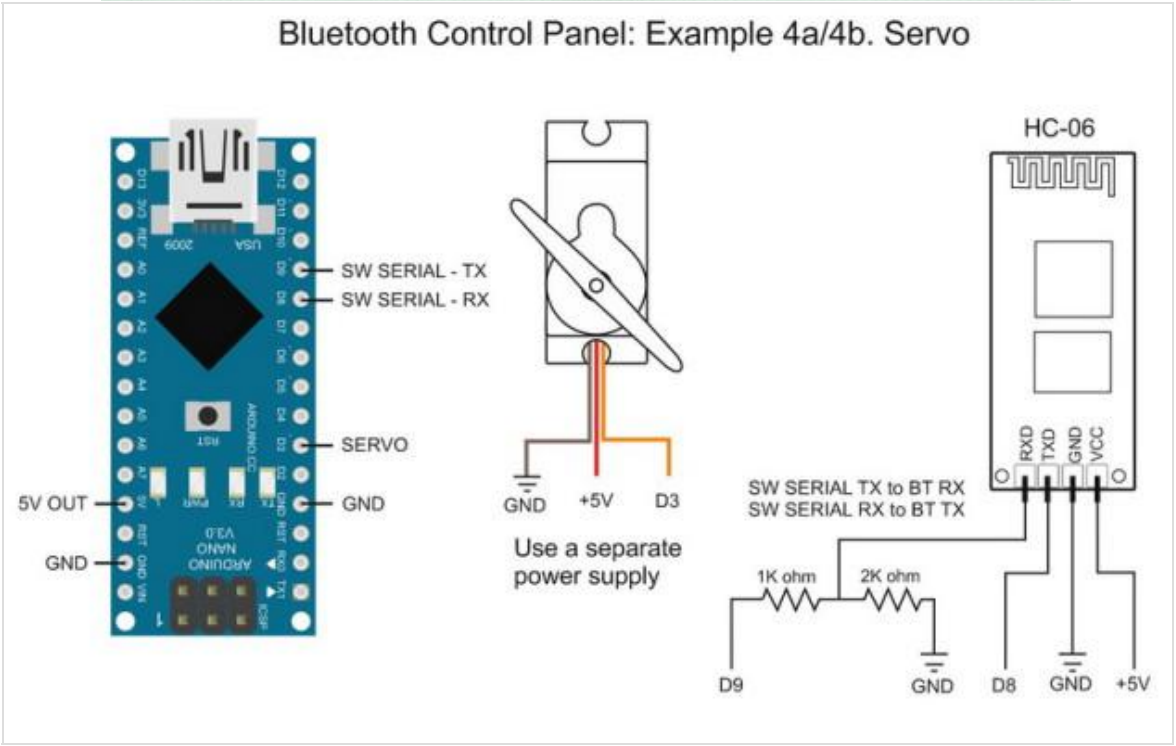
In questo esempio usiamo uno slider per controllare un signolo servo. Lo slider permette a noi di posizionare il servo in ogni angolo possibile tra 0 e 180 gradi.

Il servo è un piccolo Tower Pro 9g il quale è alimentato dall'alimentazione supplementare di una breadboard.i motori non devono essere alimentati direttamene da arduino Motors should not be powered directly from the Arduino. The servo has a range of about 180 degrees.

Nota: se non ben definito il servo e i jump possono causare conflitti fra timer di diverse librerie se si utilizza lo stesso timer. Per evitare questo, quando si utilizzano servi è sempre meglio usare hardware di serie per la comunicazione seriale.
Traduzione di Vincenzi Mattia 5^B 2016/2017 I.T.T. B.Pasqual Cesena

Circuiti





Il servo vcc è connesso alla propria alimentazione (in questo caso 5V dall'alimentazione della breadboard), il pin di terra è connesso all'alimentazione ed anche ad Arduino.
La linea di controllo servo è connessa al pin D3 di Arduino.
Dal momento che stiamo utilizzando AltSoftSerial per interagire col modulo bluetooth, che è connesso ai pin D8 e D9 che sono connessi ad Arduino.

Arduino sketch

Per pilotare servo, uso la libreria ServoTimer2 che utilizza il Timer2 di Arduino funziona con AltSoftSerial, ho cambiato i valori minimi e massimi standard di pulsazione ed i valori delle librerie standard non mi danno 180 gradi pieni.

Nel file ServoTimer2.h cerca le seguenti linee:

```
#define MIN_PULSE_WIDTH 750
#define MAX_PULSE_WIDTH 2250
```

e cambiale in:

```
#define MIN_PULSE_WIDTH 500
```

```
#define MAX_PULSE_WIDTH 2300
```

Dopo aver cambiato il file ServoTimer2.h, ti servirà ricaricare l'IDE di Arduino.

Questi sono gli attuali valori massimi e minimi per servo che sto usando, i tuoi potrebbero essere differenti.

Nella funzione setup:

– onboard LED è inizializzata come output

– il pin 3 è inizializzato come servo pin

– AltSoftSerial incomincia la comunicazione col modulo bluetooth.

```
void setup()  {
    // Setta il pin onboard LED come output
    pinMode(13, OUTPUT);
    digitalWrite(13,LOW);

    myservo.attach(3);
    myservo.write(servoMinPulse);

    //  apre la connessione seriale software col modulo Bluetooth.
    BTserial.begin(9600);

    newData = false;
    initialized = false;
    startTime = millis();
} // void setup()
```

Nel ciclo principale, mentre Arduino aspetta il messaggio CONNECT, onboard LED è attivo.

Dopo che viene effettuata una connessione il LED è acceso, Arduino poi controlla semplicemente i dati in ingresso e se dovesse ricevere un comando, lui chiama la funzione processCommand.

```
void loop()  {
    if (!initialized)
    {
        if (  millis()-startTime > 250 )
        {
            startTime = millis();
            if (LEDstate == HIGH) { LEDstate = LOW;  digitalWrite(13,LOW);  }
            else                  { LEDstate = HIGH; digitalWrite(13,HIGH); }
        }
         recvWithStartEndMarkers();
         if (newData)
         {
             if (strcmp ("CONNECT",receivedChars) == 0 or strcmp ("RESET",receivedChars) == 0)
             {
                 sendInitCommands();
             }
         }
     }
    if (initialized)
    {
        recvWithStartEndMarkers();                  // verifica se abbiamo ricevuto qualche nuovo comando
        if (newData)  {   processCommand();  }    // se abbiamo un nuovo comando, fai qualcosa a riguardo
    }}
```

C’è un comando TITLE ed uno SLIDER, che T è settato con un minimo di 0 e di un massimo di 180.

```
void sendInitCommands(){
      BTserial.print("<T,Servo Control>");
      BTserial.print("<I,SL,1,Servo Position,0,180,255250230>");
      BTserial.print("<EOI>");
      delay(100);
      initialized = true;
      digitalWrite(13,HIGH);       }
```

La funzione processCommand controlla se c’è il comando SLIDER e se ne trova uno converte da valore ascii in un valore numerico e lo copia nella variabile val.
La funzione poi controlla se il comando è, per SLIDER, il numero 1.
Se il valore di val è mappato nel valore di pulsazione di servo, esso e servo vengono spostati.

Notare che il comando map sembra un po’ strano, il valore alto (180) è prima del valore basso (0), inverte il valore 0 diventa 180 e 180 diventa 0;

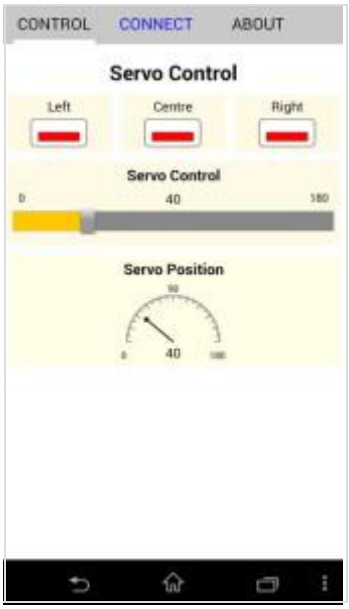
```
servoPulse = map(val,180,0,servoMinPulse,servoMaxPulse);
```

Il servo va da DX a SX e SLIDER va da SX a DX, senza invertire il valore, servo si muove in direzione opposta allo SLIDER.

```
    else if (receivedChars[0]== 'S' && receivedChars[1]== 'L')
    {
            unsigned int servoPulse = 0;
            int val =     (receivedChars[3]-48) *1000;
            val = val + (receivedChars[4]-48) * 100;
            val = val + (receivedChars[5]-48) *    10;
            val = val + (receivedChars[6]-48);

            if ( receivedChars[2]== '1' )
            {
                // converti il valore di slider 0-180 al range di pulsazione di servo di servoMinPulse-servoMaxPulse
                // servo è da destra a sinistra, slider è da sinistra a destra, quindi usa map per capovolgere il valore intorno
                servoPulse = map(val,180,0,servoMinPulse,servoMaxPulse);
                myservo.write(servoPulse);
            }
    }
```


Esempio 4b: Controlla Servo advanced

Aggiungeremo alcuni controlli avanzati sotto forma di 3 piccoli interruttori usati per accendere servo direttamente a lontano a sinistra, centro ed a destra lontano.
Il bit viene anticipato quando inviamo di nuovo i comandi all’app i piccoli interruttori ed aggiorna la posizione del cursore.

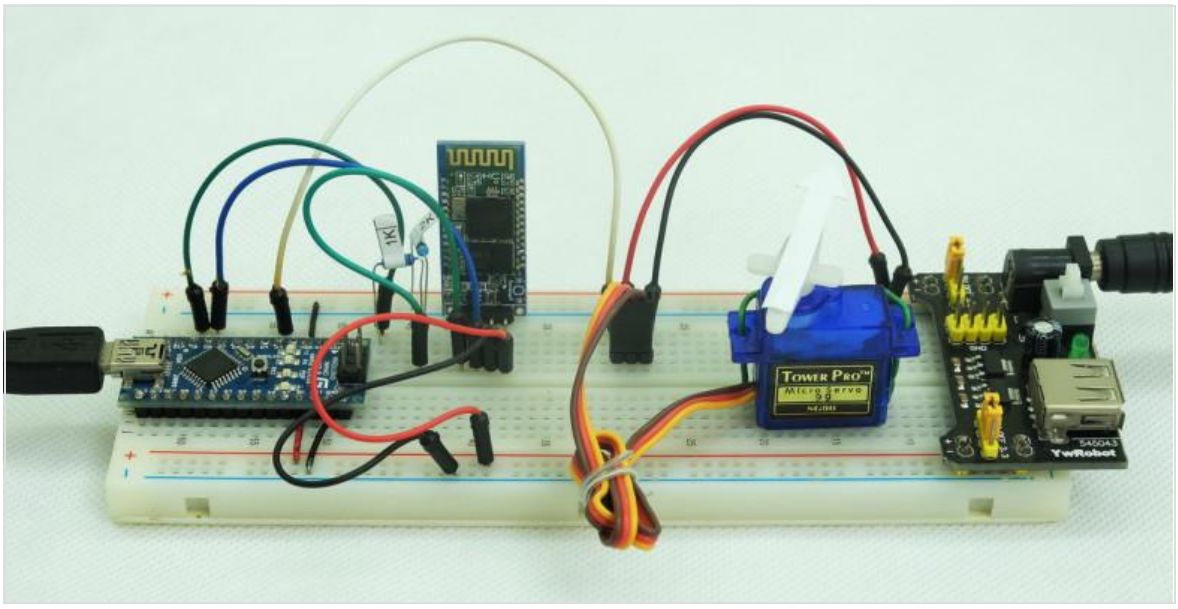


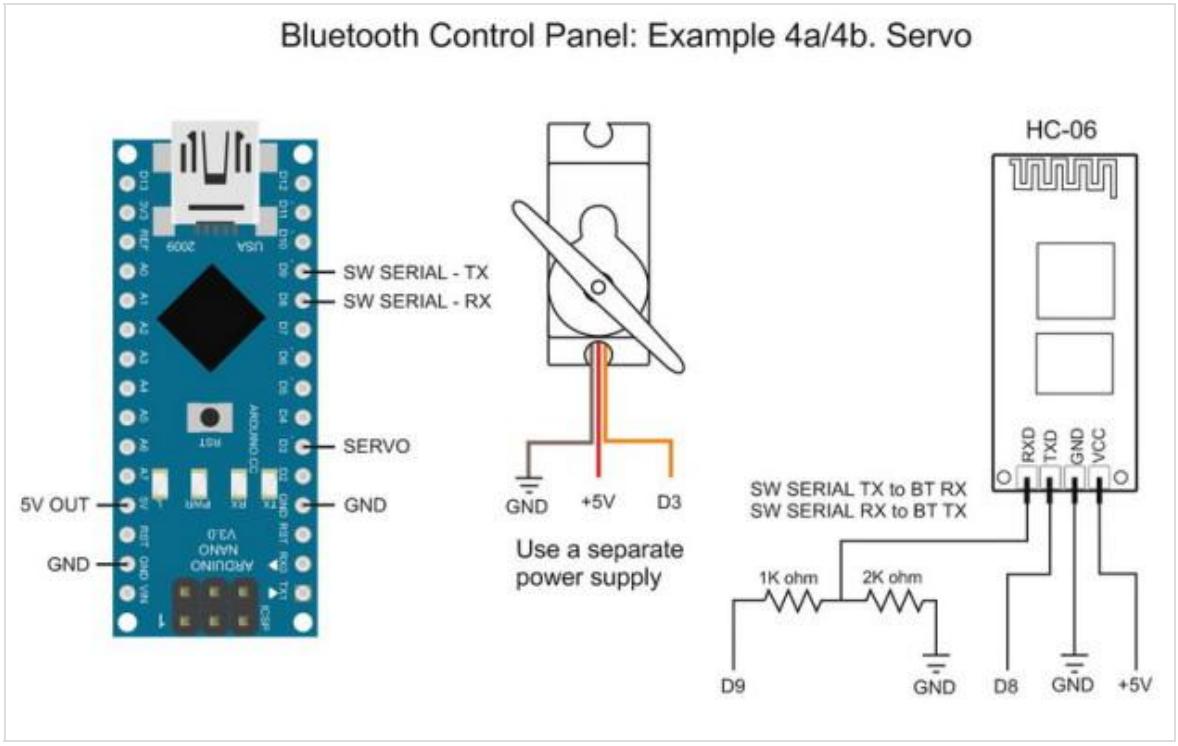
L’APP non aggiorna la posizione del cursore quando uno degli interruttori è stato premuto.
Per ovviare a questo, inviamo un comando di dati SLIDER indietro alla app con la nuova posizione.

Gli interruttori sono piccoli e basculanti; quando vengono clickati vanno dal verde al rosso al verde etc.
Significa che normalmente bisogna riclickarli per di farli tornare allo stato off.
Un modo migliore è quello di inviare un comando dati al piccolo switch dal microprocessore alla app.
Per una buona misura aggiungeremo un quadrante per mostrare il braccio direzione/angolazione di servo.

Circuito

Il circuito è lo stesso dell'esempio precedente, il modulo bluetooth è connesso ai pin D8 e D9 mentre Servo è connesso a D3.





Arduino

La struttura principale dello sketch è la stessa di quella in esempio in 4a, è stata aggiunta una inizializzazione extra di comandi per attivare elementi extra di comando e ci sono controlli extra nella funzione processCommand per i piccoli switch.

```
void sendInitCommands(){
    BTserial.print("<T,Servo Control>");                  // Titolo
    BTserial.print("<I,SS,1,Left,255250230>");           // Attiva un piccolo switch
    BTserial.print("<I,SS,2,Centre,255250230>");         // Attiva un piccolo switch
    BTserial.print("<I,SS,3,Right,255250230>");          // Attiva un piccolo switch
```

```
    BTserial.print("<I,SL,1,Servo Control,0,180,255250230>");   // SLIDER
    BTserial.print("<I,DI,1,Servo Position,0,180,255255230>"); // DIAL
    BTserial.print("<R,3>");                                    // Impostare la frequenza di aggiornamento app per 30ms
    BTserial.print("<EOI>");
    delay(100);
    initialized = true;
    digitalWrite(13,HIGH);
    if (DEBUG) { Serial.println("Init commands sent"); }}
```

Nella funzione processCommand c'è un comando in più che invia la posizione del servo di nuovo al quadrante in app.

```
BTserial.print("<D,DI,1,"); BTserial.print(val);   BTserial.print(">");
    else if (receivedChars[0]== 'S' && receivedChars[1]== 'L')
    {
            unsigned int servoPulse = 0;
            int val =     (receivedChars[3]-48) *1000;
            val = val + (receivedChars[4]-48) * 100;
            val = val + (receivedChars[5]-48) *   10;
            val = val + (receivedChars[6]-48);

            //val = 180-val;

            if ( receivedChars[2]== '1' )
            {
                // convertie il range di slider 0-180 al range di pulsazione di 500-2400 di servo pulse range of 500-2400
                servoPulse = map(val,180,0,servoMinPulse,servoMaxPulse);

                if (DEBUG) { Serial.print("val = "); Serial.print(val);Serial.print("      servoPulse = "); Serial.println(servoPulse); }
                myservo.write(servoPulse);
                BTserial.print("<D,DI,1,"); BTserial.print(val);   BTserial.print(">");
             }
    }
```

Ci sono condizioni extra per prendersi cura dei comandi dello switch piccolo: in primo luogo verifica la presenza di un comando per un piccolo switch (“SS”) poi controlla per vedere quale switch è (1,2,o 3) a seconda del commutatore val è impostato sulla posizione desiderata ed un comando dati del piccolo switchviene inviato all’applicazione per riportare lo switch indietro allo stato off.
Infine i controlli dati vengono usati per inviare la nuova posizione servo indietro allo SLIDER ed al DIAL.

```
    else if (receivedChars[0]== 'S' && receivedChars[1]== 'S')
    {
        int val = 0;
        unsigned int servoPulse = 0;
        if (receivedChars[2]== '1' && receivedChars[3]== '1')      // Sinistra
        {
            val = 180;
            BTserial.print("<D,SS,1,0>");
        }
        else if (receivedChars[2]== '2' && receivedChars[3]== '1')      // Centro
        {
            val = 90;
            BTserial.print("<D,SS,2,0>");
        }
```

```
    else if (receivedChars[2]== '3' && receivedChars[3]== '1')      // Destra
    {
        val = 0;
        BTserial.print("<D,SS,3,0>");
    }
    servoPulse = map(val,0,180,servoMinPulse,servoMaxPulse);
    myservo.write(servoPulse);

    BTserial.print("<D,DI,1,"); BTserial.print(180-val);   BTserial.print(">");
    BTserial.print("<D,SL,1,"); BTserial.print(180-val);   BTserial.print(">");
    if (DEBUG) { Serial.print("val = "); Serial.print(180-val);Serial.print("      servoPulse = "); Serial.println(servoPulse); }
}
```


Traduzione di Lucchi Nicola 5^B 2016/2017 I.T.T. B.Pasqual Cesena