# UNIVERSITA' DEGLI STUDI DI TOR VERGATA

UNIVERSITA' degli STUDI di ROMA
TOR VERGATA

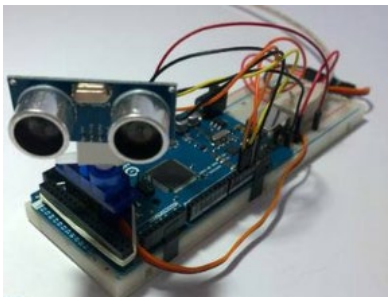
**Facoltà di Ingegneria**

Corso di Laurea magistrale in Ingegneria Medica

***Corso di Laboratorio di Elettronica 2***

## *Progetto laboratorio elettronica*

## *Rivelatore ad ostacoli con grafica Processing e App Android*



*A.A.2019/2020*

*Progetto elaborato e sviluppato da:*

*Simonelli Elisa, Spiridigliozzi Giulia, Trovato Andrea.*

# Introduzione e componenti utilizzati

In questo progetto è stato realizzato un rilevatore ad ostacoli.

Per fare ciò è stato sfruttato un sensore ultrasuoni HC-SR04, che manda un segnale all'Arduino quando rileva un ostacolo, mentre viene fatto ruotare un servomotore (tra 0° e 180°). Il tutto è stato montato su un prototype expansion module, alimentato con una batteria 9V e connesso alla seriale tramite il modulo Bluetooth HC-05.

Arduino è la più famosa piattaforma open source di prototizzazione hardware e software nel campo dei microcontrollori. La scheda Arduino dispone al suo interno di varie strutture hardware predisposte alla comunicazione, all'adattamento e al controllo delle tensioni; inoltre, dispone di vari pin per la tensione, ingressi analogici, pin digitali di I/O e PWM. Dispone di una presa femmina USB per la comunicazione e una presa femmina per il jack di ingresso dalla tensione.

Il sensore HC SR 04 è composto da 4 terminali ($V_{CC} - Trig - Echo - GND$) e da due altoparlanti, di cui uno funge da trasmettitore e l'altro da ricevitore. Il sensore, in presenza di un ostacolo, è in grado di misurare la distanza nel seguente modo: quando viene attivato il pin trigger, immediatamente l'altoparlante trasmettitore emette in linea retta un segnale ultrasuoni, che fisicamente colpisce l'ostacolo e viene riflesso in senso opposto come eco; intanto l'altoparlante ricevitore viene attivato in ascolto dal pin Echo in modo che sia pronto a ricevere il suono appena emesso: a questo punto, siccome la velocità del suono nell'aria è costante, in assenza di interferenze, Arduino può calcolare la distanza tra il sensore e l'ostacolo in base al tempo che il suono impiega a ritornare alla sorgente.

Il servomotore è un dispositivo capace di eseguire dei movimenti meccanici in funzione del segnale applicato al suo ingresso. Il servomotore è composto da un motore elettrico, un motoriduttore ed un circuito di feedback per la gestione della posizione. Il tipo di servo utilizzato ha un angolo di rotazione di cica 180° (la rotazione effettiva è un po' inferiore). Il servo presenta 3 contatti: due di alimentazione $(V_{CC} - GND)$ e il terzo è il pin di controllo collegato ad Arduino. La complessità del servo nasce dalla necessità di utilizzare un duty-cycle per impartire i comandi in grado di trasformare la volontà elettronica in movimento. Arduino dispone di uscite PWM, cioè pin digitali in grado di generare un duty-cycle utilizzabile per far muovere il servo motore.

Per la connessione seriale si è utilizzato un modulo Bluetooth HC-05 a 6 pin $(KEY, RX, TX, V_{CC}, GND, STATE)$. Il pin KEY serve a settare il modulo per la comunicazione tramite comandi AT; RX e TX sono i pin deputati allo scambio dei dati; $V_{CC}$ e GND sono i pin di alimentazione del modulo HC-05; STATE è il pin output che passa dallo stato LOW allo stato HIGH quando il modulo è connesso. Il modulo HC-05 ha un range di 10 metri e frequenza di 2.4Hz. Il modulo permette di trasformare una porta UART o USART, cioè la porta seriale, in una porta Bluetooth, generalmente con profilo SPP (Serial Port Profile), diventando così una seriale over Bluetooth. Le funzioni base di trasmissione seriale sui pin TX (1) e RX (0) sono riconosciute direttamente dal software Arduino. È necessario un partitore resistivo prima della RXD per ridurre a 3,3V i livelli a 5V della TX, mentre è facoltativo nell'altra direzione perché la linea RX di Arduino riconosce ugualmente i livelli della TXD di HC-05 a 3,3V.

I vari componenti (servomotore, sensore ultrasuoni e modulo Bluetooth) sono stati montati su un prototype expansion module che ha come caratteristiche quella di poter istallare senza saldature una mini breadbord sopra. Inoltre, presenta ulteriori pin di voltaggio e di messa a terra (3 pin 5V, 3 pin 3,3V e 3 pin GND). È stato scelto di usare questo modulo per rendere il tutto più dinamico e maneggevole.

## Realizzazione e codice Arduino

Come abbiamo sviluppato la parte hardware?

La parte hardware del progetto è stato sviluppato nel seguente modo: come prima cosa abbiamo montato il modulo di espansione sopra la scheda Arduino, inserito la piccola breadbord e si è collegato i pin del sensore ad ultrasuoni: $V_{CC}$ al pin 5V del prototype expansion module; GND al pin GND del modulo, il pin trigger al pin digitale 8; il pin Echo al pin digitale $\sim 9$. In seguito, abbiamo montato il servomotore $V_{CC}$ al pin 5V del modulo; GND al pin GND del modulo; il pin del servomotore al pin digitale $\sim 10$ (poiché su questo pin c'è un timer con una frequenza precisa. Inoltre, abbiamo collegato il modulo Bluetooth collegando: $V_{CC}$ al pin 5V del modulo, $RX$ collegata con un partitore resistivo e collegato al pin 0, $TX$ al pin 1, GND al pin di GND.  Infine, abbiamo alimentato la scheda Arduino con una batteria 9V.

Come abbiamo sviluppato la parte software?

Per la parte software il codice di questo progetto è relativamente semplice. Abbiamo utilizzato la libreria standard per il servomotore "Servo.h", la libreria per il sensore ultrasuoni "NewPing.h".

```
#include <NewPing.h>
#include <Servo.h>
#define PIN_TRIG 8
#define PIN_ECHO 9
#define MAX_DIST 200
#define PIN_SERVO 10
NewPing rilevatore(PIN_TRIG,PIN_ECHO,MAX_DIST);
int gradi;
Servo myServo;
void setup(){
  Serial.begin(9600);
  myServo.attach(PIN_SERVO);
}
void loop(){
  unsigned int cm= rilevatore.ping_cm();
  Serial.write(cm);
  delay(100);
  servo();
}
  void servo() {
    if(Serial.available()){
    gradi=Serial.read() ;
    myServo.write(gradi);
  }
}
```

La libreria New Ping è stata inserita poiché consente ad Arduino di verificare se il suono che ritorna come eco corrisponde a quello effettivamente emesso, per evitare che delle interferenze possano falsare le letture. Implementa una variabile di max distance che definisce la massima distanza a cui si vuole effettuare la misurazione. Implementa una funzione che restituisce la misurazione direttamente in centimetri. Implementa una funzione che effettua più letture del sensore e ne restituisce una media, garantendo una maggiore affidabilità e precisione delle misurazioni. Consente di unire il trigger e l'Echo sullo stesso pin. Per l'utilizzo di un sensore ad ultrasuoni, la libreria New Ping richiede che venga dichiarato: i pin di trigger ed Echo, in questo ordine, il valore della distanza massima. Le funzioni di lettura utilizzate nel nostro caso sono state: ".ping_cm()" che manda un ping e determina la distanza in cm (numero intero). Obbligatoriamente è stato inserito un ritardo per evitare che il sensore si "confonda" tra gli ultrasuoni emessi e quelli ricevuti.

La libreria Servo inserita poiché traduce angoli in segnali ed evita di dovere impazzire con i duty-cycle e i calcoli dei tempi. Contiene una serie di funzioni che permettono di controllare la posizione dell'albero dei servo, utilizzando direttamente l'angolo di rotazione. Viene generato il segnale PWM, con la frequenza e il duty-cycle necessari per posizionare l'asse di uscita del servomotore. In generale, il segnale PWM ha due caratteristiche: l'ampiezza massima e il duty-cycle. La prima dipende dal segnale, mentre il secondo parametro descrive la durata dell'impulso a livello alto, che viene misurato in percentuale ed è dato dal periodo del segnale a livello alto diviso il periodo del segnale modificato; la tecnica PWM ha la possibilità di variare il duty-cycle, ovvero l'intervallo di tempo in cui si ha lo stato alto rispetto al periodo totale. È possibile ottenere valori intermedi in una scala da 0 (0V) a 255 (5V); il valore della scala è dato dal valore predefinito dei pin PWM, ovvero 8 bit ($2^8 = 256$). Le istruzioni di questa libreria che sono state usate nel nostro codice sono state: Servo myservo che crea l'oggetto di tipo Servo, myservo è l'oggetto su cui si opera. "myServo.attach(pin-servo)" lega l'oggetto myservo al pin a cui abbiamo collegato il servo, nel nostro caso il pin 10, definito in precedenza. Con la riga di codice "myServo.write(gradi)" si passa dall'oggetto myservo ai gradi che deve raggiungere spostandosi gradualmente da 0° a 180°.

<u>Come funziona il codice?</u>

La struttura del codice si sviluppa sulla definizione di tre funzioni: "void setup()" e "void loop()" e "void servo". La prima funzione inizializza le variabili, imposta lo stato dei pin, include le librerie e imposta la velocità di comunicazione seriale, nel nostro caso la velocità impostata è stata di 9600 bps (o baud). La funzione "void loop ()" esegue ciclicamente il programma definito al suo interno, nel nostro caso l'istruzione che esegue ciclicamente il programma è che il sensore manda un ping per determinare la distanza in numero interno, lo scrive sulla seriale, utilizza un ritardo tra un ciclo e l'altro. La funzione "void servo ()" dice che se arriva qualcosa nel buffer, deve leggerlo e memorizzarlo nella variabile di tipo intero chiamata "gradi" e il servo imposta l'angolo dell'albero, spostando l'albero nell'orientamento (gradi).

# Sviluppo Interfaccia Grafica con Processing

Per sviluppare l'interfaccia grafica su PC abbiamo utilizzato processing; questo linguaggio di programmazione eredita tutta la sintassi, i comandi e il paradigma di programmazione orientata agli oggetti dal linguaggio Java ma in più mette a disposizione numerose funzioni ad alto livello per gestire in modo facile gli aspetti grafici e multimediali.

Il nostro progetto prevede l'utilizzo di un rilevatore di ostacoli (sensore ad ultrasuoni HC-SR04) sopra un servomotore (SG90) che fa muovere il primo con angoli che variano da 0 a 180 gradi. L'idea di base è stata quella di creare un qualcosa, che rilevi i dati provenienti da Arduino (ovvero le varie distanze calcolate dal sensore ad ultrasuoni) e comandi il servomotore per ampliare il suo raggio di rilevamento ostacoli.

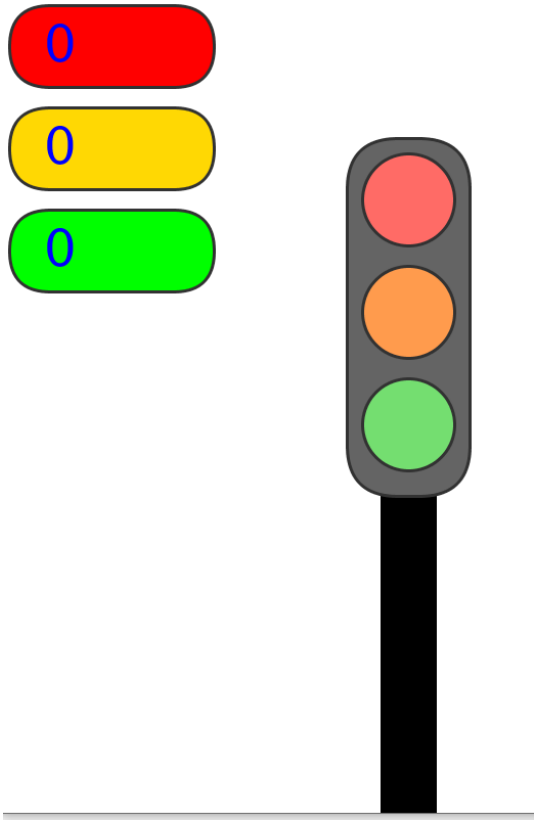
Per la parte dell'applicazione grafica abbiamo pensato di creare un semaforo e delle zone specifiche (dei rettangoli smussati) dove stampare la distanza proveniente da Arduino.

Come funziona il semaforo?

Nel "setup" è stata creata la grafica di partenza che ci raffigura il semaforo spento e i rettangoli senza alcun rilevamento al loro interno. È importante risaltare la simulazione del semaforo spento; i cerchi (funzione "ellipse"), che lo compongono, sono stati riempiti (funzione "fill") con dei colori (ROSSO, ARANCIONE E VERDE) fiochi che restituiscono un effetto realistico.

```
fill(0);
rect(375,400,50,400);
stroke(50);
strokeWeight(3);
fill(100);
rect(340,140,120,350,50);
fill(116,222,112);
ellipse(400,420,90,90);
fill(255,155,77);
ellipse(400,310,90,90);
fill(255,107,102);
ellipse(400,200,90,90);
fill(255,0,0);
rect(10,10,200,80,50);
fill(0,0,255);
textSize(50);
text("0",60,65);
pushMatrix();
translate(0,100);
fill(255,216,3);
rect(10,10,200,80,50);
fill(0,0,255);
textSize(50);
text("0",60,65);
translate(0,100);
fill(0,255,0);
rect(10,10,200,80,50);
fill(0,0,255);
textSize(50);
text("0",60,65);
popMatrix();
```

Ora analizziamo il semaforo all'interno del "draw".




Qui, ovviamente, il discorso è differente: a seconda della distanza inviata da Arduino il semaforo deve accendersi con il colore giusto. Questo effetto è stato simulato riempiendo i cerchi con colori più vivaci. Prendiamo per esempio il caso in cui la distanza dell'oggetto è inferiore a "20 cm" e maggiore di "0 cm"; si colorerà con un rosso più vivace ("fill (255,0,0)") il cerchio più in alto mentre gli altri rimarranno sui loro colori più fiochi (ovvero il colore arancione e il colore verde). Questa accensione ci allerta della vicinanza dell'ostacolo, inoltre, è presente, in alto a sinistra, un rettangolo smussato rosso che indica l'esatta distanza.

```
if((distanza>0 && distanza<20) ){
stroke(50);
strokeWeight(3);
fill(116,222,112);
ellipse(400,420,100,100);
fill(255,155,77);
ellipse(400,310,100,100);
fill(255,0,0);
ellipse(400,200,100,100);
fill(255,0,0);
rect( 10,10,200,80,50);
fill(0,0,255);
textSize(50);
text(distanza,60,65);
```

(IMMAGINE DURANTE IL FUNZIONAMENTO)

Con la stessa logica sono stati rappresentati i casi in cui il semaforo si accenderà di verde e arancione stampando la distanza, rispettivamente nel rettangolo verde e in quello

arancione. È stata, di volta in volta, inserita anche una porzione di codice che cancella le letture precedenti mentre c'è quella in corso. Per esempio, quando il rettangolo rosso indica la distanza, gli altri due vengono ricolorati ad ogni ciclo in modo da cancellare eventuali scritture che ci sono state in precedenza su di essi. Qui di seguito è indicata la cancellatura nei rettangoli arancione e verde:

```
fill(255,216,3);                                    //cancellare gli altri rilevamenti
rect( 10,110,200,80,50);
fill(0,255,0);
rect( 10,210,200,80,50);
```

Interfaccia grafica legata al comando del servomotore:

L'idea è stata di creare uno slider in modo tale da spostare, con un angolo a nostro piacimento, l'ultrasuoni quando quest'ultimo, per esempio, avverte attraverso il semaforo la presenza di un oggetto nelle immediate vicinanze.

Nel "setup" è stata messa la parte di codice che rappresenta lo slider all'apertura dell'applicazione.

```
noStroke();
fill(255);
rect(560,460,600,300);
strokeWeight(5);
stroke(0);
line(580,540,1080,540);
fill(255,0,0);
ellipse(580,540,10,10);
fill(0);
textAlign(CENTER);
text(gradi+"°",825,665);
```

0°

Il "draw" abbiamo dovuto renderlo dinamico: il cursore, rappresentato con una sfera, si muove seguendo la freccia del mouse quando questa si trova entro certi limiti X e Y; inoltre il tasto sinistro del mouse deve anche essere premuto.

```
if(mousePressed && mouseX<=1080 && mouseX>=580 && mouseY<=560  && mouseY>=520)
```

La grafica è stata resa accattivante dal punto di vista visivo: la sfera aumenta o diminuisce di raggio a seconda dello spostamento dello slider in avanti o all'indietro. Inoltre, cambia di colore di volta in volta (è stata utilizzata la funzione "random" per il colore della sfera). Sotto

lo slider sono espressi i gradi ottenuti con il suo spostamento: ricavati grazie alla funzione "map". Qui di seguito il codice:

```
raggio=mouseX/20;
if(mousePressed && mouseX<=1080 && mouseX>=580 && mouseY<=560  && mouseY>=520){
  gradi=int(map(mouseX,580,1080,0,180));
  noStroke();
  fill(255);
  rect(560,460,600,300);
  strokeWeight(5);
  stroke(0);
  line(580,540,1080,540);
  fill(random(255),random(255),random(255));
  ellipse(mouseX,540,raggio,raggio);
  fill(0);
  textAlign(CENTER);
  text(gradi+"°",825,665);
```

(IMMAGINE SLIDER IN MOVIMENTO)

Comunicazione Arduino Processing

Finora, abbiamo presentato la parte grafica dell'applicazione, senza fare accenni sul funzionamento di essa. Il funzionamento avviene grazie alla comunicazione che si instaura tra processing e Arduino: possibile grazie all'utilizzo di una libreria che in Arduino è di default e che invece qui deve essere inclusa all'inizio del codice; qui di seguito è riportata l'importazione della libreria e la dichiarazione di una variabile globale il cui nome è "ultrasuoni_servo", il cui data-type non sarà uno dei classici incontrati finora (integer, float, stringa, ecc…) ma sarà un oggetto *Serial.*

```
import processing.serial.*;  // richiama la libreria della seriale (di default su Arduino IDE)
Serial ultrasuoni_servo;     // creo una seriale per la comunicazione con il sensore ad ultra suoni.
```

All'interno del blocco di setup *()* inizializziamo la nostra variabile "ultrasuoni_servo" creando una nuova istanza dell'oggetto. In pratica, con la sintassi "ultrasuoni-servo = newSerial()" si richiama il "constructor" presente all'interno della classe.  La classe ha dei dati al suo interno che scriviamo sotto forma di variabili; tali dati sono solo dichiarati ma non inizializzati. Per questo ci serve il constructor, che è quella funzione speciale che serve per creare l'oggetto vero e proprio. Per funzionare il constructor deve avere lo stesso nome della classe.

Il constructor che richiamiamo genera oggetti con caratteristiche differenti a seconda dei parametri che si inserisce per personalizzarlo; tra questi diamo particolare importanza al secondo e al terzo, che rappresentano rispettivamente la porta attraverso la quale vogliamo comunicare e la velocità di trasmissione dei dati.

```
ultrasuoni_servo = new Serial(this,"COM4",9600);
```

Infine, nel "draw" vengono richiamate delle funzioni(metodi) di scrittura e lettura dati in modo che Arduino e processing possano comunicare. Questi sono definiti come segue:

```
ultrasuoni_servo.write(gradi);
delay(150);

distanza = ultrasuoni_servo.read();
```

Da notare, per quanto riguarda la fase di scrittura di dati in seriale (per muovere il servomotore), l'utilizzo di un "delay" per un migliore funzionamento dell'applicazione: infatti, senza di esso, si sarebbero creati dei problemi sul servo che non lo avrebbero fatto funzionare. Con la sua introduzione il servo non si muove in maniera fluida come ci si aspetterebbe, ma evita problemi ancora più gravi come il non funzionamento dell'intera applicazione.

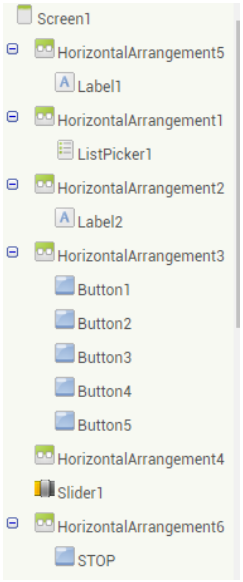
Qui di seguito riportiamo anche il codice completo e un piccolo video di dimostrazione sul funzionamento di questa applicazione:

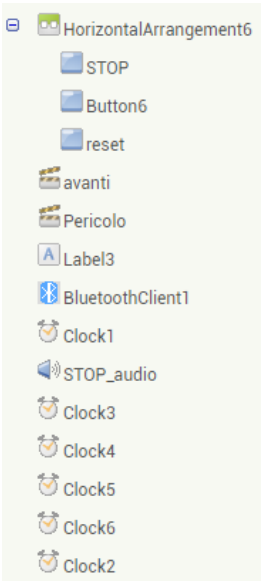
# Sviluppo App Android su MIT App Inventor

Per sviluppare l'applicazione Android abbiamo utilizzato l'ambiente App MIT Inventor di proprietà del Massachusetts Institute of Technology. L'idea è stata quella di inserire un semaforo con l'attivazione di "GiF" a seconda della presenza o meno di un ostacolo, inoltre, abbiamo inserito uno slider e dei bottoni con gli angoli più noti per far muovere il servomotore.

## Quali sono i componenti dell'app?

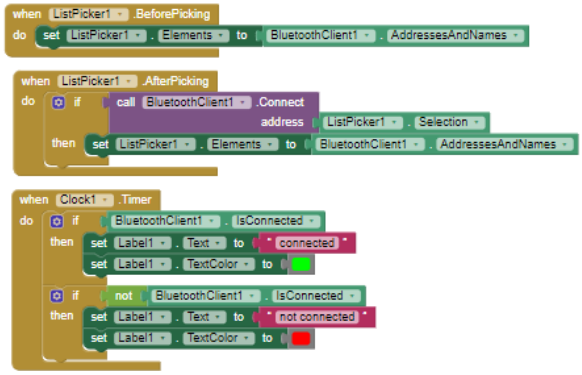
Iniziamo ad analizzare le componenti dell'app utilizzate per la parte grafica dell'app: il bottone per il collegamento Bluetooth, un "label" per informare della connessione BT, i bottoni per gli angoli noti e lo slider che con la pressione del dito possiamo indicare al servomotore a quale angolo spostarsi e al di sotto un ulteriore "label" che indica i gradi dello slider. Il semaforo rosso e verde affiancato del tasto di Reset. Le animazioni collegate con il semaforo e le scritte "nessun ostacolo" o "presenza ostacolo".



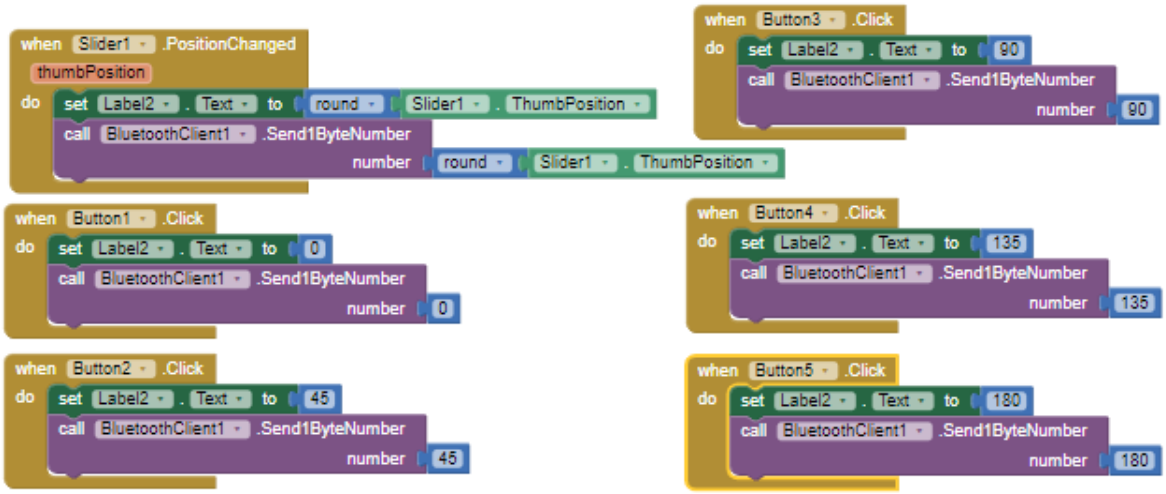




## Come sono strutturati i blocchi?

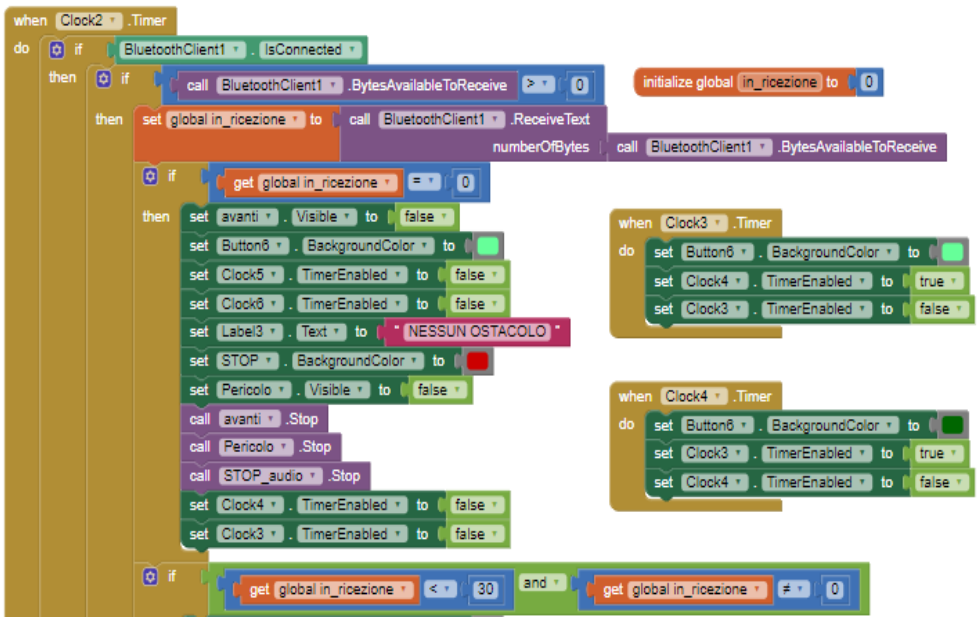
L'app è strutturata in vari blocchi che costituiscono i comportamenti (behaviors), che permettono di definire come la nostra app deve comportarsi in base alle nostre specifiche condizioni. Questi comportamenti li abbiamo strutturati all'interno della schermata blocks.

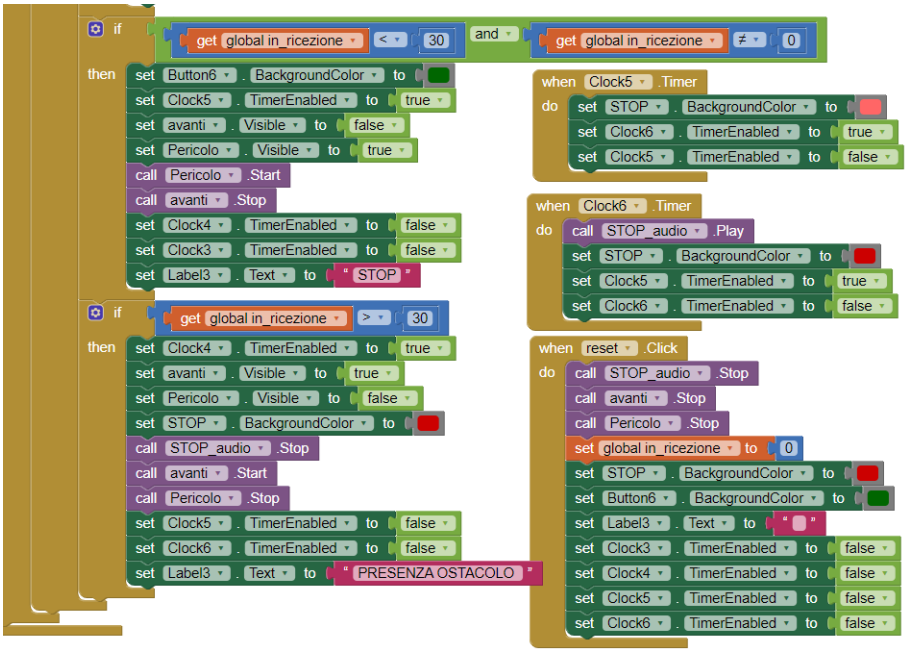


I primi due blocchi sono necessari per avviare una connessione Bluetooth, avendo preventivamente accoppiato l'interfaccia Bluetooth dell'applicazione con HC-05. Se la connessione si realizza, comparirà la scritta "connected", altrimenti la scritta "not connected" (terzo blocco). Per aprire la lista delle linee Bluetooth disponibili, basta cliccare sull'immagine:




Tutti i blocchi appena rappresentati, trasmettono dei dati ad Arduino e servono a scegliere la posizione del servomotore, facendolo ruotare tra 0° e 180°. Lo slider permette di scegliere un qualsiasi angolo verso cui far ruotare il servo motore, dove con "round" si prende il valore arrotondato al numero intero più vicino, così da inviare dei valori interi dell'angolo verso il quale deve ruotare; tale valore viene riportato come testo della label 2. I 5 bottoni presenti inviano, invece, dei valori specifici di angoli: 0°,45°,90°,135° e 180°.






Nei blocchi riportati sopra in foto, invece, l'app viene implementata con un'ulteriore caratteristica che è quella di ricevere dati, memorizzarli all'interno di una variabile e confrontare il dato ricevuto con delle soglie di valori, le quali costituiscono un parametro rispetto a cui si innescano determinate azioni da parte dell'app.

Se dal sensore ultrasuoni viene inviata una distanza pari allo zero, allora non ci sono ostacoli nelle vicinanze, sull'applicazione verrà riportata la dicitura "NESSUN OSTACOLO" e il led verde cambierà il suo colore, illuminandosi (primo ciclo if). Se, invece, dal sensore ultrasuoni arriva un valore della distanza minore di 30 (che rappresenta il valore di soglia), cioè se l'ostacolo viene percepito nelle immediate vicinanze, allora l'applicazione innesca il funzionamento del clock 5, il quale, a sua volta, innesca il funzionamento del clock 6, portando il led rosso a lampeggiare, modificando la scritta a "STOP" (secondo ciclo if), avviando una breve interfaccia grafica animata (GIF) e innescando inoltre un allarme audio, che verrà disattivato non appena l'ostacolo non è più nelle vicinanze. Il terzo ciclo if si innesca quando viene verificata la condizione che l'ostacolo si trovi in un range superiore a 30, ma minore di 200 (distanza massima inizializzata nel codice caricato sull'IDE di Arduino Uno). In tal caso si avvia il funzionamento del clock 4 che avvia a sua volta il clock 5, così da far lampeggiare il led rosso; viene modificato il testo con "PRESENZA OSTACOLO" e viene introdotta un'altra breve interfaccia grafica animata (GIF).

Infine, in caso si voglia stoppare l'azione prima della conclusione del suo normale decorso, con il tasto di reset si può inviare un segnale di arresto di qualsiasi azione l'app stia compiendo in quel momento.

Comunicazione Arduino App:

Abbiamo appena visualizzato tutte le componenti e i blocchi che abbiamo utilizzato per strutturare l'app, ora passiamo ad analizzare come l'app comunica con Arduino e i componenti utilizzati, quali sensore ultrasuoni e servomotore.

```
#include <SoftwareSerial.h>
#include <NewPing.h>
#include <Servo.h>
#define PIN_TRIG 8
#define PIN_ECHO 9
#define MAX_DIST 200

int bluetoothTx=3;
int bluetoothRx=2;
#include <SoftwareSerial.h>

#define PIN_SERVO 10
NewPing rilevatore(PIN_TRIG,PIN_ECHO,MAX_DIST);
SoftwareSerial bluetooth (3,2);

int gradi;
Servo myServo;

void setup(){
  Serial.begin(9600);
  myServo.attach(PIN_SERVO);
   bluetooth.begin(9600);
}
void loop(){
unsigned int cm= rilevatore.ping_cm();
 bluetooth.print(cm);
  delay(1000);
  servo();
}
  void servo() {
  if(bluetooth.available()>0){
    gradi= bluetooth.read() ;
    myServo.write(gradi);
  }
}
```

Abbiamo riscontrato delle problematiche riguardanti la comunicazione seriale, perciò abbiamo inserito nello sketch Arduino la libreria "Software.Serial.h". Questa libreria permette la comunicazione seriale utilizzando altri pin digitali, diversi da 0-RX e 1-TX (supporto seriake nativo, di tipo hardware), richiedendo l'inizializzazione di linee virtuali Tx e Rx (diventando di tipo software), che nel nostro caso sono state connesse sul pin 2 e sul pin 3. Mentre sull'applicazione sviluppata tramite processing abbiamo utilizzato il supporto seriale nativo, per l'applicazione Android utilizziamo questa specifica libreria. Questo ha significato l'inizializzazione della velocità bluetooth a 9600 baud. Nella struttura ciclica l'inserimento dell'istruzione "bluetooth.print(cm)" che stampa sulla seriale i centimetri rilevati dal sensore ultrasuoni. Nell'istruzione "servo()", invece, abbiamo introdotto il ciclo di if che dice che se il bluetooth manda qualsiasi carattere alla seriale allora trasforma il carattere ASCII nell'istruzione chiamata "gradi".

Di seguito riportiamo la struttura dell'app e un breve filmato sul suo funzionamento.