Scuola di Robotica

Comunicazione fra Arduino e NAO

Materiale necessario:

- NAO (e alimentatore)
- Arduino Yún (e cavo micro USB - USB per alimentazione/dati)
- Computer
- Router
- Cavo ethernet

Operazioni preliminari:

Accendere NAO e collegarlo all'alimentazione. Alimentare la scheda Arduino Yún collegandola al computer. Arduino Yún, NAO e computer devono essere collegati alla stessa rete locale.

Descrizione:

Nao inizialmente dice: *"Dimmi se spegnere o accendere il led"*. L'utente dirà: *"Spegni"* o *"Accendi".* Acquisito il comando vocale, NAO invierà un comando al server (la scheda Arduino Yún) e il led si comporterà di conseguenza.

Procedimento:

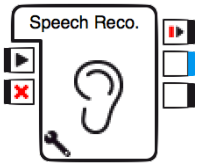
La programmazione è suddivisa in due parti:

1. Parte client - NAO
2. Parte server - Arduino Yún

---

1. Parte client - NAO

Innanzitutto, NAO dovrà ricevere il comando vocale da parte dell'utente.
Il box relativo al riconoscimento di un set di parole è "Speech Reco."



Il blocco di riconoscimento vocale presenta diversi parametri; i due più importanti sono i seguenti:

- Word list: la lista delle parole che dovrà riconoscere.
- Confidence threshold (%): soglia di accuratezza per il riconoscimento delle parole. Default: 30%.

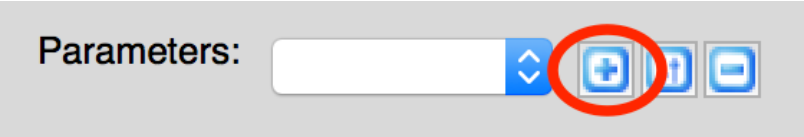
Il blocco ha due input e tre terminali di output. L'unico input che verrà utilizzato è il primo: "onStart". Mentre in output verrà utilizzato solamente il secondo: "Word Recognized".
Il terminale in questione ritorna la stringa relativa alla parola riconosciuta, che è necessariamente contenuta fra quelle precedentemente inserite nel campo "Word list" dei parametri.

Una volta acquisita la stringa è necessario trovare un modo per poter comunicare ad Arduino Yún lo spegnimento o l'accensione del LED.

La soluzione proposta è quella di utilizzare un socket, affinché la comunicazione avvenga via rete.

Un socket è una rappresentazione a livello software utilizzata per interfacciare due macchine.

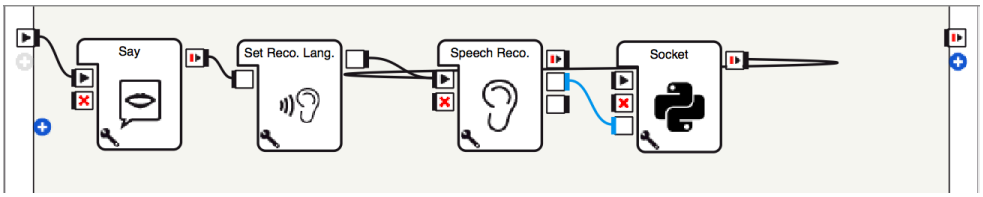
Nelle librerie di Choregraphe non sono contenuti box che realizzano il socket verso Arduino; per cui ne è stato creato uno ad hoc, sviluppandolo con il linguaggio Python.
Per creare una nuova icona Python, fare click con il tasto destro sul *Flow diagram panel*, successivamente *Create a new box, Python …*
Al fine di renderne l'utilizzo ancora più semplice, il box dovrà prevedere l'inserimento di parametri.
Per aggiungere/modificare/rimuovere parametri, click destro sul box stesso, *Edit Box.*
Nella sezione "Parameters", click sull'icona del +.
Si aprirà una finestra dove è possibile nominare il parametro, specificarne il tipo ecc.



I parametri da creare sono i seguenti:
- Yun IP Address: stringa che contiene l'indirizzo IP della scheda Arduino (server)
- Port: Numero di porta su cui il client si metterà in ascolto

L'unico output utilizzato sarà quello di tipo "bang": al termine della comunicazione esso verrà stimolato.

Programma choregraphe:



L'ultimo step, per quanto riguarda la parte Nao, è la scrittura del codice python dedicato alla comunicazione con Arduino.

L'aspetto del codice di un box appena creato è sempre lo stesso: è una classe con i seguenti metodi:
- `__init__`: costruttore; inizializza i parametri base del box. È la prima funzione che viene chiamata quando il programma è avviato.
- `onLoad`: utile solo in un *flow diagram box*
- `onUnload`: utile solo in un *flow diagram box*
- `onInput_onStart`: metodo chiamato quando viene stimolato l'input del box contrassegnato con un triangolo nero
- `onInput_onStop`: metodo chiamato quando viene stimolato l'input del box contrassegnato con una croce rossa

Gli input e gli output di una certa icona possono essere aggiunti o rimossi a piacimento.
In questo caso, l'unico input di cui si necessita è quello che accetta una stringa in ingresso.
Per aggiungere un input: click destro sul box > Edit box. Nella sezione "Inputs" click sul +; si aprirà una finestra nella quale è possibile personalizzare gli input, scegliendone nome, tipo ed altre configurazioni. Nome assegnato: ***userCommand***.

Automaticamente, verrà aggiunto del codice Python alla classe:

```
def onInput_userCommand(self, p):
      pass
```

Inoltre, è l'unico input che verrà stimolato nel programma, pertanto questo metodo conterrà la chiamata ad un altro metodo creato all'interno della classe.

Il metodo creato ad hoc svolge le seguenti funzioni:

1. Crea il socket, specificando la famiglia dell'indirizzo (INET) e il tipo di socket (STREAM)
2. Esegue il set di alcune impostazioni del socket, consentendo il riutilizzo di un indirizzo locale ed evitando il fastidioso errore: "EADDRINUSE"
3. Tenta una connessione al server, ottenendo l'indirizzo IP e la porta dai parametri (precedentemente impostati dall'utente)
4. Se la connessione è stabilita, nel pannello *Log viewer* viene visualizzato "connesso", seguito dall'indirizzo IP del server
5. Comunica ad Arduino l'accensione o lo spegnimento del led.
   Se l'utente ha detto: "accendi" viene inviato il numero `"1"`; se ha detto "spegni" viene inviato `"2"`.
6. Attende una stringa *echo* e la stampa nel pannello *Log viewer*. Lo scopo della stringa di ritorno è quello di ottenere una conferma che la comunicazione sia andata a buon fine.
7. Chiude la connessione

Il terminale di output del box socket viene stimolato. Al fine di far ricominciare il processo di acquisizione della parola, l'output del box socket è l'input del box *Speech Reco*.

Per fare un debug del programma è stato inserito self.log("…") nei punti cruciali del programma. I log vengono visualizzati nell'area denominata "Log viewer".

Codice contenuto nel box "Socket":

```
import socket #nao is a client

class MyClass(GeneratedClass):
	command = ""

	def __init__(self):
		GeneratedClass.__init__(self)

	def onLoad(self):
		#put initialization code here
		pass

	def onUnload(self):
		#put clean-up code here
		pass

	def onInput_onStart(self):
		#self.onStopped() #activate the output of the box
		#self.processing()
		pass

	def onInput_onStop(self):
		self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
		self.onStopped() #activate the output of the box

	def onInput_userCommand(self, user_word):
		self.command = user_word
		self.log("Received: " + user_word)
		self.processing()
```

```
def processing(self):
      self.log("Connessione …")
      socket_ny = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      socket_ny.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
      socket_ny.connect((self.getParameter("Yun IP address"),
      int(self.getParameter("Port"))))
      self.log("Connesso!" + self.getParameter("Yun IP address"))

      if self.command == "accendi":
            self.log("Comando ricevuto: accendi.")
            socket_ny.send('1')
      elif self.command == "spegni":
            self.log("Comando ricevuto: spegni.")
            socket_ny.send('2')
      else:
            self.log("Nessun comando ricevuto.")
            socket_ny.send('0')

      echo = socket_ny.recv(256)  #ricezione echo, buffer 256 bytes
      self.log("Ricevuto: " + str(echo))
      socket_ny.close()
      del socket_ny
      self.onStopped()
```

---

2. Parte server - Arduino Yún

Una volta terminata la parte client, risultano più chiare le funzionalità della parte server: ricevere il carattere `1` o `2` inviato da NAO, accendere o spegnere il led, inviare un messaggio *echo.*

Arduino Yún è composta da due sistemi di elaborazione: il microcontrollore **ATmega32U4** che gestisce gli ingressi/uscite analogiche/digitali della scheda e il microprocessore **Atheros AR9331** su cui "gira" il sistema operativo linino.
`Bridge` è la libreria che permette la comunicazione fra i due integrati.
Le librerie *`YunServer`* e *`YunClient`* permettono la creazione del socket.
Infine, la libreria `string` implementa alcune funzioni per l'elaborazione di stringhe.

Per praticità, il led acceso/spento da NAO è quello già integrato sulla linea digitale numero 13.

Se dalla parte client (NAO) i messaggi di debug erano visualizzati nel pannello *Log viewer*; dalla parte server (Yún) i messaggi di debug sono visualizzati sul monitor seriale.
Pertanto, nel setup sono avviate:

- comunicazione seriale con la velocità di 115200 baud
- bridge

Il metodo noListenOnLocalhost "dice" al server di iniziare ad ascoltare nuove connessioni in entrata. Dopodiché il server è avviato.

Dopo una serie di controlli condizionali ecco cosa accade:

1. Il carattere ricevuto viene salvato. Prestare attenzione! Il carattere `'1'` in ASCII è codificato con il numero 49, mentre il carattere `'2'` con 50.
2. Se il carattere è `'1'` il pin del led viene portato a livello alto. Inoltre il messaggio di *echo* è inviato al client con il metodo `write()`. Il metodo presenta due prototipi a seconda di ciò che si desidera inviare sia un carattere o una stringa. Dato che il messaggio di risposta è una stringa, write richiede la stringa da inviare e la dimensione della stessa.

Purtroppo in C++ non è concesso associare ad un array una stringa di testo nel modo seguente: array_buffer[] = “tentativo di associazione”; per poter fare ciò in modo analogo, si utilizza la funzione `strcpy(… , …)` che richiede come parametri un array (buffer) e la stringa stessa da copiare nell'array. Una volta copiata la stringa nel buffer, il messaggio *echo* viene è inviato al client.
Prestare attenzione: il buffer di questo programma è impostato a 50 caratteri.
Se il messaggio da inviare è più grande di 49 caratteri, modificare la dimensione! Altrimenti si rischia un *buffer overflow*! La dimensione del nuovo buffer dovrà sempre essere:
numero di caratteri massimi (compresi gli spazi) + 1.
Il +1 è dovuto alla presenza del carattere terminatore.

3. Comunicazione con il client chiusa.

Codice server:

```
#include <Bridge.h>
#include <YunServer.h>
#include <YunClient.h>
#include <string.h>

#define LED       13
#define PORT      7891
#define BAUD_RATE 115200

YunServer server(PORT);

void setup()
{
      pinMode(LED, OUTPUT);
      Serial.begin(BAUD_RATE);
      Serial.println("Bridge begin”);
      Bridge.begin();
      Serial.println("Bridge started”);
      server.noListenOnLocalhost();
      server.begin();
      digitalWrite(LED, LOW);
}

void loop()
{
      char echo_message[50];  // buffer

      Serial.println("Waiting server");          // for debug
      YunClient client = server.accept();
      if (client.connected())
      {
            while(client.connected())
            {
                  if(client.available())
                  {
                        Serial.println("Client available"); // for debug
                        Serial.print("Received: ");         // for debug
                        char received = client.read();
                        Serial.println(received);           // for debug
                        switch(received)
```

```
                        {
                              case 48+1: // received “1"
                              {
                                    digitalWrite(LED, HIGH);
                                    strcpy(echo_message, "Acceso da yun”);
                                    client.write(echo_message, 50);
                                    break;
                              }

                              case 48+2: // received “2"
                              {
                                    digitalWrite(LED, LOW);
                                    strcpy(echo_message, "Spento da yun”);
                                    client.write(echo_message, 50);
                                    break;
                              }

                              default:
                              {
                                    Serial.println(“Error! Char not properly
                                    received");       // for debug
                                    strcpy(echo_message, "Error! Char not
                                    properly received”);
                                    client.write(echo_message, 50);
                                    break;
                              }
                        }
                  }
            }
            client.stop();              // disconnects from the server
      }

      else
            Serial.println("\tNo client connected");      // for debug
      delay(150);
}
```

Troubleshooting

- **Non riesco a trovare l'indirizzo IP della scheda Arduino**: per trovare l'indirizzo della scheda propongo due metodi:
  1. Da IDE di Arduino: click su Tools > Port. Se la scheda è correttamente connessa alla rete locale, nella finestra verrà mostrato l'indirizzo IP.
  2. Utilizzando l'utility di rete *"nmap"*. Eseguire una scansione veloce con il comando:

     nmap -T4 -F 192.168.1.0/24
- **Il box *Speech Reco.* dà errore**: la lingua italiana è installata nel robot? Il robot su cui "gira" il programma è simulato? Perché tutto funzioni correttamente il robot deve essere reale.