

Progetto Blutis

Progetto Scolastico A.S. 2004-2005

Istituto Tecnico Industriale "G. & M. Montani" - Fermo

Bagalini Enea
Belluccini Luca
D'Ottavio Riccardo
Gaudenzi Filippo
Rao Simone
Straccia Fabrizio



ITIS MONTANI FERMO



Indice dei contenuti

Introduzione	
Tecnologie Utilizzate	4
Linguaggi di Programmazione Utilizzati	8
Comunicazione Cellulare-Dispositivo	
Java Client	16
Comunicazione Dispositivo-Cellulare	
PBASIC Server	22
Costruzione	
Struttura	28
Elettronica	30
Conclusioni	
Possibili Miglioramenti	33
Cronologia Conoscenze	35
Ringraziamenti	36
Link e Documentazione Proprietaria	37



Introduzione

L'obiettivo di questo progetto era quello di riuscire a far interagire più moduli conoscitivi al fine di creare un prodotto finito, analizzandone le problematiche di progettazione e implementazione.

Dopo una breve analisi si è giunti all'idea di realizzare un piccolo veicolo ("Blutis Car"), controllabile via Bluetooth.

L'innovazione è costituita proprio da questa ultima scelta: infatti esso garantisce un funzionamento più rapido e affidabile, oltre ad essere oramai presente come standard de facto, in molti dispositivi portatili.

Il software è supportato dal lato "Blutis Car", da una scheda madre altamente integrata, in cui i calcoli sono affidati ad un microcontrollore che riesce ad interpretare il linguaggio PBASIC (ideato dalla Parallax Inc.) e la trasmissione ad un modulo Bluetooth (prodotto dalla A7Engineering) opportunamente gestito.

Il Controllo Remoto è invece affidato ad un software scritto in linguaggio JAVA (in particolare J2ME) installato in uno Smartphone, il quale deve possedere un transceiver Bluetooth (nel nostro caso un Nokia 6600).

Nella trasmissione il lato client sarà eseguito dallo smartphone, mentre la Blutis Car fungerà da server.



Introduzione – Tecnologie Utilizzate

Bluetooth

Il Bluetooth è uno standard di comunicazione senza fili con il quale diventa possibile connettere un'ampia gamma di dispositivi in modo robusto e sicuro.

Il nome Bluetooth, deriva dal soprannome di un famoso condottiero scandinavo, re di Danimarca del medioevo, Harald II Bluetooth che conquistò la Norvegia. Lo standard Bluetooth rende possibile la creazione di reti di dispositivi che, in questo modo, possono scambiare dati rendendo possibile l'accesso a diversi servizi nell'ambito di quella che viene chiamata PAN (Personal Area Network).



L' utilizzo dello standard Bluetooth è in continua espansione: va dalla creazione di reti domestiche o aziendali wireless fino alla comunicazione negli autoveicoli. Lo standard prevede una connessione a corto raggio, piccola potenza di trasmissione, dunque, basso consumo di energia. Inoltre, le dimensioni molto ridotte dei moduli Bluetooth, portano ad una riduzione degli ingombri. La sicurezza della connessione è un punto di forza rispetto ad altri standard per evitare spiacevoli intrusioni di terzi.

Il Funzionamento e Informazioni Tecniche

Le trasmissioni wireless sono inglobate nello standard IEEE 802.11. La 802.11 prevede un'unica interfaccia a livello di Data Link e due possibili implementazioni a livello Physical Layer: Irda e a onde radio.

Le trasmissioni nel Bluetooth sono effettuate usando un ricetrasmittitore che opera nella frequenza di 2,4 GHz, frequenza assegnata per usi industriali. Le frequenze utilizzate variano da paese a paese, in Italia il Frequency Range è tra i 2400 e 2483,5 MHz.

Una piccola rete Bluetooth, chiamata piconet, può supportare un collegamento punto a punto (point to point) e multi punto (multipoint) fino ad un massimo di 8 dispositivi. Il sistema di comunicazione bluetooth è progettato per funzionare anche in ambienti con forte presenza di interferenze e campi elettromagnetici, ciò per assicurare collegamenti sempre efficienti e affidabili.

Ciò è possibile grazie alla tecnica del Frequency Hoping: ad ogni slot di trasmissione o ricezione i dispositivi sono pronti a cambiare la frequenza portante. Ogni slot corrisponde a 625us e ogni trasmissione può occupare al massimo 5 slot consecutivi.

La ri-trasmissione dati, nel caso di collisioni o disturbi, è affidato alla tecnica ARQ (Stop&Wait).

La velocità di comunicazione è prossima ad 1 Mbps anche con piccole potenze nell'ordine di alcuni milliWatt, mille volte inferiore alla potenza di un cellulare GSM, impiegando la tecnologia TDD (Time Division Duplex, nella gestione del traffico asimmetrico, come ad esempio per internet).

In un collegamento tutti gli apparecchi Bluetooth connessi sono generalmente in modalità standby, cioè di attesa, seguendo un ciclo di scansione ad intervalli di tempo di 1,28 secondi al fine di verificare la presenza di eventuali altri dispositivi; in tale modalità tutti i dispositivi Bluetooth sono a basso consumo energetico.

La ricerca delle periferiche avviene tramite una Inquiry, Paging e Scan.

Il pacchetto dati è costituito principalmente da 3 parti, per un massimo di 2745 bit:

ACCESS CODE (72bit)	HEADER (54bit)	PAYLOAD (0-2745bit)
---------------------	----------------	---------------------

I riscontri sono contenuti nell'header, quindi sono trasmessi in Piggy-Backing. Ogni periferica è identificata da un codice univoco di 12 valori esadecimali (48 bit), che corrisponde al MAC del dispositivo. E' possibile assegnare un "friendly name" per un più agevole riconoscimento.

Lo schema sottostante rappresenta gli strati del protocollo Bluetooth.

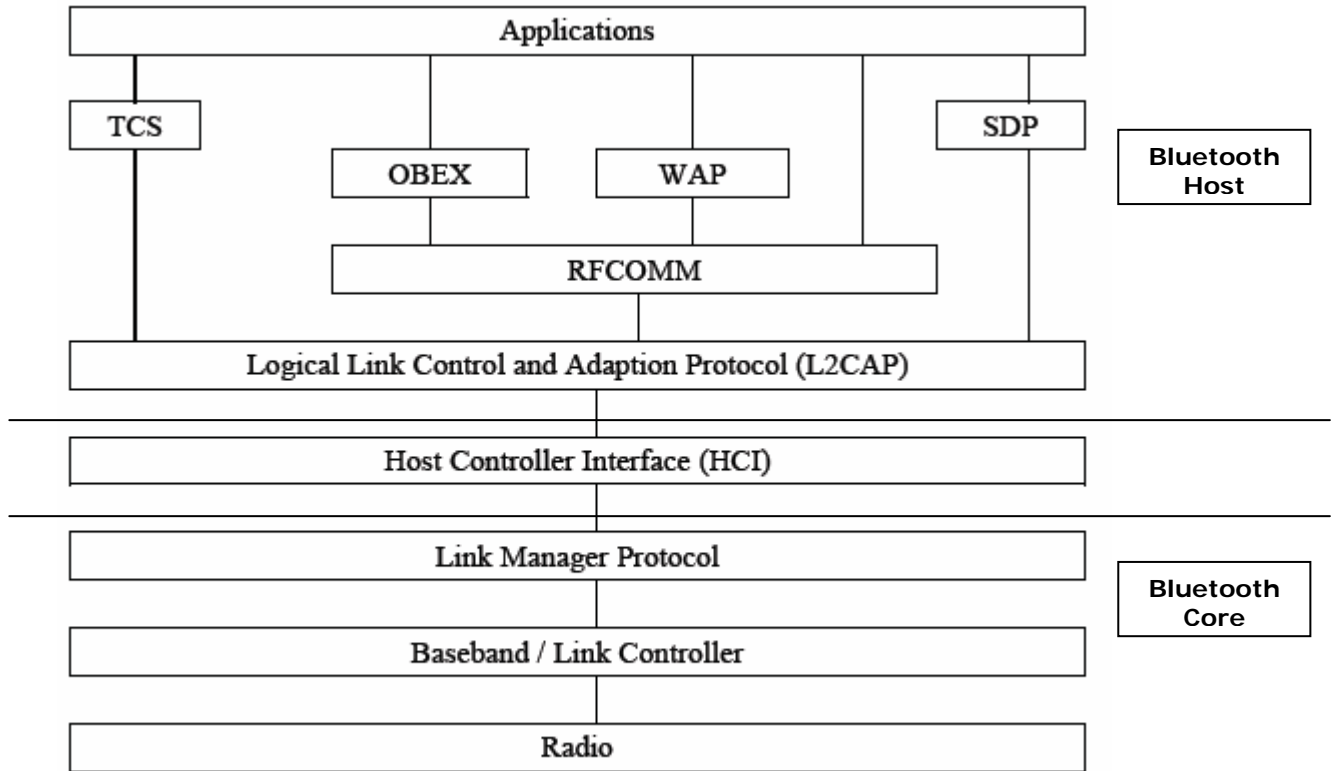


Figure 2.1 The Bluetooth protocol stack

A7Eng EB500

Nel nostro progetto, l'antenna Bluetooth utilizzata è prodotta dalla società americana A7Engineering, in stretta collaborazione con la Parallax Inc. E' un'antenna programmabile e gestibile tramite porta seriale.



Nella tabella seguente sono elencate le caratteristiche tecniche del modulo.

Transmit Power	4dBm (max) class 2 operation	
Open Field Range	eb500-AHC-IN (surface mount antenna) – 100 meters (328 feet) <i>(Actual range is dependent upon location and environment.)</i>	
Receiver Sensitivity	-85dBm	
Operating Temp.	-15° to 70°C	
Supply Power	5 to 12VDC	
Current Consumption	115.2kbps data transfer: 35mA 38.4kbps data transfer: 30mA 9.6kbps data transfer: 25mA	connected and idle: 8mA no connection: 3mA
Interfaces	5V logic level UART or RS232 with optional eb600 adapter Baud rate 9.6k – 230.4k Flow control: RTS/CTS or none	
Connector	One 10x2 AppMod compatible 20 pin 0.1" header	
Antenna	Matched internal surface mount	
Bluetooth Support	Version 1.2 compliant with profiles L2CAP, RFCOMM, SDP, SPP	
Firmware	Upgradeable via PC application with eb600 adapter	

Pololu MicroController

Il modulo in questione consente di controllare 1 o 2 motori (rispettivamente con 2 e 1 Ampere di corrente) attraverso l'utilizzo di una tensione logica molto bassa (anche 1.5V) e di una linea seriale, su cui è necessario inviare un pacchetto di configurazione iniziale.

Con dei semplici invii su linea seriale è possibile variare la velocità (da 0 a 127) e la direzione dei motori o del motore.

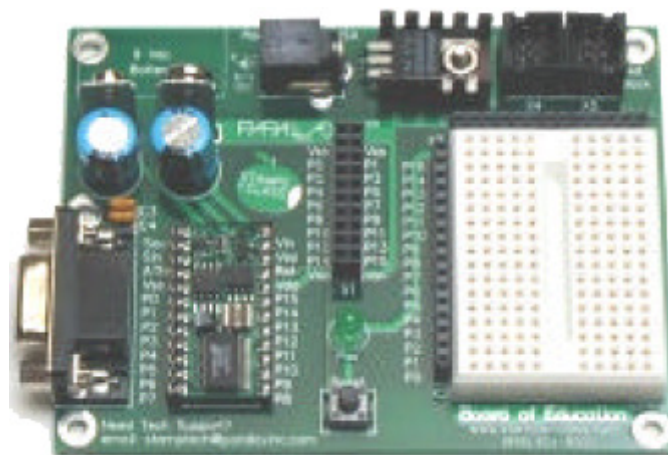
La tensione massima per i motori è di 9V.



Introduzione – Tecnologie Utilizzate

Microcontrollore BS2 e Scheda Madre BOE

Il BASIC Stamp 2 è un modulo operativo a 24-pin. E' più specificatamente un DIP (Dual Inline Package) e contiene al suo interno il processore, una sua memoria, un generatore di clock e l'interfaccia (costituita da 16 pin di I/O).



Il BS2-IC è programmabile attraverso porta seriale: dopo la programmazione, la porta potrà essere utilizzata per altri scopi, come input o output.

Tra le caratteristiche abbiamo la sua frequenza operativa (20MHz), la dimensione della RAM (32 Bytes), la dimensione della EEPROM (2KBytes), il numero di pin di I/O (16).

La scheda madre offre il supporto per tutti i vari moduli hardware sviluppati dalla stessa Parallax, il che facilita il collegamento dei componenti.



Introduzione – Linguaggi di Programmazione Utilizzati

PBASIC

Gli STAMP sono programmati con una variazione del BASIC chiamata PBASIC. Questo linguaggio è molto simile all'Assembler, ma dispone di un set di istruzioni molto più ampio e user-friendly.

Il PBASIC permette di:

- Leggere e Scrivere dalle porte di I/O
- Misurare e generare impulsi
- Elaborare dati
- Inviare e ricevere dati seriali
- Generare frequenze
- Generare toni DTMF

Inoltre, disponendo di processori più evoluti, è possibile intercettare interruzioni.



Introduzione – Linguaggi di Programmazione Utilizzati

JAVA

Java è un linguaggio di programmazione nato negli anni novanta, e destinato a diventare in breve tempo, il linguaggio più utilizzato in assoluto. Nato per puri scopi di ricerca in ambito universitario, e non come in molti altri casi, per volere di una multinazionale per conquistare nuove fette di mercato, Java si pone come un "super-linguaggio" che va oltre i limiti e i difetti appartenenti agli altri linguaggi.



Successivamente, la Sun Microsystems ha fatto in modo che dal linguaggio si siano poi evolute una serie di famose tecnologie (JSP, Servlet, EJB, Jini, RMI, etc.) che si sono diffuse in molti ambiti del mondo della programmazione. Con il termine "Java" ci si riferisce sia al linguaggio, sia alla tecnologia che racchiude tutte le tecnologie sopra elencate. Il vantaggio dell'utilizzo di Java, oltre alla portabilità del codice e l'utilizzo di un unico linguaggio per la programmazione di applicazioni dai dispositivi mobili alle applicazioni di business, è quello di consentire maggiore flessibilità nell'uso dei servizi mobili e di sfruttare il processore del dispositivo mobile per l'esecuzione di applicazioni scaricabili attraverso la rete.

Questa tecnologia consente lo sviluppo e la realizzazione di applicazioni Java per dispositivi mobili grazie programmazione con le MIDlet, le classi Java alla base della programmazione per dispositivi mobili, cominciando dai palmari fino ad arrivare ai telefoni cellulari di ultima generazione.

J2ME

La piattaforma J2ME (Java 2 Mobile Edition) fornisce un ambiente di sviluppo ottimizzato e "leggero" per girare su cellulari e Pda di ultima generazione. Come le altre piattaforme (J2SE) e (J2EE), J2me include una java virtual machine e una serie di API che offrono i comandi necessari per l'implementazione di nuovi servizi all'interno del programma. J2me trasporta la potenza e i benefici della tecnologia java al consumatore e ai dispositivi integrati (cellulari, palmari).

Ricordiamo che Java è un linguaggio di programmazione, object oriented sviluppato da Sun Microsystems con lo scopo di programmare elettrodomestici, ma la storia ha voluto che questo linguaggio si spostasse con il tempo prevalentemente sulle applicazioni di rete e applicazioni portabili da sistema a sistema.

Java 2 Micro Edition, propone un'alternativa molto valida per la programmazione di dispositivi diversi da Personal Computers, tornando quasi volutamente all'applicazione attribuitagli in origine.

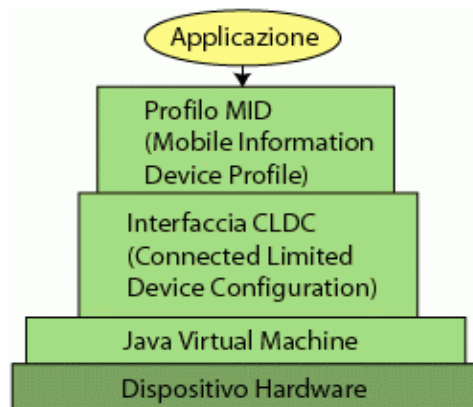
Quando non programiamo per un PC, i requisiti di sistema sono molto diversi e variano da dispositivo a dispositivo, per questo sono stati introdotti due profili standard che si traducono in due versioni di J2ME: CDC e CLDC.

CDC sta per Connected Device Configuration e comprende una fascia di dispositivi avanzati non sempre mobili. Alcuni esempi sono dispositivi di telefonia fissa e ricevitori per TV e tutti i dispositivi non mobili, cioè dispositivi che sono connessi ad un'alimentazione fissa per intenderci.

CLDC è l'acronimo di Connected Limited Device Configuration, questa versione di J2ME è progettata per sviluppare applicazioni per i normali telefoni cellulari e PDA (Personal

Digital Assistant, che possono essere programmati anche in configurazione CDC). Con questa configurazione si definisce il profilo delle applicazioni destinate ai dispositivi mobili ovvero MIDP (Mobile Information Device Profile, tradotto in italiano è qualcosa come Profilo per Dispositivi Informativi Mobili).

Analogamente a *J2SE (Java 2 Standard Edition)* la Micro Edition lavora a strati basandosi su una macchina virtuale, chiamata specificatamente *Virtual Machine*. Quest'ultima gira in modo nativo sul dispositivo perché è un programma scritto a basso livello appositamente per *quel dispositivo*. La *Virtual Machine*, si occupa di interpretare e quindi eseguire un codice composto da simboli ognuno con la dimensione di 8 bit = 1 byte che può essere associato concretamente ad un carattere, da questo prende il nome di *bytecode*.



La figura mostra i livelli di astrazione di questo sottosistema. Partendo dal livello più in basso, si ha il *Dispositivo Hardware* che rappresenta l'oggetto fisico su cui la nostra applicazione dovrà girare, ad esempio il nostro telefono cellulare con funzionalità Java (*CLDC*). Direttamente sul dispositivo, c'è la macchina virtuale Java che si occuperà dell'interpretazione del *bytecode*. Andando più su, c'è l'interfaccia *CLDC* che si tratta di un' *API (Application Programming Interface)* e si trova più a basso livello in termini di applicazione/dispositivo; essa comprende un set di funzioni che consentono di interagire a basso livello con il dispositivo stesso. Il profilo MID (*MIDP*) occupa un livello più alto di astrazione, si tratta sempre di un' *API* ma comprende funzioni di gestione della grafica, della *GUI (Graphical User Interface)* e altre funzioni ad alto livello. Nel gradino più alto infine, si trova l'applicazione che abbiamo scritto e che sfrutta tutti gli strati di astrazione per poter funzionare.

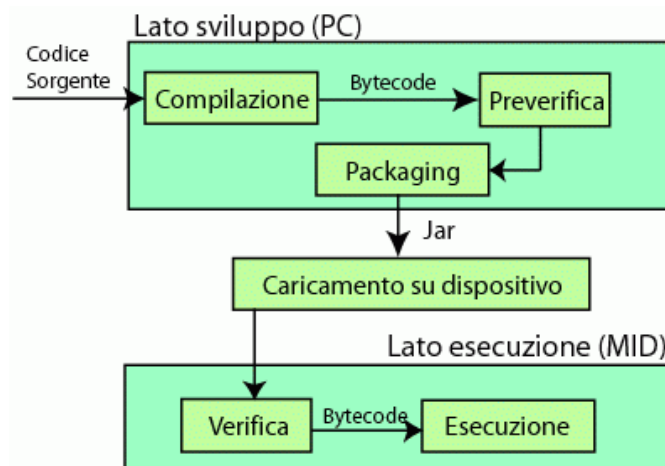
Per realizzare un programma in *j2me*, differenti sono i passi che si devono seguire per la creazione del file.jar:

- La prima fase è la scrittura del codice, se non si dispone di un ambiente di sviluppo integrato apposito per queste applicazioni (*IDE: Integrated Development Environment*) si utilizza un normale editor di testi e si inizia a scrivere il codice sorgente utilizzando come supporto una reference (si spera mnemonica, ma in mancanza di preparazione si può ricorrere alla documentazione dell'*SDK*) delle funzioni specifiche *MIDP/CLDC*
- la seconda fase è la compilazione del sorgente: per effettuare questa operazione viene utilizzato il normale compilatore *javac* fornito con *Java 2 Platform SE*. Il sorgente viene convertito in *bytecode* interpretabile dalla macchina virtuale del dispositivo

- La terza fase è la preverifica del bytecode: Il bytecode generato dal compilatore, deve essere sottoposto ad una fase di preverifica di correttezza tramite un programma apposito contenuto all'interno del pacchetto *SDK* relativo a *J2ME* chiamato *preverify*. Il bytecode preverificato con successo, è in grado di girare correttamente su dispositivo mobile.
- La quarta fase è il packaging: quando si sviluppa con *J2ME*, effettuare il packaging (impacchettamento) del codice è obbligatorio. Questo significa creare un archivio compresso di formato ZIP, ma con estensione *JAR* (*Java ARchive*) contenente tutte le classi che compongono la nostra applicazione. Per creare questo archivio solitamente si utilizza l'utility *jar* contenuta nel pacchetto *Java 2 SE Platform*.

Dopo aver creato il *JAR* siamo pronti per la quinta fase, ovvero il caricamento dell'archivio su dispositivo mobile. Questa fase è molto semplice perché si tratta di utilizzare una connessione, come Infrarosso, Bluetooth o USB per trasferire il file su dispositivo.

Ecco lo schema che viene eseguito:



Per la creazione del codice abbiamo bisogno di un emulatore: un software specifico che simula il comportamento reale del dispositivo mostrando l'esecuzione del programma su un telefonino/palmare virtuale presente sullo schermo del computer. In questo modo è possibile anche evitare il ripetuto caricamento del programma su *MID* durante la fase di testing.

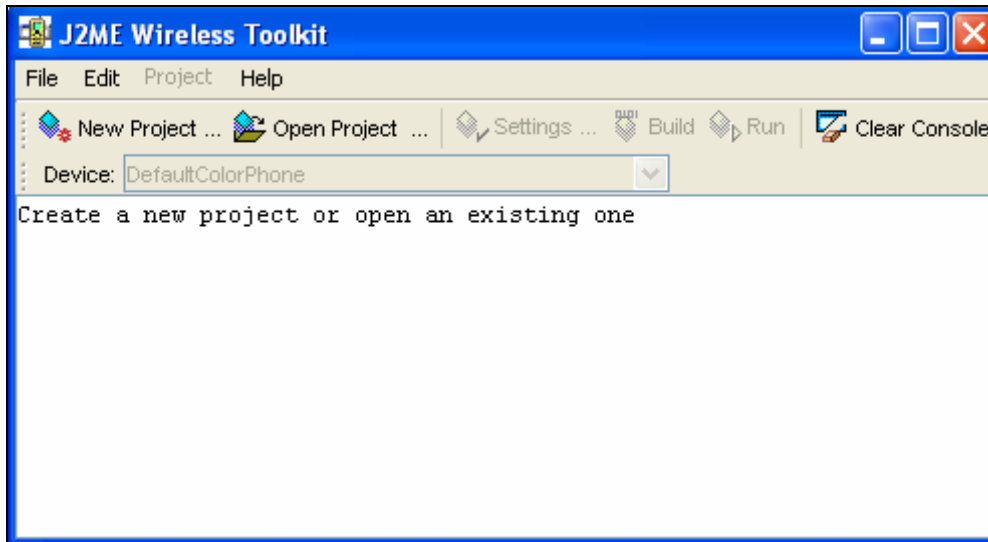
Il programma che utilizzeremo è chiamato *J2me Wireless toolkit* (Reperibile al seguente link: [HTTP://JAVA.SUN.COM/PRODUCTS/J2MEWTOOLKIT/DOWNLOAD-2_2.HTML](http://java.sun.com/products/j2mewtoolkit/download-2_2.html)).

WIRELESS-TOOLKIT

Per poter cominciare a creare una prima applicazione J2ME occorre il *J2ME Wireless Toolkit*, messo a disposizione dalla Sun, che consente di compilare e testare con degli emulatori i codice che scriviamo, implementeremo poi la spiegazione con la scrittura di un semplice programma di esempio (il classico Hello World). L'installazione del *J2ME Wireless toolkit* non richiede nulla di particolare se non un *JDK* o un *JRE* 1.3 (o superiore) e la scelta di installare il toolkit integrandolo con *ForTE for Java*, oppure in versione standalone. Per il progetto *BLutis* viene utilizzata la versione standalone che non ha un supporto diretto per la scrittura del codice che deve essere fatta con un editor esterno (per i primi esempi il Blocco Note di Windows va benissimo)

Il Toolkit fornisce una gestione di progetti, ordinandoli seguendo una precisa struttura a cartelle, che deve essere seguita se si vuole realizzare un'applicazione come si deve.

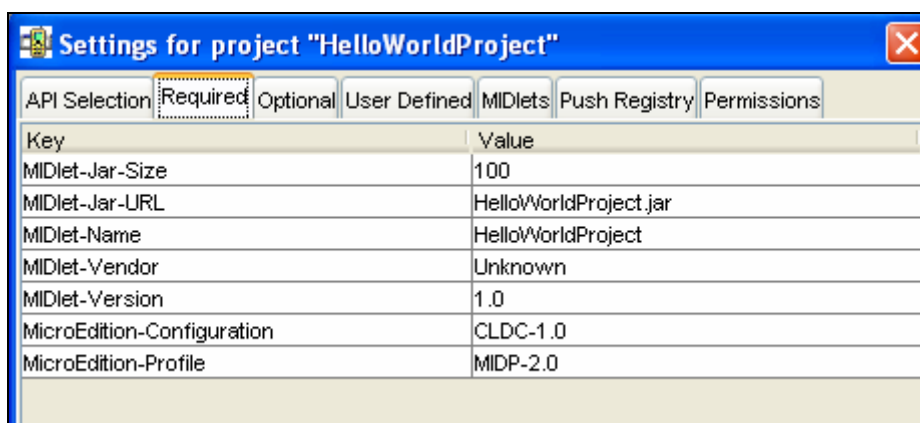
Per creare quindi un progetto apriamo la Ktoolbar, base di partenza per tutte le Midlet che vogliamo creare.



Clicchiamo su *New Project* e specifichiamo due informazioni: nome del progetto (che in verità è il nome della cartella dove viene collocato sul disco) e nome del Midlet iniziale (in futuro potremmo mettere più midlet all'interno dello stesso progetto). Apparirà poi una schermata dove ci vengono richieste quelle che sono meta-informazioni sull'JAR file da costruire: le ritroveremo poi nel descrittore e nel manifest file. Da notare l'ultimo tab, MIDlets: consente l'aggiunta di ulteriori midlet nello stesso Jar: sarà poi dal terminale che scegliamo quale eseguire in base ad una lista.

Avendo installato con l'opzione standalone, andiamo sulla directory in cui è stato installato J2ME Wireless Toolkit (di solito Wtk22), apriamo la cartella apps che contiene tutti i progetti ed apriamo la cartella HelloWorldProject (o il nome del vostro progetto), che si presenta con questa struttura:

- HelloWorldProject
 - > bin
 - > res
 - > src



all'interno di *bin* dove non si trovano le classi compilate, come sarebbe intuibile, ma il file *jar* ed il relativo descrittore; *res* è una cartella generica di risorse (spesso immagini PNG) ed *src* è la nostra cartella di lavoro, o meglio la radice del classpath da cui verranno ricercate e compilate le classi dell'applicazione. Alla prima compilazione verranno create le cartelle *classes*, dove questa volta troviamo le singole classi compilate, ed una cartella temporanea. Il descrittore viene utilizzato per eseguire dei controlli base da parte del software di gestione applicazioni del dispositivo portatile: controllo versioni, dimensioni del jar file, nome autore ecc...

La scrittura di un MIDlet è abbastanza intuitiva per chi è già pratico di Java e grazie ad uno studio della documentazione, rigorosamente javadoc e a qualche esempio già scritto, si riesce da subito a realizzare qualcosa di funzionante. Per la stesura di un MIDlet bisogna però seguire delle regole, né più né meno come si è soliti fare per le applet:

- la classe deve estendere la classe `javax.microedition.midlet.MIDlet`
- deve implementare i metodi `startApp`, `pauseApp`, `destryApp` (corrispondenti nella funzionalità ai metodi `init`, `stop`, `destroy` delle applet) che sono dichiarati `abstract` nella classe da cui si estende.

Per poter scrivere sul display occorre fare uso delle funzionalità grafiche (che non sono direttamente derivate dalle AWT/Swing) molto semplici data la natura più limitata della GUI di un CLDC. E' stato deciso di seguire il paradigma di contenitori e componenti, come tutta la tecnologia Java: il principale contenitore è il `Display Manager` implementato dalla classe `javax.microedition.lcdui.Display`. Ad un `Display` è associato un oggetto `Displayable`, che non è altro che un generico contenitore di oggetti grafici che è possibile visualizzare sul display. Quest'ultimo opportunamente esteso classifica due categorie, caratterizzate dalla gestione ad alto o a basso livello della grafica e degli eventi: lo `Screen` e sottoclassi per il primo caso, il `Canvas` nel secondo.

Come detto nella lezione dedicata al Wireless Toolkit i nostri codici sorgenti, salvati in file con estensione `.java`, vanno posizionati nella cartella `X:/wtk22/apps/HelloWorldProject/src`. Creiamo il file `HelloWorld.java`, e scriviamo il codice, iniziando dagli import:

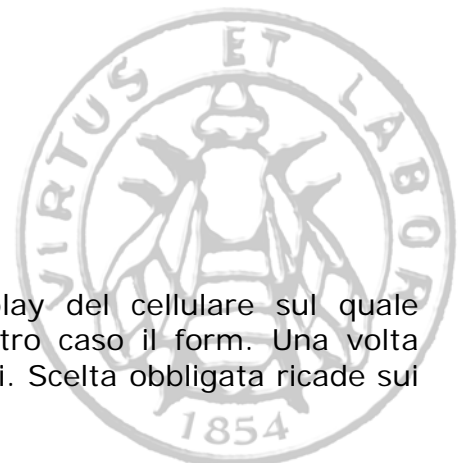
```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

Il primo package è costituito soltanto dalla classe `MIDlet` (astratta) la classe da cui derivare tutte le nostre midlet, il secondo è il package base per la gestione dell'interfaccia grafica.

Dichiaro la classe estendendo `MIDlet`:

```
public class HelloWorld extends MIDlet {
    Display display;
    Form form;
    .
    .
}
```

L'attributo `display`, come detto, è l'astrazione del display del cellulare sul quale impostare qual'è il componente da visualizzare, nel nostro caso il `form`. Una volta dichiarati gli attributi, passiamo alla definizione dei metodi. Scelta obbligata ricade sui



metodi startApp, pauseApp e destroyApp, ma nessuno ci vieta di aggiungerne altri. Per le nostre esigenze è sufficiente implementare lo startApp, invocato una volta istanziato l'oggetto, dato che dovremo gestire un unico evento, quello del lancio dell'applicazione per visualizzare una scritta.

```
public void startApp() {
    display = Display.getDisplay(this);
    //ottengo il display
    form = new Form("Itis montani Fermo");
    //creo il contenitore
    StringItem sltem = new StringItem(null, "Hello World!");
    //creo il componente
    form.append(sltem);
    //aggiungo il componente al contenitore
    display.setCurrent(form);
    //imposto come displayabile corrente
}
```

Il sorgente completo quindi dovrebbe apparire così:

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

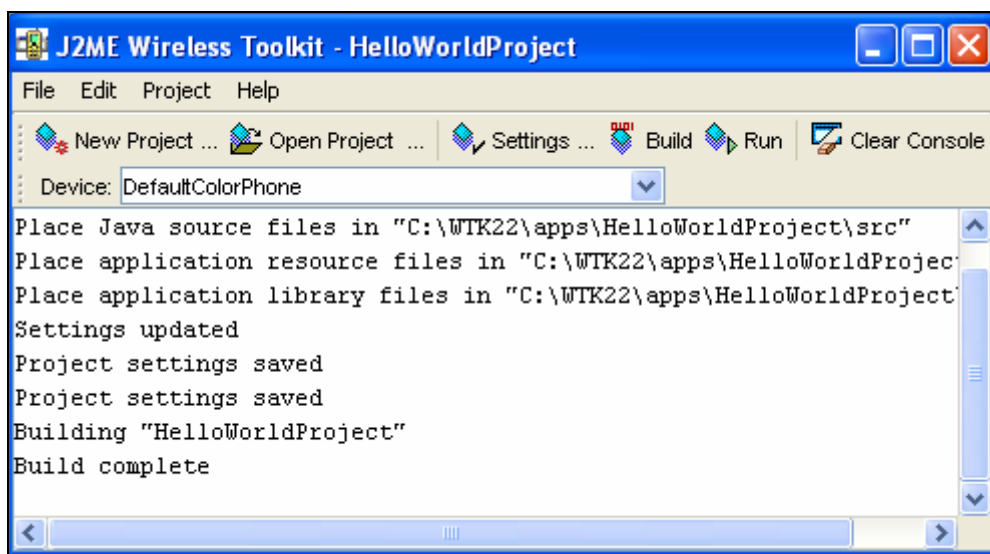
public class HelloWorld extends MIDlet {
    Display display;
    Form form;

    public void destroyApp(boolean unconditional) {
        notifyDestroyed();
    }

    public void pauseApp() {
    }

    public void startApp() {
        display = Display.getDisplay(this);
        form = new Form("Itis Montani Fermo");
        StringItem sltem = new StringItem(null, "Hello World!");
        form.append(sltem);
        display.setCurrent(form);
    }
}
```

Andiamo sul Ktoolbar del WK, clicchiamo su Build (che non fa altro che compilare i file java) e dovrebbe dare come risultato la seguente schermata:



In questo caso non ci sono stati errori di compilazione e la nostra mildet può essere eseguita con il pulsante **run** che caricherà l'applicazione su uno degli emulatori selezionati

Il risultato sarà il seguente:



Comunicazione Cellulare-Dispositivo Lato Client software Java

Il programma eseguito dallo smartphone si articola su due classi: Blutis1 e MyCanvas.

BLUTIS1

Blutis1 è la classe principale, essa contiene sia i metodi necessari all'esecuzione di una MIDlet, quali la startApp(), pauseApp() e destroyApp(), nonché i metodi per la gestione degli eventi relativi ai Command, e della connessione Bluetooth. Tuttavia per poter usufruire di metodi diversi da quelli fondamentali è necessario implementare le relative interfacce. Nel nostro caso per poter effettuare la ricerca di dispositivi Bluetooth e la gestione dell'evento commandAction (pressione dei Command) abbiamo implementato la DiscoveryListener e la CommandListener.

Di seguito, la linea di comando che crea la classe principale Blutis1 e implementa le 2 interfacce prima nominate:

```
public class Blutis1 extends MIDlet implements CommandListener, DiscoveryListener {
    public void destroyApp(boolean unconditional) {}
    public void pauseApp() {}
    public void startApp(){}
    public void commandAction(Command c, Displayable d) {}
    public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass
deviceClass){}
    public void inquiryCompleted(int discoveryType) {}
    public void serviceSearchCompleted(int transId, int responseCode) {}
    public void servicesDiscovered(int transId, ServiceRecord[] serviceRecords) {}
}
```

Il metodo commandAction (Command c, Displayable displayable) viene richiamato ad ogni click su un Command. E' possibile riconoscere quale sia il command interessato dall'evento attraverso il parametro c della funzione. Di seguito viene riportato un esempio sulla gestione di un Command:

```
if (c == cmExit){ //PRESSIONE TASTO EXIT
    //i due comandi di seguito permettono la chiusura e l'uscita dalla MIDlet
    destroyApp(false);
    notifyDestroyed();
} //end if
```

Per la ricerca di dispositivi Bluetooth l'interfaccia DiscoveryListener implementa differenti metodi, prima di andarli ad analizzare sarà necessario capire il funzionamento di una connessione tramite tecnologia Bluetooth.

Innanzitutto dobbiamo distinguere differenti fasi relative alla connessione vera e propria.

La prima fase coincide con l'inizializzazione delle variabili Bluetooth. Dato che stiamo analizzando il lato client, dovremo creare un oggetto relativo al device locale e uno relativo ai possibili device remoti.

```
//variabili necessarie per la connessione BT
public LocalDevice device;
//INIZIALIZZAZIONE VARIABILI E OGGETTI BT
try {
    device = LocalDevice.getLocalDevice(); // ottiene i dati relativi al device locale
    String address = device.getBluetoothAddress(); //ottiene il MAC del device locale
    device.setDiscoverable(DiscoveryAgent.GIAC); // set Discovery mode to GIAC
} catch (BluetoothStateException e){
    show_alert ("Errore BT\n" + e.getMessage(), 1, -2, form_inizio);
}
}
```

Il passo successivo è quella della ricerca. Il dispositivo Bluetooth che vuole iniziare una comunicazione deve effettuare una ricerca per trovare gli altri dispositivi raggiungibili (nel nostro caso tale ricerca potrebbe risultare superflua, dato che siamo in possesso di una sola antenna eb500). Esistono diversi modi per gestire la ricerca:

- Creazione di una coda di device remoti ;
- Ricerca nelle liste di device BT precedentemente salvati.

Il primo metodo necessita di un oggetto DiscoveryAgent. Questo permette di attivare la ricerca attraverso la startInquiry. Quando si richiama questo metodo va specificato il tipo di coda: GIAC (General / Unlimited Inquiry Access Code) o LIAC (Limited dedicated Inquiry Access Code).

I dispositivi remoti trovati vengono restituiti attraverso il metodo DeviceDiscovered. Al termine di ogni ricerca viene richiamato il metodo inquiryCompleted, attraverso le sue informazioni si riesce a determinare se la ricerca è giunta a buon fine o se è terminata a causa di un errore tecnico o a causa della decisione dell' utente.

```
public DiscoveryAgent agent;
//INIZIALIZZAZIONE E START RICERCA
try {
    agent = device.getDiscoveryAgent(); // obtain reference to singleton
    agent.startInquiry( DiscoveryAgent.GIAC, this );
} catch (BluetoothStateException e){
    show_alert ("Errore BT\n" + e.getMessage(), 1, -2, form_inizio);
}

//RESTITUZIONE DEVICE
public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass){
}
```

La variabile remoteDevice va ad indicare il dispositivo remoto trovato, da questo è possibile ricavarne poi l'indirizzo e altri dati, tuttavia è possibile ottenere delle informazioni più dettagliate andando ad utilizzare i metodi relativi all'altro parametro (per esempio è possibile sapere il tipo di device).

Il secondo metodo invece viene richiamato attraverso la retrieveDevices. Va specificato per questo metodo un parametro, questo può assumere il valore CACHED o PREKNOW. CACHED va a ricercare quei devices che sono già conosciuti dal dispositivo locale (specificati nella BBC), e cioè i device con i quali il dispositivo locale comunica frequentemente. PREKNOW va ad indicare invece una ricerca nella lista di device creata da una precedente Inquiry.



I metodi relativi alla ricerca di un servizio qui non vengono sviluppati e spiegati perchè, trovandosi a lavorare con un dispositivo semplice, la eb500 non permette l'erogazione di un particolare servizio.

Una volta scelto il device è possibile stabilire la connessione ed aprire uno stream per l'invio o per la ricezione dati.

```
String MAC_DEV; //mac del device remoto
StreamConnection conn;
public OutputStream out;
public String connString; //stringa che contiene l'indirizzo del device
connString = "btspp://" + MAC_DEV +
":1;master=false;encrypt=false;authenticate=false";
//INSTAURAZIONE DELLA CONNESSIONE
try{
conn = (StreamConnection)Connector.open(connString);
out = conn.openOutputStream();
} catch (IOException e1){
show_alert ("Connessione fallita al device BT fallito", 1, -2, form_inizio);
}
```

La sua chiusura viene effettuata grazie alle funzione close di entrambi gli oggetti.

```
//CHIUSURA DELLA CONNESSIONE E DEL CANALE DI OUTPUT
try{
out.close();
conn.close();
} catch (IOException e1){
show_alert ("Errore durante la scinnessione", 1, -2, form_inizio);
}
```

L'invio di dati, che avviene attraverso lo stream di output, è eseguito grazie alla funzione write. Esistono diverse modalità di write tutte contenute nelle specifiche della classe OutputStream.

```
//INVIO DI UNA STRINGA
try{
String buff="prova invio"
out.write(buff.getBytes());
} catch (IOException e1){
show_alert ("Invio Fallito", 1, -2, form_inizio);
}
```

La funzione Blutis1 oltre a contenere Tutta la parte relativa alla connessione possiede anche tutta la programmazione relativa alla costruzione e alla visualizzazione delle form, nonché gli algoritmi di ricerca.

Infatti, la classe si articola su 4 form:

- form_inizio;
- form_png;
- form_about;
- form_send.

La form_png contiene il video introduttivo ed cioè la presentazione del programma. Il video è in formato 3gp(formato leggibile da software nokia su cellulare 6600).

La form_inizio, accessibile da form_png, contiene una label con il proprio Mac e una TextBox, Targa, dove dovrebbe essere inserito il mac del dispositivo BT a cui conettersi.

La ricerca avviene attraverso una inquiry e se il dispositivo richiesto non venisse trovato o non fosse disponibile, il programma è predisposto per mostrare una lista di tutti i device BT presenti nel raggio di azione della propria antenna.

Scelto il device e instaurata la connessione viene visualizzata la form_send, coadiuvata dalla classe Mycanvas(cioè per permettere la gestione della tastiera).

La form_about non è altro che la form contenente tutte le informazioni necessarie all'utente per un uso adeguato del software. La form_about è la "Guida in linea".

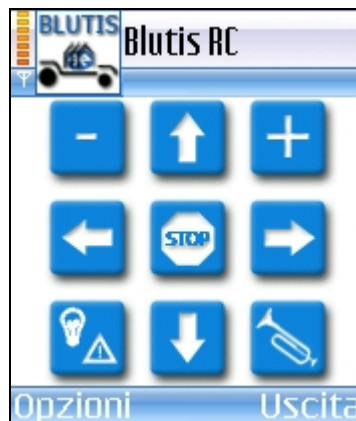
MYCANVAS

La classe Canvas fornisce all'utente più metodi, con i quali si rende possibile la gestione della grafica a basso livello sul display e la possibilità di intercettare la pressione dei tasti da parte dell'utente sul dispositivo. Per basso livello si intende la possibilità di andare a gestire, in maniera molto vicina all'hardware (in questo caso il nostro display), i pixel dello schermo. Per prima cosa andremo ad analizzare la gestione della grafica relativa al nostro programma, successivamente tratteremo l'intercettazione dei tasti premuti.

I metodi che vengono utilizzati dal programma sono soltanto una minima parte di quelli che la classe Canvas mette a disposizione. Prima di andare ad analizzare i metodi veri e propri è bene dare prima uno sguardo alle problematiche che l'utilizzo di una Canvas può causare.

Attualmente sul mercato sono disponibili differenti tipi di cellulari, i più moderni hanno schermo a colori a differenza di altri forniti di schermo bianco e nero. Le dimensioni del display variano a secondo del tipo di cellulare e della ditta produttrice. Ecco quindi che realizzare un programma capace di adattarsi a tutti i dispositivi diventa un'impresa. Il consiglio che si può dare è quello di non utilizzare delle grandezze fisse, ma utilizzando parametri variabili.

Per esempio utilizzando rispettivamente i metodi `getHeight()` e `getWidth()` avremo la possibilità di conoscere l'area in pixel del nostro schermo. In seguito verranno riportati esempi sull'implementazione di questi 2 metodi.



Il metodo indispensabile per disegnare grafica è `paint(Graphics g){}`, infatti attraverso l'oggetto `Graphics g` ai suoi metodi è possibile disegnare rettangoli, archi, scrivere testo, impostare il colore degli elementi, conoscere i colori disponibili. Diventa così necessario che ogni Canvas abbia al suo interno il metodo `paint()`. Nel nostro caso all'interno della funzione verranno semplicemente caricate su display delle immagini. Anche se all'apparenza questo metodo può sembrare banale, viste le capacità della classe, il risultato finale è di alto effetto visivo. Esempio di caricamento di una immagine:

```
Image image; // oggetto image
public void paint(Graphics g) {
```

L'immagine viene ricercata nella cartella `res` del programma e viene assegnata all'oggetto `Image`, successivamente viene disegnata tramite il metodo `drawImage` dell'oggetto `g`, i metodi `getWidth() / 2` e `getHeight() / 2` fanno in modo che l'immagine venga ridimensionata a seconda delle dimensioni dello schermo. I parametri, `g.HCENTER` e `g.VCENTER`, fanno in modo che l'immagine sia caricata al centro dello schermo.

```

    image = Image.createImage ("/connessione.png");
    g.drawImage (image, getWidth () / 2, getHeight () / 2, g.HCENTER | g.VCENTER);
}

```

Alcuni esempi di possibili metodi utilizzabili a parametri fissi:

```

//Sul display viene disegnato un rettangolo di base 20 e altezza 30 a partire
dalle coordinate x,y 220 220
g.drawRect (20, 30, 200, 220);
//Sul display viene disegnata una linea da x,y di partenza 10 10 e arrivo x,y 10
10
g.drawLine(0, 10, 10, 10);

```

una documentazione più approfondita è reperibile nella documentazione relativa all'oggetto Graphics contenuta nel pacchetto di programmazione WTK22.

Intercettare pressione dei tasti da parte dell'utente

La classe Canvas, oltre ad essere utilizzata per la grafica, possiede dei metodi che permettono di intercettare le azioni dell'utente sui tasti del dispositivo. Possiamo di fatto capire quali siano i pulsanti premuti ed associare a questi determinate azioni. I metodi keyPressed() e keyReleased() ci segnalano rispettivamente quando viene premuto e quando viene rilasciato un determinato pulsante. Questi due metodi hanno come parametro un intero (keyCode) corrispondente al codice ASCII del pulsante premuto.

Esempio di codice:

```

int tasto;

protected void keyPressed(int keyCode) {
    //Appena viene premuto un tasto il metodo keyPressed salva il rispettivo codice
    ascii nella variabile tasto
    tasto=keyCode;
    repaint();
} //end keyPressed

//L'azione associata ad ogni pulsante e poi eseguita nel metodo paint. Ciò è possibile
attraverso la funzione repaint e ad un if che controlla il valore della variabile tasto
if (tasto=='4'){

    //invio dati
    blue.invio("4");
    //Visualizzazione immagine
    //g.drawString("premuto1", 80, 40, g.TOP|g.LEFT);
    try {
        image = Image.createImage ("/sinistra.png");
    } catch (IOException e) {
        throw new RuntimeException ("Unable to load Image: "+e);
    }
    g.drawImage (image, getWidth () / 2, getHeight () / 2, g.HCENTER | g.VCENTER);
}

```

Gestione della Canvas nella applicazione Blutis

Una volta instaurata la connessione con il dispositivo tramite la funzione connessione(){} si va a richiamare la classe canvas tramite il seguente codice:

```

canvas = new MyCanvas();//creo l'oggetto Mycanvas
form_send=new Form("");
display.setCurrent(form_send);
canvas.setMidlet(this);//passo alla canvas la classe Blutis1
display.setCurrent(canvas);

```

La nostra Canvas non fa altro che attendere la pressione di un tasto da parte dell'utente e quindi visualizzare a schermo la rispettiva immagine associata a quel

tasto e passare alla funzione invio(), della classe Blutis1, il valore del tasto premuto. Di seguito viene riportato un esempio di codice:

```
if (tasto=='2'){
//invio dati
blue.invio("2");
//Visualizzazione immagine
//g.drawString("premuto1", 80, 40, g.TOP|g.LEFT);
try {
    image = Image.createImage ("/avanti.png");
} catch (IOException e) {
    throw new RuntimeException ("Unable to load Image: "+e);
}
g.drawImage (image, getWidth () / 2, getHeight () / 2, g.HCENTER | g.VCENTER);
}
```



Comunicazione Dispositivo-Cellulare Lato Server software PBASIC

Il software eseguito dalla Basic Stamp è suddiviso principalmente in 3 sezioni:

- Configurazione, dove vengono inviati i comandi di configurazione ai vari moduli hardware
- Inizializzazione, nel quale vengono inizializzate le variabili flag ed alcuni valori vengono resettati al loro default
- Controllo Remoto, preceduta dall'attesa di connessione Bluetooth da parte di una periferica esterna

Non avendo alcuna possibilità di intercettare Interrupt o eventi, abbiamo optato per un flusso ciclico delle operazioni: viene effettuato un polling sui vari pin e linee logiche leggendo il loro stato. A seconda della loro configurazione il software andrà ad eseguire un certo blocco di codice.



PSEUDOCODIFICA

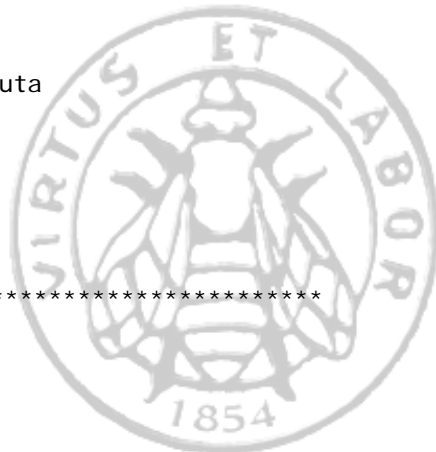
```
' ***** MAIN *****
Mai n:

Conf:
' ---- CONFIGURAZIONE
    motConfig:
    Reset Linea Seriale Controller Motori
    Reset Controller Motori
    Configurazione Controller Motori : 1 Motore
    Reset Controller Motori
' ---- ENDCONFIGURAZIONE

Ini t:
' ---- INIZIALIZZAZIONE
    Iniziali zzazione Veloci tà Motori : Massima
    Stato Iniziale della BlutisCar : ALT
    Funzione di Centrazione sterzo BlutisCar
    motPc = 0
' ---- ENDINIZIALIZZAZIONE

Cycl e:
' ---- LOOP CONTROLLO REMOTO
    IF (Status Connessione BT : Connesso) THEN
        motD = Lettura Buffer BT EB500 (Timeout di 400 ms : NoDataReceived)
        Cycl eC:
            cli Detect:
                FreqOut ir
                irF = Lettura Stato irFrontale
                IF irF = 1 THEN
                    Funzione di Retromarcia Temporizzata Autonoma
                ENDIF
                bafP = Lettura Stato Posteri ore
                IF bafP = 1 THEN
                    Funzione di Avanzamento Temporizzato Autonoma
                ENDIF
            sel Azi oni :
            SELECT motD
            CASE "2"
                Funzione di Avanzamento
            CASE "4"
                IF motPc <> 4 THEN
                    Funzione Sterzata a Si ni stra
                ENDIF
            CASE "1"
                Di mi ni uzi one Veloci tà
            CASE "3"
                Aumento veloci tà
            CASE "6"
                IF motPc <> 6 THEN
                    Funzione Sterzata a Destra
                ENDIF
            CASE "8"
                Funzione di Retromarcia
            CASE "5"
                Funzione Freno
            CASE "7"
                Funzione di Ri levazi one Temperatura
                Funzione di Ri levazi one Lumi nosi tà
            CASE "9"
                Funzione Cl acson
            ENDSELECT
            Sal vataggi o Ul ti ma Azi one Ri cevuta
        ELSE
            Funzi one di segnal azi one Ti meout
        ENDIF
        GOTO Cycl e
' ---- ENDLOOP

' ***** FUNZIONI *****
NoDataRecei ved:
' Copia ul ti ma azi one ri cevuta
motD = motDO
RETURN
```



```

Funzione di Avanzamento:
    ' Invio comando al controller motori via seriale
    SEROUT motC, motB, [$80,0,3,motS]
RETURN

Funzione di Retromarcia:
    ' Invio comando al controller motori via seriale
    SEROUT motC, motB, [$80,0,2,motS]
RETURN

Funzione Freno:
    ' Invio comando al controller motori via seriale
    SEROUT motC, motB, [$80,0,2,0]
RETURN

Funzione di Retromarcia Temporizzata Autonoma (motT millisecondi):
    Funzione Freno
    Funzione di Retromarcia
    Pausa predefinita di esecuzione
    Funzione Freno
RETURN

Funzione di Avanzamento Temporizzato Autonoma (motT millisecondi):
    Funzione Freno
    Funzione di Avanzamento
    Pausa predefinita di esecuzione
    Funzione Freno
RETURN

Funzione Sterzata a Sinistra:
    SELECT motPc
    CASE 0
        FOR i = 0 TO 30
            Invio impulsi di 1.0 ms al motore passo-passo
            Attesa 30 ms
        NEXT
        motPc = 4
    CASE 6
        FOR i = 0 TO 30
            Invio impulsi di 1.5 ms al motore passo-passo
            Attesa 30 ms
        NEXT
        motPc = 0
    ENDSELECT
RETURN

Funzione Sterzata a Destra:
    SELECT motPc
    CASE 0
        FOR i = 0 TO 30
            Invio impulsi di 2.0 ms al motore passo-passo
            Attesa 30 ms
        NEXT
        motPc = 6
    CASE 4
        FOR i = 0 TO 30
            Invio impulsi di 1.5 ms al motore passo-passo
            Attesa 30 ms
        NEXT
        motPc = 0
    ENDSELECT
RETURN

Funzione di Rilevazione Temperatura:
    riI = Lettura Termoresistenza
    IF riI > 30 THEN
        Funzione Surrialdamento:
    ENDIF
RETURN

Funzione di Rilevazione Luminosità:
    riIV = Lettura Fotoresistenza
    IF riIV > 100 THEN
        Accendi Luci
    ELSE
        Spegni Luci
    ENDIF
RETURN

```



```
Funzione Clacson:  
  Attiva PiezoSpeaker  
RETURN
```

```
Funzione di segnalazione Timeout:  
  Funzione Freno  
  Funzione Clacson  
  Attesa 1 secondo  
  Funzione Freno  
GOTO Init
```

```
Funzione di Centrazione sterzo BlutisCar:  
  FOR i = 0 TO 30  
    Invio impulsi di 1.5 ms al motore passo-passo  
    Attesa 30 ms  
  NEXT  
RETURN
```

```
Funzione Surri riscaldamento:  
  Funzione Freno  
  FOR hti = 0 TO 15  
    Accendi Luci  
    Attesa 500 ms  
    Spegni Luci  
    Attesa 500 ms  
  NEXT  
GOTO Init
```



Per effettuare operazioni come invio e ricezione su linea seriale, lettura di un fotoresistore o termoresistenza, abbiamo sfruttato alcune funzioni del set di istruzioni PBASIC:

- SEROUT e SERIN, invio e ricezione pacchetti dati su linea seriale
- RCTIME, lettura tempo di carica o scarica di condensatori
- PULSOUT, invio su una linea di un impulso ad onda quadra di periodo T

SEROUT e SERIN

La SEROUT e SERIN hanno una sintassi del tipo:

```
SEROUT Tpin {\Fpin}, Baudmode, {Pace,} {Timeout, Tlabel,} [OutputData]
SERIN Rpin {\Fpin}, Baudmode, {Plabel,} {Timeout, Tlabel,} [InputData]
```

Come possiamo vedere, la sintassi ha molti parametri e questa funzione è molto flessibile: a seconda del tipo di processore abbiamo un range di Baudmode e Tpin utilizzabili.

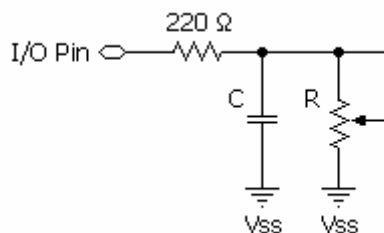
Analizziamo ora singolarmente i parametri della SEROUT: Tpin è il pin di dove sarà effettuato l'invio o ricezione seriale, Fpin è il pin opzionale di flow control (per controllo di flusso avanzato, specialmente per l'utilizzo di trasmissione dati seriali da e verso pc), Baudmode è la velocità di trasmissione, Pace è opzionale ed è il tempo di pausa tra i singoli bytes, Timeout e Tlabel sono 2 parametri opzionali ed indicano rispettivamente i ms di timeout e l'etichetta della funzione da eseguire in caso di timeout, OutputData è l'indirizzo della variabile in cui memorizzare i dati.

RCTIME

La RCTIME ha una sintassi del tipo:

```
RCTIME Pin, State, Variable
```

Dove Pin è il pin di I/O da analizzare, State è la tensione del pin di I/O con cui si andrà ad analizzare il condensatore (1 corrisponde a 5V, 0 corrisponde a 0V), Variable è la variabile dove verrà memorizzato il risultato della RCTIME.



Con un circuito come quello sovrastante, dove R è la fotoresistenza o la termoresistenza, utilizzando:

```
RCTIME ritTP, 1, riIV
```

Dove ritTP (è il pin di I/O) e riIV è il valore di ritorno della RCTIME, possiamo sapere il tempo di carica del condensatore e con opportuni calcoli, ottenere la luminosità o la temperatura.

PULSOUT

La sua sintassi è del tipo:

```
PULSOUT Pin, Duration
```



Dove Pin è il Pin di I/O da utilizzare e Duration è il periodo espresso in unità (ad ogni unità, a seconda del tipo di processore utilizzato, corrisponde un valore in μs) dell'impulso quadro.

Questa funzione è stata particolarmente utile per il controllo del servomotore per lo sterzo.

Infatti il servomotore in nostro possesso ha una possibilità di movimento compresa tra 0° e 180° .

Inviando:

- Impulsi di 1.5 ms di T il motore si posiziona a 90°
- Impulsi di 1.0 ms di T il motore si posiziona a 0°
- Impulsi di 2.0 ms di T il motore si posiziona a 180°

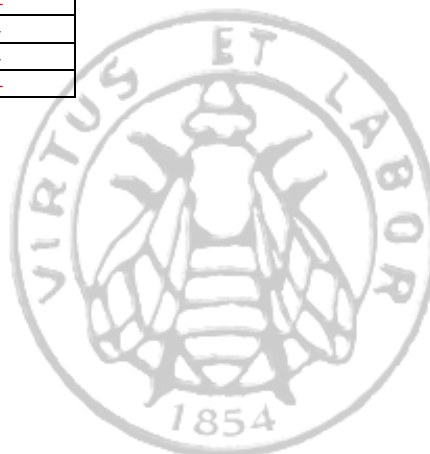
Table e Pinatura

Tabella Pololu

Byte "Motor # and Direction"		Byte "Motor Speed"		Funzione
B7	B6÷B1	B0	Byte	
0	000001	1	n	Motore 1 Avanti
0	000001	0	n	Motore 1 Indietro
0	000001	1	0	Motore 1 Frizione
0	000001	0	0	Motore 1 Frena

Tabella Pin

PIN	Funzione
BoardOfEducation	
0	Data output BT
1	Data input BT
2	Collisione Posteriore
3	Infrarossi Out
4	Infrarossi In
5	Connection status
6	BT Mode switcher
7	Attivazione Luci
8	BT Reset
9	BT Reset
10	Seriale motori
11	Reset motori
12	Sensore temperatura
13	Sensore luminosità
14	Motore Sterzo
15	Attivazione Luci
Pololu Controller	
1	Motor Supply (9V)
2	Ground (0V)
3	Logic Supply (5V)
4	Seriale motori
5	Reset motori
6	Motore 0 +
7	Motore 0 -
8	Motore 0 -
9	Motore 0 +



Costruzione – Struttura della Blutis Car

La costruzione della Blutis Car è avvenuta in più passaggi. Dopo un primo abbozzo, il modellino è stato completamente ricostruito assumendo una forma più simile ad una comune vettura.

Estetica e mobilità del veicolo

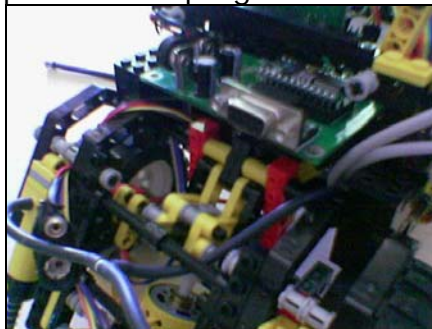
La struttura del veicolo è stata interamente realizzata utilizzando costruzioni LEGO Technic (TM) seguendo un disegno di nuova creazione, che garantisca l'ospitalità a tutti i componenti elettronici. L'unico pezzo estraneo è costituito dall'asse delle ruote posteriori realizzato in metallo a causa dell'eccessiva flessibilità delle barre di plastica della Lego. Nel procedimento costruttivo si sono adottate tecniche per garantire la compattezza del veicolo e la resistenza agli urti. A tale scopo la resistenza dei paraurti, sia posteriore che anteriore, è stata valutata con dei crash test. Il veicolo presenta una gabbia idraulica (caricata ogni qual volta il veicolo si trovi a collidere posteriormente) che, una volta chiusa, permette di riparare la BasicStamp e l'antenna da eventuali urti dovuti al capovolgimento del veicolo. Da una prima analisi di progetto che prevedeva l'impiego di due motori al fine di interagire in maniera indipendente sulla velocità delle gomme posteriori e quindi consentire la sterzata, si è passato a motori separati per la gestione del movimento e dello sterzo (ora posto anteriormente). A causa del sistema di trasmissione utilizzato (vite senza fine) non è stato possibile impiegare delle sospensioni posteriori, tecnologia invece largamente utilizzata nella realizzazione dei paraurti. Il grip delle gomme e la trasmissione della Blutis car ne garantiscono un arresto istantaneo in caso di frenata.

Approccio alla realizzazione

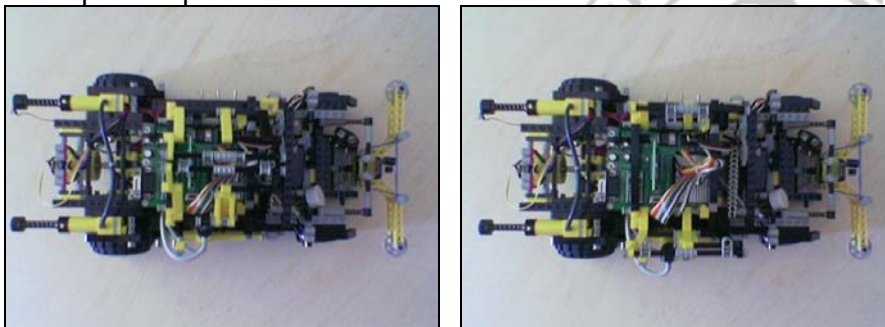
Il procedimento adottato nella realizzazione è stato di tipo modulare. Dopo aver preso confidenza con gli strumenti di lavoro si è proceduto a realizzare piccoli moduli indipendenti che col crescere delle competenze e all'avanzare del lavoro venivano assemblate fino ad essere quello che è il progetto "Blutis".

Da notare in particolare gli accorgimenti:

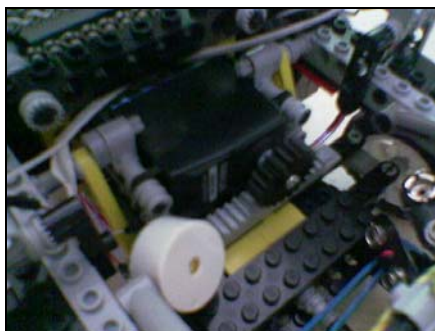
- nella parte posteriore per la facile programmazione della Board of Education



- il "tettino" apribile per l'estrazione dell'antenna Bluetooth



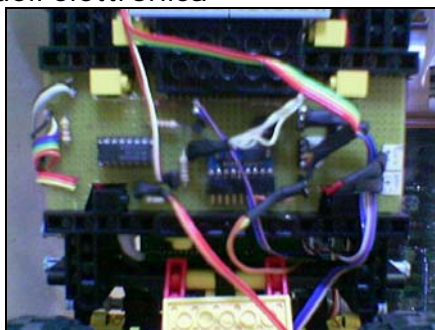
- l'impianto sterzante



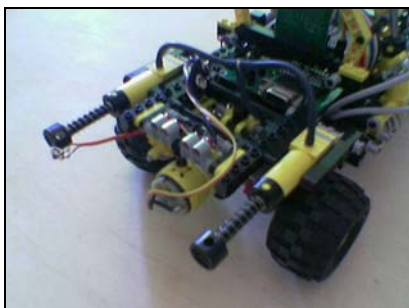
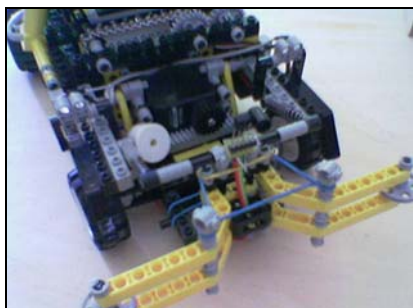
- la trasmissione all'asse posteriore



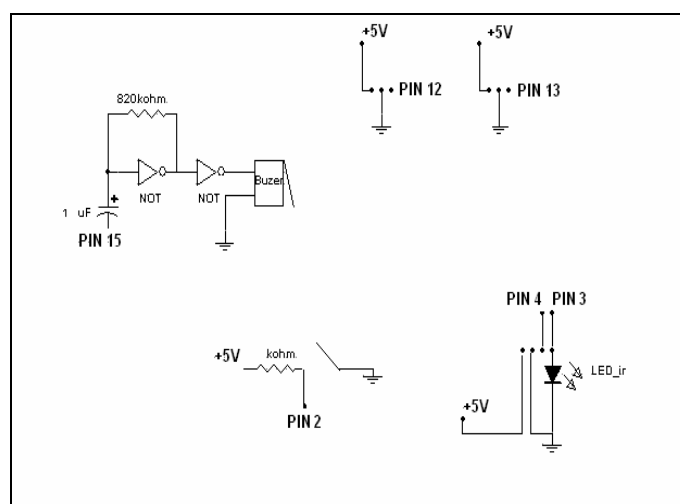
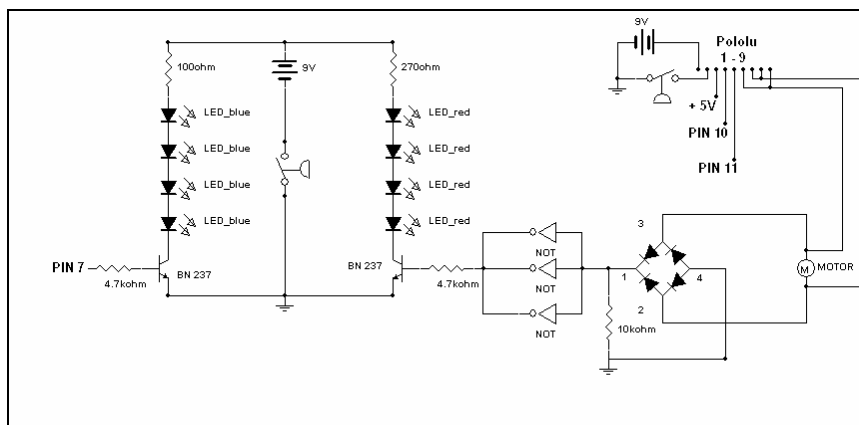
- il vano per la scheda dell'elettronica



Alcune foto panoramiche della Blutis Car



Costruzione - Elettronica



La scheda di connessione

Per praticità è stata realizzata una scheda millefori (opportunamente sagomata al fine di adattarsi al pannello di fondo della macchina) connessa mediante un bus a tutti i pin della BasicStamp (esclusi quelli relativi all'antenna) dalla quale vengono prelevate le masse e i 5V. Tutti i componenti (ad esclusione dell'antenna e del motore passo passo impiegato per sterzare) sono collegati alla millefori mediante connettori verticali. Un risultato migliore si sarebbe ottenuto impiegando una scheda stampata e connessioni laterali alla scheda di connessione da parte del bus, dei sensori, dei motori etc. Gli interruttori presenti sul lato destro del veicolo consentono da sinistra verso destra l'alimentazione della BasicStamp, dei motori, e delle luci in maniera indipendente dato che vengono utilizzate alimentazioni separate. Per motivi di sicurezza azionare gli interruttori solo quando l'interruttore montato sulla BasicStamp è in posizione ON cioè posizione 2.

Il Buzzer (Clacson)

Il buzzer è collegato al pin 15 (oltre che alla massa), che opportunamente comandato consente l'emissione di suoni monofrequenza.

Il servo motore

Il motore passo passo è collegato direttamente al pin 14 dal quale riceve i comandi oltre all'alimentazione che coincide con quella della BasicStamp.

I sensori di luce e temperatura

Per la realizzazione di tali sensori sono state impiegate fotoresistenze (resistenza il cui valore varia al variare della luce), e termoresistenze (resistenze il cui valore varia al variare della temperatura). Per semplicità di esposizione ci si riferirà alla costruzione del solo sensore di temperatura essendo gli schemi elettrici pressochè identici. Il sensore fa parte di un circuito RC nel ruolo di resistenza. Una lettura del tempo di carica dal pin al quale è connesso il circuito (mediante un'altra resistenza che funge da protezione) e piccoli calcoli, consentono di conoscere la temperatura (o la quantità di luce). Mentre il sensore di luce influirà sull'accensione dei fari anteriori, il sensore di temperatura eviterà il surriscaldamento del motore (il tutto opportunamente gestito dal software Pbasic). La miniaturizzazione del sensore è stata possibile grazie all'impiego di una scheda millefori e costruzioni Lego opportunamente modificate.

L'accensione dei fari posteriori (Stop)

L'accensione degli stop è strettamente legata al funzionamento del motore in un qualsiasi istante. Per non impegnare il processore della BasicStamp in un dispendioso polling dello stato motore, infatti, si è utilizzato un circuito elettronico autonomo. Un ponte di diodi in parallelo al motore consente di raddrizzare la corrente che vi scorre rendendo influente che il veicolo proceda a marcia avanti o in retromarcia. La resistenza all'uscita del ponte (carico generico) è necessaria al fine di rendere possibile il funzionamento dello stesso. La corrente così raddrizzata viene convogliata all'ingresso di tre porte NOT (il parallelo è necessario per problemi di scarsa corrente), che non fanno altro che negare la tensione che hanno in ingresso. Quindi se il motore è fermo avremo una tensione di circa 9V corrispondente all'alimentazione dell'integrato, mentre se è in movimento una tensione nulla. L'uscita delle NOT è utilizzata per pilotare un transistor. Il transistor è una sorta di rubinetto elettronico che ci permette di controllare grandi correnti utilizzandone di piccole. Nel caso in questione infatti è possibile utilizzare l'uscita delle NOT per consentire il passaggio ad una corrente maggiore che andrà a fluire all'interno dei diodi rossi, facendoli illuminare (transistor saturo). Logicamente se l'uscita delle NOT è nulla (poichè il motore è in funzione), il transistor impedirà il passaggio della grande corrente (transito interdetto).

L'accensione dei fari anteriori

Il funzionamento è simile a quello degli stop con la differenza che l'abilitazione del transistor proviene da un pin della BasicStamp opportunamente gestito in caso di scarsa luminosità e richiesta dell'utente di accendere i fari.

Il sensore di contatto posteriore

Come è possibile vedere dallo schema elettrico in posizione normale dell'interruttore posto nella parte posteriore del veicolo una lettura del pin 2 corrisponderà al prelievo di 5V, mentre ad interruttore chiuso di 0V. Tali letture opportunamente gestite dal software Pbasic, consentono di reagire agli urti posteriori.

Il sensore IR

Il sensore all'infrarosso presente nella parte anteriore del veicolo permette di rilevare collisioni allo stesso modo del sensore posteriore, ma evita il contatto. Il funzionamento è basato sull'emissione di raggi infrarossi ad una determinata frequenza da parte di un led collegato al pin 3 e dalla successiva lettura di un detector di raggi infrarossi, collegato al pin 4. Nel caso non sia presente nessun ostacolo davanti ai sensori, il detector non rileva nulla, in caso fosse presente un ostacolo i raggi infrarossi verrebbero riflessi e il detector li rileverebbe. La distanza di funzionamento, nei limiti hardware dei sensori, dipendono dalla frequenza con la quale sono emessi gli infrarossi. Il cambio di luce ed ostacoli di colore scuro potrebbero ingannare il detector. L'impiego di sensori ad ultrasuoni eviterebbe l'inconveniente, ma il costo sarebbe più elevato.

Il motore

Il motore impiegato per il movimento del veicolo opera in corrente continua. L'inversione di polarità ai suoi capi (per gestire il verso di rotazione) e la tensione alla quale è sottoposto (per gestire la velocità) sono gestite dal programma Pbasic con ausilio di un microcontroller per motori in continua. Il condensatore in parallelo al motore serve per evitare che parte della corrente rimasta nell'avvolgimento una volta arrestato il motore possano fluire verso il microcontroller e danneggiarlo. Il condensatore inoltre è utile in fase di avviamento poichè migliora lo spunto. La piedinatura del microcontroller e la sua connessione è visibile nella documentazione proprietaria. La connessione a coppie dei pin 6-9 e 7-8 è necessaria per sfruttare al meglio l'amperaggio delle batterie dato che connesso in tal maniera il microcontroller può fornire al massimo 2A.

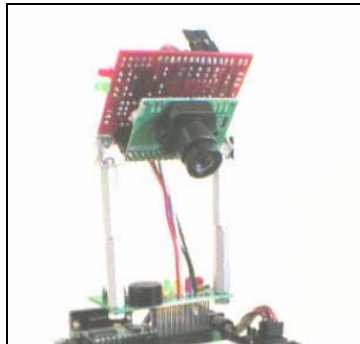


Conclusioni – Possibili miglioramenti

Una volta presa dimestichezza con gli strumenti di lavoro, solo tempo e denaro costituiscono un freno a futuri sviluppi. Il fattore denaro può essere aggirato, utilizzando materiale di recupero, filosofia che in parte è stata adottata nella realizzazione del progetto, ma ciò porta via tempo a causa della scarsa reperibilità di datasheet appropriati.

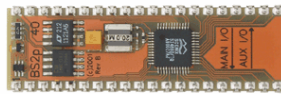
Tralasciando questi aspetti verranno analizzate in breve future implementazioni per la "Blutis car".

1. Dotare la blutis car di un sensore CCD miniaturizzato (telecamera), che periodicamente campioni le immagini (se il campionamento fosse rapido si avrebbe l'idea di un video in diretta) e le invii al cellulare sfruttando la connessione bluetooth già presente. L'utilizzo di una telecamera wireless già pronta risultava essere poco didattico.



2. Realizzare un sito web che tramite Socket interagisca con un computer remoto e permetta il controllo del veicolo e la visualizzazione delle immagini in streaming.
3. Aggiungere al veicolo un display LCD che funga da targa. Il display dovrebbe essere configurato dinamicamente all'accensione del veicolo con il MAC dell'antenna installata in quel momento. Il problema dell'implementazione potrebbe essere costituito dalla scarsa disponibilità di pin sulla BasicStamp, nel caso si volesse utilizzare un display LCD con interfaccia parallela (i più comuni).
4. Con piccole modifiche ai software coinvolti nel progetto è possibile consentire all'utente di registrare movimenti consecutivi della blutis car, e, in caso di necessità, comandare il veicolo affinché li riproduca autonomamente. Nell'eventuale realizzazione tener presente della scarsa quantità di memoria l'interno del veicolo, e quindi della necessità di evitare memorizzazioni inutili, come comandi successivi uguali. Per calcolare il tempo entro il quale ciascuna azione viene eseguita è necessario dotare il software di un timer (anche un semplice contatore all'interno del ciclo principale) o prevedere il montaggio di un encoder che interagisca con le ruote del veicolo e ne calcoli la velocità e lo spazio percorso.
5. Realizzare un garage elettronico che preveda l'apertura del cancello all'approssimarsi del veicolo, e il sollevamento del veicolo stesso una volta introdotto all'interno, la ricarica delle batterie qualora necessario, e l'eventuale connessione alla porta seriale per la riprogrammazione dell'unità mobile. Si potrebbe dotare il programma presente all'interno della BasicStamp di una funzione che una volta rilevata l'apertura del cancello del garage, chieda all'utente l'autorizzazione per parcheggiare autonomamente.
6. Modificare il programma Pbasic che gestisce le funzionalità del veicolo affinché ci sia parallelismo effettivo tra le esecuzioni di alcune funzioni, come la routine

- di rilevamento collisioni e l'accensione dello speaker o magari il movimento. A tal scopo si potrebbe prevedere l'aggiunta di un'altra Basic stamp che interagisce con la prima tramite infrarosso, porta seriale o linee fisiche.
7. Aggiungere una ventola termocontrollata che permetta il raffreddamento dell'elettronica e del motore.
 8. Dotare il veicolo di un modulo "line follower", qualora si volesse rendere completamente autonomo il movimento del veicolo e fargli seguire un percorso preparato.
 9. Realizzare un altro veicolo che interagisce con la Blutis car.
 10. Visualizzare sul display del cellulare lo stato di carica delle batterie del veicolo.
 11. Utilizzo di una Board of Education più avanzata e del processore BS2P, che aggiunge la possibilità di intercettare interrupt e dispone di una maggiore potenza di calcolo, oltre a possedere 40 I/O



Conclusioni – Cronologia delle conoscenze

- Rilevazione da seriale di ostacoli via infrarosso
- Controllo di temperatura e luminosità da seriale
- Pilotaggio motori in continua nelle due direzioni di marcia e controllo della velocità di ogni singolo motore per sterzare
- Realizzazione di una prima struttura del veicolo con gabbia manuale
- Realizzato un primo programma per lo scambio di dati via bluetooth (tra cellulari)
- Introduzione della BasicStamp nel veicolo
- Abbozzo di una prima struttura del software Pbasic e comando via tastiera delle diverse funzioni finora realizzate
- Realizzazione di una secondo struttura del veicolo e implemenazione gabbia idraulica
- Sostituzione dell'infrarosso con baffi paraurti dotati di sensori di contatto e aggiornamenti software Pbasic
- Aggiunta degli interruttori metallici per le diverse alimentazioni
- Realizzazione asse metallico ruote posteriore e realizzazione di un primo sistema di trasmissione a catena
- Sostituzione del motore e modifica del sistema di trasmissione mediante vite senza fine
- Modifica dell'impianto l'impianto sterzante con l'introduzione di un motore passo-passo a causa della lunghezza del veicolo e aggiornamento software Pbasic
- Il software J2ME è quasi terminato
- Inviati i primi caratteri da cellulare a BasicStamp
- Controllo remoto tramite cellulare dell'impianto sterzante
- Aggiunta alla struttura dei fari anteriori e posteriori
- Aggiunta di una prima millefori e connessione di tutti i dispositivi elettronici e aggiornamento software Pbasic
- Ripristino del sensore infrarosso e modifica dei baffi a puro paraurti e realizzazione di una seconda millefori con impiego di fili morbidi
- Ottimizzazione del programma di gestione remota J2ME e miglioramento interfaccia
- Controllo remoto di tutte le funzioni del veicolo e miglorie software Pbasic
- Miglorie nell'impianto sterzante



Conclusioni – Ringraziamenti

Si ringrazia il consiglio di classe, la gentile e spontanea collaborazione del prof. Cesare Peticari, Orlando, Vincezo, Lorenzo.

Grazie anche a Giulio Violoni, nostro compagno di classe, per i loghi dell'ITI.
Esternamente all'ambito didattico, ringraziamo Dario Marconi e Belluccini Emiliano.



Conclusioni – Link e Documentazione Proprietaria

Per approfondimenti è disponibile la documentazione proprietaria di:

- Pololu, PDF, Controller Motori
- A7Eng, PDF, Bluetooth
- PBASIC, Help Microsoft, Linguaggio PBASIC
- J2ME, HTML (<http://java.sun.com/j2me/docs/>), Linguaggio J2ME e JAVA

I sorgenti sono nel CD, nella cartella Sorgenti.

