Robot Pololu Zumo 32U4 Roadmap (aggiornato al 27/1/2019)

Giorgio De Nunzio
https://github.com/giorgio-denunzio/Zumo32U4/

Caratteristiche del robot e link utili

https://www.pololu.com/category/170/zumo-32u4-robot
https://www.pololu.com/docs/0J63 User's Guide
http://pololu.github.io/zumo-32u4-arduino-library/index.html
https://www.pololu.com/docs/pdf/0J63/zumo_32u4_robot.pdf Manuale dell'utente
https://www.youtube.com/results?search_query=zumo+32u4 Ricerca su youtube
www.youtube.com/watch?v=Jtwu7-T1a9o Demo
https://www.youtube.com/watch?v=3UC54XeATCE Minisumo (first tests, against box)
https://www.youtube.com/watch?v=QCqxOzKNFks Minisumo competition
https://www.youtube.com/watch?v=pvV1-vChVyk Presentazione
https://www.youtube.com/watch?v=rthMiqFCiBA Strategies
https://www.youtube.com/watch?v=GIv4jWB2158 Minisumo competition
https://www.youtube.com/watch?v=HMefw5vYu7I RC Control
https://www.youtube.com/watch?v=PV8sqnfDYVU Robot Vision (Pixy Cam)
Professor Craven (https://github.com/pvcraven/zumo_32u4_examples):
- https://www.youtube.com/watch?v=7KgZUn8ATDQ Zumo 32U4 LED Tutorial
- https://www.youtube.com/watch?v=ddPo6HQvxzQ Zumo Proximity Sensor
- https://www.youtube.com/watch?v=k2DDa9nwVx0 Running the motors with the Zumo robot
- https://www.youtube.com/watch?v=XOp22Xx7ZnU Turn/Gyro Sensor Example for Pololu Zumo 32U4

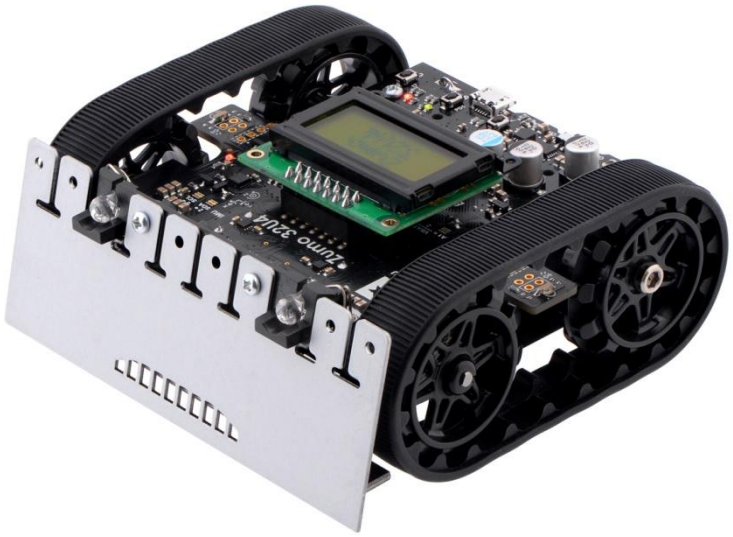

Figura 1: robot Zumo 32U4.

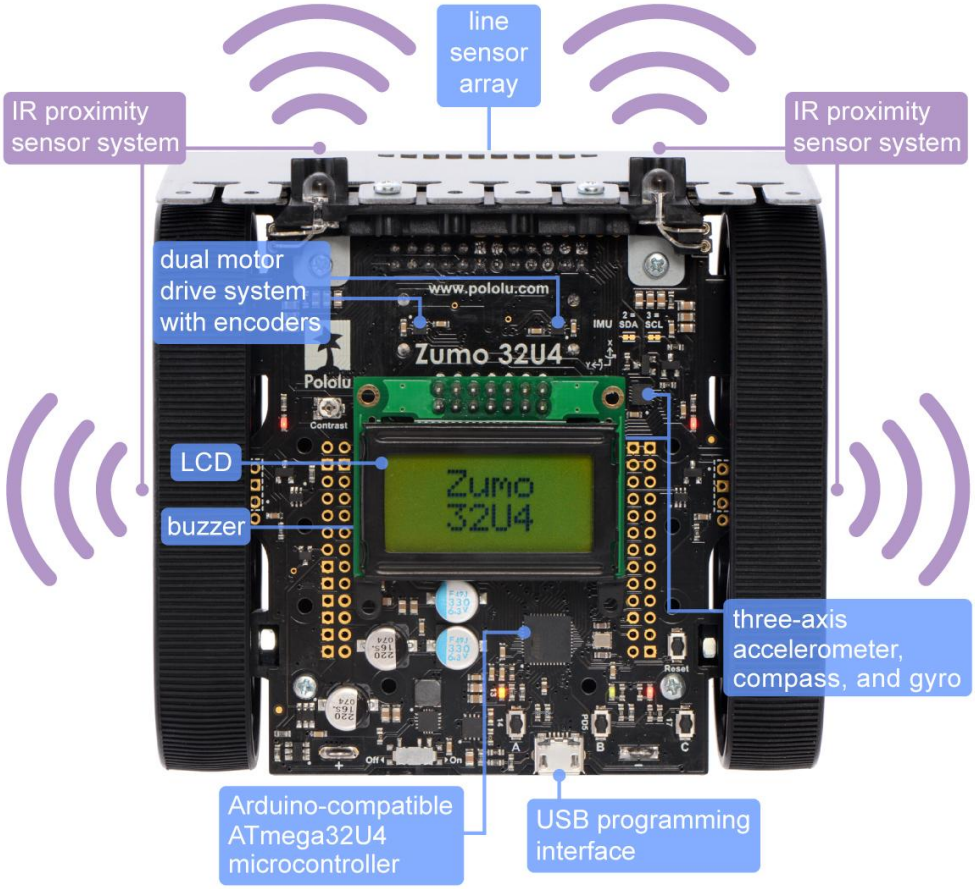

Figura 2: robot Zumo 32U4: caratteristiche principali.

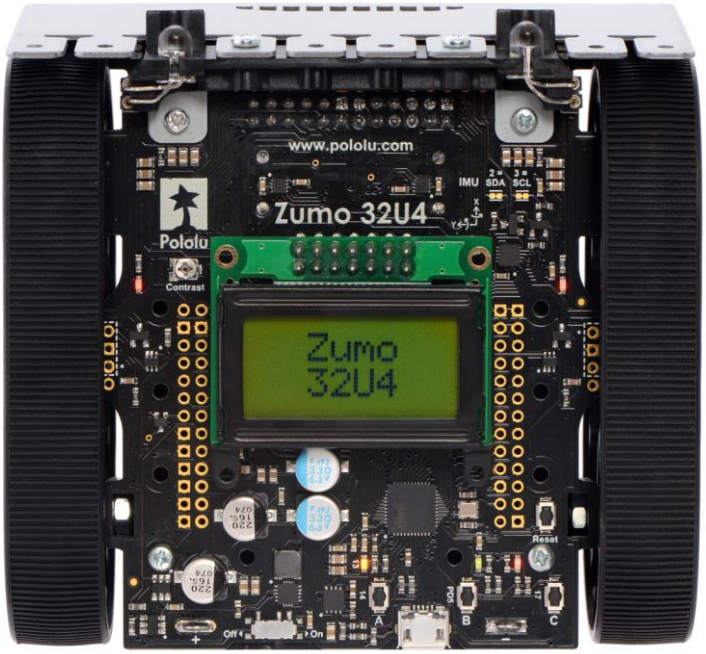

Figura 3: robot Zumo 32U4 visto dall'alto.

Figura 4: robot Zumo 32U4 visto dal basso.

1) Installazione Arduino IDE (*Integrated Development Environment*): scaricare e installare, ove non presente sul pc in uso, il software di programmazione di arduino da https://www.arduino.cc/en/Main/Software, in particolare nella sezione "Download the Arduino IDE" (versione al 27/1/2019: 1.8.8).

2) Installazione Robot Pololu 32U4: occorre installare la libreria per l'Arduino IDE (file include e altro) e i driver del device (oltre alle indicazioni sulla board A*32U4 e altro)
   a. Le indicazioni per installare la libreria sono su Internet all'indirizzo *https://github.com/pololu/zumo-32u4-arduino-library*
   Non occorre scaricare il file da github, ma solo seguire le istruzioni, riportate di seguito:
   Installing the library
   If you are using version 1.6.2 or later of the Arduino software (IDE), you can use the Library Manager to install this library:
   In the Arduino IDE, open the "Sketch" menu, select "Include Library", then "Manage Libraries...".
   Search for "Zumo32U4".
   Click the Zumo32U4 entry in the list.
   Click "Install".
   b. Le indicazioni per installare driver, board etc per la scheda A* sono su https://github.com/pololu/a-star
   Si riportano le istruzioni da seguire:
   Arduino IDE integration
   These files can be used to add A-Star and robot support to the Arduino IDE. Entries for Pololu A-Star Boards will appear in the Boards menu when you do this.
   For most people, we recommend setting up A-Star support in the Arduino IDE by installing our boards

package through the Boards Manager. (These instructions can also be found in the user's guide for each controller).
1. In the Arduino IDE, open the File menu (Windows/Linux) or the Arduino menu (macOS) and select "Preferences".
2. In the Preferences dialog, find the "Additional Boards Manager URLs" text box. Copy and paste the following URL into this box:
https://files.pololu.com/arduino/package_pololu_index.json
If there are already other URLs in the box, you can either add this one separated by a comma or click the button next to the box to open an input dialog where you can add the URL on a new line.
3. Click the "OK" button to close the Preferences dialog.
4. In the Tools > Board menu, select "Boards Manager..." (at the top of the menu).
5. In the Boards Manager dialog, search for "Pololu A-Star Boards".
6. Select the "Pololu A-Star Boards" entry in the list, and click the "Install" button.

Occorre ora scaricare il file dei driver da github (link diretto: https://github.com/pololu/a-star/archive/master.zip). Decompattare il file, scaricato, a-star-master.zip, aprire la cartella a-star-master\drivers e, come da istruzioni contenute nel file INSTALL.txt, "Right click on 'a-star.inf' and select 'Install'." seguendo poi l'installazione.

Si raccomanda di usare una versione recente di Arduino IDE! Con la 1.8.3, ad esempio, si sono verificati problemi.

Si raccomanda, inoltre, ove si verifichino – durante le operazioni di compilazione e upload – errori di upload del file compilato, di riprovare: spesso l'errore sparisce e l'upload va a buon fine.

3) Lettura del **livello della batteria**.

Come primo test, utile anche per monitorare nei giorni del Corso lo scaricamento del pacco batterie del robot, leggiamo il livello di carica delle batterie. Tenere conto che il valore atteso a batterie completamente cariche (quattro pile ricaricabili nichel-metallo idruro, NiMh, da 1,2V ciascuna) è di circa 4*1.2V = 4.8V.

Partiamo dalle informazioni reperibili nella documentazione.
A pag. 13 del manuale utente (vedere il link alla fine di questa dispensa) si legge: "*Two blue power LEDs under the rear corners of the main board indicate when the robot is receiving power from batteries (the power switch must be turned on). The left LED is connected to the reverse protected and switched battery voltage (VBAT), while the right LED is connected to the output of the main board's 5 V regulator.* ***The left blue LED will become noticeably dimmer as the total battery voltage drops below about 3 V****, and this can serve as an indication that a set of alkaline batteries has reached the end of its useful life. However, rechargeable batteries can be damaged by overdischarge, so we do not recommend allowing a set of four NiMH cells to discharge to this point. (A voltage divider is connected to analog pin 1 and can be used to monitor the battery voltage; see Section 3.8 for details.)*"
Section 3.8, Power, pag. 22: *"...The battery voltage on VBAT can be monitored through a voltage divider that is connected to analog pin 1 (PF6) by default.* ***The divider outputs a voltage that is equal to one half of the battery voltage****, which will be safely below the ATmega32U4's maximum analog input voltage of 5 V as long as the battery voltage is less than 10 V. The* ***readBatteryMillivolts()*** *function in the Zumo32U4 library [https://www.pololu.com/docs/0J63/6] can be used to determine the battery voltage from this reading. The surface-mount jumper labeled "A1 = VBAT/2" can be cut to disconnect the voltage divider and free the pin for other uses*."

Al link:
http://pololu.github.io/zumo-32u4-arduino-library/_zumo32_u4_8h.html#a9391e187045f8f5a48546b34b6c6db25
c'è il prototipo della funzione: `uint16_t readBatteryMillivolts()`.

Nella IDE di Arduino aprire un nuovo file e scrivere il codice necessario per leggere il valore VBAT mediante questa funzione, ed esprimerlo in V anziché in mV, mostrandolo su LCD.

Ecco il codice:

```
#include <Wire.h>
#include <Zumo32U4.h>
Zumo32U4LCD lcd;
void setup() {
}
void loop() {
  uint16_t VBAT = readBatteryMillivolts();
  float VBATVolts = VBAT/1000.;  // What happens if I use 1000 instead of 1000.
                                 // as the fraction denominator? :-) Why?
  lcd.clear();
  lcd.print(VBATVolts);
  delay(500);
}
```

Salvare il codice con un nome opportuno in una directory scelta per lavorare sui robot. Un nome potrebbe essere: ReadBatteriesZumo32U4.
Notare che la struttura è analoga a quella adoperata per Arduino: esso è composto da due funzioni, `setup()` e `loop()`.
Esaminare il codice e riconoscerne le diverse parti (funzioni `setup()` e `loop()`, chiamata alla funzione di libreria `readBatteryMillivolts()` di lettura del valore del livello di batteria, uso del visualizzatore a cristalli liquidi lcd…).

Collegare il robot tramite un cavetto Micro USB, dopo essersi assicurati che il robot sia munito di pile cariche (aprire con attenzione lo sportellino del robot che permette di accedere all'alloggiamento delle pile, esaminarle e riconoscerne il tipo e la capacità) e che l'interruttore di accensione (Figura 3, in basso a sinistra) sia in posizione OFF. Verificare che la voce di menu Strumenti→Scheda selezioni "Pololu A-Star 32U4", e che Strumenti→Porta indichi la porta USB effettivamente connessa al robot.

Compilare e caricare il codice nel robot. Dopo il caricamento, scollegare con delicatezza il robot dal pc, e accendere l'interruttore. Il display mostrerà il valore del livello di carica delle batterie.

Esercizio. Misurare ogni giorno il livello della batteria prima di iniziare a lavorare sul robot, e alla fine della giornata di lavoro; porre via via su un grafico tali valori. Quanto velocemente si stanno scaricando le batterie? Tengono la carica quando non sono utilizzate (ossia da un giorno a fine lavoro, fino alla mattina dopo)? Importante: **quando VBAT scende a circa 3 V occorre ricaricare le pile**! Avvisare in anticipo il Docente in modo da essere pronti al cambio.

4) Eseguire **BlinkLEDs**

Nella IDE di Arduino, selezionare File→Esempi→Zumo32U4→BlinkLEDs e compilare il codice, di seguito riportato (qui lievemente compresso eliminando alcuni spazi e righe vuote). Analizziamo il codice.

```
/* This example shows how to blink the three user LEDs on the Zumo 32U4. */
#include <Wire.h>
#include <Zumo32U4.h>
void setup()
{
}
void loop()
{
  // Turn the LEDs on.
  ledRed(1);
  ledYellow(1);
  ledGreen(1);
  // Wait for a second.
  delay(1000);
  // Turn the LEDs off.
  ledRed(0);
  ledYellow(0);
  ledGreen(0);
  // Wait for a second.
  delay(1000);
}
```

Individuare le istruzioni di accensione e spegnimento dei led.

Esercizi. Dopo aver studiato e capito il codice, svolgere i seguenti esercizi:

a) Intervenire sulle istruzioni delay() per ottenere un periodo di accensione/spegnimento di 1s anziché 2s, con il medesimo *duty cycle* del 50% (cioè, periodo totale di 1s, led accesi per 500 ms e spenti per 500 ms). Se il concetto di *duty cycle* non è ben chiaro, rivedere/approfondire (per esempio, https://it.wikipedia.org/wiki/Duty_cycle) .

b) Intervenire sulle istruzioni delay() per avere un periodo di accensione/spegnimento di 1s anziché 2s, con il *duty cycle* del 25% (cioè, il periodo totale è pari a 1s, ma i led sono accesi per 250 ms e spenti per 750 ms).

c) Nel periodo "on", sia acceso solo il led rosso e siano spenti gli altri; viceversa nel periodo "off". Periodo pari a 2s e *duty cycle* pari a 33%.

5) Eseguire **BuzzerBasics**.

Leggere il commento iniziale per comprenderne la finalità. Informazioni sulla classe Zumo32U4Buzzer sono reperibili al link http://pololu.github.io/zumo-32u4-arduino-library/class_zumo32_u4_buzzer.html.

In particolare, prestare attenzione alle istruzioni seguenti, comprendendone il funzionamento:

```
buzzer.playFrequency(440, 200, 15);
```

e

```
// Start playing note A in octave 4 at maximum volume
// volume (15) for 2000 milliseconds.
  buzzer.playNote(NOTE_A(4), 2000, 15);
// Wait for 200 ms and stop playing note.
  delay(200);
  buzzer.stopPlaying();
```

Esercizio. Inserire opportunamente nel programma le righe seguenti e… ascoltare il risultato, previa compilazione e caricamento nel robot:

```
// play a C major scale up and back down:
  buzzer.play("!L16 V8 cdefgab>cbagfedc");
  while (buzzer.isPlaying());
// the first few measures of Bach's fugue in D-minor
  buzzer.play("!T240 L8 agafaea dac+adaea fa<aa<bac#a dac#adaea f4");
```

6) Eseguire **FaceTowardsOpponent**. Leggere il commento iniziale per comprenderne la finalità. Notare innanzitutto quali sono i file `#include` adoperati; poi fare attenzione alla definizione degli oggetti che si useranno nel codice, descritti nelle righe:

```
Zumo32U4LCD lcd;
Zumo32U4Motors motors;
Zumo32U4ProximitySensors proxSensors;
Zumo32U4ButtonA buttonA;
```

Uso del display a cristalli liquidi (LCD)

Le istruzioni riferite all'uso del display sono:

```
Zumo32U4LCD lcd;
lcd.clear();
lcd.gotoXY(0, 0);  e  lcd.gotoXY(0, 1);
lcd.print(......);
```

Informazioni sulla classe Zumo32U4LCD sono reperibili al link http://pololu.github.io/zumo-32u4-arduino-library/class_zumo32_u4_l_c_d.html. La classe eredita dalla classe print di Arduino, descritta qui: https://www.arduino.cc/en/Serial/Print. Si vedrà che nel programma la stringa passata al display LCD è preceduta da una F: lcd.print(F("Press A")); La ragione è che questa stringa è contenuta in flash memory, ossia la memoria di programma, anziché nella memoria delle variabili (che è scarsa!): ad esempio, si vede al link https://www.arduino.cc/en/Tutorial/Memory la disponibilità di memoria per l'ATmega328, il processore montato su Arduino Uno:

Flash 32k bytes (of which .5k is used for the bootloader)
SRAM 2k bytes
EEPROM 1k byte

Vedere anche al link http://playground.arduino.cc/Learning/Memory una discussione più approfondita sull'argomento, dove è anche spiegata la sintassi "F()". Altro link utile: https://www.arduino.cc/reference/en/language/variables/utilities/progmem/.

Uso dei motori

Individuare le istruzioni che si occupano del pilotaggio dei motori. Sono:

```
Zumo32U4Motors motors;
motors.setSpeeds(turnSpeed, -turnSpeed);
```

dove il primo parametro di `setSpeeds` è la velocità del motore di sinistra e il secondo quella di destra (vedere il link http://pololu.github.io/zumo-32u4-arduino-library/class_zumo32_u4_motors.html)

Notare la divisione del codice in funzioni. In particolare studiare `turnRight()`, `turnLeft()`, `stop()` (con le `motors.setSpeeds()`).

Notare che – a differenza di quanto avveniva quando, nel robot ROBi-Wan-Kenobi, si pilotavano direttamente i motori scrivendo nei registri del driver L298 – in questo caso le velocità inserite nell'istruzione:

```
motors.setSpeeds();
```

sono positive se vogliamo che il motore concorra a fare andare il robot avanti, negative se vogliamo concorra a far andare il robot indietro: di conseguenza, speed è positivo per entrambi se il robot avanza, negativo per entrambi se il robot arretra. Per far ruotare il robot, occorre inserire due valori di segno opposto per i motori destro e sinistro.

Il **valore massimo di velocità** utilizzabile è 400. Mantenersi, nelle prove, su un **valore medio pari a 200** (sia per controllare meglio il robot, sia per consumare meno le batterie).

Per comprendere meglio l'uso dei motori, e imparare un'istruzione alternativa per impostarne la velocità, riferirsi all'esempio MotorTest (trattato al punto successivo di questa dispensa).

Uso dei sensori di prossimità.

Zumo 32U4 è dotato di tre sensori di prossimità (FRONT, RIGHT, LEFT) e cinque sensori di linea (per individuare una linea tracciata sul pavimento). Non tutti gli otto sensori sono attivi contemporaneamente e occorre impostare un jumper per scegliere cosa sia connesso.

A pagine 18 del manuale dell'utente si legge: *The assembled versions of the Zumo 32U4 robot ship with jumpers selecting the left (LFT) and right (RGT) proximity sensors instead of down-facing DN2 and DN4, so these versions are configured for* **three down-facing sensors and all three proximity sensors by default**.

A pag. 19: "*The brightness of the emitters can be controlled by adjusting the duty cycle of the PWM signal on pin 5. Our example code operates the proximity sensors by transmitting pulses on both the left and right LEDs at six different brightness levels. For each sensor, it generates two numbers: the number of brightness levels for the left LEDs that activated the sensor, and the number of brightness levels for the right LEDs that activated the sensor. A higher reading corresponds to more IR light getting reflected to the sensor, which is influenced by the size, reflectivity, proximity, and location of nearby objects. However, the presence of other sources of 38 kHz IR pulses (e.g. from another robot) can also affect the readings.*"

Il funzionamento dei sensori di prossimità avviene dunque con un PWM (come il controllo della velocità dei motori in corrente continua): un *duty cycle* alto accende "più a lungo" i LED, che quindi restano mediamente più luminosi e quindi possono illuminare e individuare ostacoli lontani; un *duty cycle* basso al contrario lascia i LED mediamente meno luminosi e quindi serve solo per individuare gli ostacoli vicini.

L'esempio FaceTowardsOpponent utilizza un solo sensore IR frontale per individuare gli ostacoli, e due LED di illuminazione (a 38KHz) uno situato a destra e uno a sinistra sul frontale del robot. Studiare nel codice l'uso del sensore frontale illuminato alternativamente dai LED destro e sinistro, a partire dalla dichiarazione di un oggetto globale, con:

```
Zumo32U4ProximitySensors proxSensors;
```

e con le istruzioni all'interno della funzione `loop()`:

```
proxSensors.initFrontSensor();
proxSensors.read();
proxSensors.countsFrontWithLeftLeds();
proxSensors.countsFrontWithRightLeds();
```

Queste ultime restituiscono valori a 8 bit, come interi "senza segno" (positivi): uint8_t.

In alternativa all'uso di un solo sensore centrale, illuminato a turno dai due LED, è possibile utilizzare tutti e tre i sensori di prossimità: frontale, destro e sinistro, inizializzando con `proxSensors.initThreeSensors()` e poi usando le opportune istruzioni count….

I dettagli sono al link:
http://pololu.github.io/zumo-32u4-arduino-library/class_zumo32_u4_proximity_sensors.html.
L'istruzione `constrain()` è descritta al riferimento seguente:
https://www.arduino.cc/reference/en/language/functions/math/constrain/

Uso del pulsante "A"
Prestare attenzione all'istruzione che permette all'utente di far partire il programma solo quando si preme il pulsante 'A': `buttonA.waitForButton();` E' molto utile per evitare di dover inserire ritardi alla partenza (per dare il tempo all'utente di portare il robot sul campo in cui deve muoversi). Potrebbe, quest'istruzione, essere messa nella loop()?

Esercizi.

a) Due variabili nel codice sono definite ma poi di fatto non usate (ossia, sono dichiarate, è assegnato un valore, ma poi non sono usate altrove all'interno del codice): quali? Semplificare il codice eliminandole e omettendo le istruzioni che fanno loro riferimento. Verificare che il codice continui a compilare e a girare come previsto.

b) Per sperimentare le sole istruzioni relative al sensore frontale di prossimità, senza uso dei motori, scrivere un codice in un nuovo file chiamato myProximityTest, nel quale siano presenti:
   - gli `#include` necessari (ispirandosi a FaceTowardsOpponent)
   - le dichiarazioni degli oggetti globali buttonA, lcd, proxSensors (come presenti in FaceTowardsOpponent)
   - una funzione setup() che si preoccupi di scrivere su LCD la frase "Press A", come visto in FaceTowardsOpponent e attenda che sia premuto il pulsante A; poi inizializzi il sensore frontale di prossimità come visto in FaceTowardsOpponent.
   - Una funzione loop() che richiami le funzioni read, countsFrontWithLeftLeds e countsFrontWithRightLeds del sensore di prossimità, e scriva sul display i due valori letti, sulle due righe disponibili (usare lcd.gotoXY()).

   Scollegare il robot, e accenderlo. Avvicinare e allontanare un ostacolo al robot, frontalmente e di lato, per vedere il risultato sul display.

c) Il codice FaceTowardsOpponent è abbastanza complesso, perché contiene le istruzioni necessarie per muovere i motori con accelerazione/decelerazione e in modo "smooth". Basandosi su quanto appreso con il precedente esercizio 6b, relativamente all'uso del sensore di prossimità, e sul pilotaggio dei motori come realizzato in FaceTowardsOpponent, scrivere un codice, in un nuovo file chiamato myFaceTowardsOpponent, che faccia ruotare il robot su se stesso fino a che esso non veda un ostacolo di fronte a sé. A quel punto il robot si fermi e suoni un motivetto con il buzzer. Nel codice siano presenti:

- gli `#include` necessari
- le dichiarazioni degli oggetti motors, buttonA, lcd, proxSensors, buzzer
- una funzione setup() che si preoccupi di scrivere su LCD la frase “Press A”, come visto in FaceTowardsOpponent, e attenda che sia premuto il pulsante A; poi inizializzi il sensore frontale di prossimità come visto in FaceTowardsOpponent.
- Una funzione loop() che definisca innanzitutto due variabili (di valore costante) denominate `speed` e `threshold`: la prima dia la velocità di rotazione dei motori (per esempio 200) e la seconda indichi la soglia al di sopra della quale decidiamo che il sensore di prossimità abbia visto un ostacolo. In seguito, la loop() richiami le funzioni read, countsFrontWithLeftLeds e countsFrontWithRightLeds del sensore di prossimità, assegnando i valori letti alle due variabili leftValue e rightValue, e scrivendoli sul display, sulle due righe disponibili (come nell’esercizio 6b). Poi controlli se: (i) nessuno dei valori letti sia sopra soglia (non c’è ostacolo), (ii) entrambi superino la soglia e siano uguali (ostacolo davanti al robot, vicino), (iii) almeno uno superi la soglia e sia strettamente maggiore dell’altro; nel primo caso, il robot ruoti in uno dei versi (destro o sinistro) arbitrariamente; nel secondo caso, il robot si fermi, suoni una nota o un motivo, vada verso l’ostacolo per 200 ms per spingerlo, e torni indietro; nel terzo caso, il robot ruoti verso l’ostacolo.

7) Eseguire **MotorTest**. Studiare il sorgente. Individuare le istruzioni (anche aiutandosi con il codice FaceTowardsOpponent), che servono per il movimento dei motori:

```
Zumo32U4Motors motors;
motors.setRightSpeed(speed);
motors.setLeftSpeed(speed);
```

Le istruzioni `setRightSpeed` e `setLeftSpeed` sono un’alternativa alla `setSpeeds()` vista precedentemente, e alterano solo la velocità di uno dei motori.

<u>Esercizi</u>

a) Scrivere un codice, in un nuovo file chiamato myMotorTest, nel quale siano presenti:
   - gli `#include` necessari
   - le dichiarazioni degli oggetti motors, buttonA, lcd (come presenti negli altri sorgenti precedente esaminati)
   - una funzione setup() che si preoccupi di scrivere su LCD la frase “Press A”, come visto in FaceTowardsOpponent, e poi chiami una funzione motorTest() che sarà tra poco descritta.
   - Una funzione loop() vuota.
   - Cinque funzioni: mForward(), mBackward(), mStop(), mRotateLeft90(), mRotateRight90(); le prime due facciano muovere i motori in modo che il robot vada avanti indefinitamente, o indietro indefinitamente (valori positivi o negativi di speed, pari ad esempio a 200); la terza fermi i motori (inserendo il valore di velocità 0); la quarta e la quinta inseriscano i valori di velocità opportuni per ruotare a destra e a sinistra (quindi, segni alterni), seguiti da un delay calcolato in modo da avere una rotazione di 90°, e poi fermino i motori (richiamando la stop()).
   - Una funzione motorTest() che richiami le cinque funzioni definite al punto precedente, in modo che il robot: vada avanti per un secondo, si fermi per un secondo, giri a destra

di 90° e attenda un secondo, giri a sinistra di 180° e si fermi un secondo, giri nuovamente a destra di 90° e si fermi un secondo, torni indietro e si fermi definitivamente.

8) **Affinamento dell'accuratezza nella rilevazione degli ostacol**i

Eseguendo myProximityTest si noterà che, soprattutto a brevi distanze, l'accuratezza della rilevazione degli ostacoli non è elevata. Il valore restituito sul display, per ostacoli frontali abbastanza vicini (sotto 20 cm circa) è sempre uguale al valore massimo consentito (pari a 6, per *default*). Per aumentare l'accuratezza e rendere la misura più fine, si può ricorrere al trucco di ridefinire i livelli di luminosità dei LED illuminatori del sistema sensore frontale. Per farlo, aggiungere al codice (nella setup()) le seguenti righe:

```
uint16_t myBrightnessLevels[] =
       {1,2,3,4,5,6,7,8,9,15,23,32,42,55,70,85,100,120,130,140,150,160,170,180,190};
        // default is { 4, 15, 32, 55, 85, 120 }
int numOfBrightnessLevels = sizeof(myBrightnessLevels)/sizeof(myBrightnessLevels[0]);
proxSensors.setBrightnessLevels(myBrightnessLevels, numOfBrightnessLevels);
```

e provare a rieseguire.

9) **Evitare gli ostacoli**

...

10) **Line sensors**

Sezione 3.5 del manual dell'utente: "*The board features **five line sensors** and three proximity sensors, though by default, you can only have six of these eight sensors connected to the Zumo's microcontroller at any given time*." "*The five line sensors face downward and can help the Zumo distinguish between light and dark surfaces. They can also be used to detect sources of infrared light, like the sun. Each reflectance sensor consists of a down-facing infrared (IR) emitter LED paired with a phototransistor that can detect reflected infrared light from the LED. ...*"

Nella configurazione di **default**, **tre sensori di linea sono attivi**.

Diversi esempi di uso dei sensori di linea si trovano in LineSensorTest (che usa però 5 sensori di linea, quindi non nella configurazione di default del robot che consente l'uso di tre sensori), LineAndProximitySensors (nella "configuration 1" o nella "configuration 5" si usano tre sensori di linea, quindi come di default; nelle altre se ne usano di più), BorderDetect (tre sensori), LineFollower (cinque sensori, piuttosto complesso perché usa un algoritmo PID per il controllo del robot) e MazeSolver (cinque sensori). Fare riferimento agli esempi che lavorano con solo tre sensori, per esempio a **BorderDetect**.

Le istruzioni che si occupano della gestione dei sensori di linea sono:

```
Zumo32U4LineSensors lineSensors;
```

che definisce l'oggetto della classe Zumo32U4LineSensors,

```
#define NUM_SENSORS 3
unsigned int lineSensorValues[NUM_SENSORS];
```

che definisce un vettore di interi senza segno, avente tre celle, che conterrà le letture dai tre sensori; una dichiarazione equivalente sarebbe (senza definire il valore costante NUM_SENSORS):

```
unsigned int lineSensorValues[3];

lineSensors.initThreeSensors();
```

che inizializza i sensori;

```
lineSensors.read(lineSensorValues);
```

che effettua la lettura, e infine le istruzioni che usano i valori letti, come:

```
if (lineSensorValues[NUM_SENSORS - 1] < QTR_THRESHOLD)
```

Il vettore `lineSensorValues`, dopo l'istruzione `lineSensors.read`, contiene i tre valori letti dai sensori: `lineSensorValues[0]` contiene il valore letto dal sensore di sinistra, `lineSensorValues[1]` contiene il valore letto dal sensore centrale, e infine `lineSensorValues[2]` contiene il valore letto dal sensore di destra.
Se necessario, rivedere il concetto di vettore (array) in C.

Informazioni dettagliate sull'uso delle funzioni di libreria per i sensori di linea sono all'indirizzo: http://pololu.github.io/zumo-32u4-arduino-library/class_zumo32_u4_line_sensors.html.

Esercizi

a) Lettura dei valori di tre sensori di linea: scrivere un programma (ReadLineSensors) che legga i tre valori e li scriva sul display LCD in modo che la prima riga contenga il valore del sensore di sinistra, uno spazio e il valore del sensore di destra, mentre la seconda riga contenga, un poco sfalsato rispetto all'inizio, il valore del sensore centrale, secondo la falsariga seguente:

```
500 450
  490
```

   Purtroppo lo spazio disponibile sul display non sempre permetterà di avere letture complete sulla prima riga!

b) Partendo da quanto fatto nell'esercizio precedente, scrivere un programma (DontFallDown) in base al quale il robot, poggiato su un tavolo, avanzi (previa pressione del pulsante A) e, se giunge sul bordo del tavolo, non cada, ma si fermi per mezzo secondo, torni indietro per un secondo, ruoti di un angolo approssimativamente definito, e riprenda ad avanzare. Per tarare il sistema, effettuare manualmente due letture dei sensori (usando lo stesso programma dell'esercizio 10a), una con il robot sul tavolo e una fuori dal tavolo, e si usino tali letture per individuare una soglia da usare per capire se il robot è fuori dal bordo. Quando i sensori comunicano un valore maggiore, sul tavolo o fuori da esso?
   In un ulteriore affinamento, effettuare la taratura automaticamente; la calibrazione deve funzionare come segue: si pone il robot sul tavolo e si preme il pulsante A, permettendogli di fare una prima lettura; poi lo si pone fuori dal tavolo e si preme nuovamente il tasto A memorizzando una seconda lettura; a questo punto il valore di soglia che distingue le due posizioni (sul tavolo / fuori dal tavolo) è calcolato automaticamente (con una media dei valori, per esempio), e il robot calibrato è utilizzabile come detto in precedenza.

c) Border detection. Il programma scritto nel corso dell'esercizio precedente è anche atto – con minime variazioni dovute alla necessità di invertire la logica chiaro/scuro - a rendere il

robot in grado di avanzare fintantoché non incontri il bordo del ring del minisumo. Non avendo a disposizione un dohyō, è possibile simularne una parte tramite del nastro adesivo bianco su un fondo scuro, oppure – forse più facilmente - invertire la convenzione e usare uno sfondo (pavimento) chiaro, con del nastro adesivo/isolante nero che riproduca il bordo del ring. In quest'ultimo caso (sfondo chiaro e nastro scuro) il programma DontFallDown (magari rinominato myBorderDetect per permettere future specifiche varianti) funzionerà immediatamente, senza nemmeno invertire la logica chiaro/scuro.

d) Minisumo versione 0.1. Mettendo insieme quanto visto nell'esercizio 6c e 10c, realizzare un software – denominato Minisumo0.1 – che faccia girare su se stesso il robot (posto all'interno del ring) in cerca di un avversario (che per ora immaginiamo fermo, per esempio la scatola di imballaggio del robot); quando lo individua, cerchi di spingerlo fuori dal ring, ma non superi il bordo, realizzato con nastro isolante nero.

11) **Giroscopio e Accelerometro**. Pag. 3 del manuale: "*The robot also features a variety of sensors, including … inertial sensors (accelerometer and gyro) on the main board*". Leggere il paragrafo "3.7. Inertial sensors".

Caricare ed eseguire il programma di esempio InertialSensors.ino. Lasciare il robot connesso attraverso la porta USB, e usare il monitor seriale per vedere i valori della lettura di Accelerometro, Magnetometro e Giroscopio (A, M, G) variare via via che si sposta il robot manualmente. Non è necessario accendere il robot.

Per la sintassi PSTR() utilizzata con le istruzioni del gruppo printf() per porre la stringa in flash memory, vedere http://playground.arduino.cc/Main/Printf (e la discussione fatta al punto 5 su FaceTowardsOpponent, relativa a F(), utilizzato con Serial.println()).

Ora realizziamo un programma di esempio che renda utili e leggibili le misure fatte dal **giroscopio**. Le righe seguenti riassumono quanto si può desumere dal codice di MazeSolver (e GridMovement.cpp, da esso richiamato).

Nell'interfaccia di programmazione di Arduino, aprire un nuovo file, da salvare chiamandolo TestGyroZumo32U4. Individuare la cartella automaticamente creata, avente lo stesso nome (dovrebbe essere in Documenti, sottodirectory Arduino). In essa, copiare i file TurnSensor.cpp e TurnSensor.h reperiti nella directory: Arduino\libraries\Zumo32U4\examples\MazeSolver, sottodirectory della cartella Documenti. Inserire ora, nel file TestGyroZumo32U4, il codice seguente:

```
#include <Wire.h>
#include <Zumo32U4.h>
#include "TurnSensor.h"
Zumo32U4LCD lcd;
Zumo32U4ButtonA buttonA;
L3G gyro;
void setup() {
  lcd.clear();
  lcd.gotoXY(0,0);
  lcd.print(F("Place robot"));
  lcd.gotoXY(0,1);
  lcd.print(F("Press A"));
  buttonA.waitForButton();
  // Calibrate the gyro. While the LCD
  // is displaying "Gyro cal", you should be careful to hold the robot
  // still.
  turnSensorSetup();
// This should be called to set the starting point for measuring
// a turn.  After calling this, turnAngle will be 0.
```

```
  turnSensorReset();
}
void loop() {
  // Update the angle value (contained in turnAngle, defined in TurnSensor.cpp)
  turnSensorUpdate();
  lcd.clear();
  lcd.print(turnAngle);
}
```

12) **Rimozione del display LCD e uso dei pin resi liberi.** Da "Pololu Zumo 32U4 Robot User's Guide", "3.11. Adding electronics" – "Freeing up I/O pins", pag. 29:

If you do not need the LCD, you can remove it. This frees up pin 0 (PD2) and pin 1 (PD3). These pins are the transmit (TX) and receive (RX) pins of the UART, so you can use them to establish serial communication with another microcontroller. These pins are also capable of digital I/O. These pins are the recommended pins for connecting two output channels from an RC receiver, or for controlling two RC servos, because they are arranged in a convenient way with respect to power and ground on the right-side expansion header.

If you have removed the LCD and do not need to use button A, this frees up pin 14 (PB3). Pin 14 is capable of digital input and output.

Removing the LCD frees up the LCD contrast potentiometer for other purposes. The output of the potentiometer is a 0 V to 5 V signal which is accessible on the LCD connector. It can be connected to any free analog input if you want to read it from the AVR, or it might be useful to connect it to the other electronics that you are adding.

L'header (connettore) relativo al display LCD, accessibile sfilando con cautela il display, è riportato alla pagina https://www.pololu.com/docs/0J63/all#3.9, da cui è anche possibile scaricare il file zumo-32u4-pinout.pdf, ed è reperibile all'indirizzo https://www.pololu.com/picture/view/0J6286.
Lo schema (parziale e ridotto alla parte di interesse) è indicato in Figura 5 (sinistra, e destra in alto).

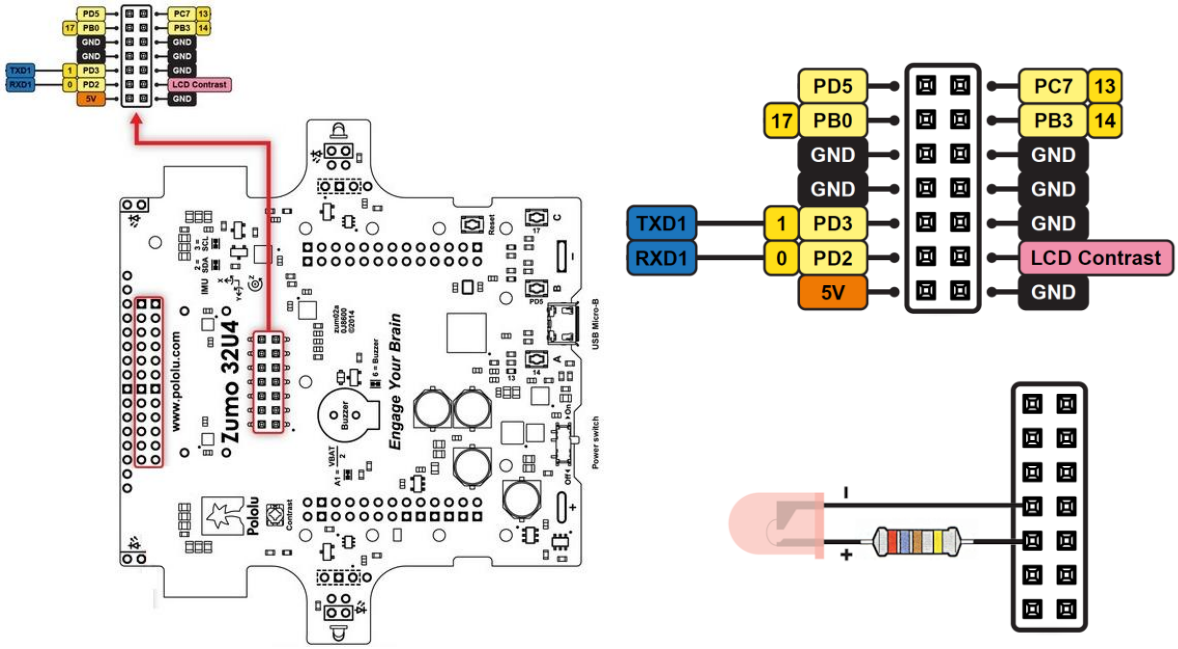

Figura 5: Sinistra, individuazione del connettore del display LCD; destra in alto, il connettore ingrandito; destra in basso, collegamento di un LED tra pin 1 e massa.

E’ possibile connettere un LED e accenderlo, mediante lo schema in Figura 5 (in basso a destra). Il codice da usare è:

```
//#include <Wire.h>
//#include <Zumo32U4.h>
void setup() {
  pinMode(1, OUTPUT);
}
void loop() {
  digitalWrite(1, HIGH);
  delay(500);
  digitalWrite(1, LOW);
  delay(500);
}
```

dove gli include sono stati commentati per far notare che, come intuitivo, per un programma così semplice essi sono ridondanti, perché non è utilizzata nessuna funzione specifica di Zumo.

13) Comunicazione Zumo <-> Arduino, mediante seriale.

There are a total of three wires between the two arduinos.
1) TX to RX (transmit pin to recieve pin)
2) RX to TX (recieve pin to transmit pin)
3) Ground to Ground (Common ground between the two arduinos)

Speed and start and stop bits must match on both CPUs.
For the (receiver) arduino you use an if statement to check if there is something to read:

```
 if (Serial.available() > 0) {
    incomingByte = Serial.read();
}
```

then you can reference "incomingByte" elsewhere in the code.

http://www.zonnepanelen.wouterlood.com/arduino-bare-basics/arduino-passing-temperature-readings-to-a-neighbor-arduino-three-ways-of-serial-communication/

https://medium.com/iotguider/serial-communication-between-two-arduino-boards-3b7c646dd8ab
(manca la Serial.available! Su alcuni siti ho visto che la Serial.available è usata anche al momento della trasmissione)
https://arduino.stackexchange.com/questions/45527/arduino-serial-communication-between-2-uno
ok

https://iotguider.in/arduino/serial-communication-between-two-arduino-boards/
https://robotic-controls.com/learn/arduino/arduino-arduino-serial-communication

I2C, software.
https://www.arduino.cc/en/Tutorial/MasterWriter
https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/
https://www.eevblog.com/forum/projects/arduino-uno-simultaneous-multiple-soft-i2c-library/
https://arduino.stackexchange.com/questions/13594/software-i2c-atmega8
https://www.arduinolibraries.info/libraries/soft-wire
https://todbot.com/blog/2010/09/25/softi2cmaster-add-i2c-to-any-arduino-pins/
http://wiki.seeedstudio.com/Arduino_Software_I2C_user_guide/
https://github.com/Testato/SoftwareWire/wiki/Arduino-I2C-libraries
https://github.com/todbot/SoftI2CMaster
https://github.com/felias-fogg/SoftI2CMaster
https://playground.arduino.cc/Main/SoftwareI2CLibrary

Fare caso che e' sempre master mode, quindi occorre mettere master il robot e slave con vera i2c arduino

14) **Comunicazione tra robot**: pag 19 del manuale dell'utente: "*You can also just read the proximity sensors without turning on any LEDs. This could allow the Zumo to detect the IR proximity sensors of other robots, or to detect commands from a typical IR remote.*"