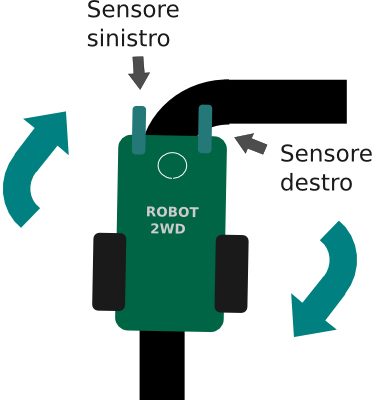
INSEGUITORI DI LINEA E ARDUINO

<https://elettronicaemaker.it/2017/12/28/inseguitori-di-linea-e-arduino/>

[Arduino & Co.](https://elettronicaemaker.it/category/arduino/" \o "View all posts in Arduino & Co.) [E&M-1-2018](https://elettronicaemaker.it/category/em-1-2018/)

[0](https://elettronicaemaker.it/2017/12/28/inseguitori-di-linea-e-arduino/#respond)

[](https://elettronicaemaker.it/wp-content/uploads/2017/12/08-Destra-copy.jpg)

Share it !

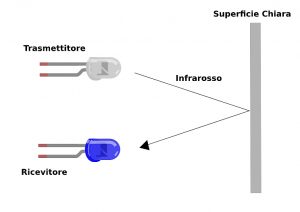
di Emanuele Paiano

Introduzione

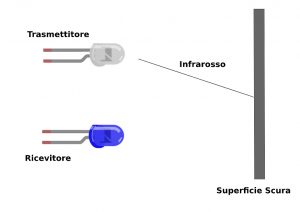
L’impiego dei sensori, in elettronica e robotica, è piuttosto vasto: si va dal semplice termostato per gli impianti di  
riscaldamento, fino all’impiego di sonar e sensori infrarossi per il rilevamento di ostacoli e superfici. Grazie ai sensori, è possibile misurare le proprietà fisiche dell’ambiente, permettendo loro di conoscerne lo stato ed eventualmente modificarlo tramite gli attuatori. In questo articolo studieremo il funzionamento dei sensori ad infrarossi implementati nel modulo KY-033 (o inseguitore di linea), che permette ad Arduino di distinguere le superfici chiare da quelle scure. Vedremo un’applicazione pratica in cui un semplice robot insegue una traccia nera stampata su una superficie bianca (come può essere un cartoncino o un pannello di legno), oltre che gestire curve,  
incroci e svolte ad angolo retto.

Il sensore di traccia: caratteristiche e test

Il modulo KY-033 (noto anche come line tracking sensor) è un sensore che permettere di rilevare il passaggio da una superficie chiara ad una scura. È composto da un comparatore di segnali, un potenziometro per regolarne la sensibilità, e due fotodiodi: il primo fotodiodo è un trasmettitore di raggi infrarossi, mentre il secondo è un ricevitore. Il principio di funzionamento è basato sulla tendenza degli infrarossi ad essere assorbiti dalle superfici scure: tale funzionamento consiste nell’inviare un segnale dal trasmettitore e attenderne l’eventuale ricezione sul  
ricevitore, in questo modo possiamo distinguere (a grandi linee) il colore bianco della superficie, da quello nero. Se la ricezione da parte del fotodiodo ricevitore è andata a buon fine, allora vuol dire che l’infrarosso è stato riflesso su una  
superficie chiara,



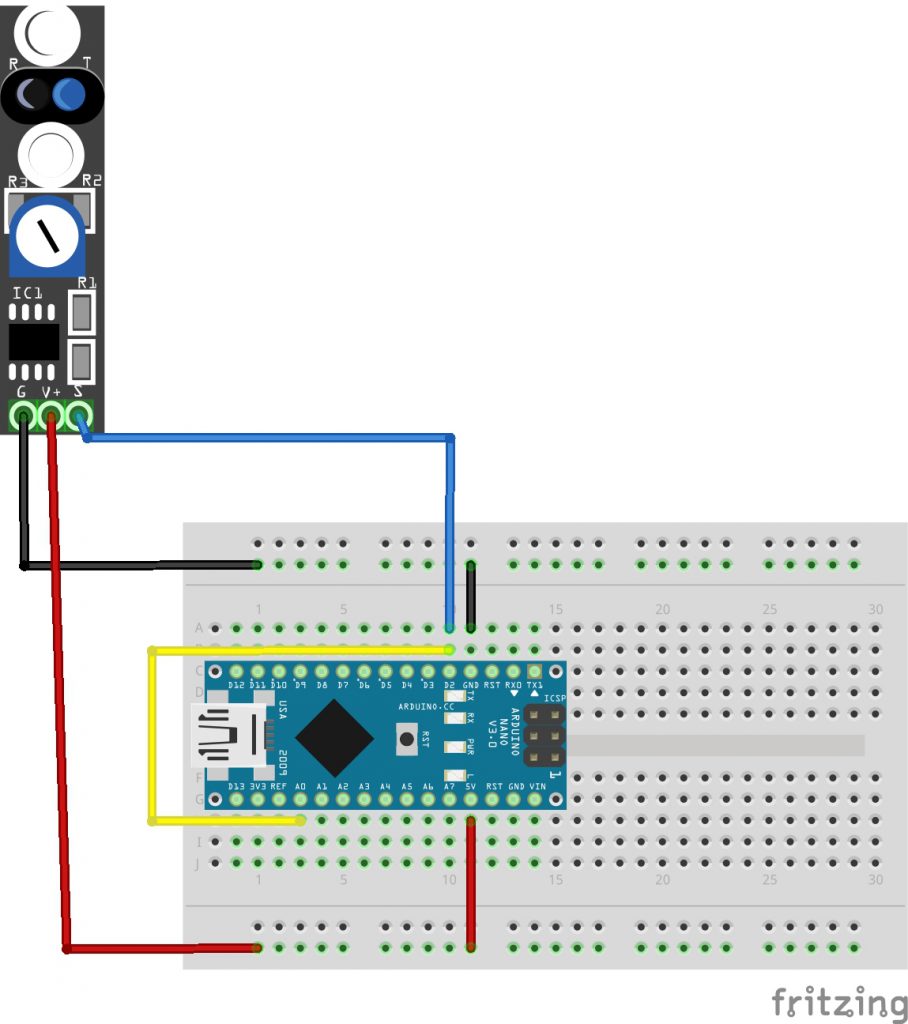
diversamente, dopo un brevissimo intervallo di tempo, si verifica un timeout e ciò indica che il segnale non è stato ricevuto: in tal caso siamo di fronte ad una superficie scura



I fotodiodi non sono gestiti direttamente da Arduino, ma da un comparatore (LM393) integrato nel modulo, il quale confronta la tensione generata dal fotodiodo ricevitore con quella di riferimento. Il risultato viene inviato sui pin del  
microcontrollore sotto forma di segnali TTL. In base a tale confronto, l’uscita DO del sensore (Data Output, è l’uscita che andrà collegata ad Arduino) sarà LOW oppure HIGH, ma può anche essere letta come pin analogico, ottenendo un intervallo di possibili valori compresi tra 0 e 1023. I valori di uscita (analogici e digitali) sono riassunti nella seguente tabella:



Per vedere il sensore all’opera, lo possiamo collegare ad un Arduino (nel nostro caso abbiamo usato la versione Nano, ma è lo stesso sulla UNO) per fare delle letture su diverse superfici. Come illustrato di seguito



i pin vcc e gnd del sensore vanno collegati rispettivamente a 5V e GND di Arduino, mentre il pin do lo colleghiamo all’ingresso digitale 2. Facciamo anche un ponte dal pin2 all’ingresso analogico A0 (filo giallo nella figura): in questo modo possiamo leggere contemporaneamente, sia valori analogici che digitali.

Il codice da caricare nella board è il seguente:

*/\* Line Track Sensors \*/*

*const int trackDigitalPin=2;*

*const int trackAnalogPin=A0;*

*void setup() {*

*/\* Inizializzo la comunicazione seriale a 19200 bit/sec \*/*

*Serial.begin(19200);*

*/\* Svuoto il buffer della seriale \*/*

*Serial.flush();*

*pinMode(trackDigitalPin, INPUT);*

*Serial.println(“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”);*

*Serial.println(” Arduino KY-033 IR Test \*”);*

*Serial.println(“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”);*

*}*

*void loop()*

*{*

*delay(1000);*

*Serial.print(“Lettura Analogica IR: “);*

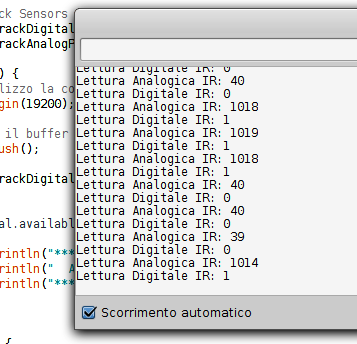
*Serial.println(analogRead(trackAnalogPin));*

*Serial.print(“Lettura Digitale IR: “);*

*Serial.println(digitalRead(trackDigitalPin));*

*}*

Tale codice è intuitivo: nel metodo loop(), il pin do del sensore viene letto eseguendo il metodo DigitalRead() sul pin 2 di Arduino mentre, tramite AnalogRead(), leggiamo il valore analogico dal pin A0 (con una pausa di 1 secondo). Infine stampiamo il risultato sulla porta seriale. Il monitor dell’IDE di Arduino va impostato ad una velocità di  
19200bps. Grazie a questo schema, possiamo vedere le diverse misurazioni in tempo reale, posizionando il sensore su vari tipi di superfici, effettuare dei test confrontando i risultati con la tabella di valori vista prima.



E’ facile notare che il comportamento dei sensori, varia anche in base al tipo di materiale: per esempio sul vetro nero o su un materiale riflettente, si osserva lo stesso risultato che si ottiene sulla carta bianca (così come anche sulle piastrelle del pavimento), quindi la vera differenza nero/bianco si ha su superfici chiare opache, come la carta o il legno.

Un robot che insegue una pista

Passiamo ora a vedere i concetti basilari sul funzionamento del robot, sulla struttura della traccia (stampata su dei fogli in formato A4) e sul posizionamento dei sensori. Per l’esperimento, è stato realizzato il robot 2WD con guida differenziale (quindi dispone di due ruote, un motore CC per ogni ruota), facente uso del modulo L298N collegato ad Arduino Uno. La programmazione del modulo motori e la realizzazione del robot, non verranno discusse  
in questa sede, ma si darà per scontata l’implementazione delle seguenti funzioni:

• forward(): entrambi i motori in avanti;

• stopLeft(): motore sinistro fermo;

• stopRight(): motore destro fermo;

• reverseRight(): motore destro indietro;

• reverseLeft(): motore sinistro indietro.

Consideriamo la funzione

*boolean trackOnLeftSensor()*

*{*

*return (digitalRead(trackOnLeftPin)==HIGH);*

*}*

la quale ritorna true quando viene rilevata una traccia sotto il sensore sinistro, false altrimenti. In maniera analoga definiamo la funzione che rileva la traccia sul sensore destro:

*boolean trackOnRightSensor()*

*{*

*return (digitalRead(trackOnRightPin)==HIGH);*

*}*

Tali funzioni saranno richiamate all’interno del metodo loop() del codice di Arduino.

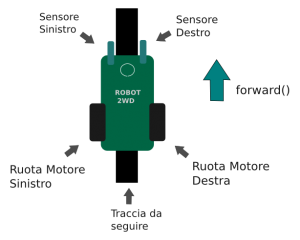
Funzionamento

Studiando il movimento che il robot dovrà fare, notiamo che con  
due soli sensori a disposizione, abbiamo due possibilità:

• Seguire una striscia bianca su una superficie nera;

• Seguire una striscia nera su una superficie bianca;

noi faremo riferimento alla seconda opzione, in quanto è più rapida ed economica. Da notare che sono entrambe implementabili allo stesso modo, basta solo invertire la logica dei sensori (scambiare HIGH con LOW nelle funzioni di controllo). L’algoritmo consiste nella continua lettura dei sensori, e nel far variare lo stato dei rispettivi motori non appena viene rilevata la traccia su almeno uno dei due. I sensori saranno posizionati fuori dalla traccia, ai lati del robot, lasciando così la pista nel mezzo



(In questa situazione, applicheremo la seguente regola: «finchè entrambi i sensori rilevano una superficie chiara, prosegui in avanti». Nel nostro codice, ciò si traduce nell’istruzione

*if (!trackOnLeftSensor() && !trackOnRightSensor()) forward();*

ed il robot andrà in avanti, finchè i sensori rilevano una superficie chiara. Se uno dei due sensori finisce sulla traccia, allora vuol dire che quest’ultima sta cambiando direzione, quindi bisogna adattare il robot alla direzione della traccia, effettuando una rotazione finchè non si ottiene lo stato precedente. La gestione avviene in questo modo:

• se la traccia è stata rilevata dal sensore sinistro (trackOnLeftSensor() ha restituito il valore true), vengono richiamate le funzioni stopLeft() e reverseLeft(), in modo da invertire la rotazione al motore sinistro, lasciando invariata la direzione di quello destro. Come risultato, si ottiene una rotazione del robot, seguita da forward() su entrambi i motori quando i sensori rilevano nuovamente la superficie bianca.

Tale azione è gestita nel codice di Arduino dalle istruzioni

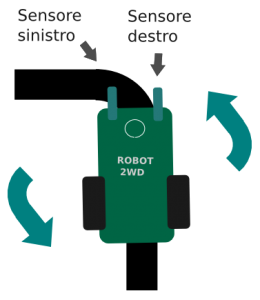
*if(trackOnLeftSensor() && !trackOnRightSensor())*

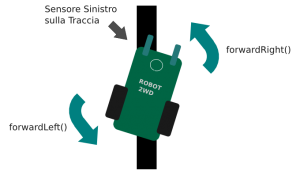
*{*

*stopLeft();*

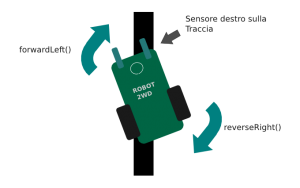
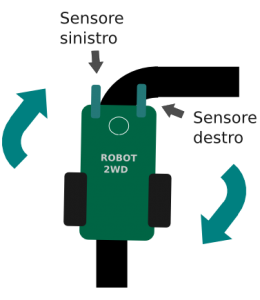
*reverseLeft();*

*}*





• se invece la traccia è stata rilevata dal sensore destro (trackOnRightSensor() ha restituito il valore true), vengono richiamate le funzioni stopRight() e reverseRight(), in modo che il motore destro vada all’indietro (il motore sinistro continua a girare in avanti), per poi rieseguire forward() su entrambi i motori quando i sensori tornano sulla superficie bianca



(anche in questo caso, dobbiamo aggiungere le rispettive istruzioni nel codice:

*if(!trackOnLeftSensor() && trackOnRightSensor())*

*{*

*stopRight();*

*reverseRight();*

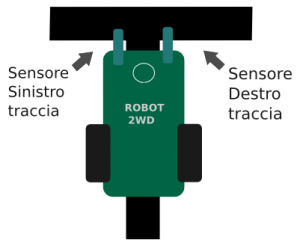
*}*

Utilizzando questo approccio, il nostro robot è in grado di gestire anche le svolte ad angolo retto e alcune curve più  
difficili, con un andamento di tipo «zig zag». Un miglioramento senza zig zag, può essere ottenuto con altri sensori, posizionati nel mezzo e programmati appositamente per aumentare la precisione. Il risultato ottenuto, sarà simile a quello visibile in questo [video.](https://www.youtube.com/watch?v=MYKhuzjc2Oc||video)  
A questo punto, potremmo chiederci:

• c’è un punto in cui il robot deve fermarsi?

• come gestire un incrocio?

Possiamo pensare di far fermare il robot quando entrambi i sensori rilevano la traccia, richiamando il metodo stopMotors():



if(trackOnLeftSensor() && trackOnRightSensor()) stopMotors();

Ovviamente ciò non sarebbe efficiente, in quanto entrambi i sensori potrebbero finire (per qualche secondo) sulla traccia di una lunga curva, che verrà erroneamente scambiata per incrocio. Sarebbe opportuno leggere i valori per un certo intervallo di tempo, prima di avviare la funzione di stop. L’idea è quella di includere una variabile contatore (nel codice l’abbiamo chiamata linesCount) che si incrementa ogni volta che entrambi i sensori rilevano la traccia, per poi reimpostarsi a zero ogni volta che si rileva la superficie bianca. Se la variabile raggiunge un certo valore, possiamo dire al robot di fermarsi, in quanto siamo certi che non si tratti di una curva o un semplice incrocio da  
ignorare, ma di una «piazzola di sosta»

*if(trackOnLeftSensor() && trackOnRightSensor())*

*{*

*linesCount++;*

*if (linesCount>6000)*

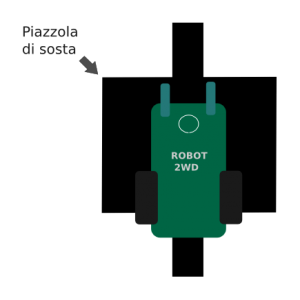
*{*

*stopMotors();*

*}else forward();*

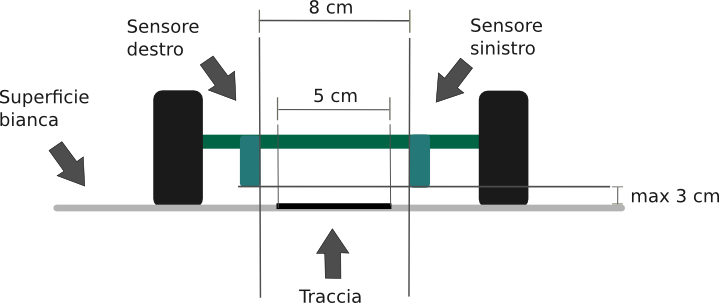
*}*

in questo modo, quando si ha una traccia a destra e una a sinistra, gli incroci saranno ignorati (il robot proseguirà  
dritto in alcuni casi, mentre sterzerà se arriverà troppo inclinato e uno dei due sensori rileverà la traccia per primo),  
ma sicuramente si fermerà ai box neri.

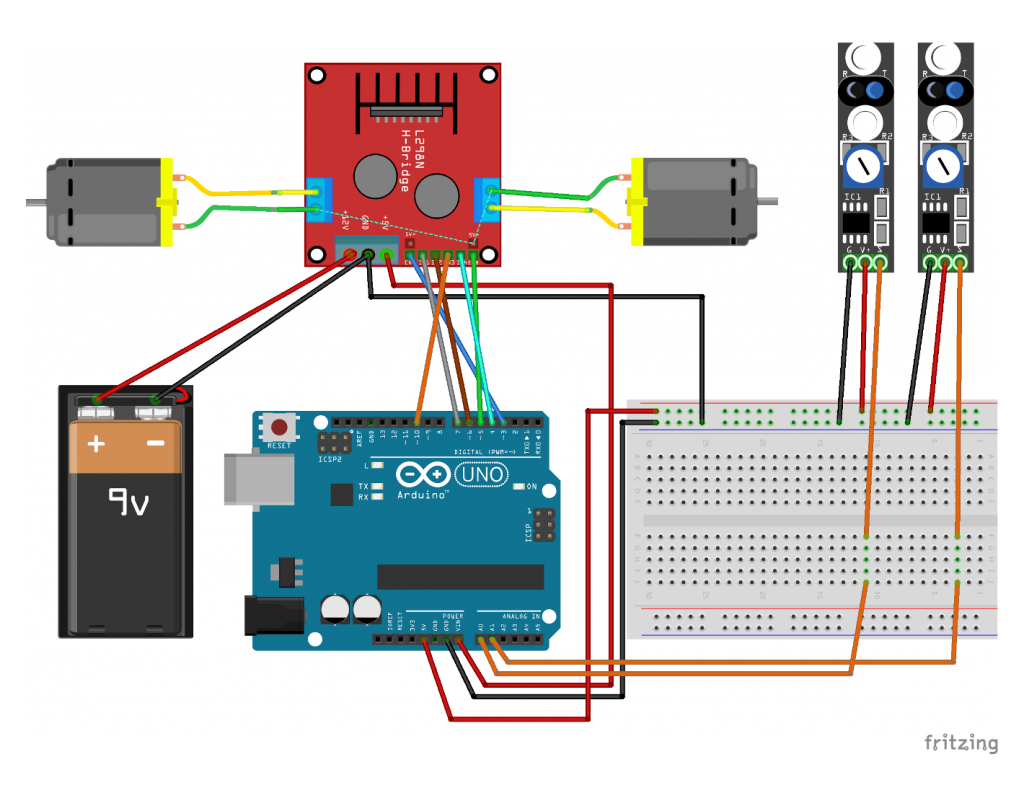


Posizionamento dei sensori, collegamento e sketch

È preferibile montare i sensori nella parte anteriore del telaio e possibilmente un pò distanti dalla traccia (poco più di mezzo cm). È bene tener presente che, maggiore è tale distanza, maggiore sarà l’effetto «zig zag». Nel nostro caso, la distanza di un sensore dall’altro è di 8 cm (in quanto la traccia è di circa 5 cm). Non ci sono delle misure rigide da rispettare, ma quelle descritte qui sono frutto di varie sperimentazioni, quindi sicuramente funzionanti. Nella figura seguente è illustrato lo schema (frontale) del nostro robot e le relative misure.



Alcuni sensori devono essere posizionati ad un’altezza massima di 3 cm dalla superficie bianca, mentre altri possono arrivare anche a 5: potete fissarli al telaio utilizzando dei distanziali reperibili in una ferramenta. Ricordiamo che i motori sono gestiti da un modulo L298N collegato ad un Arduino, il tutto alimentato da una batteria al litio da 7,4V. Il collegamento dei sensori si effettua come mostrato in figura seguente.



Negli allegati potete trovare lo sketch di Arduino, comprese anche le funzioni e variabili per la gestione del modulo, mentre riportiamo per intero il codice per la sola gestione dei sensori:

*/\* Line Track Sensors \*/*

*const int trackLeftPin=A1;*

*const int trackRightPin=A0;*

*/\* Linee nere su entrambi i sensori \*/*

*float linesCount=0;*

*/\* Omesse variabili Pin L298N \*/*

*void setup() {*

*/\* Inizializzo la comunicazione seriale usb a 19200 bit/sec \*/*

*Serial.begin(19200);*

*/\* Svuoto il buffer della seriale \*/*

*Serial.flush();*

*pinMode(trackLeftPin, INPUT);*

*pinMode(trackRightPin, INPUT);*

*/\* Omesse inizializzazione dei pin L298N \*/*

*/\* Imposto velocità iniziale dei motorini \*/*

*analogWrite(enA, 90);*

*analogWrite(enB, 90);*

*forward();*

*}*

*void loop() {*

*/\* Verifica se la traccia è solo sul sensore destro \*/*

*if(!trackOnLeftSensor() && trackOnRightSensor()) {*

*stopRight(); reverseRight();*

*}*

*/\* Verifica se la traccia è solo sul sensore sinistro \*/*

*if(trackOnLeftSensor() && !trackOnRightSensor())*

*{*

*stopLeft();*

*reverseLeft();*

*}*

*/\* Verifica se la traccia è su entrambi i sensori \*/*

*if(trackOnLeftSensor() && trackOnRightSensor())*

*{*

*linesCount++;*

*if (linesCount>6000)*

*{*

*stopMotors();*

*}else forward();*

*}*

*/\* Verifica se entrambi i sensori NON sono sulla traccia \*/*

*if(!trackOnLeftSensor() && !trackOnRightSensor())*

*{*

*linesCount=0;*

*forward();*

*}*

*}*

*/\* c’è linea nera sotto il sensore sinistro \*/*

*boolean trackOnLeftSensor()*

*{*

*return (digitalRead(trackLeftPin)==HIGH);*

*}*

*/\* c’è linea nera sotto il sensore destro \*/*

*boolean trackOnRightSensor()*

*{*

*return (digitalRead(trackRightPin)==HIGH);*

*}*

*/\* c’è linea nera sotto entrambi i sensori \*/*

*boolean trackOnBothSensor()*

*{*

*return trackOnLeftSensor() && trackOnRightSensor();*

*}*

*/\* Omesse funzioni L298N \*/*

Struttura e realizzazione della traccia

È possibile costruire una pista mettendo insieme dei fogli in formato A4 con pezzi di traccia stampati sopra, attaccandoli con colla o nastro adesivo. Negli allegati è possibile trovare immagini in formato png e jpeg, già pronte per essere stampate, ma si possono trovare anche file svg da modificare a piacere con Inkscape. I pezzi di traccia sono:

• rettilineo: una semplice linea retta;

• svolta a destra: una svolta a 90 gradi verso destra;

• svolta a sinistra: svolta a 90 gradi verso sinistra;

• curva destra: consente di far girare a destra il robot mediante  
una curva;

• curva sinistra: come prima, ma verso sinistra;

• piazzola: area di sosta, permette al robot di fermarsi;

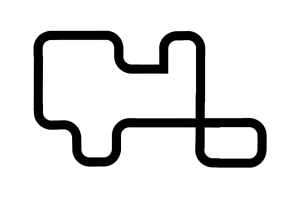
• incrocio: consente di realizzare svolte multiple oppure piste a  
forma di «otto» (come le piste polistil per bambini);

• bivio: una svolta a destra o a sinistra, può essere utilizzato  
per altri casi non descritti qui;

• curva destra alta: curva destra con rettilineo;

• curva sinistra alta: curva sinistra con rettilineo;

Con questi pezzi si possono ottenere tante combinazioni di pista,  
tra cui la pista mostrata di seguito:



A causa della svolta ad angolo retto e delle improvvise curve, tale pista è gestibile se il robot si muove con una velocità limitata. Nel nostro caso, alimentando il tutto con una batteria lipo a 7.4 V, abbiamo impostato il pwm dei motori a 90 e l’andamento appare piuttosto stabile: spesso l’incrocio viene ignorato, in quanto il robot prosegue dritto (grazie al controllo della variabile linesCount), e difficilmente esce fuori dalla traccia.

Conclusioni

Quanto realizzato in questo progetto, può essere implementato come una funzionalità aggiuntiva nei robot visti nel numero speciale Arduino Projects. Infine ci ritroveremo con un robot inseguitore di linea: i più esperti, potranno aggiungere dei sensori che, leggendo delle strisce nere aggiuntive ai lati della traccia, possono interpretarle come segnali in prossimità di un bivio o un incrocio e «lasciando» al robot, la possibilità di decidere in anticipo cosa fare. I limiti di ciò che si può fare, dipendono solo dalla fantasia del costruttore.