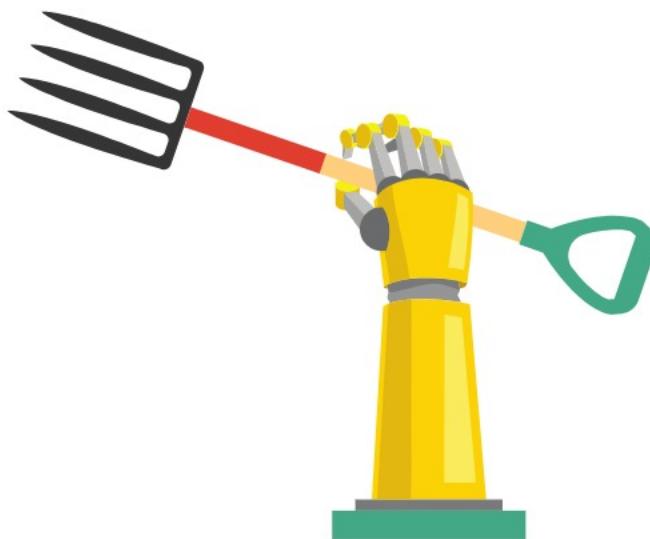


UN NUOVO FUTURO

Il ruolo dei robot nell'agricoltura



Ashraf Emhemmed

Docente responsabile: prof. Mattia Bergomi
Lavoro di maturità 2017-18, Liceo cantonale di Lugano 1

Indice

1 Premessa	4
2 Introduzione	4
3 Robot agricoli	4
4 Sviluppo del progetto e procedimento	6
4.1 L'idea	6
4.2 Fasi di lavoro	7
4.3 Componenti	8
5 Programmazione	11
5.1 Kinect	11
5.2 Arduino	15
5.2.1 Coordinate	15
5.2.2 Rotary encoder	17
5.2.3 Movimento	18
5.2.4 Diagramma di flusso	24
6 Il robot	32
7 Cosa ho appreso dal LAM	33
7.1 Ricerca	34
7.2 Programmazione	35
7.2.1 Logica del programma	35
7.2.2 Scrittura del codice	35
7.2.3 Debugging	36
8 Limiti e possibili miglioramenti	37
9 Possibili problematiche	37
10 Conclusione	38
11 Allegati	40

Sommario

È l'alba di un nuovo giorno, si sente il canto del gallo, il muggito delle mucche, e il suono dei motori elettrici che lavorano costantemente per portare a termine i loro compiti. Si sveglia il contadino, che con un gesto della mano su un display controlla l'andamento della produzione. Irrigazione piante: OK, Coltura ortaggi: OK, Mungitura mucche: OK; dopo aver letto che tutti i compiti sono stati svolti correttamente si alza, finisce la sua routine mattutina, e si dirige verso il suo ufficio per lavorare sull'algoritmo di un nuovo programma...

Le innovazioni tecnologiche nel campo della robotica stanno cambiando ogni aspetto delle nostre vite; perciò in questo lavoro di maturità ho voluto esporre le diverse applicazioni della robotica che potenzialmente migliorerebbero il settore agricolo, e il processo che mi ha condotto alla costruzione di un robot che ha la capacità di piantare semi autonomamente.

1 Premessa

Con molto piacere ho avuto l'occasione di lavorare su un LAM che riguarda uno degli argomenti che suscita il mio interesse più in assoluto, mettendolo in relazione, inoltre, con il campo forse più importante della nostra vita, ossia quello dell'agricoltura. Desidero ringraziare tutti coloro che mi hanno aiutato nella realizzazione del mio lavoro di maturità con suggerimenti, critiche, e aiuti pratici. Ringrazio anzitutto il professor Mattia Bergomi, docente responsabile, per il suo supporto e la sua disponibilità. Un ringraziamento particolare, inoltre, va a Federico D'Ignazio, per i consigli utili nella fase di programmazione e a Ilian Arigoni, per l'aiuto nella costruzione del robot. Ringrazio infine, tutti coloro che mi hanno fornito il loro prezioso aiuto, tra amici, parenti e docenti.

2 Introduzione

Il tema principale che ha dato ispirazione al mio progetto è quello dell'agricoltura; il lavoro di maturità in robotica infatti, mi si è presentato come un'ottima occasione per lavorare su un progetto che unisse la robotica ad un campo completamente differente, al quale può dare un enorme vantaggio se applicata in maniera intelligente. Perciò ho iniziato la ricerca sulle applicazioni della robotica in agricoltura, e ho trovato diversi progetti - anche a livello commerciale - che automatizzano diversi aspetti dell'agricoltura; da robot che raccolgono la frutta, ad altri che possono mantenere in modo autonomo un orto di piccole dimensioni.

In particolar modo, era di mio interesse costruire un robot che potesse piantare semi autonomamente. Le prime idee che ho avuto non erano affatto le migliori, ci sono volute molte revisioni, anche nella fase di progettazione, per arrivare ad un progetto chiaro e che avesse una funzione definita. Passerò ora a parlare delle interessanti innovazioni della robotica nell'agricoltura, per poi passare alla descrizione del progetto e dei passaggi che hanno portato al risultato finale.

3 Robot agricoli

Negli ultimi anni, la tecnologia ha avuto un forte impatto nei diversi aspetti delle attività umana. Molti esperti, infatti, ritengono che la tecnologia, e in particolare il campo della robotica, potrebbe rivoluzionare completamente il settore dell'agricoltura. Questa opinione non viene dal nulla: i robot porterebbero all'aumento della produzione agricola, ridurrebbero i costi, e colmerebbero il vuoto lasciato dalla mancanza di mano d'opera nel settore, oltre alla capacità di sfruttare superfici più estese di terreni. Le applicazioni nell'agricoltura che beneficierebbero dei robot sono molte e di vario genere; quelle maggiormente richieste sono indirizzate al raccoglimento dei frutti e ortaggi, monitorarne la maturazione, pulizia del terreno da erbacce e malattie senza l'uso aggressivo di erbicidi, e persino nel campo dell'allevamento per gestire la salute e il benessere degli animali.

Gli umani sono ancora meglio dei robot per quanto riguarda la coltura, ma c'è molta ricerca su come rendere i robot più efficaci. Alcuni di questi si basano su telecamere RGB per identificare gli ortaggi, e quindi la chiave sta in un software veloce e accurato, che sfrutta il "deep learning" per rendere il robot più autonomo e adattabile a diverse situazioni. I risultati ottenuti sono molto promettenti; ad esempio un robot che raccoglie le fragole (costruito in Gran Bretagna) riesce a raccogliere una fragola ogni due secondi, mentre un essere umano ne raccoglierebbe

soltanto da 15 a 20 al minuto. L'obiettivo, tuttavia, è di arrivare a robot che possono raccogliere qualsiasi tipo di ortaggio, e che possano farlo in maniera accurata e di alta qualità. Un altro aspetto molto importante è il tempismo: se si raccoglie un frutto troppo presto se ne perde la maturità, e quindi sarebbe utile trovare anche metodi per monitorarne la maturazione.

L'organizzazione delle Nazioni Unite per l'alimentazione e l'agricoltura stima che circa il 20-40% del raccolto annuo è perso per via di parassiti e malattie, pur usando circa due tonnellate di erbicidi. Dispositivi intelligenti come robot e droni ad esempio, potrebbero diminuire drasticamente l'uso di erbicidi, scovando infestazioni di erbacce e parassiti prima che si manifestino su grande scala. Nel mercato, inoltre, cresce sempre di più la richiesta per prodotti che contengono meno sostanze chimiche, cosa in cui i robot potrebbero aiutare. Esistono, infatti, molti progetti di droni che con l'uso di telecamere RGB e sensori, possono verificare la salute del terreno con l'obiettivo di dare informazioni su eventuali attacchi di erbacce e su dove sono localizzati. Altri applicano direttamente i pesticidi in zone di bisogno per evitare di spargerli su tutto il campo. I droni non ci forniscono solo dati sulla presenza di parassiti che danneggiano le altre piante, ma possono essere usati anche per monitorare la ricchezza di sostanze nutritive nel suolo (nitrati, sostanze azotate, etc.); questo farebbe risparmiare grandi quantità di fertilizzanti che verrebbero altrimenti usati per terreni già ricchi. Dunque queste applicazioni avrebbero un enorme effetto positivo sulla tutela dell'ambiente e della biodiversità.

Come i famosi "fitness trackers" che monitorano la nostra attività (come passi, battito cardiaco, ecc.) sono stati creati collari che monitorano quella degli animali; ad esempio si può capire quando una mucca è fertile (si muovono molto in questo periodo), e così gli allevatori vengono informati su quando sia giunto il momento ideale per l'accoppiamento. Non solo questo, ma gli "animal trackers" possono anche dare informazioni sullo stato di salute degli animali e sulla qualità del loro mangime. È vero che l'esperienza dei pastori non potrà mai essere rimpiazzata dai computer, ma con queste soluzioni la gestione di un grande numero di animali risulta molto più semplice e ottimizzata.

Osservando le diverse ricerche e applicazioni della robotica nell'agricoltura, possiamo comprendere il grande contributo che i robot possono fornire in questo campo. Essendo affascinato da entrambi i campi della robotica e dell'agricoltura, ho stabilito la tematica del mio progetto. Siccome progetti come quelli descritti in precedenza sono molto specifici ho pensato di partire dalle basi dell'agricoltura: la prima cosa da fare prima di monitorare la crescita degli ortaggi, controllare la presenza di erbacce e cogliere i frutti, è di piantare il seme che darà vita alla pianta in primo luogo; quindi la prima manifestazione dell'idea era di un robot che, essenzialmente, piantasse semi. Ovviamente, parallelamente alla definizione del progetto, consideravo un'eventuale applicazione del robot in grande scala (in modo idealistico e non proprio realistico); inizialmente pensavo ad un robot che piantasse semi di piante come i cereali, che non richiedono una alta precisione, ma poi, con la maggiore libertà e accuratezza che mi ha fornito il Kinect, l'idea è cambiata in un robot che piantasse i fiori; la maggiore precisione in questo caso può essere utile, in quanto si potrebbe usare per creare delle composizioni complicate sul terreno con i fiori, una sorta di modo per disegnare sulla terra.

4 Sviluppo del progetto e procedimento

4.1 L'idea

L'idea di partenza era di costruire un robot che potesse piantare dei semi a intervalli predeterminati e su un'area quadrata, in modo che il robot, partendo da un punto conosciuto, si muovesse in avanti fino ad arrivare alla fine del "quadrato", poi girasse di 180 gradi per ritornare e piantare semi sulla linea adiacente a quella precedente.

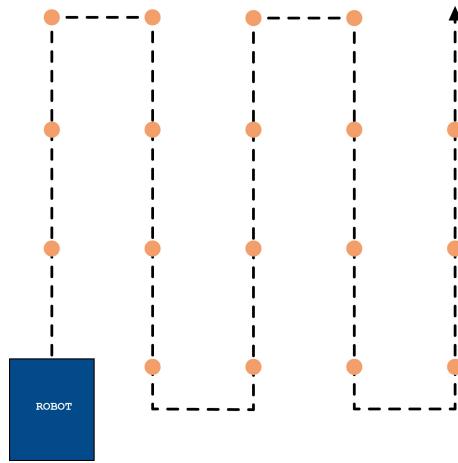


Figura 1: Schema del movimento del robot non conoscendo la posizione istantanea e conoscendo quella iniziale

Da questo progetto risultavano due problemi da risolvere; il primo, era quello che comprendeva la posizione del robot: ovvero, come si fa a sapere se esso abbia veramente percorso la distanza corretta? Questa idea sarebbe potuta funzionare solo nel caso in cui si potesse verificare la distanza percorsa dal robot; è vero che questo si poteva in parte compiere usando dei "rotary encoder" (dei dispositivi che permettono di calcolare quanti giri ha compiuto il motore, e sapendo il diametro delle ruote, quindi, anche calcolare la distanza percorsa). Questo, tuttavia, non sarebbe potuto funzionare su qualsiasi tipologia di terreno, ad esempio, se ci fosse una sorta di "collinetta", il robot percorrendo la stessa distanza (sulla superficie) che lo separa dalla destinazione, si fermerebbe prima di raggiungerla, perché la distanza effettiva che ha percorso è quella che su una superficie piana corrisponde alla distanza tra il robot e la destinazione. Siccome il mio obiettivo, e quello dell'automatizzazione in generale, è di rendere, un robot ad esempio, funzionante in diversi contesti e adattabile all'ambiente senza l'intervento umano, questa metodologia iniziale non soddisfaceva questo obiettivo, e quindi serviva un altro metodo. Il secondo problema, analogamente al primo, concerne il fatto che il robot doveva partire sempre dalla stessa posizione, e anche in questo caso il robot non si poteva definire veramente autonomo. L'ideale era quello di trovare un metodo per localizzare il robot, in questo modo si sarebbero risolti i problemi precedenti, in quanto, sapendo la posizione del robot, si poteva verificare se effettivamente avesse raggiunto la destinazione confrontando la sua posizione attuale con quella della destinazione. In aggiunta, conoscendo la posizione iniziale, il punto di partenza non doveva essere sempre lo stesso. Disponendo di un sistema di localizzazione, infine, si toglieva un vincolo

dato dal primo metodo; ovvero, i punti di destinazione non dovevano essere gli stessi descritti prima, e il movimento poteva risultare più libero. Infatti, i punti di arrivo potevano coincidere con i punti nei quali verrebbero piantati i semi, e questi, potevano essere in qualunque punto dell'orto.

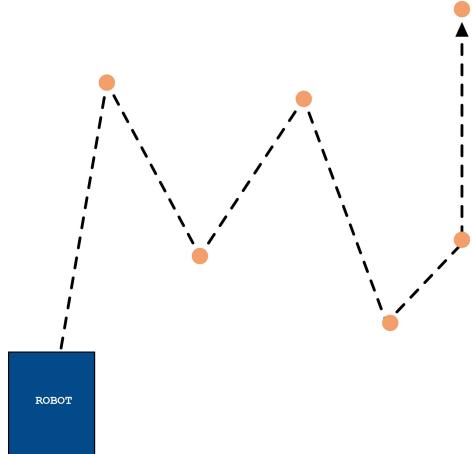


Figura 2: Schema del movimento del robot conoscendo la sua posizione in ogni momento

Il primo metodo di localizzazione considerato, che mi è stato consigliato dal prof. Strupler, consisteva nella triangolazione della posizione del robot utilizzando due sensori a infrarosso; ponendo un sensore a entrambi i vertici di un lato del quadrato, e conoscendo la distanza tra i due sensori, si possono usare i dati per calcolare l'esatta posizione del robot tramite semplici equazioni trigonometriche. Valutando il metodo precedente, si presentava il problema della mobilità. Ovvero, risultava difficile migrare il progetto su diversi terreni, in quanto si devono spostare più componenti, rendendolo poco modulare. Più tardi, tuttavia, mi sono accorto di un altro metodo più accessibile, ossia di utilizzare il Kinect per la localizzazione.

4.2 Fasi di lavoro

Siccome le mie conoscenze di elettronica erano limitate, non avendo mai costruito un robot prima d'ora, la prima cosa che ho dovuto fare è stato imparare il minimo necessario per poter affrontare un progetto simile. Dunque, ho iniziato la ricerca sui vari argomenti riguardanti la programmazione e l'elettronica, e passo dopo passo, imparando sempre cose nuove, acquisivo nuove competenze fondamentali alla realizzazione del progetto. La prima cosa su cui mi sono concentrato è stato il funzionamento dei motori DC, siccome il progetto si basava sulla mobilità del robot sul terreno; di conseguenza, è anche iniziata la fase di costruzione del robot stesso, per poterne testare i movimenti. Dopotutto, era necessario comprendere il funzionamento del Kinect in modo da poter iniziare la fase di programmazione, poiché l'algoritmo si sarebbe basato su quello. Il modo più semplice per riconoscere il robot, era di montarci sopra un oggetto di un colore particolare, io ho scelto il rosso, in modo da poterlo distinguere dallo sfondo e da altri oggetti adiacenti. Di base, il Kinect dovrebbe riconoscere i colori dell'oggetto e fornirne le coordinate esatte, ma questo verrà descritto più in dettaglio in un altro punto. Dopo essere riuscito a programmare il Kinect in modo da riconoscere il robot, il passo successivo era di mandare le

coordinate da Processing, su PC, ad Arduino tramite Bluetooth. Finalmente, quando l’Arduino poteva ricevere le coordinate correttamente, si poteva passare alla fase di programmazione dell’algoritmo principale, ossia il funzionamento del robot.

4.3 Componenti

Il funzionamento del robot necessita del funzionamento di diversi componenti insieme, che possiamo dividere in due gruppi principali: La parte “PC”, e la parte “Arduino”. La prima parte comprende solamente il computer e il Kinect, e ha il compito di calcolare le coordinate del robot e inviarle all’Arduino. La seconda parte, invece, comprende molti più componenti e costituisce gran parte del progetto. I componenti che compongono il robot sono elencati in seguito:

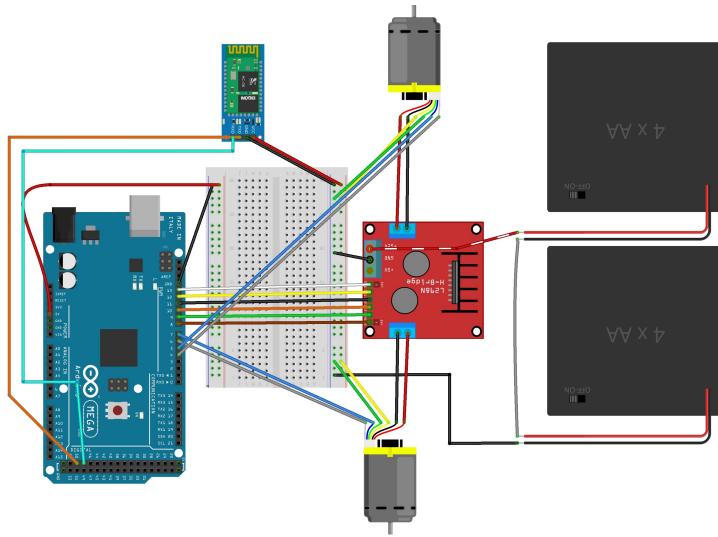
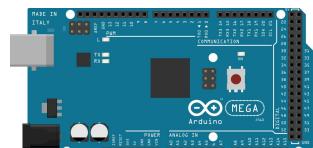
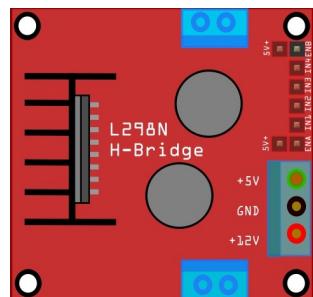


Figura 3: Schema dei componenti più importanti

Arduino Mega 2560: Il microcontrollore è la parte più importante del robot, possiamo considerarlo il “cervello” che guida tutti i componenti. Arduino è stato creato da un ingegnere italiano con l’obiettivo di fornire una piattaforma per creare prototipi a basso costo; anche se, effettivamente, Arduino può essere utilizzato senza problemi anche a livello commerciale. Arduino è nato come un progetto *open source*, questo significa che i blueprints sono forniti apertamente e che chiunque può replicarlo. Ho scelto Arduino MEGA per la quantità di pin che possiede, in quanto quelli di Arduino UNO non erano sufficienti.



H-bridge: Per usare dei motori con Arduino non basta collegarli alla scheda, ma sono necessari più componenti per garantirne il corretto funzionamento. Nel mio progetto ho usato un L298N. Questa semplice scheda ha come input l’alimentazione (12V o 5V, nel mio caso uso 12V) e il segnale PWM mandato dal Kinect per entrambi i motori; Come output, invece, ha il voltaggio relativo al segnale PWM per

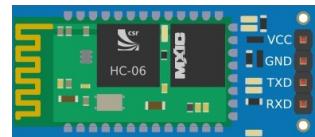


ogni motore. Quindi con Arduino si manda un segnale PWM (da 0 a 255), poi questo segnale viene interpretato dall'H-Bridge che fornisce il voltaggio corretto ai motori. Questa scheda rende il controllo dei motori molto semplice, per questo è molto usata nella costruzione di robot mobili.

2x motori DC (9V): Non si può avere un robot mobile senza motori. Esistono molti tipi di motori (Stepper, DC,...) ma per questo progetto ho scelto i motori DC.



Modulo Bluetooth HC-06: Esistono due tipi di adattatori Bluetooth per Arduino: HC-05 e HC-06. Il modulo Bluetooth è utile per sostituire una connessione seriale con cavo tra Arduino e PC, e quindi si possono trasferire dati via seriale nello stesso modo di una connessione via USB. Il modulo è relativamente semplice da usare, dispone di quattro pin: 5V, Ground, TXD (connesso a RXD dell'Arduino), RXD (connesso a TXD dell'Arduino). Ho scelto di usare Bluetooth come metodo di comunicazione tra PC e Arduino poiché il robot in questo modo non è più vincolato da cavi e può muoversi liberamente, inoltre il modulo Bluetooth risultava una scelta attrattiva per la sua facilità d'uso e per il suo basso consumo energetico. L'uso del Bluetooth tuttavia ha anche alcune limitazioni: la copertura del modulo ha un raggio che non supera i 20 metri, e ciò lo rende inefficiente per progetti che sfruttano aree estese (come si immagina per le applicazioni reali di robot agricoli simili a questo), quindi in questo caso sarebbe più opportuno usare un diverso metodo di comunicazione, come ad esempio WiFi o onde radio.



Batterie: Quando si costruisce un robot, o qualsiasi progetto di elettronica, una delle questioni più importanti che emerge è quella dell'alimentazione. Se la sorgente ha un voltaggio troppo basso il progetto potrebbe non funzionare, e se invece è troppo alto, si corre il rischio di bruciare i componenti.

Servomotori: I servomotori sono simili ai motori DC, però possono ruotare di solo 180°. Il vantaggio di questi motori, tuttavia, è che possono ruotare un oggetto di un angolo preciso. In questo progetto sono usati due servomotori, uno per alzare e abbassare l'aratro, e l'altro per ruotare la vite che fa cadere i semi. Quest'ultimo è stato modificato in modo da farlo ruotare indefinitamente, con lo svantaggio di non poter più stabilire l'angolo di rotazione.



Microsoft Kinect: Il Kinect è un sensore progettato da Microsoft, sviluppato inizialmente come periferica aggiuntiva all'Xbox 360. Questa innovazione ha suscitato l'interesse di molti sviluppatori che vedevano nel Kinect un modo per dare una nuova dimensione ai loro progetti e quindi il suo uso si è esteso anche ai computer. Il fatto interessante è che esistono molte librerie, sia sviluppate da Microsoft che da terzi, che rendono l'uso del Kinect relativamente semplice. Per il mio progetto ho usato una piattaforma chiamata Processing, ossia un linguaggio di programmazione basato su Java, che mette a disposizione funzioni di alto livello



Figura 4: Kinect v1 - fonte: cashconsoles

per gestire in modo facile elementi di grafica, e in generale per creare opere d'arte. Processing inoltre dispone di un Serial Output che permette la comunicazione con la scheda Arduino, cosa fondamentale per questo progetto.

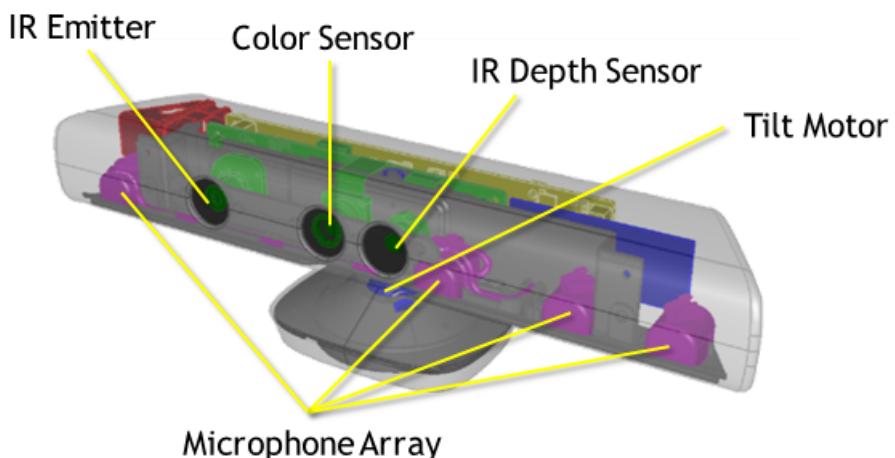


Figura 5: Componenti Kinect v1 - fonte: Microsoft

Il Kinect è composto da:

- **Telecamera RGB:** fornisce un'immagine 640x480 pixel a 30 frames/secondo;
- **Emettitore di infrarossi**
- **Sensore a infrarossi:** fornisce un'immagine 320x240 pixel a 30 frames/secondo;
- **Microfoni**
- **Motore:** permette di variare l'inclinazione del Kinect

5 Programmazione

La parte più laboriosa di questo lavoro di maturità è stata senza dubbio quella di programmazione, anche per il fatto che comprendeva non solo la parte di Arduino, ma anche quella del Kinect.

5.1 Kinect

Il funzionamento del Kinect su Processing è reso possibile da diverse librerie; quella che ho usato è *Open Kinect for Processing*, sviluppata da *Daniel Shiffman* e *Thomas Sanchez*.

Le prime informazioni di cui si necessita sono l'immagine RGB rilevata dalla telecamera del Kinect e “l'immagine a infrarossi” rilevata dal sensore del Kinect. Avendo queste due informazioni è possibile isolare il robot dal resto dell'immagine RGB e calcolarne le coordinate. Dall'immagine RGB si cercano tutti i pixel che hanno un determinato colore (quello dell'oggetto montato sopra il robot, di un colore distinguibile dallo sfondo) e si calcola la media tra i pixel per ottenere le coordinate del pixel al centro della figura.

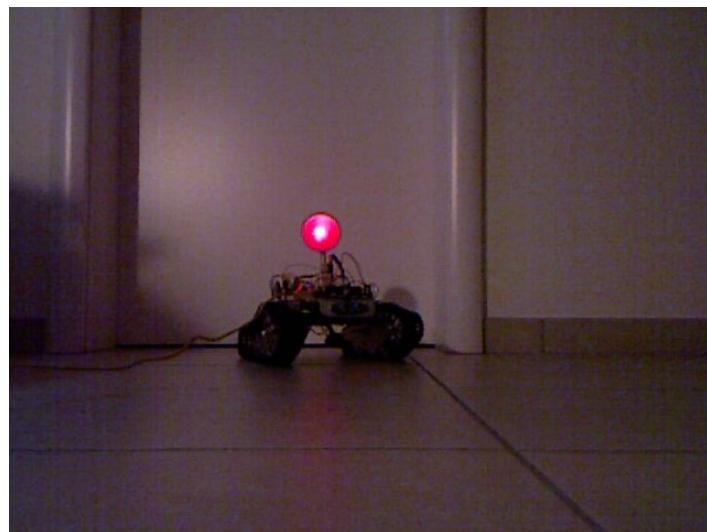


Figura 6: Immagine RGB

1. In questo estratto del codice vengono passati in rassegna tutti i pixel che compongono l'immagine RGB, e viene calcolata l'intensità del colore di ogni singolo pixel; ogni volta che un pixel risulta avere un'intensità sufficiente, viene colorato in bianco per riconoscerlo e viene aggiunto 1 alle variabili responsabili del calcolo del pixel medio.

```

1  for (int x = 0; x < img.width; ++x) {
2      for (int y = 0; y < img.height; ++y) {
3
4          int offset = x + y * img.width;
5
6          if (calcRedVal(img.pixels[offset]) >= threshold ) {
7              // Set pixel to white
8              img.pixels[offset] = color(255, 255, 255);
9
10         sumX += x;
11         sumY += y;
12         totalPixels++;
13     } else {
14         // Set pixel to black
15         img.pixels[offset] = color(0, 0, 0);
16     }
17 }
18
19 img.updatePixels();
20
21

```

2. Dopo aver ricavato tutti i pixel di un colore desiderato, si trovano le coordinate di quello intermedio, in quanto sarà il pixel di cui si ricaverà la distanza dal sensore.

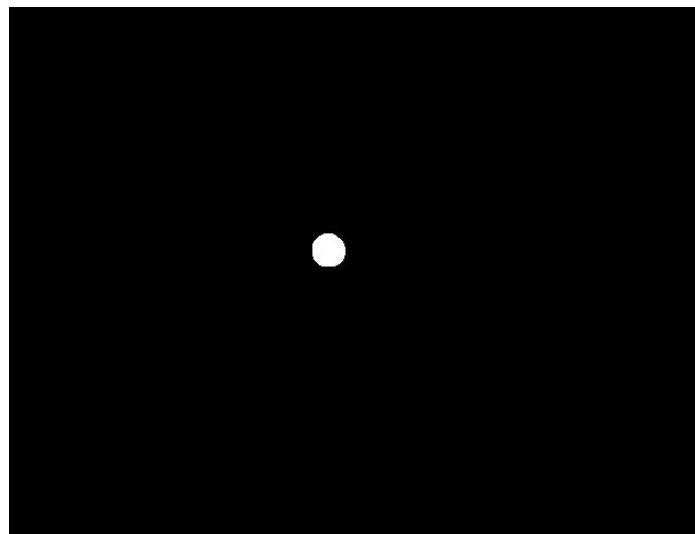


Figura 7: I pixel in bianco sono i pixel che hanno colore rosso

```

2 avgX = 1.045 * (sumX / totalPixels);
4 avgY = 0.96 * (sumY / totalPixels);
4 avgMappedX = map((long) avgX, 0, 1280, 0, 1280);
4 avgMappedY = map((long) avgY, 0, 960, 0, 960);
6 avgOffset = int(avgMappedX+avgMappedY*depthImage.width);
6 avgDepth = depth[avgOffset];
8
8 fill(150,255,0); // viene disegnato un cerchio per osservare la posizione del
8     pixel
10 ellipse(avgX, avgY, 8, 8);
11 return img;
12 }

```



Figura 8: I pixel più chiari sono più vicini al Kinect

- Ora che si conosce il pixel di cui si deve calcolare la distanza, si possono convertire i dati ricavati dal Kinect in valori applicabili al mondo reale, ossia in metri.

```

PVector depthToWorld(int x, int y, int depthValue) {
2     final double fx_d = 1.0 / 5.9421434211923247e+02;
3     final double fy_d = 1.0 / 5.9104053696870778e+02;
4     final double cx_d = 3.3930780975300314e+02;
5     final double cy_d = 2.4273913761751615e+02;
6     PVector result = new PVector();
7     double depth = rawDepthToMeters(depthValue);
8     result.x = (float)((x - cx_d) * depth * fx_d);
9     result.y = (float)((y - cy_d) * depth * fy_d);
10    result.z = (float)(depth);
11    return result;
12 }

```

-
4. Si convertono i valori in metri.

```
1 float rawDepthToMeters(float depthValue) {  
2     if (depthValue < 2047) {  
3         return (float)(1.0 / ((double)(depthValue) * -0.0030711016 + 3.3309495161))  
4     }  
5     return 0.0f;  
6 }
```

5. Infine, avendo le esatte coordinate del robot e ponendo l'origine degli assi cartesiani (0;0) nella posizione del Kinect, si possono inviare i valori all'Arduino tramite Bluetooth grazie a una connessione Serial.

```
1 void sendMessage(int tag, int value){ // Mandare l'indice e il valore alla  
    // porta seriale  
myPort.write(H HEADER); // mandare un header in modo che l'arduino sappia che stia  
    // arrivando un nuovo valore  
3 myPort.write(tag); // mandare un tag che identifica il valore  
char c = (char)(abs(value) / 256); // msb valore assoluto per correttezza di  
    // calcoli con valori negativi  
5 myPort.write(c); // convertire i valori ascii in valori numerici  
c = (char)(abs(value) & 0xff);  
7 myPort.write(c);  
if(value < 0) {  
9     myPort.write('n'); // verificare se numero negativo o positivo  
} else  
11    myPort.write('p'); // mandare n per negativo o p per positivo  
}
```

Per illustrare più chiaramente la posizione del robot rispetto al Kinect, ho creato un grafico sul quale vengono illustrate le coordinate del robot su un sistema di assi cartesiani in cui l'origine è posto il Kinect.

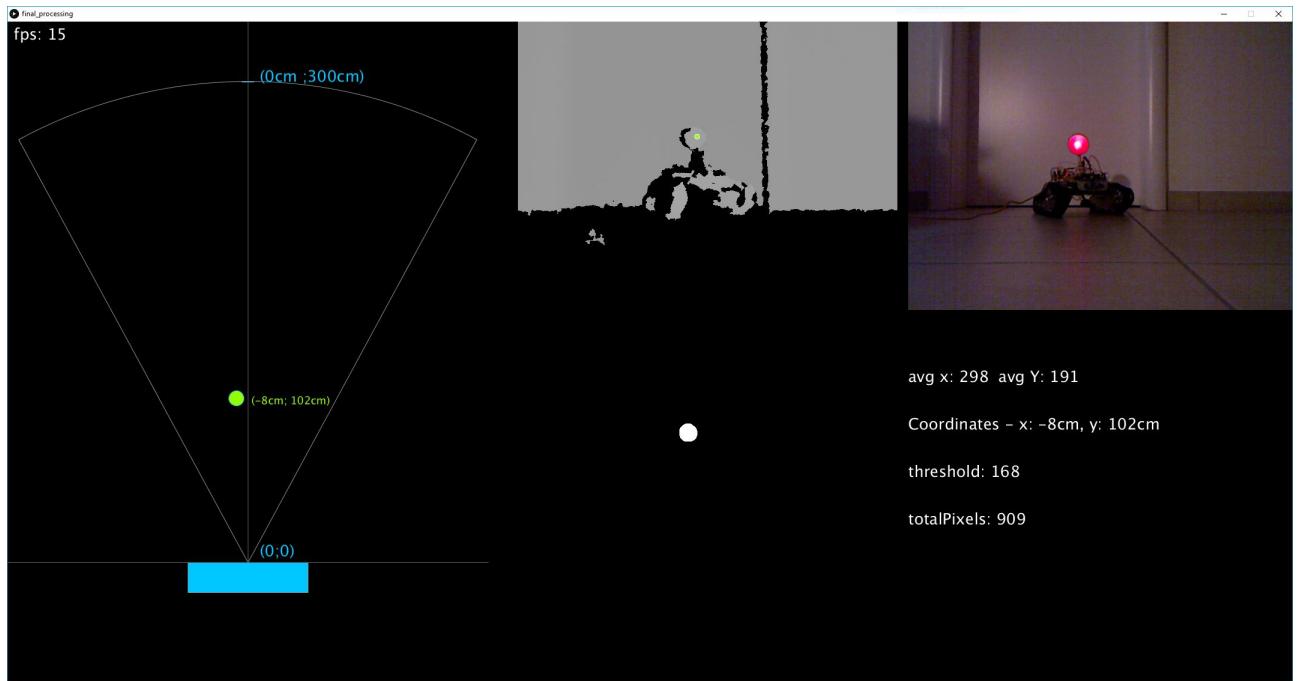


Figura 9: Risultato finale come appare in Processing

5.2 Arduino

L'Arduino è uno dei componenti più importanti di questo progetto, e in particolar modo lo è l'algoritmo che lo guida. Programmare l'Arduino, difatti, è stata la parte più impegnativa, poichè un minimo errore, voleva dire il non funzionamento del robot.

Il programma si basa su tre funzionalità principali che vengono usate più volte nelle diverse funzioni elencate nel diagramma di flusso:

5.2.1 Coordinate

- **readCoordinates()**: Questa funzione legge i valori delle coordinate dal Kinect, ricevuti tramite Bluetooth, e aggiorna le coordinate X e Y (valX e valY) nel codice di Arduino.

```
void readCoordinates() {
  if ( mySerial.available() >= MESSAGE_BYTES) // quando porta seriale BT
    disponibile
  {
    if ( mySerial.read() == HEADER)           // se riconosce il header
    {
      char tag = mySerial.read();            // var. char usata per ricevere il bit
      int sign = 1;                         // int per rendere il numero positivo
      o negativo
      if (tag == Xc)                      // se riconosce che il valore
        appartiene alla coordinata X
      {
        if (sign < 0)
          sign = -sign;
        else
          sign = 1;
        if (sign == 1)
          X = tag;
        else
          X = tag - 100;
      }
    }
  }
}
```

```

12     valX = mySerial.read() * 256;      // ricompone il numero, muovendo il
byte di una posizione
13     valX = valX + mySerial.read();    // aggiunge il bit successivo
14
15     int checkSign = mySerial.read(); // funzione per verificare il segno
16     if (checkSign == 'n') {          // se negativo Processing manda una n
17         sign = -1;
18     }
19
20     valY *= sign;                 // moltiplicare il valore per il segno
21
22 } else if (tag == Yc) {           // processo analogo ma con la
coordinata Y
23
24     valY = mySerial.read() * 256;
25     valY = valY + mySerial.read();
26
27     int checkSign = mySerial.read();
28     if (checkSign == 'n') {
29         sign = -1;
30     }
31
32     valY *= sign;
33
34     // se riceve y vuol dire che il robot e' dentro il campo visivo, se
riceve x non lo e'
35     checkBounds = mySerial.read();
36
37     Serial.println(String(valY));
38
39 } else {
40     Serial.print("got message with unknown tag ");
41     Serial.println(tag);
42 }
43 }
44 }
```

- **positionLoop():** Siccome la velocità con cui vengono inviati i dati da Processing può variare, è meglio leggere i dati più volte in modo da essere sicuri di aver ricevuto entrambe le coordinate almeno una volta.

```

1 // aggiornare dati posizione
2 void positionLoop(int n) {
3     // aggiorna le coordinate
4     serialFlush();
5     delay(100);
6     for (int i = 0; i <= n; ++i) {
7         readCoordinates();
8     }
9
10    // se il robot e' fuori dal campo visivo del Kinect eseguire lostRobot
11    if (checkBounds == 'x') {
12        lostRobot(lastDistance);
13    }
14}

```

Nel caso in cui il robot esce dal campo visivo del Kinect viene chiamata la funzione *lostRobot()*. Questa funzione fa ruotare il robot di 180° e poi lo fa muovere in avanti fino a rientrare nel campo visivo del Kinect. Grazie a questa funzione si ha la certezza che il robot non si può perdere, ma soprattutto, si possono piantare semi anche in punti che sono al di fuori del campo visivo del Kinect e non si è limitati da esso. Il problema di questo metodo è che si suppone che quando il robot viene perso di vista, prima di perdersi era dentro il campo visivo, e quindi facendolo tornare dalla stessa direzione da cui era venuto prima o poi rientrerà nel campo visivo. Un metodo che funzionerebbe in qualsiasi caso, invece, e che mi è stato consigliato dai miei professori, è di far muovere il robot in una spirale. Anche se richiede più tempo, questo metodo funziona in qualsiasi caso, anche quando il robot parte da una posizione che è al di fuori del campo visivo del Kinect.

- **serialFlush():** È usata nella funzione precedente, e serve per pulire il “buffer” seriale.

```

2 void serialFlush() {
3     while (mySerial.available())
4         mySerial.read();
5 }

```

5.2.2 Rotary encoder

- **attachInterrupt():** Nella funzione *setup()* sono impostati due “interrupt”. Questa è una funzione molto utile per gestire nel background alcune routine senza interrompere il flusso del programma. Nel caso si manifesti un evento asincrono esterno come un cambiamento di stato su un pin (ad esempio se c’è una caduta di voltaggio), l’interrupt interrompe tutte le funzioni per portare l’attenzione su quella indicata nella funzione *interrupt()*. In questo caso, ogni volta che avviene un cambiamento di stato sul pin dell’encoder, viene eseguita la funzione *countLeft()* o *countRight()* a seconda del pin sul quale è avvenuto il cambiamento.

```

// La funzione richiede tre informazioni: Pin, funzione da eseguire, tipo di
// cambiamento
2 attachInterrupt(digitalPinToInterrupt(enc_l_pin), countLeft, CHANGE);
attachInterrupt(digitalPinToInterrupt(enc_r_pin), countRight, CHANGE);

```

- **countLeft()**: Questa funzione aggiorna una variabile che tiene conto del numero di giri che ha fatto ogni motore, e quindi la distanza che il robot ha percorso. Questa funzione non tiene conto della direzione in cui gira il motore.

```

1 void countLeft() { // per l'encoder del motore destro la funzione e' analoga
  enc_l++; // aggiunge 1 alla variabile
3 }

```

5.2.3 Movimento

- **drive()**: Imposta i motori per muoversi in avanti se $power \geq 0$ o di muoversi indietro se $power < 0$.

```

1 void drive(int power_a, int power_b) {
  // Limitare power da a -255 and 255
3   power_a = constrain(power_a, -255, 255);
  power_b = constrain(power_b, -255, 255);

5   // Direzione motore sinistro
7   if ( power_a < 0 ) {
      digitalWrite(ain1_pin, LOW);
      digitalWrite(ain2_pin, HIGH);
    } else {
      digitalWrite(ain1_pin, HIGH);
      digitalWrite(ain2_pin, LOW);
    }
11  // Direzione motore destro
15  if ( power_b < 0 ) {
      digitalWrite(bin1_pin, LOW);
      digitalWrite(bin2_pin, HIGH);
    } else {
      digitalWrite(bin1_pin, HIGH);
      digitalWrite(bin2_pin, LOW);
    }
21  // Impostare velocità
23  analogWrite(pwma_pin, abs(power_a));
  analogWrite(pwmb_pin, abs(power_b));
25 }

```

- **brake()**: Ferma entrambi i motori.

```

1 void brake() {
2     digitalWrite(ain1_pin, LOW); // Imposta tutti i pin a 0V
3     digitalWrite(ain2_pin, LOW);
4     digitalWrite(bin1_pin, LOW);
5     digitalWrite(bin2_pin, LOW);
6     analogWrite(pwma_pin, 0); // Imposta il segnale pwm a 0
7     analogWrite(pwmb_pin, 0);
}

```

- **driveStraight**: Questa funzione fa muovere il robot in avanti per una certa distanza, oppure, nel caso in cui il robot si era perso, fino a rientrare nel campo visivo del Kinect. Durante il movimento vengono corrette le velocità dei due motori rendendole il più possibile simili tra di loro: la funzione sfrutta i dati degli speed encoders per calcolare la differenza di velocità tra i due motori, se uno va più velocemente rispetto all'altro, al primo viene aumentata la velocità e all'altro viene diminuita.

```

void driveStraight(float dist, int power) {
    // numero di "ticks"
    unsigned long num_ticks_l;
    unsigned long num_ticks_r;

    // impostare potenza iniziale dei motori
    int power_l = motorPower;
    int power_r = motorPower;

    // usate per stabilire in quale senso si deve ruotare il motore per la
    // correzione
    unsigned long diff_l;
    unsigned long diff_r;

    // resettare contatori encoder
    enc_l = 0;
    enc_r = 0;

    // salvare contatore encoder precedente
    unsigned long enc_l_prev = enc_l;
    unsigned long enc_r_prev = enc_r;

    // calcolare di quanto devono girare i motori
    float num_rev = (dist * 10) / wheel_c; // Convert to mm
    unsigned long target_count = num_rev * countsPerRev;

    // se il robot e' fuori dal campo visivo tornare indietro fino a rientrarsi
    if (lostRobotBool) {
        Serial.println("Moving until robot is in");

        // muoversi in avanti finche' non viene stabilito che il robot e' rientrato
        // nel campo visivo
    }
}

```

```

1      while ( checkBounds == 'x' ) {
2
3      // Salvare numero di "ticks"
4      num_ticks_l = enc_l;
5      num_ticks_r = enc_r;
6
7      // Drive
8      drive(power_l, power_r);
9
10     // numero di ticks dall'ultima volta
11     diff_l = num_ticks_l - enc_l_prev;
12     diff_r = num_ticks_r - enc_r_prev;
13
14     // salvare numero di ticks per prossima volta
15     enc_l_prev = num_ticks_l;
16     enc_r_prev = num_ticks_r;
17
18     // se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
19     if ( diff_l < diff_r ) {
20         power_l -= motor_offset;
21         power_r += motor_offset;
22     }
23
24     // se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
25     if ( diff_l > diff_r ) {
26         power_l += motor_offset;
27         power_r -= motor_offset;
28     }
29
30     //Controllare che non ci sia un ostacolo
31     obstacle();
32
33     serialFlush();
34     delay(100);
35     for (int i = 0; i <= 50; ++i) {
36         readCoordinates();
37     }
38
39 } else {
40     // ruotare finche' uno dei motori non compie la distanza target
41     while ( (enc_l < target_count) && (enc_r < target_count) ) {
42
43         // Salvare numero di "ticks"
44         num_ticks_l = enc_l;
45         num_ticks_r = enc_r;
46
47         // Drive
48         drive(power_l, power_r);
49
50         // numero di ticks dall'ultima volta
51         diff_l = num_ticks_l - enc_l_prev;
52         diff_r = num_ticks_r - enc_r_prev;
53
54         // salvare numero di ticks per prossima volta
55         enc_l_prev = num_ticks_l;
56         enc_r_prev = num_ticks_r;

```

```

88     // se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
89     if ( diff_l < diff_r ) {
90         power_l -= motor_offset;
91         power_r += motor_offset;
92     }
93
94     // se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
95     if ( diff_l > diff_r ) {
96         power_l += motor_offset;
97         power_r -= motor_offset;
98     }
99
100    obstacle();
101}
102}
103
104 lostRobotBool = false;
105
106 // frenare
107 brake();
108}

```

- **turnRobot:** Molto simile a *driveStraight()*, ma invece di far muovere i motori nella stessa direzione, se la distanza del robot dalla destinazione è maggiore alla metà della sua larghezza, si muove solo un motore e l'altro rimane fisso, in caso contrario, i due motori si muovono in direzioni opposte a seconda del senso in cui deve ruotare il robot.

```

void turnRobot(float dist, int power, bool calcdestination) {
2    // Alzare l'aratro
3    if (aratroServo.read() < 150) {
4        aratroServo.attach(aratroPin);
5        aratroServo.write(180);
6        delay(2000);
7        aratroServo.detach();
8    }
9
10   unsigned long num_ticks_l;
11   unsigned long num_ticks_r;
12
13   // impostare potenza iniziale dei motori
14   int power_l = motorPower;
15   int power_r = motorPower;
16
17   // usate per stabilire in quale senso si deve ruotare il motore per la
18   // correzione
19   unsigned long diff_l;
20   unsigned long diff_r;
21
22   // resettare contatori encoder
23   enc_l = 0;
24   enc_r = 0;

```

```

24 // salvare contatore encoder precedente
26 unsigned long enc_l_prev = enc_l;
27 unsigned long enc_r_prev = enc_r;
28
29 // calcolare di quanto devono girare i motori
30 unsigned long target_count;
31 if (dd >= r) {
32     target_count = dist * tickPerCm * 2 * movementCorrection;
33 } else {
34     target_count = dist * tickPerCm * movementCorrection;
35 }
36
37 // ruotare finche' uno dei motori non compie la distanza target
38 while ( (enc_l < target_count) && (enc_r < target_count) ) {
39
40     // Salvare numero di "ticks"
41     num_ticks_l = enc_l;
42     num_ticks_r = enc_r;
43
44     // Se dd >= r ruotare facendo muovere solo un motore, in caso contrario,
45     // muoverli in direzioni opposte
46     if (dd >= r) {
47         if (senso == "left") {
48             drive(110, 0);
49         } else if (senso == "right") {
50             drive(0, 85);
51         }
52     } else {
53         //se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
54         if ( diff_l < diff_r ) {
55             if (left == true) {
56                 power_l += motor_offset;
57                 power_r -= motor_offset;
58
59             } else {
60                 power_l -= motor_offset;
61                 power_r += motor_offset;
62             }
63
64         // se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
65         if ( diff_l > diff_r ) {
66
67             if (left == true)
68                 power_l += motor_offset;
69                 power_r -= motor_offset;
70         }
71         if (senso == "left") {
72             drive(110, -85);
73         } else if (senso == "right") {
74             drive(-110, 85);
75         }
76     }
77
78     // numero di ticks dall'ultima volta

```

```
diff_l = num_ticks_l - enc_l_prev;
80 diff_r = num_ticks_r - enc_r_prev;

82 // salvare numero di ticks per prossima volta
83 enc_l_prev = num_ticks_l;
84 enc_r_prev = num_ticks_r;

86 // breve pausa per lasciare tempo ai motori per reagire
87 delay(20);
88 }

90 // Frenare
91 brake();
92
93 delay(1000);
94 if (calcdestination) {
95   calcDestination();
96 }
```

5.2.4 Diagramma di flusso

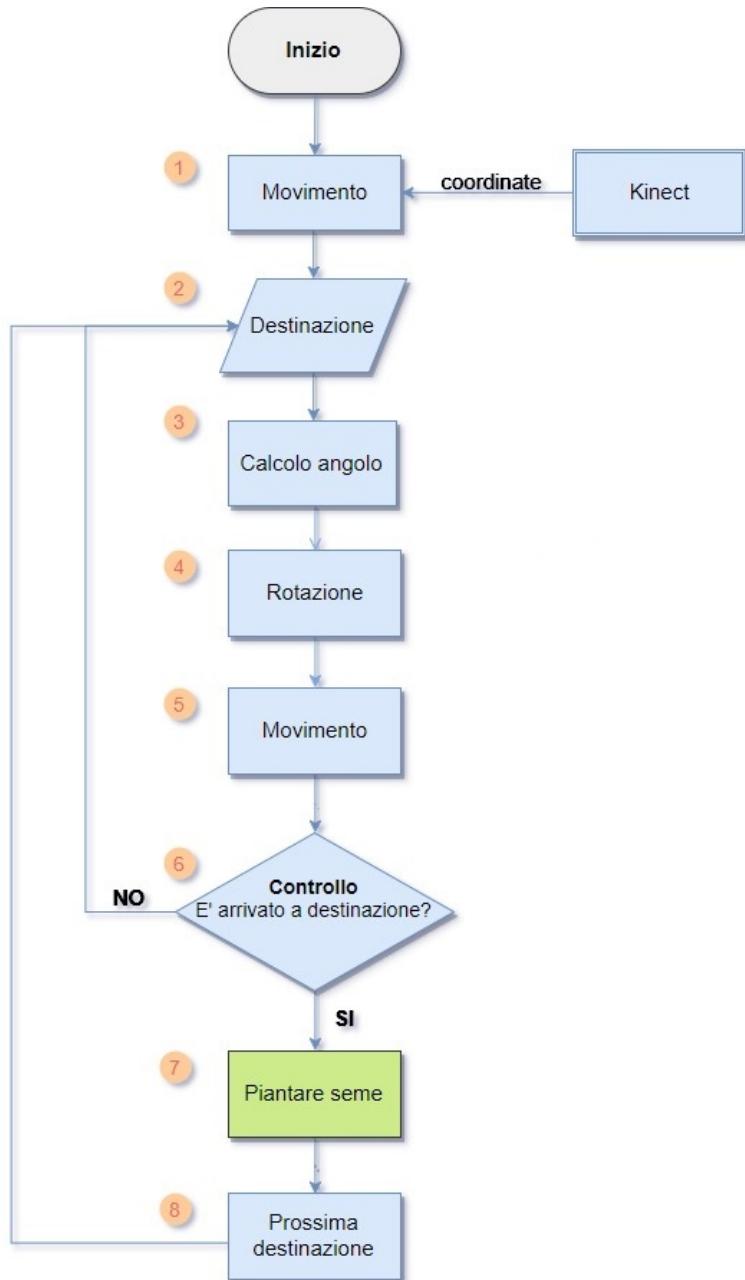


Figura 10: Diagramma di flusso

- Inizio:** Nella funzione *setup()* è dichiarata la funzione che funge da punto di partenza per il resto del programma. Nella funzione *go()* viene chiamata la funzione iniziale *getDirection()*.

```

1 void go() {
    getDirection();
}

```

- Movimento:** Questa è la prima fase che deve compiere il robot. Siccome l'unica informazione che può fornire il Kinect è la posizione del robot ad un dato istante, è necessario capire quale sia il suo orientamento iniziale, in modo da conoscere l'angolo di rotazione che lo conduce alla meta. Questa funzione consiste semplicemente nel far percorrere una certa distanza al robot in modo da calcolare la sua direzione grazie alla funzione *getDirection()* menzionata prima.

```

1 void getDirection() {
    Serial.println("Getting direction");
3
5 // Alzare l'aratro se esso e' abbassato
6 if (aratroServo.read() < 10) {
7     aratroServo.attach(aratroPin);
8     aratroServo.write(180);
9     delay(2000);
10    aratroServo.detach();
11
12    lastDistance = directionDistance;
13
14    positionLoop(positionLoopN);
15
16    // salvare coordinate della posizione iniziale
17    posI[0] = valX;
18    posI[1] = valY;
19
20    // muoversi in avanti per calcolare la direzione
21    driveStraight(directionDistance, motorPower);
22
23    positionLoop(positionLoopN);
24
25    // Salvare coordinate della posizione finale
26    posF[0] = valX;
27    posF[1] = valY;
28
29    calcAngolo();
}

```

- Destinazione:** Dopo aver scoperto la direzione del robot, vengono fatti i calcoli necessari per il suo spostamento. Per trovare di quanto deve ruotare il robot e in quale senso

(se a destra o a sinistra), occorre sapere le coordinate del punto nel quale deve dirigersi: le coordinate delle “destinazioni” sono salvate in un vettore `destArrayX[]` e `destArrayY[]`, rispettivamente per le coordinata *X* e la coordinata *Y* (lontananza dal Kinect).

```
1 int destArrayX[] = {x1, x2, x3, ...}; // coordinate x dei punti di arrivo
2 int destArrayY[] = {y1, y2, y3, ...}; // coordinate y dei punti di arrivo
```

- 4. Calcolo angolo:** Questa funzione è molto importante in quanto un piccolo errore nel calcolo dell’angolo di rotazione può conseguire ad una grande distanza del robot dal punto di destinazione. Nel caso “A”, in cui la distanza tra il punto di arrivo e il centro del robot è maggiore al raggio di rotazione del robot (equivale a metà della larghezza del robot), si calcola l’angolo tra il vettore direzione del robot e il vettore della direzione che dovrebbe avere per dirigersi verso la destinazione, muovendo solo un motore. Nel caso “B” invece, se la distanza tra il centro del robot e il punto di arrivo è minore del raggio di rotazione del robot, si calcola l’angolo tra la direzione del robot e il vettore che è diretto verso il punto di arrivo e si fa ruotare il robot sul suo asse.

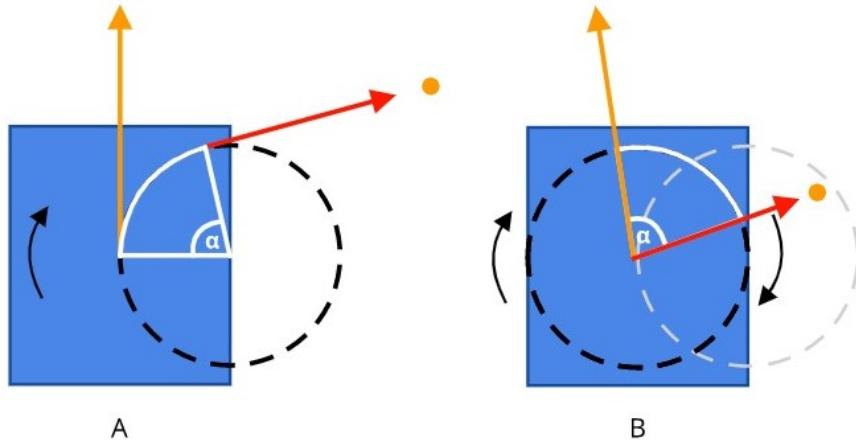


Figura 11: Schema in cui vengono raffigurati i due casi nel calcolo dell’angolo. Il cerchio in arancione indica il punto di arrivo, e α indica l’angolo di rotazione.

```
1 void calcAngolo() {
2     positionLoop(positionLoopN);
4
6     double direzioneY = (posF[1] - posI[1]);
6     double direzioneX = (posF[0] - posI[0]);
8
8     alpha = (M_PI / 2.0) - atan2(abs(direzioneY), abs(direzioneX));
10    // robot direction
```

```

12     double a1 = direzioneX;
13     double a2 = direzioneY;
14     // robot to destination
15     double b1 = destArrayX[destN] - valX;
16     double b2 = destArrayY[destN] - valY;
17
18     // CROSS
19     cross = b1 * a2 - a1 * b2;
20
21     if (cross < 0.0f) {
22         segno = -1;
23     } else {
24         segno = 1;
25     }
26
27     // calcolare coordinate (x0 e y0) del centro della circonferenza
28     if (direzioneX >= 0) {
29         y0 = valY - (segno * abs(r * sin(alpha)));
30     } else {
31         y0 = valY + (segno * abs(r * sin(alpha)));
32     }
33
34     if (direzioneY >= 0) {
35         x0 = valX + (segno * abs(r * cos(alpha)));
36     } else {
37         x0 = valX - (segno * abs(r * cos(alpha)));
38     }
39
40     // =====
41     // calcolo dei punti di tangenza
42     // =====
43
44     dx = x0 - destArrayX[destN];
45     dy = y0 - destArrayY[destN];
46     dd = sqrt(dx * dx + dy * dy);
47     // Quando dd < r a non dara' un risultato, quindi si calcola l'angolo con un'
48     // approssimazione
49     a = asin(r / dd);
50     b = atan2(dy, dx);
51     t = b - a;
52     x1 = r * sin(t) + x0;
53     y1 = -r * cos(t) + y0;
54     t = b + a;
55     x2 = -r * sin(t) + x0;
56     y2 = r * cos(t) + y0;
57
58     // Controllare quali coordinate sono piu' grandi
59     if (x1 < x2) {
60         xSin = x1;
61         ySin = y1;
62     }
63     xDest = x2;
64     yDest = y2;
65     } else {
66         xSin = x2;
67         ySin = y2;

```

```

66     xDest = x1;
68     yDest = y1;
70 }
71
72 if (cross > 0) {
73     if (destArrayY[destN] > y0) {
74         xT = xSin;
75         yT = ySin;
76     } else {
77         xT = xDest;
78         yT = yDest;
79     }
80 } else {
81     if (destArrayY[destN] > y0) {
82         xT = xDest;
83         yT = yDest;
84     } else {
85         xT = xSin;
86         yT = ySin;
87     }
88 // =====
89 // calcolo arco
90 // =====
91 if (dd >= 2 * r) {
92     deltaX = valX - x0;
93     deltaY = valY - y0;
94
95     deltaDestX = xT - x0;
96     deltaDestY = yT - y0;
97 } else { // se la distanza e' piu' piccola del raggio si fa un'
98     approssimazione
99     deltaX = direzioneX;
100    deltaY = direzioneY;
101
102    deltaDestX = destArrayX[destN] - x0;
103    deltaDestY = destArrayY[destN] - y0;
104 }
105 // viene usata la formula per calcolare l'angolo acuto tra due vettori
106 angle = acos((deltaX * deltaDestX + deltaY * deltaDestY) /
107               (sqrt(deltaX * deltaX + deltaY * deltaY) *
108                sqrt(deltaDestX * deltaDestX + deltaDestY * deltaDestY))) *
109    57.2958;
110
111 // calcolo della distanza che dovrà percorrere il robot
112 arco = 2 * 2 * M_PI * r * angle / 360;
113
114 if (cross < 0) {
115     senso = "left";
116 } else {
117     senso = "right";
118 }
119 turnRobot(arco , motorPower, true); // ruotare robot
120 }
```

5. **turnRobot:** Ora viene eseguita la funzione che fa ruotare il robot fino ad essere diretto verso la destinazione. Siccome la rotazione del robot sul suo asse (quindi i motori girano in direzioni opposte) presentava il problema di dover gestire la differenza di velocità dei due motori, ho deciso di far ruotare il robot facendo girare solo un motore, in questo modo il grado di rotazione del robot è sempre preciso, indipendentemente dalla sua velocità. È vero che si può compensare la differenza tra le velocità dei motori con lo stesso metodo che ho usato nella funzione *driveStraight()*, ma però per angoli molto piccoli i motori percorrono una distanza altrettanto piccola e non si fa in tempo a regolare le velocità.

```

1 void turnRobot(float dist, int power, bool calcdestination) {
2     // Alzare l'aratro
3     if (aratroServo.read() < 150) {
4         aratroServo.attach(aratroPin);
5         aratroServo.write(180);
6         delay(2000);
7         aratroServo.detach();
8     }
9
10    unsigned long num_ticks_l;
11    unsigned long num_ticks_r;
12
13    // impostare potenza iniziale dei motori
14    int power_l = motorPower;
15    int power_r = motorPower;
16
17    // usate per stabilire in quale senso si deve ruotare il motore per la
18    // correzione
19    unsigned long diff_l;
20    unsigned long diff_r;
21
22    // resettare contatori encoder
23    enc_l = 0;
24    enc_r = 0;
25
26    // salvare contatore encoder precedente
27    unsigned long enc_l_prev = enc_l;
28    unsigned long enc_r_prev = enc_r;
29
30    // calcolare di quanto devono girare i motori
31    unsigned long target_count;
32    if (dd >= r) {
33        target_count = dist * tickPerCm * 2 * movementCorrection;
34    } else {
35        target_count = dist * tickPerCm * movementCorrection;
36    }
37
38    // ruotare finche' uno dei motori non compie la distanza target
39    while ( (enc_l < target_count) && (enc_r < target_count) ) {
40        // Salvare numero di "ticks"
41        num_ticks_l = enc_l;
42        num_ticks_r = enc_r;
43
44        // Se dd >= r

```

```

45     if (dd >= r) {
46         if (senso == "left") {
47             drive(110, 0);
48         } else if (senso == "right") {
49             drive(0, 85);
50         }
51     } else {
52         //se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
53         if ( diff_l < diff_r ) {
54             if (left == true) {
55                 power_l += motor_offset;
56                 power_r -= motor_offset;
57             } else {
58                 power_l -= motor_offset;
59                 power_r += motor_offset;
60             }
61         }
62
63         // se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
64         if ( diff_l > diff_r ) {
65             if (left == true)
66                 power_l += motor_offset;
67             power_r -= motor_offset;
68         }
69         if (senso == "left") {
70             drive(110, -85);
71         } else if (senso == "right") {
72             drive(-110, 85);
73         }
74
75         // numero di ticks dall'ultima volta
76         diff_l = num_ticks_l - enc_l_prev;
77         diff_r = num_ticks_r - enc_r_prev;
78
79         // salvare numero di ticks per prossima volta
80         enc_l_prev = num_ticks_l;
81         enc_r_prev = num_ticks_r;
82
83         // breve pausa per lasciare tempo ai motori per reagire
84         delay(20);
85     }
86
87     // Frenare
88     brake();
89
90     delay(1000);
91     if (calcdestination) {
92         calcDestination();
93     }
94 }
```

6. Quello che resta da fare è calcolare la distanza del robot dalla destinazione, e muoversi in

avanti fino a raggiungere la destinazione.

```
1 void destinationMovement() {  
2     driveStraight(distance, motor_power); // movimento in avanti per "distance"  
3         metri  
4     control();  
5 }
```

7. **control()**: Se i calcoli e il movimento sono stati eseguiti correttamente, il robot dovrebbe essere giunto a destinazione. Per verificare ciò, si confronta la posizione attuale del robot con il punto di arrivo. Se la distanza tra questi due è minore di una certa tolleranza, si continua con il flusso del programma facendo cadere dei semi, tramite la rotazione della vite, e prendendo il prossimo punto di arrivo dal vettore, in caso contrario, si mantengono le coordinate dello stesso punto di arrivo e si ricomincia dal punto 2.

```
1 void control() {  
2     positionLoop(10);  
3     int a = valX - destArrayX[destN]; //differenza x  
4     int b = valY - destArrayY[destN]; //differenza y  
5  
6     int distance = sqrt((a * a) * (b * b)); //modulo vettore (a;b) in ticks  
7  
8     if (distance <= 10) {  
9         destN = destN + 1; //Prossima destinazione  
10    }  
11  
12    calcAngolo(); // calcolare l'angolo di rotazione per la prossima destinazione  
13 }
```

6 Il robot

La costruzione del robot è stata probabilmente la parte meno complicata di questo lavoro. La struttura di base del robot è fatta di alluminio; a questa sono attaccati i due motori, le ruote e i cingoli. Ho deciso di acquistare una base prefabbricata poichè non disponevo ne delle conoscenze, né del materiale necessario per costruire da zero una base per un robot mobile che sia adatta ai requisiti di questo progetto. La scelta di usare cingoli piuttosto che ruote è dovuta soprattutto alla migliore trazione che genera su terra e alla possibilità di ruotare il robot sul suo asse.

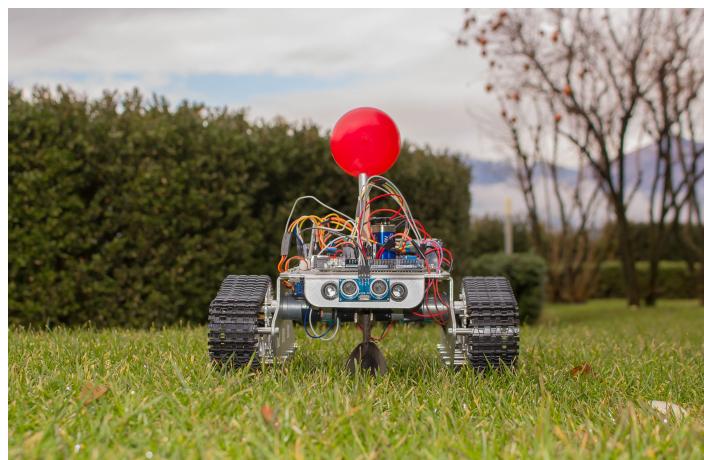


Figura 12: Foto frontale del robot

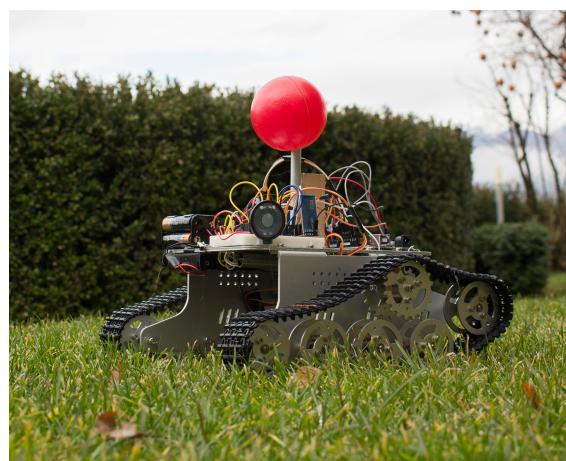


Figura 13: Foto laterale del robot

Per piantare i semi il robot utilizza un piccolo aratro per scavare nella terra e un cilindro con una vite all'interno, che girando fa cadere dei semi. L'aratro è stato costruito usando una lastra di ferro, ed è agganciato ad un perno che gli permette di alzarsi e abbassarsi con l'aiuto di un servomotore. Quando l'aratro è completamente abbassato, esso è sostenuto dalla base superiore del robot, in modo che la resistenza della terra sia contrastata dalla struttura del robot e non dal servomotore.

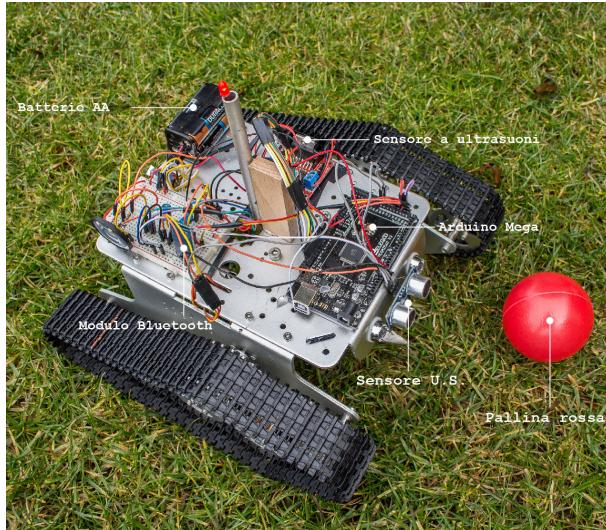


Figura 14: Parte superiore del robot

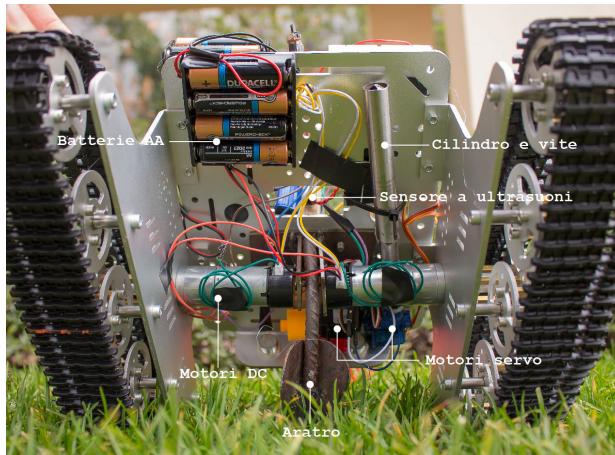


Figura 15: Parte inferiore del robot

7 Cosa ho appreso dal LAM

Questo Lavoro di Maturità è stato un lungo ma interessante percorso da cui ho appreso molto. Come tutti i progetti di ricerca questo Lavoro di Maturità non mancava di difficoltà, sia nel-

l'apprendere un concetto complicato per portare avanti il progetto, sia per i problemi riscontrati nella fase di prototipazione. In questi casi, tutto il progetto si ferma e tutto dipende dalla loro risoluzione. Perciò in questo capitolo vorrei elencare le cose che ho appreso durante il LAM grazie ai miei errori.

7.1 Ricerca

- **Dividere il problema in problemi più piccoli:**

Il Lavoro di maturità di robotica, al contrario di altri, è un progetto che inizialmente non ha un chiaro punto di arrivo. Il punto di partenza idealmente deve essere un compito, al quale poi si aggiungono, man mano che le conoscenze crescono, nuove parti fino a raggiungere il risultato desiderato. Nel mio caso, ad esempio, ero partito con una vaga idea di quello che avrei voluto ottenere alla fine, e durante la fase di costruzione ho modificato molti aspetti del progetto. La prima cosa da fare era di ordinare le diverse fasi del progetto secondo la loro importanza. Il progetto per un automa per la semina ad esempio, potrebbe avere il piano di lavoro seguente:

1. **Costruire la base per un robot mobile:** Siccome il robot deve muoversi, decido di concentrarmi innanzitutto sulla parte mobile, poiché tutte le altre funzioni servono a regolarne il movimento. In questo modo posso quindi testarle.
2. **Connessione Bluetooth con il PC:** Ora che ho un robot che può muoversi, ho bisogno che lo faccia in modo autonomo; per fare questo è necessario conoscerne la posizione e comunicarla all'Arduino, quindi mi concentro sulla connessione tra il modulo Bluetooth e il computer.
3. **Localizzazione robot con Kinect e Processing:** Ora che posso comunicare le coordinate dal PC ad Arduino, mi resta solo di localizzare il robot con la telecamera RGB e di trovare le sue coordinate.
4. **Algoritmo su Arduino IDE:** Dopo aver completato le parti essenziali al funzionamento del robot, finalmente, posso dedicarmi alla fase di scrittura del programma che Arduino deve seguire.

Come si può notare da questo esempio, la costruzione di un robot non è un grande lavoro che si svolge in un colpo solo, ma è una serie di piccole parti che messe insieme portano ad un risultato finale. La cosa più importante infatti è cercare sempre di dividere un problema in problemi più piccoli in modo da potersi concentrare solo su piccoli compiti e riuscire ad affrontare il problema più grande in maniera più semplice. Nel caso di questo LAM ogni fase poteva durare da poche ore a diversi giorni, ma non passavo al compito successivo fino al completamento di quello precedente, contrariamente alla parte scritta per la quale non ho seguito nessuno schema specifico ma potevo scrivere ogni capitolo del testo indipendentemente dagli altri. Personalmente trovo che questa metodologia di lavoro sia molto utile in quanto posso concentrarmi su un compito alla volta senza pensare alla complessità del progetto.

- **Cercare informazioni su Internet**

Ad ogni fase di quelle descritte nei paragrafi precedenti si accompagnava un lavoro di ricerca, in quanto per ogni fase di lavoro dovevo apprendere i concetti necessari per completarla. Ad esempio nella fase in cui dovevo localizzare il robot con l'uso di Kinect, ho dovuto cercare tutto quello che mi serviva su Internet, e quindi tematiche come ad esempio installare i driver corretti per il funzionamento del Kinect con il PC oppure come ottenere le diverse informazioni partendo dalle immagini del Kinect, ecc. Qui di seguito Alcuni consigli che hanno facilitato le ricerche di informazioni su Internet di temi che riguardano la programmazione o la robotica in generale.

– **Uso di parole chiave**

Per ricerche riguardanti soprattutto la programmazione è importante selezionare le parole più importanti e tralasciare quelle irrilevanti alla domanda, e specificare il linguaggio di programmazione, in quanto la ricerca riporta risultati più pertinenti quando si usano parole chiave specifiche. Un modo per indicare al motore di ricerca che si cerca un “tutorial”, inoltre, è di iniziare la domanda con “How to” nel caso della lingua inglese o un corrispettivo in italiano.

– **Generalizzare la domanda**

Più la domanda è generale e più vaghi saranno i risultati, viceversa più la domanda è specifica e meno saranno i risultati pertinenti. L’ideale dunque, sarebbe trovare un equilibrio tra specificità e quantità di parole chiave. Quello che personalmente trovo più efficace è di iniziare con una ricerca piuttosto specifica, poi se i risultati non contengono una risposta soddisfacente generalizzare la domanda per comprendere una gamma più larga di risultati.

– **Usare l’inglese**

Ovviamente si deve conoscere la lingua, però se possibile è sempre meglio usare l’inglese nelle ricerche su Internet, poichè esiste molto più materiale in questa lingua che in italiano, e quindi spesso è più facile trovare una risposta adatta.

7.2 Programmazione

7.2.1 Logica del programma

Come nella fase precedente, anche nella parte di programmazione è molto importante dividere il programma in funzioni che hanno un compito logico. Nel mio progetto, ad esempio, ho creato una funzione per ogni movimento di ogni motore: una per il movimento in avanti per il motore destro, una per il motore sinistro, una per il movimento indietro per il motore destro, ecc. In questo modo risulta più facile la gestione del programma, e si semplifica anche la fase di *Debugging*. Inoltre come già ci è stato chiesto per la redazione dello scritto, è molto utile definire un diagramma di flusso prima di cominciare a programmare, in modo da avere un piano di lavoro e non perdersi nella grandezza del progetto.

7.2.2 Scrittura del codice

- **Pseudocode:** Ragionare direttamente sul codice in Arduino IDE è piuttosto complicato. Per avere un’idea migliore del programma che si sta scrivendo invece, è utile descriverlo

prima a parole senza tenere conto della sintassi del linguaggio di programmazione, in questo modo si possono notare subito eventuali errori nella logica del programma; dopodichè la scrittura del programma vero è proprio non sarà altro che l'esportazione dello "pseudocode" nel linguaggio di programmazione, in questo caso C++.

- **Nomenclatura:** Nella scrittura del programma, soprattutto nella nomenclatura delle funzioni e delle variabili, è molto importante usare dei nomi che siano facili da ricordare ma soprattutto che siano coerenti alla funzione o variabile a cui si riferiscono. Se ad esempio devo dare un nome ad una funzione che fa muovere un motore in avanti, la chiamerei piuttosto *motoreDxAvanti()* che *andareInAvantiMotDest()*. Un altro aspetto importante è l'uso di uno stile costante nella nomenclatura, ossia di trovare una sorta di *sintassi* e mantenerla uguale. Questo, a mio avviso, contribuisce a migliorare la comprensione del programma. Personalmente trovo comodo usare la cosiddetta nomenclatura *camelBack* che consiste nell'iniziare le parole nuove in maiuscolo ma senza usare spazi o segni, in questo modo è molto più facile leggere i nomi e non si devono usare trattini o segni particolari.
- **Uso efficace di commenti:** Lo scopo principale dei commenti nei linguaggi di programmazione è di migliorare la comprensione del codice sia per l'autore che per le altre persone che devono comprenderlo. Dunque, è molto importante sfruttare i commenti ogni volta che una linea di codice non è molto chiara e spiegarla in modo comprensibile ad un essere umano.

7.2.3 Debugging

Dopo aver finito la scrittura di una funzione o una parte di un programma, nella maggior parte dei casi il codice risulta avere errori che spesso sono banali come una dimenticanza del punto e virgola, ma nel restante dei casi sembrano celarsi nella complessità di funzioni. Una delle abilità che ho accresciuto maggiormente portando avanti un progetto come questo, è la capacità di scovare errori, sia nei programmi che scrivo, ma anche nei diversi ambiti della mia vita; una cosa che forse non viene spesso espressa infatti, è che la programmazione non serve solo per creare nuove applicazioni, software, ecc. ma sviluppa un modo di pensare propenso a trovare soluzioni a qualsiasi problema. In merito a questo avrei alcuni consigli per evitare il più possibile errori in primo luogo, e poi per risolverli con il minore sforzo e nel minor tempo possibile.

- **Fare un cambiamento alla volta:** Una cosa che trovo molto utile durante la programmazione per facilitare la risoluzione degli errori è di fare solo un cambiamento o aggiunta nel codice alla volta prima di testarne il funzionamento. Questo rende la localizzazione degli errori molto più semplice, in quanto sapendo che se prima il programma funzionava correttamente, il problema sarà sicuramente nella parte aggiunta o modificata.
- **Rubber Duck Debugging:** Il metodo di debugging della "papera di gomma" consiste nello spiegare il proprio programma riga per riga ad una papera di gomma. Ovviamente la papera di gomma è usata solo come expediente, ma fattostà che questo metodo è molto efficace in quanto ci obbliga a visualizzare il nostro programma da un altro punto di vista e eventualmente trovare la parte del programma che provoca l'errore.
- **Uso della console per verifica:** La console è uno strumento fondamentale nella fase di Debugging. Essa ci permette di "comunicare" con il programma, in quanto si può verificare

il funzionamento delle diverse funzioni tramite dei messaggi. Se si vuole verificare il valore di una variabile per esempio, basta “stamparla” via *Serial.print()* e ci appare nel Serial Monitor di Arduino IDE.

- **Annotare il procedimento che porta alla risoluzione del problema:** Occasionalmente il processo per la risoluzione di un problema complesso può durare molto tempo e estendersi su diverse sessioni di lavoro; per questo è utile annotarsi tutti i diversi tentativi e i risultati che si ottengono in modo da identificare il problema più velocemente.

8 Limiti e possibili miglioramenti

Pur essendo un progetto di modesta scala, nulla vieta che ci possa essere una sua applicazione a livello commerciale. Tuttavia, per via della natura di questo progetto, sia tempo che denaro sono limitati e lo sono anche le mie conoscenze nell’ambito della robotica. Il primo limite che presenta questo automa è il vincolo che pone il Kinect come metodo di localizzazione. Il programma in Processing per Kinect è stato creato in modo da localizzare un oggetto di un colore predeterminato, se esso viene cambiato o se le condizioni di illuminazione vengono modificate notevolmente, il Kinect non sarà più in grado di localizzare il robot. Questo problema potrebbe essere risolto trovando un metodo di localizzazione sostitutivo. Se non serve un’alta accuratezza si potrebbe pensare di usare il GPS come mezzo di localizzazione, anche se con un metodo del genere l’imprecisione può essere addirittura nell’ordine di una decina di metri.

Oltre al metodo di localizzazione, ovviamente andrebbero migliorati molti aspetti dell’automma, sia a livello di componenti che a livello di algoritmi. Le prime cose che andrebbero migliorate sono le dimensioni e i materiali del robot: siccome questo progetto è stato pensato come robot agricolo, esso ha bisogno di potenza e di robustezza per riuscire a lavorare anche su terreni difficili; potrebbe addirittura essere usato un trattore come base del robot. La seconda cosa su cui si potrebbe lavorare è la logica che guida il robot, ovvero l’algoritmo: si potrebbero aggiungere nuove funzioni come una piattaforma per controllare il robot, la capacità di monitorare le condizioni del suolo, la capacità di auto-ricaricarsi, ecc. Insomma, se si possiedono tempo e conoscenze a sufficienza, l’unico limite è quello fisico.

9 Possibili problematiche

Nel corso della storia, il settore agricolo ha subito numerose innovazioni. Se si considerano le tecniche agricole che esistevano due secoli fa, si fa fatica ad immaginare che con esse saremmo arrivati al livello di sviluppo odierno, dove i campi di coltivazione possono estendersi per centinaia di ettari. L’invenzione delle macchine agricole come i trattori ha permesso la coltivazione e la gestione di terreni molto estesi, anche se ad usare queste macchine restano sempre esseri umani che compiono comunque uno sforzo: le macchine prendono il posto dei muscoli nella coltivazione di terre molto estese, ma quest’ultime hanno comunque bisogno di essere curate. Possiamo immaginare, dunque, che effetto avrebbe l’implementazione dei robot nell’agricoltura. In base a quello che abbiamo osservato fino ad ora, spesso hanno addirittura una precisione maggiore rispetto agli esseri umani.

Posso immaginare, tuttavia, che la presenza dei robot nel settore agricolo possa portare anche a delle conseguenze negative. Un fenomeno molto presente al giorno d’oggi è la perdita di posti

di lavoro per via dell'automatizzazione. Le agevolazioni che ci forniscono i robot sono inevitabilmente accompagnate da una sostituzione di lavori che venivano eseguiti dall'uomo. È comunque vero, però, che si aprirebbero nuove opportunità di lavoro nel settore informatico e della fabbricazione dei robot. Un altro possibile problema può essere legato al fatto che l'innovazione è accompagnata da un fenomeno di estensione; l'essere umano, per natura, quando qualcosa gli risulta più semplice o più conveniente, tende ad usarla più intensamente, e forse, disponendo di macchine che possono compiere il lavoro al posto suo, vorrà coltivare sempre più terre e più intensamente, provocando allo stesso tempo danni all'ambiente, in quanto non rispetterebbe più l'equilibrio della natura. Personalmente ritengo che l'innovazione che porterebbero i robot abbia un grande potenziale nel migliorare il settore agricolo e renderlo più sostenibile. Questo però avverà solo nel caso in cui ci sia un cambiamento di mentalità delle persone che utilizzano queste tecnologie; si devono sensibilizzare le persone all'uso di queste nuove tecnologie e alla delicatezza della natura, che potrebbe essere facilmente compromessa dallo sviluppo umano.

10 Conclusione

Grazie a questo lavoro di maturità ho compreso l'importanza e l'efficacia della robotica nelle nostre vite. La robotica, infatti, ha portato a risolvere problemi che l'uomo da solo non avrebbe potuto affrontare. Spesso i robot vengono associati al loro ruolo nella produzione industriale, alle loro applicazioni nel campo militare e ai veicoli autonomi, quando invece trascuriamo attività fondamentali per l'uomo, come l'agricoltura. Perciò mi premeva sottolineare in questo lavoro l'importanza dello sviluppo della robotica nel settore agricolo. A mio avviso, la robotica darà un grande contributo allo sviluppo dell'agricoltura, anche se mi rendo conto che questo sviluppo dovrà avvenire nel modo più cauto possibile per ridurre al massimo l'impronta ecologica dell'uomo. Ho appreso molto da questo lavoro, sia riguardo la programmazione e la robotica, sia in termini di capacità di gestione di un lavoro a lunga durata. La parte più impegnativa è stata senz'altro quella di programmazione, in quanto ho dovuto apprendere la maggior parte dei concetti necessari alla programmazione del robot, e la risoluzione dei problemi che incontravo durante la programmazione spesso richiedeva molto tempo. Anche la stesura dello scritto ha presentato alcune difficoltà; durante la redazione dello scritto, infatti, ho dovuto apporre molte modifiche al testo per renderlo comprensibile anche ai lettori meno familiari con gli argomenti presentati. Posso ritenermi molto soddisfatto dell'esito di questo lavoro, tenendo conto di tutto quello che ho appreso e per il fatto che mi sento molto più confidente nell'affrontare argomenti che riguardano la programmazione e la robotica in generale.

Riferimenti bibliografici

- [1] Michael Margolis, *Arduino Cookbook*, O'Reilly Media; 2nd edition, 2011.
- [2] SparkFun <https://learn.sparkfun.com/tutorials/connecting-arduino-to-processing/all>
- [3] Stackexchange <https://arduino.stackexchange.com/questions/34789/issues-using-hc-06-bluetooth-module-with-arduino> (Luglio 2017)

- [4] Nicholas Bertagnolli http://www.nbertagnolli.com/jekyll/update/2015/10/13/Object_Tracking.html# (Luglio 2017)
- [5] Arduino Playground <http://playground.arduino.cc/>
- [6] Nature <https://www.nature.com/articles/544S21a> (Ottobre 2017)
- [7] Daniel Shiffman - Getting Started with Kinect and Processing <http://shiffman.net/p5/kinect/> (Agosto 2017)
- [8] Vsblogs <https://vsblogs.wordpress.com/2012/02/09/select-the-right-battery-for-your-robot-dc-motors-part-1-of-2/> (Luglio 2017)
- [9] Kotaku <https://kotaku.com/5576002/here-are-kinects-technical-specs> (Dicembre 2017)
- [10] Maffucci <http://www.maffucci.it/2012/06/11/appunti-su-arduino-interrupts/> (Dicembre 2017)
- [11] ShawnHymel <https://gist.github.com/ShawnHymel/1de08ffaca990b65fade81cb8d01a44a> (Dicembre 2017)
- [12] SparkFun - Youtube <https://www.youtube.com/watch?v=oLBYHbLO8W0> (Dicembre 2017)
- [13] La copertina è stata creata con l'uso di illustrazioni di: macrovector / Freepik & Harryarts / Freepik

11 Allegati

Codice Arduino:

```
#include <SoftwareSerial.h> // Libreria per comunicazione Seriale
2 #include <math.h>
# include <Servo.h> // Libreria per i motori servo
4
#define HEADER '|' // dichiarazione header per riconoscere inizio messaggio
6 #define Xc 'N' // dichiarazione di tag per riconoscere valore X di coordinate
#define Yc 'M' // dichiarazione di tag per riconoscere valore Y di coordinate
8 #define MESSAGE_BYTES 7 // Il numero di byte totali che vengono mandati dal PC via
    seriale
#define trigPin 38
10 #define echoPin 39

12 SoftwareSerial mySerial(50, 48); // dichiarare pin per comunicazione bluetooth RX e
    TX del Mega non funzionano, usare RX=50 e TX=48
Servo semeServo;
14 Servo aratroServo;

16 // Parametri
const int driveDistance = 200;      // in cm
18 const int directionDistance = 20; //distanza del movimento per calcolare la
    direzione

20 const int motorPower = 80;          // 0-255
const int motor_offset = 1;           // Differenza per la regolazione dei motori
22 const int wheel_d = 55;            // diametro ruote(mm)
const float wheel_c = PI * wheel_d; // Circonferenza ruote (mm)
24 const int countsPerRev = 75;
const float anglePerTick = 1.148;
26 const float movementCorrection = 1.0;
const float tickPerCm = 4.340579207; // numero di segnali degli speed encoder ogni
    cm
28 const double r = 11.5; // raggio della circonferenza di rotazione
const float turn180 = r * 2 * 2 * M_PI * 0.5;
30 const int positionLoopN = 80;

32 // Pins
const int enc_l_pin = 2;
34 const int enc_r_pin = 3;
const int pwma_pin = 13;
36 const int ain1_pin = 12;
const int ain2_pin = 11;
38 const int pwmb_pin = 8;
const int bin1_pin = 10;
40 const int bin2_pin = 9;

42 const int aratroPin = 5;
const int semePin = 6;
44
// Variabili globali
46 volatile unsigned long enc_l = 0; // valore speed encoder
volatile unsigned long enc_r = 0;
48 int valY; //variabile coordinata Y
```

```

50   int valX; // variabile coordinata x
51   char checkBounds = 'y';
52   int destArrayX[] = {0, 20}; // coordinate x dei punti di arrivo
53   int destArrayY[] = {100, 80}; // coordinate y dei punti di arrivo
54   int destN = 0; // destination array index
55   float distance = 0; // distanza da percorrere dai motori
56   float lastDistance = 20; // salva l'ultima distanza percorsa
57
58   // calcAngolo()
59   double direzioneX = 0;
60   double direzioneY = 0;
61
62   double deltaX = 0;
63   double deltaY = 0;
64
65   double deltaDestX = 0;
66   double deltaDestY = 0;
67
68   double xT = 0;
69   double yT = 0;
70
71   double x0 = 0;
72   double y0 = 0;
73
74   double alpha = 0;
75   double cross = 0;
76   double angle = 0;
77   int segno = 1;
78   String senso = "left";
79   double dx, dy, dd, a, b, t, x1, y1, x2, y2, xSin, xDest, ySin, yDest;
80   double arco;
81
82   //posizioni - getDirection()
83   int posI[2]; // [X,Y]
84   int posF[2];
85
86   bool left = true;
87   bool lostRobotBool = false;
88
89   void setup() {
90
91     // Debug
92     Serial.begin(9600);
93
94     // Impostare i pin
95     pinMode(enc_l_pin, INPUT);
96     pinMode(enc_r_pin, INPUT);
97     pinMode(pwma_pin, OUTPUT);
98     pinMode(ain1_pin, OUTPUT);
99     pinMode(ain2_pin, OUTPUT);
100    pinMode(pwmb_pin, OUTPUT);
101    pinMode(bin1_pin, OUTPUT);
102    pinMode(bin2_pin, OUTPUT);
103    pinMode(trigPin, OUTPUT);
104    pinMode(echoPin, INPUT);

```

```

106 Serial.begin(9600); // inizializzare comunicazione seriale con console
mySerial.begin(9600); // inizializzare comunicazione seriale con modulo bluetooth
    (Processing su PC)
108 Serial.println("You're connected via Bluetooth"); // messaggio per indicare
    connessione a BT
110
112 // Interrupts per i pin degli speed encoder
attachInterrupt(digitalPinToInterrupt(enc_l_pin), countLeft, CHANGE);
attachInterrupt(digitalPinToInterrupt(enc_r_pin), countRight, CHANGE);
114
116 delay(1000);
118
120 // Iniziare il programma
go();
122
124 // TEST per verificare visivamente a quanto ammonta l'errore durante la rotazione
// turnRobot(turn180, motorPower, false);
126 }
128
130 void loop() {
    // Non fare nulla
}
132
134 // *****
136 // GO
138 // Imposta la prima funzione da eseguire
// *****
140 void go() {
    getDirection();
}
142
144 // *****
146 // GET DIRECTION
148 // movimento per calcolare direzione
// *****
150 void getDirection() {
    Serial.println("Getting direction");
}
152
154 // Alzare l'aratro
if (aratroServo.read() < 10) {
    aratroServo.attach(aratroPin);
    aratroServo.write(180);
    delay(2000);
    aratroServo.detach();
}
156
158 lastDistance = directionDistance;
positionLoop(positionLoopN);
160
162 // Salvare coordinate della posizione iniziale
posI[0] = valX;
posI[1] = valY;
164
166 driveStraight(directionDistance, motorPower);

```

```

positionLoop(positionLoopN);

160
// Salvare coordinate della posizione finale
162 posF[0] = valX;
posF[1] = valY;

164
Serial.println("initial: " + String(posI[0]) + ", " + String(posI[1]));
166 Serial.println("final: " + String(posF[0]) + ", " + String(posF[1]));

168 calcAngolo();
}
170
// *****
172 // CALC ANGOLO
// calcola angolo di rotazione
174 // *****
void calcAngolo() {

176
positionLoop(positionLoopN);

178
double direzioneY = (posF[1] - posI[1]);
double direzioneX = (posF[0] - posI[0]);

182 alpha = (M_PI / 2.0) - atan2(abs(direzioneY), abs(direzioneX));

184 // robot direction
double a1 = direzioneX;
double a2 = direzioneY;
// robot to destination
188 double b1 = destArrayX[destN] - valX;
double b2 = destArrayY[destN] - valY;

190
// CROSS
192 cross = b1 * a2 - a1 * b2;

194 if (cross < 0.0f) {
    segno = -1;
} else {
    segno = 1;
}

200 // calcolare coordinate (x0 e y0) del centro della circonferenza
if (direzioneX >= 0) {
    y0 = valY - (segno * abs(r * sin(alpha)));
} else {
    y0 = valY + (segno * abs(r * sin(alpha)));
}

206
if (direzioneY >= 0) {
    x0 = valX + (segno * abs(r * cos(alpha)));
} else {
    x0 = valX - (segno * abs(r * cos(alpha)));
}

212
// =====
214

```

```

216 // calcolo dei punti di tangenza
217 // =====
218 dx = x0 - destArrayX[destN];
219 dy = y0 - destArrayY[destN];
220 dd = sqrt(dx * dx + dy * dy);
221 // Quando dd < r a non dara' un risultato, quindi si calcola l'angolo con un'
222 // approssimazione
223 a = asin(r / dd);
224 b = atan2(dy, dx);
225 t = b - a;
226 x1 = r * sin(t) + x0;
227 y1 = -r * cos(t) + y0;
228 t = b + a;
229 x2 = -r * sin(t) + x0;
230 y2 = r * cos(t) + y0;
231
232 // Controllare quali coordinate sono piu' grandi
233 if (x1 < x2) {
234     xSin = x1;
235     ySin = y1;
236
237     xDest = x2;
238     yDest = y2;
239 } else {
240     xSin = x2;
241     ySin = y2;
242
243     xDest = x1;
244     yDest = y1;
245 }
246
247 if (cross > 0) {
248     if (destArrayY[destN] > y0) {
249         xT = xSin;
250         yT = ySin;
251     } else {
252         xT = xDest;
253         yT = yDest;
254     }
255 } else {
256     if (destArrayY[destN] > y0) {
257         xT = xDest;
258         yT = yDest;
259     } else {
260         xT = xSin;
261         yT = ySin;
262     }
263 }
264 // =====
265 // calcolo arco
266 // =====
267 if (dd >= 2 * r) {
268     deltaX = valX - x0;
269     deltaY = valY - y0;

```

```

270     deltaDestX = xT - x0;
272     deltaDestY = yT - y0;
273 } else { // se la distanza e' piu' piccola del raggio si fa un'approssimazione
274     deltaX = direzioneX;
275     deltaY = direzioneY;
276
277     deltaDestX = destArrayX[destN] - x0;
278     deltaDestY = destArrayY[destN] - y0;
279 }
280 // viene usata la formula per calcolare l'angolo acuto tra due vettori
281 angle = acos((deltaX * deltaDestX + deltaY * deltaDestY) /
282             (sqrt(deltaX * deltaX + deltaY * deltaY) *
283              sqrt(deltaDestX * deltaDestX + deltaDestY * deltaDestY))) * 57.2958;
284
285 // calcolo della distanza che dovrà percorrere il robot
286 arco = 2 * 2 * M_PI * r * angle / 360;
287
288 if (cross < 0) {
289     senso = "left";
290 } else {
291     senso = "right";
292 }
293
294 turnRobot(arco , motorPower, true); // ruotare robot
295 }
296
297 // *****
298 // TURN ROBOT
299 // ruotare il robot per dirigersi verso
300 // la destinazione
301 // *****
302 void turnRobot(float dist, int power, bool calcdestination) {
303     // Alzare l'aratro
304     if (aratroServo.read() < 150) {
305         aratroServo.attach(aratroPin);
306         aratroServo.write(180);
307         delay(2000);
308         aratroServo.detach();
309     }
310
311     unsigned long num_ticks_l;
312     unsigned long num_ticks_r;
313
314     // impostare potenza iniziale dei motori
315     int power_l = motorPower;
316     int power_r = motorPower;
317
318     // usate per stabilire in quale senso si deve ruotare il motore per la correzione
319     unsigned long diff_l;
320     unsigned long diff_r;
321
322     // resettare contatori encoder
323     enc_l = 0;
324     enc_r = 0;

```

```

326 // salvare contatore encoder precedente
327     unsigned long enc_l_prev = enc_l;
328     unsigned long enc_r_prev = enc_r;
329
330 // calcolare di quanto devono girare i motori
331     unsigned long target_count;
332     if (dd >= r) {
333         target_count = dist * tickPerCm * 2 * movementCorrection;
334     } else {
335         target_count = dist * tickPerCm * movementCorrection;
336     }
337
338 // ruotare finche' uno dei motori non compie la distanza target
339     while ( (enc_l < target_count) && (enc_r < target_count) ) {
340
341         // Salvare numero di "ticks"
342         num_ticks_l = enc_l;
343         num_ticks_r = enc_r;
344
345         // Se dd >= r
346         if (dd >= r) {
347             if (senso == "left") {
348                 drive(110, 0);
349             } else if (senso == "right") {
350                 drive(0, 85);
351             }
352         } else {
353             //se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
354             if ( diff_l < diff_r ) {
355                 if (left == true) {
356                     power_l += motor_offset;
357                     power_r -= motor_offset;
358
359                 } else {
360                     power_l -= motor_offset;
361                     power_r += motor_offset;
362                 }
363             }
364
365             // se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
366             if ( diff_l > diff_r ) {
367
368                 if (left == true)
369                     power_l += motor_offset;
370                     power_r -= motor_offset;
371                 }
372                 if (senso == "left") {
373                     drive(110, -85);
374                 } else if (senso == "right") {
375                     drive(-110, 85);
376                 }
377             }
378
379         // numero di ticks dall'ultima volta
380         diff_l = num_ticks_l - enc_l_prev;
381         diff_r = num_ticks_r - enc_r_prev;

```

```

382     // salvare numero di ticks per prossima volta
384     enc_l_prev = num_ticks_l;
385     enc_r_prev = num_ticks_r;
386
387     // breve pausa per lasciare tempo ai motori per reagire
388     delay(20);
389 }
390
391     // Frenare
392     brake();
393
394     delay(1000);
395     if (calcdestination) {
396         calcDestination();
397     }
398 }
399
400 // *****
401 //          CALC DESTINATION
402 // calcola il modulo del vettore destinazione
403 // *****
404 void calcDestination() {
405     positionLoop(positionLoopN);
406
407     float a = valX - destArrayX[destN]; //differenza x
408     float b = valY - destArrayY[destN]; //differenza y
409
410     distance = sqrt((a * a) + (b * b)); //modulo vettore (a;b) in ticks
411     Serial.println("distance to dest.: " + String(distance));
412
413     destinationMovement(distance, true);
414 }
415
416 // *****
417 //          DESTINATION MOVEMENT
418 //      movimento fino a destinazione
419 // *****
420 void destinationMovement(float dist, bool controlBool) {
421     aratroServo.attach(aratroPin);
422     aratroServo.write(90);
423     delay(1000);
424     aratroServo.detach();
425
426     lastDistance = dist;
427
428     if (controlBool) {
429         positionLoop(positionLoopN);
430     }
431
432     posI[0] = valX; //get initial position
433     posI[1] = valY;
434
435     driveStraight(dist * 2, motorPower); // movimento in avanti per "distance" metri
436
437     delay(1000);

```

```

438     if (controlBool) {
440         positionLoop(positionLoopN);
441     }
442
443     posF[0] = valX; //get final position
444     posF[1] = valY;
445
446     Serial.println("initial: " + String(posI[0]) + ", " + String(posI[1]));
447     Serial.println("final: " + String(posF[0]) + ", " + String(posF[1]));
448
449     if (controlBool) {
450         control();
451     }
452 }
453 // ****
454 //          CONTROL
455 // controllare che le coordinate siano
456 // entro un margine di errore
457 // ****
458 void control() {
459
460     positionLoop(positionLoopN);
461
462     float a = valX - destArrayX[destN]; //differenza x
463     float b = valY - destArrayY[destN]; //differenza y
464
465     float distance = sqrt((a * a) + (b * b)); //modulo vettore (a;b) in ticks
466     if (distance <= r) {
467
468         Serial.println("=====");
469         Serial.println("destination reached");
470
471         // Fare cadere i semi
472         semeServo.attach(semePin);
473         semeServo.write(180);
474         delay(1000);
475         semeServo.detach();
476
477         destN++; //Prossima destinazione
478     }
479
480     if (destN <= 1) {
481         calcAngolo(); // calcolare l'angolo di rotazione per la prossima destinazione
482     } else {
483
484         Serial.println("Tutte le destinazioni sono state raggiunte!");
485
486         // Entrare in un ciclo infinito e non fare niente. Eventualmente si potrebbe
487         // aggiungere un pulsante per far ripartire il robot
488         while (1);
489     }
490 }
491
492 // aggiornare dati posizione
493 void positionLoop(int n) {

```

```

494 // aggiorna le coordinate
495 serialFlush();
496 delay(100);
497 for (int i = 0; i <= n; ++i) {
498     readCoordinates();
499 }
500
501 // se il robot e' fuori dal campo visivo del Kinect eseguire lostRobot
502 if (checkBounds == 'x') {
503     lostRobot(lastDistance);
504 }
505
506 void driveStraight(float dist, int power) {
507     // numero di "ticks"
508     unsigned long num_ticks_l;
509     unsigned long num_ticks_r;
510
511     // impostare potenza iniziale dei motori
512     int power_l = motorPower;
513     int power_r = motorPower;
514
515     // usate per stabilire in quale senso si deve ruotare il motore per la correzione
516     unsigned long diff_l;
517     unsigned long diff_r;
518
519     // resettare contatori encoder
520     enc_l = 0;
521     enc_r = 0;
522
523     // salvare contatore encoder precedente
524     unsigned long enc_l_prev = enc_l;
525     unsigned long enc_r_prev = enc_r;
526
527     // calcolare di quanto devono girare i motori
528     float num_rev = (dist * 10) / wheel_c; // Convert to mm
529     unsigned long target_count = num_rev * countsPerRev;
530
531     // Debug
532     Serial.print("Driving for ");
533     Serial.print(dist);
534     Serial.print(" cm (");
535     Serial.print(target_count);
536     Serial.print(" ticks) at ");
537     Serial.print(power);
538     Serial.println(" motor power");
539
540     // se il robot e' fuori dal campo visivo tornare indietro fino a rientrarci
541     if (lostRobotBool) {
542         Serial.println("Moving until robot is in");
543
544         // muoversi in avanti finche' non viene stabilito che il robot e' rientrato nel
545         // campo visivo
546         while (checkBounds == 'x') {
547
548             // Salvare numero di "ticks"
549             num_ticks_l = enc_l;

```

```

550     num_ticks_r = enc_r;
551
552     // Drive
553     drive(power_l, power_r);
554
555     // numero di ticks dall'ultima volta
556     diff_l = num_ticks_l - enc_l_prev;
557     diff_r = num_ticks_r - enc_r_prev;
558
559     // salvare numero di ticks per prossima volta
560     enc_l_prev = num_ticks_l;
561     enc_r_prev = num_ticks_r;
562
563     // se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
564     if ( diff_l < diff_r ) {
565         power_l -= motor_offset;
566         power_r += motor_offset;
567     }
568
569     // se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
570     if ( diff_l > diff_r ) {
571         power_l += motor_offset;
572         power_r -= motor_offset;
573     }
574
575     //Controllare che non ci sia un ostacolo
576     obstacle();
577
578     serialFlush();
579     delay(100);
580     for ( int i = 0; i <= 50; ++i) {
581         readCoordinates();
582     }
583 } else {
584     // ruotare finche' uno dei motori non compie la distanza target
585     while ( (enc_l < target_count) && (enc_r < target_count) ) {
586
587         // Salvare numero di "ticks"
588         num_ticks_l = enc_l;
589         num_ticks_r = enc_r;
590
591         // Drive
592         drive(power_l, power_r);
593
594         // numero di ticks dall'ultima volta
595         diff_l = num_ticks_l - enc_l_prev;
596         diff_r = num_ticks_r - enc_r_prev;
597
598         // salvare numero di ticks per prossima volta
599         enc_l_prev = num_ticks_l;
600         enc_r_prev = num_ticks_r;
601
602         // se motore sinistro piu' veloce, rallentarlo e velocizzare il destro
603         if ( diff_l < diff_r ) {
604             power_l -= motor_offset;

```

```

    power_r += motor_offset;
}

// se motore destro piu' veloce, rallentarlo e velocizzare il sinistro
if ( diff_l > diff_r ) {
    power_l += motor_offset;
    power_r -= motor_offset;
}

obstacle();
}

lostRobotBool = false;

// frenare
brake();
}
void drive(int power_a, int power_b) {
// Limitare power da a -255 and 255
power_a = constrain(power_a, -255, 255);
power_b = constrain(power_b, -255, 255);

// Direzione motore sinistro
if ( power_a < 0 ) {
    digitalWrite(ain1_pin, LOW);
    digitalWrite(ain2_pin, HIGH);
} else if ( power_a > 0 ) {
    digitalWrite(ain1_pin, HIGH);
    digitalWrite(ain2_pin, LOW);
} else {
    digitalWrite(ain1_pin, LOW);
    digitalWrite(ain2_pin, LOW);
}

// Direzione motore destro
if ( power_b < 0 ) {
    digitalWrite(bin1_pin, LOW);
    digitalWrite(bin2_pin, HIGH);
} else if ( power_b > 0 ) {
    digitalWrite(bin1_pin, HIGH);
    digitalWrite(bin2_pin, LOW);
} else {
    digitalWrite(bin1_pin, LOW);
    digitalWrite(bin2_pin, LOW);
}

// Impostare velocita'
analogWrite(pwma_pin, abs(power_a));
analogWrite(pwmb_pin, abs(power_b));
}

void brake() {
digitalWrite(ain1_pin, LOW); // Imposta tutti i pin a 0V
digitalWrite(ain2_pin, LOW);
digitalWrite(bin1_pin, LOW);
}

```

```

662   digitalWrite(bin2_pin, LOW); // Imposta il segnale pwm a 0
663   analogWrite(pwma_pin, 0);
664   analogWrite(pwmb_pin, 0);
665 }
666 void serialFlush() {
667   while (mySerial.available())
668     mySerial.read();
669 }
670 void readCoordinates() {
671   if (mySerial.available() >= MESSAGE_BYT
672   {
673     if (mySerial.read() == HEADER) // se riconosce il header
674     {
675       char tag = mySerial.read(); // var. char usata per ricevere il bit
676       int sign = 1; // int per rendere il numero positivo o
677       negativo
678
679       if (tag == Xc) // se riconosce che il valore appartiene
680         alla coordinata X
681       {
682
683         valX = mySerial.read() * 256; // ricompone il numero, muovendo il byte di
684         una posizione
685         valX = valX + mySerial.read(); // aggiunge il bit successivo
686
687         int checkSign = mySerial.read(); // funzione per verificare il segno
688         if (checkSign == 'n') { // se negativo Processing manda una n
689           sign = -1;
690         }
691
692         valY *= sign; // moltiplicare il valore per il segno
693
694       } else if (tag == Yc) { // processo analogo ma con la coordinata Y
695
696         valY = mySerial.read() * 256;
697         valY = valY + mySerial.read();
698
699         int checkSign = mySerial.read();
700         if (checkSign == 'n') {
701           sign = -1;
702         }
703
704         valY *= sign;
705
706         // se riceve y vuol dire che il robot e' dentro il campo visivo, se riceve x
707         non lo e'
708         checkBounds = mySerial.read();
709
710         Serial.println(String(valY));
711
712     } else {
713       Serial.print("got message with unknown tag ");
714       Serial.println(tag);
715     }
716   }
717 }
```

```

    }

714 void lostRobot(float lastDistance) {
    Serial.println("Robot is lost, turning 180 degrees and going back for " + String(
        lastDistance));
716 turnRobot(turn180, motorPower, false);
    lostRobotBool = true;
718 destinationMovement(lastDistance, false);
    driveStraight(10, motorPower);

720 getDirection();
}
722 void countLeft() {
724     enc_l++;
}
726 void countRight() {
728     enc_r++;
}
730 long getDistance() {
// calcolo della distanza che rileva il sensore a ultrasuoni
732 long duration, distance;

734 digitalWrite(trigPin, LOW);
delayMicroseconds(2);

736 digitalWrite(trigPin, HIGH);
delayMicroseconds(10);

738 digitalWrite(trigPin, LOW);

740 duration = pulseIn(echoPin, HIGH);
distance = (duration / 2) / 29.1;

742 return distance;
}
744 void obstacle(){
    int distanceToObject = getDistance();
746 if (distanceToObject <= 20.0 && distanceToObject > 2.0) {
        brake();
748 while (distanceToObject <= 20.0 && distanceToObject > 2.0) {
            Serial.println("Distance to object: " + String(distanceToObject) );
            distanceToObject = getDistance();
        }
750 }
752 // breve pausa per lasciare tempo ai motori per reagire
754 delay(20);
756 }
758 }

```

Codice Processing:

```

2 import org.openkinect.freenect.*;
import org.openkinect.freenect2.*;
import org.openkinect.processing.*;
4 import org.openkinect.tests.*;
import processing.serial.*;

```

```

6 import blobDetection.*;
8
8 public static final char HEADER = '|';
10 public static final char Xc = 'N';
10 public static final char Yc = 'M';
12
12 Kinect kinect;
13 BlobDetection blobDetector; // Create a blob Detector
14 Serial myPort;
15
16 boolean mirror = false;
16 boolean medFilter = false;
18 boolean irState = false; // Use IR camera
19
20 PImage currentImage, redImage, depthImage; // Images to display //An initial image
20 to hold blobs
21
22 int threshold = 120;
22 int filterSize = 2;
24 int[] depth;
24 float deg;
26 float avgDepth;
26 float depthSum = 0;
28 int avgOffset;
28 long avgMappedX;
30 long avgMappedY;
30 float avgX;
32 float avgY;
32 float fX;
34 float fY;
34 float realX;
36 float realY;
36 float alphaX;
38 float alphaY;
38 boolean NaN = false;
40 float sumX = 0;
40 float sumY = 0;
42 int totalPixels = 0;
43
44 void setup() {
44 // Impostare dimensioni dell'area di lavoro
46 size(2140, 1100);
47
48 // Nome della porta seriale
48 myPort = new Serial(this, "COM5", 9600);
49
50 // Inizializzare kinect e le diverse immagini
52 kinect = new Kinect(this);
52 kinect.initVideo();
54 kinect.initDepth();
55
56 // Visualizzare immagine
56 currentImage = kinect.getVideoImage();
58 depthImage = kinect.getDepthImage();
59
60 // Creare un'immagine filtrata (pixel rossi)

```

```

62     redImage = currentImage.get();
63     redImage = redThreshold(redImage, threshold);
64     image(redImage, 640, 0);
65 }
66 void draw() {
67     stroke(255);
68     background(0);
69     currentImage = kinect.getVideoImage();
70     image(depthImage, 850, 0);
71     image(currentImage.get(), 1500, 0);
72
73     // Creare un'immagine filtrata (pixel rossi)
74     redImage = currentImage.get();
75     redImage = redThreshold(redImage, threshold);
76
77     //median filtering
78     if(medFilter) {
79         redImage = medianFilter(redImage, filterSize);
80     }
81
82     image(redImage, 850, 480);
83     fill(255);
84
85     // Ricavare le coordinate e inserirle in un vettore
86     PVector v = depthToWorld(int(avgX), int(avgY), int(avgDepth));
87
88     // Inviare coordinate via seriale
89     sendMessage(Xc, int(v.x*100));
90     sendMessage(Yc, int(v.z*100));
91
92     textSize(26);
93     text("fps: " + int(frameRate), 10, 30);
94     text("avg x: " + int(avgX) + " avg Y: " + int(avgY) + "\n" + "\n" +
95         "Coordinates - x: " + floor(v.x*100) + "cm" + ", y: " + floor(v.z*100) + "cm" +
96         "\n" + "\n" +
97         "threshold: " + int(threshold) + "\n" + "\n" + "totalPixels: " + totalPixels
98     , 1500, 600 );
99
100 /* =====
101    Grafico
102    =====*/
103 float x;
104 float y;
105 if(NaN) {
106     x = 0;
107     y = 100;
108 } else {
109     x = map(v.x*100, 0, 143, 0, 382);
110     y = map(v.z*100, 0, 300, 0, 800);
111 }
112 x = 400 + int(x);
113 y = 900 - int(y);
114

```

```

116     fill(0);
117     stroke(150);
118     arc(400, 900, 1600, 1600, PI + HALF_PI - 0.4974188, PI + HALF_PI + 0.4974188, PIE)
119     ;
120     fill(0, 199, 255);
121     rect(300, 900, 200, 50);
122     stroke(100);
123     line(400, 900, 400, 0); //vertical
124     line(0, 900, 800, 900);
125     stroke(0, 199, 255);
126     line(390, 100, 410, 100);
127     text("(0cm ;300cm)", 420, 100);
128     text("(0;0)", 420, 890);
129     fill(148, 255, 0);
130     textSize(18);
131     ellipse(x, y, 25, 25);
132     if(NaN) {
133       text("Robot out of view", x+25, y + 10);
134     } else {
135       text("(" + floor(v.x*100) + "cm; " + floor(v.z*100) + "cm)", x+25, y + 10);
136     }
137
138 // Evento seriale
139 void serialEvent(Serial p) {
140   // handle incoming serial data
141   String inString = myPort.readStringUntil('\n');
142   if(inString != null) {
143     println( inString ); // echo text string from Arduino
144   }
145
146 void sendMessage(int tag, int value){
147   // send the given index and value to the serial port
148   myPort.write(HEADER); //mandare un header in modo che l'arduino sappia che stia
149   arrivando un nuovo valore
150   myPort.write(tag); //mandare un tag che identifica il valore
151
152   char c = (char)(abs(value) / 256); // msb valore assoluto per correttezza di
153   calcoli con valori negativi
154   myPort.write(c); // convertire i valori ascii in valori numerici
155   c = (char)(abs(value) & 0xff); // lsb
156   myPort.write(c);
157
158   if(value < 0) {
159     myPort.write('n'); // Verificare se numero positivo o negativo
160   } else {
161     myPort.write('p');
162   }
163   //se non trova abbastanza pixels
164   if(totalPixels < 150 || (tag == Yc && value < 50)) {
165     NaN = true;
166     myPort.write('x');
167   } else {
168     NaN = false;
169     myPort.write('y');

```

```

168     }
170 }
172 // convert to radians
173 float toRadians(float deg) {
174     double radians = deg * 3.14159265359 / 180;
175     return (float) radians;
176 }
178 //convert to deg
179 float toDeg(float rad) {
180     double deg = rad * 180 / 3.14159265359;
181     return (float) deg;
182 }
184 // le tre funzioni per ricavare l'intensita' di un colore di un pixel
186 //calculate red channel intensity
187 float calcRedVal(color c) {
188     return c >> 16 & 0xFF - (c >> 16 & 0xFF + c >> 8 & 0xFF + c & 0xFF) / 3; //RED
190 //calculate green channel intensity
191 float calcGreenVal(color c) {
192     return c >> 8 & 0xFF - (c >> 16 & 0xFF + c >> 8 & 0xFF + c & 0xFF) / 3; //GREEN
194 //calculate blue channel intensity
195 float calcBlueVal(color c) {
196     return c >> 0xFF - (c >> 16 & 0xFF + c >> 8 & 0xFF + c & 0xFF) / 3; //GREEN
198 // Ottenere intensita' del colore rosso in ogni pixel dell'immagine RGB -- check
199     calcRedChannel()
200 PImage redThreshold(PImage img, int threshold){
201     img.loadPixels();
202     depth = kinect.getRawDepth();
204     sumX = 0;
205     sumY = 0;
206     totalPixels = 0;
208     // Step through every pixel and see if the redness > threshold
209     for (int x = 0; x < img.width; x += 1) {
210         for (int y = 0; y < img.height; y +=1) {
212             int offset = x + y * img.width;
214             if (calcRedVal(img.pixels[offset]) >= threshold ) {
215                 // Set pixel to white
216                 img.pixels[offset] = color(255, 255, 255);
218                 sumX += x;
219                 sumY += y;
220                 totalPixels++;
222             } else {
223                 // Set pixel to black

```

```

        img.pixels[offset] = color(0, 0, 0);
224    }
225}
226img.updatePixels();
228
229avgX = 1.055 * (sumX / totalPixels);
230avgY = 0.94 * (sumY / totalPixels);
231avgMappedX = map((long) avgX, 0, 1280, 0, 1280);
232avgMappedY = map((long) avgY, 0, 960, 0, 960);
233avgOffset = int(avgMappedX + avgMappedY * depthImage.width);
234//avgDepth = depth[avgOffset];
235avgDepth = depth[avgOffset];
236
237fill(150, 255, 0);
238ellipse(avgX + 850, avgY, 8, 8);
//fill(150, 255, 0);
//ellipse(avgMappedX, avgMappedY, 30, 30);

242return img;
}
244
245//convert raw depth data to meters
246float rawDepthToMeters(float depthValue) {
247    if (depthValue < 2047) {
248        return (float)(1.0 / ((double)(depthValue) * -0.0030711016 + 3.3309495161));
249    }
250    return 0.0f;
}
252
253// Convertire i valori del Kinect in valori reali (metri)
254PVector depthToWorld(int x, int y, int depthValue) {
255
256    final double fx_d = 1.0 / 5.9421434211923247e+02;
257    final double fy_d = 1.0 / 5.9104053696870778e+02;
258    final double cx_d = 3.3930780975300314e+02;
259    final double cy_d = 2.4273913761751615e+02;
260
261    // Impostare il vettore in modo da contenere le tre coordinate dello spazio fisico
262    PVector result = new PVector();
263    double depth = rawDepthToMeters(depthValue); //rawDepthToMeters(depthValue);
264    result.x = (float)((x - cx_d) * depth * fx_d);
265    result.y = (float)((y - cy_d) * depth * fy_d);
266    result.z = (float)(depth);
267    return result;
}
268
269
270//map function for java
271long map(long x, long in_min, long in_max, long out_min, long out_max)
272{
273    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
274
275
276
277// Median filter

```

```

PImage medianFilter(PImage img, int size) {
280    img.loadPixels();
281    PImage temp;
282    int[] tempPixels = img.get().pixels;

284    // Step through every pixel in the image that is not a border pixel
285    for(int y = size; y < img.height - size; y++) {
286        for(int x = size; x < img.width - size; x++) {
287            // Get a block of pixels of size (2*size+1)^2 around each
288            temp = img.get(x-size, y-size, 2*size+1, 2*size+1);
289            // Find the median element
290            tempPixels[y * img.width + x] = sort(temp.pixels)[(2*(2*size+1)-1) / 2];
291        }
292    }

294    // Update the pixels in the image
295    img.pixels = tempPixels;
296    img.updatePixels();
297    return img;
298}

300 // Mappatura tasti per diversi comandi
void keyPressed() {
301    if(key == 'h') {
302        mirror = !mirror;
303        kinect.enableMirror(mirror); // enable mirror mode
304    } else if (key == 'w') {
305        //blobDetector.setThreshold(threshold / 255.0);
306        threshold++; // turn up threshold
307    } else if (key == 's') {
308        //blobDetector.setThreshold(threshold / 255.0);
309        threshold--; // turn down threshold
310    }else if (key == 'i') {
311        irState = !irState; // toggle ir state, useful when kinect freezes
312        kinect.enableIR(irState);
313    } else if (key == 'm') {
314        medFilter = !medFilter; // toggle med filter
315    } else if (key == CODED) {
316        if (keyCode == UP) {
317            deg++; // tilt kinect upwards
318        } else if (keyCode == DOWN) {
319            deg--; // tilt kinect downwards
320        }
321        deg = constrain(deg, 0, 30);
322        kinect.setTilt(deg);
323    }
324}

```