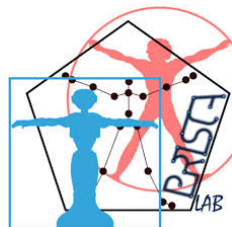


Touch the Color

Progetto di
Sistemi per il Governo dei Robot mod. A
A.A 2018/2019

Grieco Riccardo N97/28
Matarese Marco N97/280
Pollastro Andrea N97/28



Contents

1	Introduzione	3
1.1	Descrizione del Gioco	3
1.2	Nicchia Ecologica	3
1.3	Schema dei Behaviours	4
2	Navigazione	5
2.1	Campi di potenziale	5
2.2	Orientamento	8
2.3	Movimento	9
3	Modulo per la Comunicazione	10
3.1	Gestione delle Socket	10
3.2	Il Protocollo di Comunicazione	10
4	Modulo di Visione	12
4.1	Acquisizione dello stream video	12
4.2	Estrazione del blob e calcolo del centroide	13
4.3	Estrazione della profondità	14
4.4	Calcolo della direzione da seguire	15
5	Esperimenti	16

1 Introduzione

1.1 Descrizione del Gioco

Touch the color, o "Strega chiama color" nella sua versione italiana, è un semplice gioco di gruppo da svolgersi all'aria aperta. Questo prevede la selezione di un giocatore del gruppo al ruolo di strega, il quale dovrà scegliere un colore ed urlarne il nome a tutti gli altri componenti del gruppo. Quest'ultimi, udito il colore, dovranno cercare nel minor tempo possibile un oggetto del colore indicatogli per poi raggiungerlo: perde l'ultimo giocatore a "toccare il colore" e verrà selezionato come strega al prossimo giro.

Nel resto della documentazione i robot nel ruolo della strega verranno chiamati nodi Witch e quelli nel ruolo del resto del gruppo verranno chiamati nodi Kid.

1.2 Nicchia Ecologica

Nella progettazione di un sistema robotico, al progettista devono essere chiare fin dal principio tre cose:

- il task, o i task, che il robot dovrà compiere;
- le caratteristiche del robot;
- le caratteristiche dell'ambiente in cui il robot agirà.

La definizione di queste tre cose specifica quella che si chiama *nicchia ecologica* del robot. La nicchia ecologica permette di definire vincoli e potenzialità del sistema.

Per quanto riguarda il primo punto, il task, abbiamo delle differenze fra nodi Witch e nodi Kid. Per i primi si tratta di un task molto semplice, esso prevede nell'ordine:

- la scelta di un colore da un pool ben definito;
- la comunicazione del colore ai nodi Kid;
- la comunicazione della fine del gioco ai nodi Kid una volta ricevuto $n - 1$ messaggi di "colore toccato", dove n è il numero di nodi Kid.

Si tratta quindi, per i nodi Witch, di un task a tempo illimitato, robot-based, senza movimento e dipendente, in quanto la comunicazione della fine del gioco dipende da comunicazioni precedenti dei nodi Kid. Per i nodi Kid, invece, il task prevede:

- trovare un oggetto del colore comunicatogli e raggiungerlo;
- comunicare al nodo Witch che ha toccato il colore.

I nodi Kid quindi svolgono un task minimum-time, in quanto devono cercare di portare a termine il task nel minor tempo possibile, object-based, movement-to e dipendente, per motivazioni simili a quelle fatte per i nodi Witch.

Per quanto riguarda i robot, si tratta di *Pioneer* modello 3-DX munito di due ruote motrici fisse ed una mobile e dodici (o sei) sensori sonar. Sulla base superiore abbiamo installato una telecamera RGB-D (*Microsoft Kinect*) la quale viene utilizzata per le funzioni di computer vision.

Per quanto riguarda l'organizzazione dei robot, invece, abbiamo una composizione eterogenea (o omogenea in quanto si tratta dello stesso tipo di robot????), dalla grandezza limitata, la cui riconfigurabilità è comunicata, avente comunicazione esplicita con topologia ad albero.

1.3 Schema dei Behaviours

Come per il resto, lo schema dei behaviours si differenzia in base al tipo di nodo che consideriamo. La Figura 1 mostra lo schema riferito ai nodi Witch, la Figura 2 quello riferito ai nodi Kid.

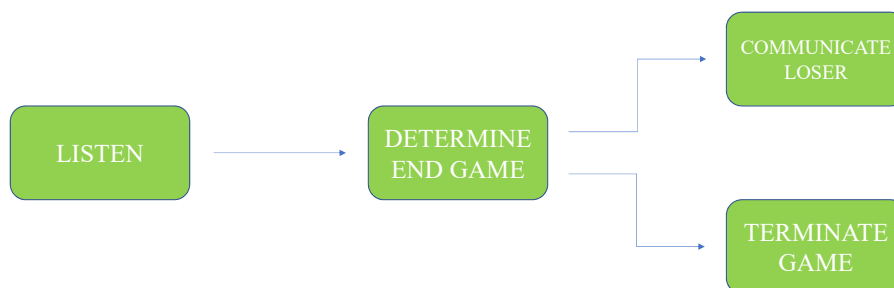


Figure 1: Schema dei behaviours dei nodi Witch.



Figure 2: Schema dei behaviours dei nodi Kid.

2 Navigazione

Per la navigazione del robot, in particolare del robot *Kid*, è stato utilizzato un approccio misto che utilizza sussunzione e metodologia a campi di potenziale.

2.1 Campi di potenziale

I campi previsti dall'architettura sono i seguenti:

- Wander
- Avoid
- Move To Goal

Le forze esercitate dai campi di potenziale hanno per convenzione intensità nell'intervallo $[0, 1]$.

Il campo **Wander**, che permette l'esplorazione dell'ambiente, applica al robot una forza costante in avanti rispetto al suo orientamento. Con cadenza costante il campo

genera una forza con un orientamento rispetto al sistema di riferimento del robot tra i -90° e i $+90^\circ$ in maniera casuale. Il vettore generato ha la massima intensità possibile.

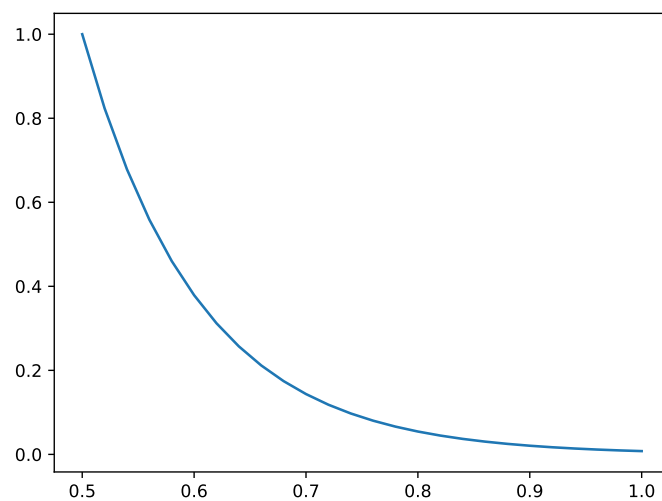


Figure 3: Plot della funzione del campo Avoid.

Il campo **Avoid** permette la *Obstacle Avoidance*. Ogni ostacolo genera un campo repulsivo con un profilo di ampiezza esponenziale. In particolare l'intensità della forza generata dal campo segue la funzione:

$$avoid(d) = b^{-c \cdot (d - D_S)}$$

dove d è la distanza del robot dall'oggetto, b e c sono fattori che determinano il comportamento della curva e D_S è una distanza di sicurezza, fissata a priori. Il campo viene percepito solo quando la distanza dall'oggetto è di almeno 1 metro. La Figura 3 evidenzia come la forza esercitata dal campo abbia intensità massima quando l'oggetto si trova ad una distanza vicina quella di sicurezza e minima al limite della percezione del campo.

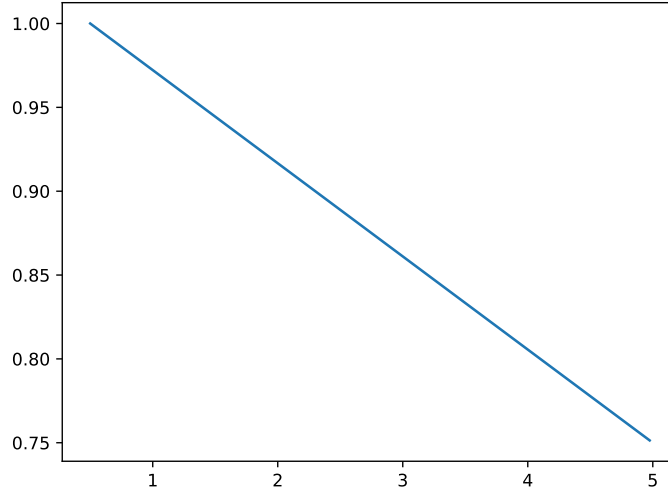


Figure 4: Plot della funzione del campo Move to Goal.

Il campo **Move to Goal** permette al robot di raggiungere un punto di interesse, che nel caso in esame è il colore a cui avvicinarsi. Il punto di interesse (POI) genera un campo attrattivo di intensità lineare sulla distanza dal robot che segue la funzione:

$$attract(d) = \frac{1 - F_{min}}{D_S - D_A} \cdot (d - D_S) + 1$$

dove F_{min} è la minima intensità garantita dal campo mentre D_S e D_A sono rispettivamente la distanza di sicurezza e la minima distanza di percezione del campo.

Il campo viene percepito solo quando l'oggetto si trova ad una distanza affidabile, ovvero tale per cui il riconoscimento dello stesso ha una certa affidabilità. Nel caso in questione, in cui il punto di interesse viene riconosciuto da una telecamera RGB-D, la rilevazione è affidabile quando l'oggetto si trova ad una distanza di al più 5 metri. La Figura 4 mostra come la forza esercitata dal campo abbia intensità massima quando l'oggetto si trova vicino al POI, mentre a distanza massima di percezione garantisca comunque un'intensità abbastanza elevata, permettendo al robot di raggiungere il punto di interesse più o meno velocemente.

2.2 Orientamento

Il sistema di coordinate utilizzato è robocentrico, ovvero ogni punto rilevante (ostacoli, punto di interesse) nello spazio è individuato da coordinate ottenute dalle misurazioni dei sensori che il robot utilizza.

Questo sistema rende molto facile l'utilizzo dei campi di potenziale per l'individuazione di un vettore che determini il movimento del robot. Un problema che sorge dall'utilizzo di questo sistema è quello di gestire punti di cui non si hanno più rilevazioni dai sensori. Nel caso specifico del robot Kid, se esso non ha più visibilità del POI che cerca di raggiungere, ad esempio perché si è dovuto girare per evitare un'ostacolo, dovrebbe comunque poter stimare la posizione dell'oggetto. Per fare ciò è necessario memorizzare lo spostamento e la rotazione effettuati dal sistema di riferimento rispetto all'ultima volta in cui tale oggetto è stato rilevato. Ottenere questi dati è possibile grazie all'utilizzo dell'interfaccia RosAria per la gestione della mobilità del robot. Il nodo ROS RosAria pubblica sul topic `/RosAria/pose` le informazioni sulla posizione e sull'orientamento del robot. I messaggi pubblicati sul topic utilizzano un sistema di coordinate mondocentrico, i cui orientamento e origine coincidono con l'orientamento e la posizione del robot al momento dell'avvio del nodo RosAria. Per calcolare lo spostamento e la rotazione effettuata dal robot, basta memorizzare le informazioni pubblicate sul topic al momento dell'ultima rilevazione del punto di interesse e le informazioni "attuali" fornite da RosAria. Il vettore di traslazione e la matrice di rotazione sono calcolati come segue:

$$t = (p_s - p_c) \cdot R_s$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

dove p_s e p_c sono rispettivamente la posizione al momento dell'ultima rilevazione dell'oggetto di interesse e la corrente. R_s è la matrice di rotazione al momento dell'ultima rilevazione del POI e θ è l'ampiezza dell'angolo di cui è ruotato il robot rispetto all'ultima rilevazione.

Il punto di interesse, una volta fuori dal campo visivo del robot, viene calcolato come segue

$$POI_c = (POI_s + t) \cdot R$$

dove POI_s è la posizione del punto di interesse al momento dell'ultima rilevazione.

2.3 Movimento

Lo schema motorio è costituito da un'unica funzione **move**, che decide il movimento del robot in base alla forza complessiva esercitata dai campi di potenziale. La coordinata x della forza determina la rotazione del robot, mentre la y determina l'intensità del movimento in avanti. La velocità lineare e quella angolare, data la forza F , sono così rispettivamente calcolate:

$$v = \frac{F_y}{4}$$

$$\omega = \frac{F_x}{2}$$

In questo modo la velocità lineare massima raggiunta è di 0.25 m/s mentre quella angolare assume un valore massimo di 1.57 rad/s ($90^\circ/\text{s}$).

3 Modulo per la Comunicazione

La comunicazione fra i robot è stata implementata tramite socket tcp. Tale tecnologia ci ha permesso, tramite la conoscenza pregressa degli indirizzi IP di tutti i robot partecipanti, di costruire delle linee di comunicazione punto-punto. Come già visto nella sezione introduttiva, il gioco prevede due ruoli: sono proprio i giocatori di ruoli diversi i soli a doversi scambiare messaggi.

3.1 Gestione delle Socket

Ogni robot gestisce almeno una socket: il Role Manager del nodo Witch ne gestisce una per ogni robot Kid e i Role Manager dei nodi Kid ne gestiscono una soltanto. Le socket, a prescindere dalla tipologia di nodo, vengono gestite attraverso il metodo `manageSocket(threadSocket)` della classe `RoleManager`. Il metodo in questione prende in input il file descriptor della socket su cui si metterà in ascolto e a cui invierà messaggi. `manageSocket` viene eseguito da un thread a parte, generato nel metodo `createAndStartConnection` solo per questo compito. In particolare, abbiamo che i nodi Kid generano un solo nuovo thread e il nodo Witch genera un thread per ogni nodo Kid a cui deve connettersi. Tale metodo non fa altro che rimanere in ascolto della socket passatagli finché la variabile d'istanza `stopThread` non viene settata a `true`: ciò accade quando la partita in corso è terminata e bisogna resettare i parametri della classe `RoleManager`. I messaggi che arrivano sulla socket vengono quindi intercettati all'interno di questo loop e gestiti da un *handler* di funzioni - implementato attraverso un array di riferimenti a funzioni - il quale, a seconda del tipo di messaggio ricevuto, invoca il metodo responsabile della gestione di quest'ultimo. Una volta usciti dal "loop di ascolto" il thread in questione viene fermato.

ROS permette la definizione di variabili globali al sistema, referenziabili a runtime dai diversi nodi. Abbiamo scelto di sfruttare questa possibilità offerta da ROS per comunicare ai robot gli indirizzi IP di tutti i partecipanti, incluso il proprio. I primi memorizzati in una lista di stringhe e il secondo in una semplice stringa, queste variabili vengono settate una volta sola (dopo aver eseguito *roscore*) attraverso le due istruzioni seguenti e recuperate nel metodo `__init__` del nodo `RoleManager`.

```
setparam /ipList "['100.101.0.10', '100.101.0.11', '100.101.0.11']"  
setparam /myIPAddress "100.101.0.10"
```

3.2 Il Protocollo di Comunicazione

Il protocollo di comunicazione è molto semplice: la struttura statica del gioco ci ha permesso di definire un protocollo rigido. I messaggi vengono scambiati solo tra

Communication Protocol

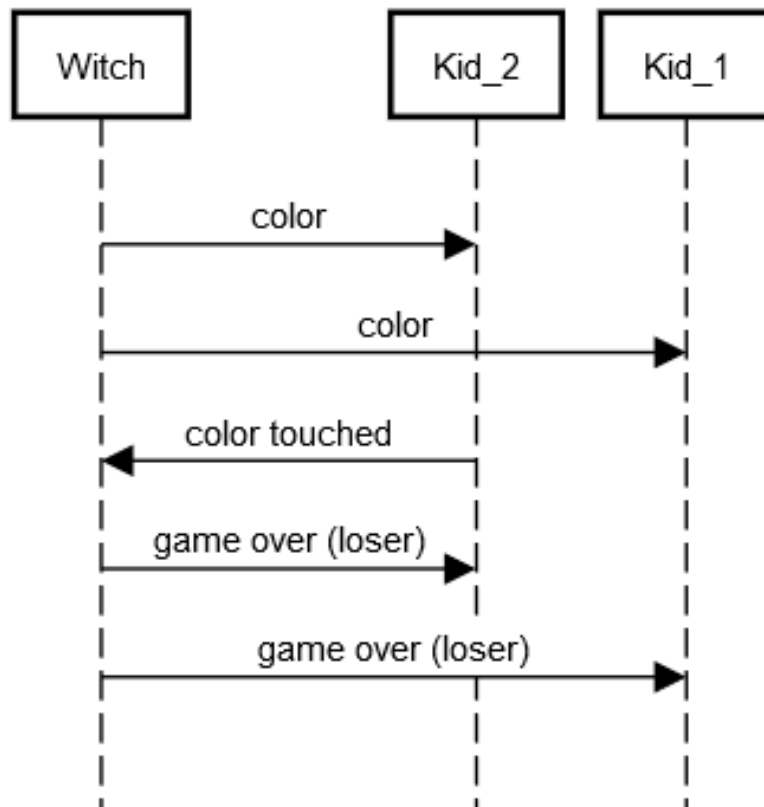


Figure 5: Rappresentazione grafica del protocollo di comunicazione.

giocatori di ruoli diversi, in particolare tra i Kids e la Witch: questo spiega la scelta di collegamenti punto-punto delle socket.

Sono stati previsti tre tipi di messaggio:

- un messaggio in cui la Witch comunica a tutti i Kids il colore da toccare, dando così il via al gioco;
- un messaggio in cui un Kid comunica alla Witch di aver appena toccato il colore comunicatogli;
- un messaggio in cui la Witch comunica a tutti i Kids che il gioco è terminato e chi fra loro è il perdente.

Una descrizione grafica del protocollo di comunicazione è presentata in Figura 5.

4 Modulo di Visione

Il modulo inerente agli aspetti di computer vision del progetto in esame è stato realizzato mediante l'uso di una telecamera RGB-D, in particolare tramite una *Microsoft Kinect*, il cui stream video è stato processato con l'ausilio dell'API *OpenCV*. Il modulo di visione in questione prevede la realizzazione della seguente pipeline:

1. acquisizione dello stream video RGB e di profondità e conversione dello spazio di colore da RGB a HSV
2. ricerca di uno o più blob del colore target associato alla sessione corrente
3. estrazione del valore di profondità associato ai centroidi dei blob trovati e determinazione del target
4. calcolo del vettore in direzione del target

4.1 Acquisizione dello stream video

L'acquisizione dello stream video tramite camera RGB-D è stata realizzata mediante l'uso del framework open-source *OpenNI*, ideato per accedere alle informazioni registrate dalla camera con un livello di astrazione alto. In particolare, l'acquisizione dello stream video RGB è stata realizzata attraverso la sottoscrizione al topic `/camera/rgb/image_rect_color` mentre l'acquisizione dello stream video di profondità è stata realizzata attraverso la sottoscrizione al topic `/camera/depth_registered/image`.

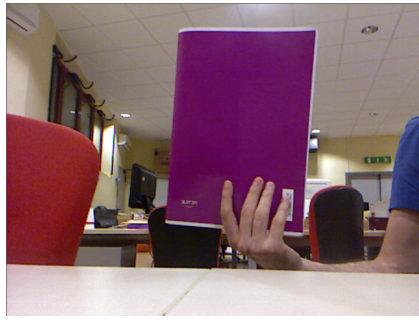
Siccome entrambi i topic forniscono informazioni mediante messaggi di tipo `sensor_msgs/Image`, per permettere l'analisi tramite *OpenCV* è necessario convertire entrambi gli stream video nel formato `cv::Mat`. La suddetta conversione è stata realizzata mediante l'uso della libreria *CvBridge* con la seguente funzione di conversione

```
imgmsg_to_cv2(frame_video, encoding)
```

dove per encoding sono stati utilizzati `bgr8` ed `32FC1` rispettivamente per la conversione dell'immagine RGB e di profondità.

Per agevolare l'analisi dei frame è stato effettuato un preprocessing dei frame RGB. Per ridurre il rumore presente è stato applicato un filtro gaussiano realizzato mediante l'uso di un kernel di dimensioni `3x3`. L'applicazione del suddetto filtro è stata ottenuta mediante l'uso della funzione `cv2.GaussianBlur()`.

Essendo questa fase fortemente soggetta ad una discriminazione dei colori presenti nei frame, è stata effettuata una conversione dello spazio di colori da RGB a HSV mediante la funzione `cv2.cvtColor()`. Così facendo viene gestita in maniera più "naturale" la distinzione dei colori, in quanto nello spazio HSV la variazione dei



(a) Frame RGB



(b) Frame RGB

colori è soggetta soltanto alla componente *Hue* a differenza dello spazio RGB, in cui un colore viene visto come una combinazione di rosso, giallo e blu.

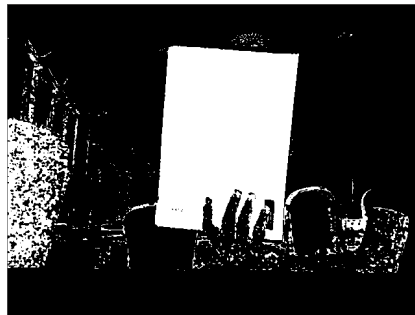
4.2 Estrazione del blob e calcolo del centroide

La *palette* dei colori a disposizione nel gioco è stata realizzata mediante l'uso di intervalli di colore espressi in formato HSV. Ad ogni colore quindi è associato un intervallo e durante la fase di estrazione del colore, ogni valore contenuto nell'intervallo associato al colore target è considerato valido.

Per realizzare tale selezione è stata utilizzata la funzione

```
cv2.inRange(frame_video, lower_bound, upper_bound)
```

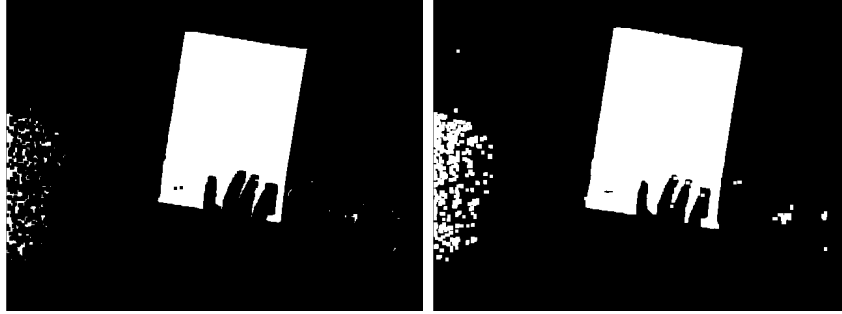
dove per `lower_bound` ed `upper_bound` sono stati inseriti gli estremi dell'intervallo associati al colore target. Dalla funzione viene ritornata una maschera il cui scopo è di isolare dal frame corrente soltanto le informazioni utili alla ricerca del blob.



(c) Maschera

Tuttavia la maschera evidenzia anche piccole porzioni di immagine che possono deviare la corretta ricerca del blob e che quindi costituiscono un rumore. Per elim-

inare gran parte di tale rumore è stata effettuata un'operazione di *erosione* tramite l'uso della funzione `cv2.erode()` seguita da un'operazione di *dilatazione* tramite l'uso della funzione `cv2.dilate()` utile a ristabilire le proporzioni delle componenti utili evidenziate dalla maschera.



(d) Erosione

(e) Dilatazione

Per tener traccia dei blob presenti nella scena, è stata utilizzata la funzione `cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`

dove `cv2.RETR_EXTERNAL` indica che vengono estratti soltanto i contorni esterni (trascurando quindi i contorni innestati in altri), mentre `cv2.CHAIN_APPROX_SIMPLE` indica che viene realizzata un'ottimizzazione dello spazio necessario a memorizzare i contorni.

Successivamente viene selezionato il blob avente area maggiore e, tramite la funzione `cv2.minEnclosingCircle()`, viene estratto il raggio del cerchio di area minima che lo include. Il blob in questione viene considerato valido soltanto se avente raggio superiore ad una soglia fissata apriori.

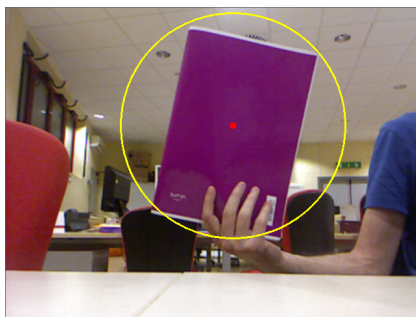
Arrivati a questo punto, viene estratto il centroide mediante i *momenti dell'immagine* associati al contorno del blob tramite la funzione `cv2.moments(contour)`. In particolare, le coordinate che identificano il centroide sono così espresse:

$$C_x = \frac{M_{10}}{M_{00}}, C_y = \frac{M_{01}}{M_{00}}$$

4.3 Estrazione della profondità

Giunti a questa fase, si ha a disposizione il centroide rappresentante il target da seguire. Possiamo quindi estrarre la distanza del blob dalla camera utilizzando la componente di profondità processata dalla camera RGB-D.

L'immagine di profondità a disposizione è nel formato `cv::Mat`, quindi per accedere alla distanza al centroide del blob è sufficiente estrarre dall'immagine la componente (C_x, C_y) . Così facendo si otterrà un valore in *floating point* a 32 bit rappresentante la distanza in metri del punto osservato nel pixel dalla camera.



(f) Blob selezionato

4.4 Calcolo della direzione da seguire

Per ottenere il vettore in direzione del target è stato utilizzato il raggio uscente dalla camera ed entrante nel pixel che costituisce il centroide del target.

Essendo l'immagine ottenuta per proiezione prospettica della scena visualizzata dalla camera su di un piano di proiezione, il raggio ottenuto potrà essere considerato valido per la generazione del vettore direzionale che vogliamo realizzare.

È stato utilizzato quindi il modello della *Pin-hole camera* per elaborare le informazioni associate alla Kinect ed estrarre il raggio in questione. Tali informazioni vengono estratte dal topic `/camera/depth_registered/camera_info` sottoforma di messaggio `sensor_msgs/CameraInfo`. Il modello della camera viene realizzato mediante l'uso dell'oggetto `PinholeCameraModel` contenuto nella libreria `image_geometry` nativa di ROS. Successivamente, mediante la funzione

```
projectPixelTo3dRay(centroid)
```

viene estratto il raggio uscente dalla camera in direzione della coppia di coordinate C_x, C_y sottoforma di versore. Infine, dalla moltiplicazione delle componenti x, z del raggio appena ottenuto con la distanza estratta allo step precedente, viene generato il vettore direzionale utile al robot per raggiungere il target.

5 Esperimenti