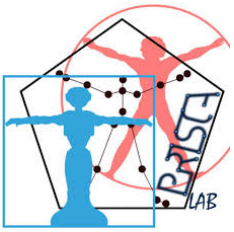
Touch the Color

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

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.

1.3 Schema dei Behaviours

2 Modulo Motorio

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

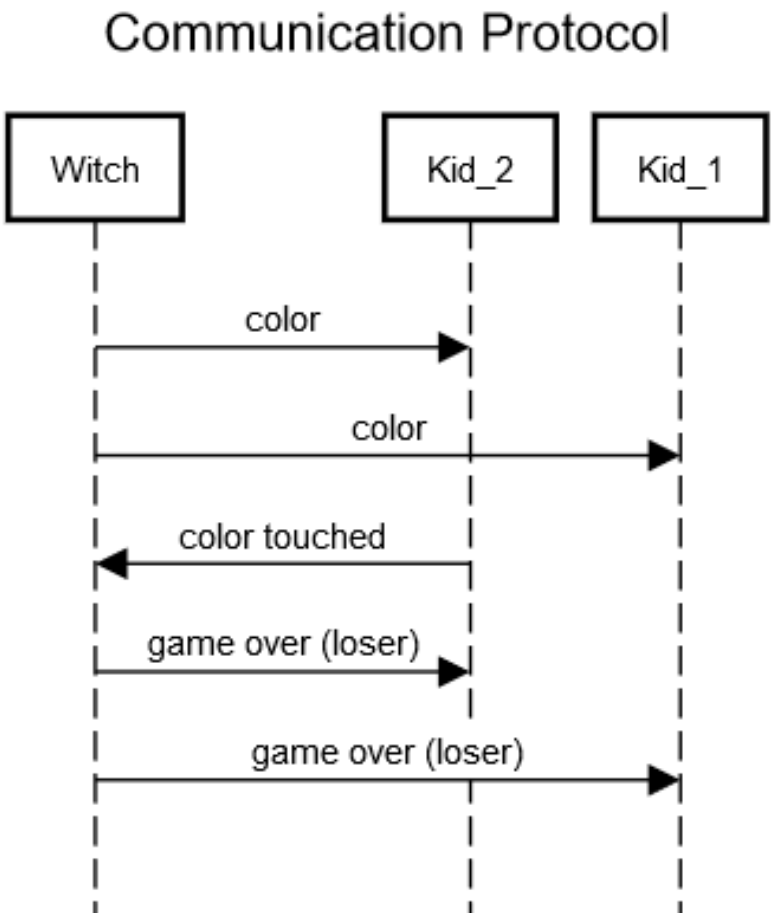


Figure 1: Rappresentazione grafica del protocollo di comunicazione.

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 1.

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 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.

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

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, per ogni blob, tramite la funzione `cv2.minEnclosingCircle()` viene estratto il raggio del cerchio di area minima che lo include. Così facendo è possibile rimuovere il rumore restante che non è stato eliminato in fase di erosione. Vengono considerati come validi soltanto i blob aventi raggio superiore ad una soglia fissata a priori.

Arrivati a questo punto, si ha a disposizione una lista di blob candidati alla ricerca del target valido da seguire. Viene quindi estratto il centroide di ogni blob mediante i *momenti dell'immagine* associati al contorno del blob ottenuti tramite la funzione `cv2.moments(contour)`. In particolare, le coordinate che identificano un 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 una lista di centroidi rappresentanti i candidati per la scelta del target da seguire. L'idea è di considerare come target il centroide avente distanza minima dalla camera. Per realizzare tale operazione viene utilizzata 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 associata ad ogni singolo pixel 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.

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