Sezione 2

Realizzazione di un framework per la gestione di un dispositivo multi-touch

DLM
SENSITIVE

2 DLM Sensitive

2.1 Introduzione

Una volta costruito il pannello multitouch secondo la tecnologia FTIR, dobbiamo realizzare qualche applicazione che ne faccia uso in modo da testarne le effettive potenzialità.
E' stato quindi deciso di implementare un apposito framework delegato alla gestione della fotocamera e all'elaborazione delle immagini, nonché al controllo degli eventi, in modo da nascondere agli occhi del programmatore tali meccanismi e permettergli la scrittura delle applicazioni in maniera semplice e veloce.

2.2 L'architettura

Il sistema è strutturato in un modello a stack. Possiamo quindi definire diversi livelli di astrazione, con cui, a partire dall'acquisizione fisica dei dati, arrivare ad ottenere set di eventi in grado di essere scatenati e gestiti dalle applicazioni.

Il livello touchRecognizer si occupa della gestione della camera e del riconoscimento dei blob. Sopra possiamo collocare lo strato incaricato della gestione degli eventi dove, a partire dai blob siamo in grado di determinare l'evento associato ad essi in base al comportamento assunto nei frame precedenti. Ai livelli superiori troviamo il modulo per la gestione degli ascoltatori e quello della definizione dei componenti grafici. In realtà gli ultimi due layer lavorano allo stesso livello di astrazione, ma è stata decisa una divisione logica in quanto il package dei componenti grafici è quello più vicino al programmatore ed è quello che mette realmente a disposizione i componenti da utilizzare nelle applicazioni.

Touch Recognizer
Il livello touchRecognizer accede alla camera ottenendo i dati grezzi (come array di byte corrispondenti al canale rosso) , la dimensione delle immagini e successivamente processa il frame scorrendolo pixel dopo pixel. Appena viene trovato un pixel illuminato (ovvero il cui valore d'intensità luminosa supera la soglia prevista) controlla che esso non faccia parte di altri blob e in questo caso tenta di riconoscere la forma a cui il pixel può essere aggregato. Crea dunque una nuova forma della dimensione dell'immagine acquisita e a partire dal primo pixel (che è automaticamente aggiunto) cerca nei punti ad esso adiacenti altri pixel illuminati da aggregare. Agendo ricorsivamente sull'intorno, è quindi in grado di ricostruire il blob e valorizzare le proprietà di area ed eccentricità. Tali caratteristiche sono utili per validare la figura.
Infatti, se l'area supera quella minima consentita, la forma è definita rilevante: in caso contrario ci troveremmo con altissima probabilità in presenza di un disturbo nell'immagine della webcam, come ad esempio punti di polvere. A questo punto ne viene verificata l'attendibilità (per evitare figure si dalla dimensione ragguardevole ma sempre e comunque disturbi, come per esempio sfarfallii) controllando l'eccentricità (ovvero il rapporto tra larghezza ed altezza): se il blob riconosciuto ha vagamente una forma che può essere definita circolare, allora viene considerato una forma attendibile.
Nel caso la forma sia accettata è recuperato il suo baricentro ed aggiunto al vettore dei punti

riconosciuti dopo essere stato adattato alla dimensione dello schermo (ricordiamo che lo scopo del lavoro è dotare lo schermo di un nuovo sistema di input, dunque è necessario mappare l'immagine dell'input a quella dell'output).

Gestione degli eventi

Una volta ottenuti i punti premuti, ne controlliamo il comportamento, cercando tra quelli presenti al frame precedente eventuali candidati a cui essi possano essere ricondotti.

Per legare tra i loro i punti effettuiamo un prodotto cartesiano creando coppie di punti: a questo punto le coppie vengono inserite in un vettore che verrà poi ordinato per distanze crescenti e visitato elemento per elemento.

Per ogni coppia visitata controlliamo se la distanza è inferiore a quella di correlazione prefissata: se non lo è, nemmeno le coppie successive lo saranno (essendo il vettore ordinato per distanze crescenti) e terminiamo quindi la visita. In caso contrario abbiamo trovato due punti correlati e discrimineremo tale relazione in base alla soglia di similarità per cui due punti possono essere ricondotti ad uno stesso. Se la distanza è inferiore a tale soglia i punti sono definiti coincidenti ed il lieve movimento trascurabile (in quanto potrebbe essere dovuto ad una leggera e non significativa variazione della posizione), in questo caso aggiorniamo la posizione del punto e inseriamo un evento di tipo press nel vettore degli eventi trovati. Se al contrario la distanza tra i due punti supera la soglia di similarità essi non coincidono tra loro ma possono essere comunque ricondotti alla stessa entità attiva in una operazione di spostamento sul piano. In questo caso aggiorniamo la posizione del punto e inseriamo un evento di tipo move al vettore. Continuiamo a scorrere le coppie di punti valutandone il comportamento fino a quando non giungiamo ad una coppia con distanza troppo alta per essere valutata come correlata. In questo caso l'esame sulle coppie termina e i punti rimanenti e non ancora visitati sono considerati come singole entità. I punti presenti al frame precedente e non legati a nessun punto del frame attuale (in quanto troppo distanti dagli altri) sono considerati come rilasci di pressione. Controllando lo spostamento assoluto dei punti possiamo determinare se tale rilascio sia successivo ad uno spostamento o ad una pressione che si traduce in eventi di tipo release nel primo caso e di click nel secondo. Eventi che anche in questo caso aggiungiamo al vettore. A questo punto esaminiamo i punti rimasti del frame attuale: i punti non correlati a nessun altro del frame precedente sono sicuramente nuove pressioni. Per ognuno di essi inseriamo un evento di tipo press nel vettore degli eventi trovati.

Gestione degli ascoltatori

Una volta ottenuto il vettore di eventi dal livello sottostante è necessario notificare tale situazione al componente appropriato. Per farlo dobbiamo tenere traccia della relazione componente-ascoltatore e a tale scopo utilizziamo una struttura hash. Per recuperare il componente visibile alla posizione in cui l'evento è stato scatenato si cicla sulla struttura e si analizzano i vari componenti; si utilizza un indicatore di profondità in modo da accertarsi che il componente ottenuto al termine della ricerca sia quello con profondità (distanza dalla radice nell'albero dei componenti grafici istanziati) più alta: ovvero abbiamo recuperato il componente di top-level sul piano dell'interfaccia.

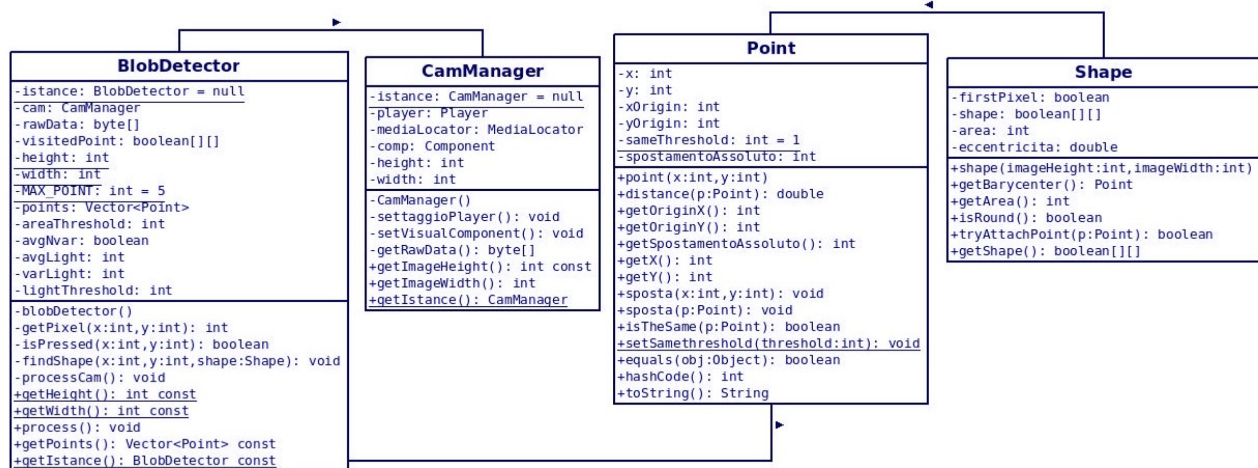
Una volta ottenuto il componente di top-level per la posizione dell'evento che stiamo esaminando, inseriamo l'evento nel vettore degli eventi per quel componente, così che una volta esaminati tutti gli eventi, li troveremo associati ai componenti appropriati. A questo punto possiamo scorrere la struttura realizzata e per ogni componente lanciare sul listener i vari eventi. Scorriamo quindi anche la lista degli eventi per il componente e li lanciamo direttamente (se click o release) o ne valutiamo il comportamento (se press o move) per la costruzione di eventuali eventi di tipo zoom (che sono realizzati da press+move e move+move).

Componenti Grafici

Il gestore degli ascoltatori invoca le azioni successivamente alla comparsa degli eventi.
Per poter legare i componenti grafici agli eventi si fa uso degli ascoltatori. Nel nostro caso, un ascoltatore è un interfaccia che definisce ma non implementa le azioni corrispondenti ad ogni evento. E' compito del programmatore implementare le azioni volute. Quando si crea un componente sarà necessario legarlo all'ascoltatore aggiungendo un riferimento Componente-Ascoltatore nella struttura hash del dispatcher in modo che quando un evento nella posizione del componente viene scatenato esso sarà in grado di invocare le azioni sull'ascoltatore definito.

2.3 Il riconoscimento delle forme

Il package touchRecognizer è composto da quattro classi: CamManager, BlobDetector, Point, Shape.



CamManager

La classe CamManager sfrutta il design pattern singleton per istanziare un unica entità di tipo CamManager, invocando getIstance saremo in grado di ottenere l'istanza se già esistente o altrimenti provvedere alla invocazione del costruttore. Il costruttore cicla sullo stato della webcam effettuando un polling temporizzato fino a quando essa non diviene disponibile. In questo modo si ritorna il controllo del flusso delle istruzioni alla funzione client (funzione chiamante) solo quando la webcam è effettivamente istanziata e pronta a catturare immagini (e si evitano così eccezioni o errori sulle immagini). Il metodo più importante della classe è getRawData che si occupa dell'acquisizione dei dati dal buffer della webcam (restituisce un array di byte quando presenti altrimenti restituisce null) ottenendo informazioni sul formato dell'immagine catturata ed effettuando una conversione da formato video ad array di byte in modo che sia possibile recuperare il solo canale rosso e velocizzare l'image processing (effettuandolo su un solo canale anziché tre).
E' possibile utilizzare un solo canale di colore per il fatto che il filtro posto sulla webcam (per bloccare la luce ambientale e far passare gli infrarossi) e le impostazioni hardware applicate, fanno si che l'immagine acquisita sia in toni di grigio e i punti premuti di colore biancastro, il che rende i canali praticamente simmetrici l'un l'altro e le informazioni ridondanti. E' stato quindi deciso di restituire il solo canale rosso.

Point

La classe Point rappresenta una delle entità più importanti del framework, i punti. Un punto ha delle coordinate di origine in cui è stato riconosciuto per la prima volta, la sua posizione attuale nello spazio (solo in questo modo è possibile tracciarne gli spostamenti) e la quantità di spazio percorsa dalla sua comparsa (spostamentoAssoluto). Quando un nuovo punto è costruito la posizione d'origine coincide con l'attuale e lo spazio percorso è ovviamente zero.
La classe Point fornisce diversi metodi di accesso alle caratteristiche dell'oggetto.

Metodo	Informazioni
distance	Ritorna la distanza tra il punto su cui il metodo è invocato e quello passato per parametro
equals	Ritorna true se il punto su cui il metodo è invocato ha stesse coordinate di quello passato per parametro
toString	Ritorna una rappresentazione in formato stringa del punto

getX	Restituisce l'ascissa del punto
getY	Restituisce l'ordinata del punto
getSpostamentoAssoluto	Restituisce il valore dell'attributo spostamentoAssoluto
sposta	Il metodo sposta (parametrizzato da una posizione nello spazio) invocato su un punto fa si che la posizione di tale punto sia aggiornata. Nel caso la nuova posizione sia distante oltre una certa soglia dalla precedente tale variazione è considerata come spostamento ed è perciò aggiornato anche l'attributo spostamentoAssoluto.
isTheSame	Se la distanza dalla posizione iniziale del punto alla posizione attuale è minore della soglia allora stiamo parlando dello stesso punto che si è mosso di pochi pixel altrimenti stiamo parlando di uno spostamento vero e proprio. Se per il punto era già in atto uno spostamento (ovvero se spostamento assoluto è maggiore di zero) dovremo continuare a mantenerne traccia restituendo false in modo che il metodo sposta (invocante) sia in grado di incrementare lo spazio percorso. Altrimenti restituiremo true in quanto stiamo parlando di un leggero movimento del punto che può essere trascurato e non valutato come movimento vero e proprio.
setSameThreshold	Imposta la soglia per cui punti in posizioni diverse possono essere ricondotti ad uno spostamento dello stesso.

Shape

La classe Shape fornisce funzionalità di gestione delle forme. Il costruttore di Shape setta a true l'attributo firstPixel e inizializza a false una matrice di booleani delle dimensioni dell'immagine catturata. Tale matrice è utilizzata per tenere traccia dei punti che compongono una forma (La matrice è ottenibile invocando il metodo getShape). Ogni forma (Shape) ha una sua area data dalla somma dei punti che ne fanno parte (essa è ottenibile mediante invocazione del metodo getArea).
La definizione del dominio delle forme avviene tramite il metodo tryAttachPoint, il quale parametrizzato da un punto, controlla se nel suo intorno sono presenti altri punti riconosciuti o se è il primo punto della forma. TryAttachPoint inserisce true nella matrice di booleani (aggregando il punto alla forma) incrementandone l'area se:

- stiamo controllando il primo pixel della forma (ovvero se firstPixel è true significa che stiamo visitando il punto per cui la nuova Shape è stata creata e quindi sicuramente ne farà parte), in questo caso per la regola di cortocircuitazione dell'or possiamo subito entrare nell'if ed incrementare l'area da 0 a 1, porre a true il valore di quel pixel nella matrice e impostare a false firstPixel per i controlli successivi.

Per i punti diversi dal primo riconosciuto:

- quando il punto è in x > 0 e alla sua sinistra (nella posizione x = 0 se il punto da verificare è in posizione x = 1) è già stato riconosciuto un altro punto.
- quando il punto è in x < (larghezza Immagine – 1) e alla sua destra è già stato riconosciuto un altro punto. (nel caso limite quando x = larghezza immagine – 2 effettueremo il controllo su x = larghezza immagine – 1 che coincide con il bordo laterale destro)
- Quando il punto è in y > 0 e sopra di se (nella posizione y = 0 se il punto da verificare è in posizione y = 1) è già stato riconosciuto un altro punto.
- Quando il punto è in y < (lunghezza immagine – 1) e sotto di se è già stato riconosciuto un altro punto. (nel caso limite y = lunghezza immagine – 2 controlleremo il punto nella posizione y = lunghezza immagine -1 che coincide con il bordo in basso)

Ogni volta che tentiamo di aggregare un punto alla matrice ci sarà restituito true o false in base all'esito di tale tentativo.
Una volta costruita la forma (la matrice di booleani) saremo in grado di ottenere il baricentro della figura rilevata invocando il metodo getBarycenter su di essa. GetBarycenter scorre la matrice in cerca di punti associati alla forma e quando ne trova uno ne somma le cordinate alle variabili locali x e y incrementando poi il contatore dei punti trovati. Per ogni punto trovato attivo ne controlla la

posizione e se superiore a quella massima o inferiore a quella minima di cui si è tenuta traccia fin ora la si sovrascrive in modo da arrivare al termine della visita sulla matrice con i limiti in x e y della forma.
A questo punto è possibile ottenere il baricentro come punto medio per x e per y semplicemente dividendo prima x e poi y per il numero di punti trovati, mentre è possibile rapportare la larghezza all'altezza per calcolare l'eccentricità della figura, proprietà che ci sarà utile ad eliminare riflessi luminosi o rumore sulle immagini che non hanno solitamente forma circolare.

BlobDetector
La classe BlobDetector si occupa di istanziare il gestore della camera e di processare i dati provenienti da essa. Anche BlobDetector utilizza il design pattern singleton ed è quindi istanziabile una sola volta. Il costruttore inizializza le soglie per la dimensione minima delle forme, l'intensità luminosa dei pixel premuti, l'energia media e la varianza associata ad essi. Nel primo frame la soglia è impostata sul bianco in modo che anche con luce ambientale forte nessun punto possa essere riconosciuto.
Tra gli attributi di blobDetector troviamo anche il vettore dei punti riconosciuti (baricentri delle forme), la matrice dei punti visitati, la dimensione delle immagini acquisite e la matrice di byte in cui tali dati sono allocati. Per accedere a tali membri sono stati inseriti appositi metodi quali getWidth, getHeight, getPoints.
L'accesso ai dati provenienti dalla webcam è garantito dal metodo ProcessCam che utilizza il riferimento all'oggetto di tipo CamManager per ottenere i dati grezzi (array di byte rappresentanti il canale rosso) e le dimensioni delle immagini (che poi memorizzerà negli attributi della classe blobDetector). Finché i dati non sono completi (ovvero finché getRawData ci restituisce null, oppure l'array restituito è incompleto ossia minore di width * height) invalideremo i dati propagandoli come null in modo da non creare errori nella futura fase di processing. Infine inizializzeremo a false la matrice dei punti visitati dopo averla eventualmente creata (se non esistente).
La fase più importante è quella di riconoscimento delle forme che avviene per opera del metodo process. Dopo aver pulito il vettore dei punti con una clear (era stato utilizzato per i frame precedenti) invoca il metodo processCam per ottenere l'immagine dalla camera.
A questo punto inizia a ciclare sulle righe e per ogni pixel (ogni colonna) invoca il metodo getPixel(x, y) con la quale ne ottiene l'intensità luminosa.
Il metodo getPixel accede alla matrice di byte rappresentante il canale rosso dell'immagine acquisita (ogni pixel è di tipo byte che in java è un intero a 8 bit con segno, il cui valore è compreso tra -128 e +127): per operare esclusivamente su dati positivi, trasliamo i punti negativi (sommando +256) rendendoli positivi.
La soglia adattiva:

> *Per permettere l'utilizzo di una soglia adattiva è stato deciso di calcolare l'intensità luminosa media e la varianza media interlacciando tale calcolo in modo da attribuire il primo ai frame pari e il secondo ai dispari. Per questo motivo, una volta ottenuto il valore del pixel, è aggiornato il valore della luminosità dell'immagine (sommandogli il valore del pixel) o quello della varianza totale (sommando il quadrato della differenza dall'intensità luminosa media) in modo che al termine della visita sulla matrice sia possibile calcolare l'intensità media come rapporto tra l'intensità luminosa totale e la dimensione dell'immagine e la varianza da essa come radice quadrata della varianza media che è a sua volta calcolata come rapporto tra la somma delle varianze dal valore medio e la dimensione dell'immagine. A questo punto, avendo a disposizione media e deviazione standard della luminosità dell'immagine, sappiamo che sicuramente i punti premuti avranno un'intensità maggiore della media sommata a due volte la deviazione standard: osservando la campana di gauss capiamo infatti che il 95.5 % delle osservazioni ricadono all'interno di m ± 2 sigma, dunque i nostri punti luminosi essendo “speciali” saranno con*

tutta probabilità nel rimanente 2,25% (dato che m – 2 sigma non ci interessa).
Con questa configurazione però, nel caso in cui non si facesse pressione con nessun dito, in quel 5% comparirebbe il rumore dell'immagine: è per questo che abbiamo deciso di aggiungere un ulteriore fattore correttivo, tarato al 20% (fissato empiricamente) della deviazione standard. In questo modo siamo ancora più vicini all'individuare correttamente solamente i punti di pressione veri e propri.
C'è un'ultima osservazione che merita l'attenzione in merito alla soglia adattiva: come detto, per non inficiare le prestazioni, media e varianza vengono calcolate in modo interlacciato. Per chiarire i problemi teorici che questo accorgimento implica prendiamo il caso in cui la media venga calcolata al frame x. Dunque al frame x, la soglia sarà relativa ancora ai frame passati, mentre la media sarà disponibile solo alla fine del frame, quando ormai avremo scorso tutti i pixel. Al frame x + 1, è il turno della deviazione standard: la nuova soglia dinamica viene calcolata con la media del frame x e la deviazione standard passata. Al frame x + 2, la soglia viene calcolata tramite la media del frame x e la varianza del frame x + 1, mentre viene ricalcolata la media. Questo ci fa capire due cose molto importanti:

- *la prima è che media e varianza, profondamente legate tra loro, vengono calcolate su basi diverse;*
- *la seconda forse più sconcertante è che al frame x + 2, stiamo utilizzando come riferimento la media del frame x, ovvero un valore calcolato 2 frame prima, quando le condizioni ambientali potevano essere diverse dalle attuali.*

Quindi potrebbero sorgere molti dubbi sull'efficacia di questo metodo, ma è sufficiente ricordare che un frame ha la durata di circa 33 millisecondi, dunque anche se i parametri vengono valutati effettivamente ogni 3 frame, lo scostamento temporale di un decimo di secondo è veramente molto basso, tanto da poter considerare i 3 frame uguali fra loro.

Per ogni pixel visitato, il metodo process cerca di riconoscere eventuali forme a cui il punto può essere ricondotto nel caso esso superi la soglia di luminosità minima e nel caso esso non faccia già parte di altre figure. Nel caso in cui il punto sia riconosciuto come illuminato e non ancora visitato nella ricerca di altre forme, si procede alla creazione di una nuova forma (un nuovo oggetto di tipo Shape della dimensione dell'immagine acquisita) e la si passa al metodo findShape per la sua identificazione.
Il metodo findShape all'interno della classe blobDetector, controlla che il punto non sia già stato visitato e in quel caso lo visita (impostando a true il suo valore nella matrice visitedPoint) e cerca di aggregarlo alla nuova forma passata come parametro invocando il metodo tryAttachPoint su di essa. In caso affermativo nella matrice shape dell'oggetto Shape è inserito true alla posizione del punto per indicare la sua appartenenza alla forma. A questo punto è necessario visitare i pixel adiacenti in modo da trovare altri punti appartenenti alla stessa forma. Se il punto alla sua destra è ancora nell'immagine (ovvero inferiore alla sua larghezza) lo visito invocando ricorsivamente la funzione findShape su esso. Allo stesso modo per il punto alla sua sinistra e nella posizione soprastante e sottostante si invoca ricorsivamente findShape per provvedere alla visita dei pixels e alla rilevazione della forma. La ricorsione termina non appena il pixel su cui è invocata risulta già visitato o non premuto. A questo punto abbiamo la nostra forma.
Il metodo process può ora proseguire con un controllo sull'area e sull'eccentricità dell'oggetto Shape costruito. Se l'area è superiore a quella minima consentita la forma viene considerata rilevante. Resta da verificare l'attendibilità di tale figura in quanto le immagini sono spesso colpite da rumore e da inquinamento luminoso. Per questo motivo viene invocato il metodo isRound sulla forma che restituisce True se essa tende ad essere circolare. Quindi solo nel caso di dimensioni rilevanti e caratteristiche di rotondità possiamo validare una forma e aggiungere il baricentro al vettore dei punti riconosciuti.
A questo punto abbiamo terminato la rilevazione della forma costruita sul pixel visitato ma è

necessario adattare la risoluzione della webcam (molto bassa) a quella dello schermo visualizzato, in modo che i punti assumano una posizione direttamente proporzionale ad esso.
La posizione di un punto sullo schermo sarà data dalla posizione relativa del punto sull'immagine moltiplicata alla dimensione dello schermo su cui sarà visualizzata:
*posizione punto sullo schermo = posizione relativa del punto sull'immagine * dimensione schermo*

dove la posizione relativa può essere vista come
ascissa relativa = ((larghezza immagine – 1) – ascissa punto) / (larghezza immagine - 1)
ordinata relativa = ((altezza immagine – 1) – ordinata punto) / (altezza immagine - 1)

** è necessario sottrarre – 1 alle dimensioni per adattarle alla posizione del punto compresa tra 0 e (dimensione - 1)*

che sul piano si traduce come:

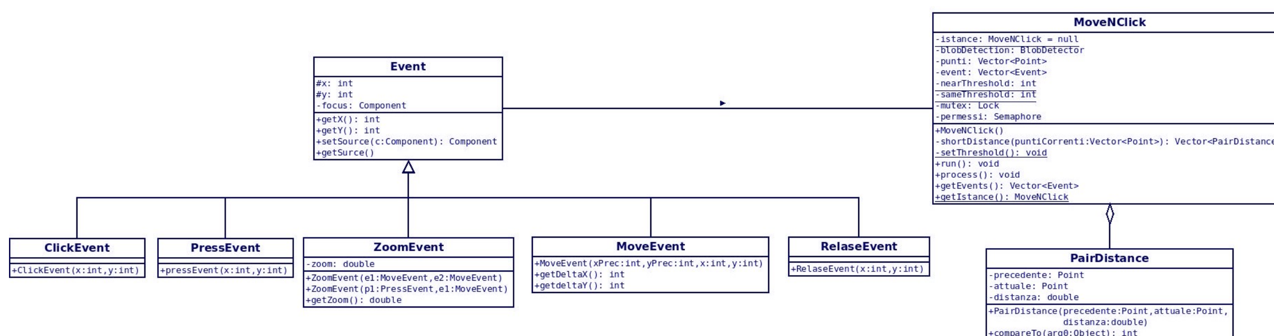
ascissa del punto = (((width – 1) – ascissa del punto sull'immagine) / (width – 1))
** ScreenSize.width*

ordinata del punto = (((height – 1) – posizione del punto sull'immagine) / (height - 1))
** ScreenSize.height*

A questo punto possiamo aggiungere il nuovo punto rilevato e se il numero di punti trovati (presente nel vettore dei punti) non supera quello limite possiamo passare a visitare gli altri pixel dell'immagine in cerca di altre forme.

2.4 La gestione degli eventi

Una volta ottenuto il vettore dei punti premuti è necessario interpretarne il comportamento nel tempo in modo che l'insieme dei baricentri possa essere partizionato secondo l'evento di appartenenza.
Un evento è una particolare situazione o configurazione che si viene a creare o "innescare" nel momento in cui l'utente utilizzatore interagisce con l'interfaccia di comando. Essendo la nostra una interfaccia di tipo multitouch, avremo sicuramente diversi punti che dovranno essere relazionati e distinti in modo da essere associati ognuno all'evento appropriato. Ciò significa che l'interfaccia dovrà permettere lo scatenarsi di più eventi allo stesso tempo.
Ma di che tipo di eventi potremo avere bisogno nelle nostre applicazioni?
Sicuramente il fatto che stiamo gestendo un sistema di puntamento ci porta a dover considerare le pressioni e i rilasci, ovvero la generazione di eventi di tipo press, release e click.
Il fatto che l'interfaccia sia multitouch ci permette di gestire anche spostamenti di punti correlati che possono essere considerati per esempio come zoom.
Sfruttando il polimorfismo del linguaggio ad oggetti utilizzato, saremo in grado di realizzare una classe astratta chiamata event e scrivere una gerarchia di eventi che la specializza.



Avremo quindi la classe evento e i suoi figli PressEvent, MoveEvent, ReleaseEvent, ClickEvent e ZoomEvent.
Ogni evento avviene in una determinata posizione dello schermo all'interno di un componente grafico.
Gli eventi di tipo click, press e release non introducono contenuto informativo rispetto a quelle della classe Event ma è utile la loro nominazione perchè permette di discriminare i diversi tipi di evento e quindi le diverse azioni che si dovranno intraprendere in loro presenza.
Gli eventi di tipo move si presentano in corrispondenza dello spostamento di un punto e per questo motivo tengono traccia della posizione precedente (che coincide con la posizione dell'evento) e della posizione attuale permettendo di calcolarne la distanza.
Gli eventi di tipo zoom scaturiti su un componente, possono presentarsi in corrispondenza di due spostamenti o di un press e uno spostamento. In questo modo quando la distanza tra due punti vicini viene modificata, la quantità di zoom sarà data dal rapporto tra la nuova distanza e la vecchia mentre il baricentro di tale evento sarà dato dal punto medio rispetto alla posizione dei due eventi coinvolti.
Quindi, in linea teorica, potremmo vedere il sistema base come set di press e release mentre ad un livello superiore potremmo collocare i click e i move ed infine gli zoom nel livello più alto.
Nella realtà però abbiamo deciso di utilizzare press per i punti non ancora rilasciati e il cui eventuale movimento è inferiore ad una certa soglia. Quando il press è rilasciato viene scaturito un evento di tipo click sul punto. Nel caso in cui il movimento supera la soglia è scatenato un evento di tipo move e al suo rilascio un evento di tipo release.
Vediamo però come tali eventi possono essere rilevati e gestiti introducendo la classe MoveNClick delegata a tale scopo.

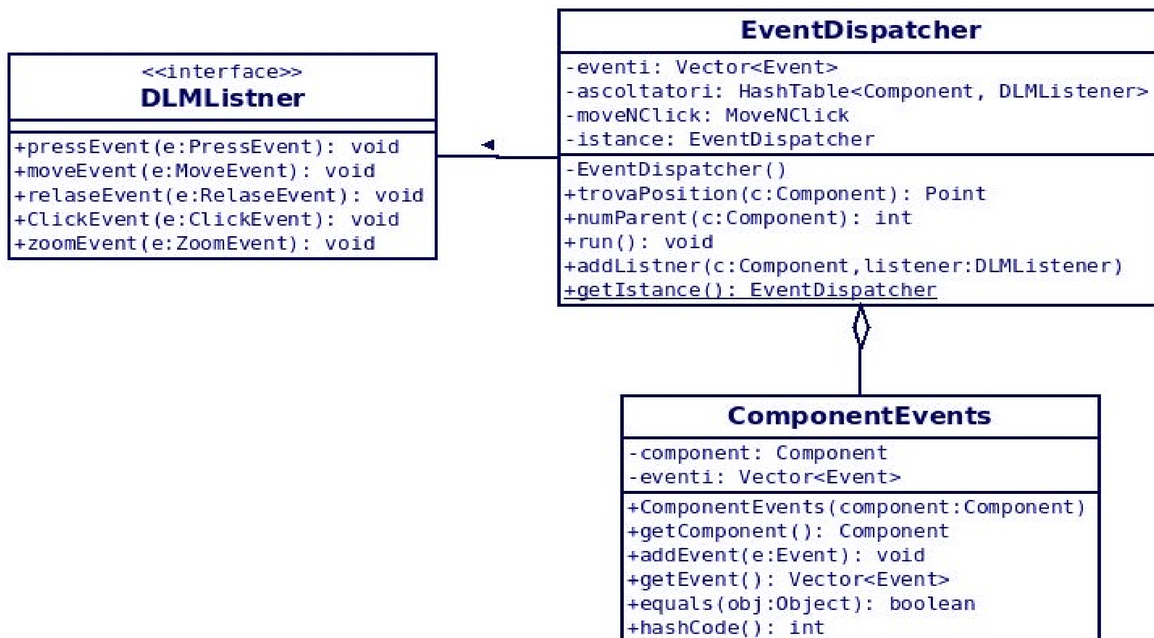
MoveNClick
La classe MoveNClick definisce le relazioni tra i punti e scatena gli eventi associati a tali relazioni.
Anche in questo caso è stato usato il design pattern singleton ed è quindi necessario richiamare getIstance per ottenere l'istanza dell'oggetto MoveNClick dopo averlo eventualmente creato.
L'oggetto MoveNClick gestisce il blobDetector sottostante e mediante un apposito thread ne processa i risultati. Il thread di servizio è chiuso in un ciclo infinito la cui esecuzione è alternata da sleep che evitano in tal modo la monopolizzazione dei cicli della CPU.
In questo modo, dato il frame rate a 30fps della webcam (30fps significa 1 immagine ogni 33 millisecondi) siamo in grado di processare in media una immagine ogni 30 millisecondi visto che se il tempo impiegato è inferiore ai 30 ms si manda il thread in sleep per il tempo restante. All'interno del thread quindi si processano i punti, si settano le soglie di similarità e si tiene traccia della durata d'esecuzione per la valutazione della sleep.
Dopo la prima esecuzione in cui eventuali forme sono scartate, vengono settate le soglie di similarità ovvero quel valore per cui due punti possono essere considerati correlati (nearThreshold) e quello per cui può essere ricondotto allo stesso punto (sameThreshold). Inizialmente era stata settata a 30 la prima e a 5 la seconda ma il fatto che le soglie dipendono dalla risoluzione dello schermo che visualizzerà i punti ci ha fatto propendere a delle soglie relative. Abbiamo trovato la dimensione della soglia rispetto alla webcam e la abbiamo moltiplicata per le dimensioni dello schermo.

*NearThreshold = (30 / dimensione Immagine Webcam) * larghezza schermo*
*sameThreshold = (5 / dimensione Immagine Webcam) * larghezza schermo*

Tali soglie sono infine propagate tramite un metodo statico alla classe punto.
Il processing è invece svolto da un apposito metodo chiamato process che dopo aver pulito il vettore degli eventi precedenti invoca il blobDetector per nuovi punti e successivamente il metodo sortDistance per ordinare le coppie di punti secondo un ordine di distanza crescente.
SortDistance è parametrizzata dal vettore dei punti trovati attualmente e cicla su tale struttura e sul vettore dei punti trovati nel frame precedente (secondo un prodotto cartesiano) creando per ogni coppia di punti visitata un oggetto di tipo PairDistance (che aggiunge al vettore delle coppie) costituito dalla posizione dei due punti e la loro distanza. Infine SortDistance prende il vettore delle coppie e le relative distanze e le ordina in maniera crescente.
Il vettore restituito all'invocante process è visitato elemento per elemento e ogni punto esaminato è registrato in un apposito vettore in modo che data una coppia possiamo valutarne il comportamento se e solo se nessuno dei due è già stato esaminato. Ciò risulta ovvio, visto che se il vettore è ordinato in ordine crescente per distanza tra punti, i primi ad essere processati saranno quelli più vicini che non sono ancora stati visitati. A questo punto controllo che la coppia che si sta esaminando sia ad una distanza inferiore a quella definita di correlazione: se non lo è, i punti non sono correlati tra loro e siccome da li in poi le coppie che esamineremo avranno distanza crescente nemmeno loro potranno essere correlate. In questo caso posso evitare di ciclare nella visita e andare avanti. Se, al contrario, la distanza tra due punti cade all'interno della soglia di correlazione essi sono in relazione tra loro e dovremo verificare se vi è possibilità che sia un movimento o una pressione dello stesso punto in frame differenti. Tale verifica viene effettuata mediante invocazione del metodo isTheSame posto sul punto a cui passiamo il punto da confrontare. Se la distanza tra i due è inferiore alla soglia definita allora siamo di fronte ad un evento di tipo pressione (con movimento trascurabile) e non ci resta che spostare il punto alla nuova posizione e aggiungere al vettore degli eventi un nuovo evento di tipo press. Se al contrario lo spostamento supera la soglia il metodo isTheSame ritornerà falso e ciò identificherà una relazione di spostamento tra i due punti. In questo caso aggiorneremo la posizione e aggiungeremo un nuovo evento di tipo move al vettore. A questo punto non resta che aggiungere al vettore dei punti esaminati i punti della coppia visitata in modo da non poterli associare più ad altri.
Una volta che la distanza inizia ad essere alta la visita delle coppie termina e i punti rimanenti e non ancora esaminati non possono essere correlati e devono quindi essere considerati singolarmente.

Ora cicliamo sui punti presenti al frame precedente e, per ognuno di essi, controllo se è presente nel vettore dei punti esaminati: se è presente esso è stato già associato con un punto del frame corrente e quindi possiamo andare avanti al prossimo. Se al contrario il punto non è presente nel vettore dei punti esaminati allora era presente al frame precedente ma è troppo distante da qualsiasi altro punto del frame attuale e quindi è considerato rilasciato. Va ora definito se tale rilascio sia successivo ad un press o ad un move (perché nel primo caso diverrebbe un evento click mentre nel secondo release) e per farlo controllo il suo spostamento assoluto. Se lo spostamento assoluto è 0 allora siamo di fronte al rilascio di un press (per scelta progettuale i press non incrementano lo spostamento assoluto in modo che se ci muoviamo lentamente e rimaniamo all'interno della soglia di spostamento non veniamo considerati come movimenti mentre se ci muoviamo lentamente ma per lunghi tratti la distanza attuale del punto rispetto alla posizione d'origine supera quella di soglia e sarà da quel momento considerato spostamento) e non ci resta che aggiungere al vettore degli eventi il nuovo click trovato (nella posizione da cui il press si è originato). Se lo spostamento assoluto è maggiore di 0 siamo di fronte ad un release e non ci resta che inserirlo nel vettore degli eventi. Concludiamo rimuovendo dal vettore dei punti del frame precedente il punto rilasciato.
Ora ciclo sui punti presenti al frame attuale e ogni punto che non è stato accoppiato con nessun altro sarà ritenuto come nuovo e sarà quindi aggiunto un nuovo evento press nella posizione data dal suo baricentro. Tale punto sarà inserito nel vettore punti che ci servirà nel frame successivo per definirne il comportamento.
Per recuperare il vettore degli eventi è possibile utilizzare il metodo pubblico getEvents. Ovviamente essendo in ambito multithread la risorsa vettore degli eventi sarà condivisa sia dal dispatcher degli eventi (che invocherà getEvents) che dal thread dell'oggetto MoveNClick che invece processerà i punti per aggiungere gli eventi associati. Per garantire la mutua esclusione nell'accesso è stato utilizzato un meccanismo di lock che permette accessi distinti da parte del lettore rispetto allo scrittore.

2.5 La gestione degli ascoltatori e dei componenti grafici



Il dispatcher degli eventi

Il dispatcher degli eventi è una entità incaricata della gestione degli ascoltatori istanziati sui relativi componenti grafici. Esso riceve un vettore di eventi dal livello sottostante e provvede a notificare al relativo componente l'evento scaturito.

Anche il dispatcher è realizzato tramite il designPattern singleton ed è quindi possibile ottenere il riferimento dell'istanza richiamando getIstance. Il costruttore inizializza una struttura dati hash per la gestione dei riferimenti componente-ascoltatore e istanzia un oggetto di tipo MoveNClick (notificatore di eventi) che autonomamente inizierà a produrre risultati.

Una volta costruito, il dispatcher avvia un thread di lavoro in cui si chiude per tutta la durata dell'esecuzione. In tale flusso di istruzioni acquisisce il vettore degli eventi dal notificatore, inizia poi a scorrerlo e per ognuno ne recupera la posizione (sullo schermo). Per l'evento che sta visitando inizia a scorrere l'hash table in cerca di un componente a cui passarlo. Tale ricerca si conclude non appena viene trovato un componente visibile presente nella posizione in cui l'evento è stato rilevato (ovvero se la x e la y dell'evento sono comprese tra la x e la x+width e tra la y e le y+height del componente). In questo caso viene invocato setSource sull'evento per associarlo al componente. Le interfacce grafiche però sono strutturate in un albero di componenti, potremmo avere infatti diversi componenti annidati e in questo caso la scelta di quello su cui scatenare l'evento si complica. Per questo motivo utilizziamo un indicatore di profondità inizializzato a zero in modo che quando viene visitata l'hash table per la prima volta viene preso il componente visibile con profondità nell'albero (numero di nodi dalla radice al componente) maggiore di zero (potenzialmente tutti visto che il padre della gerarchia ha profondità 1). Quando tale componente è preso è registrato come componente di livello più alto e la sua profondità viene utilizzata per valorizzare il valore di profondità (“da battere”) per il ciclo successivo. In questo modo quando visitiamo il componente successivo siamo in grado di determinare se la sua posizione è superiore o inferiore (sul piano) a quella del componente fino ad ora ritenuto di top-level. Quando viene trovato un componente con profondità più alta di quello considerato fino a quel punto, è sostituito al vecchio e allo stesso modo il valore di profondità viene aggiornato. In questo modo appena terminiamo il ciclo sui componenti abbiamo preso sicuramente quello con profondità più alta. Ma come è possibile trovare la profondità di un componente sull'albero? A questo scopo abbiamo utilizzato un metodo numParent

che riceve un componente e invoca su se stesso il metodo getParent che restituisce il riferimento al padre. Se tale invocazione restituisce null allora il componente che stiamo esaminando è la radice dell'albero e la sua profondità è quindi 1. Altrimenti se ci è restituito un riferimento al padre la profondità del componente è data dalla somma della profondità del padre più 1 (quella del figlio rispetto al padre). Per questo motivo agendo iterativamente siamo in grado di giungere fino alla radice e trovare così la profondità del componente rispetto ad essa.
Ogni componente dell'interfaccia grafica viene mappato con la lista degli eventi presenti in esso.
Per questo motivo una volta ottenuto il componente di top-level per l'evento che si sta esaminando recuperiamo il contenitore degli eventi per quel componente (o nel caso non esista provvediamo a crearlo) e vi aggiungiamo il nuovo evento trovato. Una volta esaminati tutti gli eventi e aggiunti ognuno al proprio contenitore di componente non resta che lanciarli sui rispettivi listener.
Scorriamo quindi il vettore dei contenitori dei componenti e per ogni componente otteniamo il listener direttamente dalla hashTable, dato il listner prendiamo il vettore degli eventi contenuto all'interno del contenitore e ciclando su di esso lanciamo in maniera diretta gli eventi di tipo click e release mentre registriamo il primo evento di tipo press (gli altri eventuali sono lanciati sul listener) e fino a due eventi di tipo move (gli altri sono lanciati sul listener). In questo modo siamo in grado di catturare eventi di tipo zoom sul componente.
Se abbiamo due move creiamo un evento di tipo zoom e lanciamo a parte l'eventuale evento press.
Se abbiamo un solo move e nessun press, aggiungiamo l'evento move sul listener.
Se abbiamo un move ed un press aggiungiamo l'evento zoom al listener.
Se abbiamo un solo press aggiungiamo l'evento al listener.
Con il termine “lanciamo gli eventi sull'ascoltatore” si intende l'operazione con cui a partire da un ascoltatore e un evento, si richiama uno tra i metodi pressEvent(PressEvent e), moveEvent(MoveEvent e), releaseEvent(ReleaseEvent e), clickEvent(ClickEvent e) e zoomEvent(ZoomEvent e) presenti. All'interno del corpo di tali metodi sono contenute le azioni da intraprendere in corrispondenza di un evento.

I componenti grafici

Un componente Java è un oggetto che può essere visualizzato sullo schermo all'interno delle interfacce grafiche. Ne sono esempi i bottoni e le scroolbar. La superclasse astratta component contenuta nel package grafico AWT definisce proprio i meccanismi base con cui un oggetto grafico può essere rappresentato e utilizzato.
I componenti di maggior utilizzo e interesse sono JFrame, JButton, JWindow e JPanel.
Il problema è che tali componenti sono sensibili agli eventi standard e quindi non in grado di servire i nostri.
Per provvedere a tale mancanza è stato introdotto il package OxygenGraphics contenente i componenti base (e potenzialmente ampliabile) utilizzabili per la scrittura di applicazioni anche da parte di chi non conosce affatto il framework.
Abbiamo quindi definito i seguenti componenti:

- **O2Button**, estende la classe JButton, implementa l'interfaccia O2Component
- **O2Frame**, estende la classe JFrame, implementa l'interfaccia O2Component
- **O2Window**, estende la classe JWindow, implementa l'interfaccia O2Component
- **O2Panel**, estende la classe JPanel, implementa l'interfaccia O2Component

In questo modo siamo in grado di utilizzare i componenti Java (e gli eventi base associati) con tutte le loro caratteristiche e allo stesso tempo utilizzare i nostri eventi e il nostro dispatcher.
Possiamo vedere il fatto che sia derivato sia il componente base che quello del package Oxygen come una ereditarietà multipla (non permessa in maniera diretta in Java).
L'interfaccia O2Component infatti contiene il solo metodo addDLMListener con cui è possibile da parte dei componenti O2 aggiungere un ascoltatore agli eventi del DLMSensitive sul componente. In questo modo gli eventi Java sono risolti dal dispatcher Java mentre i nostri parallelamente sono gestiti dall'EventDispatcher.
Così per gestire i componenti del pacchetto Oxygen basterà creare un ascoltatore, ovvero istanziare un oggetto di una classe che implementa DLMListener (DLMListener è un interfaccia quindi per istanziarla dobbiamo definire un altra classe listener che implementa tutti i suoi metodi).
A questo punto abbiamo un listener per gli eventi della nostra applicazione (e con le azioni ben definite in corrispondenza di ogni evento), possiamo iniziare a scrivere la nostra interfaccia inserendo componenti del pacchetto Oxygen (come lo faremmo per i componenti standard) e invocare il metodo addDLMListener (parametrizzato dal riferimento al listener) su ognuno di essi in modo che essi possano iscriversi alle notifiche eventi del dispatcher (se lo vediamo come un sistema publish/subscribe i componenti sono i clienti che vogliono monitorare gli eventi e per questo si sottoscrivono all' Observer, svolto in questo caso dall'eventDispatcher che notificherà “pubblicando” gli eventi trovati ai vari componenti).

2.6 Testing

Per il processo di testing è stato deciso di realizzare una piccola applicazione (comunque molto rappresentativa delle potenzialità del framework) che gestisce una cartella di foto dando la possibilità di inserirle in una scrivania virtuale, spostarle e ridimensionarle.
Vista la semplicità della applicazione non parleremo del funzionamento algoritmico o del codice ma passeremo direttamente alla esposizione dei risultati del processo di testing.

Sistemazione del proiettore e del pannello

Selezione di più elementi sul piano (press, click):

**come si nota le 4 foto selezionate hanno bordo magenta anziché arancio*

Scaling di una foto (zoom):

Scaling tramite:

– un press e un move	ok
– due move	ok

Selezione e spostamento di una foto (move, release):

**come si nota la foto selezionata ha bordo magenta anziché quello standard arancio*

Conclusioni

Il testing del framework per mezzo dell'applicativo multitouch realizzato ha permesso la formulazione di diverse considerazioni riguardo il lavoro svolto.

- **Il software è una applicazione relativamente leggera.**
  Il testing su un pc del 2002 di tipo barebone con processore Athlon XP 2200+ 1.8GHz 256MB RAM è stato svolto al 70/80% delle sue potenzialità di calcolo (utilizzo processore) mentre la stessa prova su un computer portatile del 2007 (Sempron Mobile 1.8 Ghz 512 MB RAM) è stata svolta al 50/60% dell'utilizzo della CPU.

Funzionalità	Complessità computazionale	Note
Riconoscimento forme	O(m * n)	m = righe n = colonne
Riconoscimento Eventi	O(p ^ 2)	p = numero punti
Notifica eventi	O(c * e)	c = numero componenti iscritti al dispatcher e = numero di eventi riconosciuti

- La bassa risoluzione della webcam non ha influenzato significativamente la precisione del puntamento.
  L'utilizzo di webcam migliori dal punto di vista ottico e della risoluzione ci lascia quindi ampi margini di miglioramento.