Politecnico di Torino

Computer Vision

How to use: LearnOPENCV

Bartolotta Marco

Facchini Dario

Anno Accademico 2013/2014

Funzioni principali

MainWindow

La classe MainWindowGUI.java esprime in una schermata le principali funzioni offerte dal programma:

- la colonna a sinistra contiene sorgente e destinazione con pulsanti per il caricamento con “Load Image” e il salvataggio del risultato finale con “Save Image”;
- la colonna centrale conterrà le unità di elaborazione, eseguite in ordine dall’alto verso il basso. Tramite pulsante “Add” sarà chiamato EditBlock, responsabile del caricamento degli stessi. “Clear All” consente invece di eliminare la lista corrente. Ogni blocco aggiunto presenta una piccola interfaccia che permette di:
    - attivare/disattivare l’effetto del blocco, tramite CheckBox “Active”
    - visualizzare lo stato corrente del blocco:
        - grigio: il blocco è stato disattivato
        - verde: il blocco è pronto per eseguire
        - giallo: il blocco può eseguire ma descrizione o codice non saranno disponibili
        - rosso: è avvenuto un errore, riportato nell’area di log inferiore
        - lampeggiante: il blocco è stato modificato ma non ancora eseguito
    - spostare su/giù il blocco tramite trascinamento del mouse
    - visualizzare l’istogramma del risultato del blocco (tasto “H”)
    - visualizzare la trasformata di Fourier del risultato del blocco (tasto “F”)

  I pulsanti “Check” ed “Execute” permettono di lanciare il primo una diagnostica, tenendo conto però che alcuni blocchi potrebbero non avere un input se non precedentemente eseguiti (saranno riportati almeno gli errori sui setting dell’utente), mentre il secondo di eseguire la pipeline e produrre un risultato se nessun blocco genera errore. “Clear Log” infine svuota l’area di log inferiore.
- la colonna di destra che contiene informazioni sul blocco selezionato, eccetto per il Tab “Code” che conterrà il codice equivalente all’ultima sequenza di operazioni eseguita se avvenuta con successo:
    - Tab “Info”: contiene la descrizione del blocco, se presente
    - Tab “Control”: contiene in ordine le impostazioni del blocco, che ne modificano il comportamento
    - Tab “Details”: riporta ultimo input e output del blocco

I divisori possono essere ridimensionati assieme all’intera finestra, in base alla colonna su cui ci si vuole concentrare.

Sia per la colonna sorgente/destinazione che per il Tab “Details” è inoltre disponibile una comoda funzione di “Compare”. La finestra che si apre permette pan (con trascinamento del mouse) e zoom (con rotellina) delle due immagini contemporaneamente,

mantenendo il focus sullo stesso punto e consentendo una analisi più immediata degli effetti della elaborazione di programma.

EditBlock

La classe EditBlock.java si occupa di fornire i metodi e l'interfaccia grafica tramite la quale l'utente finale può selezionare il blocco (funzione) da aggiungere alla pipe principale. Tali blocchi sono suddivisi in due macrocategorie:

- Libraries, in cui si trovano le principali librerie OpenCV.
- Algorithms, nel quale è possibile selezionare alcuni algoritmi, facenti uso di diverse funzioni OpenCV. Tali algoritmi non sono aggiungibili tuttavia alla pipe principale in quanto forniscono un risultato difficilmente integrabile o già di per se completo.

Nell'ipotesi di aver selezionato un elemento appartenente alla macrocategoria Libraries, verrà popolata la lista sottostante con le funzioni appartenenti alla libreria selezionata. Una volta cliccato sulla funzione desiderata verrà istanziato un oggetto specializzato della classe LearnBlock.java, i cui relativi controlli verranno rappresentati nella parte centrale dell'interfaccia. Nella parte sovrastante i controlli verrà rappresentata la descrizione della funzione selezionata mentre nella parte inferiore il codice mediante il quale implementare la funzione stessa,
Una volta settati i parametri secondo proprio interesse (parametri impostati inizialmente con settaggi di default) è possibile visualizzare il risultato andando a cliccare il bottone "elaborate". Un'impostazione errata o non consistente (che non rispetta cioè le regole implementate dal suo programmatore) impedirà l'aggiunta del blocco alla pipe principale. La descrizione dell'errore verrà automaticamente visualizzata nella parte centrale della finestra.
Nel qualcaso i risultati ottenuti siano di gradimento per l'utente, questi potrà aggiungere il blocco alla pipe principale tramite pressione del bottone add e salvarne il risultato tramite pressione del bottone save.

Implementare estensioni:

Il presente progetto è stato pensato per consentire lo sviluppo di nuove funzioni, in forma di blocco, da parte di utenti interessati ad estendere le funzionalità. I blocchi già presenti coprono gran parte delle funzioni offerte della libreria opencv.imgproc (e alcune funzioni essenziali da opencv.core) e la loro implementazione vuole essere d'esempio ai futuri programmatori di questo tipo. Verranno di seguito illustrate le linee guida da seguire per un corretto sviluppo del codice.

Implementazione logica

Definito il tipo di blocchetto da implementare, creare una classe che abbia un nome rappresentativo dello scopo della stessa; essa dovrà necessariamente estendere la classe *LearnBlock* (con il costrutto *extends*). Tale classe deve essere inserita nel package *learnOpenCVBlocks*.

Individuare il numero e il tipo di paramentri che si ritiene necessario impostare da interfaccia grafica, selezionando il tipo di variabili JavaFX in base allo scopo, tra quelle messe a disposizione:

- Boolean selection: CheckBox, RadioButton, ToggleButton
- List selection: ComboBox, ChoiceBox, ListView
- Scrollable Value: ScrollBar, Slider
- Manual Input Value: TextField, TextArea

Ogni variabile sarà dunque collegata al file di interfaccia FXML tramite fx:id. N.B.: se si intende mostrare all'utente finale anche il codice del blocco, la scelta non dipenderà solo dallo stile grafico della interfaccia: il codice in forma testuale è infatti fortemente legato al tipo di variabile fxml utilizzato (approfondito nel paragrafo "Implementazione descrizione e codice").

Inizializzare tali primitive JavaFX (insieme ad eventuali altre variabili necessarie al blocco) nel costruttore della classe. A tal proposito si consiglia di impostare i controlli con dei valori di default per rendere più immediato l'uso del programma all'utente.

Implementare i due metodi virtuali:

- ***validate()***: è chiamata SEMPRE prima di *elaborate()* e riceve come input una *opencv.core.Mat* (su cui dovrà essere fatta l'elaborazione) e restituisce un valore booleano a seconda dell'esito.
  Ha il compito di effettuare tutti i controlli necessari sulle variabili usate da *elaborate()*, ovvero sui parametri impostati dall'utente e sulla *Mat* di input; nell'eventualità abbiano valori non previsti impostare la variabile *error* (ereditata da LearnBlock) con una descrizione dell'errore e impostare il valore di ritorno della funzione a *false*. Ritornare *true* altrimenti;
  Si consiglia di eseguire in sequenza per primi i controlli INDIPENDENTI dalla matrice di input e per ultimi i controlli DIPENDENTI: anche in caso di errori dovuti a Mat nulla, *error* conterrà almeno gli errori indipendenti dall'input (se presenti, fornendo più informazioni all'utente). La buona programmazione prevede che tale caso sia gestito saltando i controlli per evitare che *validate()* termini prematuramente. Ad ogni modo il metodo *check()* di LearnBlock si occuperà di catturare *NullPointerException* e, per essere sicuri, ritornerà sempre false quando *inputMat* del blocco è null (ignorando il risultato di *validate()*) cosicché *elaborate()* non sia mai eseguita; sarà inoltre concatenata ad *error* una stringa informativa standard. Si noti che se *validate()* ritorna true, *elaborate()* potrebbe comunque fallire a causa di eccezioni non gestite lanciate da casi non previsti! In tal caso si consiglia di rivedere l'implementazione del blocco e se necessario scrivere una nuova versione di codice per quel caso.
- ***elaborate()***: viene eseguita SOLO SE *validate()* ritorna *true*, riceve come input e restituisce come output una *opencv.core.Mat*. Ha il compito di:
  - se sono previste più versioni dell'algoritmo, la cui esecuzione debba dipendere dagli input forniti dall'utente (ad es. dal loro numero o valore), è

possibile settare un valore numerico nella variabile *codeVersion* (già predisposta da LearnBlock) per discriminarle; di default il valore è 0;
    - eseguire (per ogni versione) le necessarie elaborazioni sulla Mat di input e impostare come valore di ritorno della funzione il risultato finale;

E' preferibile implementare qui cattura/lancio di eccezioni solo in fase di test (esse saranno infatti catturate e mostrate a video nell'interfaccia). In fase di release è buona norma implementare TUTTI i controlli necessari in *validate()* e lasciare che le funzioni interne alla elaborate lancino le eccezioni che non possono essere gestite come controlli. In questo modo l'utente finale avrà modo di constatare in maniera trasparente i limiti delle funzioni implementate nel blocco visibili attraverso il codice renderizzato equivalente.

Implementazione grafica

Per il design dell'interfaccia del blocco (e generare un file FXML valido) è consigliato l'utilizzo del software "Scene Builder" fornito da JavaFX. Come *root container* è consentito utilizzare un qualsiasi oggetto di tipo *Parent*; si consiglia di utilizzare un oggetto di tipo AnchorPane per la sua semplicità di utilizzo (permette infatti di posizionare i controlli in maniera immediata) sebbene sia di dimensioni assolute (il massimo previsto è di 450x230px, oltre la quale possono insorgere dei problemi di visualizzazione). Oggetti di dimensioni relative (es. GridPane) sono comunque consentiti, a patto di tenere sotto controllo le dimensioni ed il suo contenuto (sempre ai fini di una corretta visualizzazione).

All'interno del root Parent possono essere inseriti dei Container (es. HBox, sempre di classe Parent), con **id="control"** e ognuno di essi potrà avere al suo interno qualsiasi collezione di nodi, tra cui anche i Controls (es. CheckBox, TextField,..) ritenuti necessari per impostare i parametri del blocco. I Controls possono essere usati liberamente come variabili di programma (tramite fx:id), ma **SOLO UNO** per Container potrà comparire nei file .txt di codice del blocco (sotto forma di tag). La motivazione alla base di questa restrizione consiste nel voler raggruppare in un contenitore tutto ciò che riguarda UN SOLO controllo, per poi poterli rapprensentare in maniera ordinata all'utente nella schermata principale. All'atto dell'aggiunta del blocco alla pipe, infatti, i singoli Container (e quindi l'esatto contenuto) sono estratti dal Parent e posizionati in una lista nel tab Control.

La scelta del tipo di controllo da usare deve ricadere non solo sulla sua rappresentazione grafica, ma anche su come sarà interpretato dal parser che ne stilerà il codice (se si prevede che il blocca debba fornirne uno).

Implementazione di descrizione e codice

La descrizione della funzione o algoritmo implementato nel blocco dovrà essere scritta in un file .txt, senza particolari restrizioni, se non il fatto che il file di testo deve avere lo stesso nome della classe a cui si riferisce. Tale file dovrà essere inserito nella cartella "desc" del package *learnOpenCVBlocks,* il programma stesso si occuperà al momento dell'esecuzione a mostrarla a video.
Il codice relativo all'implemetazione del blocco dovrà anch'esso essere scritto su di un file di testo ma questa volta contenuto nella cartella "cod". A differenza della descrizione però sono presenti alcune regole da seguire per la scrittura del codice:

- Il nome del file di testo deve essere quello della classe a cui fa riferimento più il valore numerico della versione *codeVersion* cui il codice si riferisce (es: Canny0.txt, Canny1.txt ecc).
- A seconda del tipo di controllo utilizzato nei Container dell'interfaccia grafica, il parser interpreterà in maniera differente l'input inserito dall'utente. Di seguito sono elencati i diversi tipi di controllo con relativa interpretazione del parser:
    - Selezione:
        - *CheckBox*: scrive "true" se selezionato, "false" altrimenti;
        - *RadioButton*: scrive "true" se selezionato, "false" altrimenti;
        - *ToggleButton*: scrive "true" se selezionato, "false" altrimenti;
    - Selezione da lista
        - *ComboBox*: scrive la stringa del valore selezionato;
        - *ChoiceBox*: scrive l'indice dell'elemento selezionato (*int*);
        - *ListView*: scrive l'indice dell'elemento selezionato (*int*);
    - Valore regolabile
        - *ScrollBar*: scrive il valore individuato (*double*);
        - *Slider*: scrive il valore individuato (*double*);
    - Input manuale
        - *TextField*: scrive il suo contenuto;
        - *TextArea*: scrive il suo contenuto.
- Il nome del controllo (id/fx-id) di cui si vuole leggere il contenuto andrà inserito tra i simboli "<!" e ">" (eventuali spazi tra il contenuto del tag e i simboli saranno ignorati). Ogni controllo dovrà comparire complessivamente **ALMENO UNA VOLTA** nei diversi file .txt che compongono il codice della funzione.
- Dovranno comparire **SEMPRE** almeno un tag <! INPUT> e un tag <! OUTPUT> relativi, come suggerito dal nome stesso, all'input e all'output del blocco.

L'errata o mancata implementazione dei file relativi alla descrizione o al codice non pregiudica il funzionamento del blocco. Il colore giallo del led di stato del blocco, nella pipe principale, segnalerà questa situazione. Diversi debug output in *system.out* sono state lasciati in *loadCode()* per guidare il programmatore ad individuare problemi nella scrittura del codice simbolico.

Integrazione nel programma principale

L'ultimo step consiste nell'integrare il blocco sviluppato nel programma principale, per far ciò è necessaria la modifica della classe EditBlock.java.

Se il blocco appartiene ad una libreria di OpenCV già definita nel programma è necessario:

- Modificare la funzione *setLists()* aggiungendo alla lista corrispondente il nome della funzione implementata;
- Aggiungere nella funzione "*blocks**nomeLibreria**()*" un ulteriore case sequenziale, nel costrutto switch, al fine di richiamare la classe definita.

Nel caso si tratti di una funzione appartenente ad una libreria non ancora inserita bisogna:

- Creare una variabile globale privata di tipo ObservableList<string> *"nomeLibreria"*;
- Editare tale lista affinchè contenga il nome della nuova funzione;
- Aggiungere alla lista *libViewed*, presente nella funzione *setLists()*, il nome della nuova libreria implementata;
- Creare funzione "*blocks**nomeLibreria**()*" nella quale inserire costrutto switch-case nel cui interno si istanzi il blocco inserito;
- Richiamare all'interno della stessa funzione il metodo *setGUI()*;
- Modificare il listener relativo a *function*, presente nella funzione *setListner()*, aggiungendo un case sequenziale nello switch in esso contenuto, affiché richiami la funzione precedentemente creata.

Se il blocco sviluppato invece contiene un algoritmo i passi da seguire sono i seguenti:

- Modificare la funzione *setLists()* aggiungendo alla lista *algViewed* il nome dell'algoritmo implementato;
- Aggiungere nella funzione *blocksAlgorithm()* un ulteriore case sequenziale nel costrutto switch, al fine di istanziare la classe progettata.