Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Ingegneria Informatica

Porting di un algoritmo per la stima del flusso ottico su smartphone Android

Relatore:
Prof.
STEFANO MATTOCCIA

Candidato:
GUGLIELMO PALAFERRI

Appello II
Anno Accademico 2020-2021

Introduzione

Il monitoraggio costante della velocità di fiumi e correnti d'acqua può assumere notevole importanza sia nello studio di fenomeni idrologici puramente naturali, sia nella progettazione di opere ingegneristiche strettamente legate ad un particolare flusso d'acqua. Ad esempio, può aiutare ad analizzare e rilevare fenomeni come le inondazioni (specie gli avvenimenti improvvisi, che destano particolare attenzione), così come anche il trasporto di sedimenti o l'erosione delle rocce.

Molte delle tecniche tradizionali utilizzate per l'osservazione di un flusso idrico, tuttavia, non garantiscono grande efficienza e presentano costi elevati: spesso è richiesta la presenza di personale specializzato per la manutenzione di dispositivi complessi.[2] Una soluzione che preveda invece l'installazione di apparecchi ottici, e basi quindi il monitoraggio sull'elaborazione di immagini, può consentire di abbattere notevolmente i costi e di distribuire il sistema di osservazione ottenendo quindi maggiore resistenza ai guasti.

immagine esempio applicazione OTV

È proprio questo un caso di utilizzo di **OTV** (*Optical Tracking Velocimetry*), una tecnica che fa uso di particolari algoritmi di computer vision (in particolare l'algoritmo di Lucas-Kanade, utilizzato per la stima del flusso ottico) per tracciare le traiettorie e le velocità del flusso d'acqua a partire da una serie di immagini. Il tracciamento viene svolto grazie al riconoscimento di particelle quali detriti e altri residui e al confronto di fotogrammi consecutivi.

Il metodo OTV è pensato per essere applicato a dispositivi di elaborazione a basso costo e di dimensioni contenute: questi sarebbero posizionati lungo corsi d'acqua in aree geografiche remote. I dati poi raccolti da questi dispositivi potranno essere

spediti (tramite meccanismi semplici come l'invio di SMS) ad un sistema di raccolta dati centralizzato. Va da sé dunque che l'ottimizzazione dei consumi energetici dei dispositivi costituisca un punto cruciale per la realizzabilità di un tale sistema di monitoraggio. Questo tema verrà preso in considerazione e rappresenta uno degli argomenti principali dell'elaborato.

L'algoritmo è stato inizialmente testato su dispositivi della famiglia *Raspberry*, per via delle loro dimensioni molto contenute e in generale per le funzionalità da essi offerte, molto coerenti con i requisiti del progetto. Le analisi hanno riportato ottimi risultati dal punto di vista dei consumi energetici in particolare dei modelli Raspberry Pi 3B e 4.[1]

Altri dispositivi con buone potenzialità e con un profilo che si presti bene al contesto di utilizzo sono gli **smartphone**, con particolare riferimento a quelli basati su sistema operativo **Android**. L'utilizzo di tali dispositivi richiede ovviamente una seppur minima quantità di modifiche rispetto al deployment effettuato su Raspberry, ed è proprio questo il tema centrale del seguente documento.
Nei prossimi capitoli si procede quindi — dopo aver introdotto qualche informazione necessaria su OTV — a descrivere la realizzazione di un'applicazione per smartphone Android che adatti l'implementazione in C++ di OTV (disponibile su GitHub) e i risultati in termini di prestazioni e consumi energetici che ne sono conseguiti.

Indice

Introduzione **i**

1 OTV **1**

1.1 Contesto di utilizzo 1

1.2 Ciclo di funzionamento 1

1.3 Ottimizzazioni 3

1.3.1 Ottimizzazioni hardware 3

2 Processo di sviluppo **7**

2.1 Il sistema Android 7

Capitolo 1

OTV

1.1 Contesto di utilizzo

Come già brevemente descritto, OTV prevede un deployment su dispositivi di dimensioni ridotte e autosufficienti dal punto di vista energetico. In particolare, la configurazione testata su Raspberry introduceva i seguenti componenti:

- Raspberry Pi 3B/4 per l'elaborazione
- Panello solare 6 W (PiJuice Solar Panel) per sostenere i consumi energetici
- Batteria esterna (PiJuice Hat) per fornire alimentazione

[1] Una simile configurazione verrebbe usata con smartphone, salvo ovviamente l'utilizzo di una batteria aggiuntiva.

Immagine applicazione OTV 2 (quella con lo sketch del fiume usata nel paper nuovo)

1.2 Ciclo di funzionamento

Il dispositivo così composto, una volta accuratamente posizionato ed avviato, dovrebbe eseguire *quattro* misurazioni della velocità dell'acqua ogni ora, risultando quindi a regime in un ciclo di funzionamento periodico della durata di 15 minuti.

Sebbene la misurazione mediante l'algoritmo OTV sia svolta sul momento, non viene effettuata sulle immagini direttamente ricevute e lette in input dalla telecamera: il video acquisito necessita di una fase preliminare che prepari le immagini per essere elaborate. Questo viene fatto, tra le altre cose, per consentire di scegliere un settaggio particolare (ad esempio, selezionare una risoluzione diversa rispetto al video originale), utile successivamente al fine di ottimizzare l'elaborazione.

Il ciclo di funzionamento si articola quindi in questo modo:

1. Fase di **acquisizione**: le immagini vengono acquisite dalla telecamera. Questa fase ha una durata fissa e dipende dalla lunghezza del video che si vuole analizzare: tipicamente 20 secondi.
2. Fase di **estrazione** dei frame: a partire dal video acquisito, si estraggono i fotogrammi che lo compongono a seconda della configurazione scelta, in particolare è possibile specificare la risoluzione desiderata tra:
   - Full Resolution (**F**): Risoluzione originale
   - Half Resolution (**H**): Risoluzione dimezzata
   - Quarter Resolution (**Q**): Risoluzione 1/4 dell'originale
3. Fase di **elaborazione** (OTV): a questo punto le immagini estratte vengono effettivamente elaborate utilizzando OTV. Questa fase è cruciale dal punto di vista dei consumi in quanto è quella che può variare maggiormente a seconda della configurazione usata e delle ottimizzazioni implementate. È bene quindi analizzarla di conseguenza.
4. Fase di **idle**: una volta conclusa l'elaborazione (ed eventualmente spediti i dati rilevati) segue un periodo di stand-by, in cui si attende il tempo necessario prima della prossima rilevazione. Anche questa fase è molto importante per determinare i consumi energetici del processo: se il dispositivo dovesse disporre di una modalità di risparmio energetico, l'energia utilizzata potrebbe diminuire drasticamente.

Le fasi su cui è possibile effettivamente lavorare per ottenere risultati migliori sono quelle di elaborazione (in modo particolare) e di idle.

Prima di introdurre i vari livelli di ottimizzazione, va intanto fatto notare che la specifica implementazione di OTV presa in caso è basata sulla libreria open-source di computer vision **OpenCV**.
OpenCV fornisce un framework per la creazione di applicazioni legate alla computer vision e implementa una vasta gamma di algoritmi, tra cui l'algoritmo di Lucas-Kanade utilizzato da OTV e precedentemente descritto.
L'utilizzo di OpenCV prescrive una serie di passaggi di installazione che variano in base all'ambiente di sviluppo e che — nel caso specifico di Android — verranno analizzati nel successivo capitolo.

1.3 Ottimizzazioni

Le ottimizzazioni applicabili ad OTV analizzate in [3] consistono in una serie di tecniche e meccanismi che possano contribuire ad aumentare l'efficienza energetica del sistema. Tra queste, possiamo distinguere quelle legate al **software** e quelle invece a livello **hardware** (ad esempio, l'utilizzo di istruzioni particolari).

Le ottimizzazioni software consistono essenzialmente nella configurazione del dispositivo in modo ad esempio da disattivare le opzioni software che risultino superflue (Wi-Fi, Bluetooth ecc.). Si parla di ottimizzazioni derivanti dal sistema operativo utilizzato e dunque dipendenti dal dispositivo in questione. Si vedranno ottimizzazioni di questo tipo esclusive al sistema Android.

1.3.1 Ottimizzazioni hardware

Nel caso delle ottimizzazioni hardware, si parla di particolari metodi che introducono differenti modelli di esecuzione a livello di processore, facendo leva specialmente sulla parallelizzazione delle istruzioni. Questo, oltre agli ovvi vantaggi in termini di performance, può portare ad una maggiore efficienza in termini di consumi. Si delineano tre possibilità principali, eventualmente sovrapponibili, focalizzate su aspetti e modalità diverse di parallelizzazione:

- Esecuzione multi-core mediante **OpenMP** o **TBB**.
- Esecuzione di istruzioni **SIMD** tramite **NEON**

- Esecuzione su **GPU** mediante la libreria **OpenCL**

OpenMP e TBB

OpenMP e TBB sono due librerie open-source che in fase di sperimentazione sono state utilizzate per testare il parallelismo *thread-level* in modo da sfruttare i multipli core disponibili nelle moderne CPU. Le due librerie — poiché forniscono lo stesso tipo di funzionalità — sono da utilizzare in modo mutuamente esclusivo. La scelta della libreria da utilizzare cadrà dunque su quella che garantisca le migliori prestazioni.

SIMD

SIMD (*Single Instruction Multiple Data*) è una classe di istruzioni che consente di ottenere parallelismo su un singolo core. Il modello prevede l'esecuzione della stessa operazione su una molteplicità di dati mediante l'utilizzo di una singola istruzione. All'interno di un'istruzione vengono quindi inglobati diversi dati e, ovviamente, l'operazione da svolgersi.

I processori ARM, montati sulla stragrande maggioranza di dispositivi Android così come anche sui modelli di Raspberry Pi testati, dispongono di un'architettura SIMD avanzata chiamata NEON.
Poiché la libreria OpenCV fornisce nativamente supporto per le istruzioni NEON, risulta piuttosto immediato testare l'utilizzo di tali istruzioni nell'applicazione OTV.

L'utilizzo delle istruzioni SIMD può beneficiare specialmente i casi in cui una singola operazione debba essere ripetuta molte volte su un insieme di dati anche grande. È il caso dell'elaborazione di immagini, in cui spesso è richiesto operare su dati sotto forma di matrici ed eseguire operazioni anche semplici ma molto ripetitive.

GPU e OpenCL

L'utilizzo della potenza di calcolo di una GPU è giustamente considerato tra le ottimizzazioni da testare: qui il parallelismo è intrinseco al tipo di processore, che è progettato appositamente per svolgere operazioni semplici ma ripetitive su un grande insieme di dati, in particolare nei casi di elaborazione di immagini. Tuttavia,

come evidenziato in [3] il potenziale guadagno in efficienza che questa soluzione garantirebbe non è facilmente stimabile e dipende da una varietà di fattori.

La libreria OpenCL permette di sfruttare le funzionalità della GPU, qualora supportata dal dispositivo in questione.

Capitolo 2

Processo di sviluppo

Parlando ora dell'effettiva operazione di porting di OTV su dispositivo Android, si espongono gli strumenti utilizzati e le complessità riscontrate durante il processo di sviluppo. Verranno trattati inoltre i dettagli tecnici dell'applicazione e i meccanismi utilizzati propri del sistema Android.

2.1 Il sistema Android

Bibliografia

[1] A.-H. Livoroi, A. Conti, L. Foianesi, F. Tosi, F. Aleotti, M. Poggi, F. Tauro, E. Toth, S. Grimaldi, and S. Mattoccia. On the deployment of out-of-the-box embedded devices for self-powered river surface flow velocity monitoring at the edge. *Applied Sciences*, 11(15), 2021.

[2] F. Tauro, F. Tosi, S. Mattoccia, E. Toth, R. Piscopia, and S. Grimaldi. Optical tracking velocimetry (OTV): Leveraging optical flow and trajectory-based filtering for surface streamflow observations. *Remote Sensing*, 10(12), 2018.

[3] F. Tosi, M. Rocca, F. Aleotti, M. Poggi, S. Mattoccia, F. Tauro, E. Toth, and S. Grimaldi. Enabling image-based streamflow monitoring at the edge. *Remote Sensing*, 12(12), 2020.