UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Elettronica

Sviluppo di algoritmi per l'identificazione e tracking di ostacoli mediante camera ad eventi

Development of algorithms to identify and track obstacles through event cameras

RELATORE:
PROF. ADRIANO MANCINI

LAUREANDO:
MATTIA DIGNANI

ANNO ACCADEMICO 2019/2020

INDICE

SOMMARIO

La stesura della tesi è basata sulla spiegazione e comprensione delle potenzialità che le camere ad eventi possono mostrare, grazie all'utilizzo di strumenti come il simulatore ESIM e un algoritmo che permette il tracking di ostacoli attraverso le camere ad eventi.
La tesi mostra i passaggi necessari per effettuare una simulazione con ESIM e lo sviluppo dell'algoritmo finale, obiettivo principale del progetto:

- Il capitolo 1 verterà sull'introduzione dell'oggetto di lavoro, ovvero le camere ad eventi: verrà spiegato il loro funzionamento, le differenze con le camere tradizionali, i relativi vantaggi e svantaggi e le applicazioni che potrebbero avere in futuro.
  Inoltre viene introdotto brevemente YOLOv3, una rete neurale che permette la rilevazione degli oggetti in movimento;
- Nel capitolo 2 verranno mostrati gi strumenti necessari e di conseguenza utilizzati per effettuare una simulazione con il simulatore ESIM;
- Il capitolo 3 è incentrato sulla figura del simulatore ESIM, utile per la simulazione di eventi quando non si dispone di una camera ad eventi;
- Il capitolo 4 tratterà i risultati che si otterranno e la spiegazione dell'algoritmo finale;
- La conclusione porrà fine alla tesi, analizzando il lavoro svolto.

CAPITOLO 1

Introduzione

1.1 Le camere ad eventi

Una camera ad eventi è un sensore che acquisisce l'immagine attraverso la rilevazione di luminosità che impatta in ogni pixel all'interno della camera: essi lavorano in modo dipendente e asincrono, segnalando i cambiamenti di luminosità a mano a mano che si verificano e restando in silenzio in caso contrario [2][10]. Le camere ad eventi forniscono vantaggi molto importanti e significativi rispetto alle camere tradizionali, che utilizzano un otturatore ad una frequenza di campionamento fissa (clock) per acquisire l'immagine.
Si può quindi definire un evento come un cambiamento asincrono della luminosità di un pixel che può essere positivo o negativo in relazione ad un aumento o diminuzione della luce che impatta su di esso. L'insieme degli eventi genera un flusso, rappresentate l'output della camera ad eventi, che registra le coordinate del pixel, l'istante di tempo in cui avviene la variazione di luminosità e il tipo di variazione.

1.1.2 Funzionamento

Come accennato prima, le camere ad eventi sono sensori innovativi che permettono di avere vantaggi importantissimi parlando di qualità e riduzione di disturbi rispetto alle camere tradizionali.
Ne è un esempio il Dynamic Vision Sensor (figura 1), un sensore per fotocamera di iniLabs, che inventa e produce tecnologia neuromorfica per la ricerca.

Figura 1: Camera DVS. Immagine tratta da [1]

A differenza delle camere tradizionali che utilizzano il principio del *frame*, le camere ad eventi rispondono al cambiamento di luminosità nella scena in modo asincrono e indipendente per ogni pixel. L'output di una camera ad eventi è un flusso di eventi. Ogni evento fa riferimento ad un pixel che possiede una certa magnitudo in un certo istante di tempo.

Questa codifica è ispirata dalla natura dei percorsi visivi biologici, o meglio: il funzionamento della camera ad eventi è stato ideato osservando e studiando l'acquisizione delle immagini della retina umana. Ogni pixel memorizza l'intensità della luce ogni volta che la camera invia un evento e monitora continuamente la variazione di magnitudo sufficiente da questo valore memorizzato. Quando la variazione va oltre un certo valore di soglia, la telecamera invia un evento, che viene trasmesso dal chip con la posizione (x, y), il tempo t e la polarità p della modifica, rappresentata con 1 bit: se la variazione di luminosità è positiva avremo "ON", altrimenti "OFF" (figura 2).

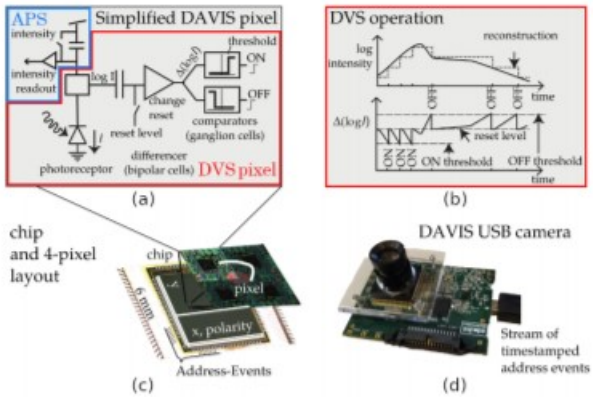


Figura 2: Riassunto di una camera DAVIS. Essa comprende un dynamic vision sensor (DVS) e un active pixel sensor (APS) nello stesso array di pixel, condividendo lo stesso fotodiodo in ogni pixel. (a) Diagramma del circuito semplificato del pixel DAVIS. (Il pixel DVS è in rosso, il pixel APS è in blu). (b) Schematizzazione delle operazioni del pixel DVS nel convertire la luce in eventi. (c)-(d) Chip DAVIS e camera USB. (e) Un quadrato bianco su un disco rotante nero visto dal sensore DAVIS che produce frames in scala di grigi e una spirale di eventi nello spazio e nel tempo. Gli eventi nello spazio e nel tempo sono codificati con colori: dal verde (passato) al rosso (presente). (f) Frame ed eventi sovrapposti in una scena.

Il flusso di eventi (figura 3), viene poi trasmesso dall'array di pixel verso l'esterno attraverso un bus di output condiviso che utilizza un protocollo di trasferimento dei dati chiamato AER (*Address-Event Rappresentation*).

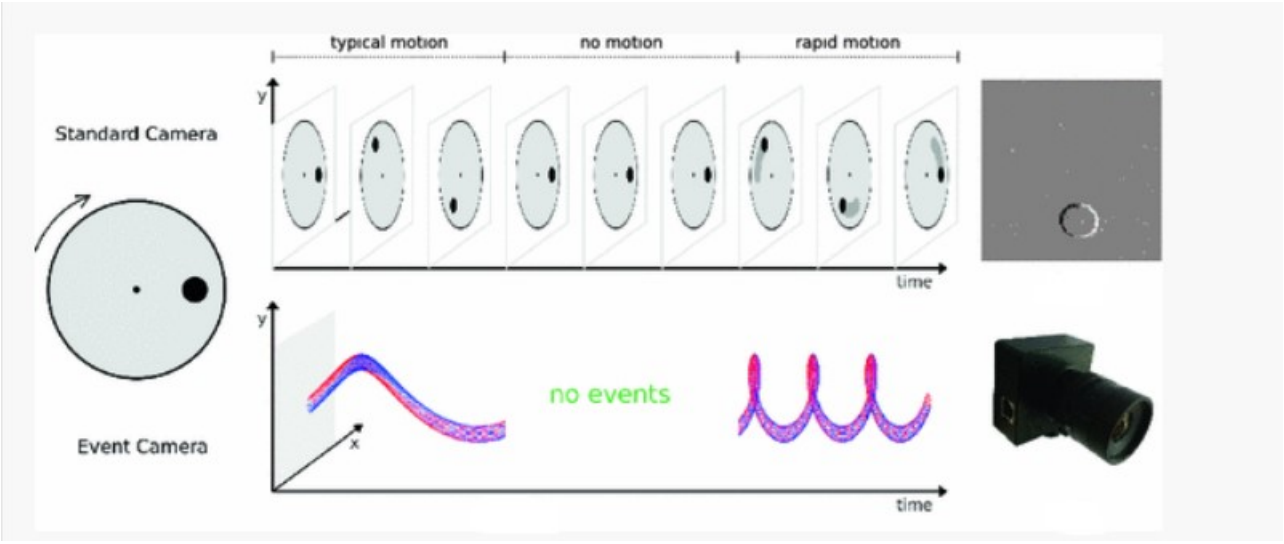


Figura 3: metodo di acquisizione delle immagini di una camera tradizionale e ad eventi. Immagine tratta da [2]

A causa delle numerose variazioni di luminosità che possono accadere durante l'acquisizione di una scena reale, si possono avere delle trasmissioni di dati ad

una frequenza che va dai 2 MHz fino ai 1200MHz che potrebbero saturare il bus. Questa larghezza di banda dipende dal chip e dal tipo di hardware utilizzato. Ogni pixel ha al suo interno un modulatore della frequenza di campionamento dell'intensità della luce. Questo è necessario perché una camera ad eventi è un sensore guidato dai dati: il loro output dipende dalla variazione di luminosità, quindi maggiore è la variazione di luminosità della scena, maggiore sarà la frequenza di campionamento del modulatore presente all'interno di ogni pixel aumentandone la frequenza di invio di dati attraverso il protocollo AER, in modo tale da catturare l'immagine nel modo corretto. Tutti gli eventi sono campionati con una frequenza dell'ordine dei microsecondi e vengono trasmessi con una latenza molto bassa (dell'ordine di microsecondi). Questi sono alcuni dei fattori che permettono alle camere ad eventi di avere bassa latenza e assenza di motion blur. La quantità di luce che irradia un singolo pixel in una scena è composta da due parti: l'illuminazione della scena stessa e la riflettanza della luce, come illustrato in figura 4

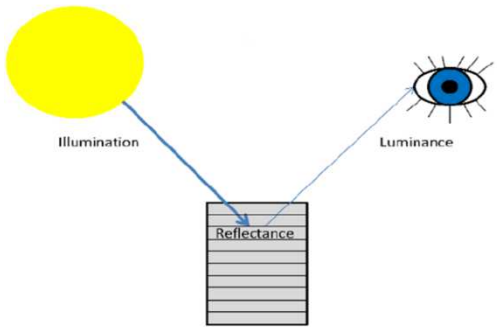


Figura 4: Riflettanza. Immagine tratta da [5]

1.1.3 VANTAGGI E SVANTAGGI

I vantaggi che queste camere possiedono sono importanti, poiché hanno prestazioni migliori rispetto alle camere tradizionali:

- *Risoluzione temporale molto alta (dell'ordine dei microsecondi)*: il monitoraggio dei cambiamenti della luminosità è veloce nella circuiteria analogica, mentre la lettura degli eventi è digitale seguendo un clock di 1 MHz (ad esempio, gli eventi vengono campionati e rilevati con una risoluzione dell'ordine dei microsecondi). Quindi le camere ad eventi riescono ad evitare il *motion blur*, tipico disturbo delle camere tradizionali quando si acquisiscono scene con movimenti molto veloci.
- *Bassa latenza*: nelle camere ad eventi, ogni pixel lavora in maniera indipendente e quindi non c'è bisogno di aspettare il tempo di esposizione dell'intero frame come accade con le camere tradizionali: appena la variazione della luminosità supera la soglia, essa viene rilevata e l'evento viene trasmesso in un tempo molto ristretto. Infatti la latenza è molto bassa (intorno ai 10 μs);
- *Range dinamico molto alto*: detto anche HDR (High Dynamic Range), con il range dinamico di 120dB delle camere ad eventi contro i 60dB delle camere tradizionali è possibile l'acquisizione e il riconoscimento di oggetti anche in condizioni di luce molto sfavorevoli sia di giorno che di notte. Questo è possibile grazie al fatto che i fotoricettori lavorano in scala logaritmica e che ogni pixel lavora in maniera indipendente.
- *Basso consumo di energia*: le variazioni di luminosità generano la trasmissione di dati e quindi l'energia viene impiegata solo per processare il cambiamento di luminosità del pixel, riducendo così il consumo di energia (intorno ai 10 μW).

Le camere ad eventi hanno caratteristiche innovative applicabili alla ricerca nella progettazione e realizzazione di nuovi algoritmi che ne accentuino le loro

straordinarie possibilità di utilizzo. I problemi ancora irrisolti sono il loro prezzo sul mercato, ancora troppo elevato ed una risoluzione molto bassa (come il sensore DAVIS346 che costa circa 5000€ [4] ed ha una risoluzione di 346X260). Inoltre, le attuali camere ad eventi hanno notevoli limitazioni, ad esempio un rapporto segnale/rumore molto basso, necessitano di complesse configurazioni e di conseguenza richiedono l'intervento di esperti per il loro corretto utilizzo. Questi problemi rendono difficile la comprensione del vero potenziale delle camere ad eventi per l'impiego in vari campi, come ad esempio la motion analysis (umana e animale).

1.1.4 LA MATEMATICA ALLA BASE DELLE CAMERE AD EVENTI

La generazione di eventi è basata su formule matematiche e variabili che dipendono dalla camera ad eventi [2]:

- La *luminosità* è il logaritmo della fotocorrente generata da un fotoricettore quando la luce impatta sul singolo pixel indipendente della camera, descritta dalla formula *L = log(I).* In assenza di disturbi, un evento $e_k=(x_k,t_k,p_k)$ si verifica al pixel $x_k=(x_k,y_k)^T$ in un tempo t_k quando l'incremento di luminosità dell'evento del pixel, definito come

$$\Delta L(x_k, t_k) = L(x_k, t_k) - L(x_k, t_k - \Delta t_k) \tag{1.1}$$

  raggiunge il *contrasto di soglia* ±*C*,

$$\Delta L(x_k, t_k) = p_k C \tag{1.2}$$

  dove $C > 0$, Δt_k è il tempo trascorso dall'ultimo evento dello stesso pixel e la polarità $p_k \in \{+1, -1\}$ è il segno del cambiamento della luminosità: +1 se la luminosità aumenta, -1 se decrementa.

- La variabile C, chiamata *contrasto*, è determinata dalle correnti di bias del pixel e sono prodotte da un generatore di bias programmato digitalmente

nel chip. Il contrasto C può essere stimato tramite la conoscenza di queste correnti di bias. Nelle tipiche camere DVS, le soglie vengono impostate tra il 10% e il 50% della variazione dell'illuminazione. Ci sono anche delle applicazioni delle camere DVS con un alto gain dei fotoricettori che operano a soglie molto basse (anche dell'1%). Ad ogni modo, esse vengono usate in circostanze molto favorevoli con condizioni meteo ideali e con un'illuminazione molto chiara. Il limite inferiore del contrasto di soglia è calcolato in base ai disturbi e alla variabilità tra pixel e pixel. Impostare un valore di C troppo basso comporta un aumento dei disturbi degli eventi che nascono da pixel con valori bassi di C.

- In un piccolo intervallo di tempo Δt_k, un incremento della luminosità può essere approssimato utilizzando l'espansione di Taylor,

$$\Delta L(x_k, t_k) \approx \frac{\partial L}{\partial t}(x_k, t_k)\Delta t_k \tag{1.3}$$

e quindi l'equazione (1.2) può essere formulata nel modo seguente:

$$\frac{\partial L}{\partial t}(x_k, t_k) \approx p_k C \Delta t_k \tag{1.4}$$

È importante notare che gli eventi della camera DVS sono scatenati da una variazione nella magnitudo della luminosità e non dal superamento della soglia da parte della derivata nel tempo della luminosità.

Assumendo che l'illuminazione sia costante linearizzando l'equazione (1.2) e assumendo che la luminosità sia costante, la causa della generazione di un evento è data dal movimento dei bordi. Per piccoli valori Δt, l'incremento dell'intensità (equazione (1.2)) può essere approssimato nel modo seguente:

$$\Delta L \approx -\nabla L \cdot v\Delta t \tag{1.5}$$

che è causato da un gradiente della luminosità $\nabla L(x_k, t_k) = (\frac{\partial}{\partial x}L, \frac{\partial}{\partial y}L)^T$ che si muove con una velocità $v(x_k,t_k)$ sul piano dell'immagine con uno spostamento $\Delta x = v\Delta t$. Nell'equazione (1.5), il prodotto scalare ci indica che se il movimento

è parallelo al bordo nessun evento è generato dal momento in cui $v \cdot \Delta L = 0$. Se invece, il movimento è perpendicolare al bordo, $(v \,//\, \Delta L)$ gli eventi sono generati con frequenza più alta.

1.1.5 MODELLI

Il primo modello di camera ad eventi è stato sviluppato da Mahowald e Mead al Caltech negli anni che vanno dal 1986-1992 nel loro tesi di dottorato premiato con il premio Clauser [2]. Il sensore utilizzava il protocollo AER per trasportare gli eventi in uscita ed aveva pixel logaritmici con tre strati di retina Kufler. Malgrado ciò, aveva delle carenze:

1. la camera aveva dei pixel molto grandi che la rendevano poco utilizzabile nella pratica;
2. ogni superficie di retina avvolta da cavi aveva bisogno di una particolare tensione di bias che doveva essere impostata in modo molto preciso con l'uso di potenziometri;
3. ogni pixel aveva una risposta molto differente l'uno con l'altro;

Un modello molto più performante del precedente è la camera DVS. Essa è basata su una struttura a retina di silicio composta da due parti accoppiate tra loro: un fotoricettore a tempo continuo e un circuito di lettura resettato ad ogni campionamento di un pixel. Malgrado molti problemi possano essere risolti con l'analisi degli eventi generati da una camera DVS (ad esempio esaminando i cambiamenti di luminosità), alcuni necessitano un output statico (ad esempio la luminosità "assoluta"). L'ATIS (Asynchronous Time Based Image Sensor) (Figura 1.5), propone una struttura diversa: ogni pixel è composto da un subpixel DVS che a sua volta attiva un altro subpixel che legge l'intensità assoluta di luce. Il trigger resetta un condensatore attraverso un alto voltaggio. La carica di esso viene fatta diminuire attraverso un fotodiodo. Maggiore è la

luce, maggiore è la velocità con il quale il condensatore si scarica. L'intensità letta dall'ATIS fornisce due eventi codificando il tempo che passa tra i due voltaggi di soglia. In questo modo solo il pixel che cambia fornisce i nuovi valori di intensità. Maggiore è il cambiamento dell'illuminazione, più corto è il tempo tra questi due eventi. L'ATIS ha un notevole range dinamico statico (maggiore di 120dB) ma ad ogni modo presenta degli svantaggi. Il pixel ATIS ha un'area almeno doppia rispetto al pixel DVS e inoltre, nelle scene in oscurità, il tempo tra due eventi potrebbe essere molto lungo e la lettura dell'intensità degli stessi potrebbe essere interrotta da nuovi eventi che si presentano nel frattempo.

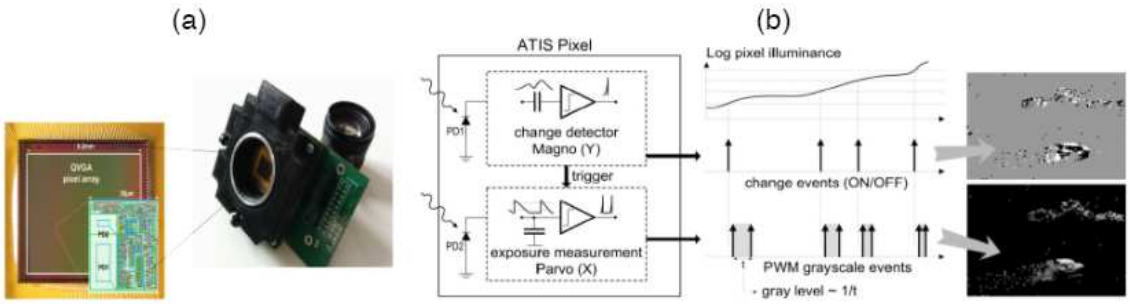


Figura 5: (a): Il sensore ATIS. Immagine tratta da [2]

Ultimo è il DAVIS (Dynamic and Active Pixel Vision Sensor) che combina un convenzionale "Active Pixel Sensor" (APS) e un pixel DVS insieme nello stesso pixel. Il vantaggio è che il circuito di lettura aumenta solo del 5% l'area della superficie del pixel DVS mantenendone le dimensioni contenute. Ad ogni modo, la frequenza di lettura dell'APS ha un limitato range dinamico (55dB) e come nelle camere standard, è ridondante se i pixel non cambiano

1.1.6 APPLICAZIONI

Gli scenari tipici in cui le telecamere per eventi offrono vantaggi rispetto ad altre modalità di rilevamento includono sistemi di interazione in tempo reale, come la robotica o l'elettronica indossabile, in cui il funzionamento in condizioni di illuminazione, latenza e alimentazione non controllate sono importanti. Le telecamere ad eventi sono utilizzate per il rilevamento di oggetti, sorveglianza e monitoraggio e riconoscimento di oggetti / gesti. Sono anche vantaggiosi per la stima della profondità, la scansione 3D a luce strutturata, la stima del flusso ottico, la ricostruzione dell'immagine HDR, e simultanea Localizzazione e mappatura (SLAM). La visione basata sugli eventi è un campo di ricerca in crescita e altre applicazioni, come la rimozione delle sfocature delle immagini o il tracciamento delle stelle, appariranno quando le telecamere degli eventi saranno ampiamente disponibili.

1.2 YOLO: RILEVAMENTO DI OGGETTI IN TEMPO REALE

YOLO (You Only Can Look Once) è un sistema di rilevamento di oggetti in tempo reale [11]. É stato sviluppato da Redmon e Farhadi nel 2015, durante il loro dottorato [12]. In questo progetto faremo riferimento alla versione YOLOv3. Il concetto di questa rete neurale è quello di formulare il rilevamento di oggetti come un singolo problema di regressione, direttamente dai pixel dell'immagine alle coordinate del riquadro di delimitazione e alle probabilità di classe. In breve, basta guardare una volta l’immagine per prevedere quali oggetti sono presenti e dove sono. YOLO è ancora in ritardo rispetto ai sistemi di rilevamento all'avanguardia in termini di precisione. Sebbene sia in grado di identificare rapidamente gli oggetti nelle immagini, fatica a localizzare con precisione alcuni oggetti, specialmente quelli piccoli.

YOLO unifica i componenti separati del rilevamento degli oggetti in un'unica rete neurale. Essa utilizza le caratteristiche dell'intera immagine per prevedere ogni bounding boxes all’suo interno contemporaneamente.

Ciò significa che la rete ragiona globalmente sull'immagine completa e su tutti gli oggetti nell'immagine.

Il design YOLO consente un end-to-end training e una velocità in tempo reale mantenendo un'elevata precisione media. Esso divide l'immagine in ingresso in una griglia S × S (figura 6). Se il centro di un oggetto cade in una cella della griglia, quella cella della griglia è responsabile del rilevamento di quell'oggetto.

Ogni cella della griglia prevede un numero *B* di bounding boxes e i punteggi di confidenza per tali riquadri. Questi punteggi riflettono quanto sia sicuro che la bounding box contenga un oggetto e anche quanto sia accurato che la box sia predetta.

Formalmente si definisce *confidenza* come $Pr\ (Object) * IOUtruthpred$. Se non esistesse alcun oggetto in quella bounding box, i punteggi di confidenza dovrebbero essere zero. Altrimenti il punteggio della confidenza sarà l'intersezione sull'unione (IOU) tra la box predetta *(prediction)* e quanto sia reale quell'oggetto (*ground truth*).

Ciascun riquadro di delimitazione è composto da cinque previsioni: *x, y, w, h* e *confidenza*. Le coordinate *(x, y)* rappresentano il centro del riquadro rispetto ai limiti della cella della griglia. La larghezza e l'altezza sono previste rispetto all'intera immagine. Infine, la *confidenza* rappresenta l'IOU tra la casella prevista e qualsiasi casella di ground truth. Ogni cella della griglia prevede anche le probabilità della classe *C*, $Pr(Class_i|Object)$.

Queste probabilità sono condizionate sulla cella della griglia contenente un oggetto. Si prevede solo un insieme di probabilità di classe per cella della griglia, indipendentemente dal numero B di boundig boxes.

Al momento del test moltiplichiamo le probabilità della classe e le previsioni di confidenza della boundig box:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

Che fornisce i punteggi di confidenza specifici per classe per ciascuna boundig box. Questi punteggi codificano sia la probabilità che quella classe appaia nella bounding box sia quanto bene la box prevista si adatti all'oggetto.

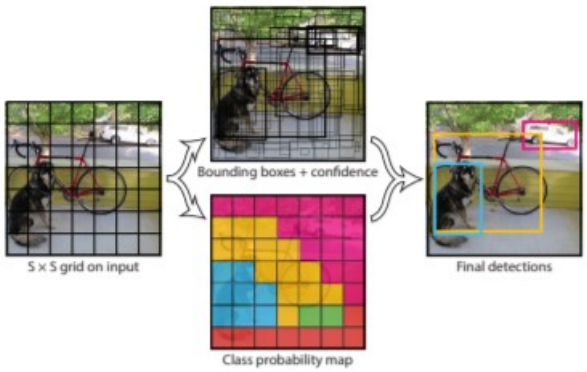


Figura 6: Modello di rilevamento degli oggetti come problema di regressione. Immagine presa da [11].

1.3 GOOGLE COLABORATORY

Google Colaboratory (noto anche come *Colab*) è un ambiente ospitato da Jupyter Notebook gratuito di Google. Consente di eseguire un notebook interamente nel cloud e di memorizzare il codice e i dati in Google Drive.
In questo progetto si è utilizzato Google Colab per simulare la rete neurale YOLOv3.
Una volta aperta la pagina di YOLOv3 per Ultralytics in GitHub, spostarsi sulla sezione “Environments” e premere “Open in Colab” in modo tale da aprire il codice di YOLO in Google Colab.

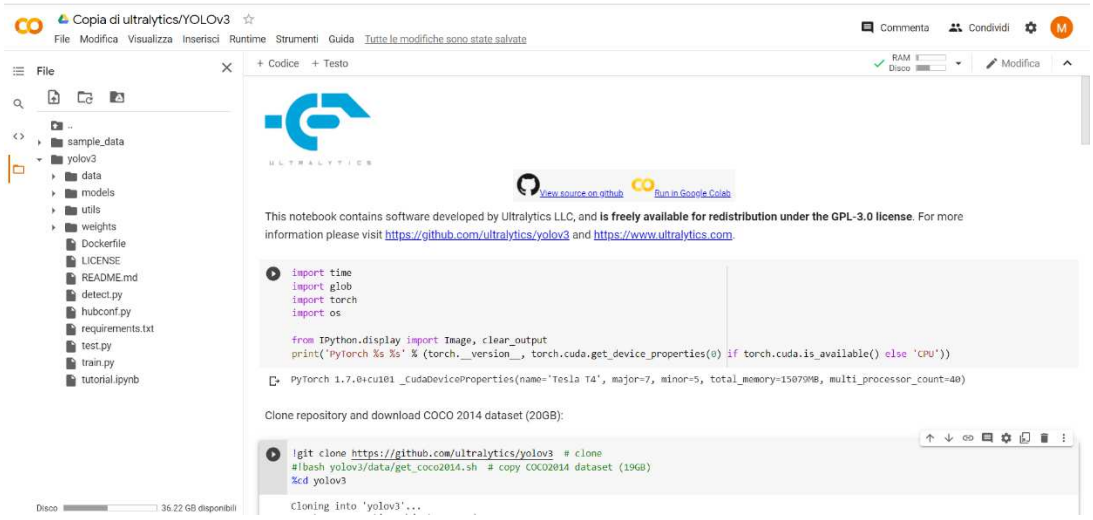


Figura 7: Screenshot di ultralytics/YOLOv3 in Google Colab.

Facendo riferimento alla figura 7, premere i due pulsanti play per clonare il repository e installare le dipendenze. Fatto questo, aprire il file Python “detect.py” nella cartella “yolov3” a sinistra su cui si effettueranno delle modifiche per ottenere una cartella dove verranno salvate i percorsi delle immagini che si stanno considerando, le classi delle bounding box al loro interno, il valore della confidenza e le dimensioni di ogni bounding boxes.

detect.py

```
# Write results
for *xyxy, conf, cls in reversed(det):
    save_path2= '/content/yolov3/runs'
    nome = 'valoriboundingbox'
    completeName = os.path.join(save_path2, nome + '.txt')
    file1 = open(completeName, 'a')

    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        with open(save_path[:save_path.rfind('.')] + '.txt', 'a') as file:
            file.write(('%g ' * 5 + '\n') % (cls, *xywh))  # label format

    if save_img or view_img:  # Add bbox to image
        xywh = str((xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist())
        xywh1 = xywh.replace("[" , "\t")
        xywh2 = xywh1.replace("]" , "\t")
        xywh3 = xywh2.replace("," , "\t")
        label = '%s %.2f' % (names[int(cls)], conf)
        plot_one_box(xyxy, im0, label=label, color=colors[int(cls)])
        file1.write(label + '\t' + save_path + '\t' + xywh + '\n')
        file1.close()

```

Figura 8: Screenshot del file detect.py modificato.

Come si evince dalla figura 8, eseguendo il file *detect.py* con le modifiche apportate, si creerà una cartella 'valoriboundingbox' nel percorso '/content/yolov2/runs'.

Se al file detect.py si passa in input un'immagine, allora si avrà una cartella 'output' in cui verrà salvata l'immagine 'detectata', ovvero l'immagine in cui gli oggetti sono rilevati attraverso le bounding boxes.

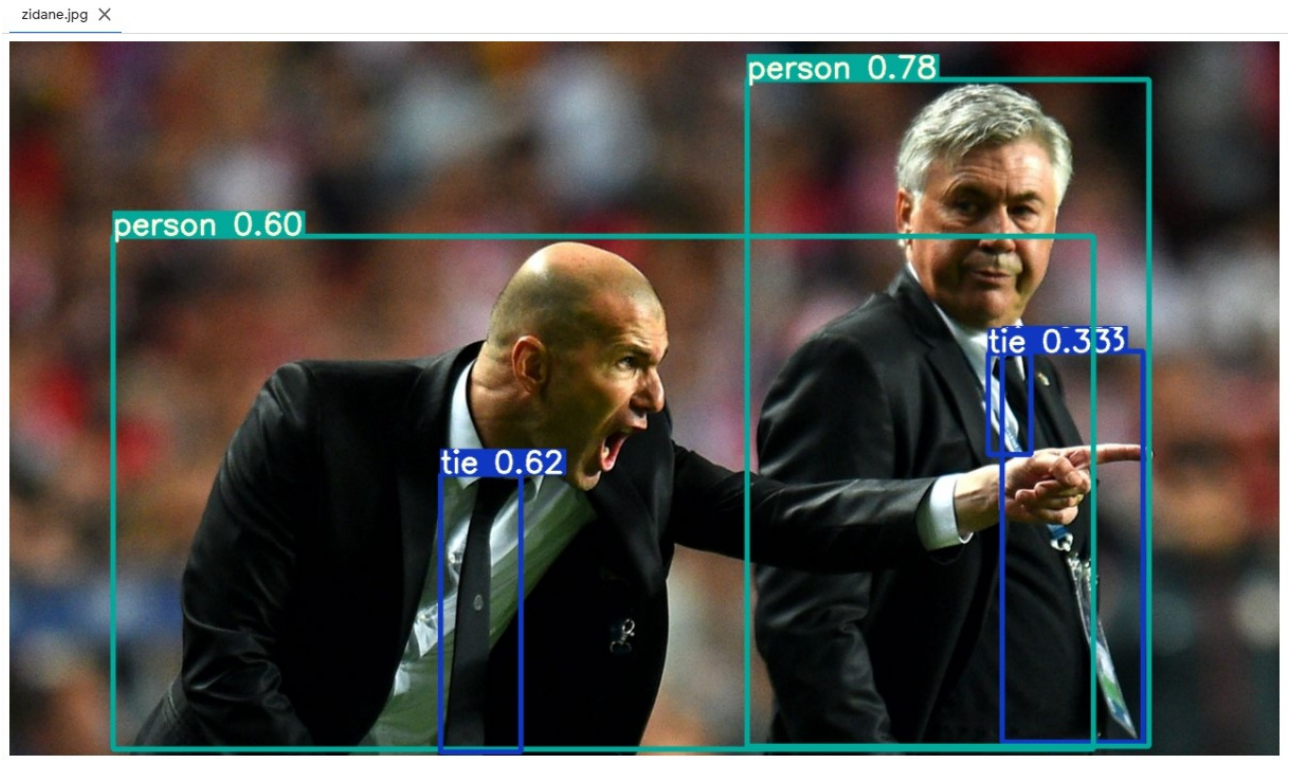


Figura 9: Screenshot dell'immagine campione dopo il detect.

valoriboundingbox.txt

```
1 person 0.25 runs/detect/exp7/bus.jpg  [0.04691357910633087, 0.675000011920929, 0.09382715821266174, 0.4888888895511627]
2 person 0.61 runs/detect/exp7/bus.jpg  [0.1796296238899231, 0.6032407283782959, 0.24567900598049164, 0.4694444537162781]
3 bus 0.65  runs/detect/exp7/bus.jpg  [0.4913580119609833, 0.4453703761100769, 0.9555555582046509, 0.4703703820705414]
4 person 0.85 runs/detect/exp7/bus.jpg  [0.9117283821105957, 0.5902777910232544, 0.17407406866550446, 0.4546296298503876]
5 person 0.90 runs/detect/exp7/bus.jpg  [0.3499999940395355, 0.5874999761581421, 0.1518518477678299, 0.42314815521240234]
6 tie 0.33  runs/detect/exp7/zidane.jpg [0.836718738079071, 0.706944465637207, 0.11093749850988388, 0.5472221970558167]
7 tie 0.33  runs/detect/exp7/zidane.jpg [0.787109375, 0.5069444179534912, 0.03359375149011612, 0.14166666567325592]
8 person 0.60 runs/detect/exp7/zidane.jpg [0.4671874940395355, 0.6312500238418579, 0.7718750238418579, 0.7180555462837219]
9 tie 0.62  runs/detect/exp7/zidane.jpg [0.3707031309604645, 0.800000011920929, 0.06328125298023224, 0.3888888955116272]
10 person 0.78 runs/detect/exp7/zidane.jpg [0.7386718988418579, 0.519444465637207, 0.31640625, 0.9333333373069763]
11
```

Figura 10: Screeshot del contenuto della cartella 'valoriboundingbox'.

Se al posto di un'immagine si passa al script detect.py in input un video, il programma lo dividerà in frames che verranno salvati all'interno della cartella 'output', mentre nella cartella 'valoriboundingbox' verranno salvati tutti i dati che fanno riferimento al detect degli oggetti all'interno del video.

CAPITOLO 2

STRUMENTI UTILIZZATI

2.1 ROS (Robot Operating System)

2.1.1 INTRODUZIONE

ROS (Robot Operating System) è un framework per lo sviluppo e la programmazione di robot [5].

Figura 11: logo del Ros. Immagine presa da [5]

Originariamente sviluppato dall'Università di Stanford nel 2007 [6], ROS non è realmente un sistema operativo, ma un Software Development Kit (SDK) che si utilizza per sviluppare applicazioni di robotica. Esso fornisce i servizi standard di un sistema operativo, come: astrazione dell'hardware, controllo dei dispositivi tramite driver, comunicazione tra processi, gestione delle applicazioni (packages) e altre funzioni di uso comune. Un insieme di processi all'interno di ROS si possono rappresentare in un grafo come dei nodi che possono ricevere, inviare e "multiplexare" i messaggi provenienti da e verso altri nodi, sensori e attuatori.

Nell'architettura di ROS i software possono essere raggruppati in tre categorie:

- strumenti per lo sviluppo e pubblicazione di software basato su ROS;
- librerie utilizzabili dai processi ROS client come roscpp, rospy e roslisp;
- pacchetti (packages) contenenti applicazioni e codice che usa una o più librerie per processi ROS client.

2.1.2 FILOSOFIA DEL ROS

Le caratteristiche del ROS sono fondamentali per l'utilizzo nel mondo della robotica. Esso, infatti, è un insieme di framework:

- ***Peer to peer:*** i singoli programmi (ROS messages, service, etc.); comunicano su API definite;
- ***Distributed:*** i programmi possono essere eseguiti su più computer e comunicare in rete;
- ***Multi-lingual***: i moduli ROS possono essere scritti in qualsiasi linguaggio per il quale esiste una libreria client (C++ e Python sono i più utilizzati);
- ***Light-weight:*** ROS sfrutta librerie già scritte nei vari linguaggi di programmazione rendendolo di fatto molto performante;
- ***Free and open-source:*** La maggior parte del software ROS è open source e gratuito***.***

2.1.3 CATKIN

Catkin è il sistema di compilazione ufficiale di ROS e il successore del sistema di compilazione ROS originale, il rosbuild. Esso, infatti, si presenta più convenzionale rispetto al suo predecessore, consentendo una migliore distribuzione dei pacchetti, un migliore supporto per la compilazione incrociata e una migliore portabilità.

Catkin combina le macro di *CMake* e gli script Python per fornire alcune

funzionalità, oltre al normale flusso di lavoro di CMake.
Affinché si possa effettuare il *build*, Catkin deve entrare in possesso di varie informazioni, come le posizioni del codice sorgente, le dipendenze del codice, dipendenze esterne e dove si trovano, quali elementi devono essere compilati e dove devo essere installati. Questo è tipicamente espresso in una serie di file di configurazione letti dal sistema di compilazione. Con CMake, è specificato in un file tipicamente chiamato 'CMakeLists.txt' e con GNU Make è all'interno di un file tipicamente chiamato 'Makefile'. Il sistema di compilazione utilizza queste informazioni per elaborare e creare il codice sorgente nell'ordine appropriato.
Nonostante *rosbuild* sia stato un buon sistema di compilazione per ROS sin dal suo inizio, la rapida crescita del codebase di ROS ha mostrato alcuni inconvenienti al suo approccio che Catkin tenta di alleviare, infatti viene utilizzato perché agevola la costruzione e l'esecuzione di codice ROS utilizzando strumenti e convenzioni per semplificare il processo. Per installare Catkin, bisogna eseguire questo comando con il terminale Linux (tutto il lavoro della tesi è stato svolto con il sistema operativo Ubuntu 16.04 e il ROS nella sua versione chiamata *kinetic*):
sudo apt-get install ros-kinetic-catkin
La creazione di un workspace di Catkin verrà mostrata nel capitolo 3.

Catkin è composto da tre cartelle fondamentali:

- ***src***: contiene il codice sorgente. È la cartella che si utilizza per poter creare, clonare ed editare codice sorgente per generare le risorse di cui abbiamo bisogno;
- ***build***: è la cartella dove il CMake viene chiamato per generare i packages a partire dai codici sorgente;
- ***devel***: è la cartella dove i files compilati sono memorizzati.

Con il comando:

```
catkin clean
```

è possibile pulire l'intero contenuto delle cartelle build e devel. Se vogliamo controllare o modificare delle impostazioni di catkin, è possibile tramite il comando:

```
catkin config
```

2.1.4 ROS MASTER

Il ROS Master gestisce la comunicazione tra i vari processi in esecuzione all'interno del framework. Il ruolo del Master è quello di consentire ai singoli nodi ROS di localizzarsi l'un l'altro. Una volta che questi nodi si sono localizzati, comunicano tra loro peer-to-peer. Il Master viene più comunemente eseguito utilizzando il comando:

```
roscore
```

che carica il ROS Master insieme ad altri componenti essenziali.
In figura 13 viene riportata la schermata principale del ROS Master.

Figura 12: ROS Mater. Immagine tratta da [6]

Figura 13: Screenshot del comando roscore.

2.1.5 ROS NODE

Un ROS Node è un programma eseguibile all'interno di un ROS package.
I ROS Nodes usano una libreria del ROS client per comunicare con altri nodi, sono organizzati in packages, possono pubblicare o iscriversi a un ROS Topic e fornire o usare un ROS Service.
Un ROS Node può essere avviato attraverso il comando:

```
rosrun nome_del_package nome_del_rosnode
```

Con il comando:

```
rosnode list
```

è possibile visualizzare la lista di tutti i nodi attivi.
Infine, con il comando

```
rosnode info nome_del_rosnode
```

si possono ottenere le informazioni su un nodo specifico.

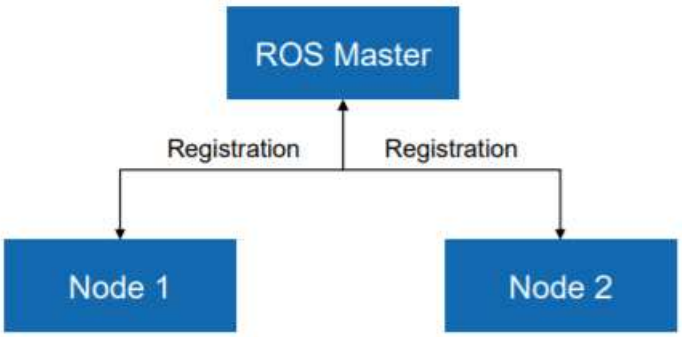


Figura 14: comunicazione tra ROS Nodes e ROS Master. Immagine tratta da [6]

2.1.6 ROS TOPICS

I ROS Topics sono bus denominati su cui i nodi si scambiano messaggi, o meglio, sono il nome di un flusso di messaggi. Un nodo può svolgere la funzione di publisher o di subscriber. Tipicamente si ha un solo nodo publisher ed n nodi subscribers.

Come per i nodi, attraverso il comando:

rostopic list

si può ottenere una lista dei topics attivi.

Con il comando

rostopic echo/topic

si può stampare il contenuto di un Topic.

Infine per ottenere informazioni di un Topic si utilizza il comando:

rostopic info/topic.

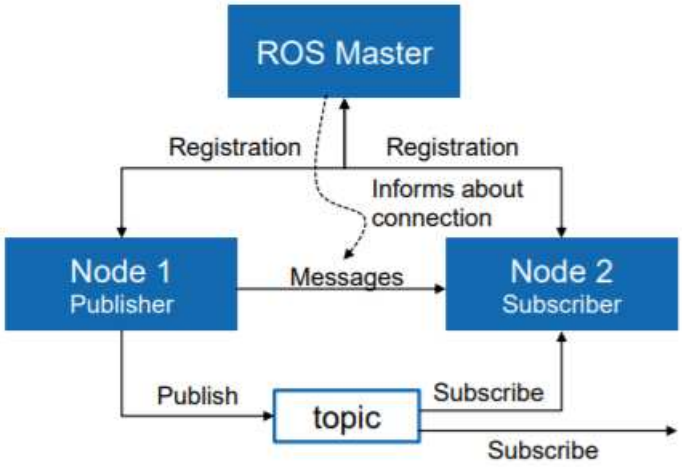


Figura 15: Comunicazione tra ROS Master, Nodes e Topic. Immagine tratta da [6]

2.1.7 ROS MESSAGES

I nodi comunicano tra loro pubblicando messaggi nei Topics. Un messaggio è una semplice struttura dati, composta da campi digitati; sono supportati i tipi primitivi standard (integer, floating point, boolean, ecc.), così come gli array di tipi primitivi e possono includere strutture e array annidati arbitrariamente, come si può vedere in figura 10.

Un ROS Message è definito all'interno di un file con estensione *.msg.

Per visualizzare il tipo di un determinato ROS Message, si usa il comando:

```
rostopic type /nome_del_topic
```

Per pubblicare un messaggio su un topic si utilizza il comando:

```
rostopic pub /nome_del_topic type data
```

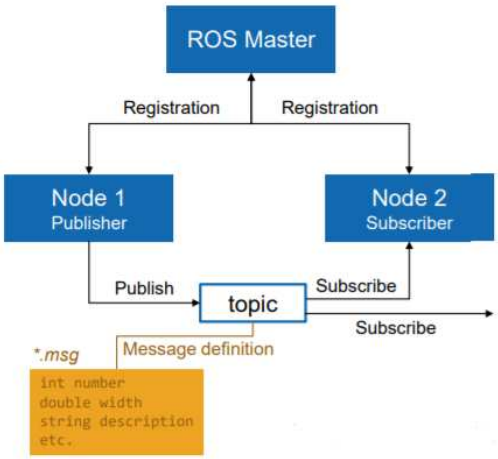


Figura 16: Comunicazione tra ROS Master, Nodes, Topic e Messages. Immagine tratta da [6].

2.1.8 ROS LAUNCH

ROS Launch è un tool che permette di eseguire più nodi. Sono scritti in XML con estensione *.launch. Se non è già stato fatto, all'avvio di un ROS Launch viene avviato anche il ROS Master. Il comando per eseguire un file *.launch è:

```
roslaunch nome_del_file.launch
```

2.1.9 ROS PACKAGES

Il software ROS è organizzato in packages, che possono contenere codice sorgente, *launch files*, file di configurazione, definizioni di messaggi, dati e documentazione.

Un package che richiede altri componenti per poter essere compilato ha bisogno di dichiarare le dipendenze con il seguente comando:

```
catkin_create_pkg nome_del_package {dipendenze}
```

Un esempio di ROS Package è mostrato in figura 11.

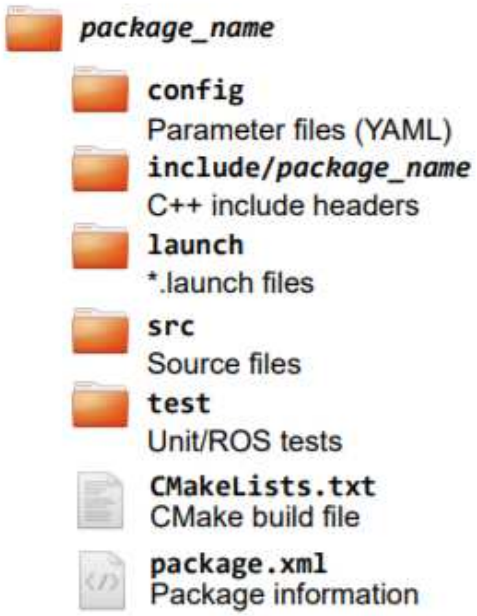


Figura 17: Esempio di ROS Package

2.1.9.1 package.xml

Il package.xml definisce le proprietà del package come il nome, autori, versione e dipendenze con altri package.

2.1.9.2 CMakeLists.txt

Il CMakeLists.txt rappresenta il file di input che viene mandato al CMakeBuild system. Esso contiene:

- la versione del CMake richiesta (cmake_minimum_required(VERSION ...));
- il nome del package (project());
- indicazioni su dove trovare gli altri packages Cmake o Catkin necessari per la compilazione (find_package());
- generatori di messaggi, servizi e azioni (add_message_files(), add_service_files() e add_action_files());
- l'invocazione nella generazione di messaggi (generate_messages());

- specifiche sui package build info export (catkin_package());
- librerie o eseguibili da compilare ((add_library() add_executable() target_link_libraries());
- test per la compilazione (catkin_add_gtest());
- metodi per installare regole (install());

2.1.10 ROS SERVICES

La comunicazione client/server tra nodi è realizzata con il ROS Service attraverso l'utilizzo di due elementi principali:

- il *service server* che promuove il servizio;
- il *service client* che usufruisce del servizio;

I Ros Services vengono definiti in file di estensione *.srv.
Il comando che ci permette di ottenere la lista dei servizi disponibili è:

```
rosservice list
```

Per conoscere il tipo di Service utilizziamo:

```
rosservice type /nome_del_servizio
```

Per chiamare un servizio fornito da un ROS Service si utilizza:

```
rosservice call /nome_del_servizio args.
```

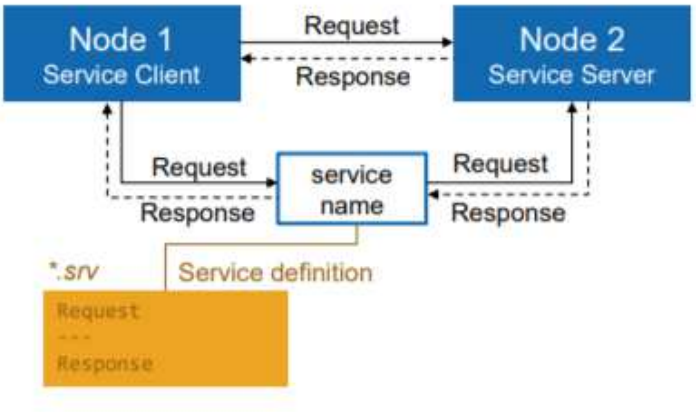


Figura 18: comunicazione tra ROS Service e due ROS Nodes. Immagine tratta da [6]

2.1.11 ROS ACTIONS

Simile alle chiamate dei ROS Services, le ROS Actions forniscono delle funzionalità aggiuntive:

- cancellare un *task*;
- ricevere un feedback sul progresso dell'attività;

Le ROS Actions vengono memorizzate in files con estensione *.action.

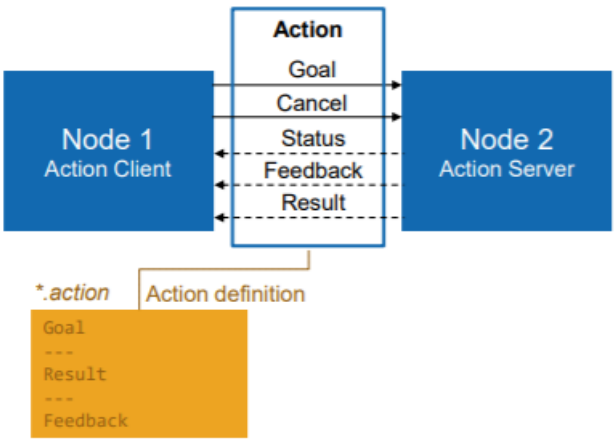


Figura 19: Comunicazione tra nodi e ROS Actions. Immagine tratta da [6].

2.1.12 ROS TIME

Il ROS Time è uno strumento molto utile nelle simulazioni quando bisogna simulare l'andamento nel tempo di alcuni comportamenti. Per lavorare con un clock simulato, si utilizza:

```
rosparam set use_sim_time true
```

2.1.13 ROS BAGS

Un Bag è un formato utilizzato dal ROS per immagazzinare i messaggi. Sono in formato binario e sono memorizzati con estensione *.bag. Per registrare tutto ciò che viene inserito all'interno di tutti i ROS Topics, si utilizza il seguente comando:

```
rosbag record –all
```

Mentre se vogliamo registrare tutti i messaggi che sono stati inviati in topics specifici, si utilizza:

```
rosbag record topic_1 topic_2 topic_3
```

La registrazione viene interrotta con l'utilizzo di ctrl+C. I file .bag vengono memorizzati con una data d'inizio e l'ora di memorizzazione ad esempio: *2020-04-12-19-00-00.bag*.

Per visualizzare le informazioni di un bag si utilizza il comando:

```
rosbag info nome_del_file.bag
```

Per leggere un file bag e stamparne tutti i suoi contenuti sul terminale si utilizza:

```
rosbag play nome_del_file.bag
```

Esistono delle opzioni di visualizzazione che possono essere fornite dopo il comando play, ad esempio:

```
rosbag play –rate=0.5 nome_del_file.bag
```

2.1.14 RQT IMAGE VIEW

Rqt Image View è uno strumento di visualizzazione per le simulazioni effettuate con ESIM. Si lancia con il comando:

```
rosrun rqt_image_view
```

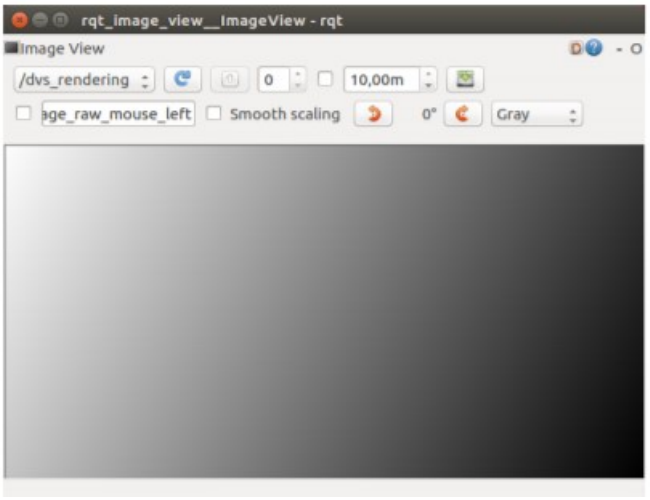


Figura 20: Screenshot della schermata iniziale di Rqt Image View.

2.3 INSTALLAZIONE DEL ROS

Per lo svolgimento del progetto si è utilizzato la versione Kinetic di ROS, compatibile solamente con le versioni 15.10 e 16.01 di Ubuntu. Ovviamente si possono utilizzare altre versioni del ROS, a seconda del sistema operativo che si sta utilizzando.

Il primo passo è impostare il proprio PC per accettare il software che proviene dal packages.ros.org utilizzando il comando:

```
sudo sh -c ‘echo “deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main” > /etc/apt/sources.list.d/ros-latest.list
```

Successivamente, si impostano le chiavi con:

```
sudo apt−key adv −−keyserver ’hkp:/ keysrver.ubuntu.com:80’
−−recv−key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Prima di tutto si verifica se l'indice del Debian package è aggiornato:

```
sudo apt-get update
```

Si avvia l’installazione completa di ROS con il seguente comando:

```
sudo apt-get install ros-kinetic-desktop-full
```

È conveniente che le variabili dell'ambiente ROS siano automaticamente aggiunte alla bash session ogni volta che una nuova shell viene lanciata:

```
echo “source /opt/ros/kinetic/setup.bash” >> ∼/.bashrc
source ∼/.bashrc
```

A questo punto si è installato tutto ciò che necessario per avviare i packages fondamentali del ROS ma se si vuole permettere la creazione di workspaces custom, bisogna installare questo tool tramite il seguente comando da terminale:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-
generator python-wstool build-essential
```

Prima di iniziare ad usare i tools presenti nel ROS, c’è bisogno di inizializzare il rosdep che permette di installare le dipendenze del codice che si vuole compilare ed è fondamentale per eseguire i componenti del ROS. Rosdep si installa con il seguente comando:

```
sudo apt install python-rosdep
```

e si analizza con i seguenti comandi:

```
sudo rosdep init
rosdep update
```

2.4 Python

Figura 21: Logo Python. Immagine presa da [7].

Si è utilizzato il linguaggio di programmazione *Python* (versione 3.5) per effettuare la simulazione degli eventi a partire da un video per ESIM e per sviluppare l'algoritmo finale, utilizzando l'editor PyCharm.

2.5 FFMPEG

Figura 22: Logo FFMPEG. Immagine tratta da [8].

Nella simulazione degli eventi a partire da un file video con ESIM, è stato utile l'utilizzo di un software per l'elaborazione di file video chiamato FFmpeg. In particolare, è stato utilizzato per tagliare il video campione e migliorarne la qualità per agevolare la simulazione su ESIM.

CAPITOLO 3

SIMULATORE ESIM

3.1 INTRODUZIONE

Le camere ad eventi sono sensori rivoluzionari che funzionano in modo radicalmente diverso dalle camere tradizionali: invece di acquisire immagini di intensità a un *framerate* fisso, le camere ad eventi misurano i cambiamenti di intensità in modo asincrono, sotto forma di un flusso di eventi, che codificano i cambiamenti di luminosità per pixel [9].

Negli ultimi anni, le loro eccezionali proprietà come, il rilevamento asincrono, il *motion blur*, l'HDR, hanno portato ad applicazioni di visione interessanti, con una latenza molto bassa e un'elevata robustezza. Tuttavia, questi sensori sono costosi e non ottimali, rallentando il progresso della comunità di ricerca. Ne è un esempio è il DAVIS 346: esso possiede ancora una bassa risoluzione (346x260), uno scarso rapporto segnale/rumore, una complessa configurazione, che richiede il supporto di esperti, e il suo prezzo è ancora troppo alto, circa 5000€. Questi problemi irrisolti non rendono giustizia alle reali potenzialità delle camere ad eventi e la rivoluzione che potrebbero portare nei vari campi di applicazione se fossero più accessibili.

Per questo motivo si è pensato di creare dei simulatori che permettessero di ottenere un output il più vicino possibile a quello di una camera ad eventi.

Il simulatore che si è utilizzato per lo svolgimento del progetto è l'ESIM: un efficiente simulatore di telecamera per eventi implementato in C++ e disponibile open source.

ESIM può simulare il movimento arbitrario della telecamera in scene 3D,

fornendo al contempo eventi, immagini standard, misurazioni inerziali con informazioni complete sulla verità del terreno, tra cui la posizione della telecamera, la velocità, nonché mappe di profondità e flusso ottico.
ESIM combina un simulatore di eventi con i motori grafici per consentire una simulazione e accurata mediante l'utilizzo dell'*adaptive sampling*.

3.2 ADAPTIVE SAMPLING

Per simulare una camera ad eventi, è necessario accedere ad una rappresentazione continua del segnale visivo in ogni pixel, che non è ottenibile nella pratica. Per aggirare questo problema, lavori precedenti hanno proposto di campionare il segnale visivo (cioè fotogrammi campione) in modo sincrono, ad un *framerate* molto alto, ed eseguire un'interpolazione tra i campioni per ricostruire un'approssimazione lineare a tratti del segnale visivo continuo, che viene utilizzato per emulare il principio di funzionamento di una telecamera per eventi.
In ESIM si utilizza lo stesso approccio generale per la simulazione degli eventi, ovvero: si campiona il segnale visivo (mediante il rendering di immagini lungo la traiettoria della telecamera) con la differenza fondamentale che, invece di scegliere un *framerate* di rendering arbitrario e campionare i fotogrammi in modo uniforme nel tempo al *framerate* scelto, si propone di campionare i frame in modo adattivo, cioè adattando la frequenza di campionamento in base alle dinamiche previste del segnale visivo.

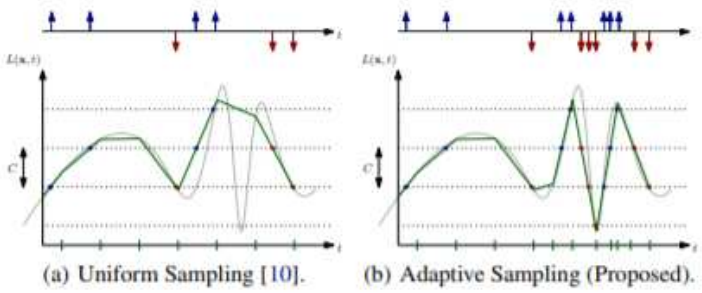


Figura 23:Uniform (a) e Adaptive (b) Sampling. Immagine tratta da [9].

Il tutto si può vedere dalla figura 16: le due strategie producono eventi simili quando il segnale varia poco; quando quest'ultimo varia considerevolmente, l'*adaptive sampling* produce una simulazione molto più accurata dell'*uniform sampling* in quanto il *framerate* cambia con il variare del segnale visivo.
Un campionamento adattativo del segnale visivo richiede una stretta integrazione tra il *rendering engine* e il simulatore di eventi.

3.2.1 VANTAGGI DELL'ADAPTIVE SAMPLING

Nella figura 3.3 a sinistra, si ha l'RMSE (*Root Mean Standard Error*) del segnale di luminosità in funzione del *framerate* per ogni strategia di campionamento ("all" per indicare che tutti i campioni sono analizzati, "slow" per indicare un *framerate* basso" e "fast" per un *framerate* alto) elaborato nell'intera sequenza di valutazione. Nella figura 3.3 a destra viene rappresentato l'RMSE del segnale di luminosità ricostruito in funzione del numero medio di campioni al secondo elaborato per ogni strategia di campionamento, esaminati nell'intera sequenza di valutazione.

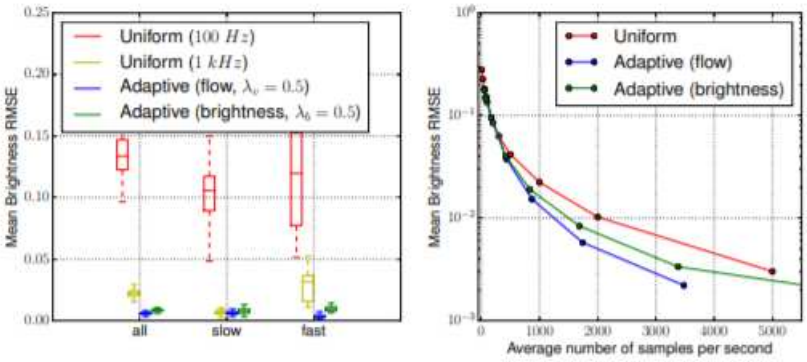


Figura 24: Confronto tra varie metodologie di campionamento. Immagine tratta da [9].

3.3 ARCHITETTURA DELL'ESIM

Come si può vedere in figura 18, l'architettura del simulatore associa direttamente il *rendering engine* con il simulatore di eventi. Questo consente all'ESIM di elaborare i frames basandosi sulla dinamica del segnale visivo.

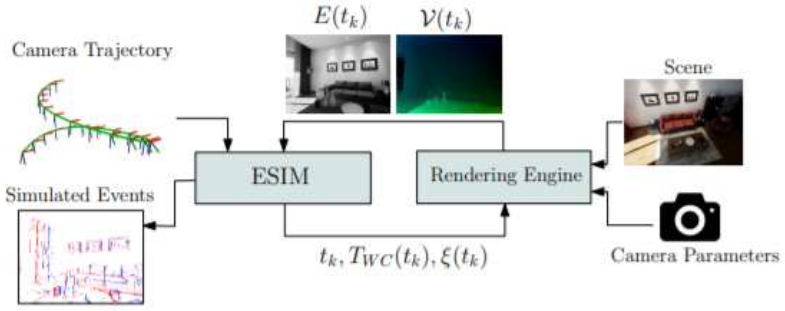


Figura 25: Architettura di ESIM. Immagine tratta da [9].

Il simulatore ESIM è composto da due elementi fondamentali:

- ***sensor trajectory***: è una funzione regolare τ che mappa ad ogni instante di tempo t, una posa, il *twist* (velocità angolare e lineare) e l'accelerazione del sensore. Si denota la posa del sensore espressa in alcuni frame inerziali W con $T_{WB} \in SE$, dove SE definisce un gruppo di

movimenti rigidi in 3D, il *twist* è definito con $\xi(t) = ({}_W v(t), {}_B\omega_{WB}(t))$, e la sua accelerazione è espressa (nel frame inerziale) con ${}_B a(t)$.

- ***rendering engine***: il *renderer* è una funzione reale che mappa ad ogni istante di tempo t un'immagine renderizzata della scena in base alla posa corrente del sensore. Esso è parametrizzato dall'ambiente ε, variabile che controlla le geometrie della scena e le sue dinamiche, dal sensor trajectory all'interno dell'ambiente τ, variabile che rappresenta la traiettoria della camera nell'ambiente e dalla configurazione Θ, variabile che rappresenta la configurazione del sensore di visione simulata e che include dei parametri della camera come la grandezza del sensore, lunghezza focale e parametri di distorsione. Il *renderer* fornisce anche il *motion field v(tk).*

Descrivendo la figura 23, il simulatore ESIM, al tempo t_k, campiona una nuova posa della camera $T_{WC}(t_k)$ e il twist $\xi(t_k)$ a partire dalla traiettoria definita dall'utente e li passa al *rendering engine* che renderizza una nuova mappa d'irradianza $E(t_k)$ e una mappa del *motion field* $v(t_k)$ che serviranno per produrre la simulazione degli eventi. Il *motion field* viene usato per calcolare il cambiamento previsto di luminosità che a sua volta permette di scegliere il tempo di rendering successivo t_{k+1}.

3.4 STRATEGIA DELL'ESIM

Il principio di lavoro dell'ESIM si basa su due metodi:

- ***cambiamento di luminosità***: Sotto le ipotesi delle superfici lambertiane, si considera una espansione di Taylor al primo ordine della luminosità:

$$\frac{\partial L(x;t_k)}{\partial t} \simeq -\langle \nabla L(x;t_k),\, v(x;t_k) \rangle \qquad (3.1)$$

dove ∇L è il gradiente della luminosità dell'immagine e v è il *motion field*. Quindi $\nabla L \simeq \frac{\partial L(x;t_k)}{\partial t}\Delta t$ è la variazione (positiva o negativa) di luminosità prevista al pixel x e al tempo t_k durante un dato intervallo di tempo Δt. Si assicura che per ogni pixel $x \in \Omega$, $|\Delta L| \leq C$. Questo significa che la variazione di luminosità è limitata dalla soglia di contrasto desiderata C della camera ad eventi. L'istante di tempo t_{k+1} viene calcolato nel modo seguente:

$$t_{k+1} = t_k + \lambda_b C \left|\frac{\partial L}{\partial t}\right|_m^{-1} \tag{3.2}$$

dove: $\left|\frac{\partial L}{\partial t}\right|_m = max_{x\epsilon\Omega} \frac{\partial L(x;t_k)}{\partial t}$ è la massima variazione prevista della luminosità nel piano dell'immagine Ω, e $\lambda_b \leq 1$ è un parametro che controlla lo scambio tra la velocità di rendering e l'accuratezza. L'equazione (3.2) impone quindi che il framerate del rendering cresca proporzionalmente con la massima variazione assoluta di luminosità prevista nell'immagine. Tuttavia, non c'è garanzia che il segnale sia stato campionato correttamente, a causa di effetti non lineari. Per far fronte a questo problema, si può scegliere un valore di $\lambda_b < 1$.

- ***spostamento dei pixel***: questo metodo si basa sull'assunzione che il massimo spostamento di un pixel tra due frames successivi è limitato. Questo metodo può essere adottato scegliendo un tempo di campionamento successivo t_{k+1} come segue:

$$t_{k+1} = t_k + \lambda_v |v(xk; tk)|_m^{-1} \tag{3.3}$$

dove $|v| = max x\in\Omega\ |v(x;tk)|$ è la magnitudo massima del *motion field* al tempo t_k, e $\lambda_v \leq 1$. Come detto sopra, si sceglie λ_v per mitigare l'effetto delle non-linearità presenti.

Nel simulatore sono implementati dei modelli standard di disturbo per le camere ad eventi. I modelli sono ottenuti campionando il contrasto in base alla

variabile $N\ (C,\ \sigma_C)$, dove σ_C controlla l'ammontare del disturbo impostato dall'utente.

3.5 INSTALLAZIONE

Prima di installare il simulatore ESIM, è necessario aver scaricato il ROS (capitolo 2, sezione 3).
Prima di tutto è necessario installare *catkin_tools*, eseguendo i comandi:

```
sudo sh −c 'echo "deb http://packages.ros.org/ros/ubuntu `lsb_release −sc` main" > /etc/apt/sources.list.d/ros−latest.list '

wget http://packages.ros.org/ros key −O − | sudo apt−key add –
sudo apt−get update
sudo apt−get install python−catkin−tools
```

In seguito, bisogna creare un workspace fatto esclusivamente per contenere il simulatore. Si effettua con i seguenti comandi:

```
mkdir −p ∼/sim_ws/src && cd ∼/sim_ws
catkin init
catkin config −−extend /opt/ros/kinetic −−cmake−args
−DCMAKE_BUILD_TYPE=Release
```

Successivamente, installare *vcstools*:

```
sudo apt−get instal  python−vcs tool
```

Si clona il repository importato (si ricorda di creare una chiave SSH per il proprio account GitHub) e si procede poi all'avvio di *vcstools*:

```
cd src/
git clone git@github.com:uzh−rpg/rpg_esim.git
vcs−import < rpg_esim/dependencies.yaml
```

Vengono poi installati vari software utili per il simulatore, quali: *pcl_ros, glm, glfw*:

```
sudo apt-get install ros-kinetic-pcl-ros
sudo apt-get install libglfw3 libglfw3-dev
sudo apt-get install libglm-dev
```

Ora, è utile disabilitare alcune dipendenze non necessarie:

```
cd ze_oss
touch imp_3rdparty_cuda_toolkit /CATKIN_IGNORE \
imp_app_pangolin_example/CATKIN_IGNORE \
imp_benchmark_aligned_allocator /CATKIN_IGNORE \
imp_bridge_pangolin /CATKIN_IGNORE \
imp_cu_core/CATKIN_IGNORE \
imp_cu_correspondence /CATKIN_IGNORE \
imp_cu_imgproc/CATKIN_IGNORE \
imp_ros_ro f_denoising /CATKIN_IGNORE \
imp_tools_cmd/CATKIN_IGNORE \
ze_data_provider /CATKIN_IGNORE \
ze_geometry /CATKIN_IGNORE \
ze_imu/CATKIN_IGNORE \
ze_trajectory_analysis/CATKIN_IGNORE
```

e compilare il nodo *event_camera_simulator* con:

```
catkin build esim_ros
```

Infine, si crea un alias per il workspace in modo tale che esso può essere richiamato velocemente ad ogni avvio della *bash*:

```
echo "source ~/sim_ws/devel/setup.bash" >> ~/setupeventsim.sh
chmod +x ~/setupevent sim.sh
```

Aggiungiamo le modifiche al file *.bashrc* con la seguente riga di comando:

```
alias ssim='source ~/setupeventsim.sh'
```

Quindi adesso, con il comando *ssim* si inizializzerà il workspace del simulatore. Ciò è fondamentale per lavorare con il simulatore ESIM.

CAPITOLO 4

RISULTATI

4.1 FASI PRELIMIARI

4.1.1 DOWNLOAD DEL VIDEO

Come video di prova per la simulazione di eventi, si è scelto un video ripreso da un drone Cinewhoop FPV, che mostra l'interno di una casa, scaricato dall'applicazione 'Youtube: Convertitore' (file MP4 per video).

Per prima cosa, si crea una nuova cartella chiamata *example* all'interno della cartella *tmp*. Successivamente ci si sposta all'interno di essa:

```
mkdir −p /tmp/ example 2 cd /tmp/ example
```

Adesso si utilizzano gli strumenti offerti da *FFmpeg* per ridurre la durata del video: verrà creato un secondo file video in formato *.mkv*, che contiene soltanto la parte del video che va dal minuto *0'13* e finisce al minuto *1'13*, attraverso il seguente comando:

```
ffmpeg -i drone.mkv -ss 00:00:13 -t 00:01:00 -async 1 -strict -2
drone_cut.mkv
```

Con -i si indica il video da fornire in input a *FFmpeg*, in questo caso drone.mkv. Con -ss si indica da quale punto del video iniziare a salvare il video. Con *-t*, si indica la durata della lettura del video in input (in questo caso si salverà *1* minuto a partire dal minuto *0'13*).

Se il video presenta un valore di fps troppo elevato, si consiglia di ridimensionare il video.

4.1.2 PRE-PRECESSING DEL VIDEO PER ESIM

Attraverso i comandi:

mkdir frames

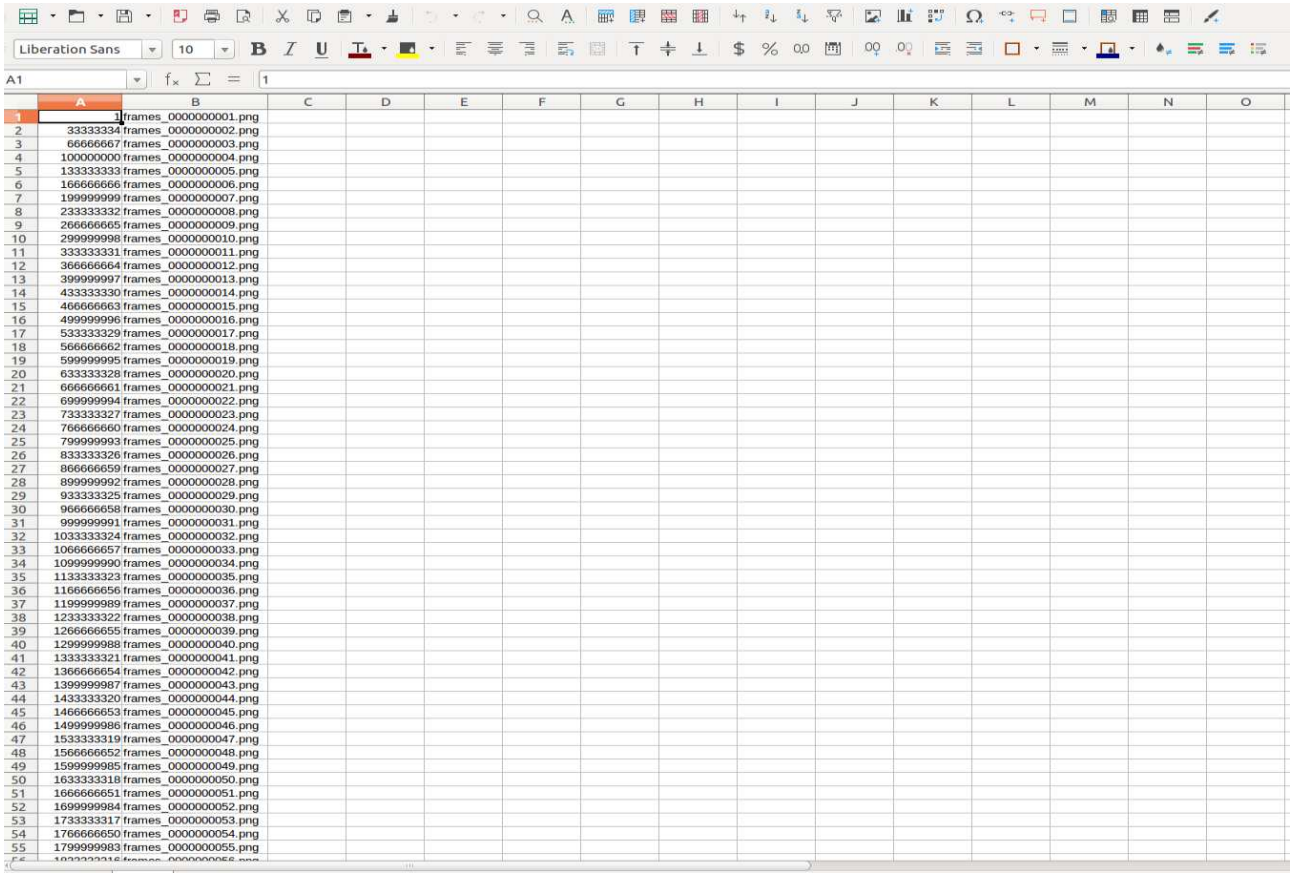
f fmpeg −i vi d eo_ t e s t . mkv f rame s / frames_%010d . png

si creerà una cartella frames che conterrà tutti i frames in png del video stesso, necessaria per la creazione del file images.csv con lo script python generate_stamp_file.py, contenuto all'interno della cartella scripts (percorso:sim_ws/src/rpg_esim/event_camera_simulator/esim_ros/scripts), mostrato nella figura sottostante.

```
import argparse
from os import listdir
from os.path import join

if __name__ == "__main__":

    parser = argparse.ArgumentParser(
        description='Generate "images.csv" for ESIM DataProviderFromFolder')

    parser.add_argument('-i', '--input_folder', default=None, type=str,
                        help="folder containing the images")
    parser.add_argument('-r', '--framerate', default=1000, type=float,
                        help="video framerate, in Hz")

    args = parser.parse_args()

    images = sorted(
        [f for f in listdir(args.input_folder) if f.endswith('.png')])

    print('Will write file: {} with framerate: {} Hz'.format(
        join(args.input_folder, 'images.csv'), args.framerate))
    stamp_nanoseconds = 1
    dt_nanoseconds = int((1.0 / args.framerate) * 1e9)
    with open(join(args.input_folder, 'images.csv'), 'w') as f:
        for image_path in images:
            f.write('{},{}\n'.format(stamp_nanoseconds, image_path))
            stamp_nanoseconds += dt_nanoseconds

    print('Done!')
```

Figura 26: Screenshot del file generate_stamps_file.py.

Questo script genera un file formato in .csv, formato da due colonne: la prima contiene l'istante di tempo relativo ad un determinato frame del video fornito in input, mentre la seconda fornisce il nome del file .png associato al frame.



	A	B
1	1	frames_0000000001.png
2	33333334	frames_0000000002.png
3	66666667	frames_0000000003.png
4	100000000	frames_0000000004.png
5	133333333	frames_0000000005.png
6	166666666	frames_0000000006.png
7	199999999	frames_0000000007.png
8	233333332	frames_0000000008.png
9	266666665	frames_0000000009.png
10	299999998	frames_0000000010.png
11	333333331	frames_0000000011.png
12	366666664	frames_0000000012.png
13	399999997	frames_0000000013.png
14	433333330	frames_0000000014.png
15	466666663	frames_0000000015.png
16	499999996	frames_0000000016.png
17	533333329	frames_0000000017.png
18	566666662	frames_0000000018.png
19	599999995	frames_0000000019.png
20	633333328	frames_0000000020.png
21	666666661	frames_0000000021.png
22	699999994	frames_0000000022.png
23	733333327	frames_0000000023.png
24	766666660	frames_0000000024.png
25	799999993	frames_0000000025.png
26	833333326	frames_0000000026.png
27	866666659	frames_0000000027.png
28	899999992	frames_0000000028.png
29	933333325	frames_0000000029.png
30	966666658	frames_0000000030.png
31	999999991	frames_0000000031.png
32	1033333324	frames_0000000032.png
33	1066666657	frames_0000000033.png
34	1099999990	frames_0000000034.png
35	1133333323	frames_0000000035.png
36	1166666656	frames_0000000036.png
37	1199999989	frames_0000000037.png
38	1233333322	frames_0000000038.png
39	1266666655	frames_0000000039.png
40	1299999988	frames_0000000040.png
41	1333333321	frames_0000000041.png
42	1366666654	frames_0000000042.png
43	1399999987	frames_0000000043.png
44	1433333320	frames_0000000044.png
45	1466666653	frames_0000000045.png
46	1499999986	frames_0000000046.png
47	1533333319	frames_0000000047.png
48	1566666652	frames_0000000048.png
49	1599999985	frames_0000000049.png
50	1633333318	frames_0000000050.png
51	1666666651	frames_0000000051.png
52	1699999984	frames_0000000052.png
53	1733333317	frames_0000000053.png
54	1766666650	frames_0000000054.png
55	1799999983	frames_0000000055.png

Figura 27:screenshot del file images.csv.

4.1.3 SIMULAZIONE DEGLI EVENTI CON ESIM

Dopo aver eseguito il ROS Master, attraverso il comando *roscore* da terminale, si inizia il processo di creazione degli eventi con ESIM con il comando:

```
rosrun esim_ros esim_node\
--data_source=2\
--path_to_output_bag=/tmp/out.bag\
--path_to_data_ folder=/tmp/video_test/frames\
--ros_publisher_frame_rate=60\
--exposure_time_ms =10.0\
--use_log_image=1\
--log_eps =0.1
--contrast_threshold_pos=0.15\
--contrast_threshold_neg=0.15
```

eseguendo un noto chiamato *esim_node* del *package esim_ros* in cui vengono indicati i seguenti argomenti:

- data_source=2 dice all'ESIM di leggere dati da una cartella che contiene immagini. Esso guarderà all'interno di un file .csv che associa gli istanti temporali alle immagini da leggere;
- –path_to_data_folder fornisce il percorso alla cartella dove sono memorizzati i file .csv e i frames in png;
- –path_to_output_bag fornisce il percorso dove salvare il file .bag;
- –ros_publisher_frame_rate indica il framerate del sensore APS simulato (in fps);
- –exposure_time_ms fornisce il tempo di esposizione del frame del sensore APS (in millisecondi);
- –use_log_image dice all'ESIM di operare nel dominio a intensità logaritmica. Quindi ogni immagine verrà convertita nel modo seguente: L = log $\frac{L}{255+eps}$, dove eps è impostato con il parametro –log_eps;
- –contrast_threshold_pos e –contrast_threshold_neg impostano i valori della soglia di contrasto (positiva e negativa). Un valore più basso comporta una sensitività più alta (e quindi molti eventi). Un valore più alto comporta una sensitività più bassa (e quindi meno eventi);

4.2 VISUALIZZAZIOME DEGLI EVENTI SIMULATI

Eseguendo il comando precedente, viene avviato l'ESIM e gli eventi vengono salvati nel file binario */tmp/out.bag*.

Successivamente si utilizza il package dvs_renderer con il comando:

```
rosrun dvs_ renderer dvs_renderer events:=/cam0/events
```

Per visualizzare gli eventi su un'immagine (utilizzabile quando non si dispone di un DAVIS) e il comando:

rosbag play /tmp/out.bag −l

Per riprodurre in loop (con -l) il file out.bag.

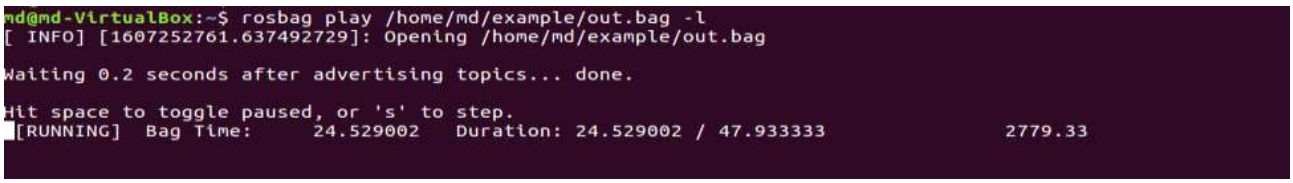


Figura 28: Screenshot comando rosbag.

Infine è possibile visualizzare gli eventi simulati aprendo rqt_image_view in un'altra finestra del terminale:

rqt_image_view / dvs_rendering

Viene aperta una finestra di rqt_image_view dove è possibile selezionare tra la visualizzazione dei soli eventi (figura 4.4) oppure quella dei frames APS (figura 4.5).

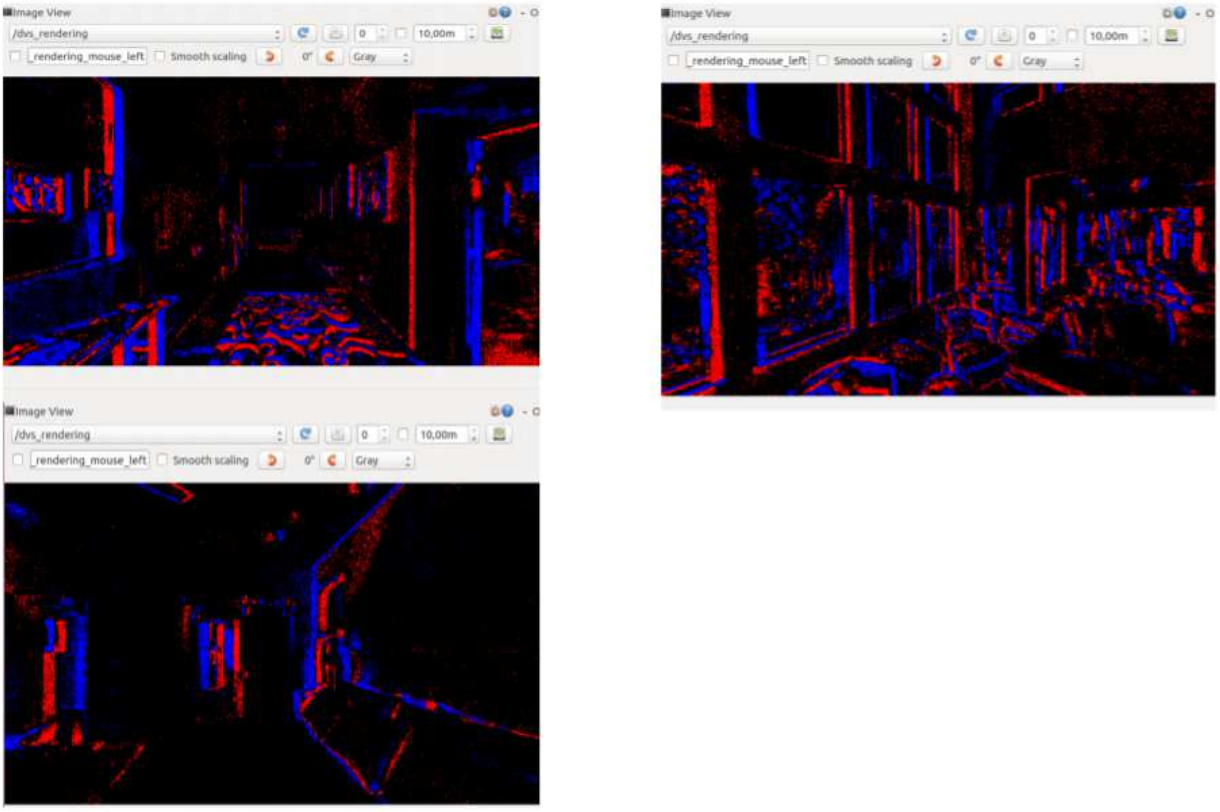


Figura 29: Screenshots della simulazione di eventi del video drone_cut.mkv

4.3 ALGORITMO FINALE

In questa sezione si spiegherà l'algoritmo per arrivare al tracking di ostacoli attraverso camera ad eventi.

Per prima cosa, nella cartella *tmp/example* si va a prendere il file binario out.bag dove sono memorizzati tutti gli eventi.

Convertiamo il file .bag in un file .csv con questo comando:

rostopic echo /cam0/events -b /home/md/example/out.bag -p > out.bag

in modo tale da aprirlo correttamente con LiberOfficer. Il file *out.csv* è descritto in figura

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	$etime	field.header.seq	field.header.stamp	field.header.frame_id	field.height	field.width	field.events0.x	field.events0.y	field.events0.ts	field.events0.polarity	field.events1.x	field.events1.y	field.events1.ts	field.events1.polarity
2	33333259	0	33333259		360	640	186	262	3438739	1	398	61	3447058	1
3	66666144	0	66666144		360	640	132	203	33339496	0	387	262	33344990	1
4	99999709	0	99999709		360	640	502	140	66669241	1	201	340	66672290	1
5	133332092	0	133332092		360	640	177	309	100003241	1	588	226	100003490	0
6	166666453	0	166666453		360	640	121	297	133335680	0	48	82	133340344	1
7	199999934	0	199999934		360	640	103	14	166666818	0	27	48	166669542	1
8	233333213	0	233333213		360	640	560	227	200001018	1	149	58	200001348	1
9	266666664	0	266666664		360	640	437	28	233333471	1	72	66	233334772	0
10	299999151	0	299999151		360	640	113	26	266669198	1	444	29	266672021	1
11	333333206	0	333333206		360	640	257	244	300000968	1	478	145	300002215	0
12	366666213	0	366666213		360	640	380	187	333335225	1	455	56	333335682	1
13	399999080	0	399999080		360	640	534	229	366667893	0	563	245	366670982	1
14	433333253	0	433333253		360	640	588	106	400000244	0	456	250	400001199	1
15	466666554	0	466666554		360	640	480	272	433333433	1	287	325	433333572	1
16	499999869	0	499999869		360	640	225	217	466667825	0	508	21	466667974	1
17	533332110	0	533332110		360	640	279	22	500001314	1	72	291	500002484	1
18	566665655	0	566665655		360	640	624	165	533336333	0	497	209	533338523	1
19	599999250	0	599999250		360	640	359	145	566669137	1	79	105	566670568	1
20	633331925	0	633331925		360	640	46	167	600004418	0	240	116	600004637	1
21	666666054	0	666666054		360	640	606	106	633335199	1	633	151	633338243	1
22	699999481	0	699999481		360	640	150	67	666667843	1	587	153	666671989	1
23	733333276	0	733333276		360	640	417	301	700011499	1	426	269	700014341	1
24	766665295	0	766665295		360	640	167	183	733334185	1	533	231	733343458	1
25	799999584	0	799999584		360	640	485	212	766672478	1	175	206	766674747	0
26	833333145	0	833333145		360	640	481	170	800001407	1	100	343	800009349	0
27	866666374	0	866666374		360	640	153	346	833333352	0	452	310	833334923	0
28	899999861	0	899999861		360	640	457	268	866669467	1	202	40	866674872	0
29	933331518	0	933331518		360	640	354	97	900009191	0	96	136	900011797	0
30	966666075	0	966666075		360	640	481	338	933334939	0	175	72	933337663	0
31	999997408	0	999997408		360	640	453	179	966667790	0	117	172	966669992	1
32	1033333101	0	1033333101		360	640	568	248	1000001415	0	49	314	1000010883	1
33	1066666494	0	1066666494		360	640	232	280	1033337624	0	466	323	1033344345	1
34	1099999773	0	1099999773		360	640	479	333	1066668797	1	55	342	1066671955	0
35	1133333138	0	1133333138		360	640	165	78	1100001835	1	0	248	1100006521	0
36	1166666281	0	1166666281		360	640	508	232	1133333480	0	477	239	1133343641	1
37	1199999354	0	1199999354		360	640	550	324	1166667493	1	395	200	1166671719	0
38	1233333207	0	1233333207		360	640	385	251	1200000520	1	615	248	1200006465	0
39	1266666372	0	1266666372		360	640	5	10	1233339221	0	570	120	1233341477	0
40	1299999451	0	1299999451		360	640	586	98	1266671448	0	586	358	1266671944	1
41	1333332388	0	1333332388		360	640	294	310	1300001422	0	7	42	1300003795	0
42	1366666307	0	1366666307		360	640	97	74	1333336201	0	318	23	1333337661	1
43	1399999078	0	1399999078		360	640	99	261	1366671508	1	292	162	1366671740	1
44	1433332141	0	1433332141		360	640	291	83	1400000466	0	464	102	1400003499	1
45	1466666292	0	1466666292		360	640	508	252	1433336475	0	69	171	1433339538	0
46	1499999899	0	1499999899		360	640	282	188	1466672583	0	309	214	1466673957	0
47	1533333166	0	1533333166		360	640	561	114	1500002656	1	387	81	1500004963	0
48	1566664671	0	1566664671		360	640	138	74	1533335069	1	409	215	1533335542	0
49	1599999892	0	1599999892		360	640	428	68	1566671047	0	523	273	1566672182	0
50	1633333141	0	1633333141		360	640	486	312	1600002705	0	87	35	1600006376	0
51	1666663398	0	1666663398		360	640	156	352	1633335930	0	164	25	1633338346	0
52	1699999023	0	1699999023		360	640	326	47	1666666830	0	289	174	1666667638	0
53	1733333078	0	1733333078		360	640	249	245	1700000338	0	147	160	1700001370	0
54	1766666381	0	1766666381		360	640	445	126	1733335021	0	50	336	1733335147	0
55	1799999646	0	1799999646		360	640	478	65	1766666750	0	462	306	1766669343	0

Figura 30: Screenshot del file out.bag

Attenzione: il file *out.csv* risulterà molto grande (avendo tagliato il video campione, in questo caso sarà grande 1,6Gb circa) quindi LiberOfficer non lo aprirà completamente, ma solo in parte).

Le colonne su cui si lavorerà per la stesura dell'algoritmo sono:

- field.height, fa riferimento alla larghezza della camera;
- field.width, fa riferimento alla lunghezza della camera;
- fields.eventsN.x/y, fanno riferimento all'N-esimo evento che si verifica sul pixel di coordinate *(x,y);*
- field.eventsN.ts, fa riferimento all'istante di tempo, espresso in nanosecondi, in cui si verifica l'N-esimo evento;
- field.eventsN.polarity, fa riferimento alla polarità dell'N-esimo evento: 1 se il pixel percepisce un incremento di luminosità, 0 se percepisce un decremento.

Per ottenere il tracking di ostacoli, è necessario suddividere la simulazione di eventi in blocchi: ogni blocco farà riferimento ad un range di tempo arbitrario e

racchiuderà la matrice dove verranno inseriti i valori che fanno riferimento alla colonna field.eventsN.polarity.
L'algoritmo è stato pensato e sviluppato con l'editor PyCharm per rendere la sua stesura più veloce. Di seguito la stesura del codice:

```
import csv
import sys
import numpy as np
import matplotlib.pyplot as plt

maxInt = sys.maxsize
while True:
    try:
        csv.field_size_limit(maxInt)
        break
    except OverflowError:
        maxInt = int(maxInt / 10)

with open("/home/md/Documents/file.csv") as file_csv:
    reader = csv.reader(file_csv, delimiter=",")
    ncol = len(next(reader))
    file_csv.seek(0)
    pixels = np.zeros((360, 640))
    blocks = []
    i = 0
    blocksize = 200000000
    cond1 = 0
    cond2 = blocksize
    for riga in reader:
        i = i + 1
        if i == 1:
            continue
        for i in range(ncol):
            events = [(riga[6 + 4 * i]), (riga[7 + 4 * i]), (riga[8 + 4 * i]),
(riga[9 + 4 * i])]
            if cond1 < int(events[2]) <= cond2:
                if int(events[3]) == 1:
                    pixels[int(riga[7 + 4 * i]), int(riga[6 + 4 * i])] += 1
                else:
```

```
                pixels[int(riga[7 + 4 * i]), int(riga[6 + 4 * i])] -= 1
            else:
                blocks.append(pixels)
                pixels = np.zeros((360, 640))
                cond1 += blocksize
                cond2 += blocksize

plt.imshow(blocks[5])
plt.show()
```

Vengono inizialmente importate le librerie utili al lavoro svolto, quali *csv*, *sys*, *numpy* e *matplotlib*.

Dato che il file è troppo grande, si potrebbero avere problemi con l'apertura dello stesso con il comando *open()* della libreria csv, quindi si è utilizzata questa porzione di codice per evitare questo sgradevole inconveniente:

```
maxInt = sys.maxsize
while True:
    try:
        csv.field_size_limit(maxInt)
        break
    except OverflowError:
        maxInt = int(maxInt / 10)
```

Si apre il file attraverso la funzione open() della libreria csv

```
with open("/home/md/Documents/file.csv") as file_csv:
    reader = csv.reader(file_csv, delimiter=",")
    ncol = len(next(reader))
    file_csv.seek(0)
```

il reader legge il file .csv indicato e successivamente viene calcolato il numero di colonne che esso possiede ed il seguente valore viene salvato con la variabile ncol.

Poiché l'output di una camera è un'immagine contenente un determinato numero di pixel, la si può vedere come una matrice *n*x*m*, in questo caso 640x360, in cui verrà salvata la polarità dell'evento ogni volta che quest'ultimo

si manifesta, in modo tale da ottenere come risultato finale un flusso di eventi in un determinato lasso di tempo molto piccolo.

Per prima cosa si crea suddetta matrice attraverso la libreria *numpy* con *pixels = np.zeros((360, 640))* e si crea una lista *blocks* dove verranno salvate le matrici per ogni intervallo di tempo scelto.

Iterando sul reader, si salta la prima riga, chiamata anche *header*, dove sono scritti i nomi delle colonne a cui si riferiscono

```
for riga in reader:
        i = i + 1
        if i == 1:
            continue
```

Poi si itera sul numero di colonne:

```
for i in range(ncol):
            events = [(riga[6 + 4 * i]), (riga[7 + 4 * i]), (riga[8 + 4 * i]),
(riga[9 + 4 * i])]
```

in modo tale da prendere in considerazione le colonne interessate, in cui:

- riga[6 + 4 * i] ≡ fields.eventsN.x
- riga[7 + 4 * i] ≡ fields.eventsN.y
- riga[8 + 4 * i] ≡ fields.eventsN.ts
- riga[9 + 4 * i] ≡ fields.eventsN.polarity

Poi si impone la condizione sul tempo, ovvero:

```
            if cond1 < int(events[2]) <= cond2:
                if int(events[3]) == 1:
                    pixels[int(riga[7 + 4 * i]), int(riga[6 + 4 * i])] += 1
                else:
                    pixels[int(riga[7 + 4 * i]), int(riga[6 + 4 * i])] -= 1
```

cond1 e cond2 sono le condizioni inizializzate precedentemente: cond1 ha valore 0, cond2 è arbitrario, a seconda di come si vuole

sl’intervallo di tempo.
Se il valore intero di events[3], ovvero gli elementi di riga[8 + 4 * i] è 1, cioè se avviene un incremento di luminosità, allora si accede alla posizione della matrice attraverso (riga[7 + 4 * i], int(riga[6 + 4 * i]) e si aggiunge il valore 1, altrimenti, se avviene un decremento di luminosità, si aggiunge -1. In questo modo, quando la condizione sul valore di events[2] non è più rispettata:

```
else:
    blocks.append(pixels)
    pixels = np.zeros((360, 640))
    cond1 += blocksize
    cond2 += blocksize
```

si aggiunge la matrice alla lista blocks, si azzera la matrice pixels e vengono incrementate le condizioni in modo tale da continuare a suddividere le simulazioni di eventi in blocchi in un intervallo di tempo stabilito.
Per visualizzare una matrice in un certo istante di tempo, basta utilizzare:

```
plt.imshow(blocks[5])
plt.show()
```

come visto nella figura.

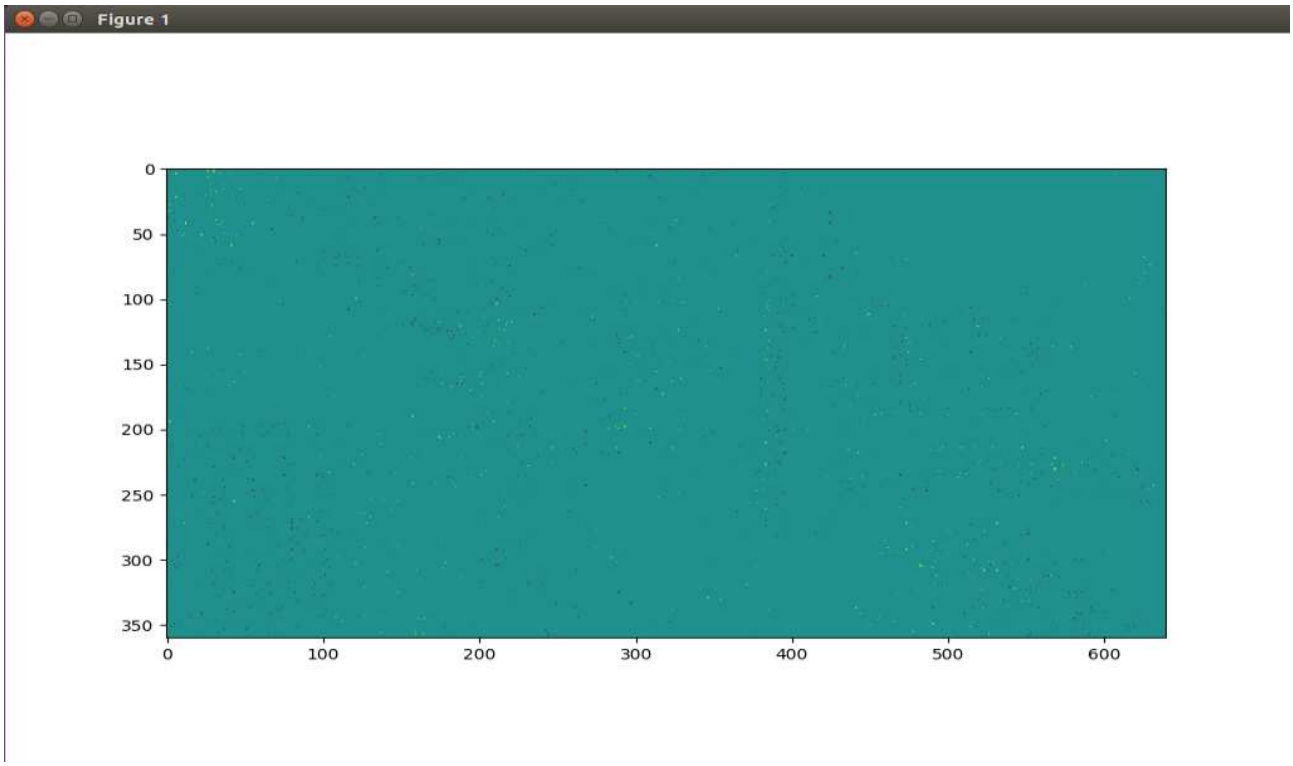


Figura 31: Screenshot del blocco 5 di eventi simulati

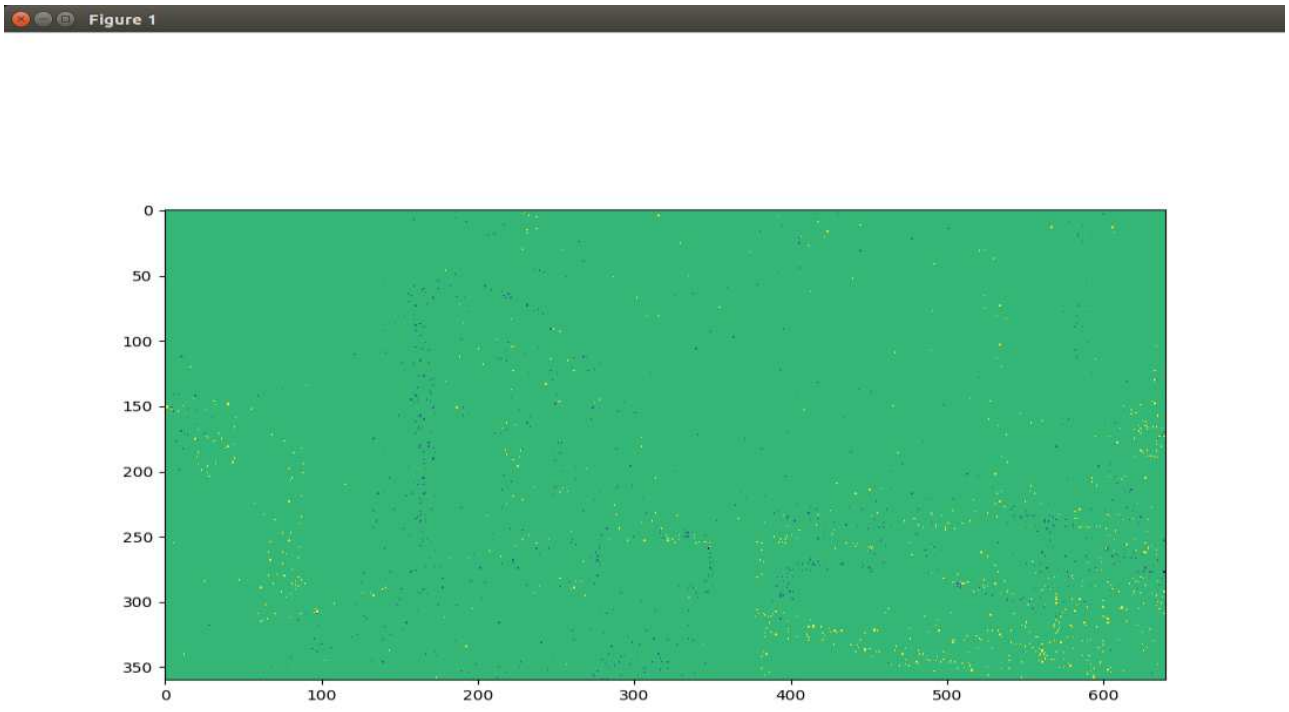


Figura 32: Screenshot del blocco 50 di eventi simulati

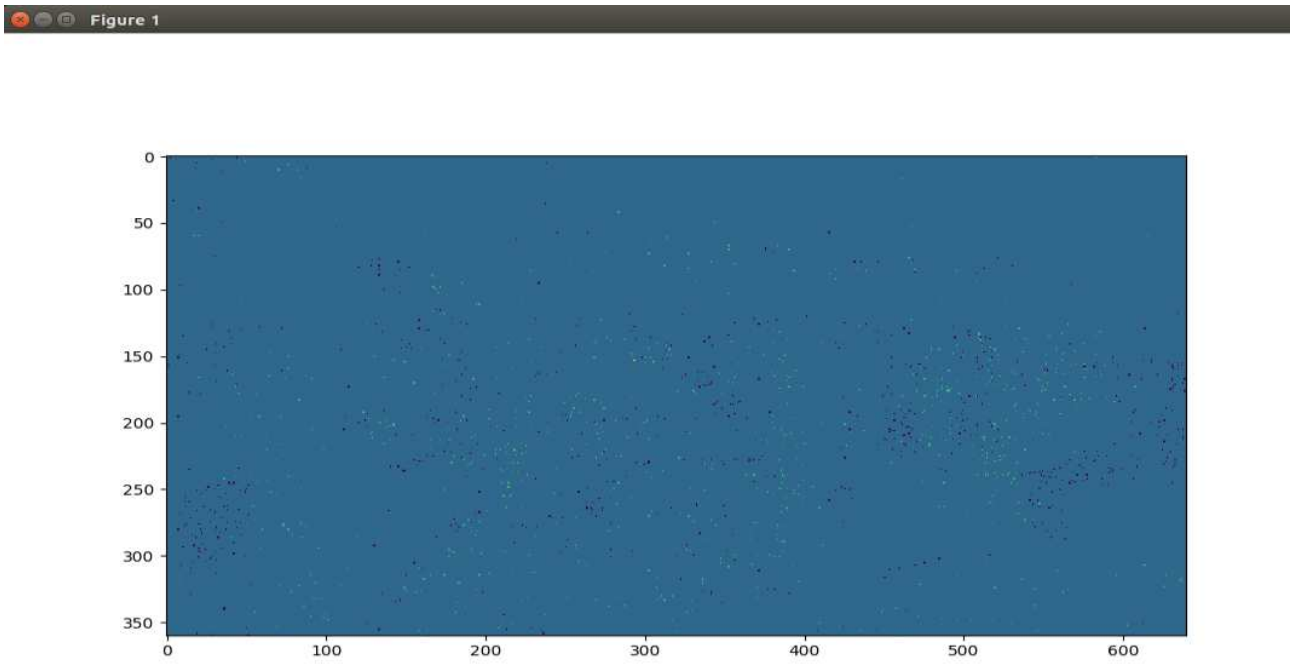


Figura 33: Screenshot del blocco 100 di eventi simulati

CONCLUSIONI

Le telecamere ad eventi sono sensori rivoluzionari che offrono molti vantaggi rispetto alle telecamere tradizionali basate su frame, come bassa latenza, bassa potenza, alta velocità e alta gamma dinamica. Pertanto, hanno un grande potenziale per la visione artificiale e le applicazioni robotiche in scenari difficili attualmente inaccessibili alle telecamere tradizionali. In questa tesi si è fornita una panoramica sul nuovo mondo degli eventi, descrivendo le camere ad eventi, gli strumenti utili per permettere una simulazione attraverso l'ESIM, con un algoritmo sperimentale per sbloccare le loro eccezionali proprietà in applicazioni selezionate.

Eseguendo i passi fatti finora e osservando attentamente i risultati ottenuti, le simulazioni dell'ESIM sono andate a buon fine e hanno dimostrato la potenzialità che questo simulatore offre in termini di:

- **affidabilità:** il simulatore è molto stabile e non presenta particolari problemi nel suo utilizzo;
- **leggerezza**: ESIM si è dimostrato incredibilmente fluido ed efficiente durante le operazioni, utilizzando le risorse dell'elaboratore in modo intelligente e limitato;
- **efficacia**: la simulazione si avvicina molto al funzionamento reale delle camere ad eventi, non solo in termini di verosimiglianza nell'output video, ma anche nella quantità di impostazioni disponibili (IMU, traiettorie della camera, calibrazione ecc.) che la rendono molto utile nello sviluppo di algoritmi;
- **documentazione**: la quantità e la qualità della documentazione disponibile rende l'ESIM facile e intuitivo nell'utilizzo pratico.

Grazie alla comprensione alquanto interessante e stimolante degli argomenti tratti finora e all'utilizzo del simulatore, si riesce inoltre a lavorare molto bene all'interno del mondo delle camere ad eventi.
In particolare l'obiettivo della tesi era il tracking di ostacoli utilizzando appunto un simulatore di eventi in mancanza della camera ad eventi.
Il codice permette la suddivisione in blocchi, relativi ad un certo intervallo di tempo, del flusso di eventi generato dal simulatore. Questo risultato si mostrerà utile per il tracking degli oggetti combinando l'algoritmo spiegato nel capitolo precedente con YOLOv3 che permette la rilevazione di oggetti attraverso le bounding boxes fornendo la classe dell'oggetto rilevato e la confidenza. Combinando frame ed eventi sarà dunque possibile effettuare un tracking robusto ed a alta velocità.

RINGRAZIAMENTI

Mai, e sottolineo mai, avrei pensato di arrivare a scrivere i ringraziamenti alla fine della mia tesi di laurea. Anzi, mai avrei pensato di laurearmi.
Ho sognato per tanto tempo questo momento: dopo tante gioie, dolori, problemi e tanta ma tanta garra posso dire anche io che ce l'ho fatta.
Devo essere sincero, non so quante persone dovrei ringraziare: in quattro anni di università ho conosciuto tantissime persone con cui ho condiviso tante cose nel bene o nel male. Tutto ciò che mi è capitato l'ho trasformato in esperienze positive, credo che anche questo mi abbia fatto arrivare alla fine di questo percorso.
Innanzitutto, vorrei ringraziare il mio Relatore, il Professor Adriano Mancini, che mi ha permesso di svolgere questo progetto. È stato molto stimolante per me conoscere questo nuovo argomento e non la ringrazierò mai abbastanza per l'opportunità. Lei è veramente un grande Professore.
Vorrei ringraziare i miei due gruppi di amici, la 'Crew dei Cristalli' e 'KK', per essere stati al mio fianco in ogni momento della mia vita.
Non scorderò mai tutte le volte che ci sono divertiti insieme, abbiamo condiviso qualunque cosa e senza di voi non ce l'avrei mai fatta.
Vi voglio troppo bene ragazzi.
Volevo ringraziare i miei amici dell'Università, Antonio, Giulia, Paolo, Tommaso, Giacomo, Francesco e Carlo. Siete delle persone fantastiche, studiare con voi è stato bellissimo e vi ringrazio per avermi aiutato a crescere come studente e ampliato il mio bagaglio culturale.
Un ringraziamento speciale va ad Antonio, Paolo e Giulia che sono state le persone che mi sono state più vicino in questi 4 anni di Università, spero che le nostre strade si incoceranno di nuovo in futuro, che il vento vi sia sempre in poppa.
Ovviamente non dimenticherò di ringraziare i miei coinquilini Matteo, Luigi e

Luca ‘il ninja’. Grazie per aver dato un po' di colore alla mia vita universitaria, siete stati i coinquilini migliori che una persona potesse desiderare.
Ultimi ringraziamenti, ma non per importanza, vanno alle mie due famiglie.
Non riesco a esprimere la gratitudine che ho nei vostri confronti per avermi sopportato e supportato in ogni situazione, per avermi fatto crescere come uomo. Grazie per avermi regalato questa esperienza favolosa che ho vissuto, tutti i sacrifici che avete fatto per me verranno ripagati.
Ora che questo ciclo si è finalmente concluso, ne aprirò un altro, più grande e sicuramente più bello del precedente.
Questa tesi la dedico ai miei fratelli Luca, Riccardo, Pietro e Tommaso:
siete la mia vita.

BIBLIOGRAFIA

[1] User Guide -DVS128 Dynamic Vision Sensor

[2] Event-based Vision: A Survey Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, ¨ Stefan Leutenegger, Andrew J. Davison, Jorg Conradt, Kostas Daniilidis, Davide Scaramuzza

[3] KIM, Hanme; LEUTENEGGER, Stefan; DAVISON, Andrew J. Real-time 3D reconstruction and 6-DoF tracking with an event camera. In: European Conference on Computer Vision. Springer, Cham, 2016. p. 349-364. [4] Inivitation https://inivation.com/

[5]Robot Operating System (ROS) https://www.ros.org/

[6] ETH Zurigo - Programming for Robotics - https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/ROS2019/ROS%20Course%20Slides%20Course%201.pdf

[7] –Python (https://www.python.org/)

[8] –Ffmpeg (https://ffmpeg.org/)

[9] ESIM: an Open Event Camera Simulator Henri Rebecq, Daniel Gehrig, Davide Scaramuzza Robotics and Perception Group Depts. Informatics and Neuroinformatics University of Zurich and ETH Zurich

[10] Event Camera - https://en.wikipedia.org/wiki/Event_camera

[11] YOLO: Real-Time Object Detection https://arxiv.org/abs/1804.02767

[12] Spindox – YOLO, un algoritmo ultraveloce open source per la computer vision in tempo reale - https://www.spindox.it/it/blog/yolo-riconoscimento-oggetti-real-time/