Relazione progetto di "Image and Video Analisys"

Luca Angioloni

Università degli Studi di Firenze
lucaangioloni@gmail.com

Sommario

Il progetto si pone come obiettivo di verificare il livello di precisione nella ricostruzione di traiettorie 3D di oggetti osservati in ambito automotive, utilizzando immagini provenienti da una singola camera e metadati estratti da un sistema GPS montati su un' auto, utilizzando una detection basata su due o più diverse CNN.

I. Introduzione

Ad oggi un sistema di guida autonoma evoluto è in grado di accorgersi dell'imminente impatto con un oggetto (pedone o altro veicolo) ed in conseguenza a ciò mette in atto una azione di frenata.

Lo scenario che si immagina per il futuro è diverso: un sistema non solo dovrà essere in grado di accorgersi del rischio di impatto, ma dovrà effettuare una predizione sulla traiettoria futura di ciascun "oggetto" osservato ed in base a questa pianificare una traiettoria di sicurezza che consenta di evitare la collisione non semplicemente frenando, ma anche attraverso un cambio di traiettoria.

Osservando una scena con una camera calibrata posta su una macchina e sfruttando delle assunzioni sul moto degli oggetti nel piano stradale è possibile effettuare una *detection* degli oggetti nella scena e ricostruirne la posizione in un sistema 3D.

Una volta che saranno ottenute grandi quantità di traiettorie ricostruite per classe di oggetti sarà possibile stimare, attraverso vari approcci (Machine Learning e non), la traiettoria futura di altri oggetti delle medesime classi.

II. Metodi e Obiettivi

Per lo sviluppo del progetto, è stato utilizzato il *dataset* messo a disposizione dal **Karlsruhe Institute of Technology** (KIT) per il **KITTI Vision Benchmark**[1].

Il *dataset* è composto da immagini catturate guidando attraverso la città di medie dimensioni *Karlsruhe*, in aree rurali e in autostrada. In ogni immagine sono visibili fino a 15 auto e 30 pedoni.

Le immagini ed i relativi metadati vengono raccolti attraverso un' auto station-wagon equipaggiata con diverse videocamere (due a colori e una in bianco e nero), un sistema di localizzazione GPS ed uno scanner laser Velodyne. (Vedi Figura 1)

Obiettivo del progetto sarebbe quello di verificare il livello di precisione nella ricostruzione della traiettoria 3D degli oggetti osservati utilizzando immagini provenienti da una singola camera, i metadati GPS e una *detection* basata su due diverse CNN[2] (da confrontare).

Si vogliono esprimere le traiettorie degli oggetti nel sistema di riferimento fissato con centro $O(0, 0)$ nel punto di partenza della mac-

[1] http://www.cvlibs.net/datasets/kitti/
[2] Convolutional Neural Network

Figura 1: *Immagine dell'auto equipaggiata con i sensori*

china, orientato con l'asse y rivolta nel verso dell'andamento della macchina.

Si effettua la detection utilizzando delle Reti Neurali Convolutive che ci forniscono una stima della posizione degli oggetti all'interno del frame ed una loro classificazione. Attraverso queste informazioni è possibile calcolare la posizione assoluta degli oggetti nel sistema di riferimento scelto.

Infine è necessario identificare gli stessi oggetti all'interno dei vari frame per poter poi generare le rispettive traiettorie a partire dalle posizioni stimate nei singoli frame.

III. Sviluppo

Il codice sviluppato è composto dai seguenti moduli:

- Estrazione informazioni dal dataset
- Detection attraverso rete neurale
- Calcolo posizione relativa all'auto al tempo corrente
- Cambio sistema di coordinate per esprimere la posizione secondo il sistema scelto
- Riconoscimento degli oggetti tra i vari frame per la ricostruzione della traiettoria

i. Estrazione delle informazioni dal Dataset

Il dataset è composto da diversi gruppi di immagini in successione, ognuno indipendente dall'altro, catturati durante un breve tragitto. Ogni gruppo può rappresentare diverse condizioni stradali come: città, campagna o autostrada; trafficate e non.

Per ogni gruppo di immagini è fornita una cartella contenente:

- Le immagini catturate dalle tre camere (due a colori e una in bianco e nero).
- I time-stamps relativi all'acquisizione di ogni immagine.
- I dati GPS (oxts) il cui formato verrà discusso in seguito.
- I time-stamps relativi all'acquisizione dei dati GPS.
- Output raw dello scanner laser Velodyne.

I dati GPS forniti contengono le seguenti informazioni nel seguente formato:

```
lat:   latitude of the oxts-unit (deg)
lon:   longitude of the oxts-unit (deg)
alt:   altitude of the oxts-unit (m)
roll:  roll angle (rad)
pitch: pitch angle (rad)
yaw:   heading (rad)
...
...
```

Latitudine e longitudine sono espresse in gradi secondo il sistema di coordinate geografico.

Dei tre angoli di rotazione, quello di nostro interesse è *Yaw*; per semplicità gli altri vengono ignorati. L'angolo di rotazione *Yaw* è positivo per rotazioni antiorarie, i valori sono espressi nell'intervallo $(-\pi, \pi]$ e il valore zero (0) si ottiene quando l'auto è ruotata verso Est. (vedi figura 2)

Figura 2: *Informazioni contenute nel dataset: Roll, Pitch e Yaw*

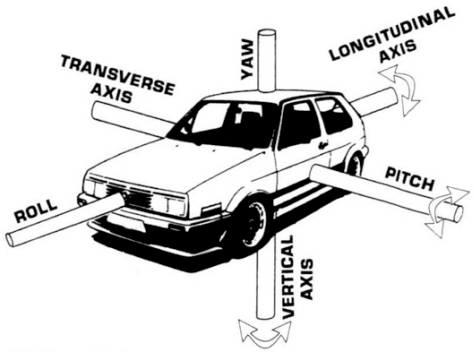
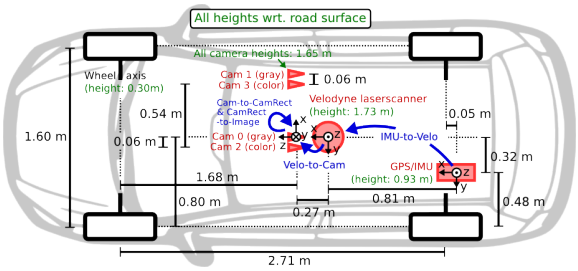


Figura 3: *Disposizione dei sensori sull'auto*



I dati GPS sono acquisiti ad istanti ed intervalli differenti rispetto alle immagini, questi sono stati infatti **interpolati** per ottenere i valori negli istanti di nostro interesse, ovvero gli istanti in cui sono scattate le immagini.

Oltre alle informazioni fornite insieme alle immagini sono disponibili altri dati che riguardano la disposizione dei sensori sull'auto (vedi figura 3) ed i sensori stessi. Queste informazioni saranno fondamentali nelle fasi successive.

Le informazioni riportate riguardanti il dataset sono disponibili sia sul sito internet del *KITTI Vision Benchmark*, sia nell'articolo [1].

ii. Detection

La *detection* degli oggetti all'interno dei frames viene effettuata da un modulo basato su **Reti Neurali Convolutive** dette anche *ConvNets* or *CNNs*. Queste sono una categoria di Reti Neurali che si sono dimostrate molto efficaci in aree come il riconoscimento e la classificazione di immagini.

I due approcci proposti sono i seguenti:

- **YOLO** (You Only Look Once - v2)
- **SSD** (Single Shot MultiBox Detector)

In entrambi gli approcci è stato utilizzato per l'addestramento un dataset conosciuto come **COCO** (Common Objects in Context)[3] reso disponibile da *Microsoft*.

Le reti non sono state addestrate per questo progetto ma si sono utilizzati, per entrambe, i pesi disponibili online di reti pre-addestrate per ovvi motivi di tempo e qualitativi.

Dato l'utilizzo del dataset *COCO*, le reti sono in grado di individuare oggetti appartenenti alle classi di oggetti contenuti nel dataset. Sarebbe possibile una fase detta di *finetuning* con la quale perfezionare e personalizzare, a seconda delle necessità del progetto, le performance della rete neurale. Non si è seguita questa strada in quanto non è lo scopo del progetto.

Dalla detection vengono quindi generati dei *bounding box* per i differenti oggetti rilevati. Per ogni oggetto inoltre viene fornito lo *score* della predizione (composizione dello score sulla stima della classe e del bounding box) e la classe stimata dell'oggetto.

Queste informazioni vengono sfruttate per effettuare una prima classificazione degli oggetti: vengono scartati quelli di classi non rilevanti per il progetto (ricordiamo che il dataset COCO contiene anche altre classi) e quelli con uno score troppo basso.

ii.1 YOLO

YOLO invece di modificare ed utilizzare la classificazione di oggetti per effettuare detection, vede il sistema nel suo insieme come un singolo problema di regressione, dai pixel dell'immagine alle coordinate dei bounding box e alle probabilità delle classi.

Una singola CNN predice simultaneamente diversi bounding boxes e diverse probabilità di classi per queste finestre.

Visto che l'intera pipeline di detection è una singola rete, può essere ottimizzata direttamente sulla detection.

Questo approccio è estremamente veloce, ma possiede alcune limitazioni per quanto riguarda la precisione raggiungibile. In ogni caso le prestazioni sono ottime.

YOLO non si basa su architetture di CNN conosciute, ne usa una personalizzata. [2]

ii.2 SSD

SSD è un metodo per effettuare *object detection* utilizzando una singola *Deep Neural network*.

[3] http://mscoco.org/dataset/#overview

Questo approccio, discretizza lo spazio di output delle finestre di contorno (*bounding boxes*) in un set di finestre di default a diverse *aspect ratio* e scale per la locazione degli oggetti. A tempo di predizione, la rete genera valutazioni (*scores*) per la presenza di ogni categoria di oggetto in ogni finestra di default e produce aggiustamenti per le finestre, per combaciare meglio con le forme degli oggetti. Inoltre, la rete combina predizioni a differenti risoluzioni per gestire oggetti di varie dimensioni. SSD è molto semplice se confrontato con metodi che richiedono la proposta di regioni, perché elimina completamente il bisogno di queste e i conseguenti stadi di ricampionamento di pixel o mappe di caratteristiche. Inoltre incapsula tutta la computazione in una singola rete. Tutto questo rende SSD semplice da addestrare e immediato da integrare in sistemi che richiedono la detection di oggetti.

SSD utilizza la stessa architettura di VGG Net alla base (precisamente VGG-16). [3]

ii.3 Approcci alternativi

Altri approcci recenti come R-CNN (o nella sua versione veloce Faster R-CNN) sono stati presi in considerazione. R-CNN usa dei metodi di proposta di regione per generare dapprima potenziali bounding boxes in un immagine e poi applica una classificazione all'interno di questi . Dopo la classificazione viene effettuata una fase di post-processing per rifinire i box, eliminare duplicati e rivalutare lo score di un box in base agli altri oggetti nella scena. Questi flussi di lavoro complessi sono lente e difficili da ottimizzare poiché ogni componente individuale deve essere addestrato separatamente.

Infine sono ovviamente possibili anche altri approcci non basati su Reti Neurali ma non sono stati considerati in questo progetto.

iii. Coordinate relative

Una volta trovati i **bounding box** si procede alla stima della posizione dell'oggetto relativamente alla posizione e rotazione della macchina al tempo corrente. (Vedi figura 7)

Si presume che l'oggetto sia a contatto con il terreno, in questo modo è possibile stimarne la posizione con dei calcoli trigonometrici.

Si sceglie il pixel al centro in basso del bounding box come punto per il calcolo della posizione dell'oggetto.

Con il dataset sono fornite anche delle informazioni riguardo al sensore della camera, l'ottica utilizzata e il posizionamento della camera sull'auto (vedi [1]).

La posizione relativa viene espressa nel sistema di coordinate (x_k, y_k) relativo alla posizione e rotazione dell'auto all'istante k (vedi figura 7) e viene così calcolata:

1. Conoscendo la posizione (x, y) del pixel in basso al centro del bounding box, si calcola il displacement rispetto al centro dell'immagine (Vedi figura 4). Si considerano solo oggetti nella metà inferiore dell'immagine.

2. Conoscendo quindi il displacement (in pixel) sull'asse x e y dell'immagine si possono stimare gli angoli α e β sull'immagine della prospettiva dal punto focale. (Vedi figura 5 e 6)

3. Si calcola quindi (con dei semplici calcoli trigonometrici) le coordinate relative all'istante k (sull'asse x_k e y_k) come:

$$\hat{y}_k = \frac{f}{dy \cdot p} \cdot c$$
$$\hat{x}_k = \frac{dx \cdot p}{f} \cdot \hat{y}_k$$

   con:

   - $f =$ lunghezza focale
   - $dy =$ displacement sull'asse y
   - $dx =$ displacement sull'asse x
   - $p =$ dimensione di un singolo pixel del sensore
   - $c =$ altezza della camera da terra.

Figura 4: *Displacement dell'oggetto all'interno del frame*

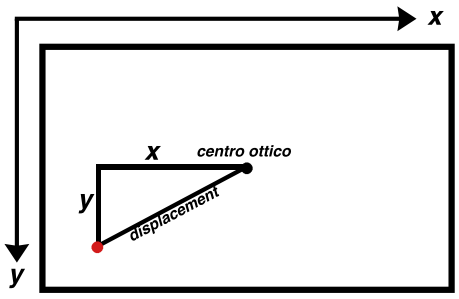


Figura 5: *Calcolo angoli di incidenza sull'immagine (α) e calcolo distanza ($\hat{y}_k$). Vista laterale.*

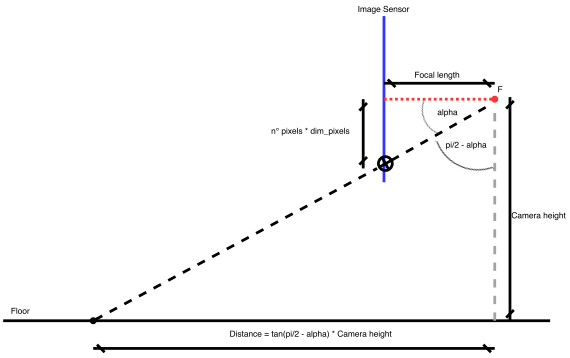
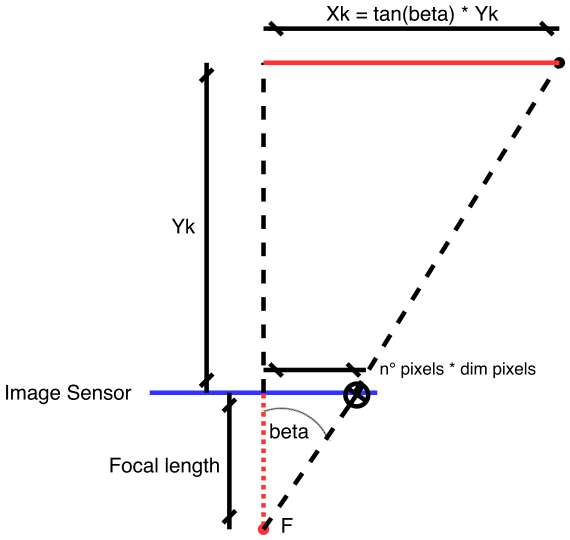


Figura 6: *Calcolo angoli di incidenza sull'immagine (β) e calcolo distanza laterale ($\hat{x}_k$). Vista dall'alto.*



iii.1 Precisazione sui punti 2 e 3

Innanzi tutto è necessario un appunto sui valori delle costanti f, p e c.

L'altezza della camera da terra (c) viene fornita insieme al dataset ed è uguale a $1,65m$ (vedi figura 3).

La dimensione dei pixel è stata ottenuta dal sito internet[4] della casa produttrice dei sensori. Il modello è fornito sia sul sito internet di *KITTI* sia in [1]. Il valore di p è di $4.65\mu m$.

Il valore della lunghezza focale invece è fornito in [1] ed è uguale a $4mm$, ma sembra che il valore non sia sempre costante e non uguale nei diversi gruppi di immagini del dataset, infatti già il sito internet di *KITTI* riporta un dato lievemente differente e propone come valore un range $(4 - 8mm)$.

Questi possibili errori possono generare anche grandi imprecisioni nella stima delle coordinate relative. Quindi è stato anche implementato un sistema alternativo che si basa sugli angoli di visone della camera (anch'essi forniti in [1]).

La posizione $(\hat{x}_k, \hat{y}_k)$ viene quindi calcolata come:

$$\alpha = (\frac{y - \frac{h}{2}}{h} \cdot a_y)$$
$$\beta = \frac{x - \frac{w}{2}}{w} \cdot a_x$$
$$\hat{y}_k = tan(\frac{\pi}{2} - \alpha) \cdot c$$
$$\hat{x}_k = tan(\beta) \cdot \hat{y}_k$$

con:

- α = angolo verticale di visione dal punto focale
- β = angolo orizzontale di visione dal punto focale
- y = coordinata y sull'immagine del punto in basso al centro del bounding box
- x = coordinata x sull'immagine del punto in basso al centro del bounding box
- h = altezza in pixel dell'immagine (375 px)

[4] https://www.ptgrey.com/flea2-14-mp-color-firewire-1394b-sony-icx267-camera

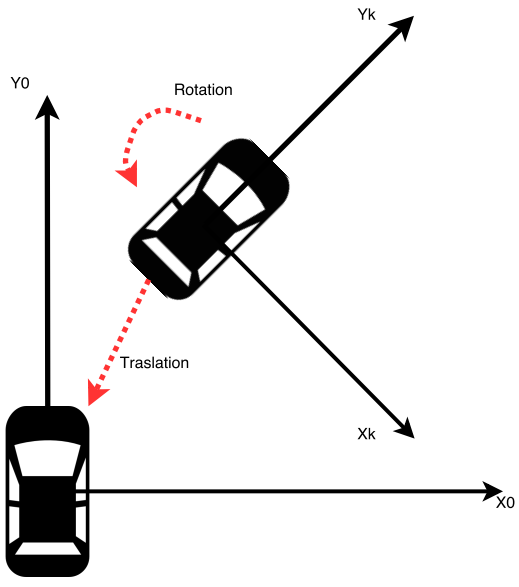
- w = larghezza in pixel dell'immagine (1242 px)
- a_x = angolo di visione totale in orizzontale (90°)
- a_y = angolo di visione totale in verticale (35°)
- c = altezza della camera da terra.

Questo secondo approccio, seppur non perfetto, sembra ottenere risultati migliori. Di fatto, probabilmente, neanche gli angoli di visione forniti sono precisi, ma evidentemente meno sbagliati che degli altri dati.

iv. Trasformazioni affini

Dal sistema di coordinate relative alla macchina nello specifico istante di tempo, vogliamo esprimere le posizioni degli oggetti nel sistema di riferimento scelto ovvero con $O(0,0)$ nella posizione di partenza della macchina con l'asse y rivolto verso la direzione della macchina all'istante iniziale, chiamato sistema (x_0, y_0). (Vedi figura 7)

Figura 7: *Trasformazioni per passare dal sistema di coordinate relative (x_k, y_k) al sistema di coordinate selezionato (x_0, y_0)*



Per fare questo utilizziamo due trasformazioni affini: una di Rotazione ed una di Traslazione.

Le coordinate relative all'istante k espresse secondo (x_0, y_0) saranno quindi date da:

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = TR \begin{bmatrix} x_k \\ y_k \\ 1 \end{bmatrix}$$

Dove T ed R sono rispettivamente delle matrici di Traslazione e di Rotazione.

iv.1 Rotazione

La rotazione viene effettuata in funzione della differenza tra gli angoli all'istante k (θ_k) e all'istante iniziale (θ_0).

La differenza è così calcolata:

$$\delta\theta = \theta_k - \theta_0$$

Segue quindi la definizione della *matrice di rotazione antioraria*:

$$R = \begin{bmatrix} cos(\delta\theta) & -sin(\delta\theta) & 0 \\ sin(\delta\theta) & cos(\delta\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Gli angoli θ considerati sono positivi per rotazioni in senso antiorario e le informazioni vengono estratte dai dati GPS del *dataset*, precisamente dall'angolo di *Yaw* (vedi sottosezione i).

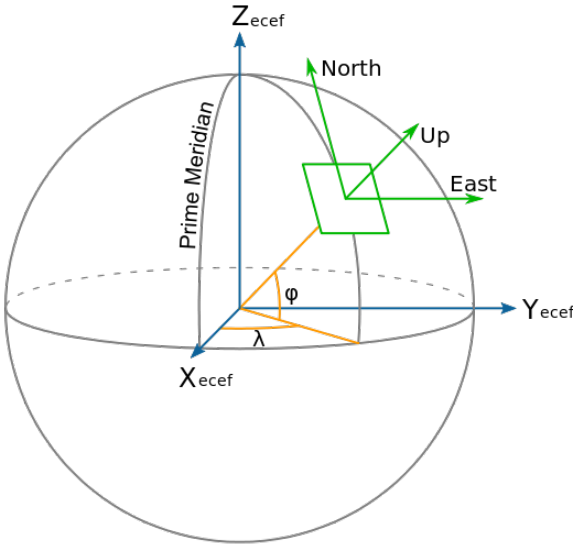
iv.2 Traslazione

La traslazione viene invece calcolata in funzione dello spostamento che la macchina effettua dall'istante iniziale. Ricordiamo che per il nostro sistema di coordinate (x_0, y_0) la macchina all'inizio si trova in posizione $(0,0)$.

Lo spostamento effettuato dalla macchina è calcolato come la differenza delle coordinate GPS della macchina all'istante corrente e di quelle allo stato iniziale, esprimendo anche le coordinate GPS nel sistema di riferimento (x_0, y_0).

Le coordinate GPS sono un sistema di coordinate **geodetico** in cui le posizioni sul geode terrestre sono espresse come angoli al centro

Figura 8: *Sistema di coordinate GPS geodetico (angoli λ e φ) e il sistema di coordinate cartesiano ENU (East-North-Up)*



del geode; Latitudine e Longitudine con gli zeri rispettivamente sull'*equatore* e sul *meridiano di Greenwich*.

É quindi necessaria una trasformazione in un sistema di coordinate cartesiano superficiale con cui poi lavorare per esprimere la posizione dell'auto secondo il sistema (x_0, y_0). É stato scelto il sistema di coordinate **ENU** (East-North-Up) rappresentato in *figura 8* di cui si utilizzano le informazioni *North* e *East*.

La trasformazione viene effettuata utilizzando la concezione di distanza superficiale sul geode terrestre. Nello specifico si è utilizzata la distanza di **Vincenty** descritta in [4] e [5].

Le coordinate della posizione della macchina nel sistema ENU devono ancora essere trasformate, con una rotazione per essere espresse secondo il sistema desiderato. Il sistema ENU (se chiamiamo *North* con y e *East* con x) è orientato con l'asse y verso nord, il sistema (x_0, y_0) è invece orientato con l'asse y_0 verso dove punta la macchina, ossia l'angolo θ_0 che è positivo per rotazioni in senso antiorario e che ha lo zero nella direzione est (vedi iv.1 e i).

Il sistema ENU è quindi in generale ruotato di $\frac{\pi}{2} - \theta_0$, per cui utilizzando la matrice di rotazione R_{GPS} si ottengono le coordinate della posizione della macchina nel sistema di riferimento (x_0, y_0):

$$R_{GPS} = \begin{bmatrix} cos(\frac{\pi}{2} - \theta_0) & -sin(\frac{\pi}{2} - \theta_0) & 0 \\ sin(\frac{\pi}{2} - \theta_0) & cos(\frac{\pi}{2} - \theta_0) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Adesso, è finalmente possibile definire lo spostamento dell'auto e quindi calcolare i parametri della matrice di traslazione come:

$$\delta x = x_k - x_0$$
$$\delta y = y_k - y_0$$

dove x_0 e y_0 sono entrambi 0.

La matrice di traslazione risultante per esprimere nel sistema di coordinate (x_0, y_0) le coordinate relative rilevate all'istante k è data da:

$$T = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix}$$

v. Riconoscimento degli oggetti tra i frames

A questo punto, per ogni frame, abbiamo rilevato un certo numero di oggetti, classificati con una certa classe. Per ognuno di questi abbiamo calcolato una posizione nel sistema di coordinate scelto (x_0, y_0).

Queste informazioni non sono però correlate a quelle degli altri frame in alcun modo. Si rende quindi necessario un sistema per riconoscere gli stessi oggetti attraverso i frames per poter poi tracciare la loro traiettoria nel tempo (come successione delle loro coordinate negli istanti di tempo di acquisizione dei frames).

Si è deciso di implementare un riconoscimento di oggetti basato sul **confronto di istogrammi**:

1. Per ogni oggetto in ogni frame, si calcola l'istogramma (a colori) della regione di immagine all'interno del bounding box.

2. Scorrendo i frame in ordine, per ogni oggetto nel frame, si confronta l'istogramma dell'oggetto con tutti gli istogrammi degli oggetti precedentemente analizzati (nei frame precedenti). Si è deciso, sia per ragioni di costo che per questioni logiche, di limitare il confronto all'indietro con un numero finito di frames precedenti (valore costante deciso a tempo di scrittura).

3. Se non si ottiene nessun match con gli oggetti precedenti, all'oggetto viene assegnato un nuovo *ID*, altrimenti all'oggetto viene assegnato l'*ID* dell'oggetto riscontrato.

4. Alla fine del processo, per ogni frame, ogni oggetto rilevato ha un proprio ID. Selezionando uno specifico ID è possibile tracciare gli spostamenti dell'oggetto nel tempo.

IV. Software Sviluppato

Il software è stato sviluppato in Python (versione 3.x) utilizzando come base per l'implementazioni delle reti neurali le seguenti repositories di *GitHub*:

- YOLO: YAD2K - Yet Another Darknet 2 Keras
- SSD: Single Shot MultiBox Detector

Per entrambe si utilizza un interfaccia basata su *Keras*, con backend *TensorFlow*.

I pesi per le reti sono forniti con il codice, altrimenti sono disponibili in rete.

Le dipendenze del progetto sono i seguenti pacchetti (moduli) Python:

- Pillow
- Numpy
- GeoPy
- Keras
- TensorFlow

Il codice è organizzato in due cartelle, una per tipo di Rete Neurale, per mantenere separati ed organizzati i componenti necessari al funzionamento delle stesse. Si utilizzano quindi due script principali chiamati *YOLOVideo.py* e *SSDVideo.py* collocati nelle rispettive cartelle. Il dataset, anche solo una parte, può essere posizionato a piacere; infatti la cartella di input per la sequenza di immagini a colori viene passata come argomento agli script.

I parametri degli script sono i seguenti:

- *model_path*: percorso del file h5 o hdf5 contenente il corpo del modello (i pesi) (opzionale, di default utilizza i pesi forniti con il codice)
- *input_path*: percorso della cartella contenente le immagini di input. La cartella di input deve essere una cartella che contiene in maniera sequenziale una successione di immagini corrispondenti ai frame video del dataset.
- *output_path*: percorso della cartella di output (immagini, log, traiettorie)
- *oxts*: percorso della cartella oxts contenente i dati GPS del dataset e i time stamps relativi. (opzionale se la cartella di input si trova nella gerarchia di cartelle del dataset originale)
- *time_stamps*: percorso del file contente i time stamps dei frame (immagini di input) (opzionale se la cartella di input si trova nella gerarchia di cartelle del dataset originale)
- *score_threshold*: soglia per gli score dei bounding box, default = 0.3 (Solo per YOLO) (opzionale)
- *iou_threshold*: soglia per la suppressione non massima IOU, default = 0.5 (Solo per YOLO) (opzionale)
- *anchors_path*: percorso al file delle anchors, default = *model_data/yolo_anchors.txt* (Solo per YOLO) (opzionale)
- *classes_path*: percorso al file contenente le classi, default = *model_data/coco_classes.txt* (Solo per YOLO) (opzionale)

Gli script seguono i seguenti passi:

1. Si leggono gli argomenti di input

2. Si leggono le informazioni relative alla Rete Neurale (classi e pesi) e quelli relativi al dataset (timestamps e dati GPS) (vedi sezione i); inoltre si creano le cartelle e file necessari se non esistenti

3. Si crea il modello della Rete Neurale e si inizializza

4. Per ogni immagine del dataset si effettua la detection degli oggetti (vedi la sezione ii)

5. Si calcolano le coordinate e l'istogramma a colori per ogni oggetto (vedi le sezioni iii e iv)

6. Le informazioni ricavate vengono salvate: Immagini di output (con sovraimposti i bounding box), coordinate e istogrammi

7. Infine dalle informazioni ricavate si ottengono le traiettorie effettuando il riconoscimento degli oggetti confrontando gli istogrammi (vedi sezione v)

Infine è stato sviluppato un ultimo script, che permette la visualizzazione grafica delle traiettorie plottando i dati, chiamato *plot.py*. Eseguendolo si vede con lo scorrere del tempo il movimento degli oggetti e le loro traiettorie.

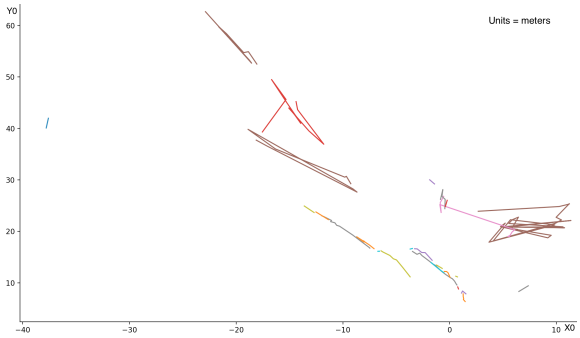
Insieme al codice, comunque, sono forniti dei file README per guidare l'utilizzo.

V. Risultati

I risultati non sono eccellenti e forse non sono quelli sperati ma sono comunque buoni viste le criticità presenti. Le traiettorie generate sono consistenti (ossia un oggetto più distante viene rilevato effettivamente ad una distanza maggiore, oggetti che si muovono da destra a sinistra hanno una traiettoria da destra a sinistra), ma il grado di precisione rilevato non è ottimale ed in generale non adeguato ad una situazione di analisi in tempo reale in cui la sicurezza di persone dipende dalle traiettorie rilevate (vedi figura 10).

Anche il riconoscimento degli oggetti tra i frame non è ottimo e questo è dovuto dalle difficili condizioni del sistema, in cui ogni oggetto durante la traiettoria viene occluso da altri oggetti o dall'ambiente, molti oggetti si sovrappongono anche per molti frame, e molto spesso ci sono numerosi oggetti da seguire (vedi figura 11).

Figura 9: *Traiettorie ricostruite. Generate con YOLO. Ogni colore è un oggetto diverso rilevato.*



Un esempio di traiettorie rilevate può essere visto in figura 9.

Circa gli stessi risultati si ottengono utilizzando entrambi gli approcci con le Reti Neurali Convolutive: YOLO e SSD.

SSD risulta essere di poco più veloce rispetto a YOLO, ma rileva meno oggetti all'interno della scena, a volte anche oggetti di interesse come pedoni o biciclette. In alcune situazioni SSD non riesce a rilevare correttamente nessun oggetto o addirittura rileva oggetti dove non ce ne sono. Nel complesso forse YOLO risulta più stabile nel seguire la traiettoria di un oggetto.

Entrambe non sono sempre consistenti con le classi che assegnano agli oggetti e questo risulta problematico nella fase di riconoscimento per determinare le traiettorie (vedi sottosezione v). Contribuisce inoltre il fatto che i bounding boxes rilevati non mantengono le stesse dimensioni e proporzioni tra un frame e l'altro.

Sebbene si siano scelte varianti di Reti Neurali Convolutive per *object detection* dai costi ridotti e con un numero ridotto di classi di oggetti rilevabili, i risultati dimostrano che sarebbe difficoltosa un esecuzione in sistemi embedded non dedicati alla computer vision. Infatti utilizzando solo la CPU anche un portatile piuttosto potente non riesce ad elaborare un feed video in tempo reale (poco meno di un frame al secondo). Sono necessarie quindi implementazioni che sfruttino la parallelizzazione delle GPU e quindi il sistema elaborante ne dovrà possedere una.

Figura 10: *Dimostrazione della consistenza delle traiettorie rilevate*

Figura 11: *Problemi di riconoscimento dovuti alle sovrapposizioni di oggetti e occlusioni*

Riferimenti bibliografici

[1] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, 2013
Vision meets Robotics: The KITTI Dataset
International Journal of Robotics Research (IJRR)

[2] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, 2015
You Only Look Once: Unified, Real-Time Object Detection
pjreddie.com

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. C. Berg, 2016
SSD: Single Shot MultiBox Detector

[4] Vincenty T., Aprile 1975
Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations
Survey Review. XXIII (176): 88–93

[5] Wikipedia, 2017
Vincenty's formulae
https://en.wikipedia.org/wiki/Vincenty%27s_formulae