

# DAIT Project

## Pedestrian trajectory prediction

---

Rodolphe Farrando, Romain Gratier

23.05.2018

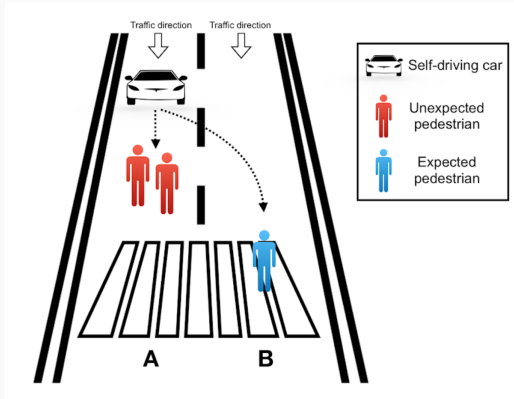
EPFL

## Introduction

---

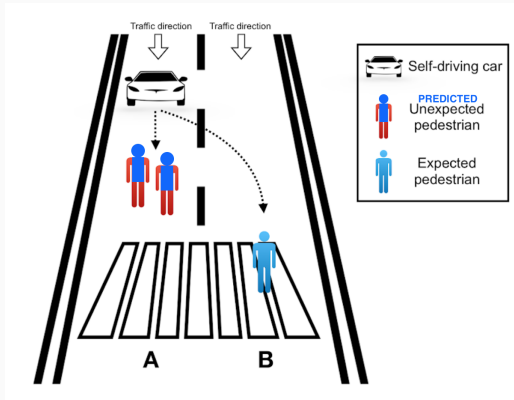
# Introduction

- Trajectory prediction is crucial for improving autonomous vehicles behaviour



# Introduction

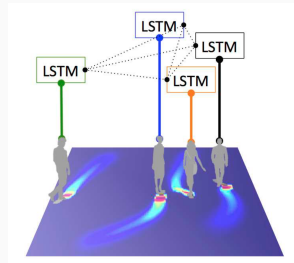
- Trajectory prediction is crucial for improving autonomous vehicles behaviour
- Could avoid situations seen in the ethical lectures



# Previous work Social LSTM : Human Trajectory Prediction in Crowded Spaces

In their project, they used different components to make the structure:

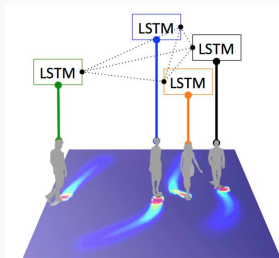
- One LSTM per pedestrian
- Social Pooling
- Prediction per frame



# Previous work Social LSTM : Human Trajectory Prediction in Crowded Spaces

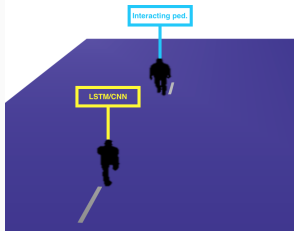
In their project, they used different components to make the structure:

- One LSTM per pedestrian
- Social Pooling
- Prediction per frame



In our project we only use:

- One CNN, or one LSTM
- Prediction per pedestrian



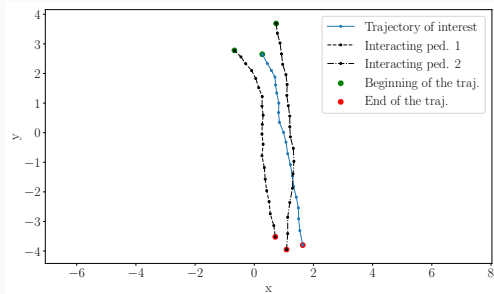
## Data

---

# Pre-processing

The preprocessing is divided in 4 steps:

1. We isolate each trajectory along with his interaction, that is the other trajectories that are around within the same frames

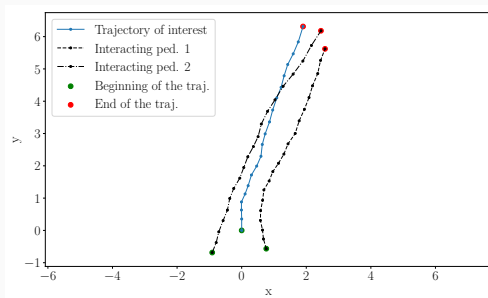




# Pre-processing

The preprocessing is divided in 4 steps:

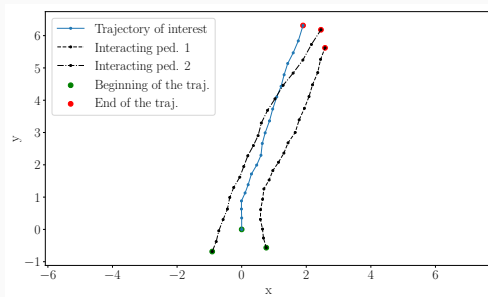
1. We isolate each trajectory along with his interaction, that is the other trajectories that are around within the same frames
2. We normalize the trajectories such that the first point is at  $(0,0)$  and the second is at  $(0, y_1)$



# Pre-processing

The preprocessing is divided in 4 steps:

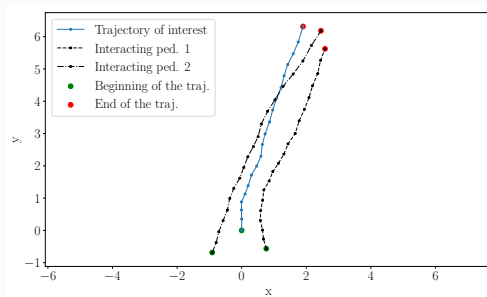
1. We isolate each trajectory along with his interaction, that is the other trajectories that are around within the same frames
2. We normalize the trajectories such that the first point is at  $(0, 0)$  and the second is at  $(0, y_1)$
3. We calculate axis velocities  $V_x$  and  $V_y$



## Pre-processing

The preprocessing is divided in 4 steps:

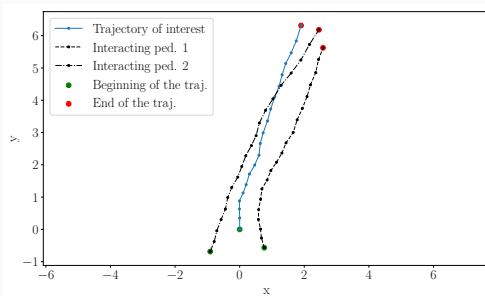
1. We isolate each trajectory along with his interaction, that is the other trajectories that are around within the same frames
2. We normalize the trajectories such that the first point is at  $(0,0)$  and the second is at  $(0, y_1)$
3. We calculate axis velocities  $V_x$  and  $V_y$
4. For each frame, if there is a interacting pedestrian we add its coordinates and speed otherwise we add zeros



## Pre-processing

The preprocessing is divided in 4 steps:

1. We isolate each trajectory along with his interaction, that is the other trajectories that are around within the same frames
2. We normalize the trajectories such that the first point is at  $(0,0)$  and the second is at  $(0, y_1)$
3. We calculate axis velocities  $V_x$  and  $V_y$
4. For each frame, if there is a interacting pedestrian we add its coordinates and speed otherwise we add zeros
5. Data augmentation (flip and add noise) cf. last slide



# Data

We have a file with:

- Pedestrians ID
- Frame number
- Twenty sets of  $x$  and  $y$  coordinates per pedestrian

Frame Number	ID	$x$	$y$	$V_x$	$V_y$
0	$i$	0	0	0	0
10	$i$	0	$y$	0	$V_x$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

# Data

We have a file with:

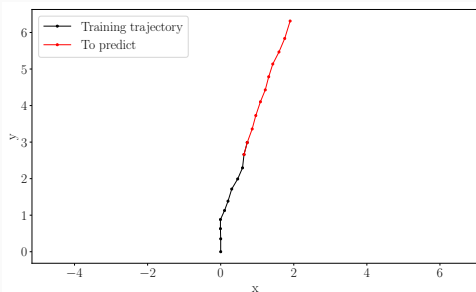
- Pedestrians ID
- Frame number
- Twenty sets of  $x$  and  $y$  coordinates per pedestrian

Frame Number	ID	$x$	$y$	$V_x$	$V_y$
0	i	0	0	0	0
10	i	0	$y$	0	$V_x$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

We want :

- Train on the 10 first coordinates and speed and their interaction
- Predict the next 10
- Inputs have the following shape:  $[10, N, 4 * N_{inter}]$

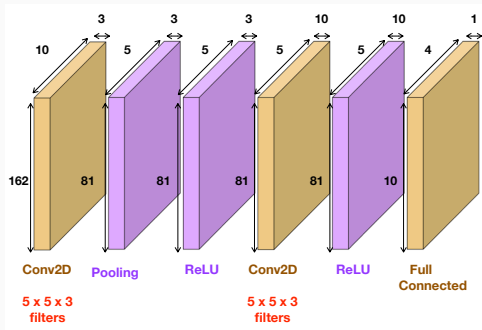
Example of trajectory to predict



## Models

---

# CNN

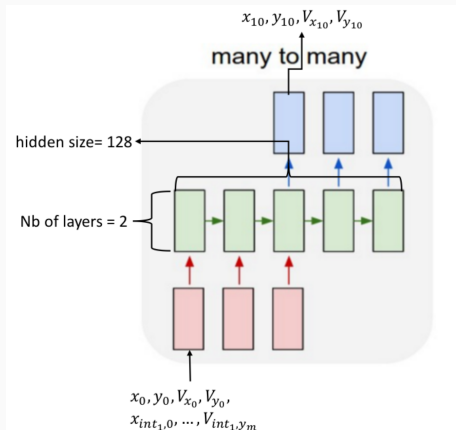


**Inputs:** sequence of coordinates and velocities of the pedestrians

**Outputs:** sequence of predicted coordinates and velocities for a trajectory of interest



# LSTM



**Inputs:** sequence of coordinates and velocities of the trajectory of interest and of the interacting trajectories

**Outputs:** sequence of predicted coordinates and velocities for the trajectory of interest

## Results

---

To calculate the correctness of the prediction two indicators are used:

1. The final displacement error:  $e_{fin} = \sqrt{(X_n - X_{pred,n})^2}$
2. The mean displacement error:  $e_{fin} = \sqrt{\frac{\sum_{i=0}^n (X_{gt,i} - X_{pred,i})^2}{(n)}}$

Depending on the inputs two ways are possible to find the predicted coordinates:

1. If the coordinates are predicted: directly use them
2. If the velocities are predicted:  $X_t = X_{t-1} + V_t \cdot 0.4$ , with 0.4 the time between two frames in seconds

## Results: Introduction

Four different losses are tested for each model:

1. Predict coordinates with loss defines as  $L_1 = (X - X_{pred})^2$  with  $X = [x, y]$
2. Predict speeds with loss defines as  $L_2 = (V - V_{pred})^2$  with  $V = [V_x, V_y]$
3. Predict both coordinates and speeds with loss defines as  $L = L_1 + L_2$
4. Predict both coordinates and speeds with loss defines as  $L = L_1 + L_2 + L_3$ ,  
with  $L_3 = (X - X_{t-1} + V_t * 0.4)^2$

The fourth case ensure that coordinates and speeds are not predicted independently.

Each case is tested on three trajectory types:

1. Static
2. Linear trajectories
3. Non-linear trajectories

# Results

Linear prediction results:

- Type 1: *Mean* = 0.141, *Final* = 0.322
- Type 2: *Mean* = 0.541, *Final* = 0.93
- Type 3: *Mean* = 0.651, *Final* = 1.457
- Total: *Mean* = 0.512, *Final* = 0.982

		CNN				LSTM			
		Type 1	Type 2	Type 3	Total	Type 1	Type 2	Type 3	Total
Coord.	Mean	4.696	5.144	4.674	4.176	1.309	0.777	0.862	0.877
	Final	10.246	7.009	10.501	5.602	1.385	0.92	1.108	1.037
Speed	Mean	0.567	5.133	1.911	4.17	0.726	0.573	0.651	0.616
	Final	0.77	6.971	3.882	5.587	1.412	1.045	1.231	1.148
2 Loss	Mean	1.269	5.134	1.762	4.163	0.695	0.532	0.627	0.581
	Final	2.727	6.978	3.546	5.57	1.302	0.963	1.2	1.076
3 Loss	Mean	0.549	5.135	3.829	4.163	0.748	0.607	0.681	0.647
	Final	0.758	6.983	4.962	5.573	1.364	1.072	1.308	1.177

LSTM and CNN comparison:

- For both LSTM and CNN, the "2 Loss" model have the better results
- For the CNN, results are much higher  $\Rightarrow$  the absence of memory is really penalising

LSTM and linear prediction comparison:

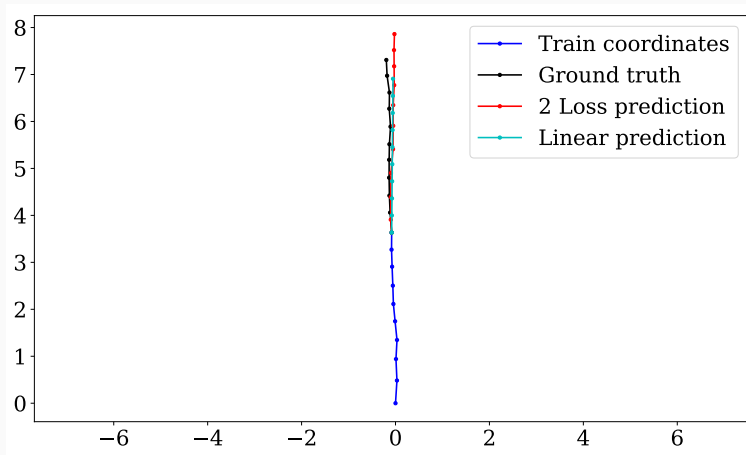
- Except for the first type trajectory, our "2 Loss" LSTM have better results
- Even for static trajectory, the LSTM tries to predict dynamic trajectory
- A few examples will show the difference

## Representation

---

## Representation: a few examples

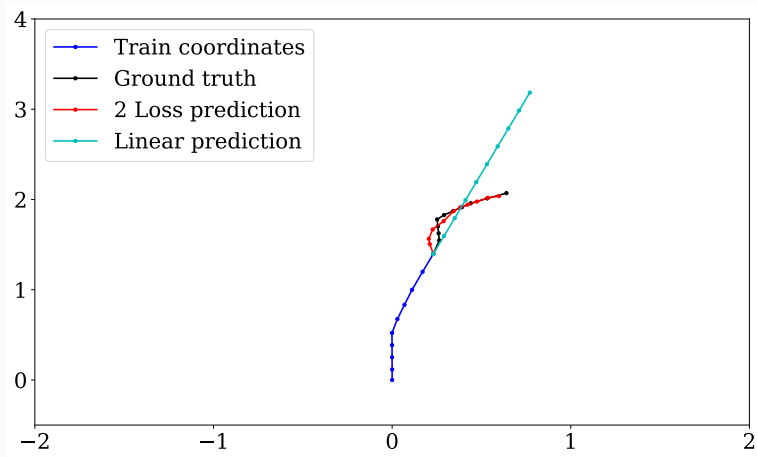
Linear trajectories:





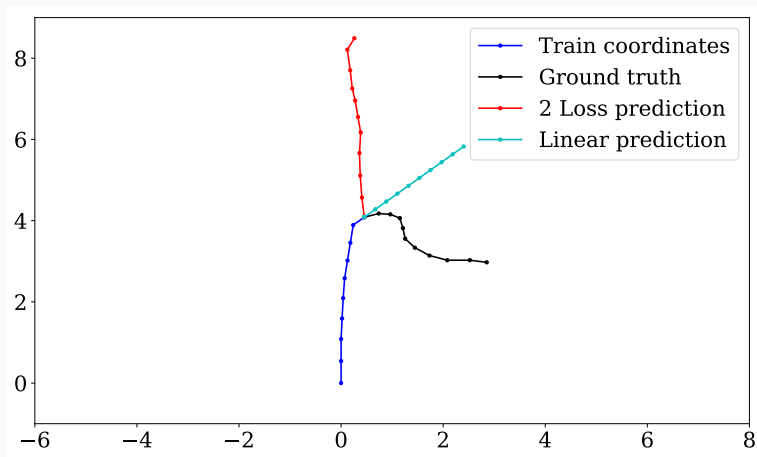
## Representation: a few examples

Type 3 trajectory:



## Representation: a few examples

But sometimes prediction fails:





What can we conclude ?

- CNN is not well structured for this application
- LSTM gives better trajectories compare to the other models
- The statics trajectories are the most difficult to predict

What can be debated ?

- The inputs of the CNN model can be improved
- The Inputs can be discussed concerning the initialisation of the matrix of interactions
- Our data augmentation can be discussed

## **Annex**

---

# Data Augmentation

The data augmentation is divided in two steps:

1. Flipping the trajectory, i.e  $x = -x$  and  $y = y$
2. Add noise to all points (except two first):  $0 < \epsilon_x < 0.1$  and  $0 < \epsilon_y < 0.1$ . Only for type 2 and 3 trajectory, otherwise for type 1, trajectory become unrealistic.

What is the problem:

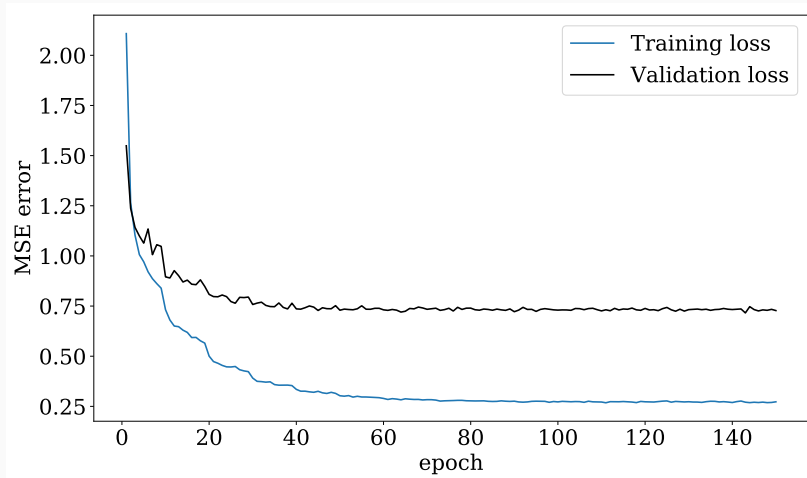
1. Flipping trajectories can remove behavioural pattern. If pedestrians tend to deviate on the right (or left) to avoid interaction, this pattern is lost.
2. Adding noise doesn't change much the trajectory: two very similar trajectories can be in the train AND in the test set.

The first problem is hard to overcome. The solution would be to not flip the trajectory  $\Rightarrow$  we tried and with less data, we have worst prediction.

The second problem is not a really one. Trajectory are often similar for a lot of them, the issue can be problematic for "unconventional" trajectories, and the model can "learn by heart" the output.

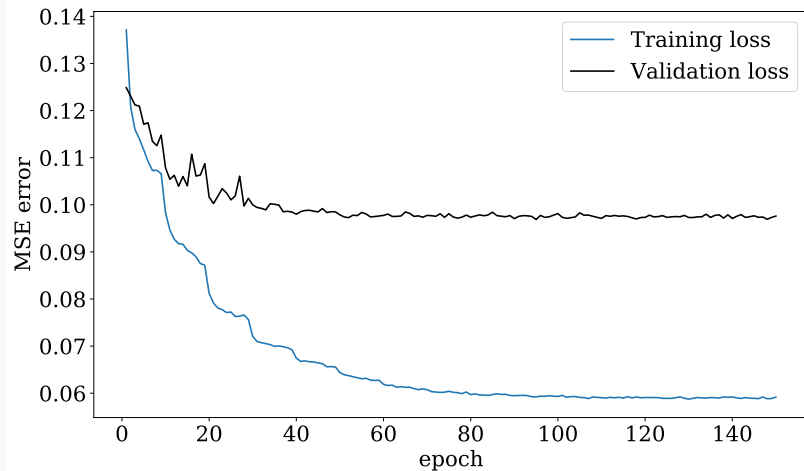
## Loss Functions: Coordinates model

Coordinates model:



## Loss Functions: Speed model

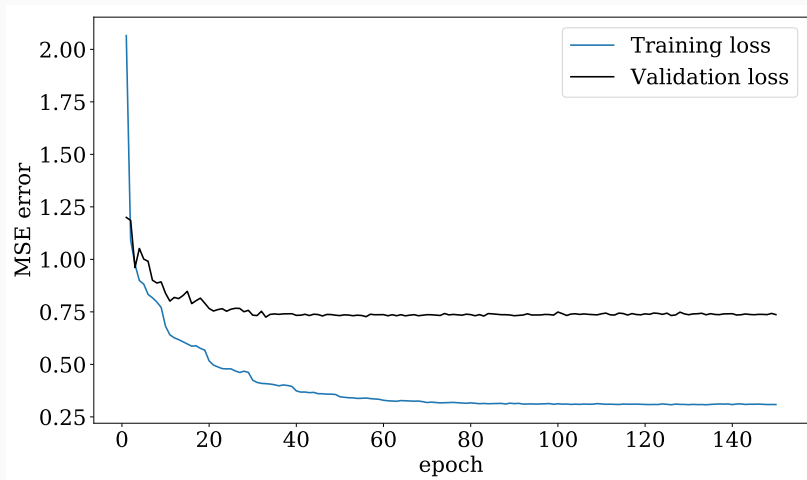
Speed model:





## Loss Functions: 2 Loss model

2 Loss model:



## Loss Functions: 3 Loss model

3 Loss model:

