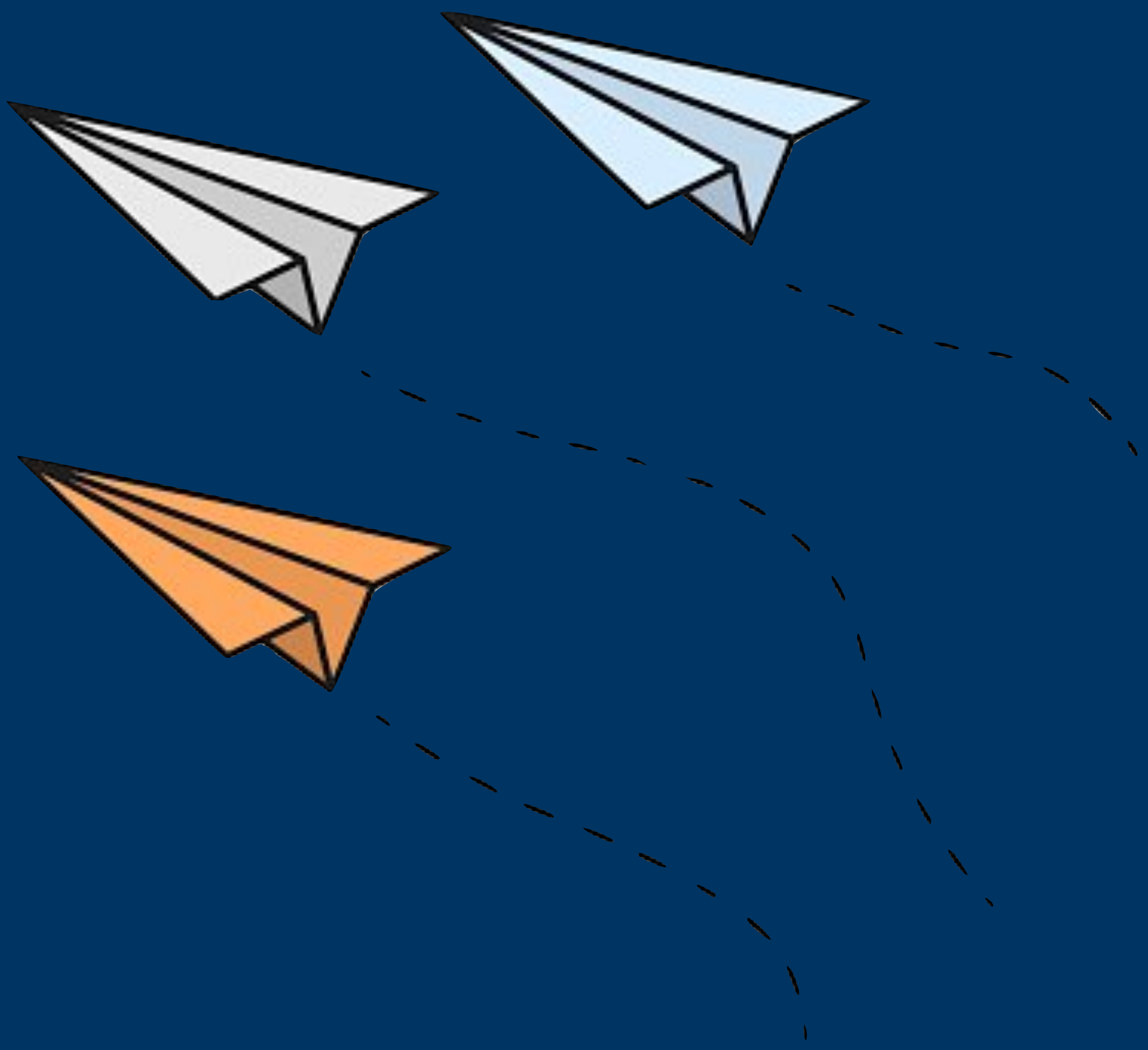
Physics and IA : Predicting Paper Airplane Trajectories



Arthur Oudeyer - 2024

TIPE

Problematic

In what cases and to what extent can a model developed by AI be more relevant than a physical model in predicting the trajectories of paper airplane ?

- **Accuracy**
- **Robustness**
- **Amount of data**

Summary

I. Technical Aspects

1. Problem formalization and constraints
2. Data collection
3. Data processing

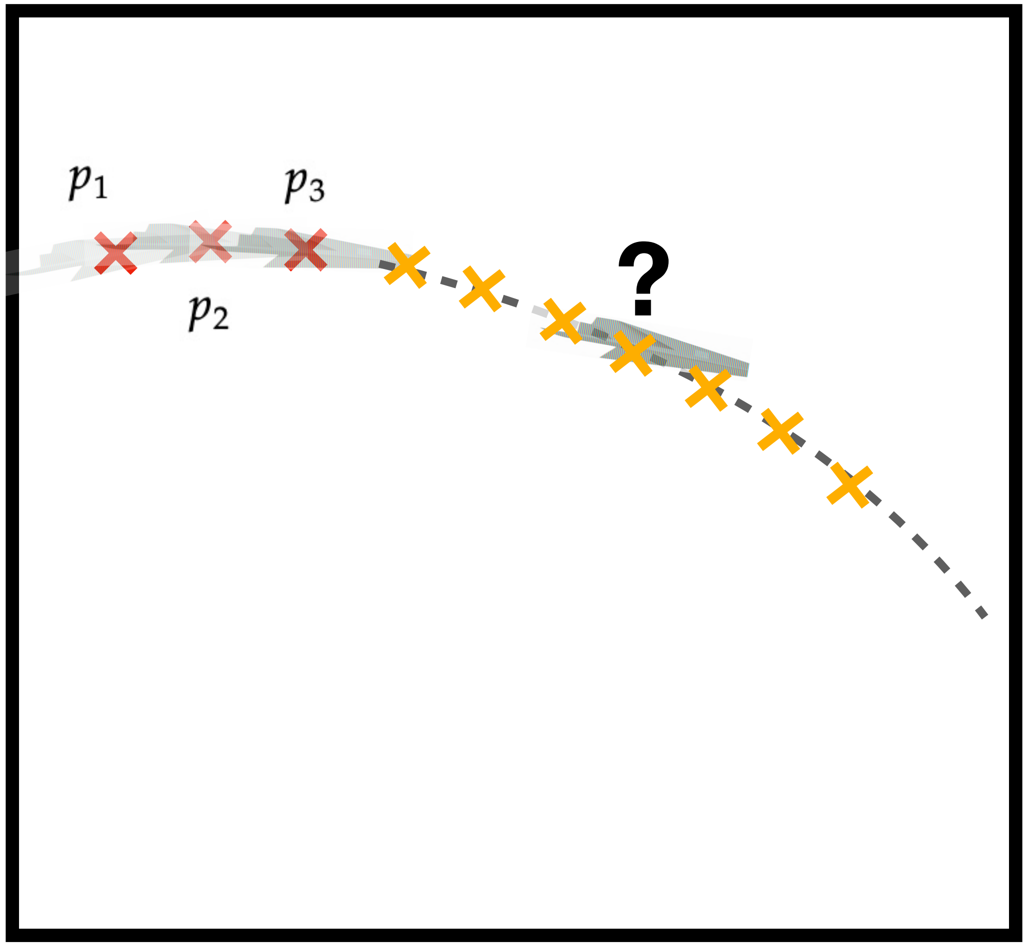
II. Modeling

1. Physic Model
2. IA Model

III. Comparison

1. Models validity verification
2. Comparison of the models

I. Stakes : Problem Formalization



$$p_1 = (x_1 , y_1 , \theta_1, t_1)$$
$$p_2 = (x_2 , y_2 , \theta_2, t_2)$$
$$p_3 = (x_3 , y_3 , \theta_3, t_3)$$

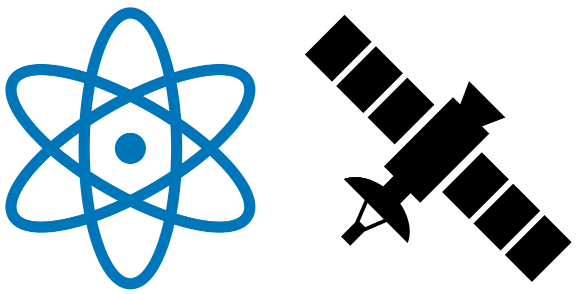
$$\downarrow$$

$$p_i = (x_i , y_i , \theta_i, t_i)$$

I. Stakes : Problem Formalization

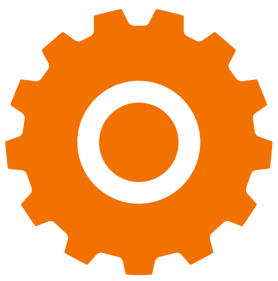
Physical Model

Physic's Laws



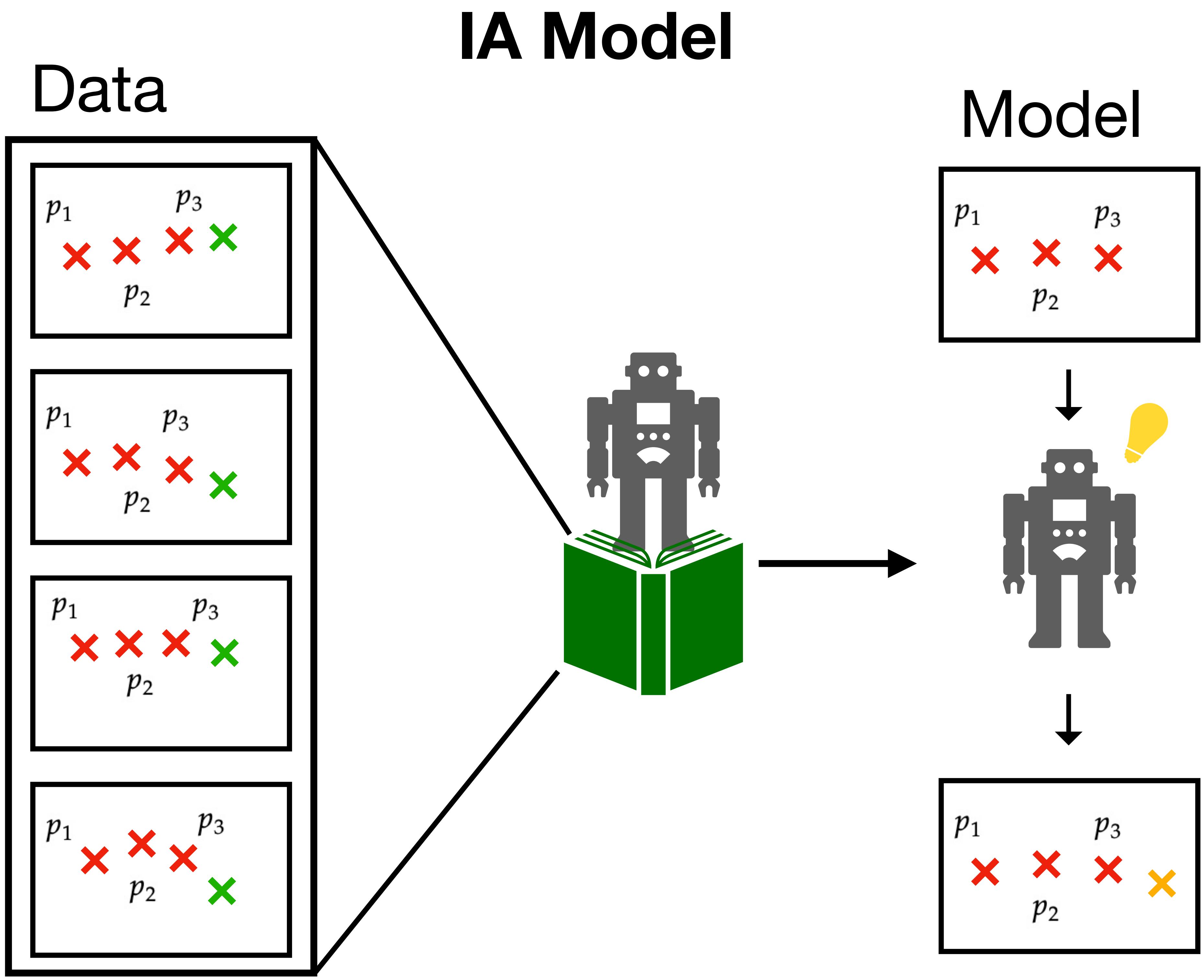
$$m\frac{d\vec{v}}{dt} = \sum \vec{F}$$

Numerical Simulation



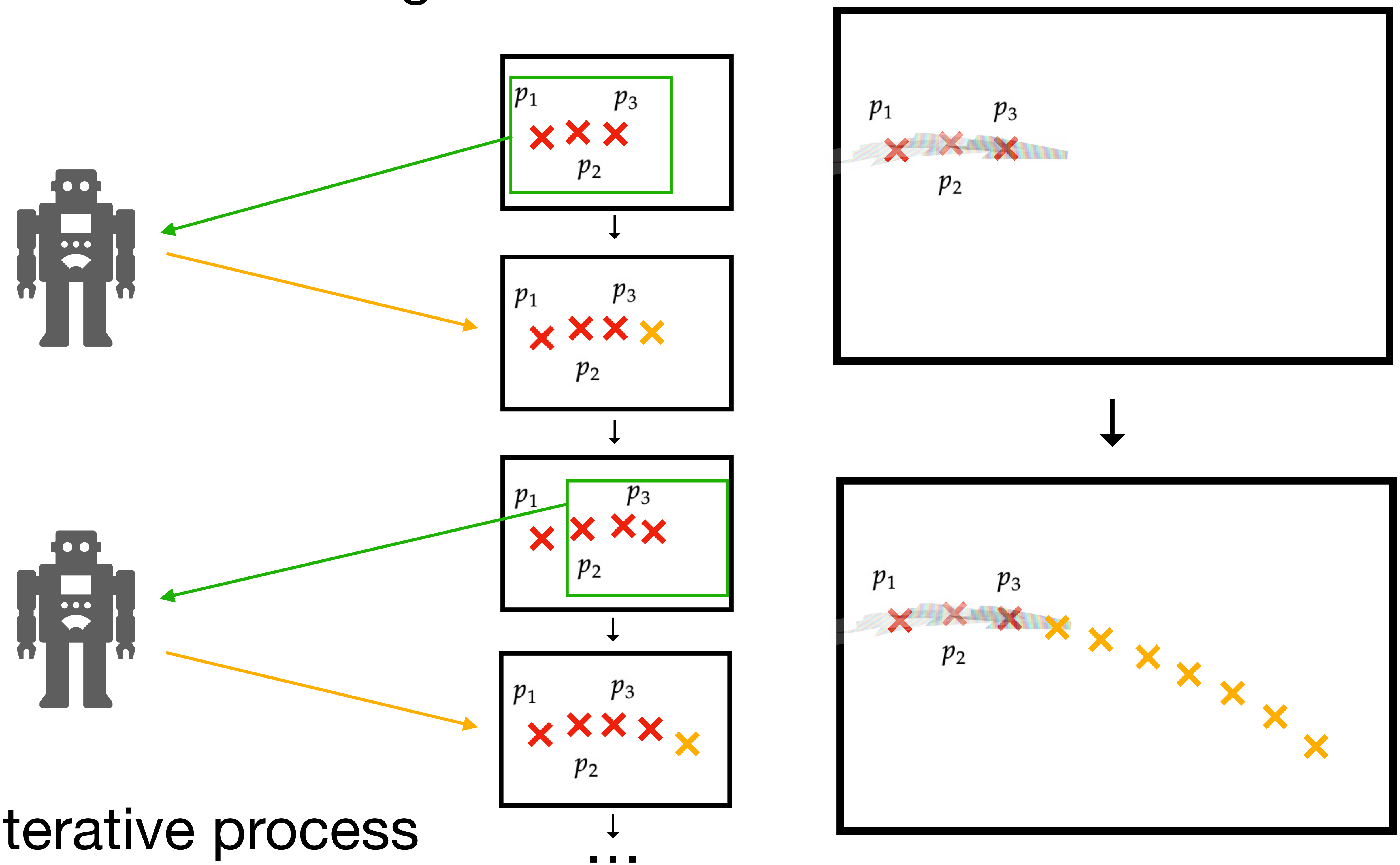
$$x_{i+1} = x_i + v_i dt + a_{i+1} dt^2$$

I. Stakes : Problem Formalization



I. Stakes : Problem Formalization

Auto-regression



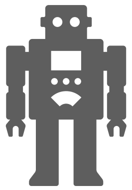
→ Iterative process

I. Stakes : Constraints

To predict the trajectories, models needs :

Physic :
- drag and lift coefficients

IA :
- Model training



→ **Data Collection needed**

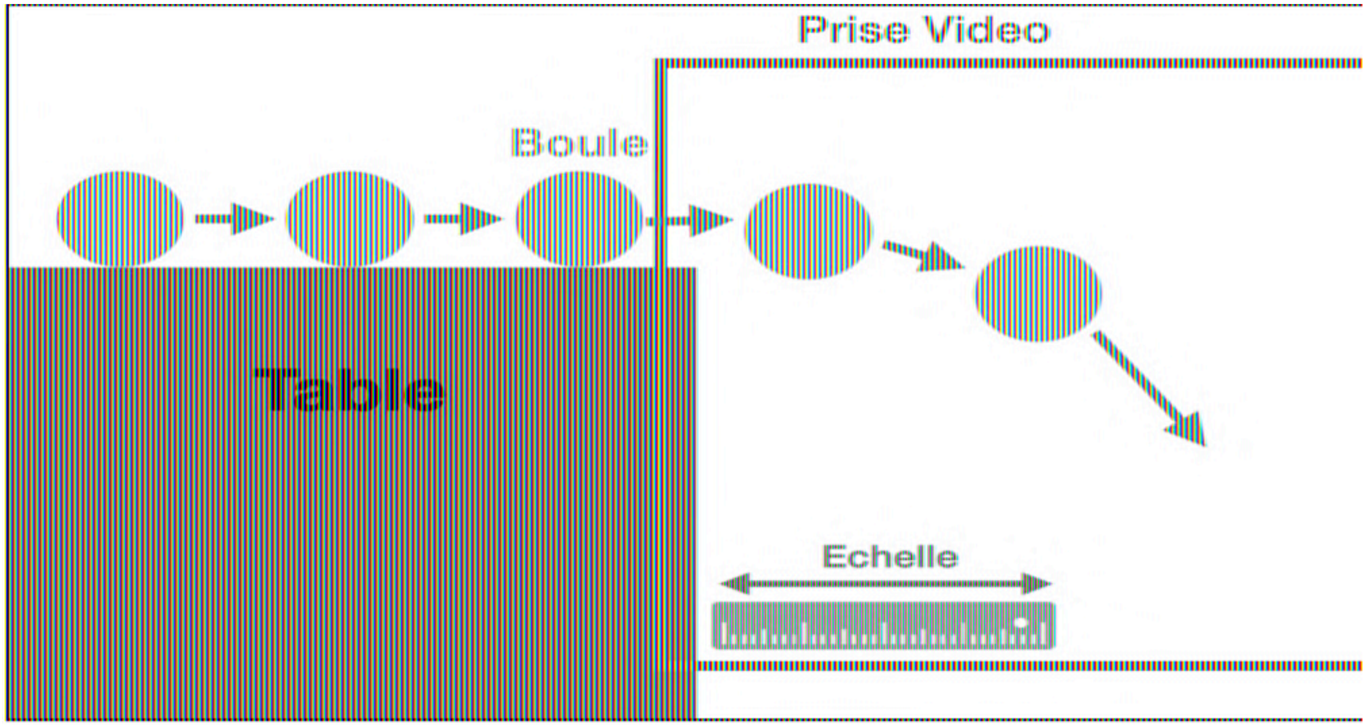
I. Stake : Experiment

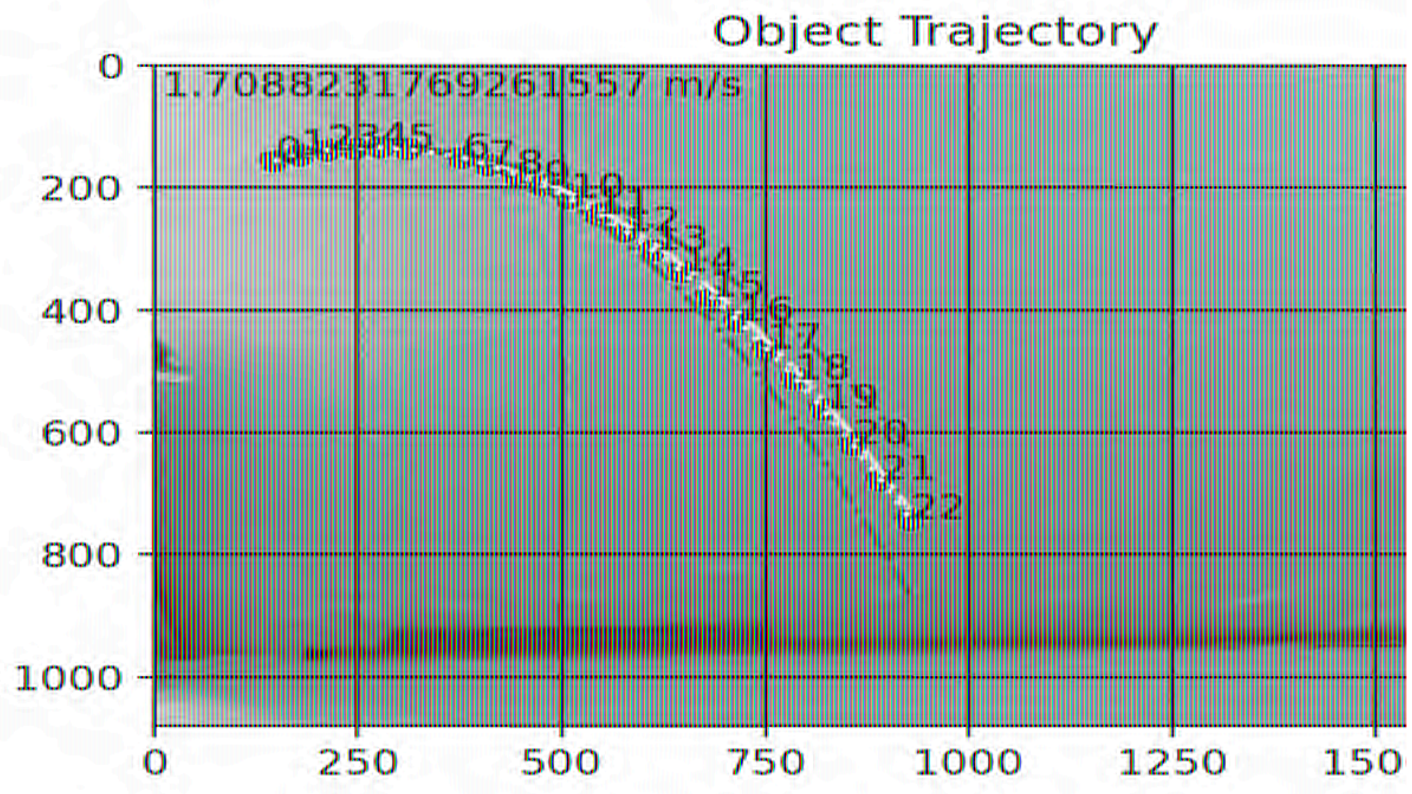
Dialog with
Christophe Airiau

https://www.imft.fr/en/personal-page/airiau-christophe-en/

Scientist at Fluid Mechanics Institute of Toulouse

→ Prototype experiment



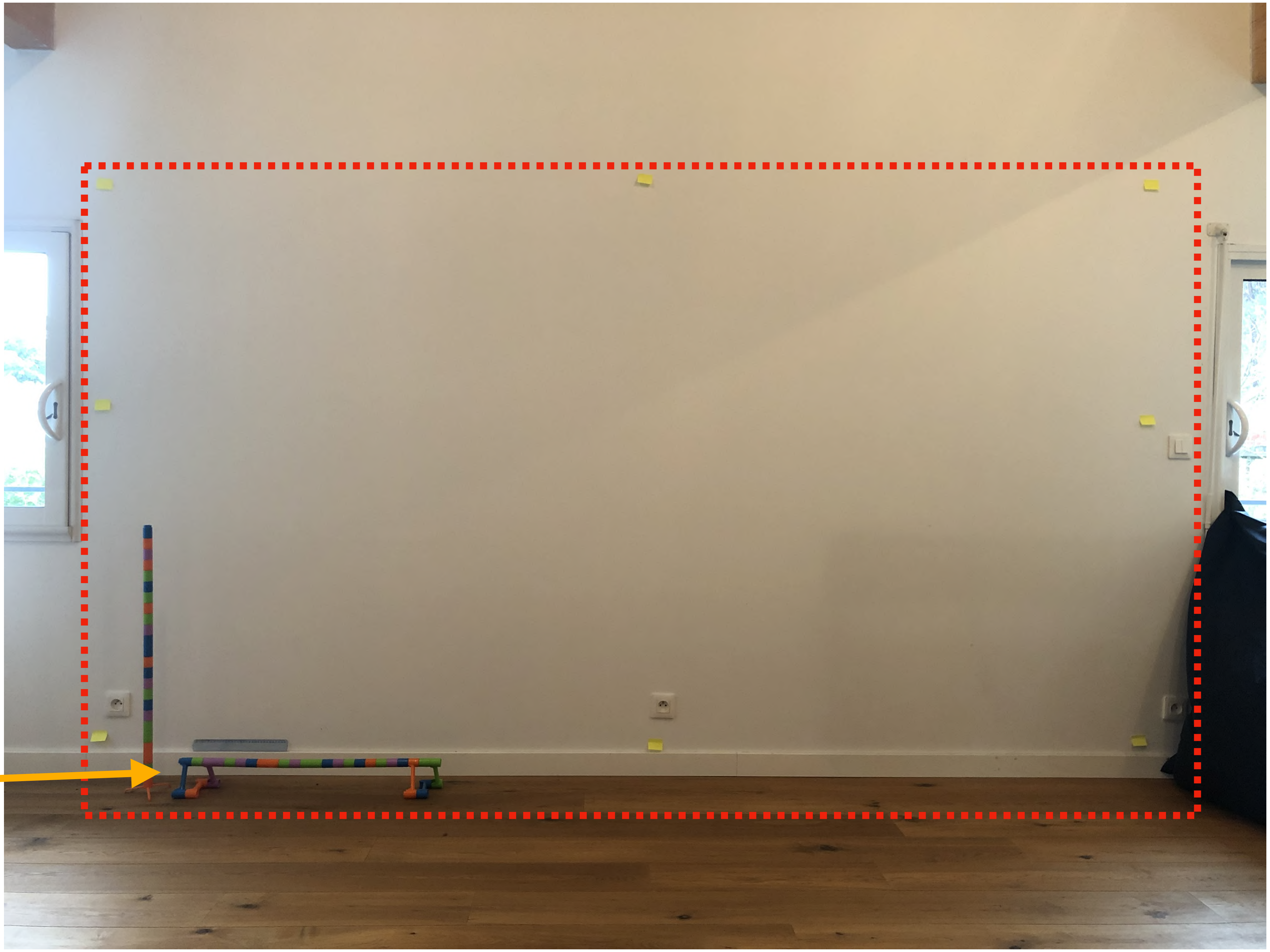


I. Stake : Experiment

paper airplane

Scale

(future conversion)



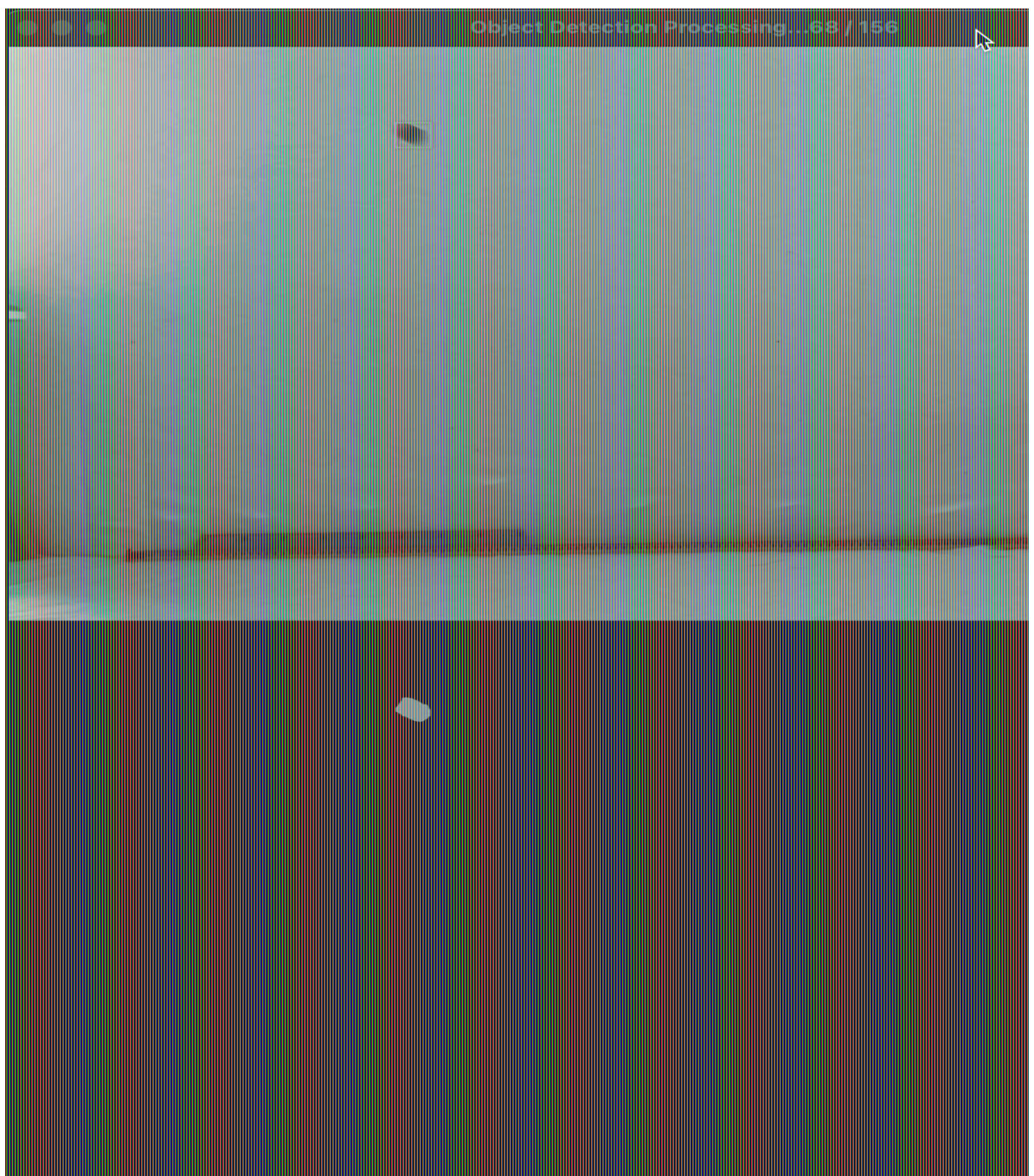
Video capture zone

I. Technical Stakes : Data acquisition

https://scholar.google.com/citations?user=ZLA7iioAAAAJ

→ YoloV8

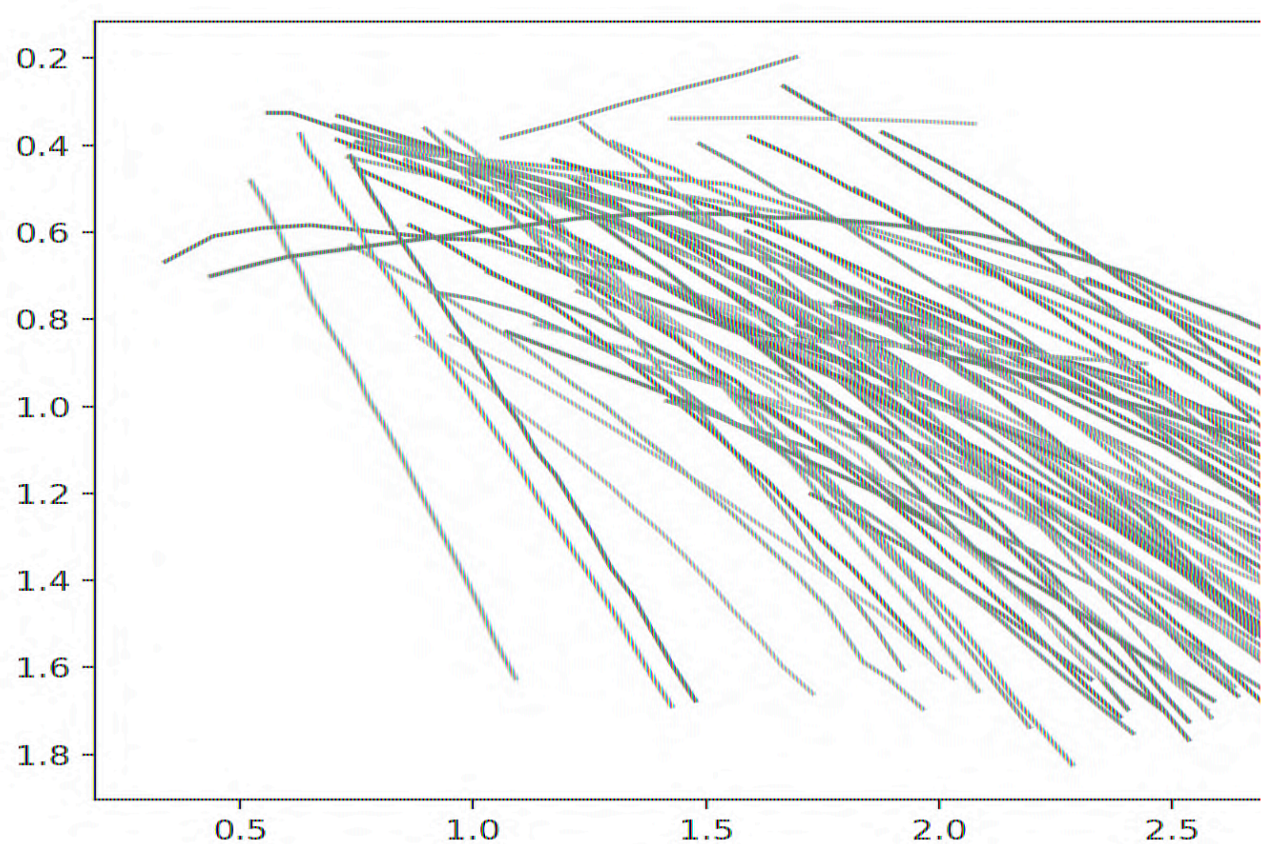
OpenCv Mask



Grgur Kovač (INRIA)

I. Technical Stakes : Data processing

- Conversion
- Encoding
- Speed / Acceleration (Euler)



x,	y,	θ,	t
1.224	0.736	-19	2.
1.405	0.811	-22	2.
1.501	0.851	-23	2.
1.587	0.895	-27	2.
1.682	0.941	-26	2.
1.771	0.99	-29	2.
1.859	1.041	-30	2.
1.953	1.087	-26	2.
2.039	1.136	-30	2.
2.139	1.191	-29	2.
2.222	1.243	-32	3.
2.324	1.307	-32	3.
2.412	1.363	-33	3.
2.505	1.422	-32	3.
2.597	1.482	-33	3.
2.694	1.552	-36	3.
2.78	1.614	-36	3.

90 trajectories (77 ‘classics’ / 13 ‘originals’)

II. Modeling : Physic

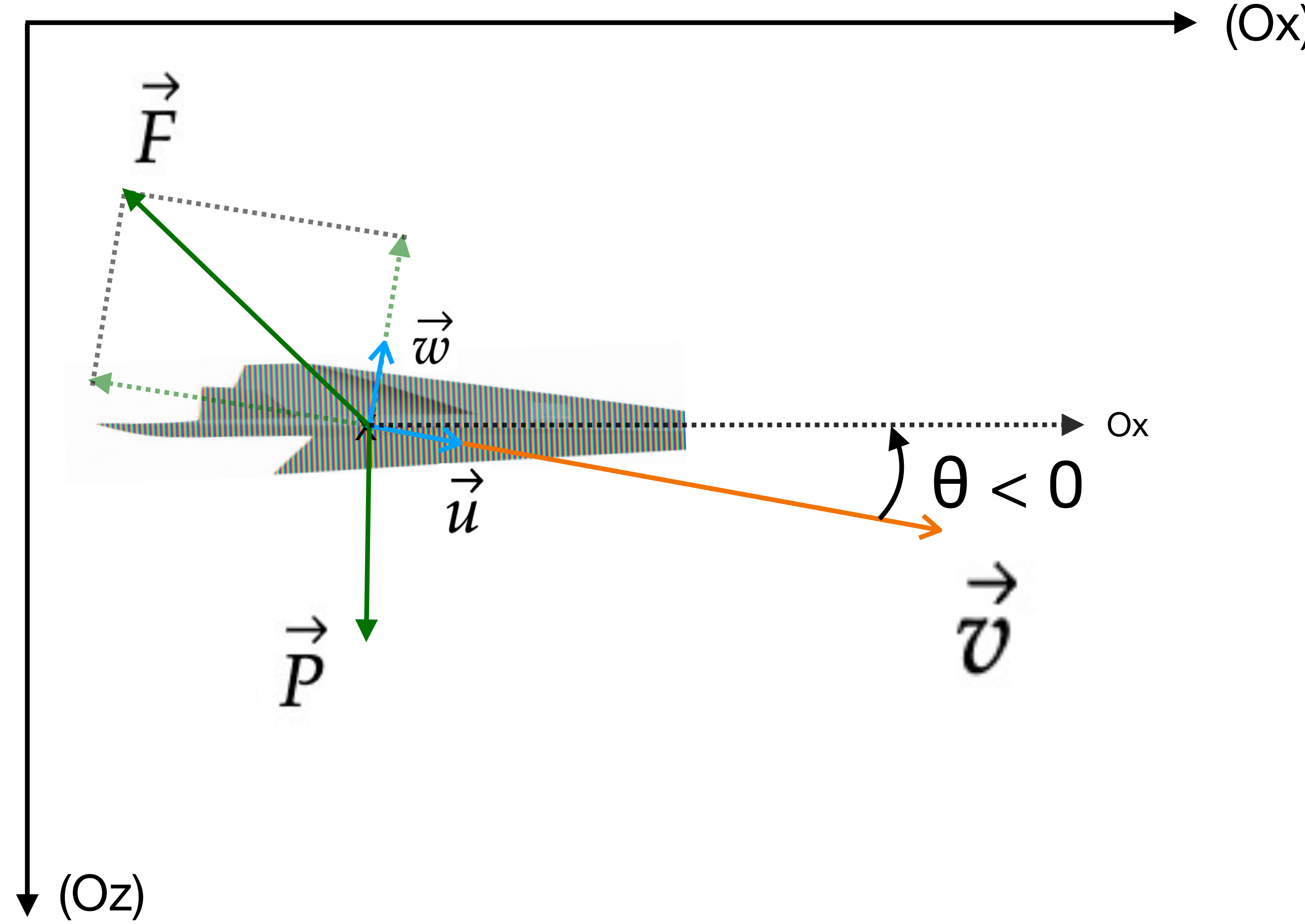
Mass : m = 10 g

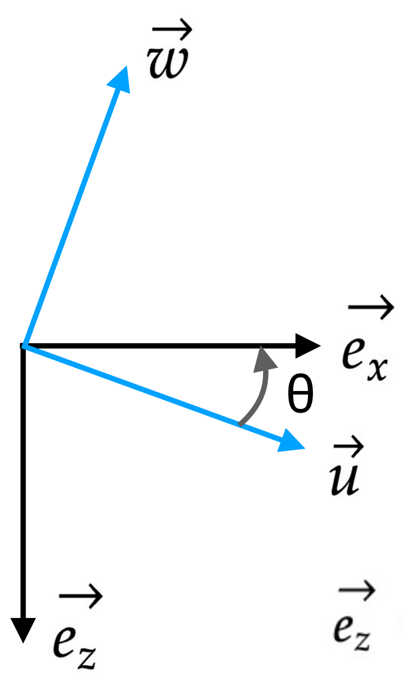
Surface : S = 0.03 m²

$$Re = \frac{\rho V L}{\eta} \approx 10^5$$

Hypotheses :

- weight, lift and drag
- Cx, Cz constants



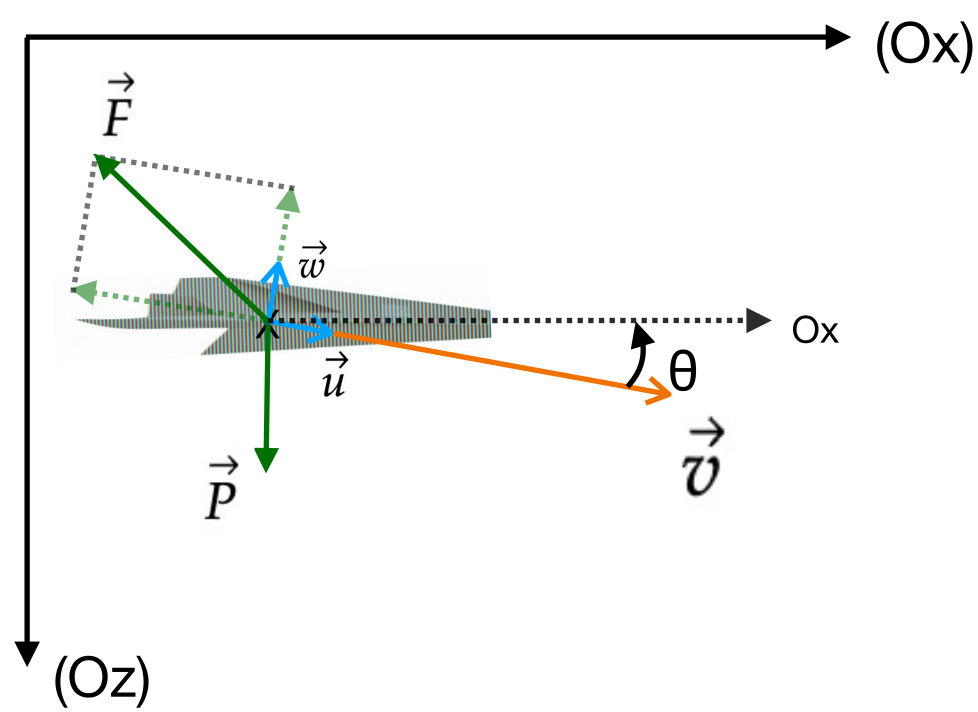


$$\vec{e_z} = -\sin(\theta)\,\vec{u} - \cos(\theta)\,\vec{w}$$

$$Poids\ :\ \vec{P} = mg\,\vec{e_z}$$

$$Aéro\ :\ \vec{F} = \frac{1}{2}\rho S V^2(-\boxed{C_x}\vec{u} + \boxed{C_z}\vec{w})$$

II. Modeling : Physic

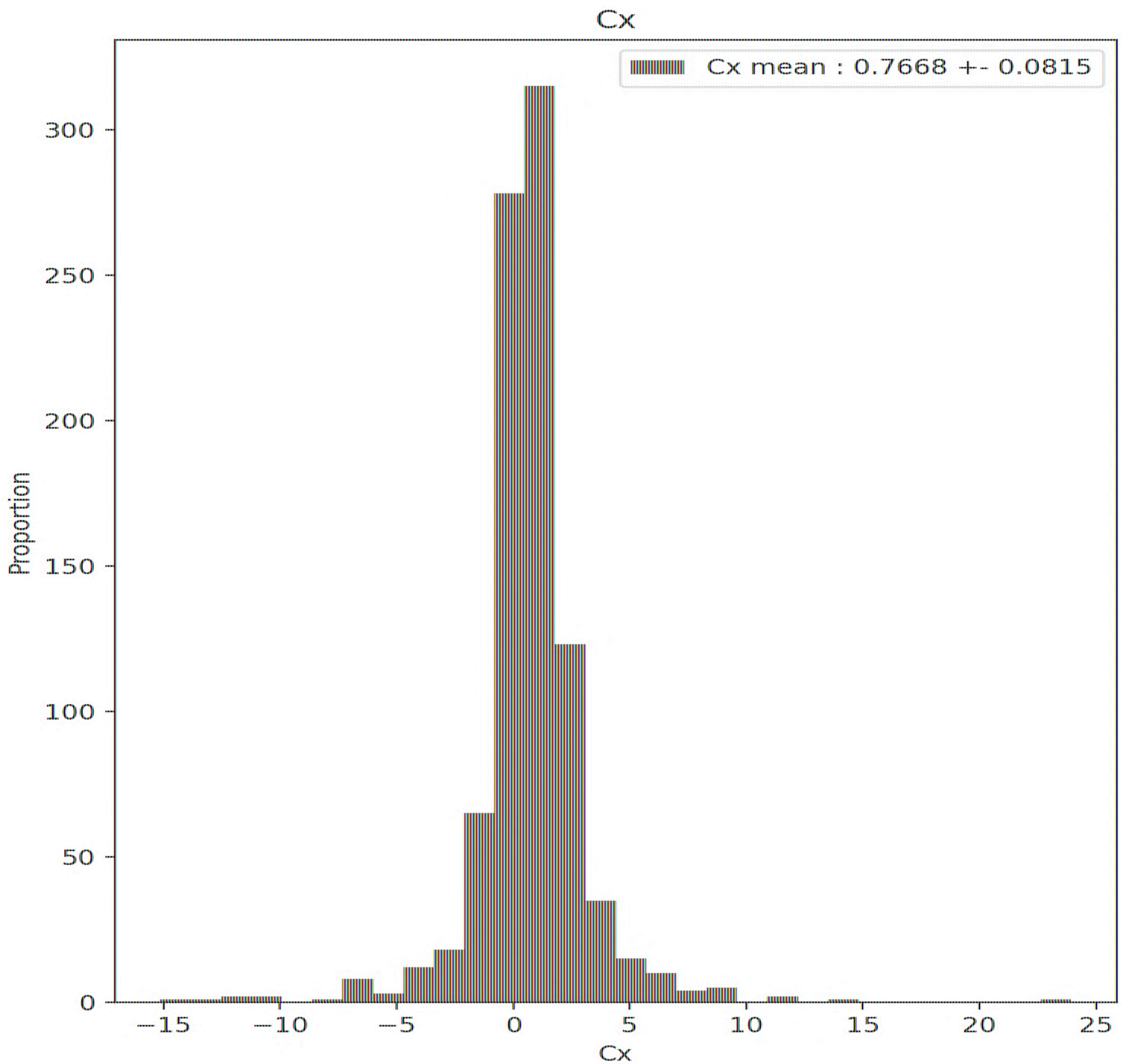


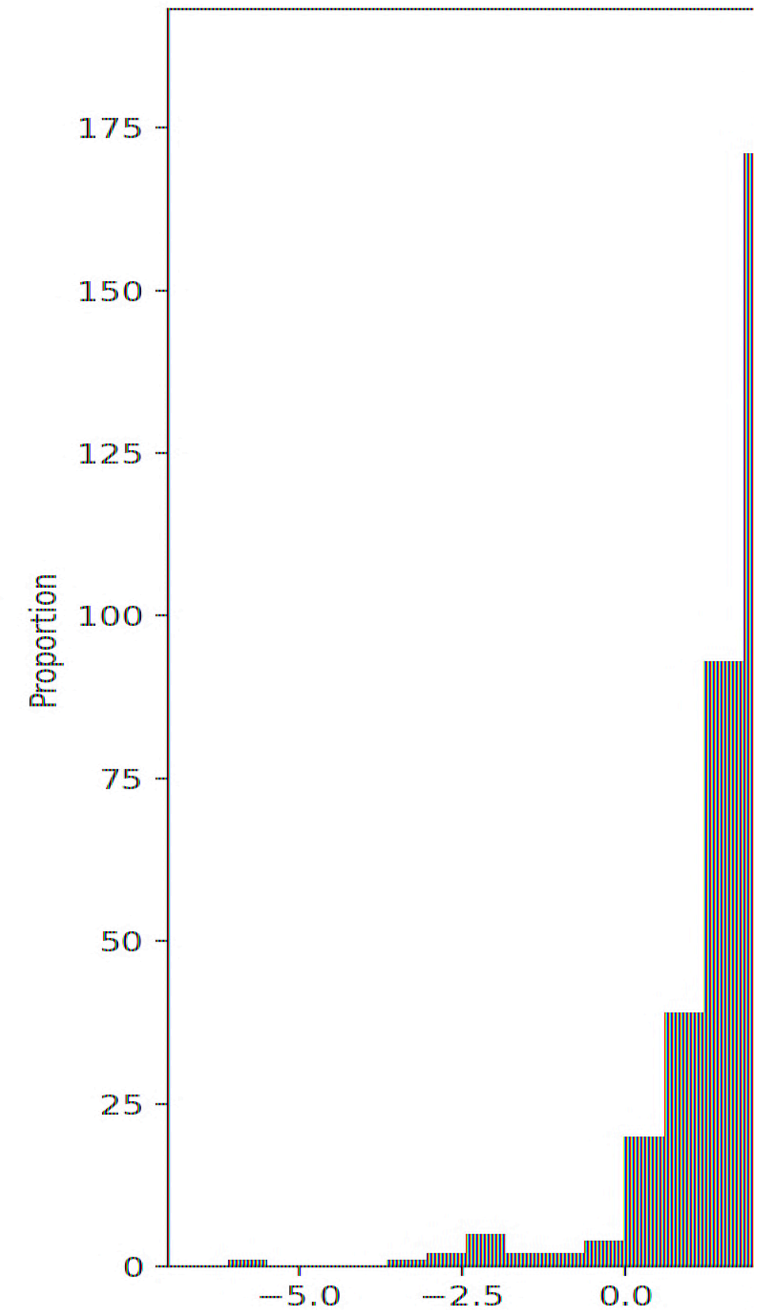
$$/\vec{u} \;:\; m\dot{v} = -\frac{1}{2}\rho S V^2 C_x - mg\sin(\theta) \qquad (1)$$

$$/\vec{w} \;:\; -mv\dot{\theta} = \frac{1}{2}\rho S V^2 C_z - mg\cos(\theta) \qquad (2)$$

$$\boxed{\begin{aligned} (1) \implies C_x &= \frac{-2m(\dot{v} + g\sin(\theta))}{\rho S V^2} \\ (2) \implies C_z &= \frac{-2m(v\dot{\theta} - g\cos(\theta))}{\rho S V^2} \end{aligned}}$$

II. Modeling : Physic

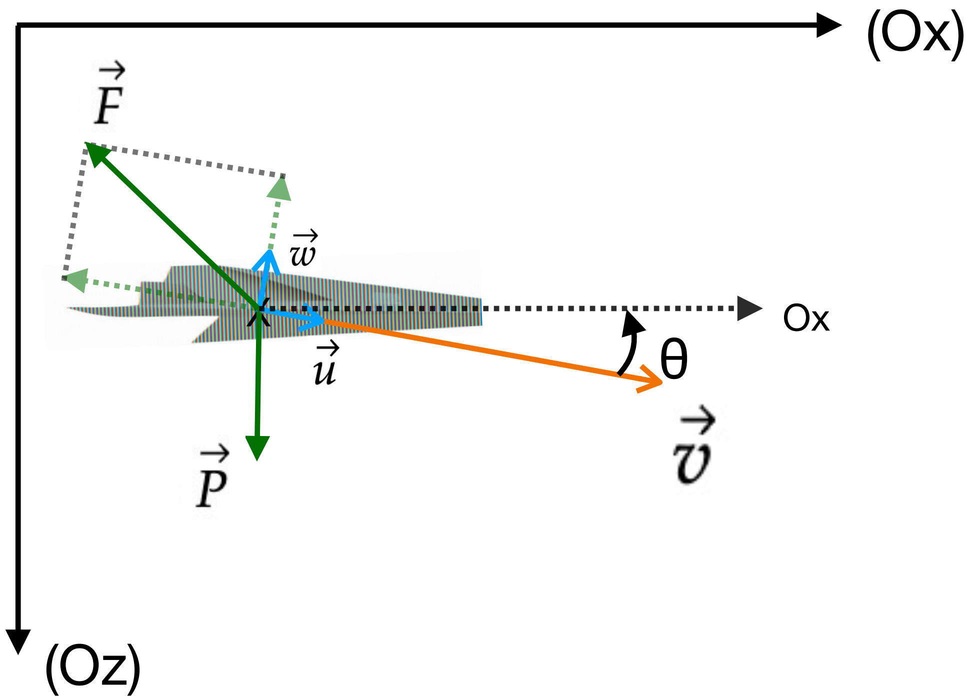




$$Type\ A : u(C) = \frac{s(C)}{\sqrt{N}}$$

$$\boxed{\begin{array}{l} Cx = 0.77 \mp 0.08 \\ Cz = 3.01 \mp 0.06 \end{array}}$$

II. Modeling : Physic



$$/\overrightarrow{e_x} \; : \; m\dot{v}_x \; = \; \overrightarrow{F_{trainée}} \cdot \overrightarrow{e_x} \; + \; \overrightarrow{F_{portance}} \cdot \overrightarrow{e_x}$$

$$/\overrightarrow{e_z} \; : \; m\dot{v}_z \; = \; \overrightarrow{F_{trainée}} \cdot \overrightarrow{e_z} \; + \; \overrightarrow{F_{portance}} \cdot \overrightarrow{e_z} \; + \; mg$$

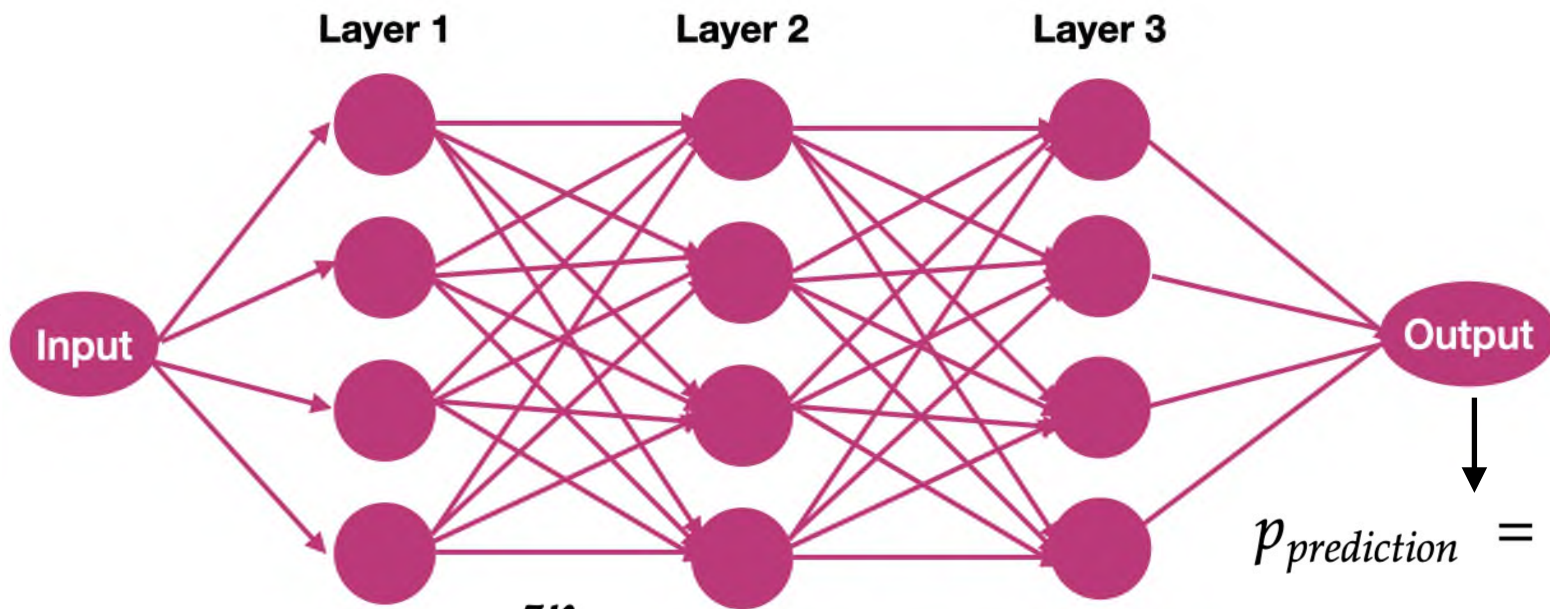
$$\boxed{\begin{aligned} /\overrightarrow{e_x} \; &: \; m\dot{v}_x \; = \; F_{trainée} \cdot \cos(\theta) \; + \; F_{portance} \cdot \sin(\theta) \\ /\overrightarrow{e_z} \; &: \; m\dot{v}_z \; = \; -F_{trainée} \cdot \cos(\theta) \; - \; F_{portance} \cdot \sin(\theta) + \; mg \end{aligned}}$$

II. Modeling : IA

→ Sklearn : MLPRegressor (neural network)

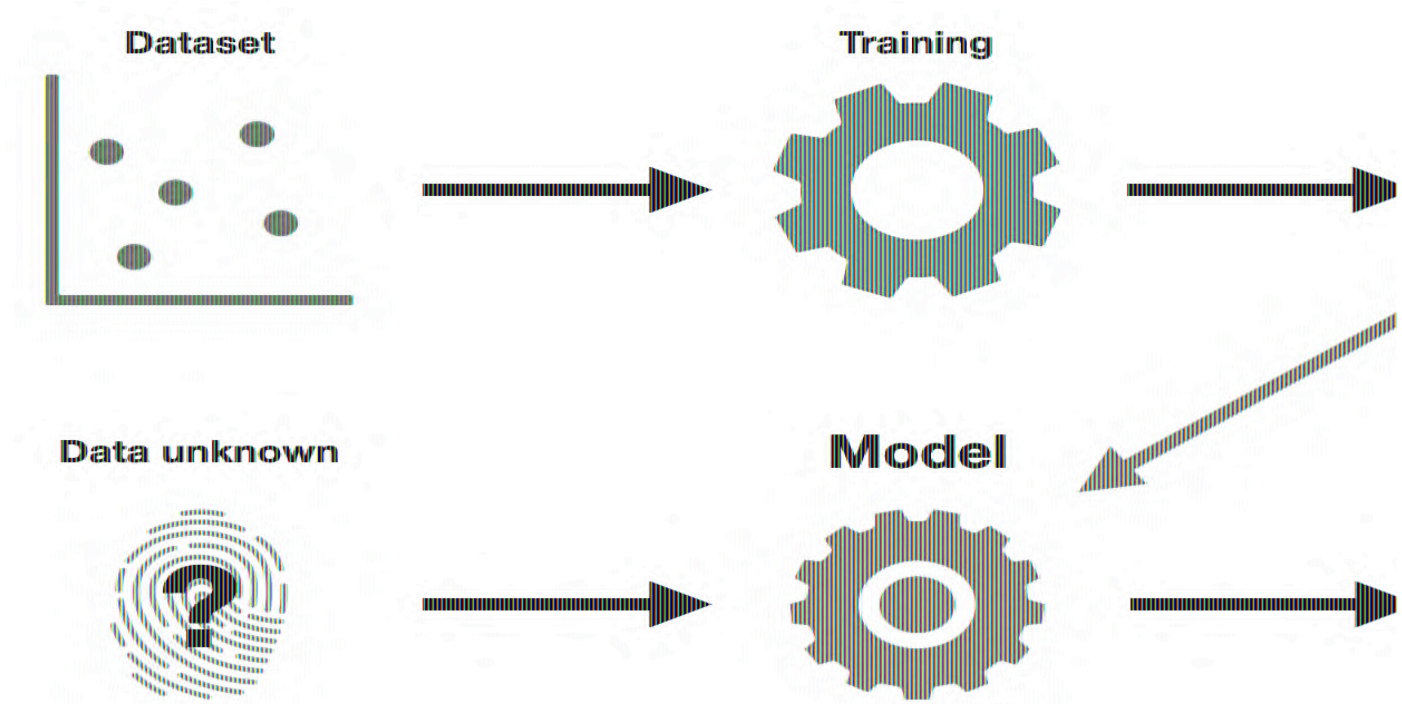
$p_1 = (x_1\ ,\ y_1\ ,\ \theta_1,\ t_1)$
$p_2 = (x_2\ ,\ y_2\ ,\ \theta_2,\ t_2)$
$p_3 = (x_3\ ,\ y_3\ ,\ \theta_3,\ t_3)$

$t_{objectif}$



w_i

$p_{prediction} = (x_p\ ,\ y_p\ ,\ \theta_p,\ t_{objectif})$

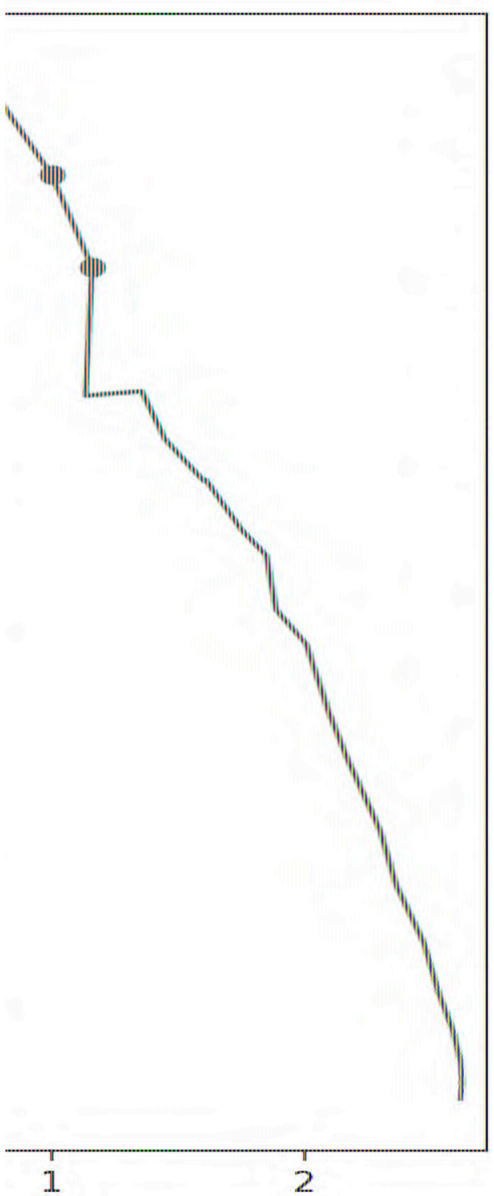


Gradient Descent

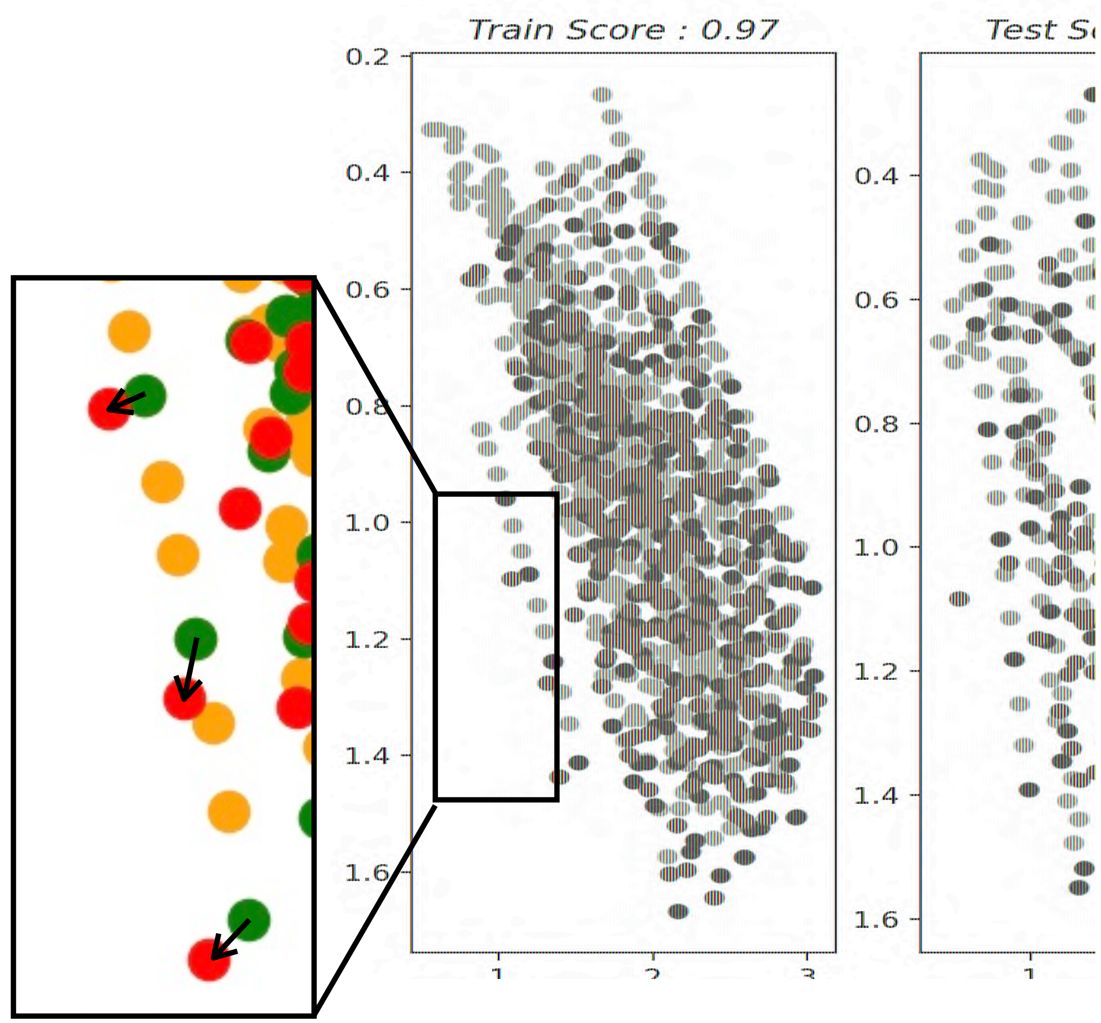
$$Loss = \sum_i (y_i - y_{i,\,predict})^2$$

II. Modeling : IA

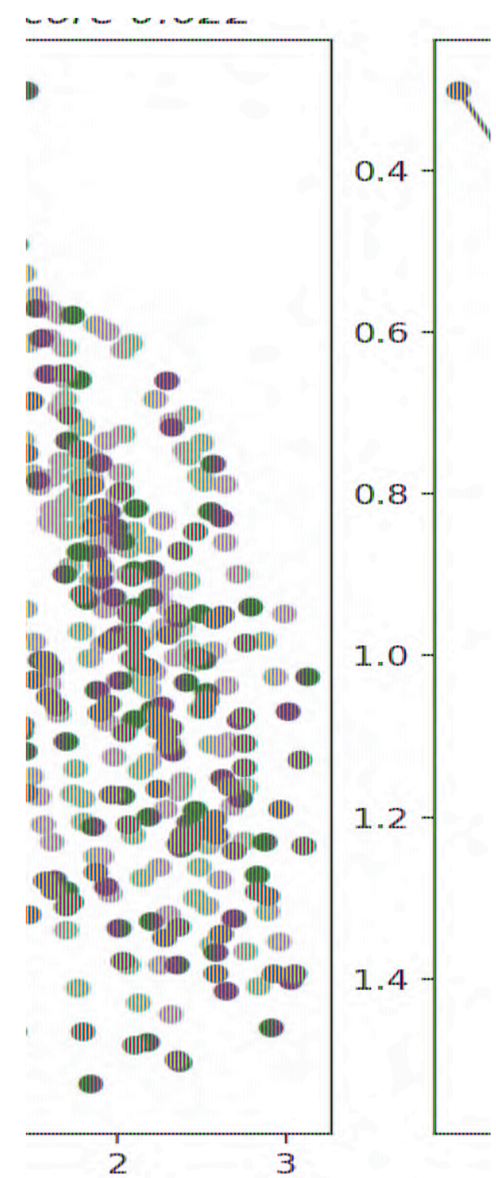
Network size (50x50)



Loss (Error)

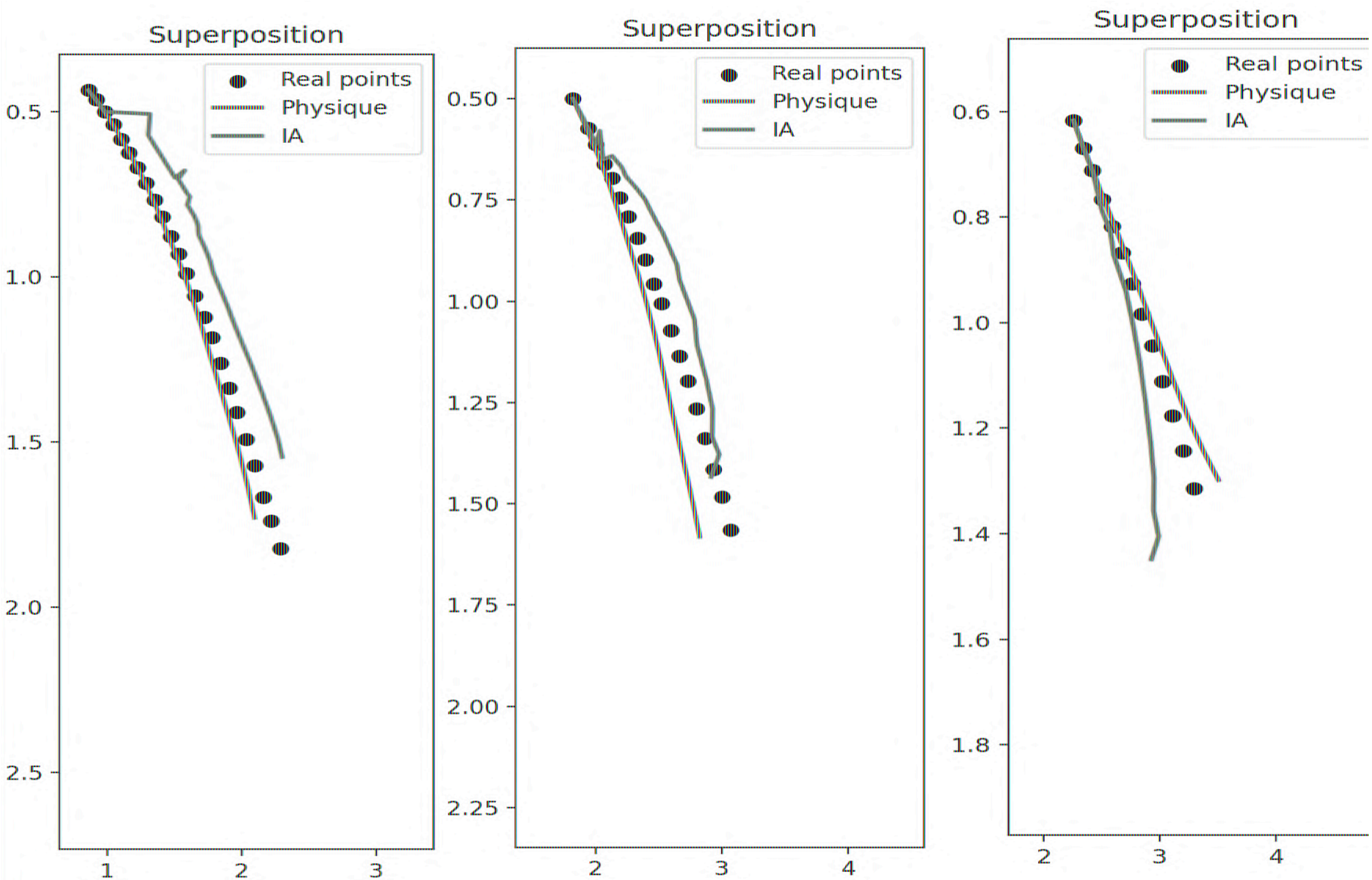


Training



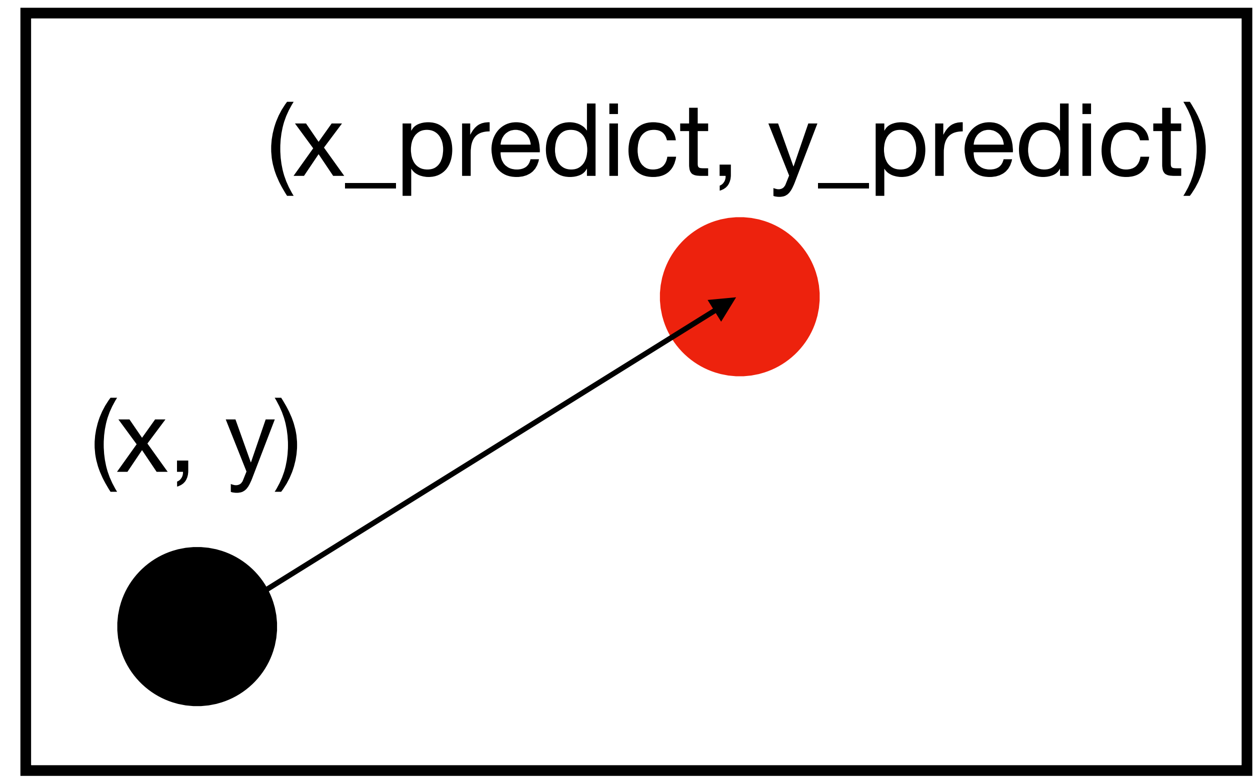
Auto-regression

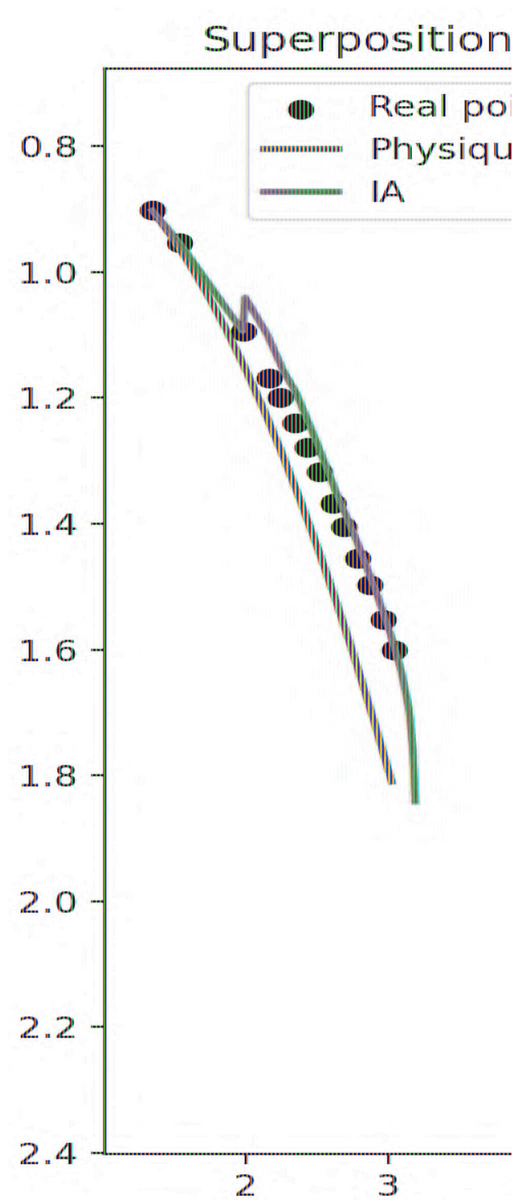
III. Comparison : Models validity



III. Comparison : Error quantification

How to compare models ?
→ Distance Function (Error)





$$d_i = \sqrt{(x_{predict,\,i} - x_i)^2 + (y_{predict,\,i} - y_i)^2}$$

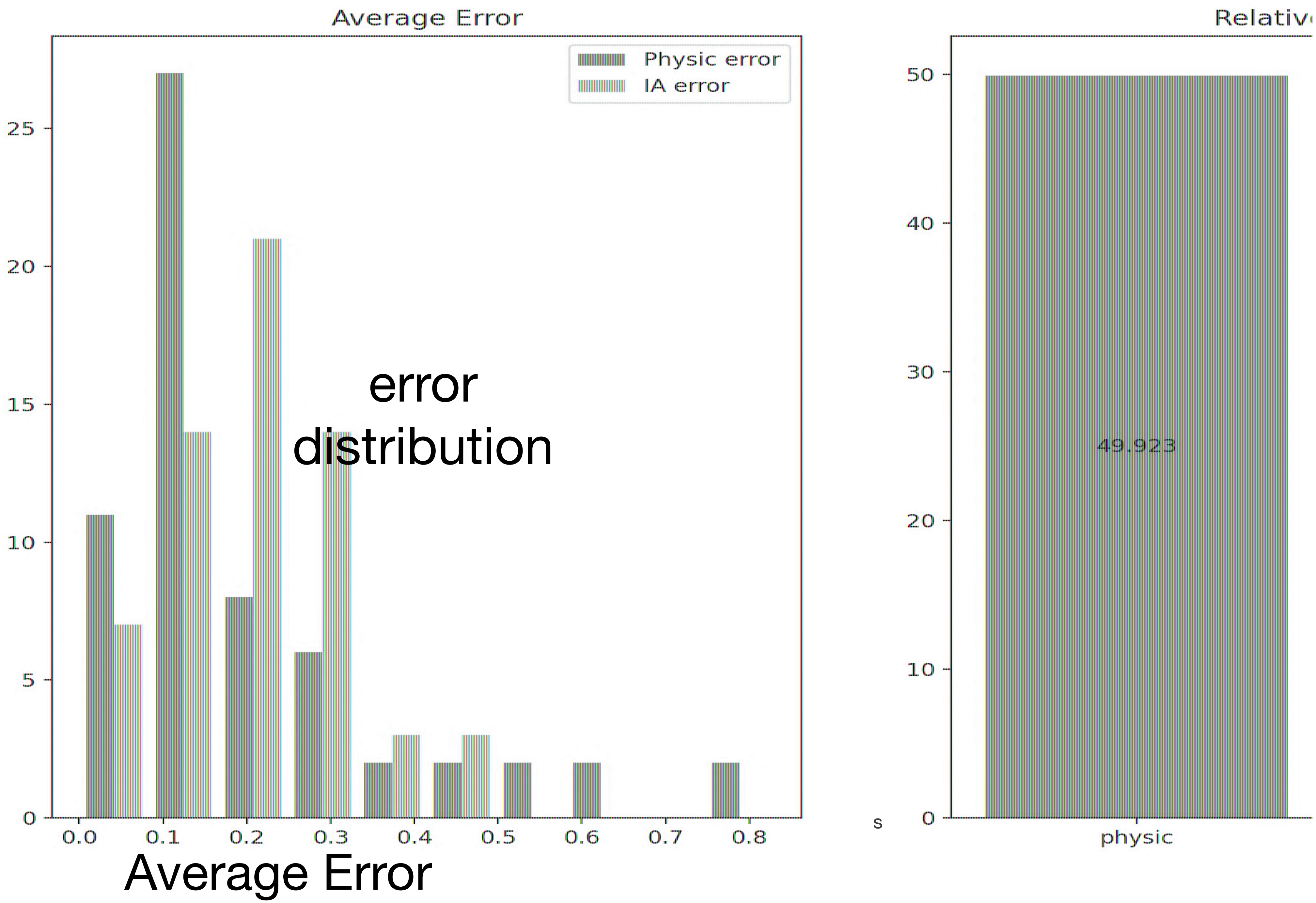
$$Error = \frac{1}{n}\sum_{i}^{n} d_i$$

III. Comparison : Accuracy

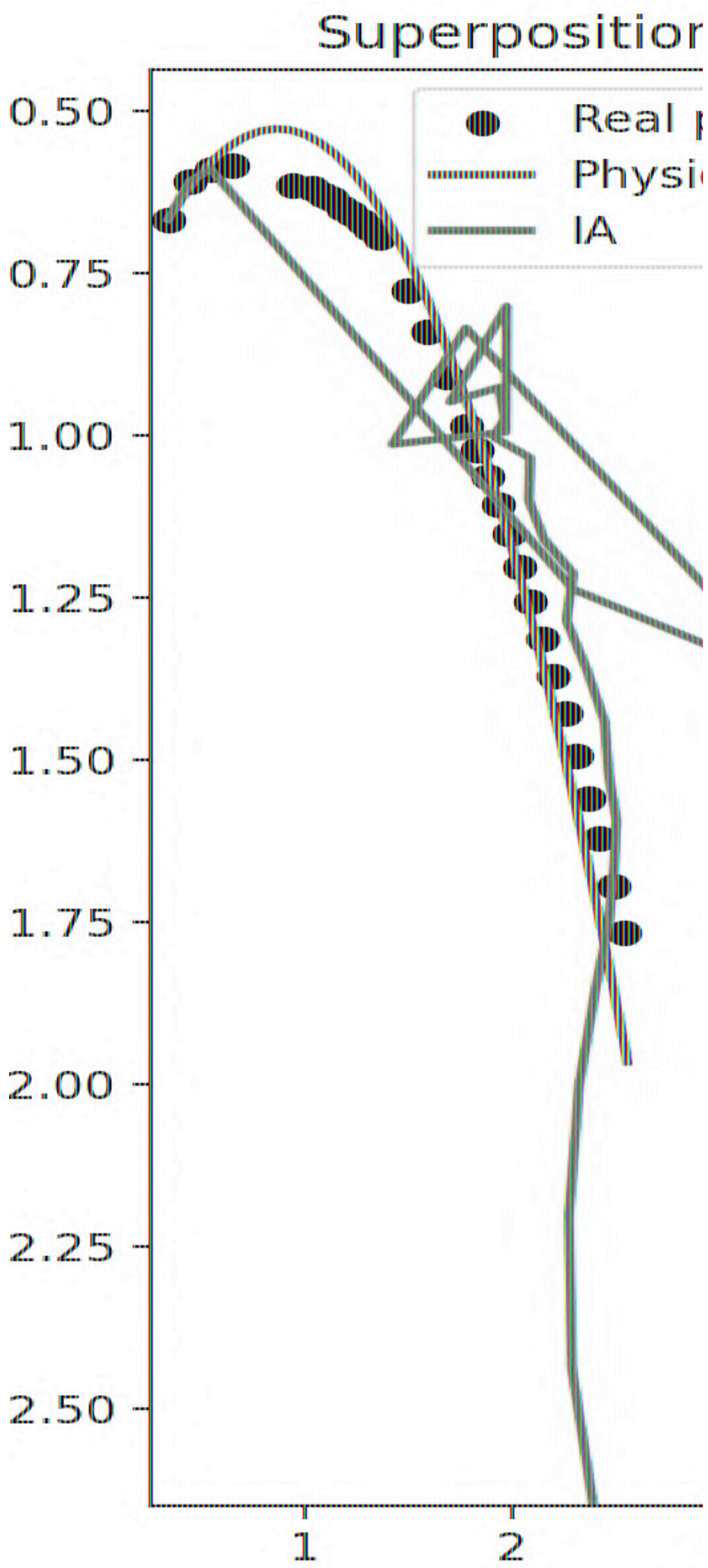
General Accuracy

IA trained with 77 trajectories 'classics' (100%)

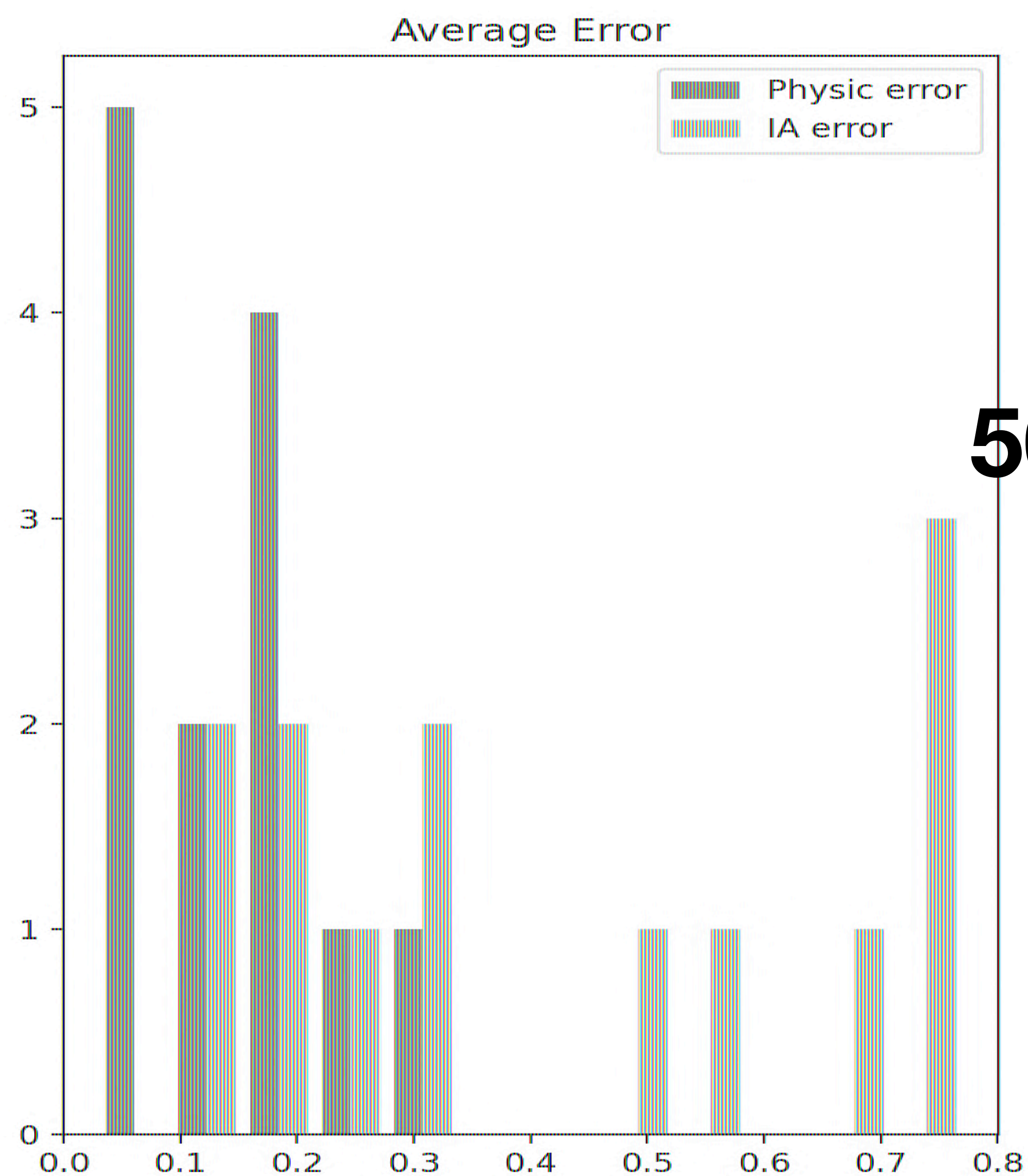
number of trajectories

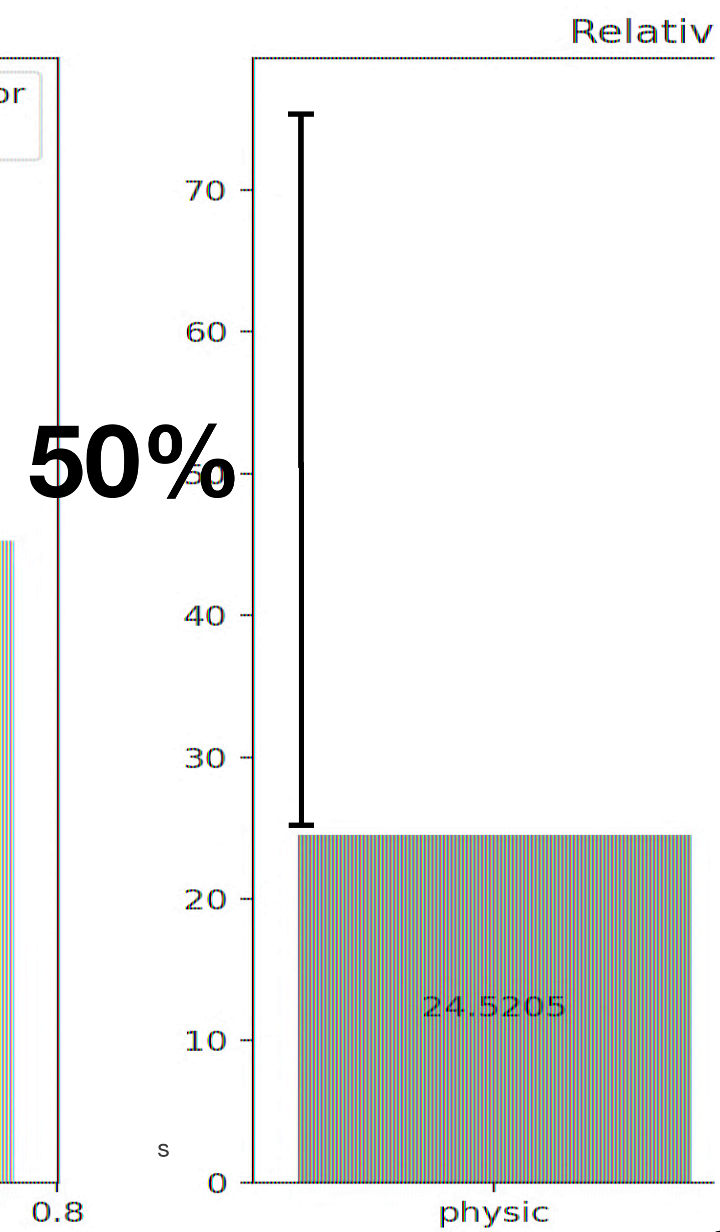


III. Comparaison : Robustesse



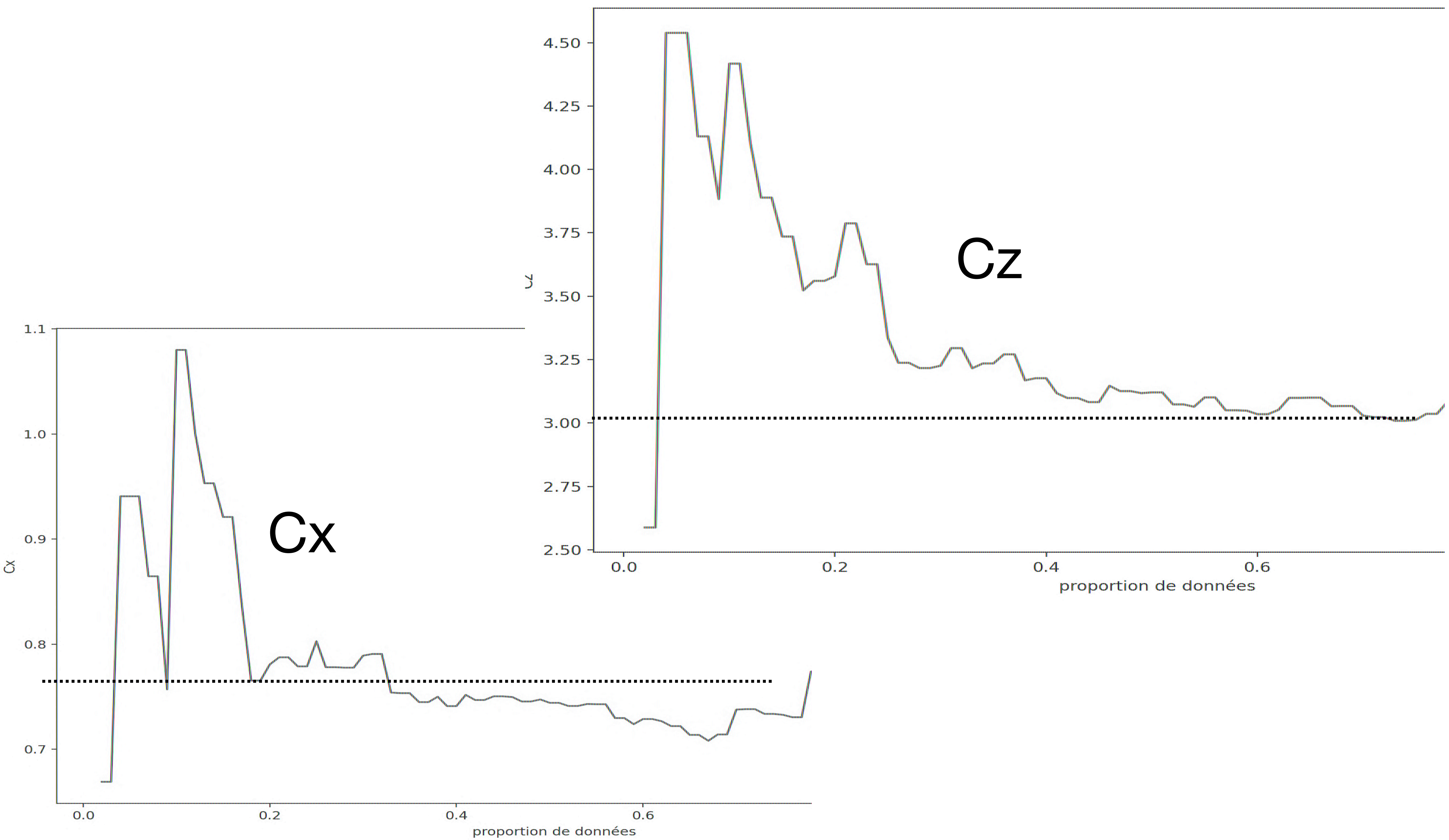
Test on more originals situation,
out of domain initial condition





III. Comparison : Amount of data

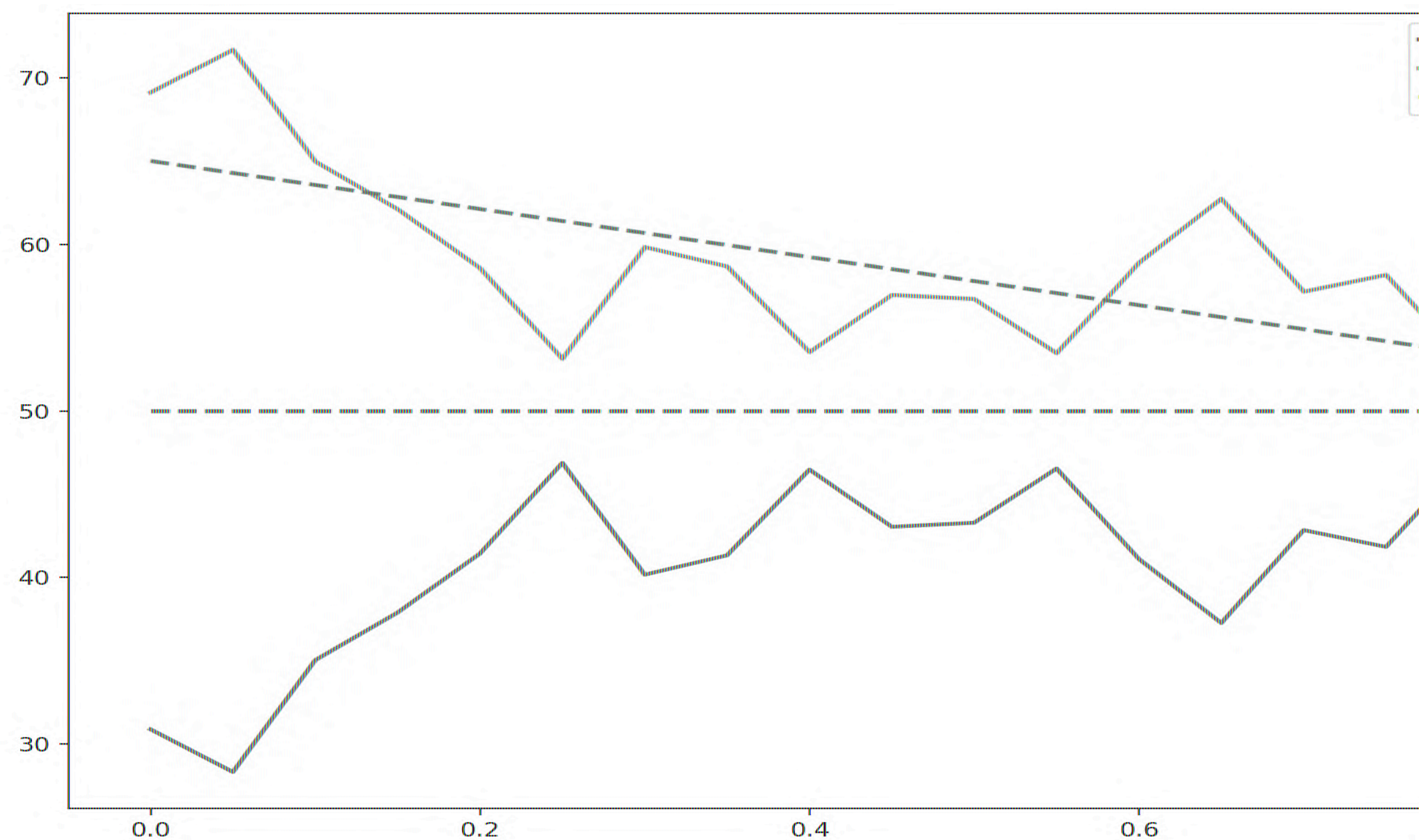
Cx/Cz as a function of the **amount of data available**



III. Comparison : Amount of data

Relative Error as a function of the **amount of data available**

Relative Error (%)



Proportion of data available used for the AI training

Conclusion

- The **amount of data** matter :
    - for Cx and Cz determination
    - for the AI model precision

- IA efficient only in its **training range**

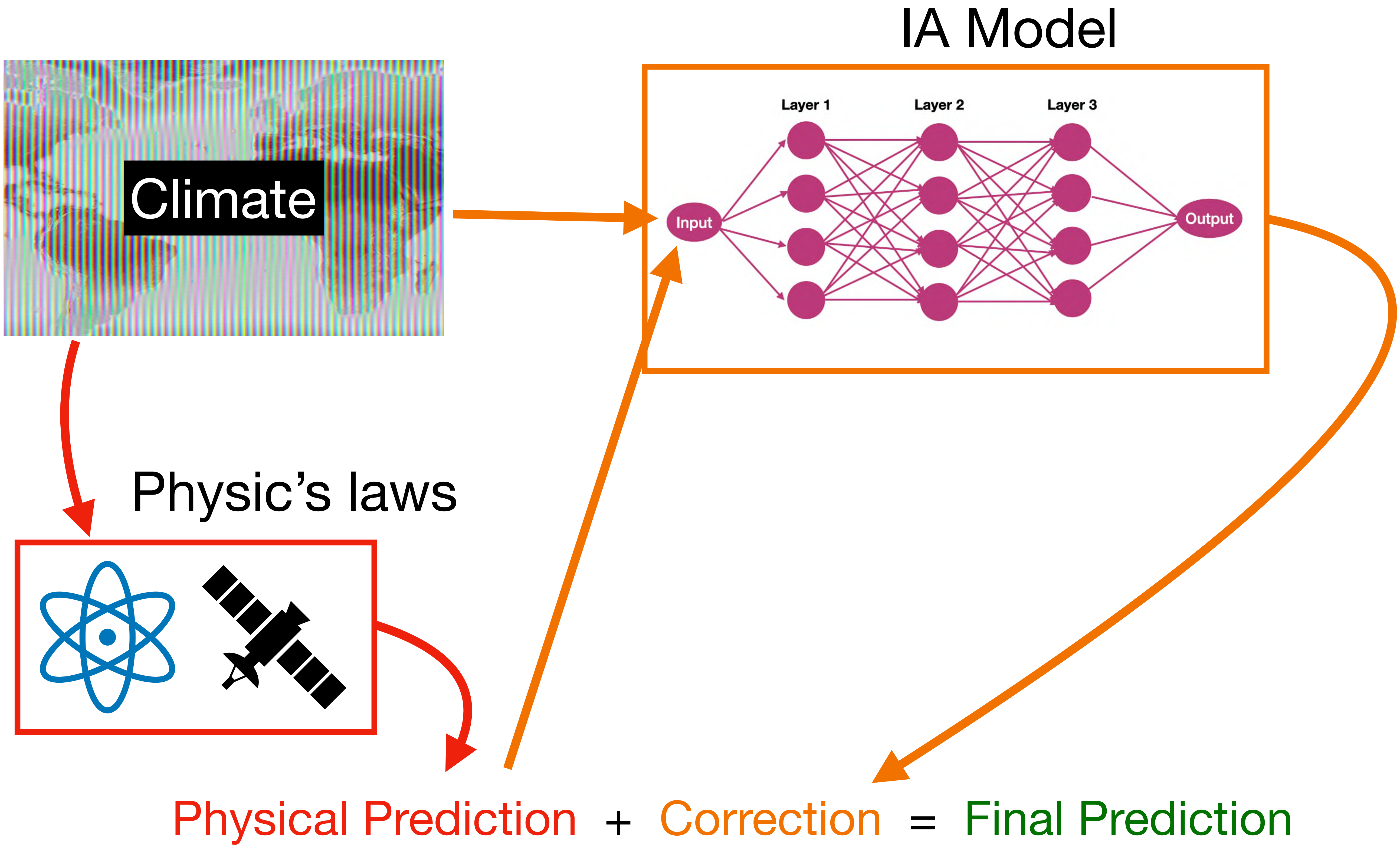
The use of AI model is a viable alternative,

but it still limited by the data used for its training.

Le physical model needs the knowledge of the laws behind the phenomenon.

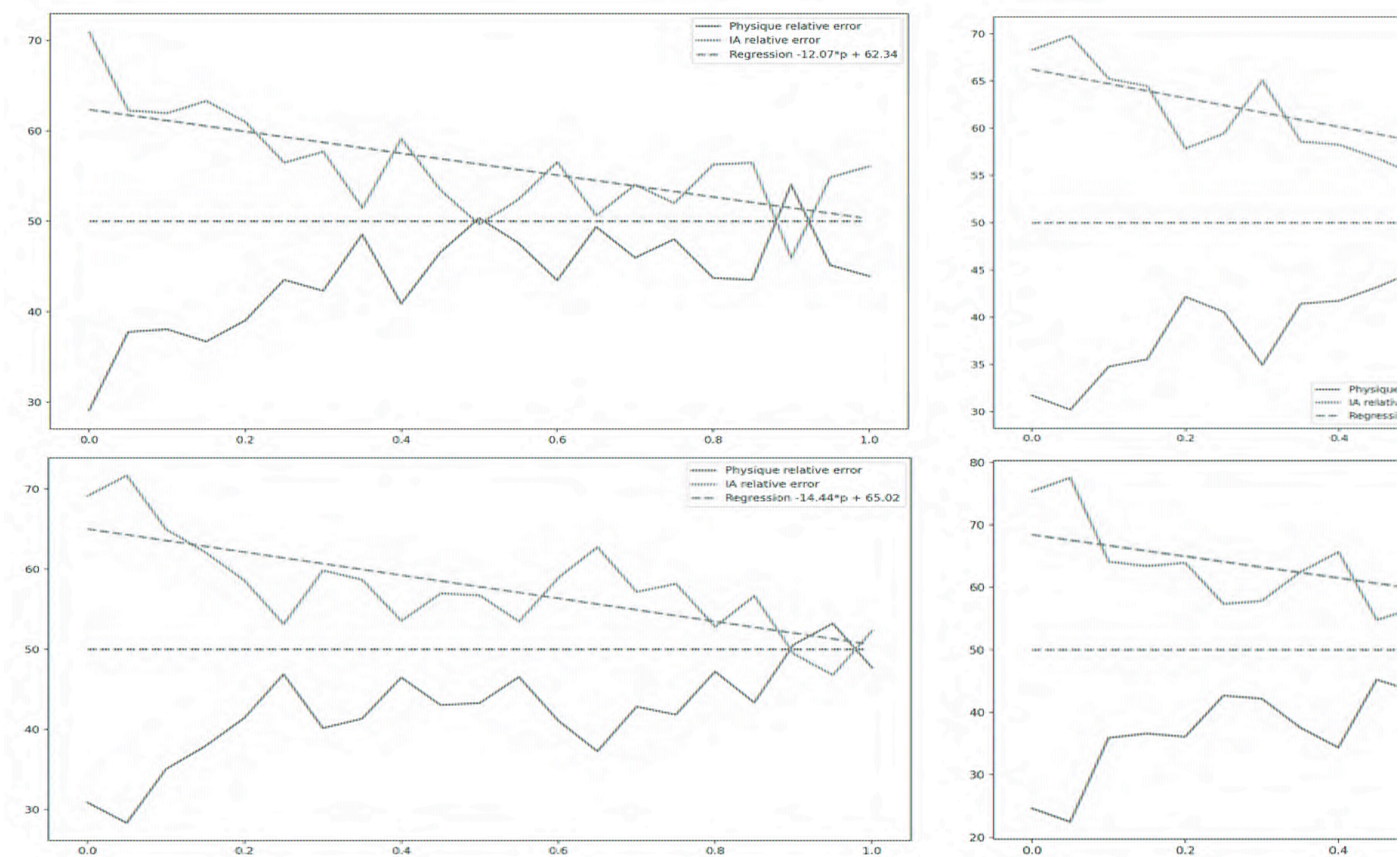
Conclusion

Physical Model supervised by AI



Appendix

Relative error as a function of the amount of data



Data a re-shuffled for each diagram

Appendix

Physical Model : Intergration

$$\vec{a}(t + dt) \approx \frac{\vec{v}(t + dt) - \vec{v}(t)}{dt}$$

$$\vec{v}(t + dt) \approx \frac{\vec{x}(t + dt) - \vec{x}(t)}{dt}$$

$\rightarrow$

Time Discrete

$$\boxed{x_{i+1} = x_i + v_i dt + a_{i+1} dt^2}$$

```
def getSpeed(traj, i):
    | dt = traj['t'][i] - traj['t'][i-1]
    | v = (traj['coord'][i] - traj['coord'][i-1]) / dt
    return v

def getAcceleration(traj, i):
    | dt = traj['t'][i] - traj['t'][i-1]
    | a = (getSpeed(traj, i) -  getSpeed(traj, i-1))/ dt
    return a
```

Appendix

```
def integrate(p, v, angle, dt, durée):
    | t = 0
    | trajectory = {‘position’: [p], ‘angle’: [angle], ’t’: [t]}
    | tant que t < durée:
    |     | a = force(v) / masse
    |     | t = t + dt
    |     | v = v + a * dt
    |     | p = p + v * dt
    |     | angle = angle + v_angle * dt
    |     | trajectory ← p, angle, t
    return trajectory
```

$$Poids\ :\ \vec{P} = mg\,\vec{e_z}$$

$$A\acute{e}ro\ :\ \vec{F} = \frac{1}{2}\rho S V^2(-\boxed{C_x}\vec{u} + \boxed{C_z}\vec{w})$$

Appendix

Drag and Lift coefficients

$$C_x = \frac{-2m(\dot{v} + g\sin(\theta))}{\rho S V^2}$$

$$C_z = \frac{-2m(v\dot{\theta} - g\cos(\theta))}{\rho S V^2}$$

```python
def cx(vitesse, acceleration, theta):
    return -2 * MASSE * (acceleration + g * sin(theta)) / (RHO_AIR * SURFACE * (vitesse ** 2))
def cz(vitesse, theta_point, theta):
    return -2 * MASSE * (speed * theta_point - g * cos(theta)) / (RHO_AIR * SURFACE * (vitesse ** 2))
```

Appendix

MLPRegressor : Training

```
regr = MLPRegressor(hidden_layer_sizes=network_size, random_state=3, max_iter=8000,
                    tol=1e-12, activation="logistic", solver='adam', learning_rate='adaptive',
                    shuffle=False, epsilon=1e-8).fit(X_train, y_train)
```

MLPRegressor : Auto-Regression

```
def autorégression(points_initiaux, dt, durée):
    | trajectoire = [points_initiaux]
    | t = 0
    | tant que t < durée:
    |    | t = t + dt
    |    | point_suivant = prédiction_IA(points_initiaux, t)
    |    | trajectoire ← point_suivant
    |    | points_initiaux = points_initiaux[1:] + point_suivant
```

Appendix

MLPRegressor : Normalization ‘min-max’

x = (x - x_min)/(x_max - x_min)

```
def normalisation(trajs):
  | x_max, x_min = max(trajs[‘x’]), min(trajs[‘x’])
  | y_max, y_min = max(trajs[‘y’]), min(trajs[‘y’])
  | pour chaque traj dans trajs:
  |    | traj[‘x’] = (traj[‘x’] - x_min)/(x_max - x_min)
  |    | traj[‘y’] = (traj[‘y’] - y_min)/(y_max - y_min)
```

Appendix

MLPRegressor

Function score :

$$score = \left(1 - \frac{u}{v}\right)$$

$$u = \sum_i (y_i - y_{i,\,predict})^2$$

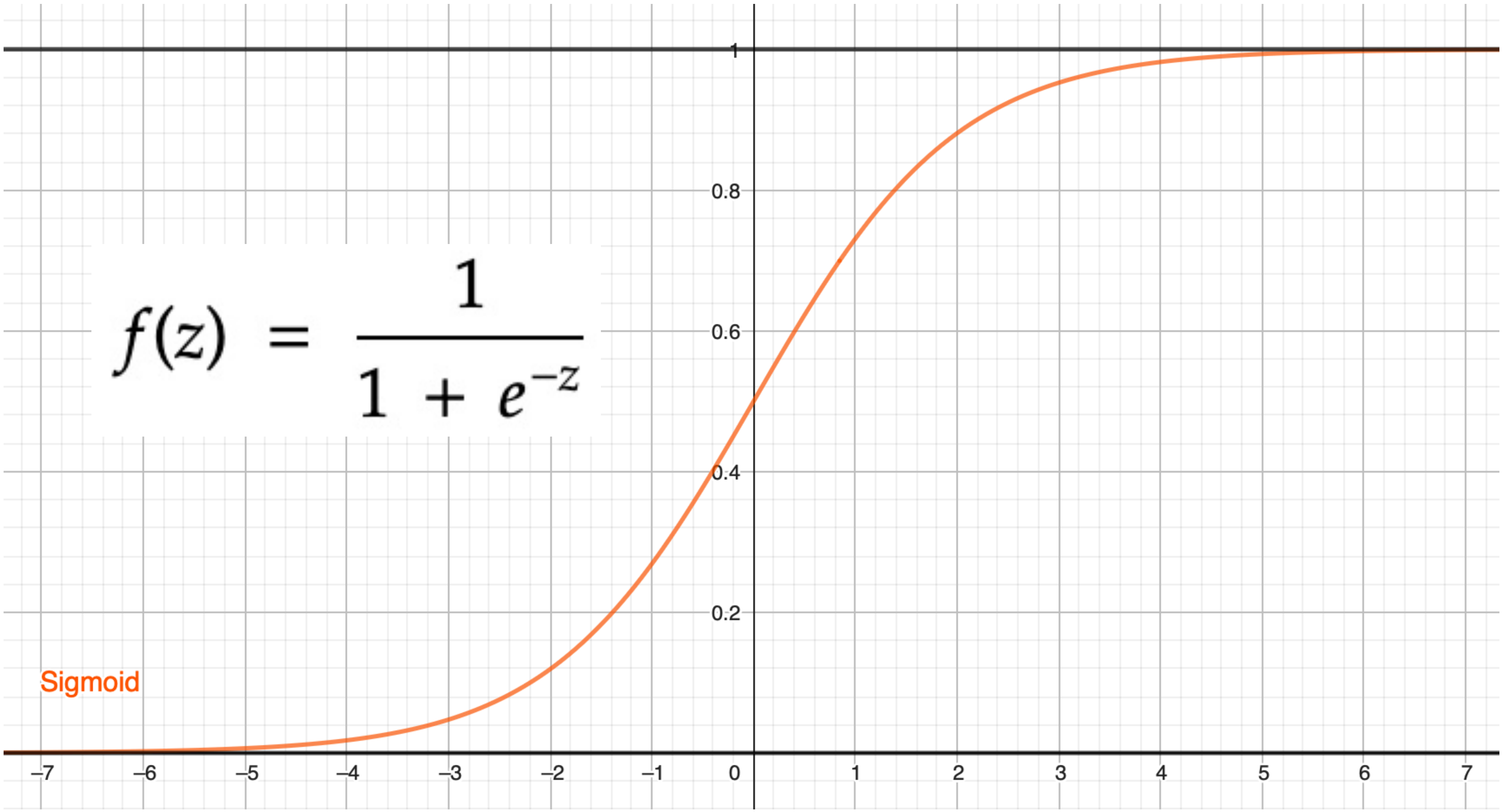
$$v = \sum_i (y_i - y_{i,\,moyen})^2$$

Function loss :

$$Loss = \sum_i (y_i - y_{i,\,predict})^2$$

Appendix

Logistic function use the MLPRegressor

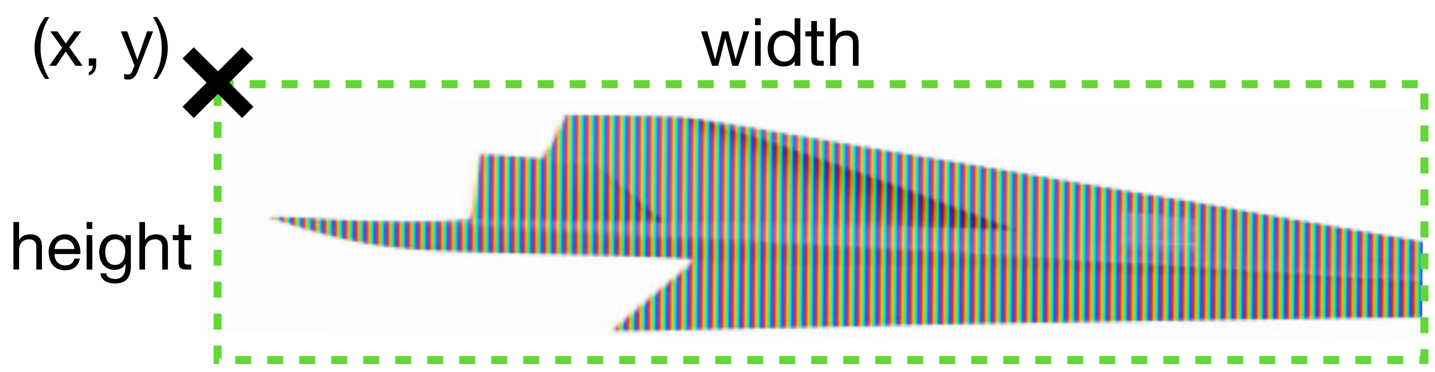


Appendix

video Analysis :

Bounding box

```
detections = model(frame, verbose=False)[0].boxes.data.tolist()
```



YoloV8

Yolo is a neural networks pre-trained in a wide range of data.

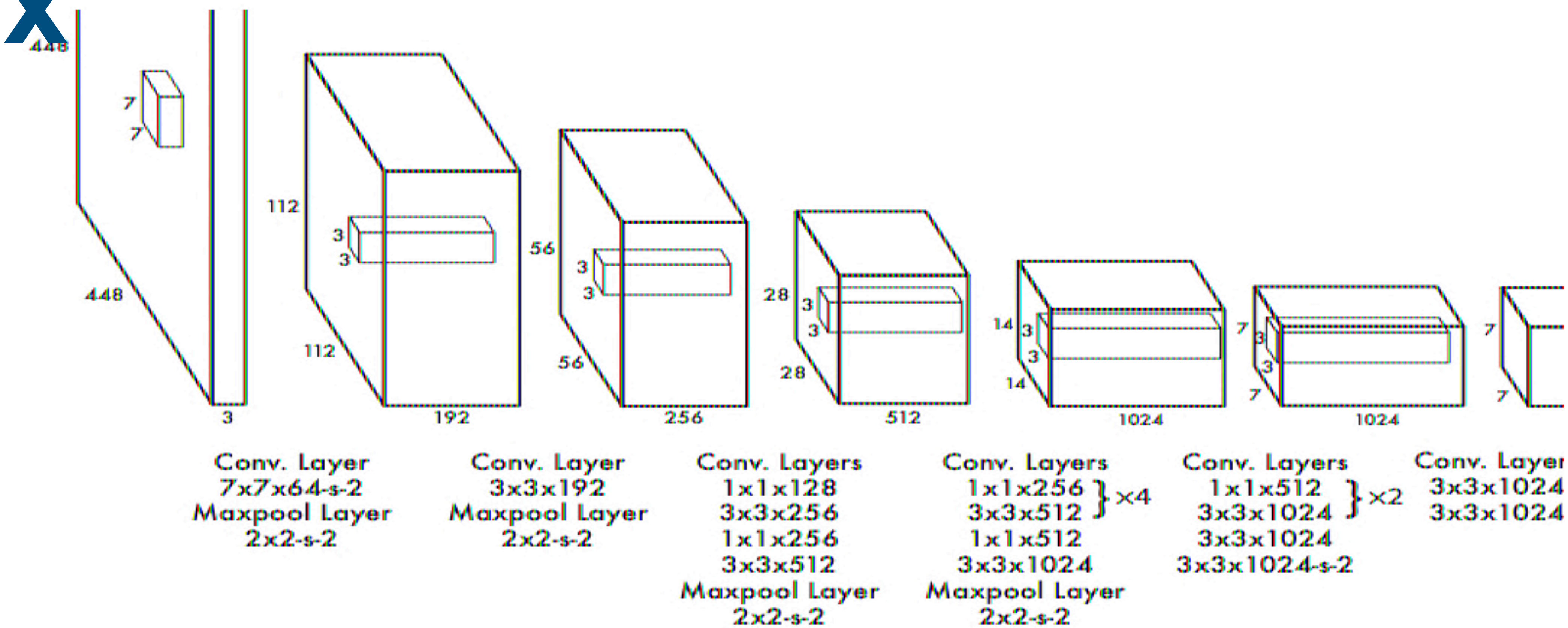
Ultralytics : https://docs.ultralytics.com

YOLO (You Only Look Once), a popular object detection and image segmentation model, was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, YOLO quickly gained popularity for its high speed and accuracy.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

Appendix

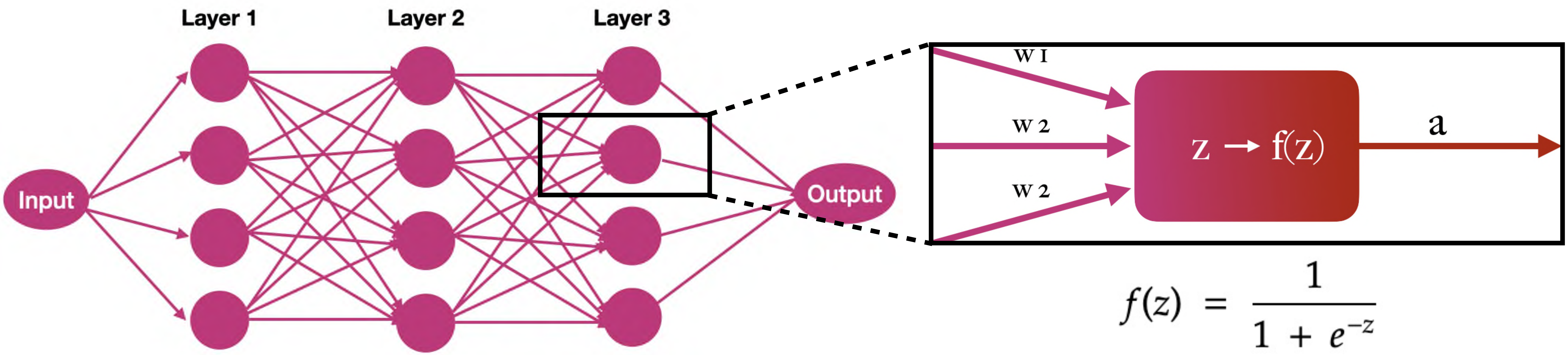
YoloV8



https://arxiv.org/pdf/1506.02640

Appendix

IA Models



$$f(z) = \frac{1}{1 + e^{-z}}$$

Different types of IA :
- Supervised (K-Neighbors)
- Non-Supervised (K-Mean)
- Reinforcement Learning