UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERIA ELÉCTRICA

MULTI-OBJECT TRACKING WITH CAMERA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

ANDRÉS ATTILIO THOMAS BRIGNETI

PROFESOR GUÍA:
MARTÍN ADAMS

MIEMBROS DE LA COMISIÓN:
LEONARDO CAMENT RIVEROS
ANDRÉS CABA RUTTE

SANTIAGO, CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: ANDRÉS ATTILIO THOMAS BRIGNETI
FECHA: MAY 2019
PROF. GUÍA: MARTÍN ADAMS

MULTI-OBJECT TRACKING WITH CAMERA

In this work we will test object tracking algorithms for the application of problem of pedestrian surveillance, where given footage from a camera, we are interested in correctly assigning a label to each individual pedestrian in each frame, and maintaining this label through over time, minimizing the number of wrongly assigned labels and undetected objects.

For this purpose we will use Multi Target Tracking based on Random finite Sets algorithms, which use past estimates of the position of tracked objects to predict a multi target future state, while also managing the possibilities of new target births or the deaths of these targets. These algorithms were created originally for tracking targets which undergo simple manoeuvres and using detection's with high uncertainty. This drastically differs from our case where the detections have a high accuracy (>75%) and almost no noise, but the movement of the people is highly not linear and very unpredictable.

So our work will focus mainly in studding how this ones will perform in this completely opposite conditions. Ba Tuong Vo has an open library where many of this filters are implemented and so we will make use of it to analyze how this algorithms perform in our case of interest and comparing the results using traditional Computer Vision metrics, as well as *RFS* metrics.

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: ANDRÉS ATTILIO THOMAS BRIGNETI
FECHA: MAY 2019
PROF. GUÍA: MARTÍN ADAMS

MULTI-OBJECT TRACKING WITH CAMERA

En este trabajo se evaluarán distintos algoritmos de trackeo para el problema de seguimiento de peatones, donde teniendo un video obtenido de una camara de seguridad, nos interesa reconocer correctamente cada individuo a traves del tiempo, buscando minimizar la cantindad de etiquetas mal asignadas y objetos (peatones) no identificados.

Para esto se ocuparán algorimos basados en el concepto de Conjuntos Aleatorios Finitos (Random Finite Sets - RFS), los cuales usan mediciones pasadas de los objetos para predecir posiciones futuras de todos ellos simultaneamente, mientras que también se consideran los casos de nacimientos y muertes de los objetos. Estos algoritmos fueron concebidos para el trackeo de objetos con movimientos simples y predecibles en condiciones de una gran cantidad ruido en las mediciones. mientras que las condiciones en las que se evaluarán son drasticamente opuestas, con un nivel muy alto de certeza en las mediciones pero con movimientos altamente no linear y muy impredecible.

Se ocupará una libreria abierta creada por el investigador Ba Tuong Vo, donde están implementados varios de los más clásicos algoritmos en esta área. Es por esto que el trabajo se basará más en el análisis de los resultados en estas nuevas condiciones y observar como se comparán a los algoritmos actuales del area de *Computer Vision* (**CV**)/ *Machine Learning* (**ML**), usando tanto métricas de RFS como del área de CV.


THE WHOLE SOLAR SYSTEM NEEDS READJUSTING...
WHAT CAN WE, AS INDIVIDUALS, DO?

Acknowledgements

Quiero agradecer a mi familia y los amigos que conocí, la gente del DIM, los urzares de la muerte, el grupo de Kung Fu, a la gente pulenta que me enseñó a ver el mundo de otra manera, y en especial al grupo de la pasta que han estado desde el principio de esta aventura y quien sabe que otro rumbo hubiera tomado esta sin ellos, solo se que alfinal no son las notas o los ramos aprobados lo que quedan, solo las amistades y no pude haber elegido mejor grupo. Igual de importante son y serán YBO que han estado desde siempre, en los altos más altos y en los bajos más bajos.

Contents

List of Tables

List of Figures

Introduction

Inside the world of automatizing processes in industries, we have seen a strong influx of implementations based in Machine Learning (**ML**) in the last years, even some that until now were considered more of a research subject. In particularly, the work with images, known as Computer Vision (**CV**), have been in the middle of this "revolution", the concept of deep learning and its outstanding results in the area have brought a new wave of interest in the idea of machine learning once again.
There have been two "classic" problems that have been studied exhaustible in CV, face recognition and pedestrian detection, both of which focuses in finding and recognizing people in static images. With the advance of technology, these problems evolved from working with simple images to video and merging with each other, these are known as pedestrian tracking or re-identification. There are many approaches to this problem, mainly it can be worked as just a re-identification in each frame, comparing the new detection with a database of the previous assigned IDs, Siamese Nets, Triad Nets, Generative Adversarial Neural Networks (**GANs**), feature descriptors, object segmentation, etc... or it can also be worked as a temporal series using trajectory estimators like Long-Short Term Memory Networks (**LSTM**), Random Finite Sets (**RFS**), etc... and its based on the idea of using information of previous states to estimate where to find the person or object previously detected, and not to just detect and re-identify each person in each frame.

One of the biggest problems when working in object tracking in video is occlusion, the fact that in any moment of the trajectory, a tracked target can go behind structures or other objects, and because of the positioning of the camera, it will not be able to be detected visually. This can generate a number of different errors, from the lost of the tracked object, miss match identification between objects that have crossed paths or even to incorrectly assign a id because of the trajectory estimation was wrongly calculated.

In particular, in this work we will test a solution based on Random Finite Sets (**RFS**). As the name suggest, the main idea behind it is to estimate the future position of the objects considering multiple different possibilities of solutions and choosing the most suitable one, which has to be corroborated with an actual detection of the object using a separated algorithm. There's a great amount of algorithms used to detect objects, as mentioned before, it is a very well documented problem, specially for the case we are interested in that's the detection of people. For the tracking method a more probabilistic approach is also going to be used in this work.

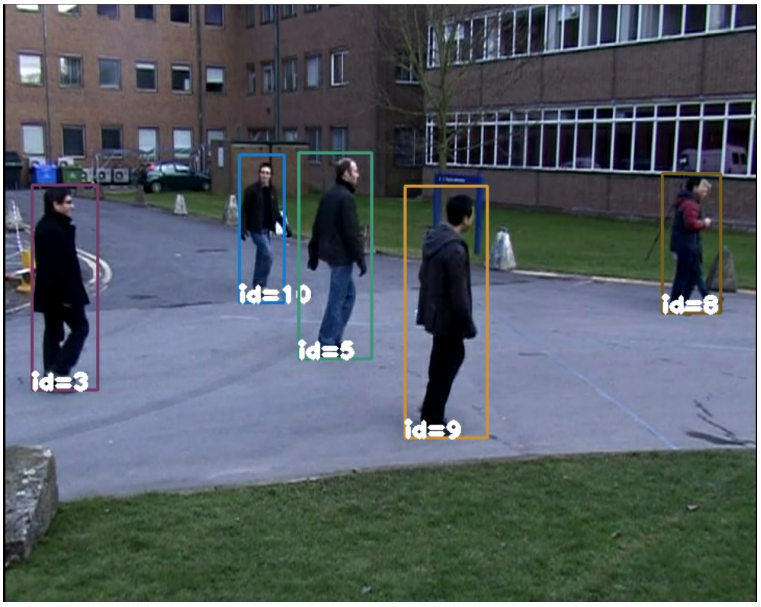


Figure 1: Example of detection in video.

The approach of *RFS* used is based on Multi-Bernoulli distributions, which in more a general way, what it does is to calculate the probability distribution of survival, death, being born and detection for each object, and how to realize the "re-identification" of "tracking" using this probabilities distributions. There is a hand full of this algorithms, δ-generalized labeled multi-Bernoulli filter (δ-**GLMB**), Poisson multi-Bernoulli mixture filter (**PMBM**) and the Probability Hypothesis Density Filter (**PHD**) have been the more cited, and in this work they will be studied with simulated and real data.

Chapter 1

Theorical Framework

1.1 Bayes filter.- Random finite sets

1.1.1 Bayes filter

A Bayes filter an iterative algorithm used for the estimation of a value using the Bayes rule shown in equation 1.1. The prediction is made from previous steps knowledge call prior information and the likelihood of the model, which means the probability of taking value given how the system is expected to work. In the case of tracking x_t could be the position and velocity of the tracked object in time "t", u_t is the action taken by that object, and $bel(x_t)$ is the believe or estimation of the values, as can be seen in equation 1.3 and 1.4, there are two of this values, $\overline{bel}$ for the prediction and bel for the correction of this prediction, the correction is done by a adding measurements presented by the variable z_t. This then iterate as seen in equation 1.3, where the prior information for the prediction is the believe correction from the previous time step.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1.1}$$

$$\tag{1.2}$$

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})\, bel(x_{t-1})\, \mathrm{d}x_t \tag{1.3}$$

$$bel(x_t) = \eta\, p(z_t|x_t)\, \overline{bel}(x_t) \tag{1.4}$$

In here the Bayesian approach is to generate a posterior $\hat{X}$ state based on the prior X state and the new detection Z using the Bayes rule, its used to generate an estimation and a correction in the estimation of a value, this is what is called a Bayesian filter.

$$\pi(X|Z) = \frac{g(Z|X)\pi(X)}{\int g(Z|X)\pi(X)\delta X} \tag{1.5}$$

$$\tag{1.6}$$

Where $\pi(X|Z)$ is the posterior density, Z are the measurements, X is the multi-target state. $g(Z|X)$ the multi target density of measurement Z given the state X and $\pi(X)$ a prior multi target density. The Bayesian approach is used because it can give you an optimum solution under the right circumstances.

1.1.2 Gaussian Mixture Solution (GMS)

A Gaussian mixture model or solution is maybe the simplest Bayes filter, where the *bel* is just a lineal combination of Gaussian distributions, where a general Gaussian distribution is defined in the equation 1.7, x is an vector of length N, Σ is the variance-covariance matrix of all N-dimensions of the distribution and μ is a point in space that represents its mean.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{1.7}$$

1.1.3 Random finite Sets (RFS)

When tracking multiple objects many problems can arise, the fact that the amount of targets can changes constantly, from occlusion to objects leavening the scene or noisy measurements that can generate false positives, makes extremely complicated to have precise estimations of where targets are and how they move through time. A way to model this problem is presented in [25] that will be used in this work based in the concept of Track Before Detect.

The general idea is that by having an X set state that describe the *RFS MTT* position of objects in a frame, we can compute an amount of different possible future outcomes, which are called updated states, from this we will choose the most probable to occur. Also we have the detection made in a new frame that we will call Z, and by comparing them with the predictions done with X, we will have a matrix of correlation between all of them. Once this matrix is build, there are two different ways to assign the correct object with its detection: Murty and Gibbs.

For our case, because we are working with multiple targets to be tracked, X will be a multi-target state. So in a future step, when calculating the new possible states of the objects, all of the possible combinations are to be considered, and so the number of multi

target hypotheses will be way to big to compute. For example if we have 5 objects and each one have 7 possible states to be in the next frame, the number of combinations to result is $5^7 = 78125$ different results, and this case is not taking in consideration the cases of new object detected, objects not been detected nor objects leaving the frame. So for making it possible to calculate it, some decision rules are implemented to reduce the amount of calculation.

Citing [25]: "The random finite set (RFS) approach provides an elegant Bayesian formulation of the multi-target filtering/tracking problem in which the collection of target states, referred to as the multi-target state, is treated as a finite set". Also if we would keep all the chosen states from previous iterations, and build new ones for each one of this, the number of hypotheses would grow exponentially making it impossible to calculate after some iterations. That's why this framework is usually assumed as a Markov process, meaning every state is dependant only of the previous state, which is coherent with the usage of Bayes' rule, but mainly that the number of possible states doesn't explode.

The multi-target states X, which represent the kinematic state (position and velocity) of the objects through the video, is obtain using Kalman filters, which will be explained in more detain in the next subsection. But in a general way they are recursive algorithms that estimate the real value of variables based on a combination of predictions and that that is assumed to be noisy.

The concept of RFS based in multi-Bernoulli filter has been studied intensively in many works [11] [14] [7] [22] [24] [6], who have not only proposed some implementations, but have also worked deeply on the mathematics behind its application, with mathematical propositions, proofs and corollaries on the behaviour of the filter.

Probability Hypothesis Density (PHD)

Presented originally in [10] by Zhe Chen, where is propose the propagation of a first-order statistical moment of the multi-target posterior. This moment, the probability hypothesis density (**PHD**), is the function whose integral in an region of state space is the expected number of targets in that region. They derive recursive Bayes filter equations for the PHD that account for multiple sensors, non-constant probability of detection, Poisson false alarms, and appearance, spawning, and disappearance of targets. They also show that the PHD is a best-fit approximation of the multi-target posterior in an information-theoretic sense. This filter can be interpreted as a multi-target statistical analog of a constant-gain Kalman filter and is it how is used for the estimation of new states.

The implementation used in this work is the Gaussian mixture Probability Hypothesis Density Filter (**GMPHD**) [18] done by Ba Tuong Vo et al using a recursive algorithm:

"This work shows that under linear, Gaussian assumptions on the target dynamics and birth process, the posterior intensity at any time step is a Gaussian mixture. More importantly, closed form recursions for propagating the means, covariances and weights of the constituent Gaussian components of the posterior intensity are derived. The proposed algorithm combines these recursions with a strategy for managing the number of Gaussian components to increase efficiency. This algorithm is extended to accommodate mildly nonlinear target dynamics using approximation strategies from the extended and unscented Kalman filters".

1.1.4 Delta-Generalized Labeled Multi-Bernoulli Tracking Filter (δ-GLMB)

The *δ-GLMB* filter presented in [5][20][21], is a type of labeled *RFS* which allows individual track labels for targets, not only recognizing individuals, but also knowing their trajectory. this label contains information of the time the target its born, and the birth region. For this a new discrete label space $\mathbb{L}$ is presented such that now a the *RFS* comes from the space $\mathbb{N} \times \mathbb{L}$. Here each element of an *RFS* is augmented with a unique label $l \in \mathbb{L} = \{\alpha_{\mathrm{i}} : \mathrm{i} \in \mathbb{N}$, where $\mathbb{N}$ denotes the set of positive integers and the α_{i}'s are distinct. The new states $\mathbb{X}$ are now defines as $\mathbf{X} \subset \mathbb{N} \times \mathbb{L}$, where $\mathbf{x} = (x, l)$. To obtain the label of a single-object a projection is defined as $\mathcal{L} : \mathbb{N} \times \mathbb{L} \to \mathbb{L}$ where $\mathcal{L}((x, l)) = l$ and is said that $\mathbf{X}$ have distinct labels if only if $|\mathbf{X}| = |\mathcal{L}(\mathbf{X})|$, that can calculated using the distinct label indicator:

$$\Delta(\mathbf{X}) = \delta_{|X|}(|\mathcal{L}(X)|)$$

A *δ-GLMB RFS* is a labeled *RFS* with state space $\mathbb{X}$ and a discrete label space $\mathbb{L}$, that distribute according to:

$$\pi(\mathbf{X}) = \Delta(\mathbf{X}) \sum_{(I,\xi) \in \mathcal{F}(\mathbb{L}) \times \Xi} \omega^{(I,\xi)} \delta_l(\mathcal{L}(\mathbf{X})) \left[p^{(\xi)}\right]^{\mathbf{X}}$$

Where Ξ is a discrete space while $\omega^{(I,\xi)}$ and $p^{(\xi)}$ satisfy:

$$\sum_{(I,\xi) \in \mathcal{F}(\mathbb{L}) \times \Xi} \omega^{(I,\xi)} = 1$$

$$\int p^{(\xi)}(x, l)\mathrm{d}x = 1$$

jointGLMB

the *joint-GLMB* is a faster implementation of *δ-GLMB* to reduce the amount of calculations as presented in [24]. This is done by having a joint prediction and update, as opposite of the

original algorithm that does a different calculation for each step, as explained in the paper: 'In contrast to the original implementation which requires different truncation procedures for each component in the prediction and update, the joint strategy involves only one truncation per component in the filtering density, thus drastically reducing the number of computations without affecting the filtering performance.'[19]

We will use this implementation through this work, so whenever we mention the δ-*GLMB* filter in the next chapters we will be referring to it.

1.1.5 Poisson Multi Bernoulli Mixture (*PMBM*)

For this work we will use a Poisson Multi Bernoulli Mixture (*PMBM*) filter, which differs from other random finite sets algorithms mainly in the way it manage new detection, where other algorithms generate a grid of Gaussian distributions through the space to represent the probability of a new object been born in there, the position of the Poisson distribution in the space is determined by previous births, so it can also help in reducing the mount of calculations.

In this type of implementations, the following parameters must be chosen: the targets survival probability p_s and a probability p_d if they are detected. Also new targets can appear with intensity β and false positives detection can happen with intensity κ. This last one is assumed constant to make the calculations easier, finally the existence probability r is used. In some cases this one can be considered a constant of value 1, to simplify calculations.

Each target will have a Kalman filter associated for predicting their future position, and it will be updated with new measurements every time a iteration of the *PMBM* is done and the detection in Z are assign to their respective objects in X.

with all this a matrix of size $(N+1 \times M+1)$ is generated, where the rows represent the new detection but the last row contains the values of the probability of a new object been born combined with the probability of been a false alarm, while the columns represent the live objects and the last column is the probability of not been detected or that disappear, the rest of the values represent the correlation between the new detection and the live objects. Once we have all this values calculated, we can proceed to assign the id's to the detections using the Moore or Gibbs implementation.

1.1.6 Kalman filters

We will use a Kalman Filter for the estimation of the new position of the tracked objects. A Kalman filter is a recursive algorithm that estimate the future value of variables [17] [3], based on the measured data over time. This estimation is made through a probabilistic approach, assuming a Gaussian distributions to represent the possible measurments and a linear model for the variables. Because it is still a Bayes filter, it is separated in two main parts, a prediction state and an correction state, both used to make a prediction of the future. Both of these two parts are regulated by a constant called Kalman gain, which try's to minimize the variance of the estimation by getting as much information as possible from uncertain measurements. For this work we pretend to use an already existing Extended Kalman Filter (**EKF**) implementation, found in Ba Tuong Vo library, taht we will explain how it differentiate from the general case afterwards.

Prediction

Given a n-dimentional Gaussian distribution $\vec{X_{k-1}}$ that represents the state of the tracked object in time $k-1$, we are interested in estimating its new state vector in the next time step, what we will call $\hat{X}_k$. Because it is a Gaussian distribution, it's defined by a mean $\vec{\mu}$ and a co-variance matrix P_k.

We define this prediction through the matrix F_k (This matrix can also be presented as "A" in the literature), which is presented in the eq. 1.8. This state transition matrix will represents the kinematics of the system as is defined previously, because it will not update or adjust though time, unlike $\vec{X}$. There are controlled factors inside the estimation, but as the name says, they exist if there is a way to control this variables internally, these are called control matrix B and vector $\hat{u}$. Finally we have w_k and Q_k which represent the noise in the estimations, specifically the predicted state noise vector and the process noise covariance matrix. For the covariance matrix of our estimation 1.9, is just the covariance of our previous state combined with the prediction and the process noise added to that.

$$\hat{X}_k = F_k \hat{X}_{k-1} + B\mu_k + w_k \tag{1.8}$$

$$P_k = F_k P_{k-1} F_k^T + Q_k \tag{1.9}$$

Update

For the update step we first have to understand the measurement input Y_k, as shown in eq. 1.10, is composed of two parts, a state vector X_k with a matrix transform C that defines what are we observing and finally Z_k that represents once again the noise or error, in this case of the measurement.

The idea is to combine the information contained in the measurements with the predictions to minimize the variance of the process. For this a constant called *Kalman Gain* is presented in the eq 1.11. This Value is a weighing between the confidence in the estimation and the measurement. Its result isn't very intuitive, because it comes from the solution of multiplying two multidimensional Gaussian distributions, but understanding its purpose as a controller between both the estimation and the measurement is more important, a simplification is presented in the eq 1.12.

$$Y_k = CX_k + Z_k \tag{1.10}$$
$$K' = P_k H_k^T (H_k P_k H_k^T + R_k)^{-1} \tag{1.11}$$

$$K = \frac{E_{est}}{E_{est} + E_{mea}} \tag{1.12}$$
$$\tag{1.13}$$

On the measurements we model the sensor with a matrix H_k (eq. **??**) the same way we modeled the estimation with the matrix F_k and the co-variance of the measurements with be R_k (eq.1.15).

$$\hat{X}_k' = \hat{X}_k + K'(Y_k - H_k \hat{X}_k) \tag{1.14}$$
$$P_k{}' = P_k - K' H_k P_k \tag{1.15}$$

In a more simple way, the update of both the state and the co-variance are a combination of the previous state and the how much the new information modifies it, as the simplified version is shown in eq. 1.16, where as mentioned before the Kalman gain K is used as a regulator depending on the certainty of each case.

$$\vec{X}' = \vec{X}_0 + K(\vec{X}_1 - \vec{X}_0) \tag{1.16}$$
$$\Sigma' = \Sigma_0 - K\Sigma_0 \tag{1.17}$$

Extended Kalman Filter (EKF)

The Extended Kalman filter is used in reality because unlike what the Kalman filter expects, the real world is not linear nor their measurements are Gaussian distributions. This is solved by the *EKF* by linearlizing locally the estimated function, to fit all the steps needed to solve the problem. This linearization is done by calculating the first order derivate in a point and

then studying how far from the linearization our measurements are.

this modifications to the real function causes the *EKF* to not be an optimal estimator, unlike the Kalman filter for the basic model. In any case it should allows a good estimation of the curve if the time discretization is fine enough.

1.2 Pedestrian Detection

The more modern approach for pedestrian detection comes from the area of deep learning, using deep convolutional networks able to copy up to certain level the way human eyes work. Starting from a defined graph structure, the network re-adjust its arcs and nodes values to optimize the association between an input, in this case images, and specific labels assigned to each image.

Inside this world, the problem of detection and classification of object classes have had an enormous weight on the direction this technology have taken. With modern architectures able to detect and classify hundreds if not thousands of different classes in ms. making of the old solution obsolete technology. Between this modern approach's we find You Only Look Once (**YOLO**), Faster R-CNN, Inception, ResNet, RetinaNet, etc... For this work, we will use *YOLOv3*[8] (Figure 1.3), that have shown a great combination of accuracy and execution time (Figure 1.1). Though its not the one with the best mAP, it still maintain great results while having a significantly lower time of execution.

YOLOv3 is a 53 layered Convolutional Neural Network (Figure 1.2) implemented over Darknet by Joseph Redmon, "*Darknet is an open source neural network framework written in C and CUDA.*" [8] which allows a faster and better implementation for neural networks. It can also allows for multiple inputs by loading the net and leaving it in stand by and then inputting different images, which works great with real time videos, like security surveillance. It can also be retrained o readjust its parameters for a better performance in a specific case. In particular for the detection in this work, we will filter them to get only the ones with a 'person' label, and also the threshold of probability of each class is set in 75%, meaning that any prediction bellow this threshold will be discarded by *YOLO* itself.

1.3 Evaluation & Datasets

There are a number of different proposed methods on how to measure the performance of visual tracking, and so, it's performance can be change drastically depending on way its done. Multi-object filter has their own specific metrics: **OSPA** and **OMAT**[16] [13] that in first instance we will not work with, because their are not useful for comparing the results with

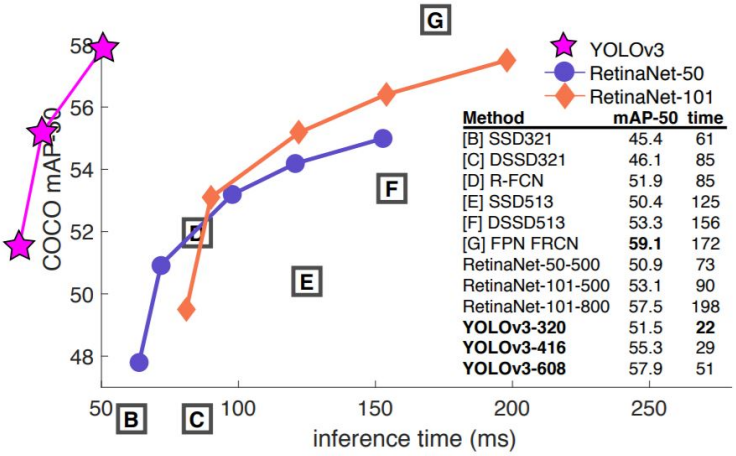


Figure 1.1: Graph comparison with RetinaNet.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 1.2: *YOLOv3* Arquitecture.

general tracking systems. For evaluating the performance of the software, we will use the The Multiple Object Tracking Benchmark given by the *MOT challenge* [2], who have worked on the standardization of multiple target tracking evaluation. They don't only supply the

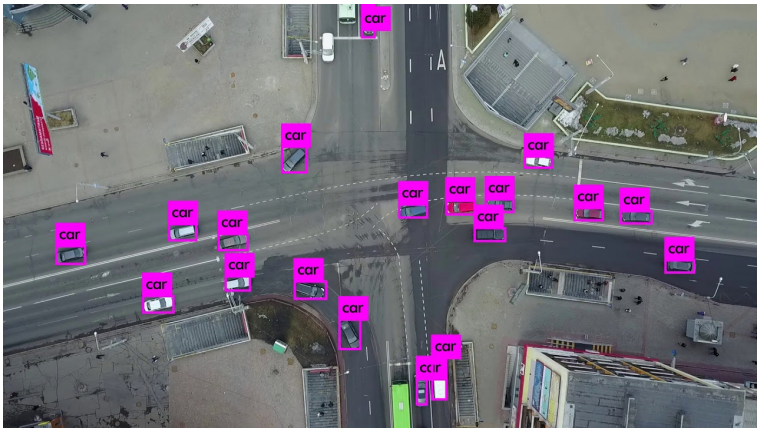


Figure 1.3: Example of *YOLOv3* detection.

most known data sets of surveillance videos and their ground truth, like PETS2009[1], but also standardize the metrics used to measure the performance. We will use the this data set as a basis and use the metrics defined by the benchmark[12], from which the most important ones are the following:

1.3.1 Error Types

The main types of error that can be generated during the tracking of multiple objects is shown in the figure 1.4 presented in [4], where the small circles h_{i} are the detection, its color represent its true label, and the color areas 0_{i} are the estimations for each time: (a) If the distance between o_1 and h_1 is bigger than a threshold T, it is not possible to keep assigning the same label, even though in reality are the same object, generating a false positive of a new object. (b) The near movement of objects can create the illusion of intersections that are not actually occurring, generating mismatches in the labels, what we will be calling an id switch. (c) An example of a false negative is shown, where two different objects (h_1 and h_2) are label as the same because they share a common path. (d) Finally a different case of False Positive is shown where after a miss, a new object is detected closer that the actual true one, which because of the miss has a distance bigger than expected from the estimation, generating a mismatch and calling the old object a new detection.

Number of ID Switches (IDSW)

$$IDSW = \mathrm{N^o} \text{ the Switched IDs}$$

This value represents the number of miss matches on the Id assignation.

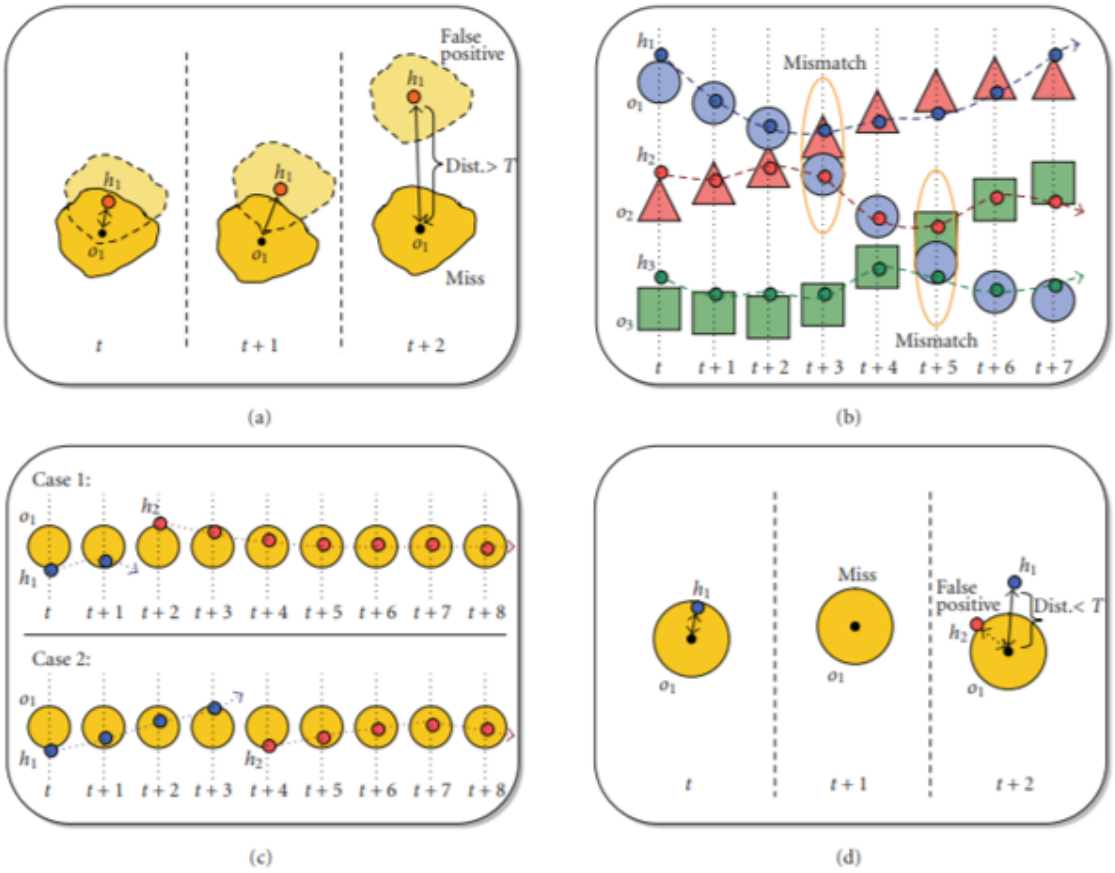


Figure 1.4: Different types of errors in multi-object tracking.

Multi Object Tracking Accuracy (MOTA)

$$MOTA = 1 - \sum_{t=1}^{n} \frac{(FN_t + FP_t + IDSW_t)}{GT_t}$$

Where t is the frame number, FN_t denotes the False Negatives in the frame t, FP_t denotes the False Positives, $IDSW$ is the Number of ID Switches defined before and GT_t is the number of tracked objects defined on the ground truth of that frame. The range of $MOTA$ is $(-\infty, 100\%]$.

Multi Object Tracking Precision (MOTP)

$$MOTP = \sum_{t=1,\mathrm{i}=1}^{n,I} \frac{\mathrm{d}_{t,\mathrm{i}}}{c_t}$$

Where c_t denotes the number of matches in frame t and $\mathrm{d}_{t,\mathrm{i}}$ is the bounding box overlap of target i with its assigned ground truth object. $MOTP$ thereby gives the average overlap

between all correctly matched hypotheses and their respective objects, and ranges between $\mathrm{d}_t = 50\%$ and 100%.

False Alarm per Frame (FAF)

$$FAF = \frac{\sum_{t=1}^{n} FA_t}{N}$$

This value represents a mean defined by the number of false detections (FA_t) that ocurr per frame, divided by total amount of frames, represented by N.

Optimal Sub Pattern Assignment (OSPA)

This metric is not obtain from the MOT benckmark, but presented in shumacher et al. work [15] an referenced in works like [16] [13], which is used specifically for random finite set implementations of multi-object trackers. *OSPA* metric measures the distance between two finite sets of target states denoted by $\mathrm{d}^{(c)}(x, y) := min(c, \mathrm{d}(x, y))$ where $c > 0$ is a limit for a max distance, and $x, y \in W$ a closed and bounded observation window. Then considering Π_k the set of permutations on $\{1, 2, 3, ..., k\}$ for any $\mathrm{k} \in \mathbb{N} = \{1, 2, ...\}$. For $1 < p < \infty$, $c > 0$ and arbitrary finite subsets $X = \{x_1, ..., x_m\}$ and $Y = \{x_1, ..., x_n\}$ of W, where $m, n \in \mathbb{N}_0 = \{0, 1, 2, ...\}$:

$$\bar{\mathrm{e}}^{(c)}_{p,card}(X, Y) := \left(\frac{1}{n} \left(\min_{\pi \in \Pi_n} \sum_{\mathrm{i}=1}^{m} \mathrm{d}^{(c)}(x_\mathrm{i}, y_{\pi(\mathrm{i})})^p + c^p(n - m) \right) \right)^{\frac{1}{p}}$$

This metric can be divided in its location $\bar{\mathrm{e}}^{c}_{p,loc}(X, Y)$ and cardinality $\bar{\mathrm{e}}^{c}_{p,card}(X, Y)$ which are defined as next:

$$\bar{\mathrm{e}}^{c}_{p,loc}(X, Y) := \left(\frac{1}{n} \cdot \min_{\pi \in \Pi_n} \sum_{\mathrm{i}=1}^{m} \mathrm{d}^{(c)}(x_\mathrm{i}, y_{\pi(\mathrm{i})})^p \right)^{\frac{1}{p}}$$

$$\bar{\mathrm{e}}^{c}_{p,card}(X, Y) := \left(\frac{c^p(n - m)}{n} \right)^{\frac{1}{p}}$$

In the paper it was proposed, its mention that although the decomposition is not necessary for generation a metric of comparison between results, it may be useful for a better understanding of the results, as it show how the values evolve through time. We will also compute the mean of the distance though time, so we can have a simple value to make comparison between these.

1.3.2 Data Base

PETS2009

The videos used are taken from the *PETS2009* data set [1] (Figure 1.5), which gives a great amount of different options of videos for tracking people though cameras, intended for different problems which include: counting, general tracking, tracking inside of crowds, and more. videos work at different camera ratios, positions, angles, lighting and also has cameras with overlapping fields of view. We will work with the video *S2L1* which works great as security cam from height and some occlusion generated by physical object in the scene rather than just the natural occlusion generated by the people transiting through the place.

The specifications of the video are shown on the table 1.1. The N of trajectories doesn't measure the number of people that appear on the video, but the number of time a pedestrian walks in frame, so one person can be count more than once if he walk out of frame and back. The Density value is the average number of pedestrian per frame.

Name	**FPS**	**Resolution**	**Length [Frames]**	**Length [minutes]**	**Tracks**	**Density**
S2L1	7	768x576	795	01:54	19	5.6

Table 1.1: *S2L1* video specifications.

Figure 1.5: Caption of video *S2L1*, from *PETS2009* dataset.

Towncenter

The TownCenter database is also used for testing pedestrian tracking algorithms, released too in 2009 by the Active Vision Laboratory from Oxford University [9] with their results

of the "Coarse Gaze Estimation in Visual Surveillance" project. This video presents around 20 minutes of pedestrian walking though a street, mainly in the same direction of the street as can be seen in fig 1.6. The frame is from a surveillance camera with natural light, in almost the same conditions from the *S2L1* video from *PETS2009*. Other is that the id of each pedestrian is maintained during a whole trajectory inside the frame, so if a person walks out of frame and reappears is considered a new id, different from the one before. Finally the whole specifications of the database are presented in table 1.2.

Name	**FPS**	**Resolution**	**Length [Frames]**	**Length [minutes]**	**Tracks**	**Density**
TownCenter	2.5	1920x1080	450	03:45	226	15.9

Table 1.2: TownCenter video specifications.

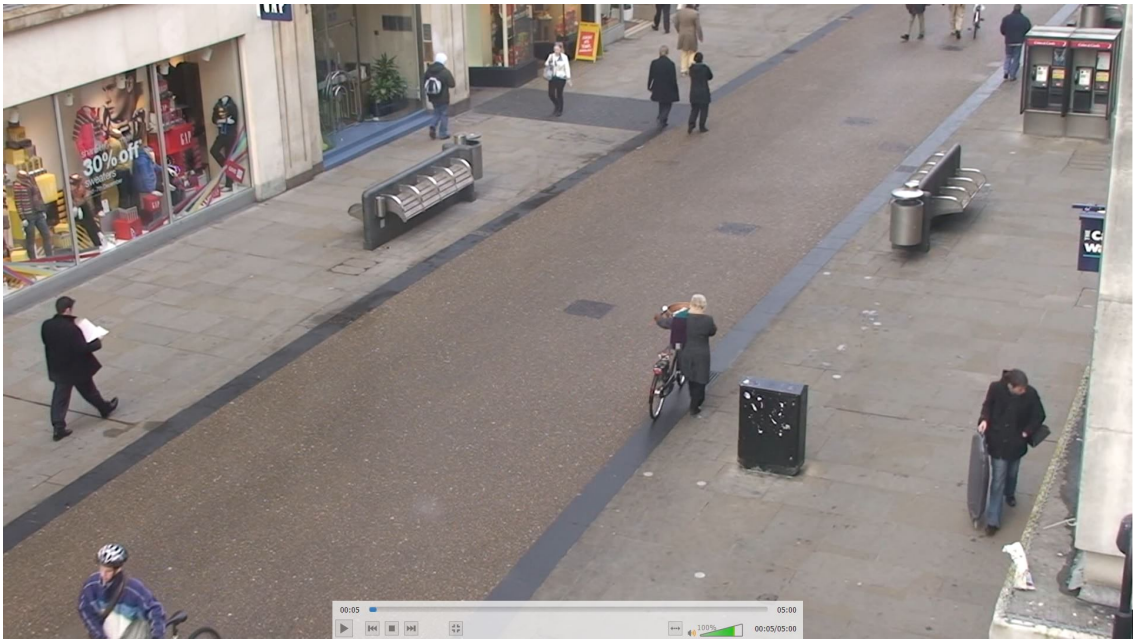


Figure 1.6: Example of frame from TownCenter data base.

Chapter 2

Methodology

2.1 General Specifications

As mentioned in the preview section, we will be working with the PETS2009 data set, particularly with the video *S2L1* which is a recording using an angle and position similar to security cameras, thou acting as an surveillance video. Making it the most suitable of all 8 videos of the data set for our work.

There is no official ground truth included in the *PETS* site, so an unofficial data set was used, which contain almost all of the trajectories available in the video, except for one that occurs in the background. It was extensively studied to assure it wont have any mistake.

Also this work will be realized on a notebook with a 7th generation i5 Intel CPU of 2.5GHz, 16 GB of DDR4 RAM and a NVIDIA Geforce GTX1050 ti (4GB) GPU in Ubuntu 18.04 . Because the library that Ba Tuong Vo has released is in MATLAB, most of the work will be done in this language, while the process of detection of the pedestrian in the video using *YOLOv3* will be done in Python, and after wards the data will be loaded in MATLAB.

Ba Tuong Vo, the main publisher in these area, has released in his personal site a code set of different tracking implementations [23], this are all wrote in MATLAB, so the main part of this implementation will be done in this programming language. This library contain multiple filters: *'singletarget'*, *'bernoulli'*, *'phd'*, *'cphd'*, *'cbmember'*, *'glmb'*, *'lmb'*, *'jointglmb'* and *'jointlmb'*. Each has different implementations: *'gms'*, *'ekf'*, *'ukf'*, and *'smc'*.

2.2 Implementation

Once the video is obtained, *YOLOv3* is used to get the detection, which are wrote in a *.txt* file, saving each detection position and the frame it was obtained. *YOLOv3* library comes with a Python wrapper, so its implementation is almost direct, only small modifications had to be made to it so the data needed could be obtain.

About the structure of the algorithms implemented by Vo, it is important to highlight what is commented in the library documentation:

- All linear Gaussian examples use a 4D CV model (x/y position and velocity) with 2D observations (position only).
- All non linear examples use a 5D CT model (x/y position, velocity and unknown turn rate) with 2D observations (range and bearing).

Our ground truth contains only position and its corresponding id, so based in that information, it was able to calculate their velocity and turn rate using some naive heuristics, mainly the difference between a value and its value in the previous frame.

The main problem of this implementation came in that the models are used originally for the tracking of space derby or objects that have a more linear movement, while human behavior can be completely erratic. Specially in the video used, where people movement was intentionally stranger than what can be expected from pedestrian. This can be seen in figures 2.1 and 2.2, where the lines are the true trajectories of objects and the grey crosses *'x'* are the measurements generated. In this case, figure 2.1 show an example generated by the library, of data similar to what the algorithms and models were created for. While the figure 2.3 show the ground truth of the video *PETS2009 S2L1*, which has a completely non-linear movement. The way of reading this last figure, is that circles represent the birth of an object, and the triangles its death, while the black line connecting them is the trajectory of that object. As one can see, there are some cases that doesn't end on a triangle, this particular objects appear at the end of the video and it is cut before they reach the end of their trajectory.

Also are presented the number of existing trajectories in each frame (figure 2.4) and the *OSPA* metrics (figure 2.5). So the main work will be the modification of the model to adjust it to this problem, making it able to use the tracking implementations in the data set and the implementation of the metrics defined in the theoretical framework. The *OSPA* output shown in figure 2.5 shows first (top) the value of the distance on each time step, while the other two are the decomposition explained in the previous chapter. We will use this plot together with the mean of their values as a tool to compare and understand the results obtained.

The model we work with consider not just the movement of already existing objects, but the probability of birth and death of them, and all of this must be adjust for each new case so the model can get the best results. In this case, the model generates four Gaussian distribu-

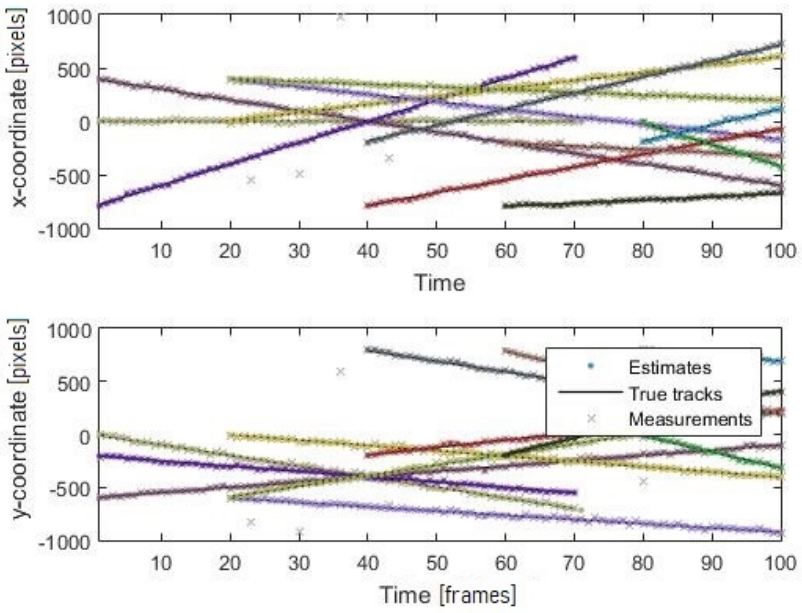


Figure 2.1: Tracker example generated by the library.

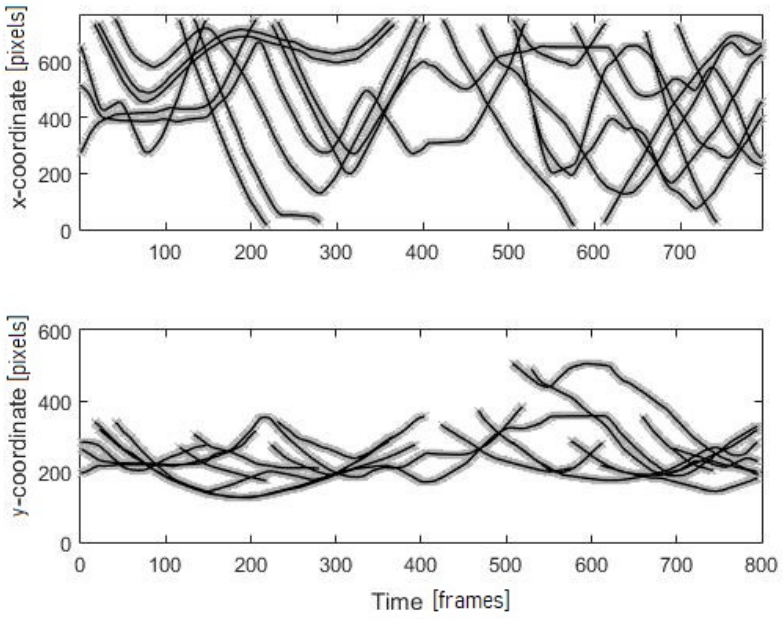


Figure 2.2: Video Ground Truth through time.

tions, each of 4 dimensions, which represent the possible position of birth of each new object, with its respectively velocity and probability distribution, as so its variance, co-variance matrix and its noise. This Gaussian distributions must be correctly laid on the sample space, so the model can correctly detect any new object, otherwise it will not only generate more false negatives by not detecting any new birth, but also generate more false positive, by considering objects already been tracked as a new birth, generating also id switches. In the figure 2.7 is shown a representation of where the Gaussian distributions are placed in the

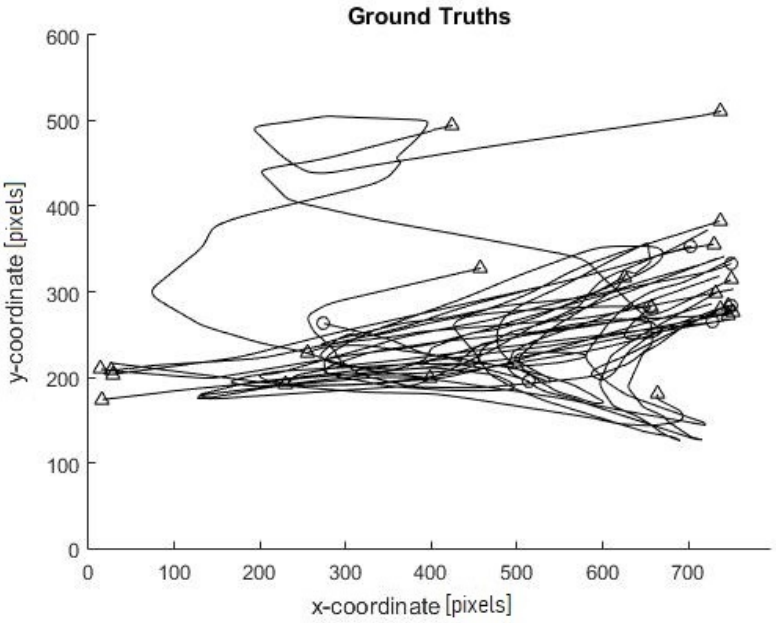


Figure 2.3: Video Ground Truth in 2D.

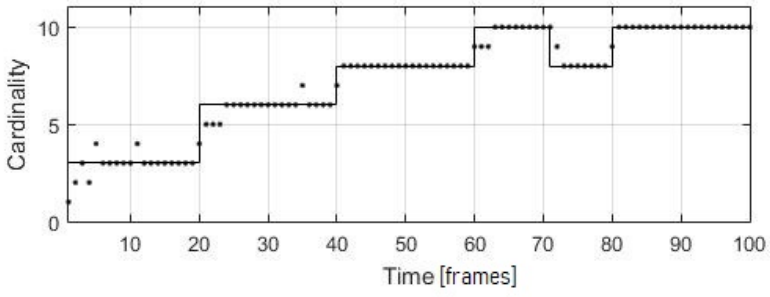


Figure 2.4: Cardinality of estimation compared to ground truth.

demo example, as you can see they are centered in points with a high amount of actual birth occurring around it.

The code presented below is part of the MATLAB file that defines the model we work with for *GMS*, which returns a model object containing the general parameters of the model we will work with. As mentioned before, expects that the ground truth values are represented as a vector of 4 scalar components in the following order: x-axis position, x-axis velocity, y-axis position and y-axis velocity. For the measurements only the position is expected, and this is defined in lines 4 and 5 as *model.x_dim* and *model.z_dim*. Next the Dynamic Model Parameters are defined, some of this will be the parameters we will have to adjust for our data to get better results, mainly the sampling period and the sigma velocity, defined as *model.T* and *model.sigma_v*. The Survival parameter is assumed ~ 1, and so the death parameter ~ 0, this because of the type of problem we are working with, witch has almost no error in detection and pedestrians don't suddenly disappear inside of the frame, only when

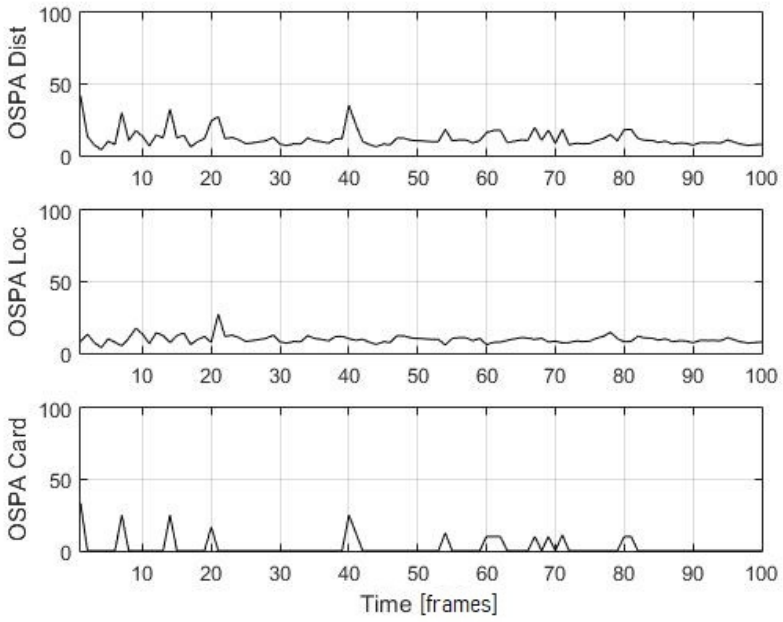


Figure 2.5: *OSPA* results per iteration for demo.

some occlusion may occur, but these are not deaths, but failed detection, the death of an object means no more detection of him, essentially that it is not in frame anymore, but that case is controlled buy another parameter afterwards. the survival ($model.P\_S$) and death ($model.Q\_S$) variables are defined in lines 17 and 18. In line 21 its defined the number of Gaussian Distributions that will be placed in frame for detecting and generating new objects as mentioned before, for this case, 4 Gaussians distributions are defined, the same amount as the example shown in figure 2.7.

```
1  function model= gen_model
2
3  % basic parameters
4  model.x_dim= 4;   %dimension of state vector
5  model.z_dim= 2;   %dimension of observation vector
6
7  % dynamical model parameters (CV model)
8  model.T= 5;                                    %sampling period [s]
       = fps
9  model.A0= [ 1 model.T; 0 1 ];                          %transition
       matrix
10 model.F= [ model.A0 zeros(2,2); zeros(2,2) model.A0 ];
11 model.B0= [ (model.T^2)/2; model.T ];
12 model.B= [ model.B0 zeros(2,1); zeros(2,1) model.B0 ];
13 model.sigma_v = 15;
14 model.Q= (model.sigma_v)^2* model.B*model.B';   %process noise
       covariance
15
16 % survival/death parameters
```

```
model.P_S= .999999;
model.Q_S= 1-model.P_S;

% birth parameters (LMB birth model, single component only)
model.T_birth= 4;          %no. of LMB birth terms
```

Finally some general parameters are assigned, the detection value (*model.P_D*) is set very close to 1, because we know from the detection obtain by *YOLO* that the probability of been an false positive is extremely low. The *clutter* parameter represents the amount of 'noise' in the detection in an area, but again because of the good performance of *YOLO*, this is considered to be as low as necessary, in comparison with a traditional problem for this kind of algorithms, like the example shown in fig 2.6, which has a clutter value orders of magnitude bigger. Lastly the *model.range_c* defines the margins of the frame in where the tracking will be done this will affect the results of the estimations too so they must be set to the size of the frame.

```
% detection parameters
model.P_D= .999999;   %probability of detection in measurements
model.Q_D= 1-model.P_D; %probability of missed detection in
    measurements

% clutter parameters
model.lambda_c= 0.1;                              %poisson average
    rate of uniform clutter (per scan)

%###%%
%model.range_c= [ -1000 1000; -1000 1000 ];      %uniform clutter
    region
model.range_c= [ 0 795; 0 800];       %uniform clutter region
model.pdf_c= 1/prod(model.range_c(:,2)-model.range_c(:,1)); %uniform
    clutter density
```

For the *EKF* model a directional value is added to the ground truth values, making its vector of 5 parameters instead of 4. Also for the measurements the positions is expected to be in polar coordinates, while the *GMS* works with a Cartesian system. The same happens with the *model.range_c*, where the boundaries must be defined in polar coordinates.

Apart from the adjustment of the parameters for our new data bases, different sensibility test are made with it as so with the original examples. For the first one its proposed to see at which level of clutter the different filters stop working, this is because if we are proposing to check how they work in new conditions we need to understand first their own limitations, so the clutter will be moved in both directions, higher and lower to see how it changes the results of them. Next we will evaluate different values of *clutter* and *sigma_v* to observe the how it vary the results, so we don't only have a final result of finely tuned parameters, but to understand their relevance in how this algorithms work. The clutter represent the amount of noisy detection or false positives occurring in the measurements, while the sigma_v is the

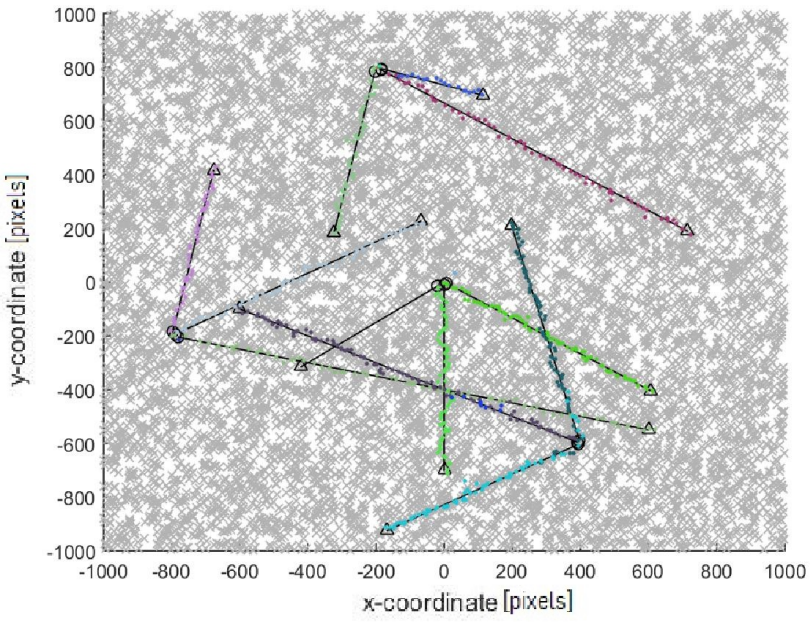


Figure 2.6: Example with clutter value of 100.

constant of acceleration for the moving objects assumed for the model.

Finally for the implementation of the new metrics, as said before, they are mainly counting the number of false positive and false negatives generated through the test, so is a comparison between the actual label and the one given in the ground truth. Also for defining if a label was correctly assign we are setting a minimum distance as shown in figure 1.4 *(a)* and *(d)*, so the value of this thresholds will affect the results of the calculations and will be a point of consideration in the adjustments.

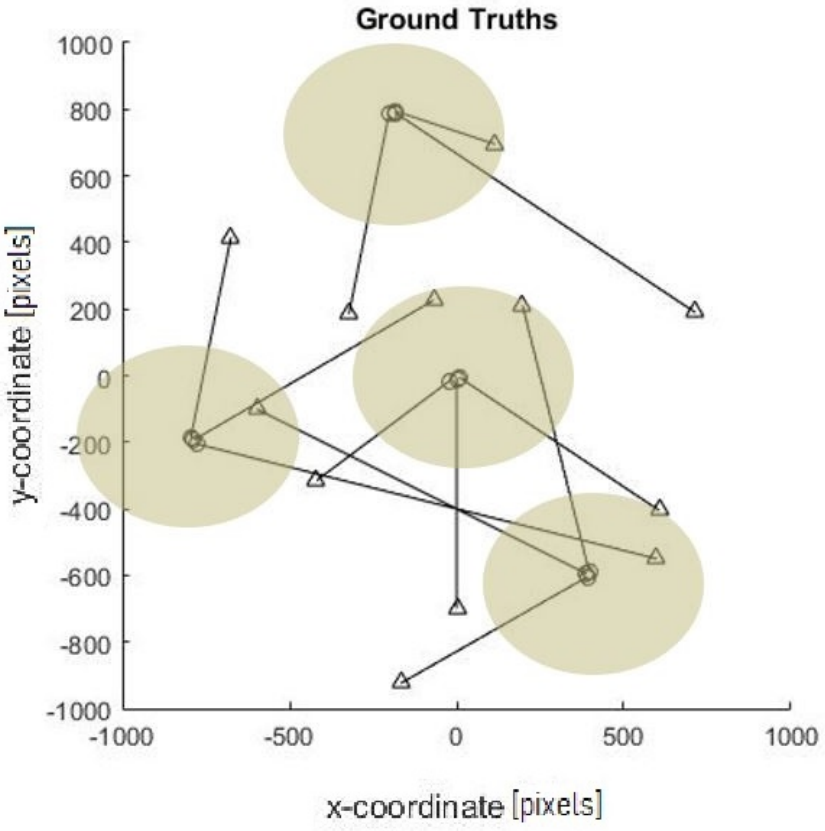


Figure 2.7: Representative example of Gaussian distributions.

Chapter 3

Results

3.1 New Visualization

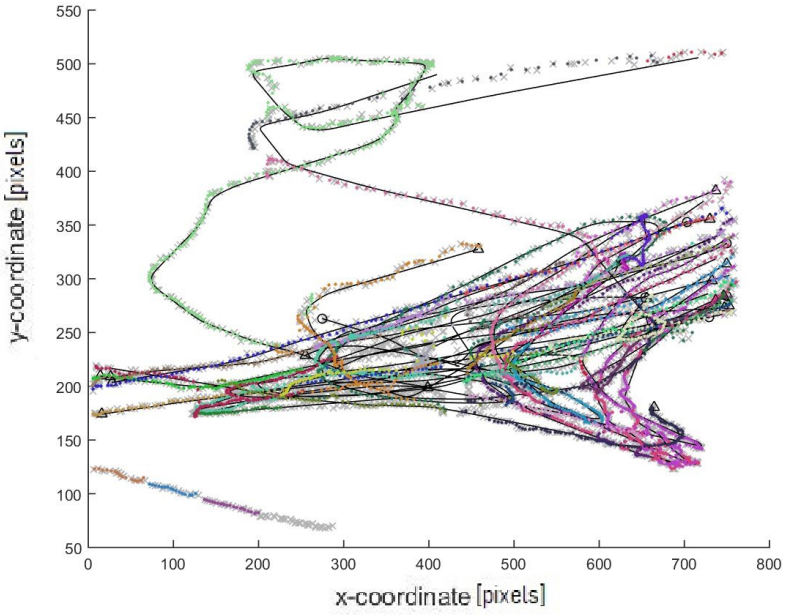
First, a visual analysis is made for the original output generated by the library, which prints the trajectories of the tracked objects separated by its X and Y axis, each vs time. We print a new view, between both X and Y axis and superimposing the time lapse in it. This is done because some cross information is lost when the axis are separated, the main example can be seen in fig 3.1a, where the detection made by *YOLO* are not as precise as thought compared with the ground truth, detail that can't be seen in the fig 3.1b, this can be attributed to the deformation of the data with the different margins of the plot which causes a 'squeeze' effect on the way the data is presented. This doesn't only makes the false idea that the detection are closer to the ground truth, but also the trajectories in have less space to be plotted so the data is superimposed with itself, making harder to see any possible error. The new view is free of this problems other but is considered an improvement and for cases with a big amount of trajectories, plots of sub sections of the time lapse will be generated so they don't end up having the same problems as before. This new view also makes it easier to detect and classify the type of problem with the estimation, between an ID swap, new ID assignation or no tracking at all, the fig 3.2 are an example of how they facilitate this analysis. As same as before, is it much easier to detect visually this kind of errors with the new visualization leading us to a better study and comprehension of the results.

On the other hand, fig 3.2 is also a good example to show that the original results are not completely disposable, as they show in a better way the consistency of the estimation through time, as it is more 'tidy' and well presented because of the less amount of superimposition of information occurring in the plots. So both views will be used for the analysis of the results as they complement in the information they present and we will use the one we consider best for each case or both.

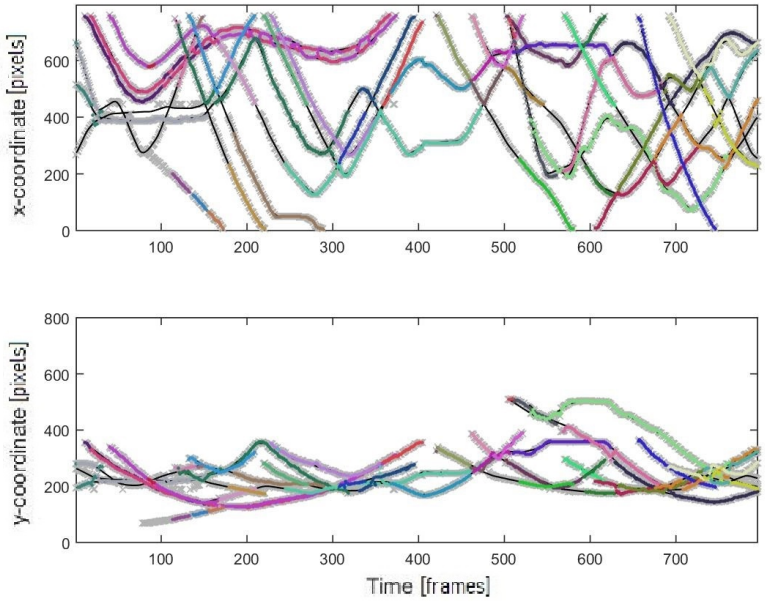
For the Extended Kalman Filter, two interesting thins can be appreciated with the new visualization that wasn't able with the original one. First is that the predictions made by

the *EKF* is more of an oscillation around the true track, what is expected because of how Kalman filters work, trying to balance the weight of the estimation and the measurement, giving more to one or the other depending on the one is more stable, measured with the Kalman gain. This can be seen on the left tracks on the fig 3.3a, thing that is not so much noticeable in the second fig 3.3b.

Second is that the range calculated for the clutter, which is defined by the size of the image does not consider the whole space of possibility for the trajectories. We present two cases, the first one (fig 3.4a) is for a standard value of clutter, 30 that is a level the filter is expected to work with, and and extreme case with a value of 250, shown in fig 3.4b, where the range of the clutter is much more demarcated. This lets us know that is probable that some trajectories will fall short or have more error at the end of it, where there will be less measurements to make a good estimation.

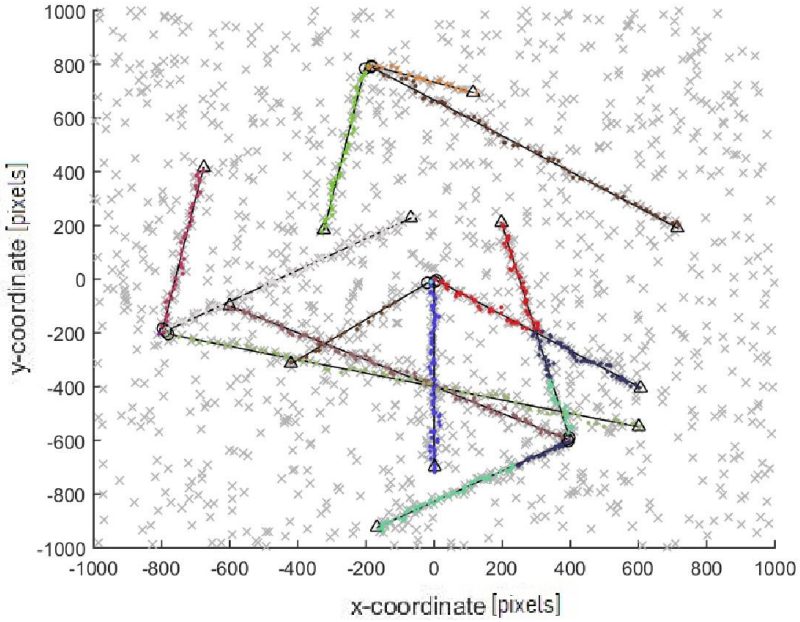


(a) X vs. Y.

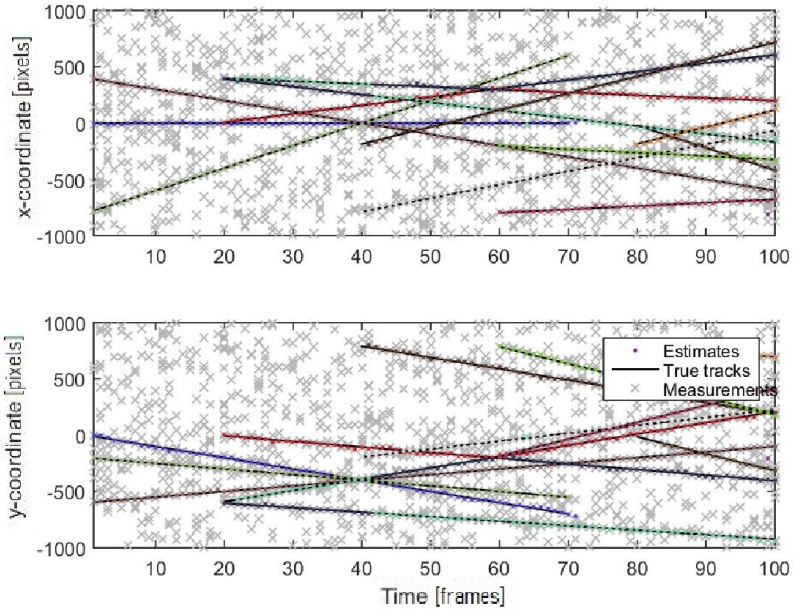


(b) X Y vs. Time.

Figure 3.1: Comparison of views in PETS2009 Dataset.

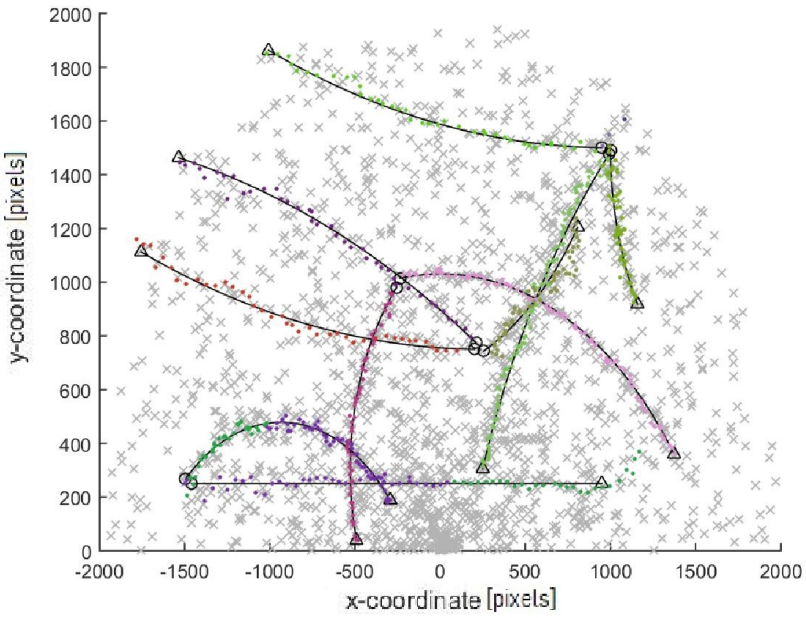


(a) X vs. Y.

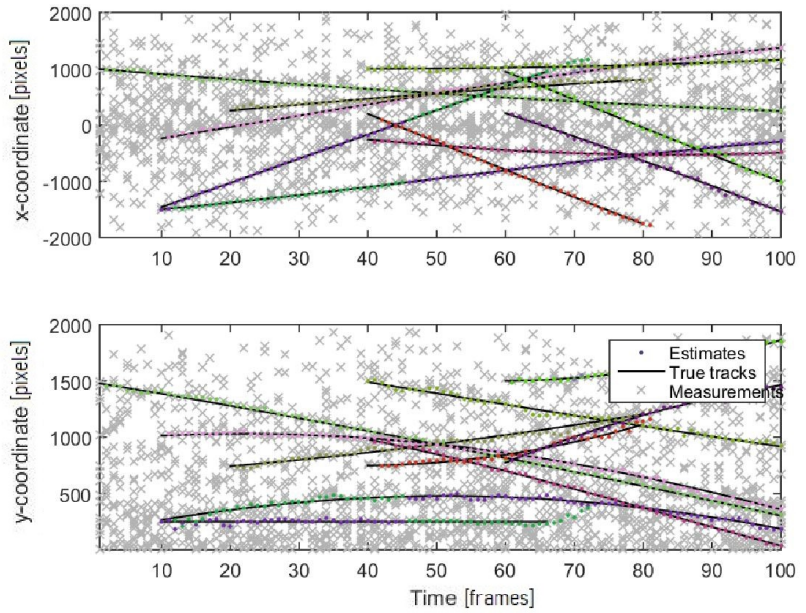


(b) X Y vs. Time.

Figure 3.2: Comparison of views in the simulation.

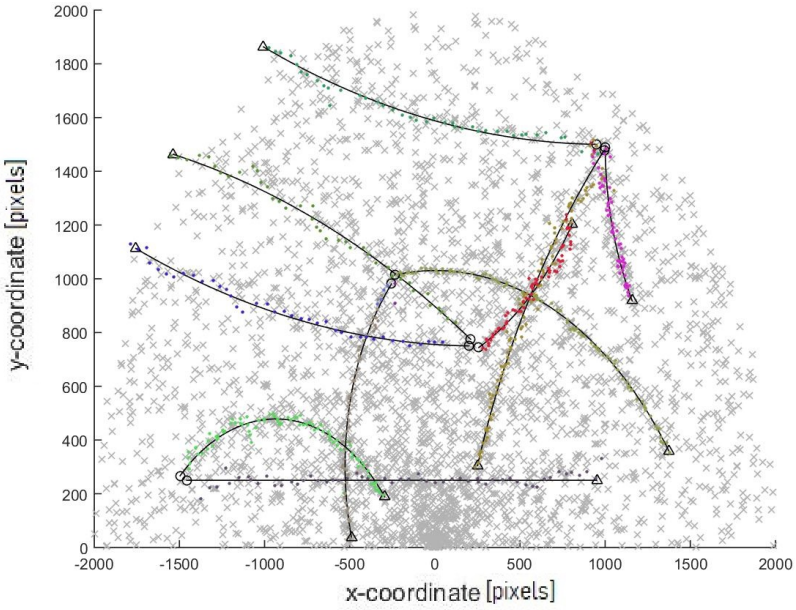


(a) X vs. Y.

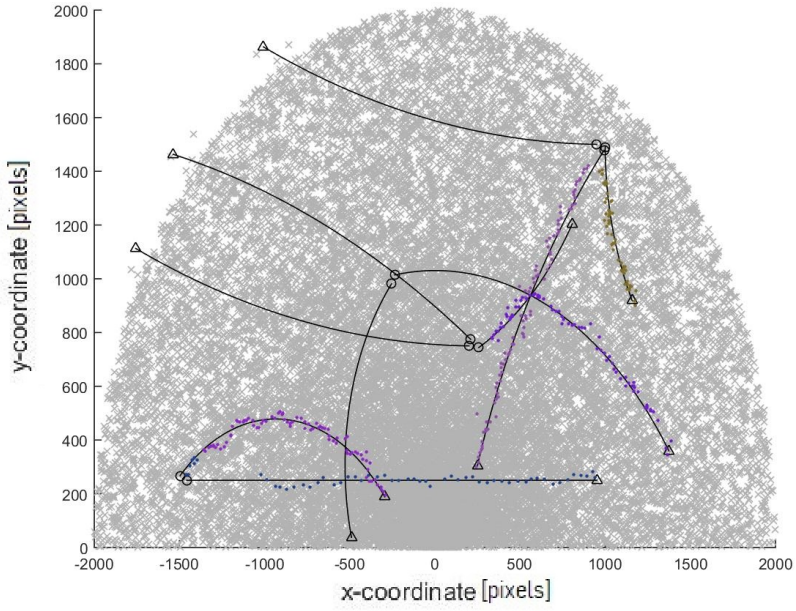


(b) X Y vs. Time.

Figure 3.3: Comparison of views with EKF.



(a) Example with clutter level of 30.



(b) Example with clutter level of 250.

Figure 3.4: Comparison of views in the simulation.

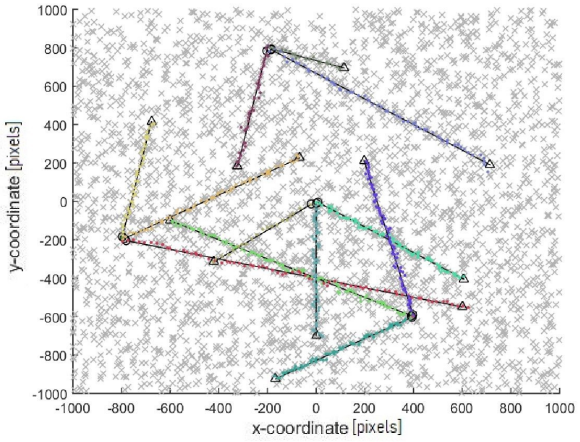
3.2 Variability of results.

Because we are working with stochastic algorithms, a same experiment may have different outcomes if its reproduced several times, so what is important is that these results are consistent within the frame we are working with. As shown in fig 3.7 we even got some rather different results for the exact same values, where in the first (fig 3.5a) one there is no big error, but for the second one (fig 3.5b) there is an ID swap for the violet and gray labels, both over tracks that start on position {400,-600}. Finally the last one (fig 3.5c) has an untracked object of the simulation and the ID swap mentioned before but this time occurring in a different moment in time. For this reason, for all the next test realized and presented at multiple samples were generated for each case and what is presented is a generalization of each one.

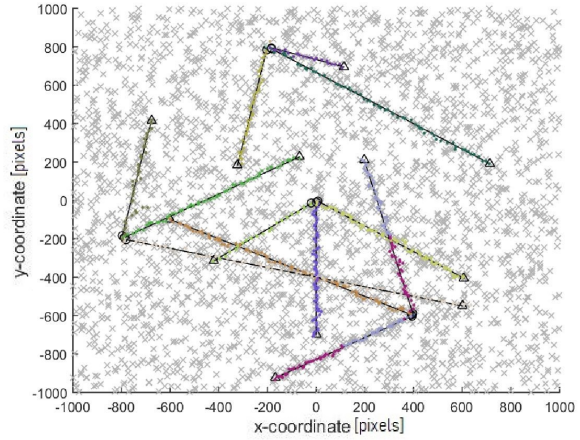
Then the same experiment is repeated for the *EKF*, where many different results are obtained, although the same kind of problems appear as before, having for a same set of variables, different amount of the types of errors we just discuss. So we present the results obtained with the *OSPA* distance, the figure 3.6a is an standard result for this conditions, with a high peak at the start and the rest of the values mostly under the 50 mark, its important to remember that this type of distribution depends completely on the model we define and the values c, and p of the *OSPA* distance, so what we are interested in is the differences between the results and not the results per se, results for other data will return completely different results.

For the figure 3.6b, an out layer outcome is presented where a second peek occurs right after the first one, this one within the first 10 time steps, where as can be seen in figure 3.7b there is only one object been tracked, which has a not so stable estimation of position, but because is the only object this error of estimation has a much bigger impact in the *OSPA* value than the same error when there are more objects in frame. This can cause major differences in the interpretation of the data if its not considered with the care it deserves.

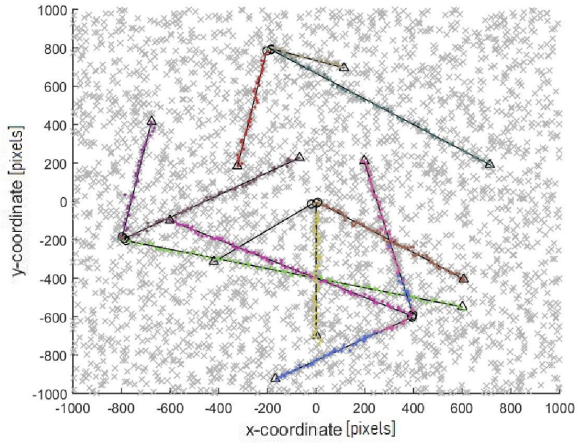
For the third fig 3.6c the same case can be done the other way around, where there is no peak is at the beginning, reducing the mean value of the *OSPA* Distance by a huge margin. We present this results to show that its important not only to understand how the model work and the data we are working with, but that because we are working with an stochastic algorithm, many iterations must be done and we can't just take the first result obtained, but that a general conclusion must be made for each case, for now on all results presented will be the standard result for each set of variables selected, trying to be the most faithful to the mean of the data obtained.



(a) Example without errors.

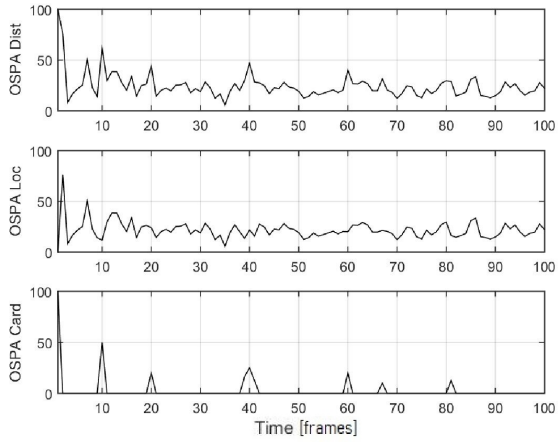


(b) Example with ID Swap.

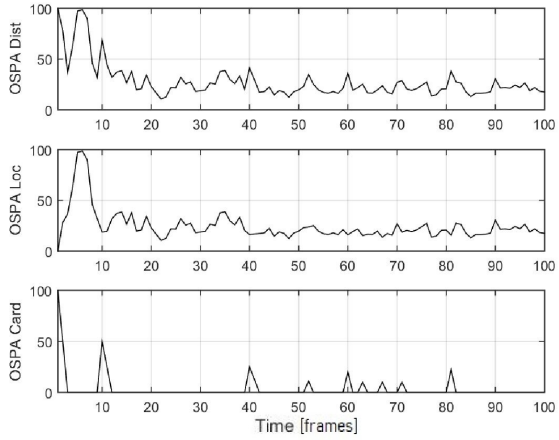


(c) Example with an untracked object.

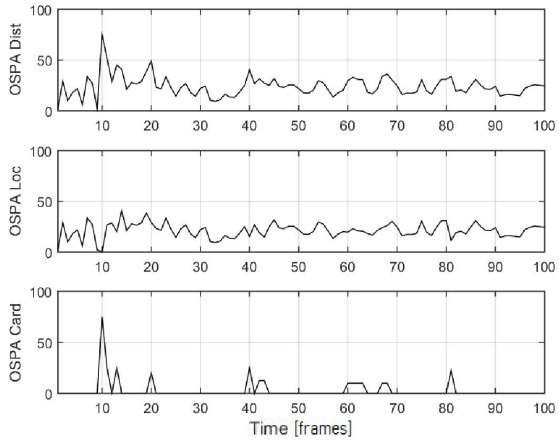
Figure 3.5: Comparison of different results for the same experiment.



(a) Example of standard result.

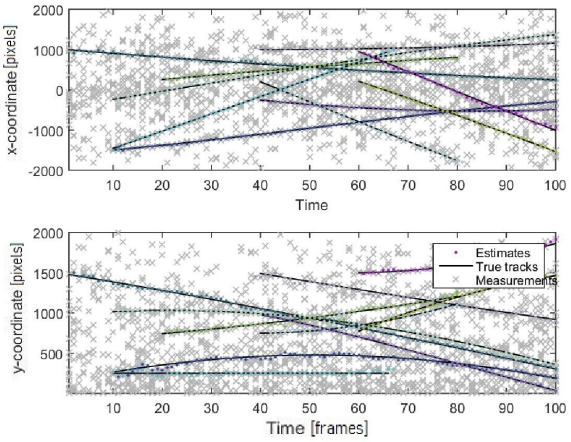


(b) Example with double peak.

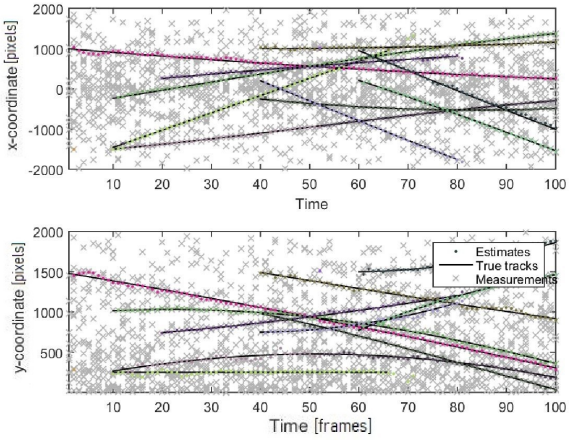


(c) Example with no peaks.

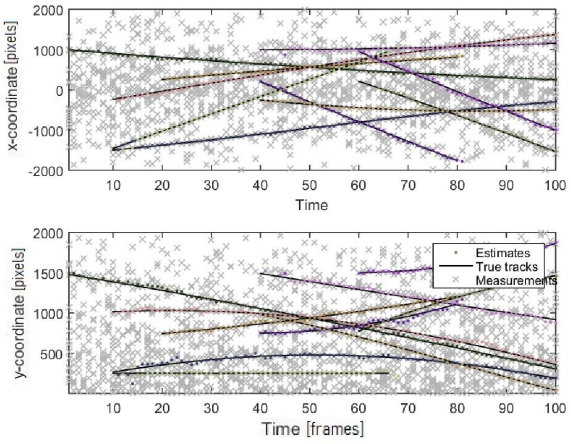
Figure 3.6: Comparison of different OSPA values for the same experiment.



(a) Example of standard result.



(b) Example with double peak.



(c) Example with no peaks.

Figure 3.7: Comparison of different OSPA values for the same experiment.

3.3 Sensibility Tests

3.3.1 Sensibility Test - *Clutter*

Next we present the study of the variation of the value of clutter in the model, making a sensibility study of the value and so to see how it affects the results. It is important to mention that this parameter is not used in the real data sets, because it represents the amount of noisy measurements that can be obtained in the simulation, but we are interested in showing the resilience of the algorithms before we use them. In the simulation, test where made for values $1, 5, 10, 100, 200, 300, 400, 500$, but only the results for $10, 100, 200$ are shown because the most differences can be seen between them. The value of 10 is the original value of the example so it return some of the best results, and reducing it wont cause any new error to appear, because we are just reducing the possibility of error by removing false positives. This experiment is done for the *PHD* and *GLMB* filters, as mentioned before the first one only returns the selected measurements, but it doesn't assign an specific ID to it, while the second one does, there for in fig 3.10 all the selected values by the filter are highlighted with the same color (black), but the results in fig 3.12 are highlighted with different colors which represent each other label or ID.

It can be noticed that with the increasing amount of false positives the filters are less capable of correctly detect the measurements corresponding to the real trajectories of the simulated objects. For the first two cases of the *PHD* filter (figs 3.10a and 3.10b) is easier to notice that it is able to assign the correct values, although in the second one is clearly less precise because the measurements are farther away from the ground truth. For the last one (fig 3.10c) the filter is still able to find the trajectories, but the amount of error is way higher, having many values that are extremely far from the ground truth, making it even hard to know to which tracks are they even assigned.

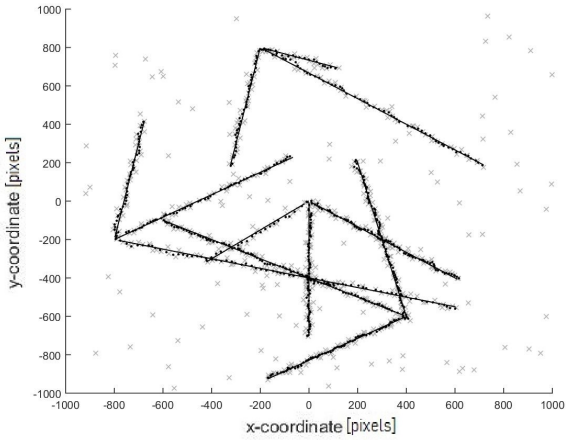
We choose to show the values of $10, 100, 200$ also because are mid points that are good representations of the errors found in each level of clutter for the *GLMB* filter. For the value of 10, the results of the *GLMB* are good because is a small value, considering the original value is 30 in this case, as same of before reducing the values wrong add any type of error because we are just reducing the amount of false positives. In general not many error appears, mainly a little amount of ID swaps in some of the 10 results generated for each case. Then for 100 the miss tracking of at least one object is common with in the sample, with more swapping of IDs happening. Finally for the value of 200 not only there are more miss tracks and ID switches, but wrongly estimated trajectories appears for some cases and the correctly estimated ones are less precise, giving higher values of *OSPA* .

For the value of 100 we can see differences between both filters, mainly that in the *PHD* filter all trajectories has some amount of detection, while the *GLMB* there's already one that is not been detected completely. For the value of 200 the amount of false negative is larger in this filter, compared with the *PHD* filter as seen in fig 3.17c and 3.10c respectively, where

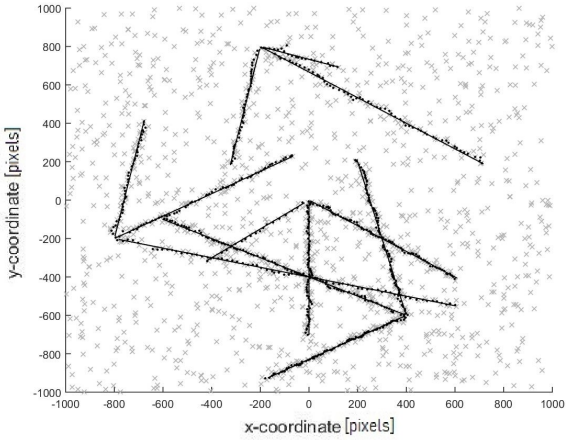
almost any object is detected and tracked, but not in the *PHD* filter which is able to detect at least to some amount almost every object.

Using *EKF* and $\delta - GLMB$, the results where a bit different in the way that the errors appear way sooner than with the *GMS* filter, in both figs 3.10a and 3.10b there is already existing errors that weren't that frequent in the last part. ID switches and mismatches that previously occur for bigger values of clutter here are present for all values making of this filter a less robust one, at least for this simulation. For the value of 100 (fig 3.10c), multiple targets weren't tracked and ID switches occur more frequently too.

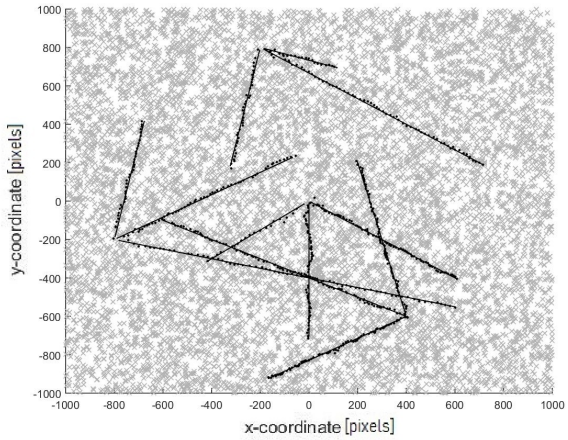
But as seen in fig 3.10, the results using *PHD* are similar to the previous part, the filter is capable of maintaining a coherent estimation of the true tracks, but with the increasing value of clutter, this estimations become more erratic and less accurate, but they are able to track all objects in frame, while the $\delta - GLMB$ couldn't. Comparing the results of the *OSPA* distance in fig 3.11 another point appears, mainly with the fig 3.11c, where the first values get off the chart for the cardinality graph, this mainly because only one object is present in frame at the time but the estimations generate more than one tracker, creating a huge difference and so obtaining this level of error compared with the other results. In particular the mean *OSPA* value for each one is 23, 33 and 47 respectively, highlighting how much more error this estimations have. But also is important to notice that it shows that the the first case is more precise than could be noticeable visually, when both the first and the second figs are very much alike.



(a) clutter value = 1

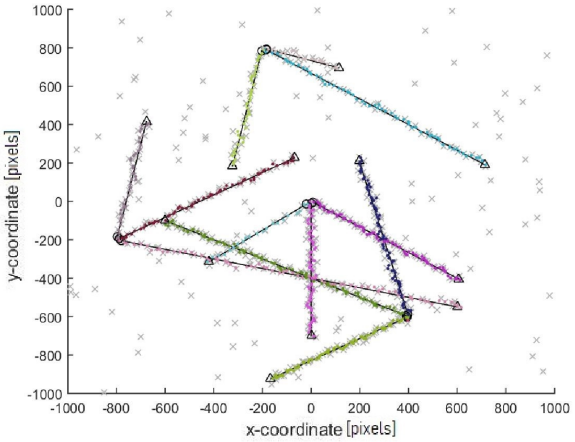


(b) clutter value = 10

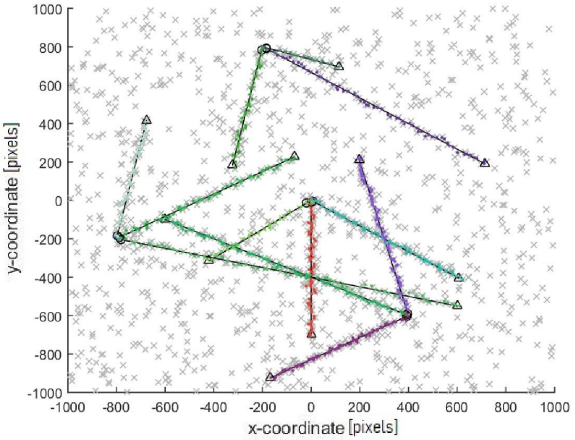


(c) clutter value = 100

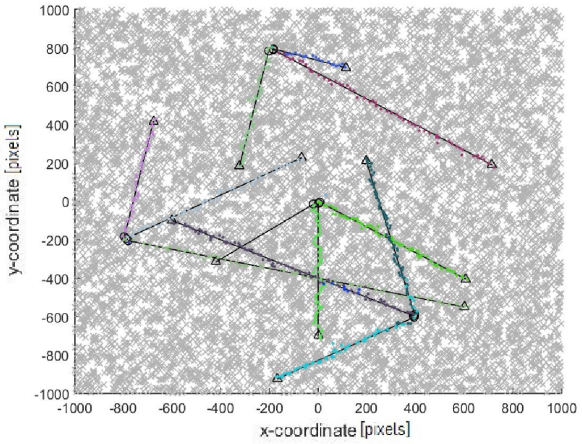
Figure 3.8: Variation of clutter value for the simulation using the PHD Filter.



(a) clutter value = 1

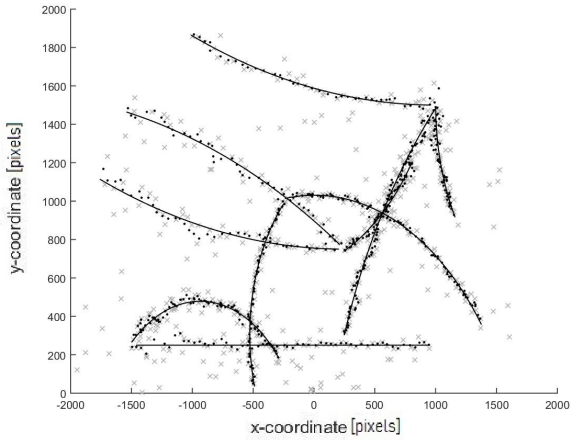


(b) clutter value = 10

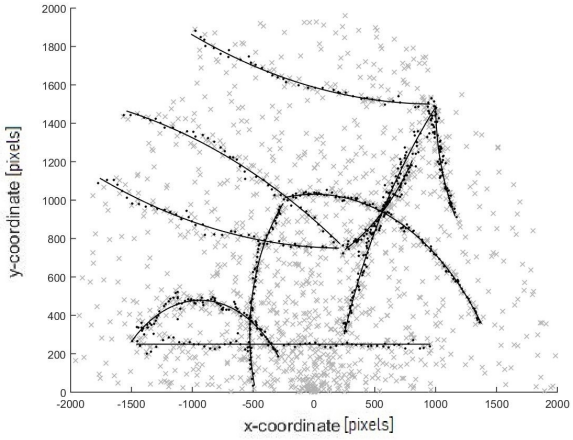


(c) clutter value = 100

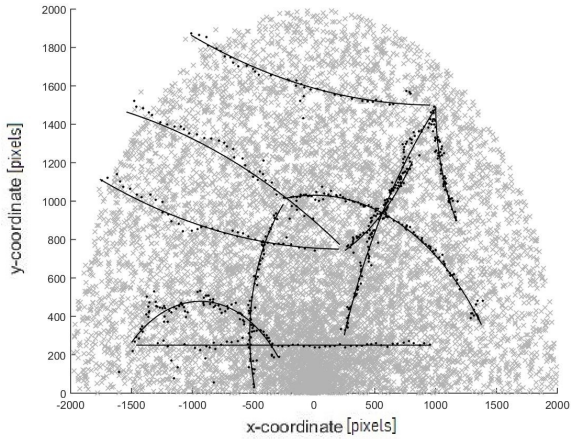
Figure 3.9: Variation of clutter value for the simulation using the GLMB Filter.



(a) clutter value = 1

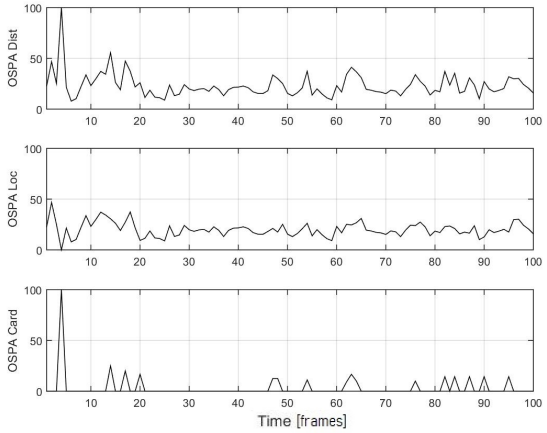


(b) clutter value = 10

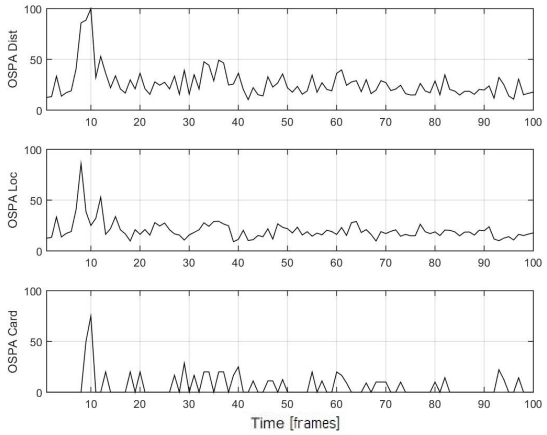


(c) clutter value = 100

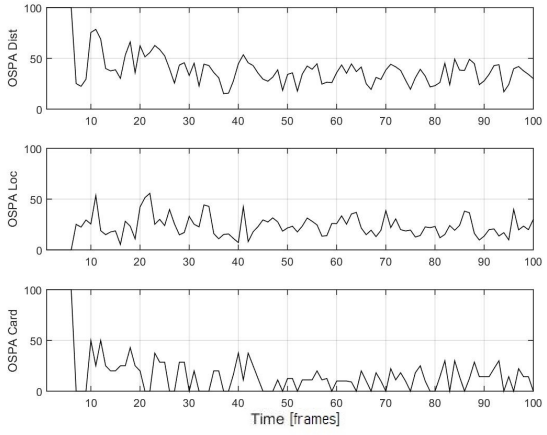
Figure 3.10: Variation of clutter value for the simulation using the EKF and PHD.



(a) clutter value = 1

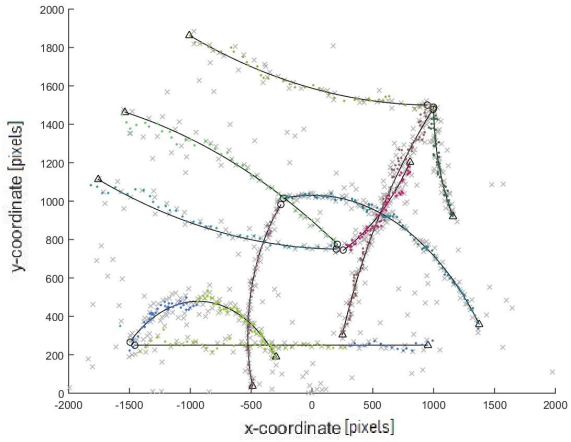


(b) clutter value = 10

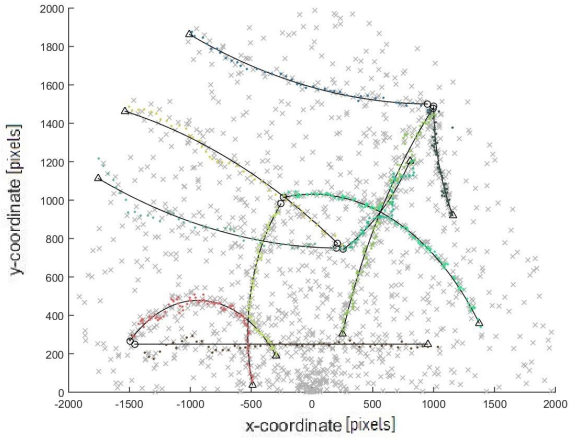


(c) clutter value = 100

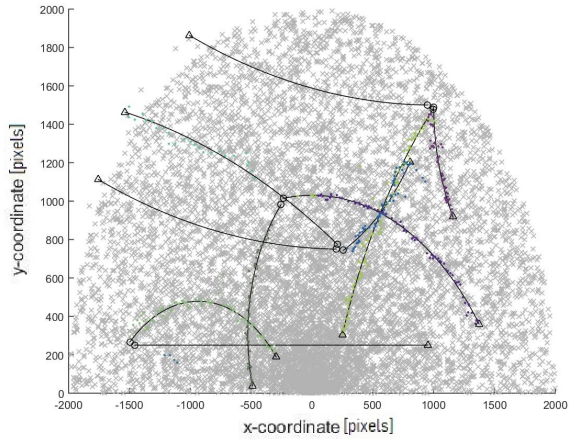
Figure 3.11: Variation of clutter value for the *OSPA* results using the EKF and PHD.



(a) clutter value = 1



(b) clutter value = 10



(c) clutter value = 100

Figure 3.12: Variation of clutter value for the simulation using EKF and the GLMB Filter.

3.3.2 Sensibility test - *sigma_ v*

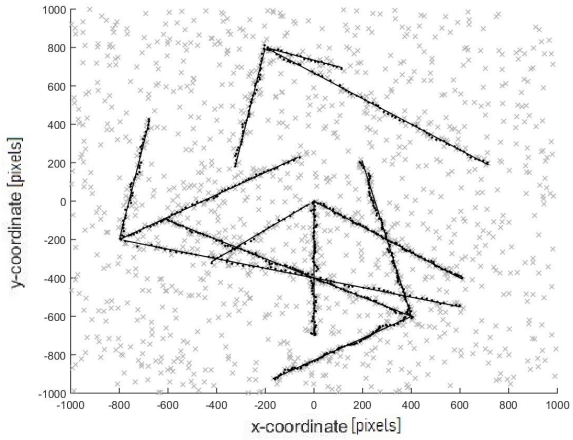
As it can be seen in the fig 3.16, with the increase of this variable, the variation of the results for the *PHD* filter are very similar to the previews test, where the sparsity of the selected estimations is bigger with each new value. This means that the distance between the assigned measurement and the true track is bigger for the last example compared with the ones with lower values of σ_v, although for this variable new values were chosen to be presented=$\{1, 30, 150\}$. As same as before, this are selected specifically to show the scenarios where the difference are more notorious, mainly for the $\delta - GLMB$ filter, where different type pf errors can be seen.

The main difference between both *PHD* and $\delta - GLMB$ filters is that for the first one, with the increase of the value of the variable, the results start to be more disperse, while the results of the second one is a bigger amount of errors in the ID assignation, more than a more erratic selection of measurements. This can be observed in both the cases of the *GMS* and *EKF* filters, where in the first one it is a bit more robust to the variance of the tracking values. In fig **??** is shown the most common mistake in for this value, that is an ID switch caused by an intersection of two objects, the second one most common error is the generation of a new ID for already tracked objects, this can be seen in the fig 3.14c where the red trajectory is mismatched after the intersection with the green one, so the true track is assigned a new ID of color blue instead.

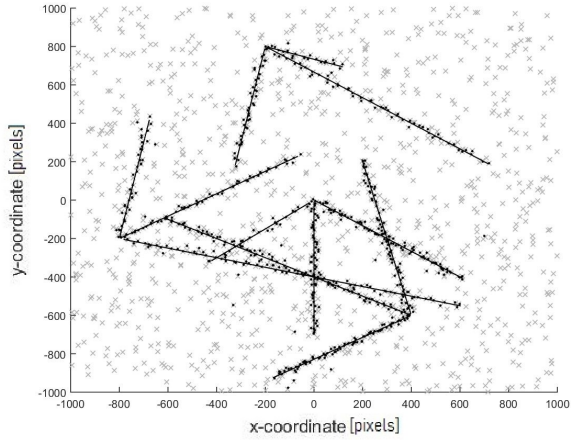
Finally the value of 150 was chosen because it is around the break point for the $\delta - GLMB$ with *GMS* filter, as can be seen in fig 3.14c, where a true track is not tracked and most of the other ones have very erratic results or ID switches because of intersections or poor tracking. This results are expected for the filter, because we are making the acceleration step way to big to generate any consistent tracking, so erratic behaviour is expected.

Although, different to the previous experiment, for the lower values of sigma_v it shows a more stable $\delta - GLMB$ filter over the *PHD* for the *GMS* like the value of $\sigma_v = 1$. This is more noticeable in the fig 3.18, where *PHD* has a higher value on average, with a value of: 11 and 15 corresponding to $\delta - GLMB$ and *PHD*. As we mentioned before, this is not the optimal way to compare this results, because a lot of different edges of the problem are collapsed into a single value, loosing information in the process, but it gives a general idea of which example had the most stable results.

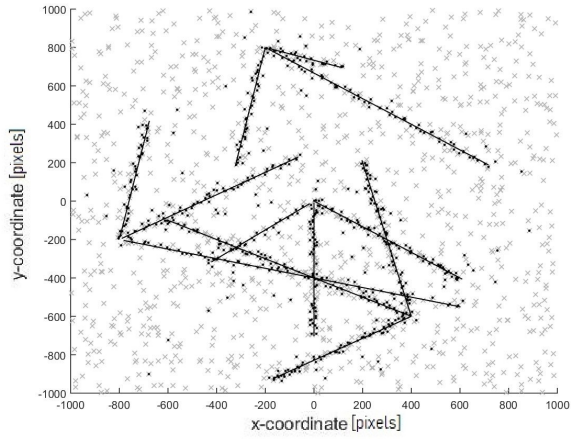
For most of the results obtained for the *EKF*, all said before keeps been true, but we have to remember that because of how the the approximation works, the resulting tracked trajectory will be less stable than the *GMS*, so for example the *OSPA* mean for the *EKF* is much higher than the values presented before, with values of 23 and 27 for $\delta - GLMB$ and *PHD*. Also the errors of the first one occur much earlier than for *GMS* solution and so by the value presented of 150, more true tracks are not tracked and errors generated by intersections or faulty tracking.



(a) *sigma_v* value = 1

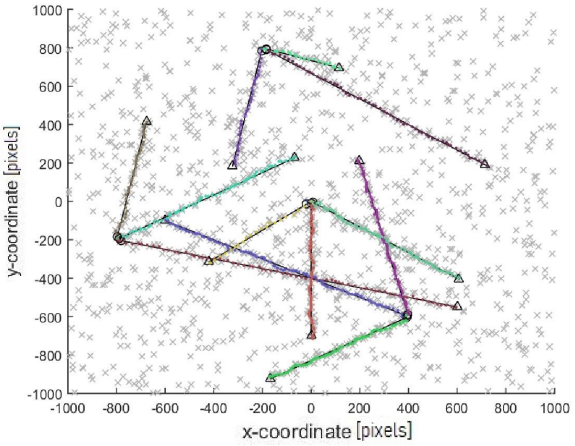


(b) *sigma_v* value = 30

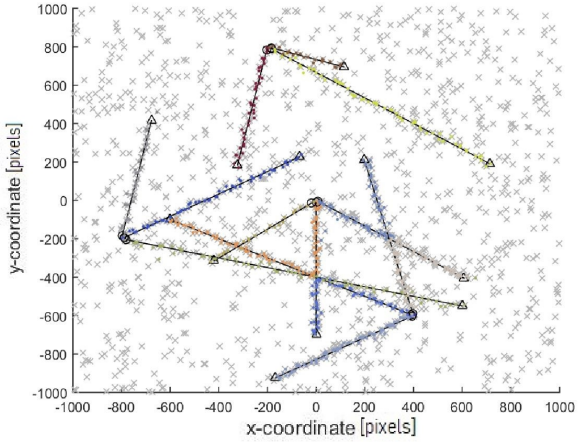


(c) *sigma_v* value = 150

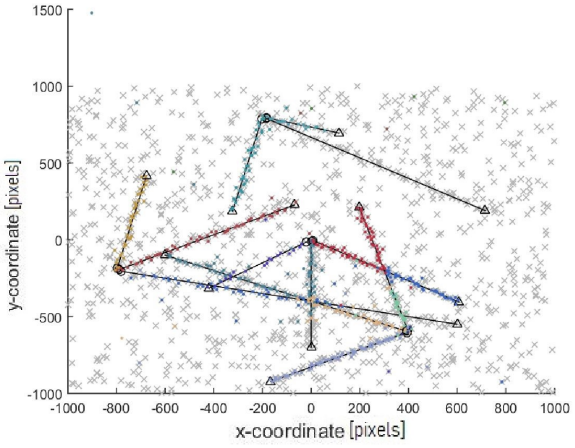
Figure 3.13: Variation of sigma_v value for the simulation using the PHD Filter.



(a) *sigma_ v* value = 1

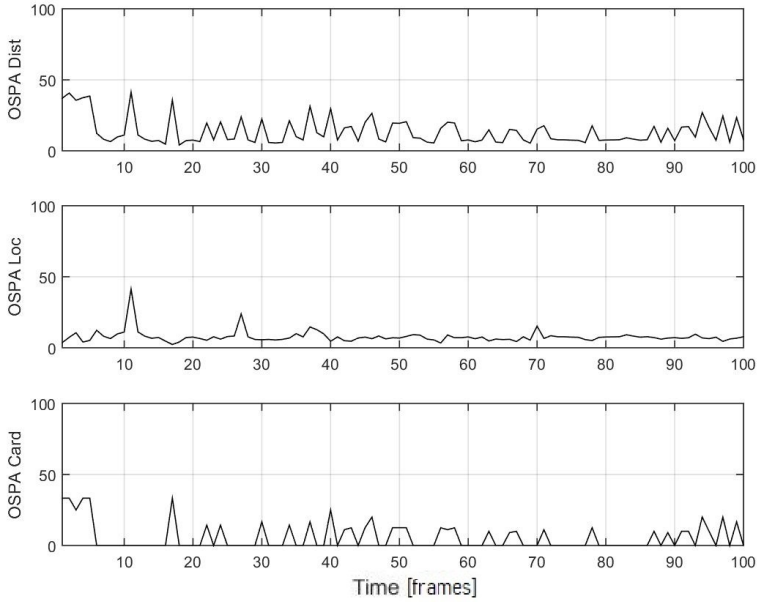


(b) *sigma_ v* value = 30

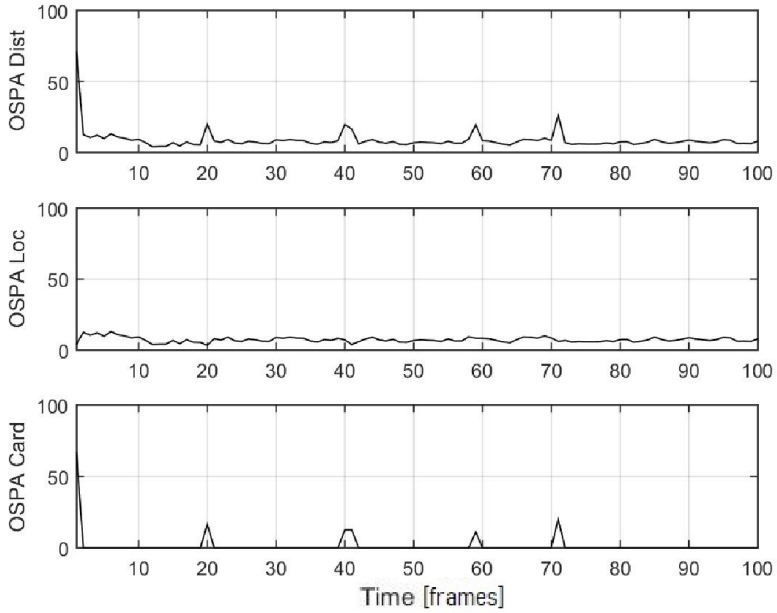


(c) *sigma_ v* value = 150

Figure 3.14: Variation of sigma_v value for the simulation using the GLMB Filter.

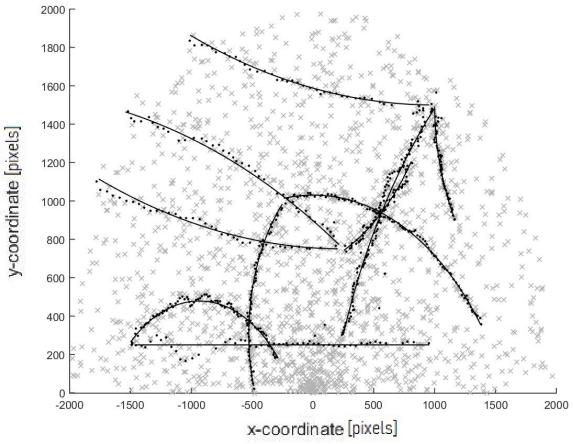


(a) *sigma_v* PHD's OSPA result

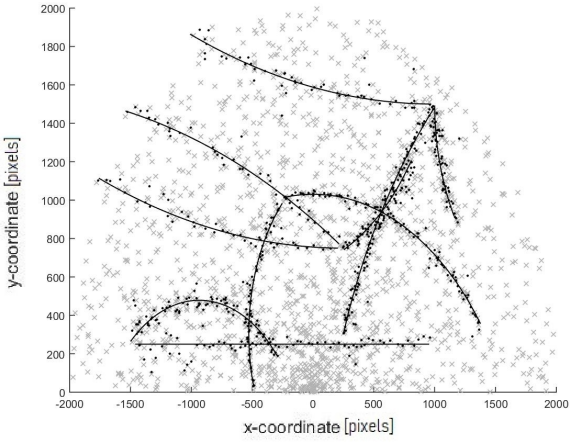


(b) *sigma_v* jointGLMB's OSPA result

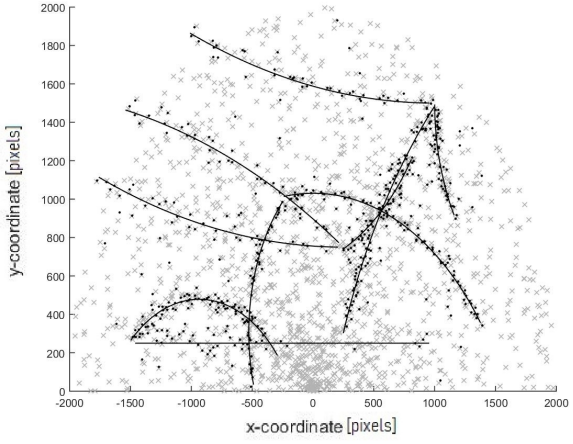
Figure 3.15: Comparition of OSPA results.



(a) *sigma_ v* value = 1

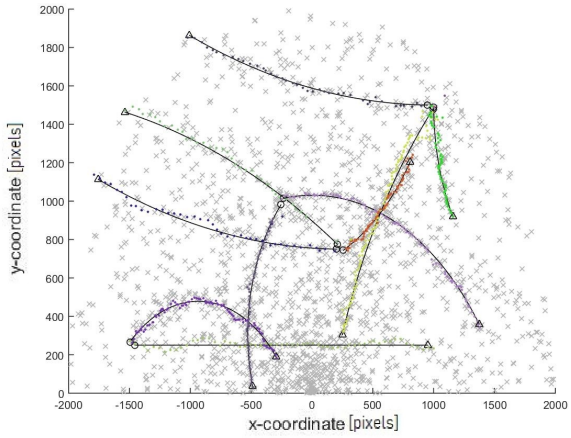


(b) *sigma_ v* value = 30

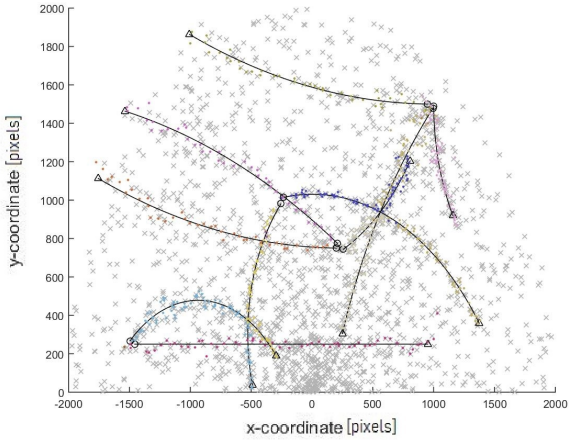


(c) *sigma_ v* value = 100

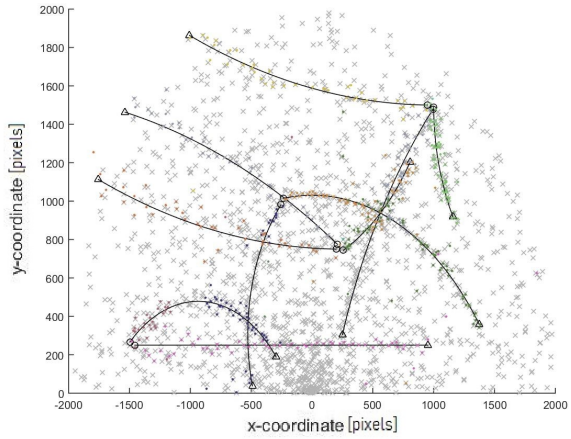
Figure 3.16: Variation of sigma_v value for the simulation using the PHD Filter.



(a) *sigma_v* value = 1

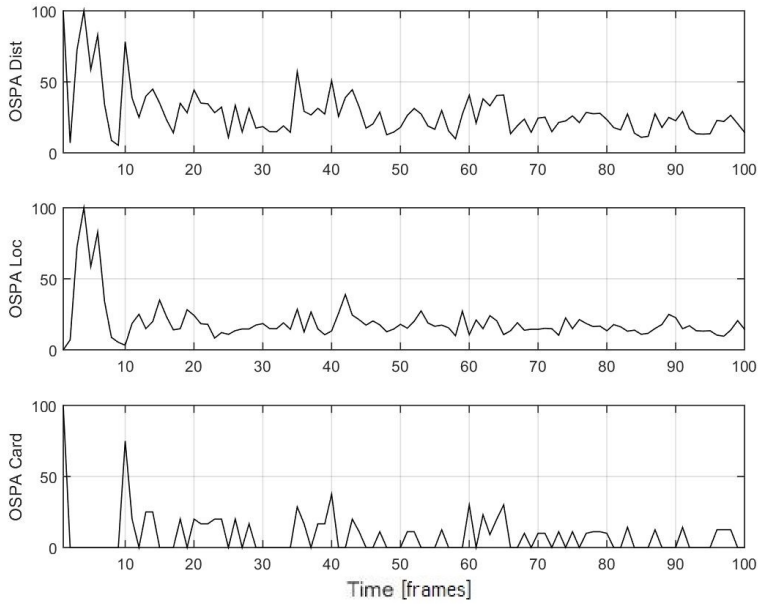


(b) *sigma_v* value = 30

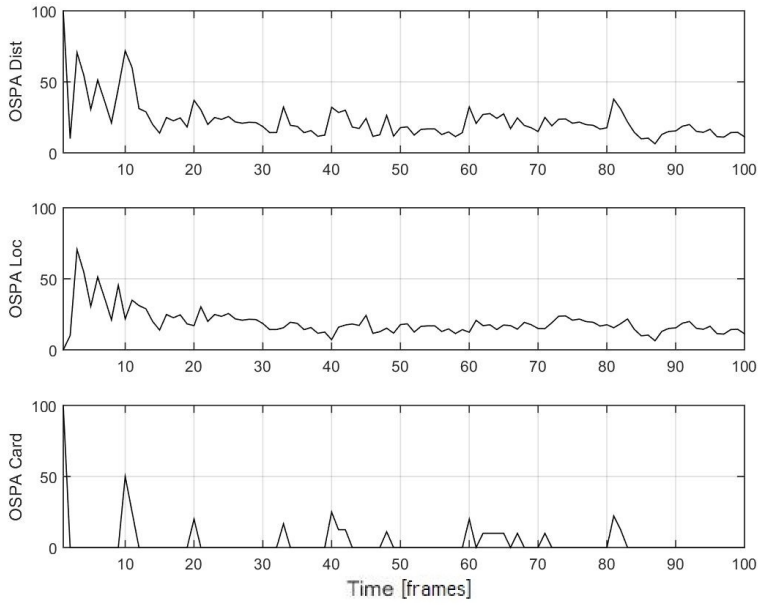


(c) *sigma_v* value = 150

Figure 3.17: Variation of sigma_v value for the simulation using the GLMB Filter.



(a) *sigma_ v* PHD's OSPA result



(b) *sigma_ v* jointGLMB's OSPA result

Figure 3.18: Comparition of OSPA results.

3.3.3 Sensibility test - PETS2009

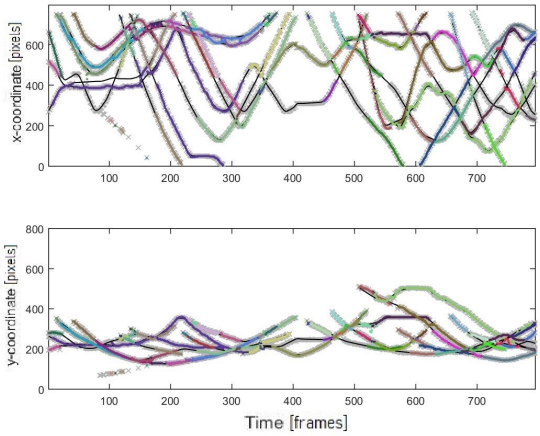
From the previous part we can make a more educated guess, and choose to work in the range of [1-45] for the adjustment of value sigma_v for the new data sets, in particularly from the results of this sensibility test it can be seen in fig 3.19 that there is a trade-off between the amount of measurements assigned to each label, and the amount of labels generated. For the labels in 3.19a, there is not much ID swapping or new ones been generated, but there are some big gaps in time where some object are not been tracked and then are assigned a new label, this can be seen in the fig 3.19c in the x-coordinate figure. But as the value increase the amount of new id generated start to increase too, this is most noticeable in the y-coordinates figure, where the amount of color change is easier to detect. Once the value start to increase too much, the weight of increasingly new labels for each trajectory is too big compared with the amount of correctly assigned measurements that we get, making it not acceptable, and so we decide to stay with a sigma value of 15.

For the *EKF* the results are more variable than the ones of the *GMS*, while the first one tries to interpolate between the estimation of the true track and the measurement, the second one "plays it safer" and the results obtained by it are almost always indistinguishable from the measurement done by *YOLOv3*. This has to do with the way each solution works, and in part the level of certainty given to the probability of detection for a measurement that was almost 1. Now this clearly depends completely in the quality of the detector and how much accuracy and precision it has, also in our case, we decided to choose a high threshold for *YOLO* to detect pedestrians, this way we can assure a high level of confidence on it by sacrificing some detection in the process, but taking care of maintaining enough ones for a trajectory to me detected.

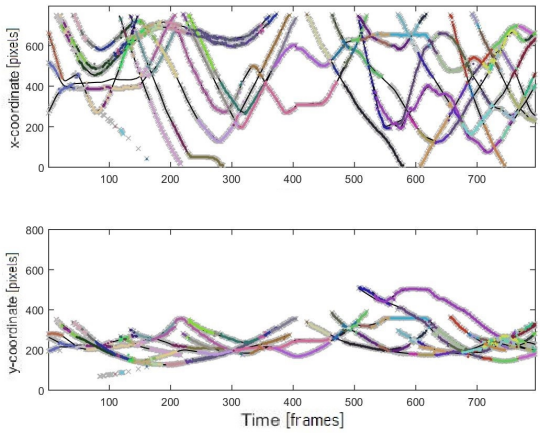
An interesting result is the mean *OSPA* value obtained, mainly because from figs 3.21 and 3.23 not much information can be obtained. But we expected it to have a more noticeable difference for the sigma_v value of 45, but for the *GMS* the results where in average similar for the three cases, as seen in the table 3.1, and for the *EKF* the first case has the highest mean in average. Also it can be seen that the mean obtained for all cases of *EKF* are "worse" than the ones of *GMS*, but as it will mentioned later, this is just one of many ways to evaluate the performance of the trackers and it is important to understand what it really measures before using it as a tool for naming one tracking better than the other.

Tracker	$\sigma_v = 1$	$\sigma_v = 15$	$\sigma_v = 45$
GMS	20	19	20
EKF	26	22	22

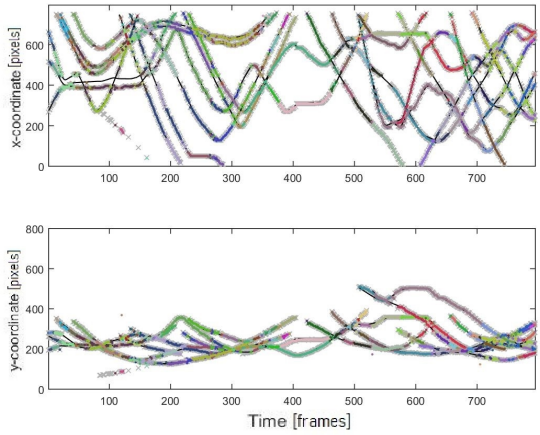
Table 3.1: Result Comparison of *OSPA* mean.



(a) *sigma_v* value = 1

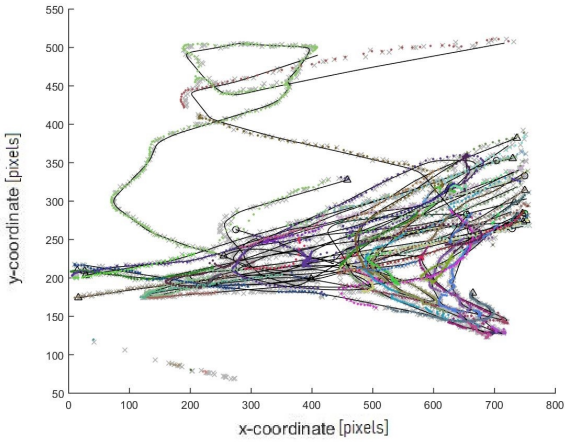


(b) *sigma_v* value = 15

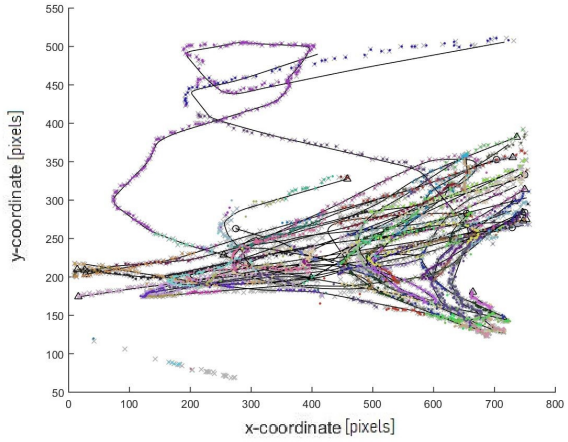


(c) *sigma_v* value = 45

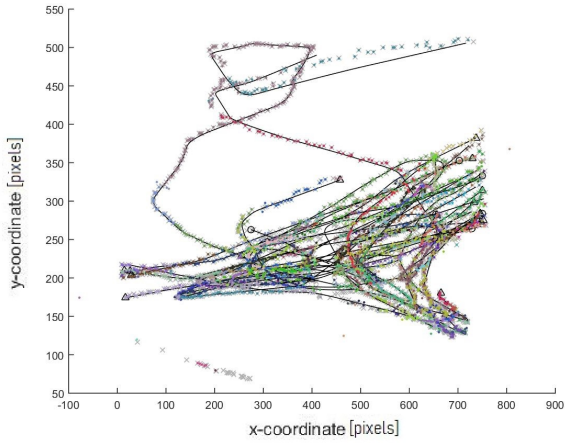
Figure 3.19: Variation of sigma_v value for PETS2009 dataset using GMS, coordinate vs time.



(a) *sigma_v* value = 1

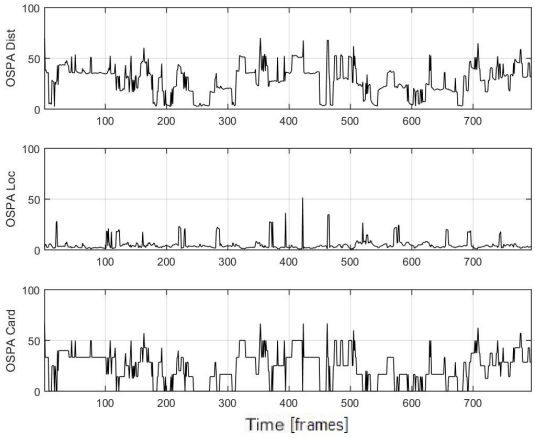


(b) *sigma_v* value = 15

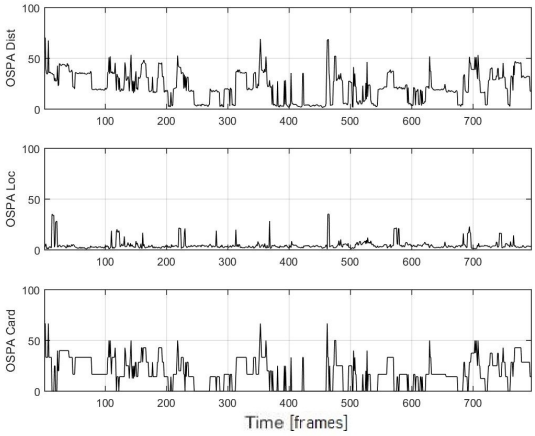


(c) *sigma_v* value = 45

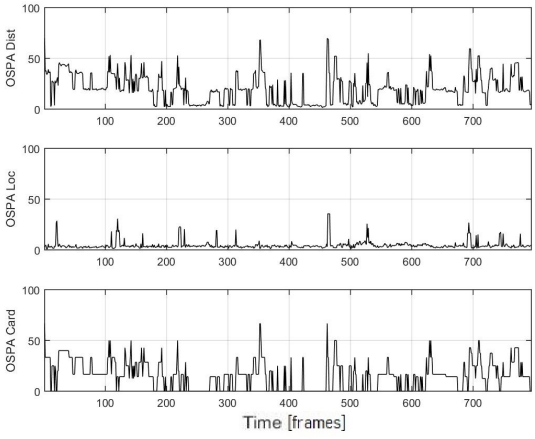
Figure 3.20: Variation of sigma_v value for PETS2009 dataset using GMS, X vs Y representation.



(a) *sigma_ v* value = 1

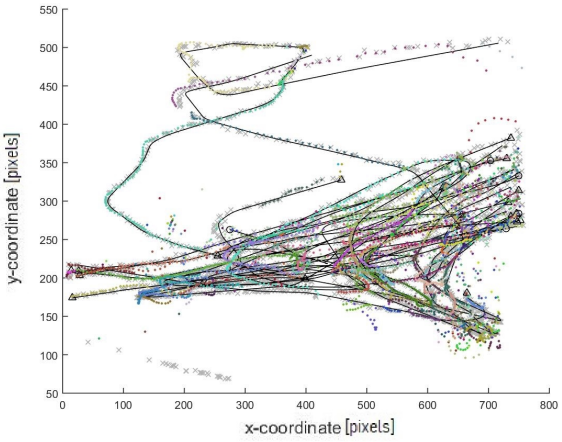


(b) *sigma_ v* value = 15

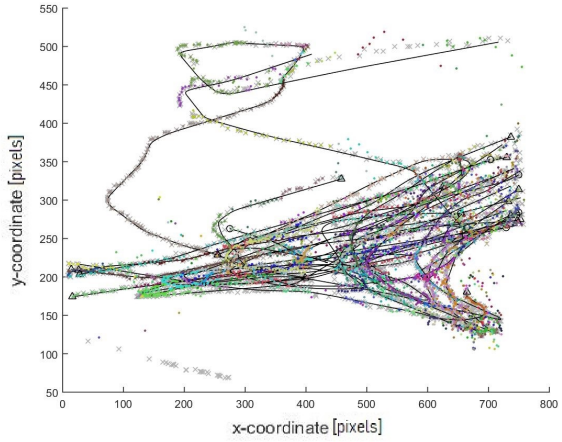


(c) *sigma_ v* value = 45

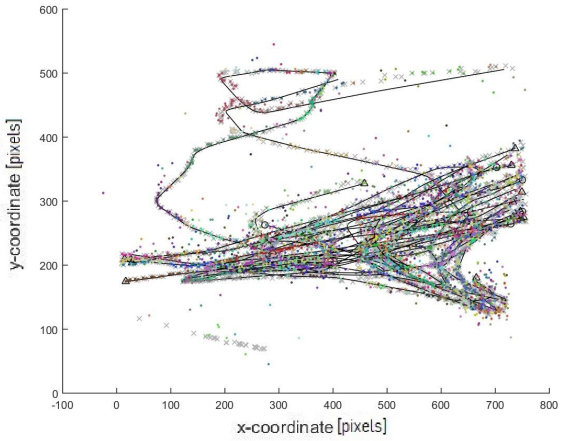
Figure 3.21: Variation of sigma_v value for PETS2009 dataset using GMS, OSPA output.



(a) *sigma_v* value = 1

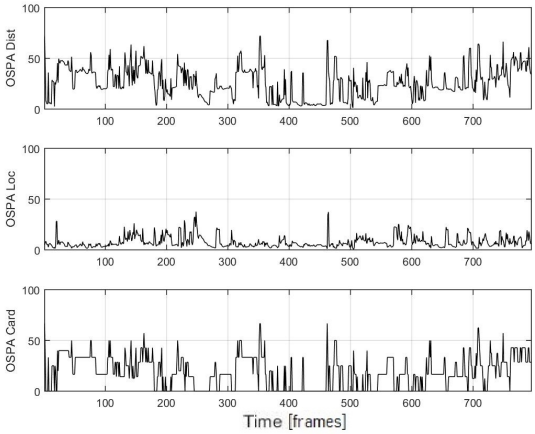


(b) *sigma_v* value = 15

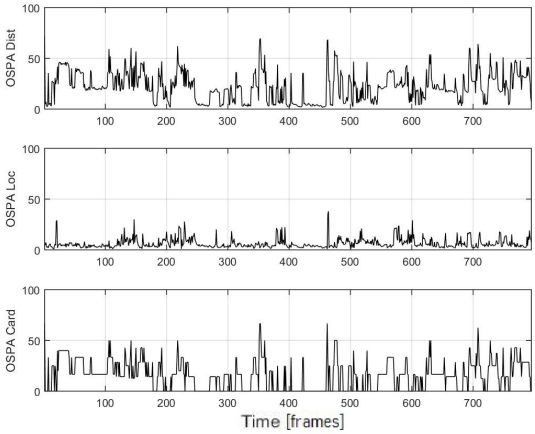


(c) *sigma_v* value = 45

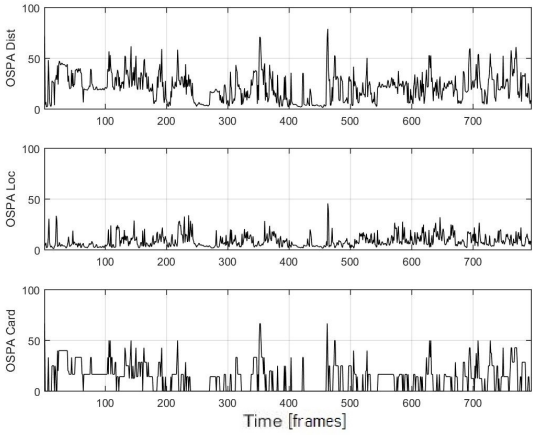
Figure 3.22: Variation of sigma_v value for PETS2009 datasey using EKF, coordinate vs time.



(a) *sigma_ v* value = 1

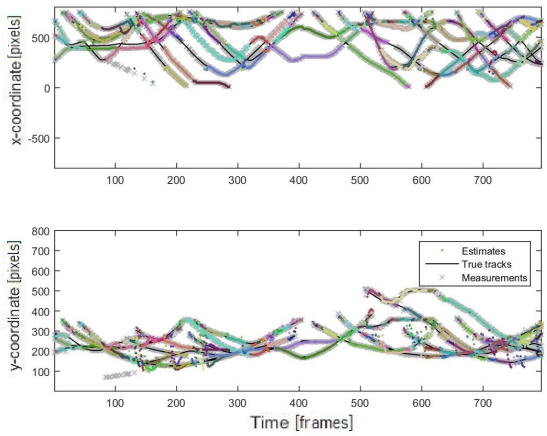


(b) *sigma_ v* value = 15

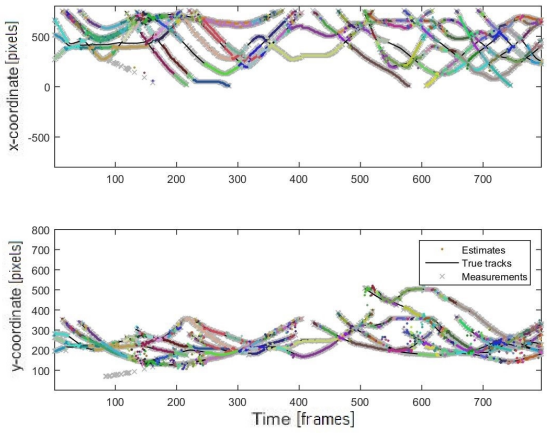


(c) *sigma_ v* value = 45

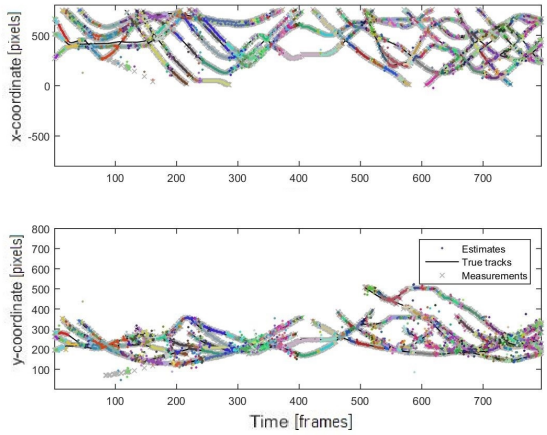
Figure 3.23: Variation of sigma_v value for PETS2009 dataset using EKF, X vs Y representation.



(a) *sigma_v* value = 1



(b) *sigma_v* value = 15



(c) *sigma_v* value = 45

Figure 3.24: Variation of sigma_v value for PETS2009 dataset using EKF, OSPA output.

3.4 General Results

3.4.1 JointGLMB - PETS2009

As told before, the values of the variables of the model will affect the results, and so their adjustment is crucial, as shown in figure 3.25 after some sensibility test, and fitting the Gaussian distributions that defines the possible births within the frame to the entrance points in the video. Because of the accuracy of *YOLOv3* as a detection algorithm, we adjusted the variables of the survival probability "*model.P_S*" and the probability of detection in measurements "*model.P_D*" to .999999 as an approximation to 1, we can't assign the value of 1 because of how some calculations are made that would generate *NaN* errors. As it can be seen, we get an stable tracking in the general part of the true tracks, but we still have errors occurring in the intersections. Also as mentioned before, a big problem that is not completely resolved is the lack of tracking in some segments of the trajectories. Even by assigning values of probability of detection in measurements as close as possible to "1", there still ones that are not selected nor considered enough as to add them to the tracking of the object.

A lot of false positives are generated in the beginning and end of this trajectories, this can be explained by a general understanding of the problem, mainly that he pedestrian walking in and out of frame are doing so in closed street (fig 1.5), so all movements will be happening inside this delimited area, and so they are entering and leaving frame through the same places. This implies that the birth Gaussian distributions will be in the set in the same place where the true tracks die, so is understandable that all trajectories will be very prone to get assigned as a new birth and get Re ID if it goes through one of these Gaussian distributions.

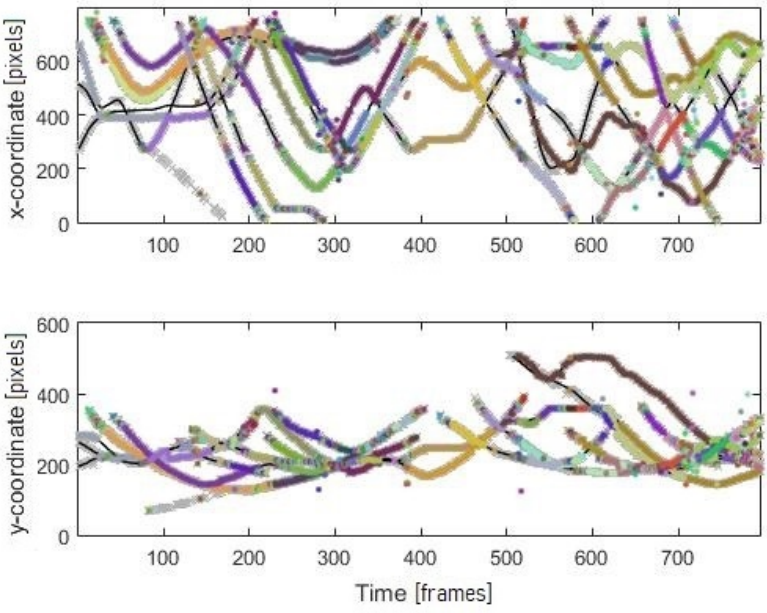


Figure 3.25: Tracked objects trajectories.

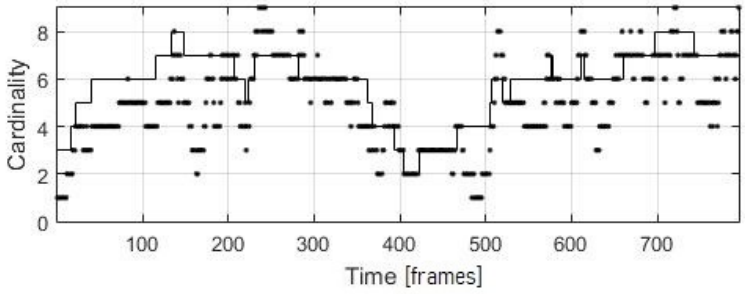


Figure 3.26: Number of trajectories per frame.

From figure 3.26 we can see that there is no consistency though time of the number of objects tracked, not maintaining any value for much time compare with the ground truth where its values are less variable. Although a stable number of objects tracked doesn't assure a correct tracking of the objects in the video, this variability does confirm that there isn't one. This directly influence the *OSPA* results in figure 3.27, as one of the two parts of its is to check the correct amount of tracks compared with the ground truth. This will cause all of *OSPA* values to be higher than we could expect, this doesn't mean that the tracking in general will be completely faulty, so it must be taken with a grain of salt when analysing its performance.

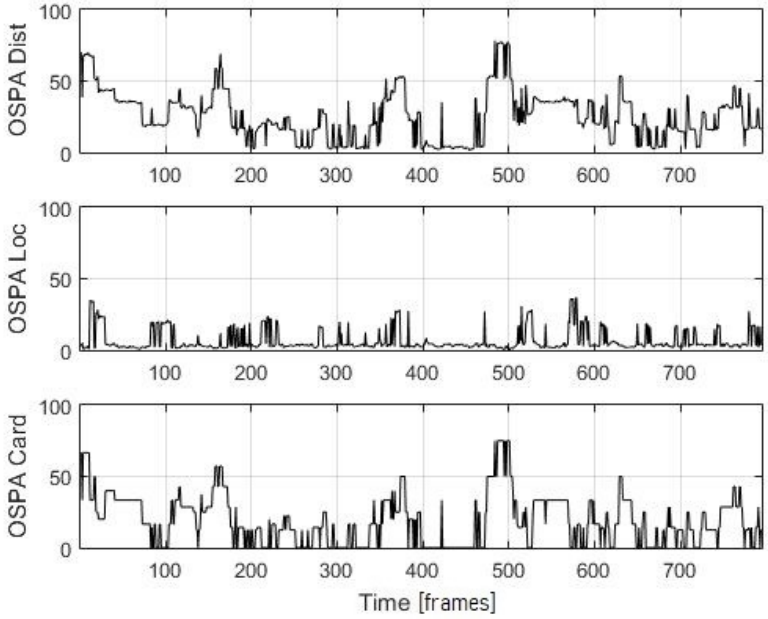


Figure 3.27: *OSPA* measurements.

With a better adjustment of the values many of the problems mentioned before can be reduced as seen in figure 3.28, where trajectories has less changes of id (fewer color changes in each trajectory), which can also be seen in figures 3.30 and 3.31, were the number of trajectories through the video isn't changing as much as the previous example. As has been

mentioned before, the biggest contributor in the results of the *OSPA* metric in our data set is the amount of false IDs, as can be seen in figure 3.31, where the *Cardinality* plot has much more variability than the *Location* plot.

In the fig 3.29, the trajectory of the pedestrian that was not added to the ground truth and that has no true track, can be seen alone in the bottom left, is a great example of what happens when we have a high threshold in our detector, because if this there are some specific locations where it is not detected for a couple of frames and so when it reappears a new id is given, this happens 2 times and the third time it is just never reassigned an ID even having a simple movement. This will also generate a bigger error value in the *OSPA* metric that we have to consider at the moment of using it as tool to evaluate the tracker.

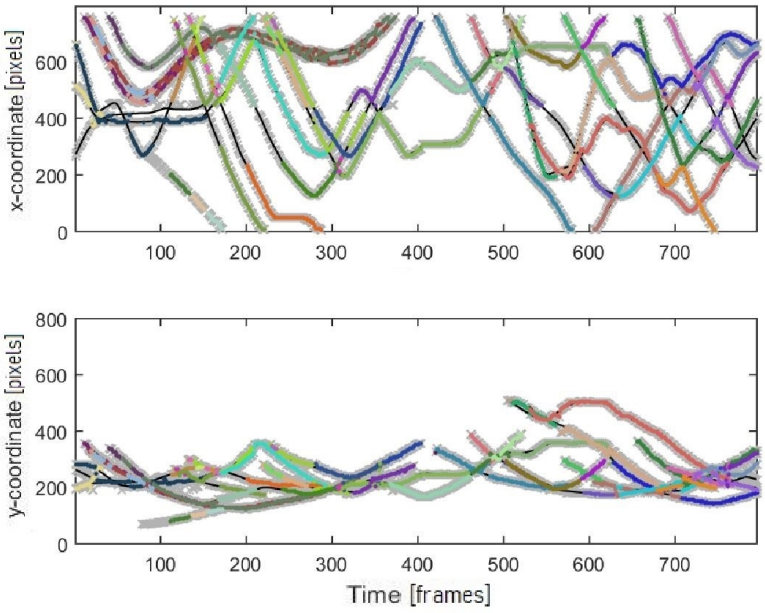


Figure 3.28: Tracked objects trajectories.

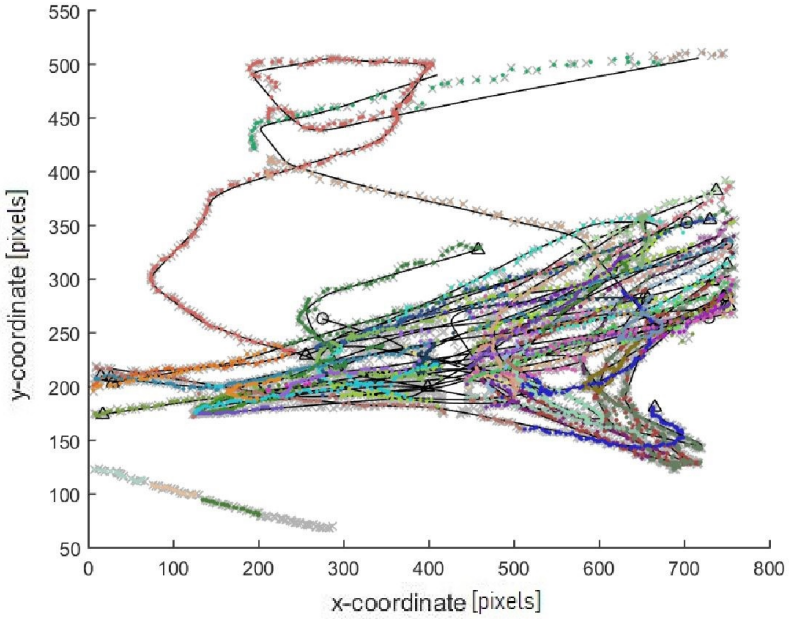


Figure 3.29: Tracked objects trajectories in 2D.

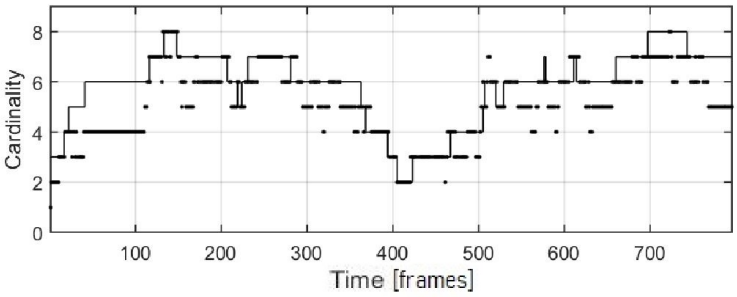


Figure 3.30: Number of trajectories per frame.

As shown in the table 3.2, the results are not as good compared with the best results of the MOT challenge. Although the results presented are of a number of different data sets while our results is obtain with only one of them, this mainly because we didn't have access to the ground truth of the rest of them. This is the reason why we don't show the values of IDSW and FAF, because is a sum of all those data sets.

Tracker	MOTA	IDSW	FAF
Ours	-229.7655	110	0.3522
BnW	42.9	-	-
CRF_RNN15	38.9	-	-
AP_HWDPL_p	38.5	-	-
AMIR15	37.6	-	-

Table 3.2: Result Comparation.

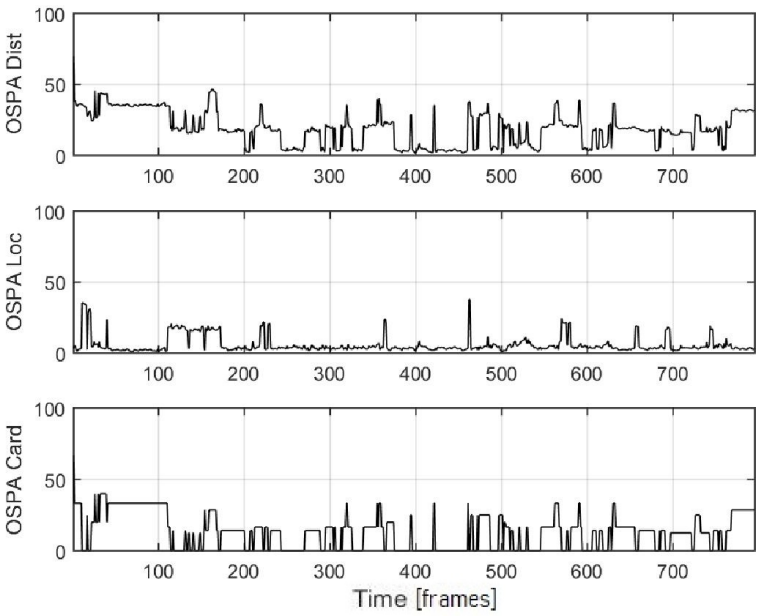
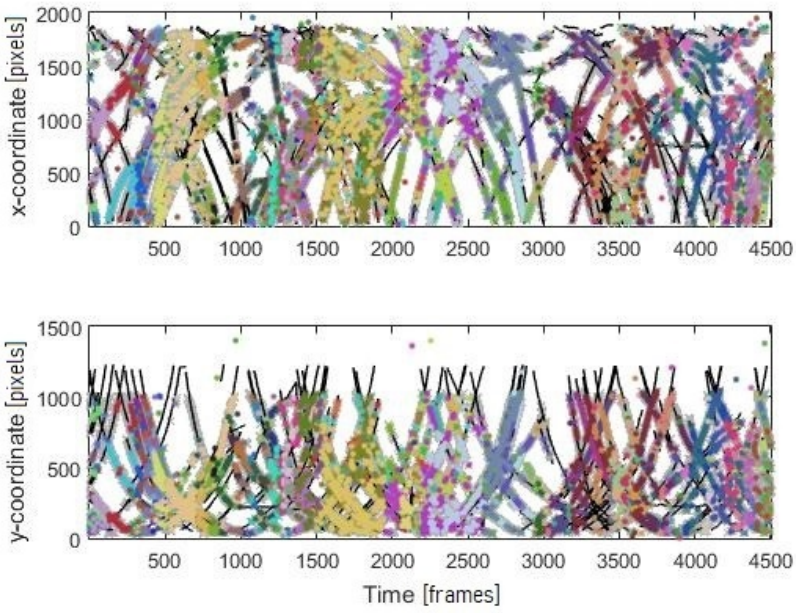


Figure 3.31: *OSPA* measurements.

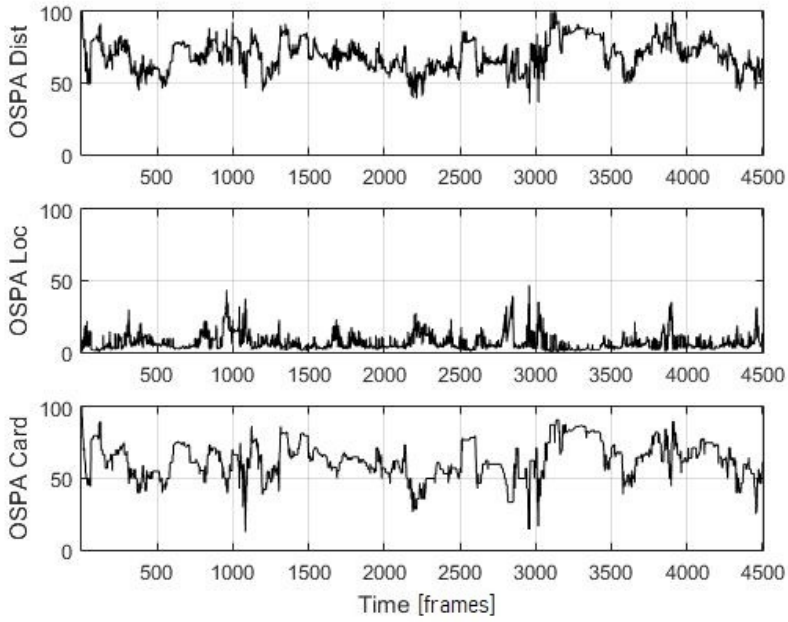
3.4.2 JointGLMB - Towncenter

As mentioned before, we were not able to find according values for the parameters so to generate better results for this data set, even though we choose this specific data set assuming it would generate better results by having more simple trajectories. The fig 3.32a is an example of this, the amount of objects tracked doesn't even allows to make a good visual analysis but the fig 3.32b let us know that too much false ids are been generated, this is the main reason the new visualisation plot isn't shown here.

Other problem that didn't allows us to to make further test, was that because of the high frame rate of the video, it was necessary a higher amount of frames to show the trajectories, by reducing the amount of frames less and shorter trajectories would be tested, while we are interested in see how they perform in a general case. But by testing with the full video the amount of time it takes for obtaining this results go to around 1:30 hours approximately per example.



(a) Towncenter Results.



(b) Towncenter *OSPA* results.

Figure 3.32: Towncenter Test.

Conclusion

Through this work, we tested how Random Finite Sets (*RFS*) algorithms perform on a problem outside of their scoop, while they are design to expect a high amount of noise in the measurements and more simple movements, we try them in a scenario with almost no noise but with a very unpredictable movements. This scenario been the tracking of pedestrian on video. We generated their measurement though state of the art implementations with high levels of accuracy and precision, compared with the detection generated with radar or laser (lidar) that are usually used in *RFS* implementations. All of this expecting to see if they could still perform correctly on this conditions. Also we studied how different variables of the model affect the results and how the visualization of the data can affect the analysis made.

From the results obtained, it was shown that adjusting the model parameters completely defines the results, and even though the final results presented for the PETS data set weren't as good as the ones shown in the MOT challenge, with more time and adjustment it is possible to get results closer to the ones expected. Also sensibility tests where made over these parameters, with the intention to understand in more deep how this affect the results, and learning from it, a more educated adjustment could be made for our data sets.

We used this new visualization of the data in conjunction with the already existing ones to complement the information we can obtain, using the X vs Y projection for a more accurate understanding of how close the measurements and estimation are to the true tracks, given that with the X/Y vs time plot wasn't able to truly show any discrepancy. On the other hand, this plot can let us understand the amount of tracks that happen in a specific time and combining the information of both, we can know the intersections that occur between tracks, and so see if a change of ID is a result of this or just a wrongly assigned new birth.

Also the mean of the *OSPA* distance was calculated for each example, because it allows to quantifies the "cuality" of the tracking done in each case. The original output of this measure was a plot against time of the whole distance and its decomposition of the location and cardinality, but with this we are able to compare different results and have a more objective differentiation between them than a visual analysis of their trajectories, and so giving us a more precise tool.

Other point we would like to emphasize is that this results can depend strongly on the

quality of the detection algorithm used. in this work we used an state of the art deep learning algorithm, *YOLOv3*, which at the moment of the realization of this work was categorized as the best general detector. This could be as a curse as a blessing, because of its high level of understanding of what a pedestrian is, it can generate confusing results that at the end could be more a problem than any thing else. Particularly for occlusions or intersections its able to correctly label a pedestrian, even if it is only partially visible. So through multiples frames a detection will be done in the same position, lets say next to a pole or other occlusive object, generating what in the data is a stationary object, but in reality is moving pedestrian with different level of occlusion. So when it finally appears by the other side its given a new ID an the stationary label dies. This is just one example of why we decided to use high levels of threshold for *YOLOv3*, even if it would hurt the results. Other reason of this is that even if its the best detector as right now, its not perfect, and for level still usually used in other applications, it gave us some false positives like assigning the label of pedestrian to a stationary cone in the video.

We would like to argue that even though the true tracks we had for the PETS2009 data set were very precise, a trajectory was missing, of a person walking on the back of the video, this is because the original use of this video is to test the count of people in areas and other problems were its more defined the space to work with, so because we were detecting this pedestrian our results would always had her detected, this affect the *OSPA* measurements, influencing in the final result of it.

The final comparison done with this results are done with the *MOTA* metric, this because the ground truth we work with only had the center position of the detection, but not its bounding box. Having this the *MOTP* can be calculated, and so we could have a more representative view of the results. The *MOTP* checks the intersection over union (IOU) of each detection assigned the corresponding label. As can be noticed, this metric relies completely on how this ground truth is generated, this is specially important in our case of pedestrian detection, since the definition of a "correct" bounding box may vary greatly depending on the convention you are considering, this because of the human movement while walking. The bounding box can consider just the head, torso and legs, with out taking account of the arms, because while walking human tend to extend them, generating a bounding box bigger than what it really is. This is why it was chosen to give more attention to the *MOTA* metric first, which doesn't vary so much from ground truth to ground truth.

Also although test were made for the Towncenter data set, we couldn't get good results for it, we believe that there must be some bug still in the code that we weren't able to detect. We expected better results for this data set because it is a video showing people walking in mostly straight lines down a street, so the tracking should had been easier to predict by been a more similar to what this algorithms are made for. Also because the video was in a higher frame rate we made some small test of the sensibility on the variable that adjust the time step of the model, but no noticeable improvement was measured, so was set a side so more time was given to the main test and results.

For the *EKF* we were able to see how the linearizing small portions of the trajectory can generate an oscillating pattern in the estimations by trying to readjust and constantly over-estimating the value because of trying to fit a linear equation where it should be a function of higher order so to evade this king of problems.

For improving the analysis as a whole, is necessary to do more testing and try other benchmarks for getting more precise results, this could include the addition of our own data to test in a real case scenario. Also the library itself has other filters and models that should be tested too, this was left out because of time, but should be considered for a more in depth study. finally is proposed for a future work to use the intrinsic qualities of the video and add visual descriptors to this implementations, since we are working with the raw data obtained from the video, but not taking advantage of this. A visual descriptor is any form of data obtained from an image that retains the visual characteristics of it, that allows for a classification or comparison between images that may contain the same information but are physically different. An example is how neural networks generate descriptors that allow them to classify different type of objects: cats, dogs, airplanes, etc... even though the photos or images contain different breeds of dogs or different model of planes.

Other area of machine learning that could be used for improving the results, would be genetic algorithms for adjusting the parameters. This kind of programs try to optimize a reward function design by the user and by imitating nature tries a population where individual has a different pool of value for the parameters to adjust. They are evaluated and then a new generations 'breed' maintaining the cases that give good results (*selection*) and generating a new pool of samples by a combination of them (*crossover*), adding small variations in some of them (*mutation*), mirroring how nature and evolution works. Thou its implementation would be of a great interest, their implementation is out of the scope of this work.

Bibliography

[1] Pets2009. *Eleventh IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2009.

[2] Mot challenge. 2016.

[3] Tim Babb. How a kalman filter works, in pictures. 2015.

[4] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008.

[5] Daniel Bryant, Ba Tuong Vo, Ba Ngu Vo, and Brandon Jones. The delta-generalized labeled multi-bernoulli tracking filter with target spawning. 05 2017.

[6] Kohei Fujimoto and Toshifumi Yanagisawa. Statistical track-before-detect methods applied to faint optical observations of resident space objects. 2015.

[7] Hung Gia Hoang, Ba Tuong Vo, and Ba Ngu Vo. A fast implementation of the generalized labeled multi-bernoulli filter with joint prediction and update. *IEEE 18th International Conference on Information Fusion*, 2015.

[8] JosephRedmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[9] Active Vision Laboratory. Towncenter. *Coarse Gaze Estimation in Visual Surveillance*, 2009.

[10] R. P. S. Mahler. Multitarget bayes filtering via first-order multitarget moments. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1152–1178, Oct 2003.

[11] Ronald Mahler. "statistics 102" for multisource-multitarget detection and tracking. *IEEE Journal of Selected Topics in Signal Processing*, 2013.

[12] Anton Milan, Laura Leal-Taixe, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *ArXiv*, 2016.

[13] Branko Ristic, Jamie Sherrah, and Ángel F. García-Fernández. Performance evaluation of random set based pedestrian tracking algorithms. *arXiv*, 2012.

[14] Branko Ristic, Ba Tuong Vo, Ba Ngu Vo, and Alfonso Farina. A tutorial on bernoulli filters: Theory, implementation and applications. *IEEE Transactions on Signal Processing*, 2013.

[15] Dominic Schuhmacher, Ba-Tuong Vo, and Ba-Ngu Vo. A consistent metric for performance evaluation of multi-object filters. 2008.

[16] Dominic Schuhmacher, Ba Tuong Vo, and Ba Ngu Vo. On performance evaluation of multi-object filters. *IEEE 2008 11th International Conference on Information Fusion*, 2008.

[17] James Teow. Understanding kalman filters with python. 2018.

[18] B.-N. Vo and W.-K. Ma. The gaussian mixture probability hypothesis density filter. *IEEE Transactions on Signal Processing*, 54(11):4091–4104, Nov 2006.

[19] B.-N. Vo, B.-T. Vo, and H. Hung. An efficient implementation of the generalized labeled multi-bernoulli filter. *IEEE Transactions on Signal Processing*, Apr 2017.

[20] B.-T. Vo and year=2013 month=Jul volume=61 number=13 pages=3460-3475 B.-N. Vo journal=IEEE Transactions on Signal Processing, title=Labeled Random Finite Sets and Multi-Object Conjugate Priors.

[21] B.-T. Vo, B.-N. Vo, and D. Phung. Labeled random finite sets and the bayes multi-target tracking filter. *IEEE Transactions on Signal Processing*, Dec 2014.

[22] Ba-Ngu Vo, Ba-Tuong Vo, Nam-Trung Pham, and David Suter. Joint detection and estimation of multiple objects from image observations. *IEEE Transactions on Signal Processing*, 2010.

[23] Ba Tuong Vo. Tracker library.

[24] Hung Gia Hoangand Ba-Tuong Vo and Ba-Ngu Vo. A fast implementation of the generalized labeled multi-bernoulli filter with joint prediction and update. 2015.

[25] Ángel F. García-Fernández, Jason L. Williams, Karl Granström, and Lennart Svensson. Poisson multi-bernoulli mixture filter: direct derivation and implementation. *arXiv*, 2017.