TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Trajectory Prediction for Telescope Based UAV Tracking

BACHELORARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Bachelor of Science (B.Sc.)

unter der Leitung von

Univ.-Prof. Dr.sc.techn. Georg Schitter
Dipl. Ing. Denis Ojdanic

eingereicht an der

Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von

Maximilian J. Vieweg
Matrikelnummer: 11806443

Wien, im Juli 2022

Advanced Mechatronic Systems Group
Gußhausstraße 27-29, A-1040 Wien, Internet: http://www.acin.tuwien.ac.at

Abstract

The rising availability of Unmanned Aerial Vehicles (UAVs) has led to an increase in their use for malicious activities. This development underlines the need for UAV detection and identification systems. Telescope based systems are currently researched extensively, which allow to extend the optical detection and tracking distance to long distances. Due to the narrow field of view of such systems, accurate target tracking is indispensable. Therefore, filter based approaches are of interest to estimate the UAV flight path and thus improve the system's tracking performance.

This work compares two different approaches to the trajectory prediction of UAVs, namely using a Kalman filter and a Particle filter. Their performance is evaluated through chosen metrics across relevant use cases, by comparing the predicted trajectory and the one measured by the tracker against the ground truth position of the drone. Especially the cases of continuing the prediction of the flight path with an obstruction of the line of sight of the object and having a faulty UAV tracker are evaluated.

This thesis shows, that in the general case of curved flight paths with added accelerations of the object of interest, the performance of the investigated KCF tracker could not be enhanced with the implemented filters. However, in cases of estimation failures, such as noisy or jumping trackers, tests demonstrate that the Kalman filter offers a better improvement on tracking quality, compared to the Particle filter. Even during UAV occlusions, the Kalman filter is able to keep the object tracked for an average of ten measurement iterations, while being more robust than the Particle filter.

Zusammenfassung

Die zunehmenden Verfügbarkeit von kommerziellen Dronen (UAVs) führt zu einem Anstieg von deren Missbrauch. Diese Entwicklung unterstreicht den Bedarf an UAV Erkennungs- und Identifizierungssystemen. Derzeit wird intensiv an teleskopbasierten Systemen geforscht, die es ermöglichen, den optischen Erkennungsradius auf große Entfernungen auszudehnen. Aufgrund des engen Sichtfeldes solcher Systeme ist eine genaue Zielverfolgung unerlässlich. Daher sind filterbasierte Ansätze von Interesse, um die Flugbahn des UAV zu schätzen und damit die Verfolgungsqualität des Systems zu verbessern.

In dieser Arbeit werden zwei verschiedene Ansätze zur Flugbahnvorhersage von UAVs verglichen, nämlich die Verwendung eines Kalman-Filters und eines Partikel-Filters. Ihre Leistung wird anhand ausgewählter Metriken für relevante Anwendungsfälle bewertet, indem die vorhergesagte Flugbahn und die vom Tracker gemessene, mit der tatsächlichen Position der Drohne verglichen werden. Insbesondere werden die Fälle bewertet, in denen die Vorhersage der Trajektorie mit einer Behinderung der Sichtlinie des Objekts fortgesetzt wird, oder in der ein fehlerhafter UAV-Tracker vorliegt.

Diese Arbeit zeigt, dass die Verfolgungsqualiftät des untersuchten KCF-Trackers im allgemeinen Fall von gekrümmten Flugbahnen mit zusätzlichen Beschleunigungen des UAVs ausreicht, um die Position des Objekts zu bestimmen. In Fällen von Fehlern des Trackers, wie verrauschten oder springenden Positionsschätzungen, zeigen Tests jedoch, dass der Kalman-Filter im Vergleich zum Partikel-Filter eine Verbesserung der Tracking-Qualität bietet. Selbst bei Verdeckungen vom UAV ist der Kalman-Filter in der Lage, das Objekt für durchschnittlich zehn Messiterationen zu verfolgen, und ist dabei gleichzeitig robuster, als der Partikel-Filter.

Contents

CHAPTER 1

Introduction

Unmanned Aerial Vehicles (UAVs) have become more and more widespread in recent years and are employed in a wide-ranging number of fields, such as infrastructure, agriculture or security [1]. Their applications lie mainly in aerial surveillance, maintenance and monitoring, where they offer an efficient and cost-effective alternative to established methods [1].

However, due to their rising popularity, there has been an increase in malicious use of UAVs. Not only may criminals using drones cause threatening situations , but security vulnerabilities make UAVs prone to hijacking and consequential misuse [2]. For example in 2018 two UAVs caused the Gatwick Airport to shut down for 33 hours, cancelling more than 1000 flights and affecting over 140,000 passengers [3].

This underlines the need for UAV detection and identification systems. Currently, most commercial available systems operate with either radar, radio-frequency, acoustic or visual signals [4]. While each technology has its own benefits and drawbacks, visual detection allows acquiring an image of the object and therefore enables human cross-examination. Camera based systems are currently under research [5], which rely on the output of computer vision algorithms to track an object.

In order to achieve higher accuracy, predictions of an incoming UAVs movement are necessary. This bachelor thesis compares two different ways to predict object movement, namely the Kalman and Particle filter, and evaluates their performance in differing scenarios. Especially, a focus will be set on the brief disappearance of an object behind another, e.g. a tree, or the UAV exiting the visual cone of the telescope. Chapter 2 will discuss the current state of the art and go into detail about the two main approaches of this thesis, the Kalman and the Particle filter. The general implementation of the system is discussed in the chapter three, which defines the system models used and the implemented software architecture. Afterwards the generated results will be compared using defined metrics and a conclusion will be reached.

CHAPTER 2

State of the Art

The aim of a trajectory prediction for Unmanned Aerial Vehicles (UAVs) is to improve on inaccuracies of the tracker. For this end, estimation algorithms are employed to predict the motion of incoming UAVs. This chapter will explain the basics of the Kalman filter and the Particle filter and then discuss recent works that use these filters to predict and/or estimate the motion of an object are described.

2.1 Kalman filter

The Kalman filter (KF) is a recursive set of equations, which estimate the state of a process, while reducing the mean and square error of the estimation. Due to its property of being an optimal filter, it is currently wide in use, especially in the field of autonomous or assisted navigation [6].
The Kalman filter is a useful tool for estimating the current state of a linear system x, with a measurement z. Consider the following equations representing the state of a system

$$x_k = Ax_{k-1} + Bu_k + w_k \ , \tag{2.1}$$

$$z_k = Hx_k + v_k \ , \tag{2.2}$$

where x_k refers to the state vector at the time step k, A is the state matrix, B represents the input matrix and H the output matrix. v and w refer to the process and measurement noise respectively, which assume to be normally distributed and independent.
To estimate the current and future state of the aforementioned system the following algorithm can be used [6]. Essentially the process consists of two steps, the time update and the measurement update. The time update predicts the following state of the system, while taking in account its current state and its error variance estimates. The measurement step incorporates the actual measured data and provides feedback for the next estimate.

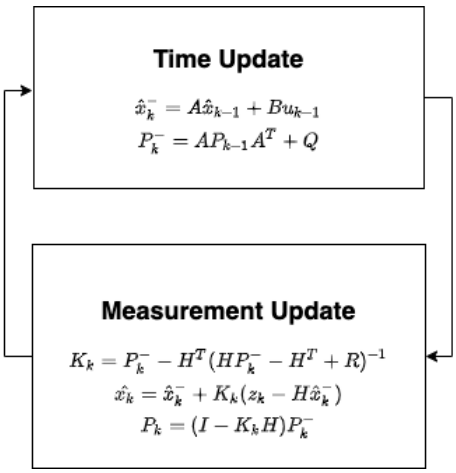

Figure 2.1: Kalman filter process: The time update provides a prediction of the future state, while the measurement update corrects the prediction with the measured value. These two steps alternate for each measurement made.

The time update is described as follows,

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \ , \tag{2.3}$$

$$P_k^- = AP_{k-1}A^T + Q \ . \tag{2.4}$$

Since the discrete implementation of the filter is relevant, due to the computer system, time steps are denoted with the subscript k, while the superscript "$-$" describes the system's *a priori* state, i.e. it does not yet take into account the new measurement. First, the predicted state of the system is computed using the state matrix A while taking into account the input from the last time step. The error covariance is updated accordingly with Q as the process noise covariance.
During the following step, the measurement step, the Kalman gain K_k is computed,

$$K_k = P_k^- - H^T(HP_k^- - H^T + R)^{-1} \ , \tag{2.5}$$

which describes the weighting between measurements and the current state of the system. After measuring z_k the estimated state is updated with an *a posteriori* value, as well as the *a posteriori* error variance as follows,

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \ , \tag{2.6}$$

$$P_k = (I - K_kH)P_k^- \ . \tag{2.7}$$

The Kalman filter continually runs recursively through these two steps, as illustrated in Figure 2.1. The presented method is memory efficient, as only the current state is compared to the previous, which makes it suitable for real time problems, such as in navigation and computer vision. It should however be noted that the noise functions are assumed to be Gaussian distributions, which can be a limiting factor for certain applications, such as localisation and mapping. [6].
To graphically illustrate how the Kalman filter works, consider Figure 2.2. On the left

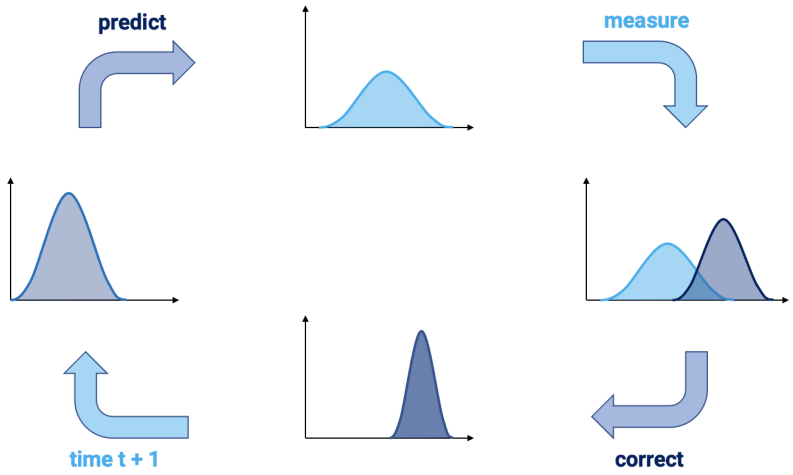

Figure 2.2: Kalman filter visualized: A first measurement on the left is used to predict the future state of the system. After a measurement the two normally distributed estimations of the state are compared and a final combined estimation, with a higher confidence is reached.

side a measurement with normally distributed noise is received first. Then the next state of the system is predicted, after which the estimate is compared to the actual measurement. By comparing these two distributions an estimation of the true state behind the measurements is possible. This way the former state as well as the current measurement affect what the state of the system is believed to be. Should either the former or the current measurement be faulty, a sufficiently accurate estimation may still be achieved. The weighting between the predicted state and the newly measured value is dependent on the Kalman gain K_k. A high gain puts more trust into the current measurement, whereas a low gain distrusts potentially noisy measurements [7].

2.2 Particle filter

A Particle filter is a recursive implementation of Monte-Carlo-based statistical signal processing, where the state of a system with discrete particles is simulated in order to find estimations about its state. The system is assumed to be a Hidden Markov Model [8], i.e. that the current state of the system depends only on previous observations, however no assumptions about the state space or the distribution of errors are made. This makes the Particle filter a commonly used method for visual localisation and tracking and the de facto standard for robot localisation. Due to their nature, they are increasingly becoming popular for systems with non-linear, non-gaussian and high dimensional properties. Other domains include visual analytics, and reinforcement learning [9].
For the Particle filter a set number of N state vectors z_k is initialised first,

$$Z_k = \{z_0, ..., z_k\} \ , \tag{2.8}$$

which shall be called particles, with an associated weight w_k, according to a probability function [10]. The probability function p of the posterior distribution can be described using the equation below,

$$p(x_{k+1}|Z_k) = \int p(x_{k+1}|x_k)p(x_t|Z_t)dx_t \ , \tag{2.9}$$

by taking into account each of the previous measurements. Since the true distribution of the data points is unknown, it is only possible to rely on the current and past measurements as an approximation for the particles. Taking into account the current measurement z_k, an update of the weight of each particle is made,

$$\begin{aligned} w_k^i &= w_{k-1}^i p(z_k|x_k^i) \ , \\ w_k^i &:= w_k^i / \textstyle\sum_i^N w_k^i \ , \end{aligned} \tag{2.10}$$

according to how close it comes to the measurement, denoted by the probability function p. After that the weights are normalised [11]. The predicted state of the system is found, by taking the weighted average of each of the state estimations,

$$\hat{x}_k \approx \textstyle\sum_{i=1}^N w_k^i x_k^i \ , \tag{2.11}$$

due to the reweighing according to the closeness of each one of the particles. Importance sampling is used in order to reduce the variance and efficiency of the distribution. In other words the particles are resampled the particles based on the current estimate of the probability density function and obtain more particles where they are expected to be during the next measurement and less, where they would be unlikely to be. A graphical illustration of this process is shown in Fig. 2.3. An even distribution of particles can be seen, which is then distributed during the resampling step according to the PDF in the background [12].
An algorithm describing the process [10] is given below:

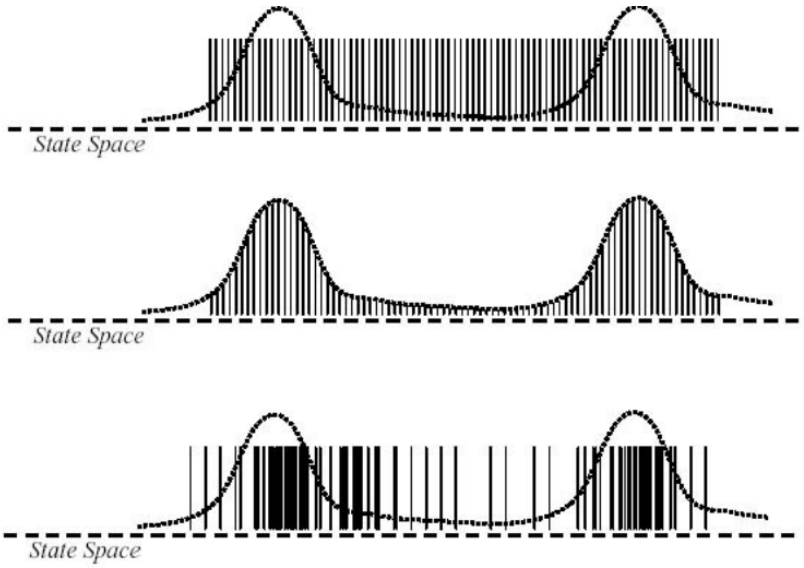

Figure 2.3: Importance Sampling: First N particles are distributed along a prior belief, in this case a uniform distribution; the posterior is represented by the dotted line. Afterwards each of the particles is weighed according to their fit to the posterior measurement. In the final step the resampling leads to a particle distribution resembling the posterior distribution [12].

Algorithm 1: Particle filter Algorithm: For every received measurment the weight of each initialised particle is updated according their closeness to the measured value. The final prediction is obtained using the weighted average of the particles. The prediction is obtained using the weighted average of the particles. Finally they are resampled according to these weight.

Data: Measurement in form of an image
Result: A Prediction of the next state
Initialisation
$X_0 \leftarrow \{x_0, ..., x_k\}$;
while *getting measurements* **do**
Measurement
$z_k \leftarrow \{z_0, ..., z_j\}$;
Measurement Update
$w_k^i \leftarrow w_{k-1}^i p(z_k|x_k^i)$;
$w_k^i \leftarrow w_k^i / \sum_i^N w_k^i$;
Prediction
$\hat{x}_k \leftarrow \sum_{i=1}^N w_k^i x_k^i$;
Resampling
$X_k \leftarrow \{x_0, ..., x_k\}$, according to corresponding w_k;
end

2.3 Trajectory Estimation

After having explained the basics of the Kalman filter and the Particle filter, the following section describes a few approaches on the prediction of object motion.
Tracking algorithm for Pan-tilt-zoom (PTZ) cameras combining the Extended Kalman filter (EKF) with the Particle filter have been implemented [10]. Essentially the camera first scans the environment for an object of interest, once a target has been detected its motion is being tracked with EKF. Should the target get lost, a particle filter is employed to restore visual contact with the UAV. The combination of EKF and PF has achieved better results than using an EKF alone. A different system implements a method for moving target tracking based on CamShift Approach and Kalman filter [13]. CAMShift (Continuously Adaptive Mean Shift) is a colour-based object tracking method, where a sample colour histogram of an object is used, to maximise the pixel density of those colors in a frame. Using a CAMShift to continuously track the object and, in the case of occlusion, resorting to the KF has resulted in a more robust and more efficient implementation that using either filter alone. Both previous approaches have used alternating filter methods for different states of the system, however the Kalman filter may also be used on specific features of the input image. For example, in [14] an object is segmented and extracted by color. The Kalman filter then uses the extracted colour in HSI colour space as a feature to detect the moving target and adjusts its parameters, such as the occlusion ratio, during the estimation step. The procedure allows for robust tracking for real-world situations, especially in the case of occlusion or changing lighting conditions. An object tracking algorithm using an adaptive Kalman filter combined with Mean Shift (MS) is implemented [15]. First the KF predicts the center of an object, which is then used as an input to the Mean Shift. After the MS converges its position is fed back as a measurement for the KF. During each iteration the estimate parameters of the Filter are adjusted with the Bhattacharyya coefficient. As a result the object tracking is practical and robust and allows for the tracking of partially, as well as totally occluded objects. A combination of Kalman and Particle filters for target tracking have been implemented [16]. The probability density function of the particle filter is being updated with the help of a Kalman filter in the linear case and with an Extended Kalman Filter in the non-linear one. This helps to integrate the likelihood function and the prior distribution and in turn improves the accuracy of the particle filter, without sacrificing real time tracking behaviour. Particle filtering techniques for a number of features, such as shape, colour, motion and sound are tested and compared in Reference [17]. An observation model taking in multiple clues is presented as an alternative to single clue models, since it allows for the compensation of the weaknesses of each filter alone. Notably the ambiguity of colour cue and the need for a complete model in for shape modelling are mentioned.
Many approaches to object tracking have been made using Kalman and Particle filters, even for the use case of PTZ cameras. However, since mostly the research focus lies on general object tracking, the specific case of trajectory estimations for telescope systems has not been investigated. The goal of this thesis is to compare two popular filter types, the Kalman filter and the Particle filter in telescope systems. As shown in the following chapters assumptions can be made to simplify the tracking scenario and approaches to continuous tracking in case of faulty trackers or object occlusion are presented.

CHAPTER 3

System Implementation

The following chapter describes the details of the implementation of the system. First a model of the telescope is described, after which the state-space representation of the drone is discussed, taking into consideration the chosen model dynamics. The software architecture of the project, along with implementation details of the Kalman and Particle filter are explained in the following sections. Finally, the generation of test videos for the experiments of the following chapter is discussed, as well as metrics used to judge the accuracy of the system.

3.1 Kalman Filter Implementation Details

This section describes the parameters used for implementing the Kalman filter. First, the state extrapolation equation

$$\hat{x}_{k+1} = A\hat{x}_k + u_k + w_k \ , \tag{3.1}$$

is used, in order to be able to predict the future state using system dynamics. The state transition matrix A relates the current state to the future state, according to standard kinematic considerations, i.e.

$$s_k = s_{k-1} + v_{k-1}\Delta t + a_{k-1}\frac{\Delta t^2}{2} \ , \tag{3.2}$$

where s_k refers to the current distance at time step k, v_{k-1} and a_{k-1} refer to the velocity and acceleration of the object, respectively. Δt describes the time between subsequent time steps. The parameters of the state transition matrix can be read as follows

$$A = \begin{pmatrix} 1 & 0 & \Delta t & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.3}$$

where Δt again describes the time period in between the frames. For the implementation of the filter it is set to $\Delta t = 20$. The current estimation of the state of the system $\hat{x}_k$ consists of the object's current position in three coordinates, as well as its velocities and accelerations in all three dimensions

$$\hat{x} = \begin{pmatrix} x & y & v_x & v_y & a_x & a_y \end{pmatrix}^T \quad . \tag{3.4}$$

w_k as used in Eq. 3.2 refers to normally distributed noise, due to the inaccuracy of the system. As defined in Section 4.1 this noise is assumed to be normally distributed with $\sigma = 30\ pixels$.
Since the telescope system is able to rotate, the effects of these rotations is taken into account using

$$u_k = \begin{pmatrix} \Delta\phi_x & \Delta\phi_y & 0 & 0 & 0 & 0 \end{pmatrix}^T \quad , \tag{3.5}$$

where $\Delta\phi$ describe the angles of rotation between two time-steps. Since the frame rate of the camera is sufficiently high, the small-angle approximation can be made, in order to linearly add the turning angles of the mechanical setup.
As described in 2.4 the Q matrix refers to process noise. Its values are set as such

$$Q = \begin{pmatrix} q & 0 & 0 & 0 & 0 & 0 \\ 0 & q & 0 & 0 & 0 & 0 \\ 0 & 0 & q & 0 & 0 & 0 \\ 0 & 0 & 0 & q & 0 & 0 \\ 0 & 0 & 0 & 0 & q & 0 \\ 0 & 0 & 0 & 0 & 0 & q \end{pmatrix} \quad , \tag{3.6}$$

where $q = 10^{-5}$. The process noise refers to the idea, that the state of the system changes over time, it is set to a low value, since it is assumed that the model is an accurate description of the system. In this implementation it is also assumed that the noise in one of the system properties is independent of the others. The measurement matrix

$$H = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}^T \quad , \tag{3.7}$$

shows, how it is only possible to measure the x and y values on the screen. The velocities and accelerations of the tracked object are estimated through the KF itself. The measurement covariance matrix as used in 2.5 is defined as follows

$$R = \begin{pmatrix} r_x & 0 \\ 0 & r_y \end{pmatrix} \quad , \tag{3.8}$$

with $r_x = r_y = 10^2$ describes the inaccuracy of our measurements. Due to the testing scenarios of noisy trackers this value has been set to a high value, in order to reflect the deviations of the measurement. Since there is no correlation between the measurements in x or y direction, they are assumed to be independent. Finally, the initial uncertainty covariance matrix P_0 is set, with

$$P_0 = \begin{pmatrix} p_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p_0 \end{pmatrix}, \tag{3.9}$$

$p_0 = 10^{-3}$. A comparably low value is chosen, leading to a high Kalman gain, resulting in a high trust of the first few measurements of the filter.

3.2 Particle filter Implementation Details

When the *ParticleFilter* class is instantiated, it first initialises a set number of particles $n = 1000$ randomly, according to a normal distribution. In this implementation of the Particle filter only the current two-dimensional position, as well as its corresponding velocities are considered. The particles are described using the following form, where the subscript i indicates the current particle index

$$x_i = \begin{pmatrix} x_i \\ y_i \\ v_{x,i} \\ v_{y,i} \end{pmatrix}. \tag{3.10}$$

Each particle is initialised according to a Gaussian normal distribution with a set standard deviation,

$$f_Y(x) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x-m)^T \mathbf{\Sigma}^{-1}(x-m)\right), \tag{3.11}$$

configurable in its JSON file. For each frame the *predictObject*() function is called, which first predicts the next state of each particle. This is done by adding each particle's current velocity to its location. In other words the $v_{x,y}$ values correspond to the amount of pixels one object may move between each iteration. After having predicted the future state, the observation obtained from the tracker is compared, via the *updateWeights*() function, as described in the algorithm for the Particle filter in Eq. 2.10. A multivariate Gaussian distribution can be used to then update the weight of each particle proportional to its distance from the observation. The general formula can be seen here 3.11. Now that the weights have been changed proportionally to the distribution, they only need to be normalised and resampled, just as described in Section 2.9 in the Particle filter introduction.

3.3 Software Architecture

The general software structure for the project is described in the following diagram Fig. 3.1. A central Controller starts threads on the computer system. Each thread is

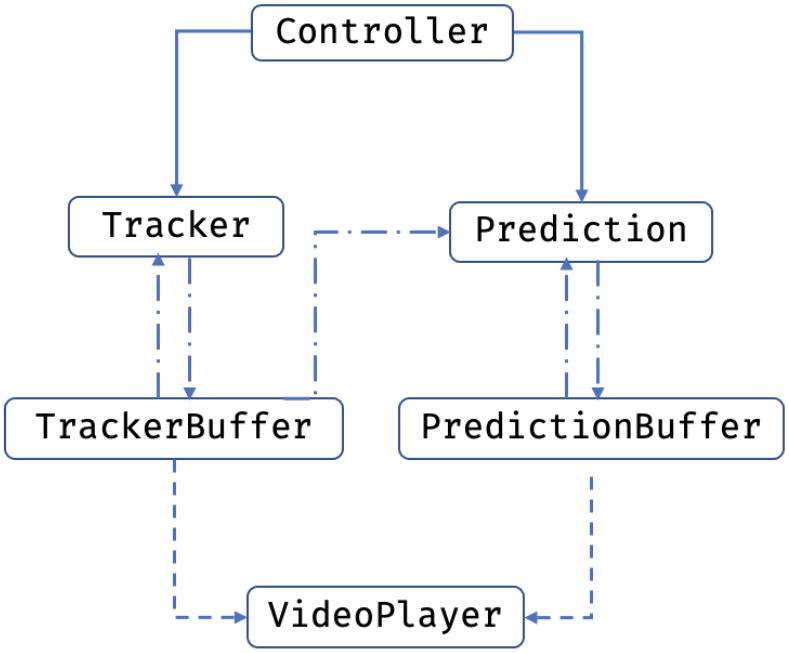

Figure 3.1: Sequence Diagram for Prediction Thread: A central controller starts threads. The *Tracker* provides the *Prediction* with positional data. Both the tracked and predicted value are then sent to the *VideoPlayer* in order to be displayed on the screen.

responsible for specific tasks, such as displaying an image to the user (*VideoPlayer*), tracking a UAV (*Tracker*) or predicting the next coordinates of the UAV (*Prediction*). Threads are able to pass messages between each other by writing them to the appropriate buffers. The modules shown in Fig. 3.1 are important. After the Controller starts the necessary threads, the *Tracker* receives information about the position of a UAV on the camera image by reading from the buffer. Using information provided by the simulator described below, the simulated tracker can retrace the ground truth flight path and compare estimation differences between the *Predction* and *Tracker* in the case of occlusion. The *Prediction* module reads out the current (u, v) coordinate pair from the *Tracker* module and performs calculations further described in 3.4. It then provides its prediction to the *VideoPlayer*, where both the tracker's estimate, as well as the predictions are rendered in the form of bounding boxes.

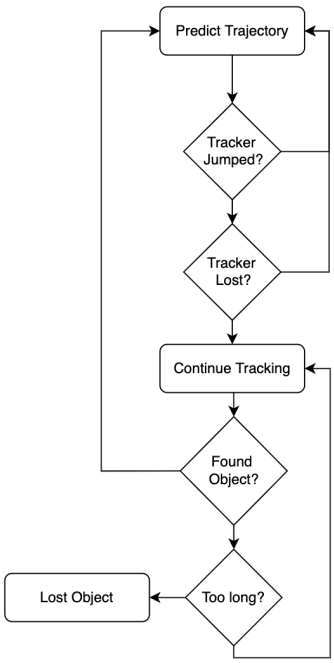

Figure 3.2: Flowchart for Continuing Tracking

Fig. 3.2 illustrates the general process of the trajectory prediction. Once the tracker is initialised the module can be in either of three states; the "Predict Trajectory", the "Continuing Tracking" and the "Lost Object" state. In the first state the future trajectory is predicted from information of the tracker using the current filter. The "Continuing Tracking" state continues to predict the trajectory using the prediction it made during the last iteration as the measured input. The last state is set, when the object is definitely lost. During each of the iterations of the "Predict Trajectory" state, it is checked, whether the tracker has jumped, i.e. it suddenly reappears on the other side of the screen due to tracking errors, or whether the target has been lost, e.g. due to occlusion. Should either case occur, the future position will be predicted for a set number of time steps, until which the object may reappear. Should this number cross a predefined threshold, the object can be considered lost.

3.4 Implementation Details

The general class structure may be seen in Fig. 3.3. The *PredictionBase* provides a uniform interface across the two subclasses *KalmanFilter* and *ParticleFilter* and is responsible for changing modes between following the tracker or estimating the motion of a lost target.
The *KalmanFilter* first reads out necessary configuration data from the corresponding handler, after which it instantiates the Kalman Filter objects and matrices from the *OpenCV* package. Here it is decided to utilise OpenCV's robust implementations for speed and stability. Inside the *KalmanFilter* class the following matrices are used to estimate the state (3.4, 3.3). In order to achieve a better performance and to be able to account for turning movements of the drone acceleration in x and y direction are

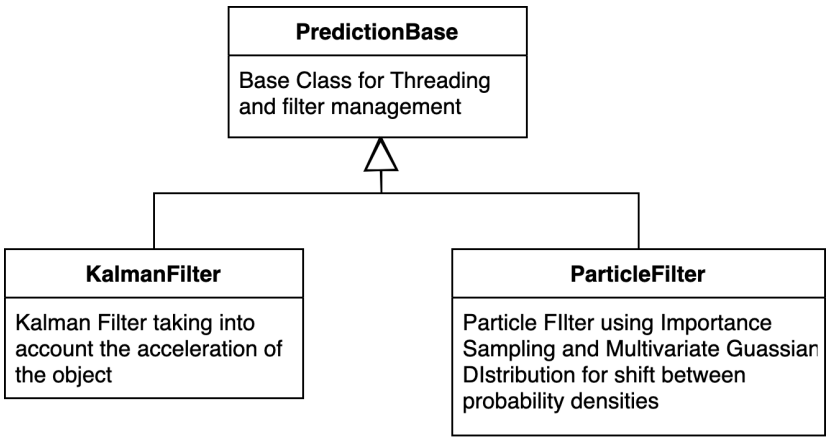

Figure 3.3: Class Diagram for Prediction: The *PredictionBase* class acts as the super class for both filters, uniting common functionality and a common threading configuration.

considered as well. What needs to be noted is, that the there is no built-in way to instantiate the first position the object is seen in. Due to this the assumed starting position for the Kalman Filter is $(0, 0)$, causing an overshoot at the beginning. To counter this the initial state is saved as an origin point onto which the estimations are added. As in Fig. 3.2 the *predictObject*() and the *continueTracking*() routines are called, depending on the current state, the module is in. Should the object get out of focus and stop being tracked, the *continueTracking*() routine is initialised. For a preset time interval the predicted position of the drone is fed back as a measurement for the Kalman Filter. This way, instead of linearly extending the target's motion, curved motions of an UAV may also be captured. Essentially the *predictObject*() function is continuously called, which reads out the current (u, v) coordinates from the tracker.

3.5 Generation of Test Videos using Simulator

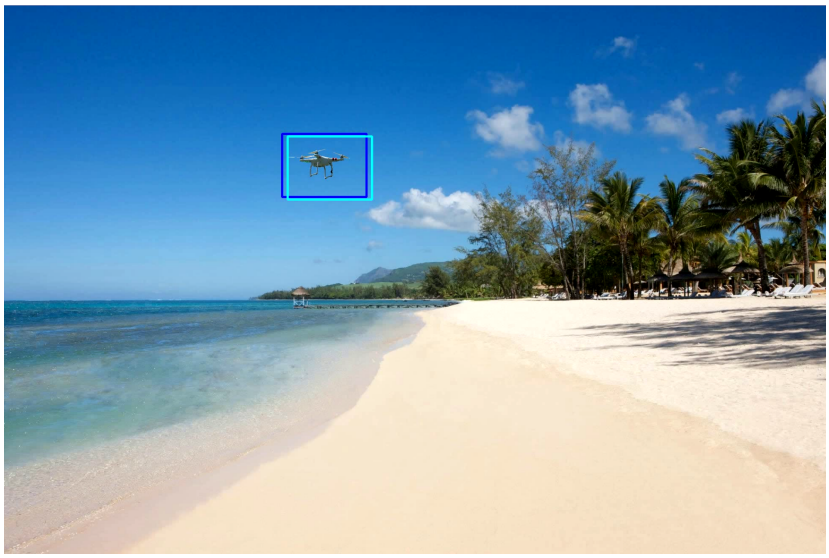

Figure 3.4: Simulator Video: Using the background of a sunny beach with open areas, such as the sky and visually demanding backgrounds, such as the palm trees, a moving drone can be seen in the sky. The bouding box defined by the tracker is shown in dark blue; the turquoise bounding box is the prediction provided by the current filter.

Since acquiring accurate data with full positional information about flight objects is difficult and time-consuming, it is decided to generate videos using a simulator, which allowed for specifying various flight paths of a UAV, along with timestamped data on its position. This makes rapid testing of different scenarios possible, as well as adapting parameters in case of needed corrections. Essentially a set number of points the object is travelling between is specified, and a flight trajectory is constructed. Using this trajectory, the position of an object, e.g. a UAV, is displayed on the video with its size dependent on the distance to the observer. This way objects simulated farther away automatically appear smaller in the testing video. As part of the configuration different background images can be specified, allowing for the simulation of visually cluttered, noisy backgrounds. As part of this thesis the possibility for changing the speed mid-flight according to specified functions is added. Fig. 3.4 shows a frame from one of the output videos of the simulator onto which the tracking and prediction have already been applied to. The blue bounding box shows the current estimate by the tracker, whereas the turquoise bounding box predicts the next location of the UAV.

3.6 Evaluation Metrics

3.6.1 Intersection over Union (IoU)

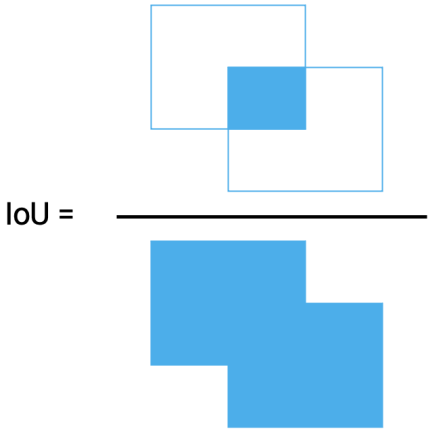

Figure 3.5: Intersection over Union: The IoU is calculated by dividing the area of overlap between two bounding boxes, by their total area.

The Intersection over Union, also called Jaccard index, is a metric used for evaluating the overlap of two boxes. It is often used in object detection and has become the standard way to compare how accurate systems are, compared to a ground truth [18]. It can be described using

$$IoU = \frac{|A \cap B|}{|A \cup B|} , \tag{3.12}$$

where A and B stand for the areas of the corresponding bounding boxes, $\cap$ describes the intersection and $\cup$ the union of those boxes. A graphical representation can be seen in 3.5. Generally in the case of prediction an IoU of $> 50\%$ shall be sufficiently good for estimating the future position [19].

3.6.2 Normalised Distance to Ground Truth (NDGT)

The Normalised Distance to Ground Truth can be obtained by calculating the distance between the centers of the predicted bounding boxes and the ground truth as follows,

$$d_{norm} = \sqrt{\left(\frac{x_{pred} - x_{truth}}{w}\right)^2 + \left(\frac{y_{pred} - y_{truth}}{h}\right)^2} , \tag{3.13}$$

In order to have comparable results these values are normalised by dividing the width w and height h of the bounding boxes [20].

CHAPTER 4

Results

This chapter will discuss the results of the implemented trajectory prediction and compare the two different filter types. In order to achieve comparable results test videos have been created from a simulator according to the scenarios described in this chapter. First, a comparison of the predicted frames to the observed ones is made, after which the performance of each filter type is discussed according to each scenario.

4.1 Test Scenarios

The two implementations of the Kalman filter and Particle filter are tested on the following scenarios, in order to cover common use cases for trajectory prediction. First the prediction module's performance is evaluated against the ground truth position given by the scenario. Afterwards normally distributed noise with a standard deviation of $\sigma = 30\ pixels$ is added to the ground truth position, in order to test the robustness of the prediction. Finally, a Kernelized Correlation Filter (KCF) is used in once in combination with noise and once without, in order to simulate realistic conditions.

4.1.1 Linear Motion with Constant Velocity

In order to have a baseline for comparison, the trivial example of object trajectory prediction in the case of linear motion with constant velocity is included in the analysis. In the three-dimensional space of the simulator the object moves between two set points with a set distance of $v = 15$ pixels per time step. Since this motion leads away from the simulator, in the resulting test video, the UAV becomes smaller, the farther it gets from its original coordinates.

Figure 4.1: The drone begins at its starting position A in the top left corner and ends at B at the bottom right. Due to its movement away from the observer, it is scaled accordingly.

Fig. 4.2 shows the approximate movement of the UAV over the course of the experiment. It should be noted that complexities in the background are indirectly related to the performance of the prediction, since its accuracy is dependent on the tracker. Visually complex scenery negatively impacts the tracker by making it get stuck on unrelated parts of the image.

4.1.2 Curved Motion with Constant Velocity

The following test scenario describes the curved motion of a UAV using the same velocity as in the previous section. The object follows a curved path between three set direction points with $v = 15$.

Figure 4.2: The drone begins at its starting position A in the top left corner and continues its curved motion until it reaches its end position, also in the upper left. Due to the chosen coordinates, the distance to the observer is approximately constant during the video, resulting in a uniform size.

Similarly to the previous section, the UAV starts in the top left corner, flying two curves around the center of the screen before returning to the start. It should be noted that towards the end, the UAV almost leaves the screen, allowing for the testing of this

behavior during the curved flight path. Due to its flight path the scaling of the object stays approximately the same throughout the entire experiment.

4.1.3 Linear Motion with Stopping and Acceleration

This test case describes the linear motion of the UAV with its velocity changing during flight. For comparability, the trajectory chosen for this experiment is the same as in 4.2. The velocity of the UAV for the creation of the object in the simulator can be described by the following equation:

$$v(t) = at^3 + bt^2 + ct + d, \tag{4.1}$$

where a, b, c are parameters of the polynomial function and t describes the current time step. The value of v can be understood as the number of pixels the UAV moves in the three-dimensional space each time step. In the specific case used for the thesis, the speed function can be graphically illustrated as such:

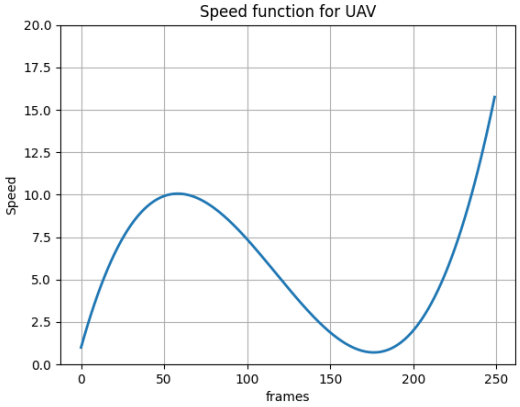

Figure 4.3: Speed Change during flight of UAV: The speed parameter describes the number of pixels moved between two frames in the three-dimensional space.

It is important to capture at least one acceleration and one stopping phase of the UAV, as well as one stopping phase. The speed first increases fast, after which the object decelerates, coming almost to a stop. Finally, the motion is ended with a strong acceleration phase.

4.1.4 Curved Motion with Stopping and Acceleration

The Curved Motion with stopping and acceleration follows the same trajectory as described in 4.1.2. Again, for comparability it follows the same speed function as 4.5.

4.1.5 Faulty Tracker jumping around Screen

As in 4.1.2 the curved trajectory with constant velocity is used. In this case, however, a faulty tracker jumps across the screen at random intervals for short periods of time. The prediction should counter these errors until the tracker correctly follows the target again.

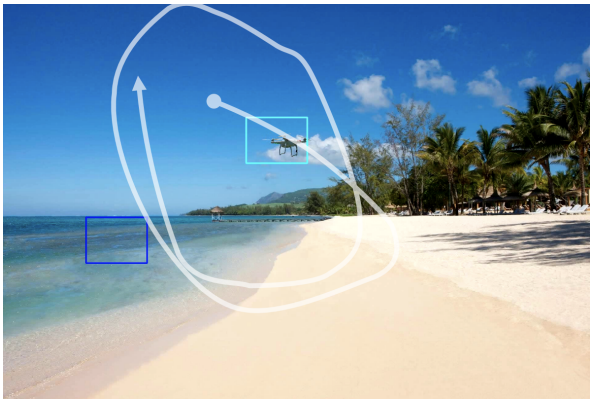

Figure 4.4: Faulty Tracker: The tracker does not accurately follow the object on the screen. During random time intervals it jumps to different locations (blue), while the *Prediction* module (turquoise) continues tracing the trajectory.

This can be seen in Fig. 4.1.5: The same trajectory as in the previous examples is used with a constant velocity. The figure shows a moment, where the faulty tracker (blue) has jumped from the object, while the prediction (turquoise) keeps the UAV in focus.

4.1.6 UAV being occluded during Flight

A common and important use case for the trajectory prediction is still being able to follow an object's movement, even if the tracker is not able to detect it anymore. In this testing setup, a palm tree obscures the UAV twice during its trajectory. During the evaluation, the metrics first compare both the tracker and the prediction with the object's ground-truth coordinates. Afterwards, a comparison between the prediction and the tracker is made. It is assumed that the tracker turns off when it cannot accurately track the object anymore due to an obscuration. The prediction module is not able to detect this case by itself, since it only relies on the tracker's data.

Figure 4.5: UAV Flight trajectory: The being occluded during Flight test case follows the same trajectory as the previous curved ones. However, during the test the UAV is occluded twice by a palm tree, as indicated by the transparent parts of the arrows.

During this test case, the UAV is being occluded twice during its flight time. The first time during a shorter period of time, when the object flies through the palm tree horizontally, and during a longer period of time in its vertical descent. As before, the UAV follows the same trajectory as in the previous curved ones.

4.2 Kalman Filter Results

The following section describes the results obtained for the test cases described in the previous chapter using a Kalman filter.

4.2.1 KF Straight Line Constant Velocity

This case describes the trajectory prediction using a Kalman filter for a linear motion with a constant velocity, as detailed in Section 4.1.1. Fig. 4.6 shows a highly accurate IoU

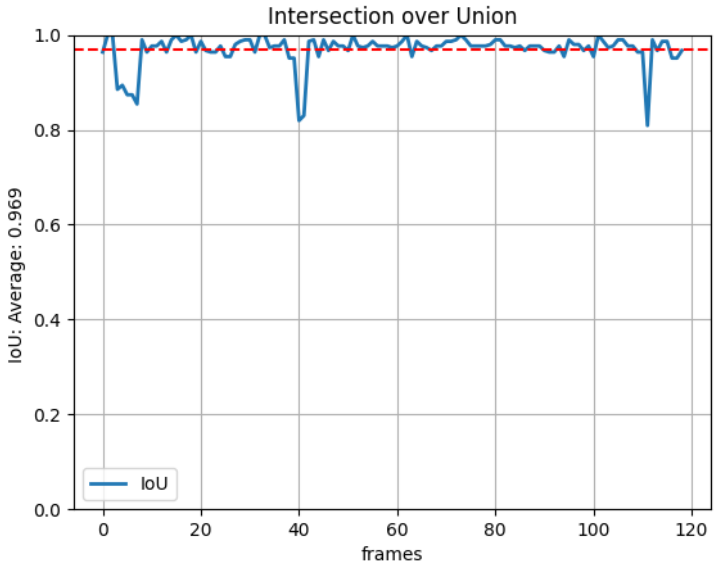

Figure 4.6: IoU for the straight line constant velocity case: This case shows a high accuracy of the trajectory prediction with an average $IoU = 0.969$.

between the ground truth and the prediction of the KF. With an average $IoU = 0.969$ the object remains tracked throughout the experiment, without a great variation in quality. As defined in Section 3.6.1 the minimum IoU for the prediction to be considered correct, lies at 50%; with a performance of overall $IoU = 0.969$ the prediction can be considered tracked. However, since this is the simplest of cases, it shall only be used as a reference for the prediction performance; i.e. other trackers should ideally perform as well as in this case.
Fig. 4.7 illustrates the IoU between the ground truth and the prediction bounding box, as well as a noisy tracker. During the experiment the location of the tracker stays within a normal distribution around its ground truth position, with a standard deviation of $\sigma = 30\ pixels$. As can be seen in the figure, the KF is able to filter out some of the Gaussian noise and almost consistently perform better in the tracking of the ground truth position of the UAV.

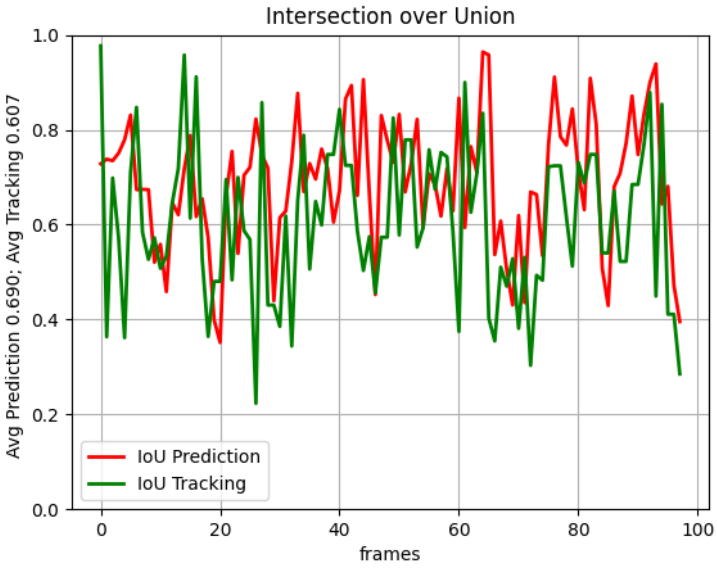

Figure 4.7: IoU for the straight line constant velocity case: The comparison between the noisy tracker, as well as the Kalman filter's improvement is seen here.

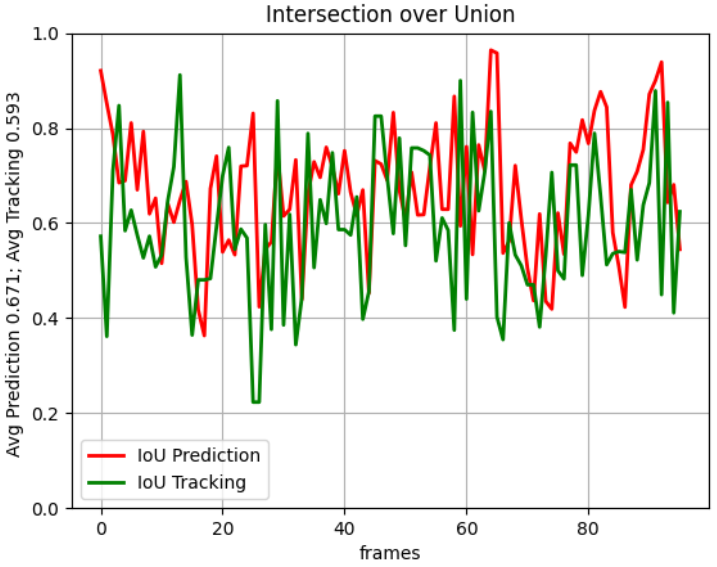

Figure 4.8: NDGT for the straight line constant velocity case: This test case performs similarly to the noisy tracker case, however sometimes the tracker gets stuck on visual clutter in the background. During this time the prediction follows the UAV, until the tracker can be reinitialized at the UAVs position.

Using a KCF tracker for the following experiment, the average for the IoU lies around $IoU = 0.16$ for the prediction, compared to $IoU = 0.266$ for the tracker. Although this is leads to a slightly lower performance compared to the ground truth, the object can still be consistently tracked using the prediction.

4.2.2 KF Curved Line Constant Velocity

This case describes the trajectory prediction using a Kalman filter for a curved motion with a constant velocity.

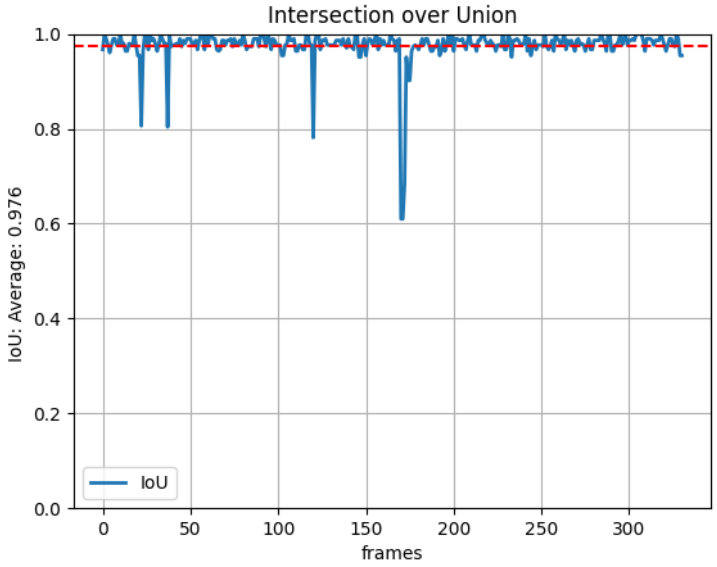

Figure 4.9: IoU for the curved line constant velocity case: The IoU stays approximately constant around an average value of 0.976. Except for a few peaks, the prediction is able to accurately follow the tracker.

Fig. 4.9 shows the IoU between the tracking bounding box and the prediction bounding box. With a performance of overall $IoU = 0.976$. This is higher than in the case of linear motion, which is not expected, since the curved motion is more complex than the motion in a straight line. This may suggest, that general noise due to non-uniform updates of the discrete system exceeds the error made by the filter itself.

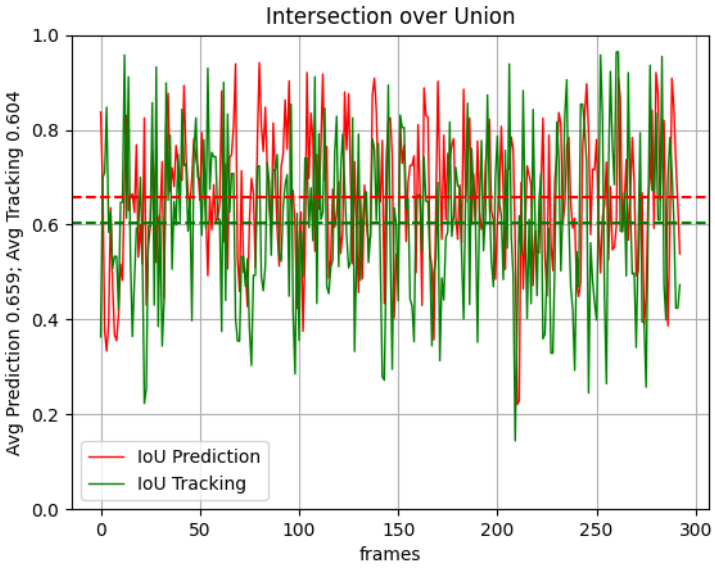

Figure 4.10: IoU for the curved line constant velocity case: The KF is able to improve on the noisy tracker, which is affected by Gaussian noise with $\sigma = 30\ pixels$ added to its position data.

While comparing the IoU between the prediction and the noisy tracker, which changes its position within a normal distribution around the ground truth with a standard deviation of $\sigma = 30\ pixels$, it can be seen that the prediction is able to improve on the tracker.

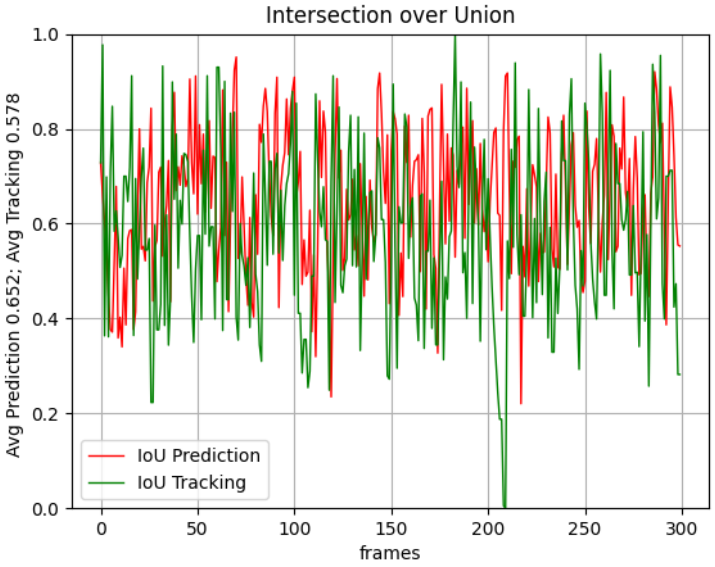

Figure 4.11: IoU for the curved line constant velocity case: The prediction stays around an average $IoU = 0.652 > 0.578$, being disrupted by small peaks when the KCF tracker is not able to accurately follow the UAV anymore.

Comparing the KCF tracker, with the prediction module shows a similar trend to the noisy tracker case. However, during the experiment the tracker became stuck because of the visually complex background, leading to a lower tracking performance during these time periods. After reinitialisation of the tracker at the object, the tracking quality continues similarly to the noisy tracker case. Since the biggest cause for poorer tracking performance lies in the tracker getting stuck in the background this can be expected.

4.2.3 KF Straight Line with Stopping and Acceleration

This case describes the trajectory prediction using a Kalman filter for a linear motion while accelerating and slowing down during the test scenario.

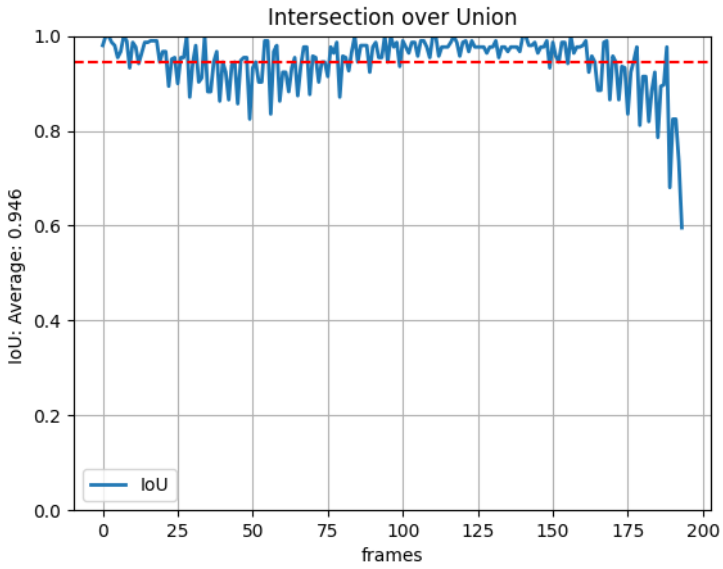

Figure 4.12: IoU for the straight line with stopping and acceleration case: In the IoU a clear relationship between the acceleration and stopping phases of the UAV can be noticed.

As can clearly be seen, there is a direct correlation between the velocity of the object and the error that the prediction makes. Generally, the faster the tracked object moves, the less accurate the prediction becomes.

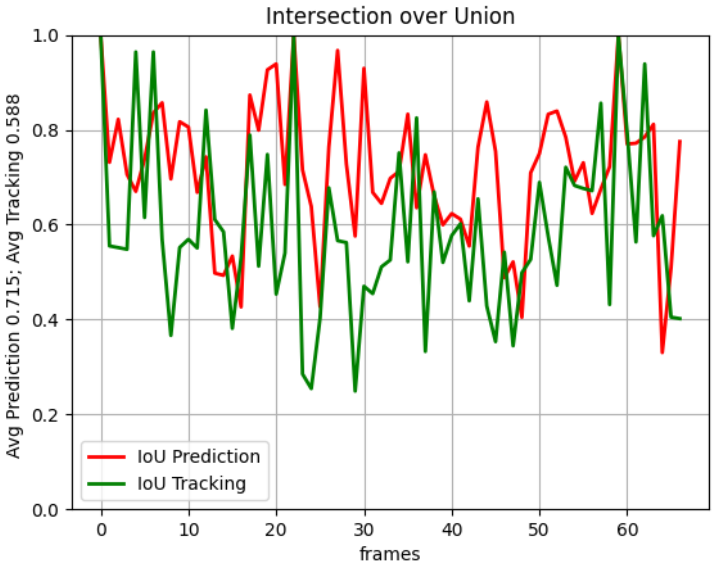

Figure 4.13: IoU for the straight line with stopping and acceleration case: In general the prediction outperforms the noisy tracker.

As in the previous examples the KF outperforms the noisy tracker, with an average $IoU = 0.715 > 0.588$. While the connection between acceleration and deceleration phases are clearly visible in the first example, it seems that the normally distributed noise $\sigma = 30\ pixels$ overshadows this effect in the experiment. Noticeably, the growing distance towards the end of the experiments, where the UAV starts accelerating again very strongly, can still be seen, which shows how sudden changes in speed have a big impact on prediction performance. It must however be noted, that the filter has continuously been able to correctly track the UAV.

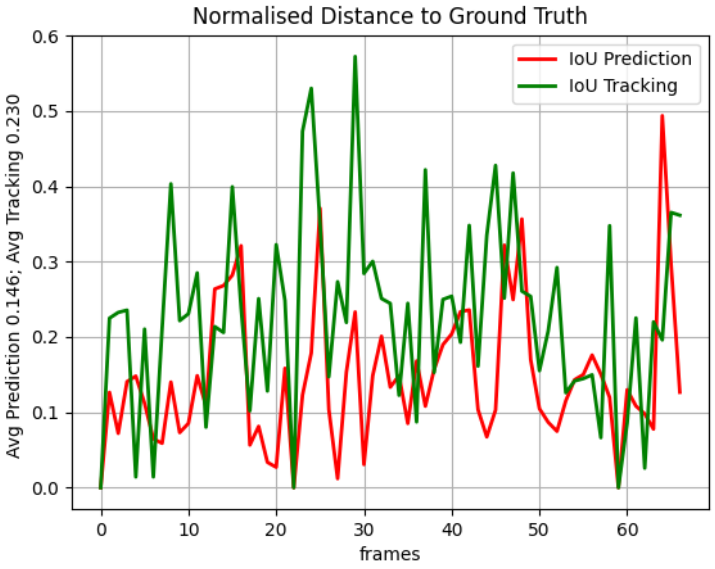

Figure 4.14: NDGT for the straight line with stopping and acceleration case: Acceleration and deceleration phases have a direct linear impact on the quality of the trajectory prediction, as can be seen by comparing this figure to Fig. 4.5.

The value of the NDGT stays low during the whole prediction process at $NDGT = 0.146 < 0.23$, indicating that the tracking error is bigger than the prediction error. When directly comparing this figure to Fig. 4.5 it becomes apparent that the acceleration and deceleration phases have a slight impact on the tracker performance, however the prediction quality is largely unaffected by this effect.

4.2.4 KF Curved Line with Stopping and Acceleration

This case describes the trajectory prediction using a Particle filter for a linear motion while accelerating and slowing down.

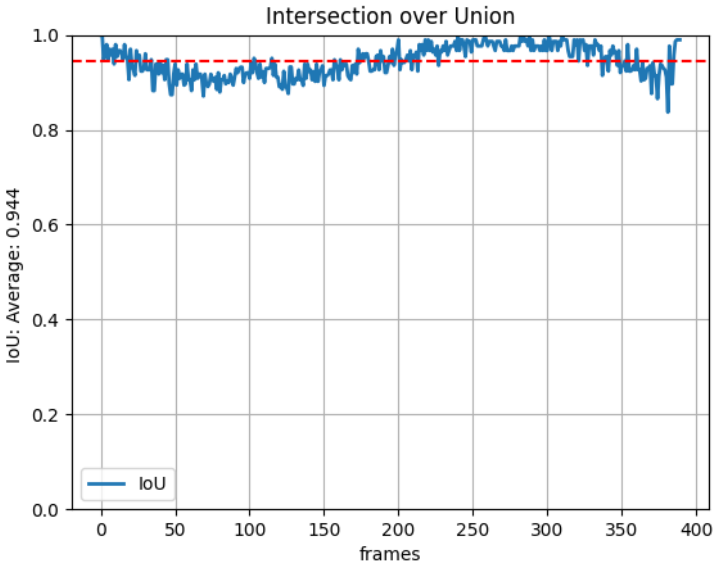

Figure 4.15: IoU for the curved line with stopping and acceleration case: The change in prediction performance can clearly be seen during the experiment; acceleration phases of the UAV are accompanied by poorer IoU.

As expected the IoU between the ground truth value and the prediction has a slightly lower average $IoU = 0.944$ than the Curved Motion with a constant velocity. As in the previous example the acceleration phases are clearly recognizable in the prediction performance.

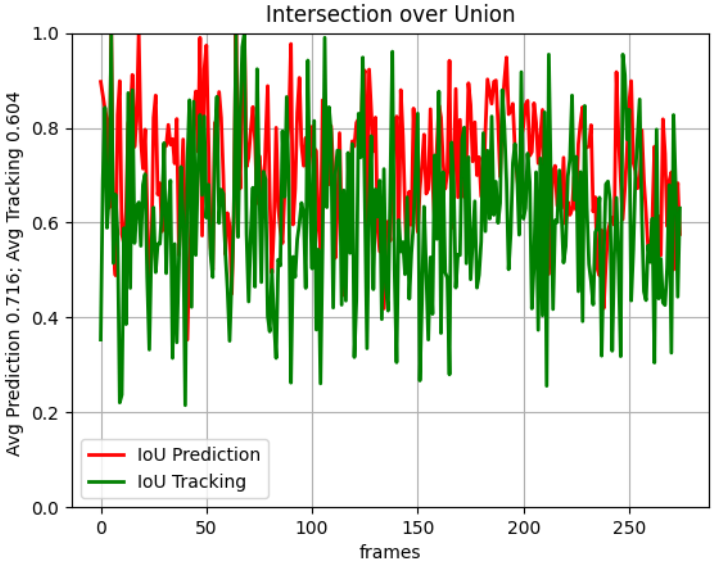

Figure 4.16: IoU for the curved line with stopping and acceleration case: The average prediction performance is higher than that of the tracker, almost throughout the experiment, with an average $IoU = 0.716 > 0.604$.

Fig. 4.16 shows how the KF is able to improve on the noisy tracker, keeping the object in focus throughout the experiment.

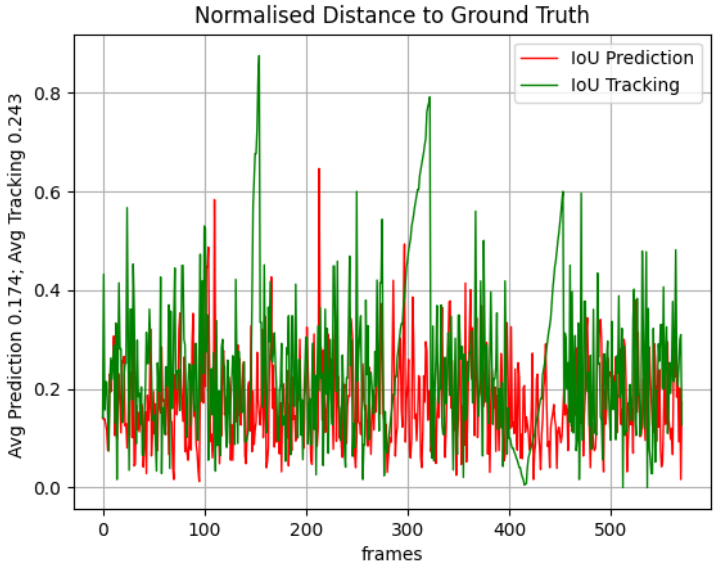

Figure 4.17: Normalised Distance from Ground Truth for the curved line with stopping and acceleration case: As in the previous case the increase in distance during acceleration phases can clearly be seen.

In Fig. 4.17 the effect of tracker getting stuck can clearly be seen. During the experiment the KCF tracker got stuck three times, leading to an increasing NDGT, while the prediction tries to continue its path along the ground truth. After the error becomes too big, the tracker is manually reinitialized on the UAV. Using the KF leads to an increase in performance during the times, the tracker is stuck.

4.2.5 KF Inaccurate Tracker

This case describes the trajectory prediction using a Kalman filter for a curved motion with a faulty tracker, which malfunctions in random time intervals by reappearing on different parts of the video screen for short periods of time.

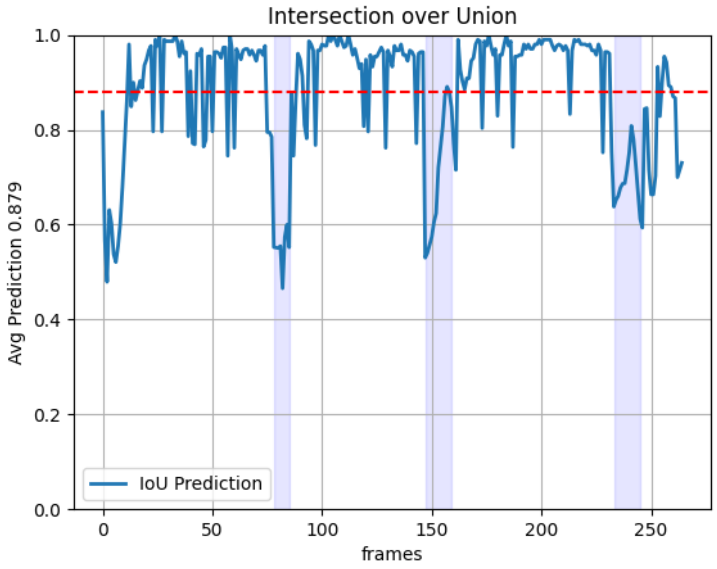

Figure 4.18: IoU between Ground Truth and Prediction for the Inaccurate Tracker case: In the course of the experiment the tracker randomly jumps to different parts of the screen, indicated by the blue shaded areas. While the IoU between the ground-truth trajectory of the UAV and the tracker falls to zero, the prediction is still able to follow for the duration of the tracker jump.

The first experiment shows the accuracy of the Kalman filter, following the ground truth tracker value of the UAV position. In three time intervals the tracker jumps to a random position on the screen. Even during these jumping phases the KF based prediction is still able to follow the UAV independently, until the tracker is reinitialized at the UAVs position.

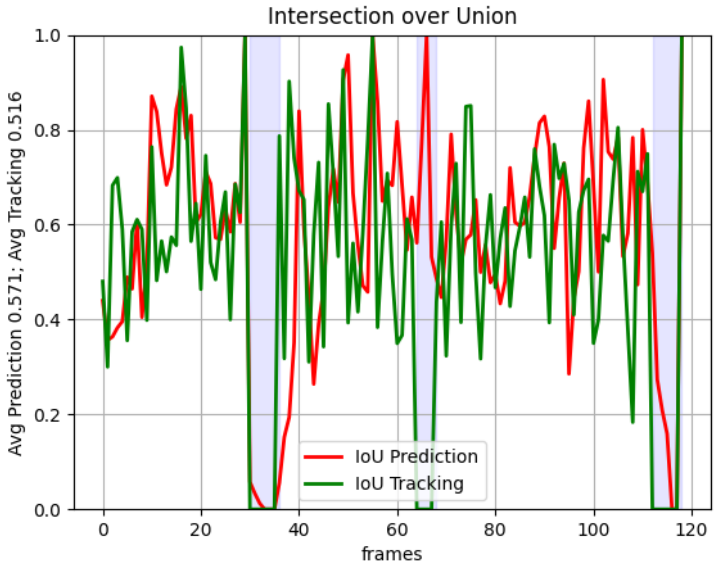

Figure 4.19: IoU between Ground Truth and Prediction / Tracker for the Inaccurate Tracker case: Due to the noise the prediction sometimes follows the wrong direction, as indicated in the first jumping period. This results in a poor performance. However, depending on the current flight path, the prediction may sometimes even drastically improve, as seen in the second period. The third is the expected case, where the IoU falls during subsequent time steps.

In Fig. 4.18 the IoU between the tracker and the ground-truth flight path, as well as between the prediction and the ground-truth are shown. The background shaded in blue indicates a jump of the tracker. This example shows the three main situations that can occur during the prediction: During the first jumping period the prediction moves into the wrong direction due to noise, leading to a low accuracy. In the second period, the prediction is perfectly able to follow the path of the UAV. In the third, the prediction partly tracks the trajectory, however quickly loses focus while the tracker has jumped.

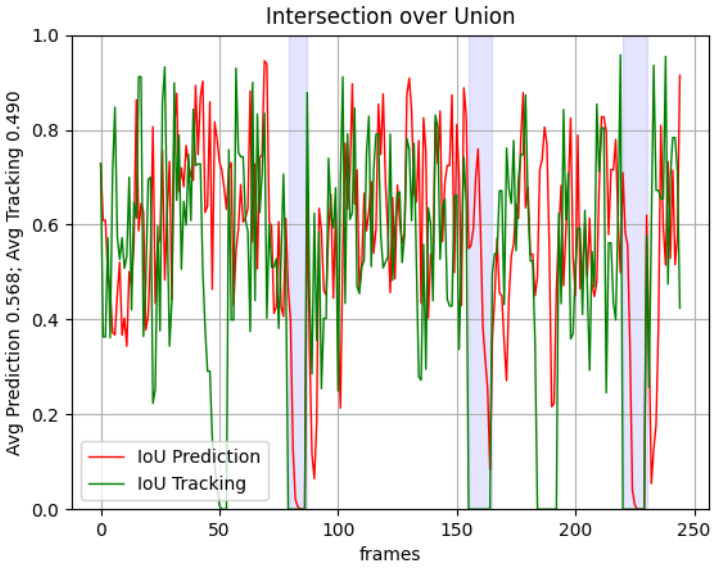

Figure 4.20: IoU between Ground Truth and Prediction / Tracker for the Inaccurate Tracker case: The time periods where the KCF is left behind, staying on features of the background can clearly be seen. However, the prediction continues tracking the object.

The phases where the KCF tracker is left behind can clearly be seen in Fig. 4.20. Nevertheless, the prediction module keeps on following the UAV satisfactorily during these times. When the tracker has jumped to different portions of the screen, the prediction is still able to keep the UAV tracked a few frames longer, while losing its accuracy quickly.

4.2.6 KF Occlusion during Flight

This case describes the trajectory prediction using a Kalman filter for a curved motion while being partially occluded during a period. This means that the *Tracker* has stopped following the object and the *Prediction* module tries interpolating the future flight path, based on prior data.

Figure 4.21: IoU for the occlusion during Flight case: During the areas shaded in blue, the UAV is being occluded by an object. The prediction continues following it, while its performance deteriorates continuously.

Fig. 4.21 shows a comparison between the IoU between the ground-truth and the *Prediction* module. The area shaded in blue indicates a time period, during which the object is being occluded and the *Tracker* stops functioning properly. As can be seen in the graph, the *Prediction* follows the object even while being occluded while losing its accuracy quickly.

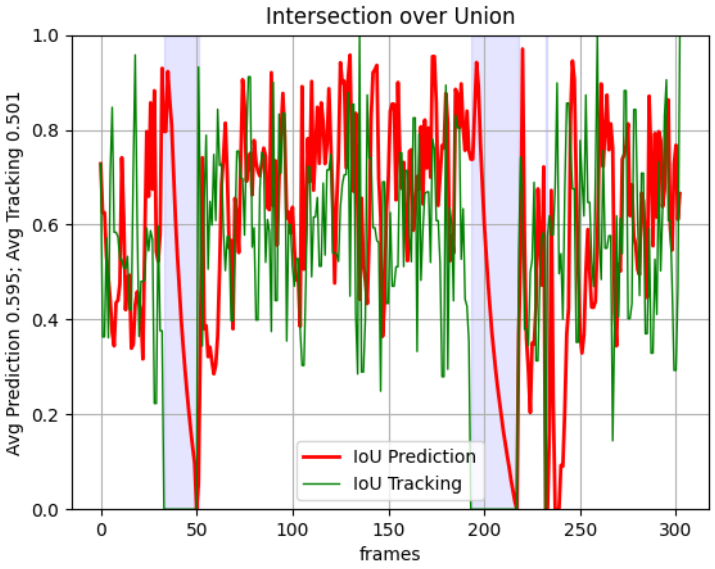

Figure 4.22: IoU for the occlusion during flight case: The prediction module follows the noisy tracker and continues following the trajectory, while the object is being occluded. Depending on the current movement of the noisy object the prediction is either able to correctly follow the UAVs path (first occlusion), or move into the wrong direction (second occlusion).

Fig. 4.22 shows the performance of the KF prediction, compared to that of a tracker with added normally distributed noise ($\sigma = 30\ pixels$). As can be seen in the figure, the prediction is able to improve on the tracker during the non-occluded timeframes, as seen in earlier experiments. During the occlusion phases the quality of the prediction depends on the current movement of the predicted position; the current positional noise has a strong effect on the continued movement of the prediction. In the first occluded phase the module filters out these influences and is able to follow the object for ten frames during the first occlusion and eight frames during the second, until losing track. During the second occlusion phase however, the noise directs the prediction into the wrong direction, resulting in a lost object. This shows the strong influence positional noise has on the system.

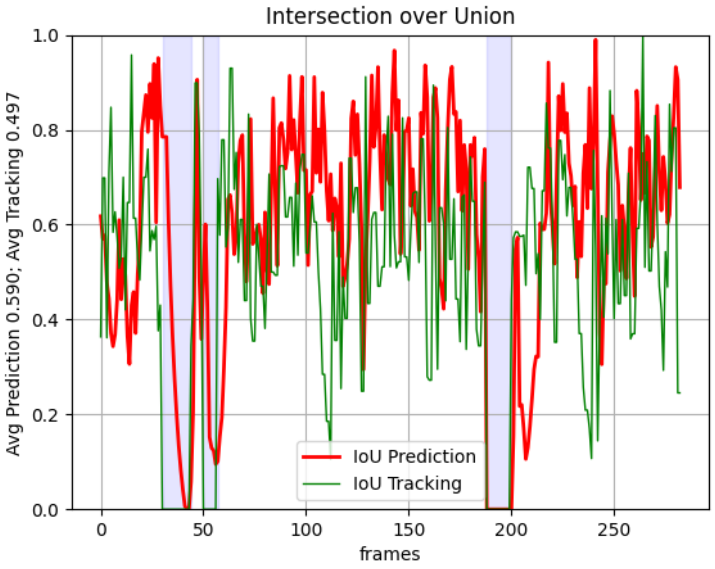

Figure 4.23: IoU for the occlusion during flight case: The prediction module follows the noisy tracker and continues following the trajectory, while the object is being occluded. Depending on the current movement of the noisy object the prediction is either able to correctly follow the UAVs path (first occlusion), or move into the wrong direction (second occlusion).

The test case using a noisy KCF tracker illustrated in Fig. 4.23 shows similar behaviour to the noised ground truth example, with

4.3 Particle Filter Results

4.3.1 PF Straight Line Constant Velocity

This case describes the trajectory prediction using a Particle filter for a linear motion with constant velocity. Fig. 4.25 shows an $IoU = 0.895$, which compared to the Kalman

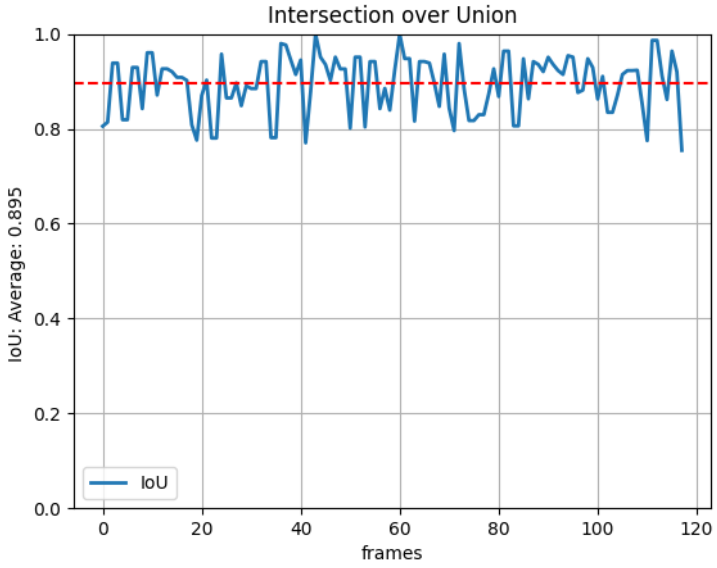

Figure 4.24: IoU for the straight line constant velocity case: While the average $IoU = 0.895$ stays consistently high, the fluctuation in the performance due to the filter's stochastic nature is bigger compared to the KF.

filter's $IoU = 0.969$ is lower for the same scenario. Comparing the two graphs Fig. 4.25 and Fig. 4.6, it becomes apparent that the PF shows a stronger fluctuation in its prediction. The stochastic nature of the filter inherently leads to these fluctuations, which can only be reduced by increasing the number of particles. Since $n \approx 800$ in this case, the performance is quite similar, however its stochastic influence cannot be removed from the result, only reduced by changing standard deviation parameters during the generation of the particles.

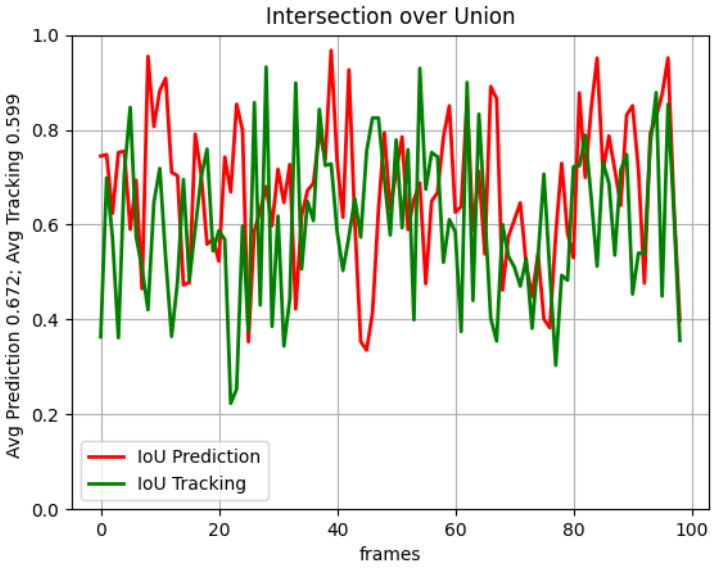

Figure 4.25: IoU for the straight line constant velocity case: The average $IoU = 0.672 > 0.599$ of the performance is high and almost consistently stays over the performance of the noisy tracker.

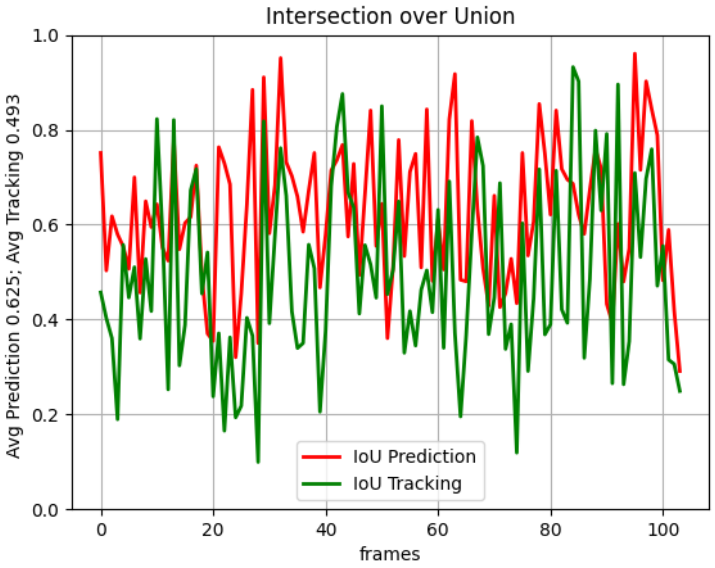

Figure 4.26: IoU for the straight line constant velocity case: Since the tracker does not stop tracking the image throughout the experiment, major differences between this case and the noisy tracker case cannot be seen, besides slight loss in performance.

Apart from a better tracking performance of the prediction compared to the tracker with an average $IoU = 0.625 > 0.493$ the experiment runs similarly to the noisy tracker. As before, since the KCF tracker does not lose track due to the background in this

experiment, this is to be expected.

4.3.2 PF Curved Line Constant Velocity

This case describes the trajectory prediction using a Particle filter for a curved motion with constant velocity, as described in Section 4.1.2.

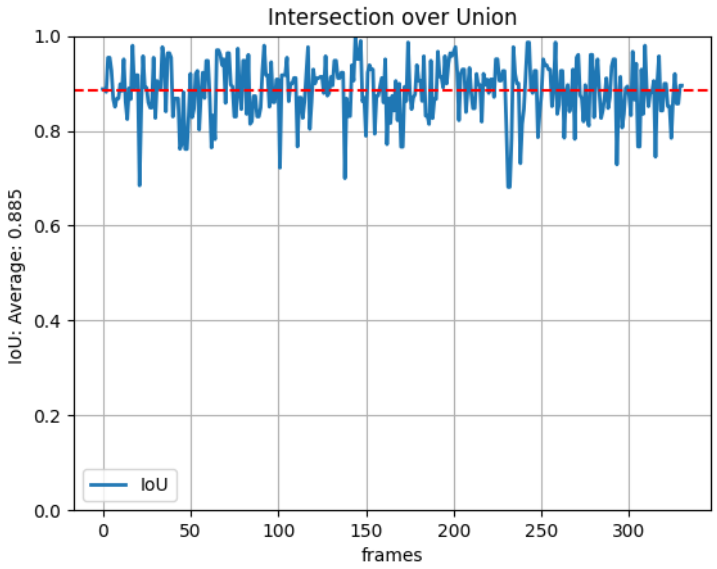

Figure 4.27: IoU for the curved line constant velocity case: An $IoU = 0.885$ in average leads to a lower performance than the KF's average $IoU = 0.976$.

Similarly to the case above, a greater variation in the prediction compared to the KF is noticeable. As a result, a lower IoU average of 0.885 is obtained, lower than in the Kalman filter case of $IoU = 0.976$.

Figure 4.28: IoU for the curved line constant velocity case: An $IoU = 0.885$ in average leads to a lower performance than the KF's average $IoU = 0.976$.

The PF is able to remove some of the normally distributed tracker noise, leading to a higher performance of the PF, compared to just the tracker.

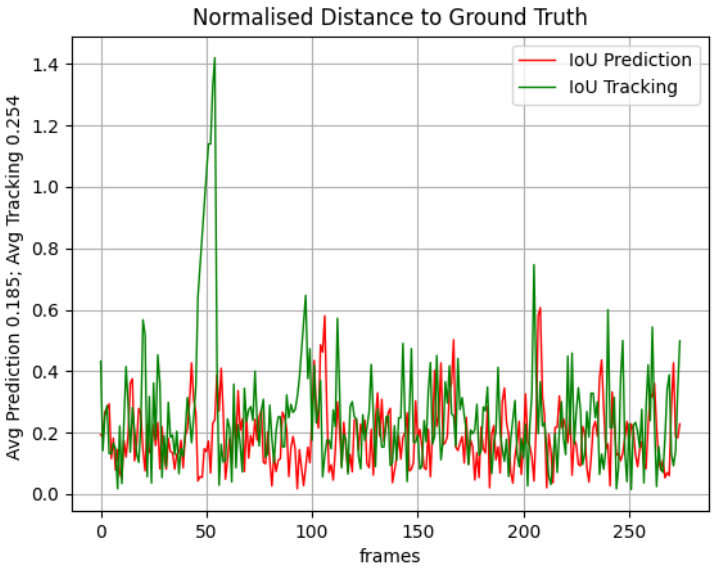

Figure 4.29: NDGT for the curved line constant velocity case: As in the previous case a higher fluctuation in distance compared to the KF is noticeable.

Due to errors in the KCF, it sometimes focuses on the background and stops following the UAV. This is especially well visible in Fig. 4.29, leading to an increasing NDGT, until the tracking is reinitialized at the position of the UAV.

4.3.3 PF Straight Line with Stopping and Acceleration

This case describes the trajectory prediction using a Particle filter for a curved motion while accelerating and slowing down.

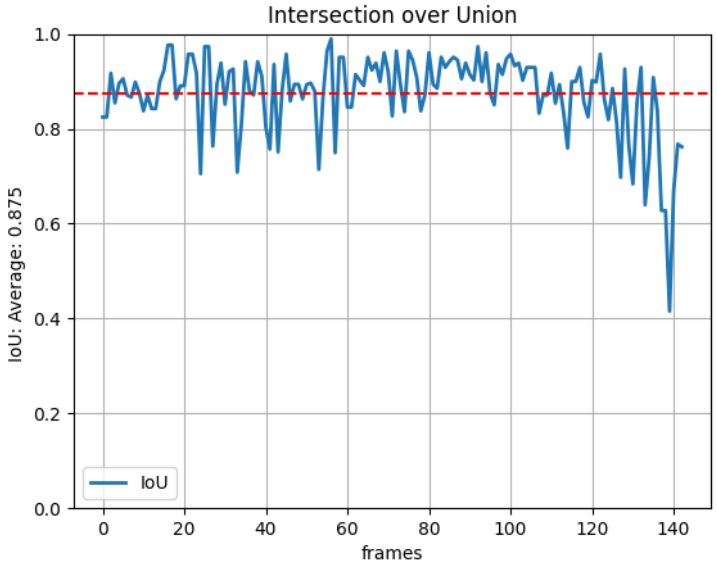

Figure 4.30: IoU for the straight line with stopping and acceleration case: With an average $IoU = 0.875$ the prediction is similar to the tracker, however stochastic noise worsens its performance.

Fig. 4.30 shows a similar relationship between the object's acceleration and its corresponding decrease in prediction performance. It is however noticeable, how the characteristic S-curve, that can be seen in the experiments with the Kalman filter, is only hinted at due to the prediction noise.

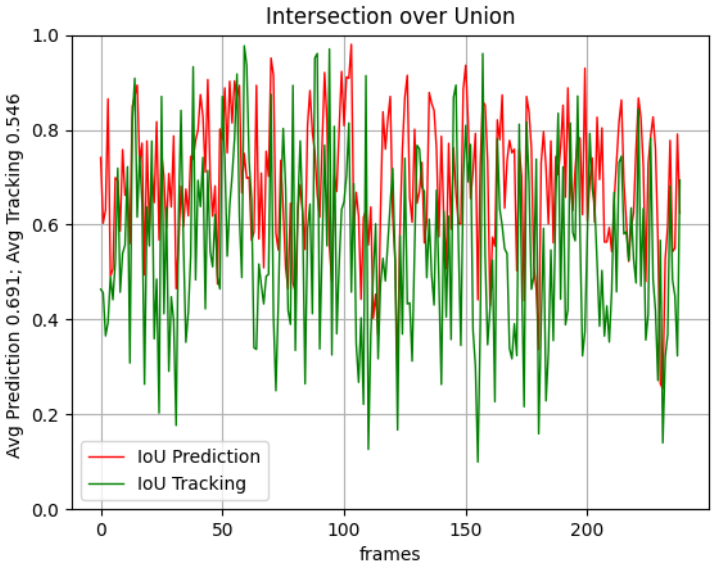

Figure 4.31: IoU for the straight line with stopping and acceleration case: The PF leads to a better performance $IoU = 0.691 > 0.546$, over the course of the experiment.

As in the previous experiments the PF leads to a better performance, compared to just using the tracker.

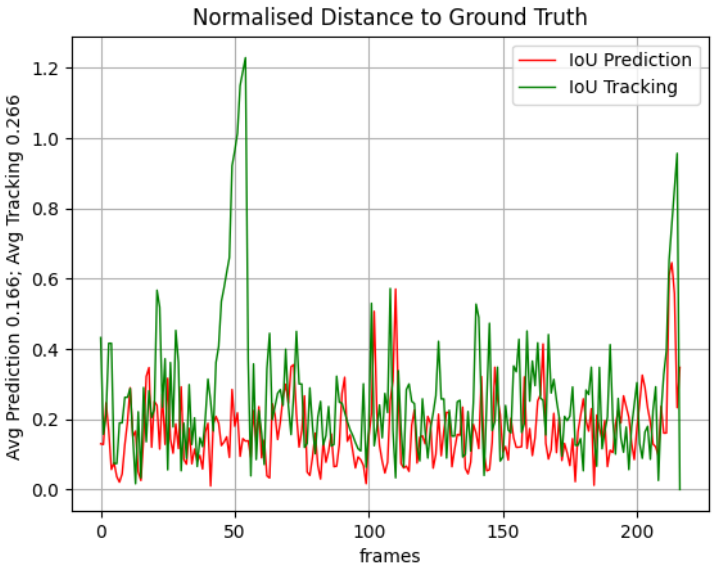

Figure 4.32: IoU for the straight line with stopping and acceleration case: The random noise makes recognizing the acceleration phases harder, showing that the filter error approaches the same size as the one usually made during accelerating.

In the NDGT it is apparent when the KCF tracker got stuck during the experiment,

leading to a lower performance of $NDGT0.266 => 0.166$ for the tracker.

4.3.4 PF Curved Line with Stopping and Acceleration

This case describes the trajectory prediction using a Particle filter for a curved motion while accelerating and slowing down.

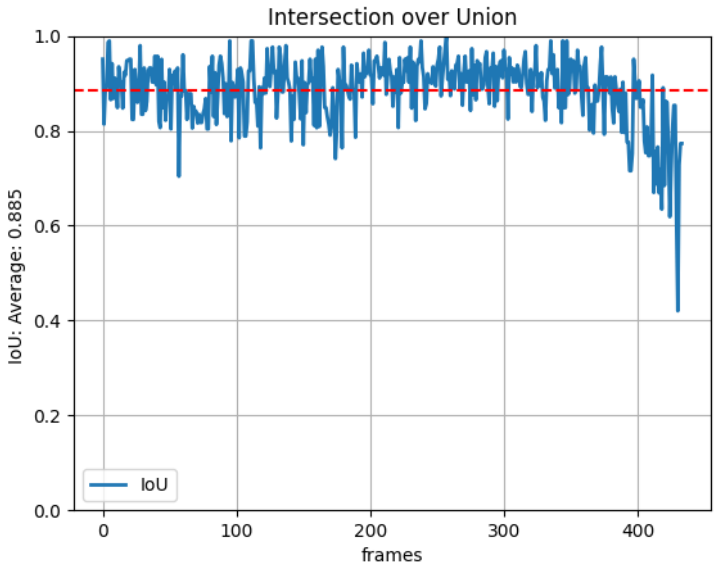

Figure 4.33: IoU for the curved line with stopping and acceleration case: Due to the complex movement of the UAV the filter the expected S-shape is hidden behind stochastic noise and not as pronounced, as in Fig. 4.15

The Particle filter offers a slightly worse performance of $IoU = 0.885$ compared to the KF with $IoU = 0.933$. Again this is due to the greater noise being amplified through the more complex movement. What may be seen in Fig. 4.35 as well is the same characteristic curve stemming from the same pattern of acceleration and slowing down, although the stochastic noise makes the pattern less pronounced as in the KF test case.

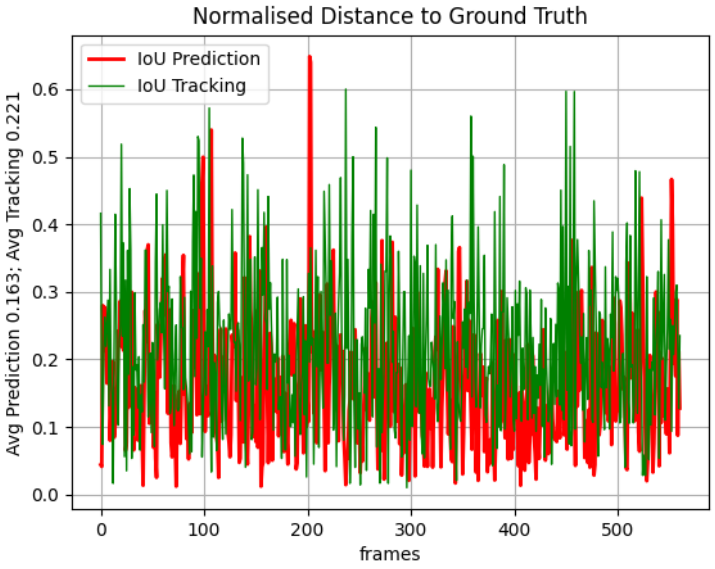

Figure 4.34: NDGT for the curved line with stopping and acceleration case: As in the IoU for this test case, the noise of the PF is dominant to the system's acceleration phases.

The tracker changing its position around a normal distribution with $\sigma = 30$ *pixels* evidently performs worse, than the PF implementation. The average value of $IoU = 0.679$ is significantly higher than the $IoU = 0.528$ of the tracker.

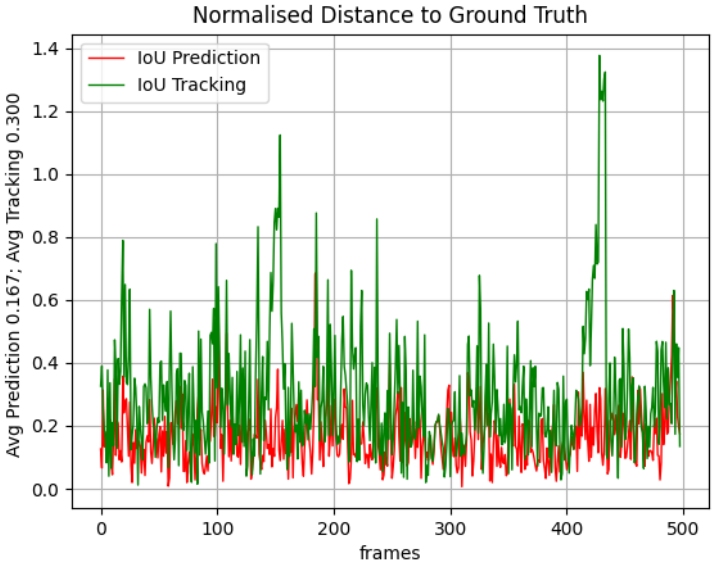

Figure 4.35: IoU for the curved line with stopping and acceleration case: The prediction outperforms the tracker with noise in the estimation of the UAVs position, as can be seen in their average IoU values.

During the experiment the KCF sometimes loses track of the UAV due to a visually

complex background. These time frames can be seen in Fig. 4.34. The high distance peaks of the tracker NDGT indicate that the tracker has latched onto the background. In these time periods however the prediction is continuing to follow the UAV until the tracking is reinitialized on the UAV.

4.3.5 PF Inaccurate Tracker

This case describes the trajectory prediction using a Particle filter for a curved motion with a faulty tracker, that jumps around the screen randomly while tracking the object.

Figure 4.36: IoU between ground truth and Prediction for inaccurate tracker case: The faulty tracker jumps to other parts of the screen during time periods shaded in blue.

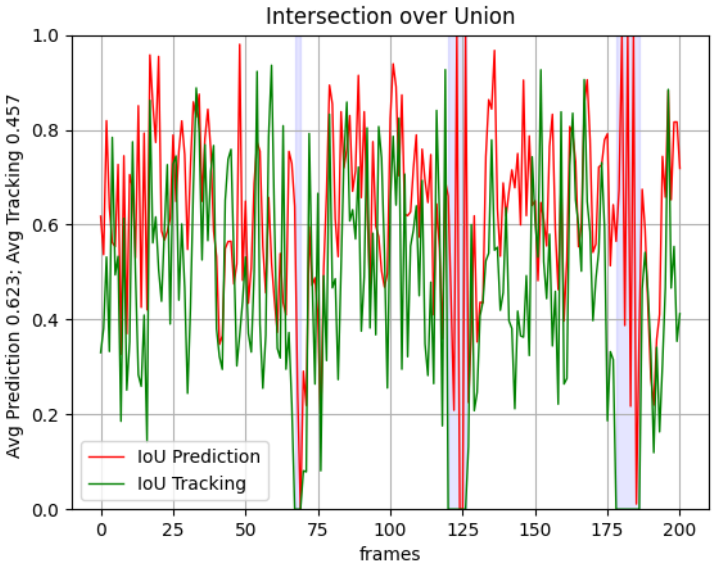

Figure 4.37: IoU between ground truth and tracker/prediction for inaccurate tracker case: During the period, where the tracker has jumped, the prediction is still able to follow the ground truth position for a few frames.

In this case the tracker jumps to different parts of the screen for shorter time frames. Fig. 4.36 shows how for short tracker errors the *Prediction* is able to follow the ground-truth trajectory of the UAV, with a high loss of accuracy during the jump. What can be noticed however, is that during the last two tracker-jumps the accuracy is rising, contrary to what would be assumed. This seems to be coincidental and specific to this simulator training video.

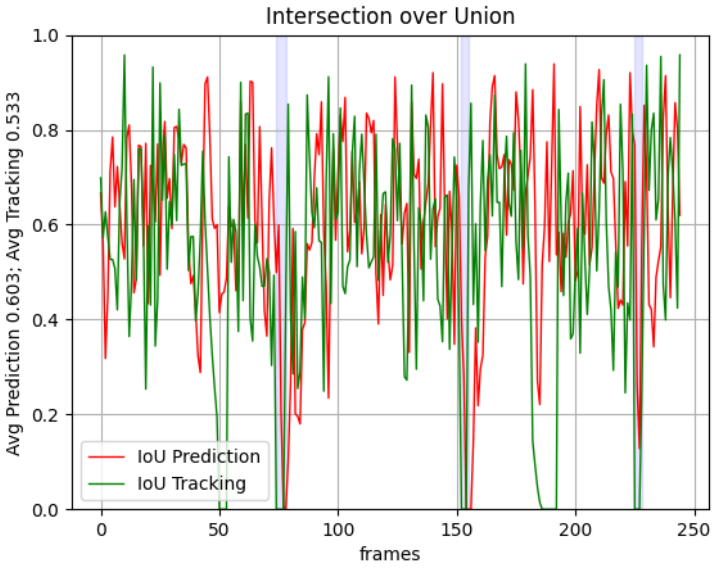

Figure 4.38: IoU between ground truth and tracker/prediction for inaccurate tracker case: During the period, where the tracker has jumped, the prediction is still able to follow the ground truth position for a few frames.

As expected the prediction fails during the jumps; the error seen as a spike in the previous diagram is visible as a short falsely tracked frame as well.

4.3.6 PF Occlusion during Flight

This case describes the trajectory prediction using a Particle filter for the object being occluded during the flight. In the test video the UAV flies behind a tree twice throughout the experiment.

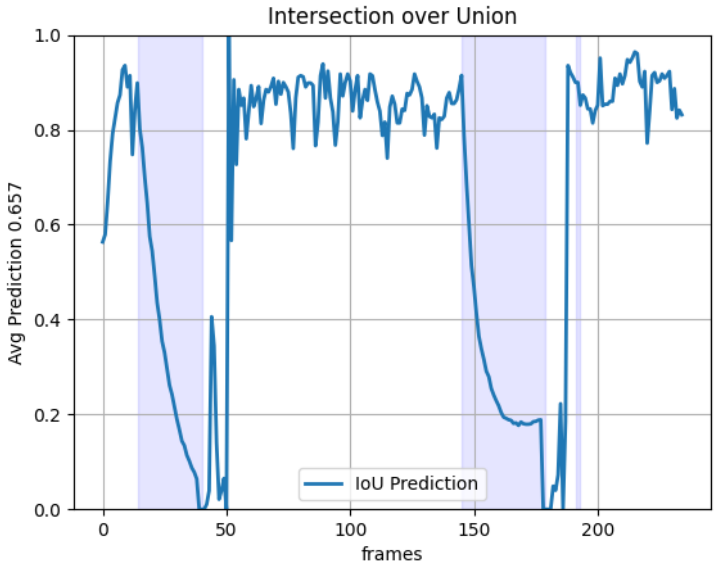

Figure 4.39: IoU for the occlusion during Flight case: The UAV is occluded twice during the experiment, indicated by the blue-shaded areas.

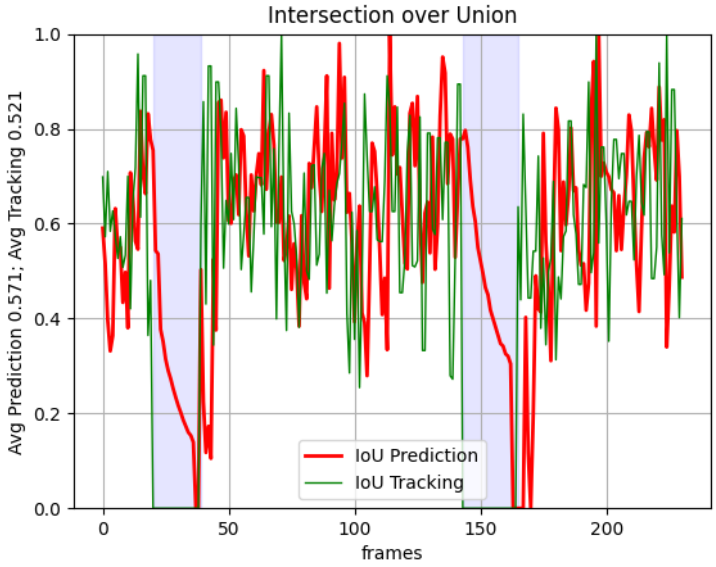

Figure 4.40: IoU for the occlusion during Flight case: The UAV is occluded twice during the experiment, indicated by the blue-shaded areas.

Figure 4.41: IoU for the occlusion during flight case: The UAV is occluded twice during the experiment, indicated by the blue-shaded areas.

Similarly to the KF case Fig. 4.41 the comparison between the *Tracker* and *Prediction* modules, compared to the ground-truth. As before the frames where the UAV is occluded are shadowed in blue. During these frames the PF follows a similar pattern of deteriorating prediction quality over a sequence of frames. While the difference in performance is small, what could be observed during testing is a much smaller stability of the Particle filter compared to the Kalman filter. Should a prediction during the occlusion appear opposite to its actual movement distance due to stochastic noise, the PF moves in the wrong direction. Since random errors accumulate during this phase, it is improbable to stay near the ground-truth value. This renders the occlusion case unstable to handle for the Particle filter.

4.4 Summary of the Result Data

The following section summarizes the results of the previous experiments. The Kalman filter and the Particle filter are each compared in their IoU and NDGT performance. Each one of the filters is first tested on the ground truth position of the UAV in the "GT+Pred" case, after which random noise, as described in the previous chapter, is documented under "Noise+Pred". Finally, in order to test the real life performance of the actual system, the prediction on the KCF filter "KCF+Pred", as well as the KCF filter with added noise "KCF+Noise+Pred" is investigated, which can be compared with the metrics of the tracker alone "KCF".

KF	Straight Const	Curved Const	Straight Acc	Curved Acc	Tracker Fault	Occ
GT+Pred	0.969	0.976	0.946	0.944	0.879	0.760
Noise+Pred	0.690	0.659	0.715	0.716	0.571	0.595
KCF	0.860	0.720	0.905	0.879	0.615	0.752
KCF+Pred	0.637	0.467	0.660	0.489	0.765	0.727
KCF+Noise+Pred	0.671	0.652	0.672	0.656	0.568	0.590

Table 4.1: IoU for the KF test cases

PF	Straight Const	Curved Const	Straight Acc	Curved Acc	Tracker Fault	Occ
GT+Pred	0.895	0.885	0.875	0.885	0.769	0.657
Noise+Pred	0.672	0.644	0.691	0.685	0.623	0.571
KCF	0.883	0.753	0.896	0.873	0.638	0.612
KCF+Pred	0.704	0.444	0.636	0.489	0.640	0.609
KCF+Noise+Pred	0.625	0.654	0.681	0.685	0.603	0.560

Table 4.2: IoU for the PF test cases

As can easily be seen the performance of the Kalman filter in the "GT+Noise" cases is superior to that of the Particle filter. This likely stems from the fact, that the PF has a stochastic nature, which leads to a higher variation in steady movement cases. In the "Noise+Pred" category the KF slightly outperforms the PF in the first four test cases, however as soon as it comes to a misfunctioning tracker, or to the object being occluded, the PF seems to be more appropriate. Comparing the test videos for those experiments it becomes apparent, that the performance is heavily dependent on the initial direction of the prediction: When the KF initially moves along the object,

the prediction is usually highly more accurate than the PF, however due to noise that is often not the case. The PF moves slower along the path and generally keeps its direction in the path of the object, resulting in higher IoU in the test cases.
As can be seen from the tracking quality of the KCF alone, it is sufficiently high compared to the tracking qualities of either filter. Only in the test cases, where the tracker malfunctions, the filters are able to improve on the trajectory estimation. From these results of the experiments it can be said, that in real conditions, using a KCF tracker, the Kalman filter outperforms the Particle filter. Thus, the tracker provides an accurate estimation of the object position, on which the filters can improve in faulty tracker, occclusion or noisy tracker use cases.

KF	**Straight Const**	**Curved Const**	**Straight Acc**	**Curved Acc**	**Tracker Fault**	**Occ**
GT+Pred	0.012	0.011	0.021	0.05	0.057	0.196
Noise+Pred	0.136	0.166	0.146	0.143	0.292	0.46
KCF	0.333	0.156	0.057	0.049	1.222	0.704
KCF+Pred	0.087	0.132	0.057	0.069	0.111	0.153
KCF+Noise+Pred	0.160	0.186	0.173	0.181	0.284	0.423

Table 4.3: NDGT for the KF test cases

PF	**Straight Const**	**Curved Const**	**Straight Acc**	**Curved Acc**	**Tracker Fault**	**Occ**
GT+Pred	0.044	0.050	0.055	0.055	0.141	0.283
Noise+Pred	0.173	0.185	0.160	0.163	0.237	0.316
KCF	0.130	0.142	0.063	0.081	0.789	1.192
KCF+Pred	0.066	0.118	0.047	0.056	0.229	0.342
KCF+Noise+Pred	0.203	0.190	0.166	0.174	0.241	0.292

Table 4.4: NDGT for the PF test cases

Similarly to the IoU, the NDGT in the "No Noise" cases are lower for the KF, indicating a smaller deviation from the UAV position for those cases. As in the previous example, in the noisy tracker experiments, the KF has a better performance, when the object stays tracked throughout, i.e. in the first four experiments. As in the IoU data, it can be seen, that the KF generally leads to lower NDGT to the ground truth position of the UAV, compared to the PF.

CHAPTER 5

Conclusion and Outlook

5.1 Summary of the Results and Analysis

This work presented an implementation and comparison of two different approaches to trajectory estimation in cases likely to be encountered during telescope UAV tracking. The first implementation centered around using a Kalman Filter, which takes into account the velocity and the acceleration of the object. The second implementation used a Particle Filter approach generating $n = 1000$ particles and updating their position and velocity. A focus is set on the cases of a faulty tracker, as well as being able to continually track a UAV temporarily hiding behind an object.
The results show, that under normal conditions, such as accelerated curved motion, the KCF tracker outperforms the estimations of the investigated filters. However, should the tracking algorithm malfunction by e.g. jumping around the screen or by providing a noisy estimation, using a Kalman filter significantly improves the tracking quality. In the case of an object occlusion, a KF is able to continue the estimation of the trajectory in order to keep the object in focus longer.
In these special cases of a faulty tracker and an occluded object, the output of the prediction module is fed back as an observation and could temporarily provide an estimation of the object's location. Comparing the Kalman Filter and the Particle Filter shows that the first approach is more suitable in this scenario. Fluctuations in the Particle filter lead to accumulating errors, with each iteration of feedback, resulting in an earlier loss of the object. The Kalman filter is the recommended prediction method for each of the tested cases.

5.2 Possible Adaptations of the System

A possible improvement on the current system could be obtained with a combination of differing filter types. As mentioned in the state-of-the-art chapter, [10], [15] and [13] offer improvements on quality and stability in the estimation of object motion. Thus,

concurrently running filters with an enabled dynamic switching based on prediction results may increase the quality of the prediction. Also, since the tracker provides information on the probability of a correct classification, changing parameters during of the observation during the operation of the prediction, for example by dynamically adapting the Kalman gain, may offer improvements.

5.3 Extension of the System using different Modalities

Using a telescope system for tracking UAVs is mostly part of a larger infrastructure containing other sensors, such as radar, radio-frequency or acoustic signals. Since the telescope system is essentially limited to detecting angular information of the object's direction, depth information cannot be considered during the prediction step. For the purposes of thesis taking into account acceleration and speed of common commercially available UAVs, this does not tremendously restrict prediction quality, however, faster UAVs in combination with low visibility may disproportionately lower the system's performance. Using data from sources providing depth information gives the prediction enough data to model the object in 3D space and might lead to big quality improvements.

Bibliography

[1] M. Mazur, R. Abadie, and A. Wiśniewski, "Clarity from above PwC global report on the commercial applications of drone technology," Tech. Rep., 2016, (accessed on 2022-04-11). [Online]. Available: https://www.pwc.com/gr/en/publications/assets/pwc-global-report-on-the-commercial-applications-of-drone-technology.pdf

[2] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," *Internet of Things*, vol. 11, 2020.

[3] S. Shackle, "The mystery of the Gatwick drone," December 2020, (accessed on 2022-04-29). [Online]. Available: https://www.theguardian.com/uk-news/2020/dec/01/the-mystery-of-the-gatwick-drone#:~:text=The%20drone%20incident%20was%20over,John%20Strickland%2C%20an%20aviation%20consultant.

[4] B. Taha and A. Shoufan, "Machine Learning-Based Drone Detection and Classification: State-of-the-Art in Research," *IEEE Access*, vol. 7, 2019.

[5] D. Ojdanic, A. Sinn, C. Schwaer, and G. Schitter, "UAV Detection and Tracking with a Robotic Telescope System," *ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2021.

[6] G. Welch, G. Bishop *et al.*, "An introduction to the Kalman filter," 1995, UNC-Chapel Hill, NC, USA.

[7] D. Jurić. (2015) Kalman filter explained graphically. (accessed on 2022-04-06). [Online]. Available: https://www.codeproject.com/Articles/865935/Object-Tracking-Kalman-Filter-with-Ease

[8] R. Muñoz-Salinas, R. Medina-Carnicer, F. Madrid-Cuevas, and A. Carmona-Poyato, "Particle filtering with multiple and heterogeneous cameras," *Pattern Recognition*, vol. 43, no. 7, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320310000488

[9] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking," *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 425–437, 2002.

[10] D. M. Raimondo, S. Gasparella, D. Sturzenegger, J. Lygeros, and M. Morari, "A tracking algorithm for PTZ cameras," *IFAC Proceedings Volumes*, vol. 43, no. 19, pp. 61–66, 2010.

[11] Y. Boers and J. Driessen, "Particle filter based detection for tracking," in *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, vol. 6, 2001.

[12] K. Hsiao, H. de Plinval-Salgues, and J. Miller, "Particle filters and their applications," 2005. [Online]. Available: https://web.mit.edu/16.412j/www/html/Advanced%20lectures/Slides/Hsaio_plinval_miller_ParticleFiltersPrint.pdf (accessed on 2022-04-06).

[13] K.-h. Hsia, S.-F. Lien, and J.-P. Su, "Moving target tracking based on CamShift approach and Kalman filter," *Int J Appl Math Inf Sci*, vol. 7, no. 1, pp. 193–200, 2013.

[14] S.-K. Weng, C.-M. Kuo, and S.-K. Tu, "Video object tracking using adaptive Kalman filter," *Journal of Visual Communication and Image Representation*, vol. 17, no. 6, 2006.

[15] X. Li, T. Zhang, X. Shen, and J. Sun, "Object tracking using an adaptive Kalman filter combined with mean shift," *Optical Engineering*, vol. 49, no. 2, p. 020503, 2010.

[16] Y. Xu, K. Xu, J. Wan, Z. Xiong, and Y. Li, "Research on Particle Filter Tracking Method Based on Kalman Filter," in *2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2018, pp. 1564–1568.

[17] L. Mihaylova, P. Brasnett, N. Canagarajah, and D. Bull, "Object tracking by particle filtering techniques in video sequences," *Advances and challenges in multisensor data and information processing*, vol. 8, 2007.

[18] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union," June 2019.

[19] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang, "Acquisition of localization confidence for accurate object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[20] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.