UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science

Jorgen Juurik

Vehicle tracking and speed estimation in aerial footage

Bachelor's thesis (9 ECTS)

Supervisor: Amnir Hadachi, PhD

Tartu 2020

Vehicle tracking and speed estimation in aerial footage

Abstract:

The field of object detection and object tracking has seen great improvements over the last few years with the innovation of modern machine learning algorithms and neural network models. Object tracking models can be utilized in many subjects, such as autonomous driving and surveillance. The goal of this thesis is to explore modern object detection and object tracking methods to construct a model which is able to track vehicles in top-down aerial footage. The YOLO method is used for creating the object detection model while a simple object tracking approach with Kalman Filtering is implemented.

Keywords:

Neural Networks, Object detection, YOLO, Object tracking, Kalman Filtering

CERCS: P170, Computer science, numerical analysis, systems, control

Sõidukite jälgimine ja kiiruse hindamine aerokaadritega videodes

Lühikokkuvõte:

Pildituvastuse ja objektijälgimise valdkond on viimaste aastate jooksul näinud suuri arengusamme, kasutades efektiivseid masinõppe algoritme ja tehisnärvivõrkudel põhinevaid süsteeme. Taolised süsteemid leiavad rakendust isesõitvate autode arendamises ja järelvalve alal. Selle lõputöö eesmärk on uurida erinevaid pildituvastuse ja objektijälgimise meetodeid, et luua mudel, mis suudab tuvastada autosid ülalt-alla vaates videokaadritelt. Mudeli loomiseks kasutatakse YOLO meetodit ning objektijälgimine tagatakse süsteemiga, mis kasutab Kalmani filtrit.

Võtmesõnad:

Tehisnärvivõrgud, Pildituvastus, YOLO, Objektijälgimine, Kalmani filter

CERCS: P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Aknowledgements

I would like to thank the supervisor of this thesis, Amnir Hadachi, for introducing me to new machine learning concepts in the field of computer vison, both thematic to this thesis and vehicle detection methods that were initially investigated. These methods include the YOLO method for object detection and approaches for object tracking.

Contents

1 Introduction

The world of computer vision has seen many improvements throughout the last years, much of which has to be attributed to the algorithms and methods that have been researched in the subject of neural networks, as well as advancements in computer hardware. Object detection is one of the key concepts of computer vision. Methods that are capable of recognizing and locating objects in images are used in fields such as autonomous driving, security, surveillance and much more. These methods are often utilized for object detection in videos, including efficient models which are capable of running in real-time, detecting objects on the go.

Often, in the case of video footage, the context of the objects in subsequent frames is important. If multiple objects are detected in different frames, knowing the identity of each object and being able to keep track of them can be useful. For example, this could be used for counting the objects, finding the path of each object and telling different objects apart. The approximation of an objects state based on previous data is called object tracking.

The goal of this thesis is to study modern solutions for object detection and tracking to create a model which is capable of detecting and tracking cars in aerial video footage. This includes investigating the YOLO detection method and using Kalman Filtering for implementing a simple object tracking system.

Although large-scale models have been able to give good results on the YOLO detection system [5], using the method with moderate and small amounts of data can become a limiting factor. This thesis explores if using such small-scale models is still feasiable and can result in accurate object tracking.

This thesis is divided into 5 parts, including the introduction. The second part seeks to give an overview of the background of the related topics. The third section goes more into detail exploring the methods and configurations used for the practical implementation of the aerial vehicle tracking model and gives an overview of the code. In section four the experimental results of the implementation are presented. The demonstration videos which visualize the results are linked in the appendix of the paper. The thesis is concluded by discussing problems with the given approach as well as exploring possible future perspectives.

2 Background

2.1 Machine Learning

Machine learning is the use of algorithms that are able to learn from prior data to construct models which can make predictions about new data. The models work by mapping features of input data into predicted labels [1]. Machine learning can be categorized by different aspects such as supervised and unsupervised learning, regression and classification [1] problems and more.

Supervised learning uses data and known labels to train models which are able to then predict the desired labels for new data. Unsupervised learning uses data without labels with the goal of the model being able to learn properties without finding the exact answer in the form of a specific label [2]. Regression problems require models which are able to predict continous values (e.g. the price of an object based on some features). A classification model on the other hand categorizes data and solves yes-or-no type problems (e.g. if an image contains an object or not).

2.2 Neural Networks

Neural Networks are a subtype of machine learning models. The networks consists of nodes which are structured into different layers. Each node takes a point of data and uses a trained weight to calculate the output. The output of one node can be used as an input for another node. Using the nodes with the trainable weights allows the model to represent complex multidimensional problems which can not be solved by simple linear functions. The layers of nodes that make up the structure of the model are called hidden layers.

However, without training the model, the mere structure is useless. Human knowledge can not be used to directly determine the values of the weights[6]. Just as any other machine learning model, the neural networks have to be trained. The goal of the training process is to find „suitable“ values for the weights of the nodes, such that the final model will give accurate predictions for new data. The weights are initialized as random values and are incrementally modified during the training process. A loss function is used to calculate the accuracy of every prediction [6]. A high loss function value describes an inaccurate prediction while a low loss function value means that the prediction is accurate. As such, the training proccess can be formulated as the optimization of the model by minimizing the loss function. The exact loss

function used to evaluate the model depends on the problem at hand. After a successful training process the model is capable of making predictions for new data.

2.3 Deep Learning

Deep learning is the process of neural network training where the goal of the model is to achieve general accuracy and a deeper understanding of the data by using more hidden layers. Having more parameters (trainable weights) allows the model to express complex nonlinearities and learn more features and deeper patterns based on the training data. Deep learning models have shown higher accuracy over shallow models and other more classical methods in many machine learning tasks, for example image classification [6].

2.4 Convulutional Neural Networks

Convulutional Neural Networks (CNNs) are a deep learning approach that use certain recurring operations on the input data to map a hierarchical structure of the given data. The structure can then be used to make conclusions about the data, such as categorizing the input data. Because of the nature of these operations, CNNs are used to create models that can learn from complex spatial training data. For example, CNNs are most commonly used for visual problems such as image recognition, however they have also been utilized to train models for recognizing speech and analyzing text. The rest of this thesis analyzes CNNs in the context of visual imagery.

2.6 Object detection

While object recognition answers the question whether an image contains an object from a certain class, it is easy to understand that recognition by itself has limited practical uses. For more complex problems it may be necessary to recognize multiple classes in one image as well as detect multiple instances of the same class.

Object detection is the process of finding and locating all instances of objects in an image [3]. A trivial approach for implementing object detection is dividing the input image into multiple smaller images (subimages), classifying each subimage seperately and concluding the instances and locations of the objects in the image; this method is also known as the sliding window method [4]. However, the approach has restrictions. Firstly, to decrease the chance of any objects getting „skipped“, the image would have to be divided densely into a large amount of subimages. Also, different subimages could include (or partly include) another

objects. And lastly, as in a practical application the sizes of the objects may vary, the image would have to be divided into subimages of different sizes. As such, this method of object detection is slow and inefficient [4].

Specifically designed object detection models look to solve these problems by pipelining the different stages of object detection into a coherent system which takes into account the flaws of the previously described approach. A traditional object detection process could be described in the following steps: region proposal (and selection) and image classification [4]. Image classification could further be seperated into feature extraction and classification based on the extracted features.The job of region proposals is to find potential areas of interest in the image – locations, where an object is likely to be present. Image classification is then applied to the regions to recognize any object classes.

2.7 YOLO Method

To allow the use of object detection in realtime, an efficient implementation of the necessary processes has to be achieved. YOLO, abbreviated from You Only Look Once, is an object detection method which strives for this by optimizing the entire process as an end-to-end solution[5]. Instead of finding regions separately and repurposing a classification model to perform the classification, YOLO uses a single neural network to predict the bounding boxes for the objects as well as the class probabilities for each bounding box [5]. To allow for the whole pipeline to be trained jointly, a custom loss function is used [5]. The loss of the model is consequently affected by the predicted bounding boxes, by the accuracy of these boxes and the class predictions for each bounding box.

The process of finding the bounding boxes and class probabilities for an image can be visualized as the image being divided into a grid. For each grid cell, multiple bounding boxes are predicted, with the center of each bounding box residing in that particular cell (the size of the bounding box is not predefined or limited). The confidence for each bounding box is also predicted, representing the likelihood of any object residing in the bounding box. Additionally the class probabilities for each cell are predicted. The existence and locations of the objects in the image can be concluded from the output of the trained model. If a cell contains a center of a bounding box with a high likelihood of an object then a detection for the most likely object class in that cell is made.

In the original YOLO paper [3], the implementation of the method is defined as a model which outputs a S × S × (B * 5 + C) prediction. **S × S** is the size of the grid. **B** is the number of bounding boxes for one grid cell while each bounding box includes the center point of the box in the grid (x, y), the width and height of the box (w, h) and the probability that the box contains an object, 5 values in total. **C** is the number of class probabilities for one cell. Because of this, the maximum number of detected objects is limited to S × S × B. Due to the limited amount of bounding box predictions per cell, densely positioned objects might not get detected [5]. The same goes for occluded objects. In regard to aerial image detection for vehicles, the former limitation may affect the results of the detection as aerial images tend to contain a higher number and concentration of objects. Occlusion however does not affect this case of detection as occlusion in aerial images is minimal (altough there are exceptions, like birds flying over objects etc).

Because of the limited number of predicted bounding boxes and the fact that YOLO requires only one evaluation per image to find both the regions of the objects as well as the classifications, it means that the YOLO models can be fast, being capable of working in real-time [5].

2.8 Updates to the YOLO method

In the YOLOv2 paper [6], the authors explain how the newer YOLO version seeks to improve the relatively poor recall and localization of the original version. The main change to the architecture of the model comes in the form of the way the bounding boxes are predicted by the model. Instead of two bounding boxes per grid cell, the YOLOv2 predicts five bounding boxes by default. Furthermore, the location data of the bounding box predictions is not encoded as absolute values but rather by an offset from predefined bounding boxes. These predefined boxes are known as anchor boxes. In other object detection methods, anchor boxes are often hand-picked based on presumptions about the most likely scale and ratio of the objects found in a particular database. In the YOLOv2 implementation however, the anchor boxes are rather based on „dimensional clusters“ – clusters of most likely scales and ratios for object bounding boxes. The clusters are found using k-means clustering on the VOC and COCO datasets. The output of the model then describes the predicted bounding boxes as an offset from the predefined boxes. This is done to increase the stability of the model in the training process and to make the learning process faster.

The class predictions in the YOLOv2 are defined for each bounding box rather than, like in the original version, only for each cell. The size of the output tensor is $S \times S \times [B * (5 + C)]$, where $S \times S$ is the size of the grid, B is the number of bounding boxes predicted per cell and C is the number of class probabilites predicted for each box [6]. This allows multiple detections per cell, both for objects in the same class as well as in different classes. In essence this means densely positioned objects can be better detected. It is also more likely that the model will detect occluded objects. In 2018 the further improved YOLOv3 [7] was released, with no significant changes to the core idea of the method, but rather with smaller optimizations and improvements.

2.9 Object Tracking

The goal of object detection is to find all objects of particular classes and determine the exact location in the image. Object detection works on a single image at a time and as such does not require any data abou the previous or future states of the objects. However, additional methods can be applied to capture the context of consequtive frames in a video. The goal of analyzing multiple frames at a time could be to maintain the identity of detected objects throughout the frames. Using previous data to make estimations about the objects in the future is called object tracking.

Object tracking could refer to different problems. In the context of this thesis we will concentrate on multi-object tracking (MOT). This means that throughout the frames, the tracked objects could leave the scene and new objects could enter the scene. Another constraint based on the goal of the thesis is that the object tracking should work in real-time. The whole video can not be analyzed as a post-process but the tracks have to be updated frame by frame in parallel to the detections.

2.10 Kalman Filtering

Kalman filtering is a state estimation method which was invented by Rudolf E. Kalman in 1960 [9]. The filtering technique is used to extract useful information from noisy data [9]. Due to the low memory and computational requirements of the filtering method, different variations are widely used in many areas of application such as aviation [10], robotics [11], autonomous vehicles [12] and other systems which require accurate sensory measurements.

Intuitively Kalman Filtering can be thought of as combining multiple sources of estimations in order to reach a more accurate approximation, while allowing for estimations about future data[9]. Mathematically, the sources of prediction are assumed to be Gaussian distributions with variances and covariances. Kalman filtering uses these distributions and combines them through a process of operations to not only extract more accurate approximations for current data but also update the filtering system for more accurate future approximations. The theory and implementation of Kalman Filtering will be more thoroughly explained in the next section in parallel with the code implementation of the method.

3 Methodology

The goal of the thesis is to train a detection model which is able to track vehicles in aerial footage (videos) and approximate the speed of the vehicles. The detection model is implemented as a custom YOLOv3 model. After a somewhat reliable detection model is trained, tracking is added and Kalman Filtering is used for more accurate and persistent tracks. Kalman Filtering is additionally used for smoother bounding boxes and for extracting the approximate speed of vehicles. This section will go through the configuration of the model as well as explain the code behind the implementation.

3.1 Data and YOLO configuration

To train the custom YOLOv3 model for detecting vehicles in aerial imagery, a dataset of approximately 450 images is used, 100 of which are used for testing the custom model.The custom dataset includes images and bounding box annotations from the VisDrone dataset [8], VEDAI dataset [14] and aerial cars dataset by Yauhen Kharuzhy [15] which in turn includes imagery from the KIT AIS Data Set and frames from aerial footage captured by drones. The VisDrone dataset includes images from many camera angles (including top-down aerial images). Because of this, the large dataset is used only partially and balanced with the other datasets which contain strictly top-down aerial images. To be able to use images from the VisDrone and VEDAI datasets the annotations have to be converted into YOLO format. For the aerial cars dataset, the annotations have already been converted by the author of the dataset.

The model is trained on a single vehicle class. The default yolov3.cfg config file is used, customized to work with the specific dataset. The training data is inputted as 512×512 pixel images and 0.001 learning rate is used. Batches of 64 images subdivided into 16 mini-batches are used to feed the images into the model during training.

3.2 Trivial tracking

As a trivial implementation for vehicle tracking simple Euclidean distance pairing can be used. On every frame of the input video, all of the detections are compared to the assigned tracks of the previous frame and the closest detection to the track is considered to belong to that track. This however introduces problems in the case of less than perfect detections. If a vehicle is not detected for a frame then the track of the vehicle is temporarily lost (no detection can be assigned to the existing track). If this happens for only a few frames then it

might not create problems, as the track will be able to “catch up” to the vehicle in subsequent frames. However, if the vehicle is not detected for a longer period of time, then implementing an approach that is able to pair the lost track and the new detection can be difficult. Even less desirable is when a detection of one vehicle is assigned to a track of another vehicle.

3.3 Kalman Filtering

To improve tracking and reduce the effect of previously mentioned problems Kalman Filtering is added. The following section goes through the code implementation of Kalman Filtering in the context of aerial vehicle tracking while the theory of Kalman Filters is referenced throughout the section. The formulas explained in this section are part of the Kalman Filtering theory [9]. Much of the implementation was possible due to the intuitive explanation of the Kalman Filtering formulas given by Tim Babb in the online article “How a Kalman filter works, in pictures” [13].

First we will take a look at the KalmanFilter.py class that is responsible for the implementation of Kalman Filtering (Appendix 2). This object is utilized in the object_tracking.py class. For every frame of the input video, the detections of the frame are extracted using the YOLOv3 model. For the first frame of the video, as no Kalman Filters exist for any vehicle, new KalmanFilter objects are initialized. The KalmanFilter object of each vehicle will be responsible for containing the updated and optimized approximation of the vehicle state. The KalmanFilter can be thought of as the track of the vehicle. When the vehicle exits the scene, the respective KalmanFilter object is removed. When a new vehicle appears, a new KalmanFilter object is created.

When a KalmanFilter object for a vehicle is initialized, the center coordinates **(x, y)** of the corresponding bounding box are set as variables of the state. Furthermore, the velocity of the vehicle for both axes is initialized **(x_vel, y_vel)**. However, they are initialized as (0, 0) due to there being no direct measurements for the velocity. The **P** variable, which describes the covariance of the state (the correlation between the different parameters of the state), is given a value of a 4 x 4 identity matrix or a matrix with 1’s along the diagonal and 0’s elsewhere. The covariance matrix is necessary to estimate the new state as a transition from one multivariate Gaussian distribution to another one. The initial value of **P** will most likely be inaccurate, but it will be updated throughout the filtering process. The **expired_age** of the KalmanFilter object is initialized as 0. The age will be used to detect and remove expired KalmanFilter objects (more on that later). The object is also given a distinct id.

```
def __init__(self, _id: int, x: float, y: float, x_vel: float, y_vel: float):
    self.id = _id
    self.x = np.array([
      [x],
      [y],
      [x_vel],
      [y_vel]
    ])
    self.P = np.eye(4)
    self.expired_age = 0
```

Figure 1. *The initialization of the KalmanFilter object. To follow common notation, the state vector is named* **x** *(not to be confused with the x coordinate of the object in the parameters).*

On the very first frame of the input video, the KalmanFilter objects for all detected vehicles are initialized. It is evident that by just initializing the KalmanFilter object nothing is gained. The model will benefit from the method in subsequent frames.

In the next frames, the existing KalmanFilter objects are used to predict the approximate states of the given vehicles. For this the **predict()** function is used for each KalmanFilter object. The function does not take any parameters and makes the prediction based on the previous state **x** and covariance **P**. The goal of the predict function is to make the best estimation for the new state of the vehicle (**x_new**) and for the covariance matrix (**P_new**) using prior data.

```
x_new = F.dot(self.x) + B * a
P_new = F.dot(self.P).dot(F.T) + Q * var_a

self.x = x_new
self.P = P_new
```

Figure 2. *The predict() function is used to make estimations for the new state and covariance.*

The operations necessary for the estimations (Figure 2) can be understood by expanding the formulas and introducing new constant matrices **F**, **B** and **Q**. The **F** matrix is called the state transition matrix. The **B** and **Q** matrices are used as tuning parameters. When the formula is expanded (Formula 1) it can be seen that the state is updated based on the kinematic equations for change in position and change in velocity (this is based on the specific problem at hand and can be changed for other tasks). The changes are added to the initial values to get the new values. The **dt** in the formula represents the change in time, for this application it is set as a constant 1. The constant acceleration **a** has a value of 0.005. These values were found to be

efficient in this particular application of Kalman Filtering. For other implementations it may be necessary to modify the **dt** dynamically based on the time elapsed and to set a different value for acceleration.

$$\overbrace{\begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}^{\mathbf{F}} \otimes \overbrace{\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}}^{\mathbf{x}} + \overbrace{\begin{bmatrix} \frac{dt^2}{2} \\ \frac{dt^2}{2} \\ dt \\ dt \end{bmatrix}}^{\boldsymbol{B}} \cdot a = \overbrace{\begin{bmatrix} x + v_x \cdot dt + a\frac{dt^2}{2} \\ y + v_y \cdot dt + a\frac{dt^2}{2} \\ v_x + a \cdot dt \\ v_y + a \cdot dt \end{bmatrix}}^{\boldsymbol{x_{new}}}$$

Formula 1. *Formula for estimating the new state. The subscripted v variables represent x_vel and y_vel state values in code respectively. The resulting matrix is the new state matrix, the values of which are based on kinematic equations.*

The estimated new state **x_new** represents the mean of the new Gaussian distribution. The second formula (Figure 2) can be used to find the new covariance matrix **P**. This uses the property $Cov(Ax) = A\ Cov(x)\ A^T$ of linear combinations of random variables to find the new covariance. The constant matrix **Q** supplies the estimation with uncertainty (the estimate should not be too confident). After the new values for the state variables and the covariance matrix are found, the previous values are overwritten. This concludes the prediction step of the Kalman Filtering method.

Based on the new estimations, the closest detection for each KalmanFilter object is found. If no close detection for a KalmanFilter object is found, then the estimated new state is considered to be the new location of the vehicle. If a close detection is found, then the **update()** function (Figure 3) of the KalmanFilter object is used to combine the estimated new state and the measured new state (detection) to find the optimized prediction.

```
z = np.array([
    [x_meas],
    [y_meas]
)]
z_est = H.dot(self.x)

S = H.dot(self.P).dot(H.T) + R
K = self.P.dot(H.T).dot(np.linalg.inv(S))

x_new = self.x + K.dot(z - z_est)
I = np.eye(4)
P_new = (I - K.dot(H)).dot(self.P)

self.x = x_new
self.P = P_new
```

Figure 3. *The update(x_meas, y_meas) function is used to update the state and covariance based on the new detection. The x_meas and y_meas variables are the center coordinates of the new detection bounding box.*

The goal of the update() function is to find the reliability of the measurement and the estimation, and to use this data to make a balanced prediction which combines both data. First a variable **z** which includes the true measurements is constructed. We would like to be able to compare the true measurement and the previously estimated state, however the estimated state includes variables which we have no measurements for (the **x_vel** and **y_vel** velocities). Matrix **H** transforms the estimated state into an estimated measurement **z_est** (Formula 2). The covariance of the estimated measurement can be calculated using the already familiar property $Cov(Ax) = A\ Cov(x)\ A^T$ (Formula 3).

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}^{\mathbf{H}} \otimes \overbrace{\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}}^{\mathbf{x}} = \overbrace{\begin{bmatrix} x \\ y \end{bmatrix}}^{\mathbf{z_{est}}}$$

Formula 2. *Estimated measurement based on the estimated state. The H Matrix will be used in the next formulas to transform the estimated states and covariances into estimated measurements and error of the estimation respectively.*

$$P_{z_{est}} = H\ P\ H^T$$

Formula 3. *Covariance of the estimated measurement based on the estimated state.*

To understand how the **x_new** and **P_new** variables are derived let us take a look at the formulas used for finding the corrected measurement and updated covariance for the measurement (Formula 4*).* The first formula is for calculating the corrected measurement. The second formula is for calculating the corrected variance for the measurement. The **H** matrix can be omitted from both sides of the first formula to find the new state x_{new}. The **H** and $\mathbf{H^T}$ matrices can be omitted from both sides of the second formula giving the updated covariance P_{new} (Formula 5).

$$K = \frac{HPH^T}{HPH^T + R}$$

$$Hx_{new} = Hx + K\left(z - z_{pred}\right)$$

$$H\ P_{new}\ H^T = H\ P\ H^T - K(H\ P\ H^T)$$

Formula 4. The formulas for finding the corrected measurement and covariance for said measurement.

$$K' = \frac{PH^T}{HPH^T + R}$$

$$x_{new} = x + K'\left(z - z_{pred}\right)$$

$$P_{new} = P - K'P = (I - K')P$$

Formula 5. After simplifying the previous formula (Formula 4) the corrected state (including all variables) and covariance for the state can be found.

The Kalman Gain **K** variable in the formula represents the difference between the estimated and true measurements, where $\boldsymbol{HPH^T}$ is the covariance of the estimated measurement (Formula 3) and **R** is the set covariance for the true measurement. **R** is a constant parameter and, like the **Q** and **B** matrices introduced in the predict() function, should be modified as necessary, based on the assumed accuracy of the measurement system. If the measurement system is accurate, R should have low values. We can observe that if this is the case, then the denominator of the K formula is dominated by the covariance of the estimated measurement instead and the whole value of K approaches **1** (identity matrix with suitable size). The opposite is true for when the measurement system is thought to be inaccurate: when the error of the true measurement dominates the denominator of the formula, then the value of K approaches **0**. In other words, the larger the value of K, the more trust is put into the true measurement of the system. This property is utilized to find out how much the estimated state should be corrected. The same is true for updating the estimated covariance of the state. The larger the value of K, the lower the covariance of the state becomes.

The update() function is used when a new detection is found for the KalmanFilter object. When no new detections can be used to update the data, then the predict() function is used to rely on the new location of the vehicle (Figure 4). However, if the filter is not updated for a long time then the error of the state increases. This is why the **expired age** of the KalmanFilter object is increased every time the object is used to predict the new state of the vehicle but updating is not possible. After a certain expiration age is reached the KalmanFilter object is removed and is not used for any new predictions.

Another feature of the Kalman Filtering process is that some data can be extracted from the system although no direct measurements are available. In this case, the state variable x_vel and y_vel can be used to find the overall velocity of the vehicles. The x_vel and y_vel variables get updated based on the kinematic equations used for calculating the predicted state. Using these values the speed of the vehicle can be found. The real life approximate km/h speed of the vehicle can be calculated by multiplying the value with the assumed number of pixels in one kilometer and the number of frames in one hour (depends of the frame rate of footage) (Figure 4).

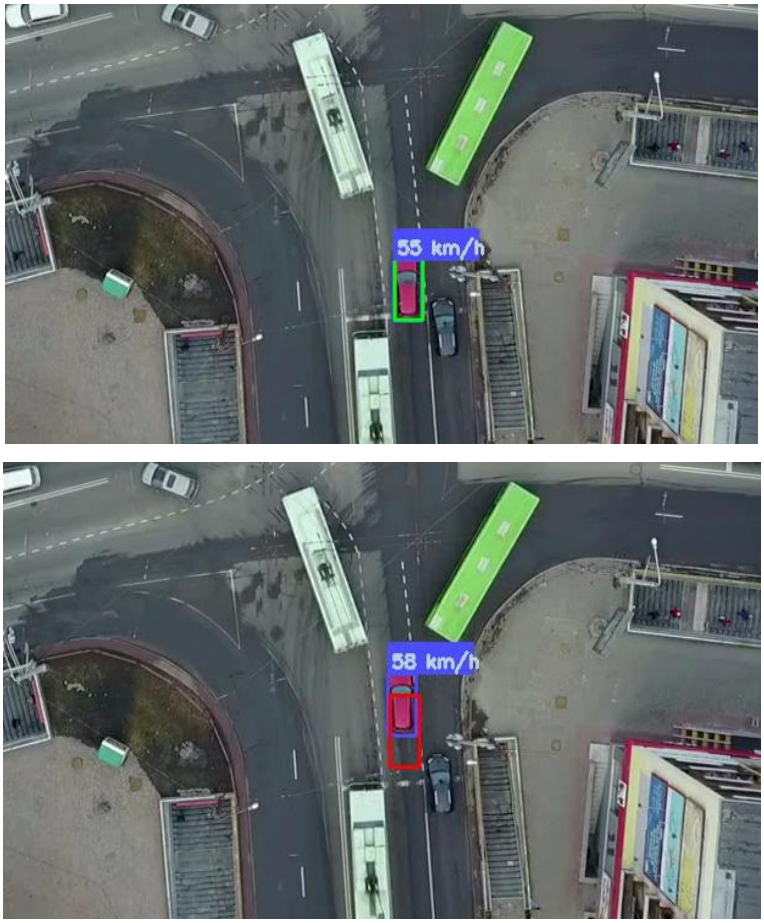

Figure 4. In the first frame the vehicle is able to be accurately detected by the YOLO model. As such the detection is used to update the Kalman Filter of the object. The velocity of the vehicle is able to be extracted from the filter. In the second frame, no YOLO detection for the vehicle is found. The Kalman Filter is used to predict the location of the vehicle based on previously learned data (purple bounding box). The red bounding box represents the last true detection that was matched to the vehicle.

It is important to understand that the state transition matrix and other hyper-parameters used in this specific implementation of Kalman Filtering may not work for all applications. The reason why Kalman Filtering can be used for many different fields is because the state transition matrix and other constant matrices used for predicting the new state can be customized based on the problem at hand. For example, other properties of physics may be introduced to better estimate the future state of the object.

4 Results

The following section explains the experimental results of the constructed aerial vehicle detection and tracking model by visualizing the advantages of using Kalman Filtering while also bringing up flaws of the model. The model is used on multiple videos of aerial footage to demonstrate the overall effectiveness of the approach (Appendix 1). Demo Video 1 and 2 demonstrate the general results of the method. The green bounding boxes in the demonstrations (both imagery and demo videos) represent the current YOLO detection for the vehicles, while the purple boxes are the updated estimations based on previous data and the current detection. If no detections are found for a vehicle, the estimations are used to predict the location of the vehicles. Demo Video 3 compares the results with and without using Kalman Filtering and Demo Video 4 shows how including the bounding box dimensions in the Kalman Filter can help the model improve the accuracy of the bounding boxes and reduce jittering and erratic changes in bounding box size.

4.1 Detection

The effectiveness of the model is limited due to the relatively small amount of data being used for training the model. However, this illustrates that with even modate amounts of data effectiveness is possible. As seen in the demonstration videos, different scenes of aerial footage give different results. Demo Video 1 shows good results, except for buses (which were not included in the training dataset) not getting reliably detected. Demo Video 2 is a good example of a scene that does not work as well with this model. It has sparse detections, many vehicles are not getting detected accurately and some false positives are being detected (Figure 5). This demonstrates that the accuracy of the trained model depends on the scene, because each scene can have different lighting, different types of cars and other variables. Each of these variables can influence the effectiveness of the model.

4.2 Evaluation of the detection model

The precision, recall, F1-score and avarage IoU metrics were evaluated on datasets of aerial images of cars (Table 1). The first dataset contains 40 frames from the same scene as Demo Video 1. The second dataset contains 184 top-down aerial images from the VisDrone Dataset. The third evaluated dataset includes images taken from more varied camera angles, also from the VisDrone dataset (Figure 6). It should be noted that the VisDrone Dataset was also used for training the model, however none of the scenes used for evaluating the model were

included in the training data. This means that the model is not overfitted for the data used for evaluating the model. As seen from the metrics table, the detection model works well on the first two evaluation datasets, however on the third dataset with more varied camera angles, the model has relatively poor results. This can be attributed to the fact that the model was trained with top-down aerial imagery and has a shallow understanding of cars from different angles. If a more general model were to be trained, more images from different angles would be necessary.

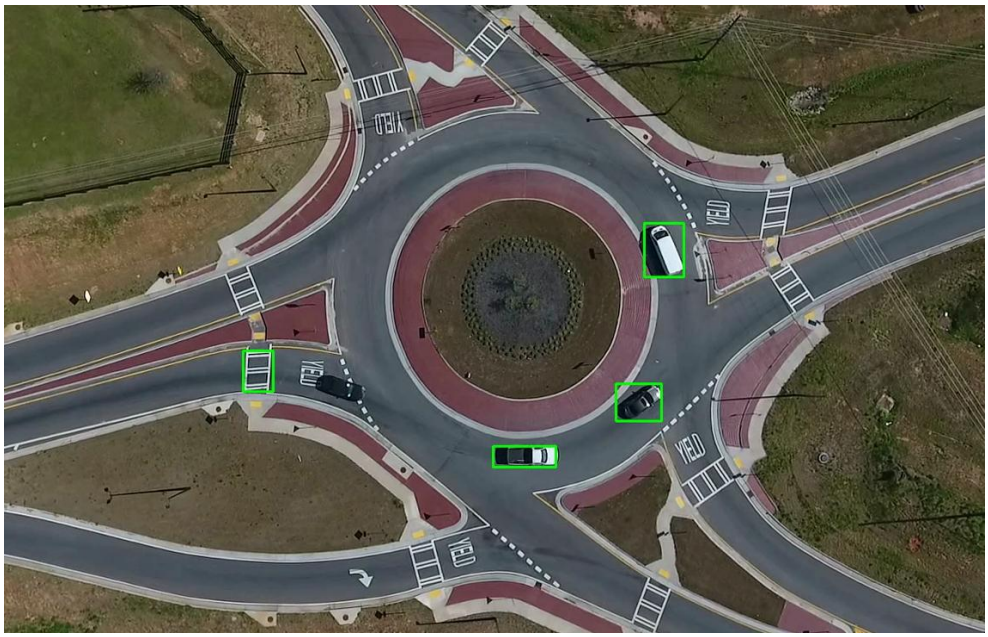

Figure 5. The green boxes represent the YOLO detections for the vehicles. The black vehicle is not detected, while the pedestrian crossing causes a false positive detection.

Figure 6. Frames from the 3 datasets used for evaluating the model. The first and second dataset include images from top-down aerial footage, while the last dataset has images with varied camera angles.

Evaluated data	Precision	Recall	F1-Score	Avarage IoU
Frames from Demo Video 1	0.8	0.89	0.84	62.64%
VisDrone top down imagery	0.76	0.91	0.83	59.77%
VisDrone varied camera angles	0.56	0.47	0.51	40.91%

Table 1. The evaluated metrics of the detection model. The precision represents the accuracy of the model while recall represents the amount of detections found in comparison to the actual amount of objects in the frame. The F1-score gives an overall estimation of the effectiveness of the model, combining the recall and precision in a single measurement. Avarage IoU or Intersection over Union represents the accuracy of the bounding box sizes, the percentage showing the accurate area of the bounding box.

4.2 Tracking

Using Kalman Filtering seems to improve the consistency of vehicle tracking. Even when no detections for a vehicle are found for multiple frames (up to 50 frames, after which the expired Kalman Filter is removed) the Kalman Filter can still make accurate predictions if the velocity of the vehicle does not change too much. However, when significant changes in speed or direction occur during this time, the Kalman Filter predictions can give inaccurate estimations for the location of the vehicle. This results floating bounding boxes which can be seen in some frames of the demonstration videos (Figure 7, Figure 8). In other cases, the estimations of the vehicle might get initialized but no new detections can be matched to the filter. Because of this, the velocity of the vehicle can not be estimated, as there is no prior data. This results in boxes with estimations of no velocity or very low velocity being „left behind“ by the vehicles they were estimated for. Demo Video 3 demonstrates how the overall results of object tracking are improved by Kalman Filtering, showing the differences of object tracking with and without Kalman Filters.

Figure 7. The red box represents the last detection matched with the vehicle. For the consequent frames, Kalman Filtering is used to estimate the location of the car. Due to the long distance without any updates for the location, the Kalman Filter estimate is “drifting off” and is no longer very accurate.

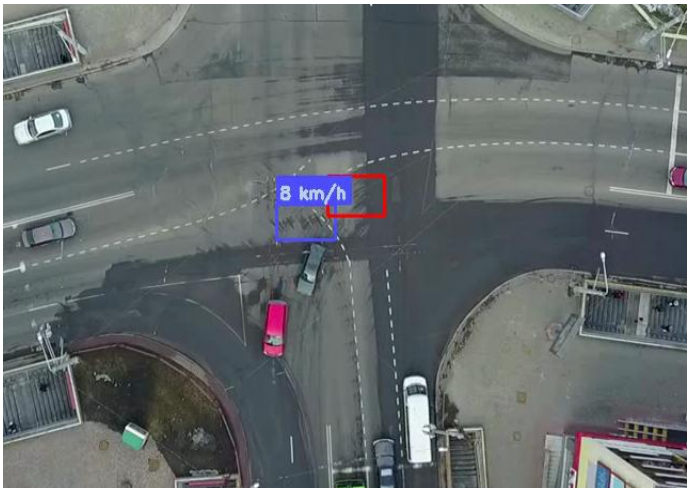

Figure 8. Another example of the Kalman Filter failing to estimate the accurate location of the (dark) vehicle when no detections are present for multiple frames, due to the vehicle significantly changing the direction of movement.

4.3 Smoothing the Bounding Boxes

Demo Video 4 explores the use of bounding box dimensions in the Kalman Filtering method to make the bounding box sizes more accurate and increase the smoothness of bounding boxes throughout the video frames. Without the bounding box smoothing the sizes of the bounding boxes tend to vary significantly from frame to frame.

5 Conclusion

5.1 Conclusion

YOLO is capable of training sufficiently accurate models for custom object detection with a relatively small training dataset, however the generality of these models is limited. In this particular case, the trained model does tend to work with some video footage but in other cases the output of the model is not accurate.

Kalman Filtering significantly improves object tracking as it makes the model more reliable in the case of inaccurate or inconsistent detections. It can be difficult for detection models which are trained on moderate amounts of data to be able to detect objects in every frame consistently, so having a system in place which is able to reference prior data for approximating the new location of the objects can be useful. Kalman Filtering makes the model customizable and scalable for other object detection problems. It is also computationally efficient and can be used for real-time object tracking problems.

5.2 Future Perspectives

The custom YOLO model trained for this thesis could be improved relatively easily. A larger dataset of higher quality images can be used to increase the accuracy of the model. Different classes of vehicles could be used to train the model, such as trucks and motorcycles, giving better results with these vehicles.

The particular approach used for object tracking can be improved by testing different parameters for the constant matrices used for predicting the new states of objects. In this thesis relatively little effort was put into finding the most effective constants and coefficients for the Kalman Filtering process. It is possible that adapting a more complex equation for finding the new states may be benefitial.

Another improvement could be the use of more complex variations of the Kalman Filter, such as Extended Kalman Filtering and Unscented Kalman Filteringing [7]. These methods are capable of using non-linear equations for the state transition, allowing for a more sophisticated model. The theory behind the methods should be further explored to get a better understanding of possible new perspectives.

Finally, since this thesis provides only experimental evidence to the accuracy and relative effectiveness of the particular object tracking approach, other methods should be studied and different methods compared.

References

[1] Simeone, O. A Very Brief Introduction to Machine Learning With Applications to Communication Systems. *IEEE Transactions on Cognitive Communications and Networking* 4, 2018, pp. 648-664. *ArXiv, abs/*1808.02342.

[2] Jung, A.C. Machine Learning: Basic Principles. *ArXiv, abs/*1805.05052.

[3] Felzenszwalb, P. F., Girshick, R. B., McAllester, D. and Ramanan, D. Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627-1645, Sept. 2010, doi: 10.1109/TPAMI.2009.167.

[4] Zhao, Z., Zheng, P., Xu, S. and Wu, X. Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, Nov. 2019, doi: 10.1109/TNNLS.2018.2876865.

[5] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[6] Redmon, J. and Farhadi, A. YOLO9000: Better, Faster, Stronger. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.

[7] Redmon, J. and Farhadi, A. YOLOv3: An Incremental Improvement. 2018. *ArXiv, abs/1804.02767*.

[8] Zhu, P., Wen, L., Bian, X., Ling, H., & Hu, Q. (2018). Vision Meets Drones: A Challenge. *ArXiv, abs/1804.07437*

[9] Pei, Y., Biswas, S., Fussell, D. and Pingali, K. An Elementary Introduction to Kalman Filtering. *Communications of the ACM 62*. doi: 10.1145/3363294.

[10] Ehrman, L. M. and Lanterman, A. D. Extended Kalman filter for estimating aircraft orientation from velocity measurements. *IET Radar, Sonar & Navigation*, vol. 2, no. 1, pp. 12-16, February 2008, doi: 10.1049/iet-rsn:20070025.

[11] Hartley, R., Ghaffari, M., Eustice, R. M., & Grizzle, J. W. Contact-aided invariant extended Kalman filtering for robot state estimation. *The International Journal of Robotics Research, 39*(4), 402-430. 2020. doi: 10.1177/0278364919894385.

[12] De Marina, H. G., Pereda, F. J., Giron-Sierra, J. M. and Espinosa, F. UAV Attitude Estimation Using Unscented Kalman Filter and TRIAD. *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4465-4474, Nov. 2012, doi: 10.1109/TIE.2011.2163913.

[13] Babb, T. How a Kalman Filter works, in pictures (Online source). 2015. https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/#mjx-eqn-gaussequiv.

[14] Razakarivony, S. and Jurie, F.. Vehicle detection in aerial imagery: A small target detection benchmark. *J Vis. Commun. Image R,* 34:187–203. 2016. doi: 10.1016/j.jvcir.2015.11.002.

[15] Kharuzhy, Y. Aerial Cars dataset (Online source). 2017. https://github.com/jekhor/aerial-cars-dataset

Appendix

1. Demo videos

1) Demo Video 1: https://youtu.be/SL-Vyp9jTok
2) Demo Video 2: https://youtu.be/MhYS-tfyhIM
3) Demo Video 3: https://youtu.be/fC9i6jNf3Vg
4) Demo Video 4: https://youtu.be/-UaFI5atUqw

2. Source code

The Python source code can be downloaded from the following GitHub repository: https://github.com/jorgen5/yolo-tracking. This repository uses the TensorFlow compatible version of YOLOv3 made by GitHub user zzh8829 for detecting the vehicles. The weights of the custom model are available in the repository, as well as one of the demo videos used for testing the model. The code Kalman Filtering is implemented in Python. Further instructions are given in the README page of the repository.

3. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Jorgen Juurik

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, **Vehicle tracking and speed estimation in aerial footage**, supervised by Amnir Hadachi.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Jorgen Juurik

08/08/2020