

https://dspace.org/

Ball tracking by object detection using deep neural network aided Kalman filtering in a real-time simulated RoboCup Soccer environment

Nagy, Marcell Douglas

2024-08
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/46674

Nagy, Marcell Douglas. (2024). Ball tracking by object detection using deep neural network aided Kalman filtering in a real-time simulated RoboCup Soccer environment. [Master's dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace. https://hdl.handle.net/10539/46674

Downloaded from a DSpace repository. https://dspace.org/

BALL TRACKING BY OBJECT DETECTION

using deep neural network aided Kalman filtering in a real-time simulated RoboCup Soccer environment

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Marcell Nagy
2628829

Supervised by Prof. Richard Klein
and Dr. Pravesh Ranchod

August 6, 2024



A research report submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

Computer vision has the ability to provide an abundance of environmental information to a robotic system, which makes perception and interaction with a dynamic world possible. In this research report the problem of real-time ball tracking in the context of simulated RoboCup soccer is considered. A tracking-by-detection framework is utilized to take advantage of the high performance of modern neural detection techniques as well as the use of neural assisted Kalman filtering for tracking which strikes a balance between the ease of interpretation of the traditional model-based approach and the ability to learn complex dynamics using neural networks. The results show that a modern keypoint based detector can outperform established real-time RoboCup ball detection techniques such as traditional Viola-Jones based detection as well as popular anchor based detection approaches, in the context of simulated RoboCup soccer. Further, a neural approach to Kalman filtering is able to outperform the popular extended Kalman filter for simulated RoboCup soccer kick tracking by implicitly learning the system dynamics.

Declaration

I, Marcell Nagy, hereby uniquely declare the contents of this research report to be by my own efforts. This research report is prepared for the degree of Master of Science (Robotics) at the University of the Witwatersrand. This material has not been submitted to any other university, or for any other degree.

Acknowledgements

Thank you Prof. Richard Klein and Dr. Pravesh Ranchod for your constructive comments, feedback and support throughout my studies. I'm inspired by your unfailing willingness to go out of your way to help students. Your enthusiasm for computer science is contagious.

Contents

Preface
Abstract i
Declaration ii
Acknowledgements iii
Table of Contents iv
List of Figures vi
List of Tables viii

1 Introduction **1**

2 Background **3**
2.1 Object Tracking 3
2.1.1 SORT 4
2.1.2 Deep SORT 6
2.1.3 KalmanNet 7
2.1.4 Performance measures 8
2.2 Object Detection 9
2.2.1 Traditional object detection 9
2.2.2 Modern object detection 11
2.2.3 Performance measures 14
2.2.4 Optimizing models for real-time performance 16
2.2.5 Neural network development and deployment 16
2.3 RoboCup Soccer 17
2.3.1 RoboCup soccer simulator 18

3 Related Work **19**
3.1 Ball Detection 19
3.2 Ball Tracking 20

4 Problem Statement, Research Aims and Objectives **21**
4.1 Problem Statement 21
4.2 Research Hypothesis 21
4.3 Research Aims And Questions 22

5 Research Methodology **23**
5.1 Phase One: Object Detector Investigation 23

5.1.1 Dataset generation . . . 23
5.1.2 Model selection . . . 24
5.1.3 Training . . . 24
5.1.4 Evaluation . . . 24
5.2 Phase Two: Object Tracking Investigation . . . 25
5.2.1 Dataset generation . . . 25
5.2.2 Training . . . 25
5.2.3 Evaluation . . . 25

6 Object Detection Investigation **26**
6.1 Detection Dataset . . . 26
6.1.1 Software environment . . . 26
6.1.2 Preprocessing . . . 27
6.1.3 Data exploration . . . 28
6.2 Baseline Detection . . . 29
6.2.1 Viola-Jones . . . 29
6.2.2 YOLO v4 Tiny . . . 33
6.3 Proposed Detection . . . 36
6.3.1 Naive latency evaluation . . . 36
6.3.2 NanoDet . . . 37
6.4 Evaluation Of Detection . . . 40

7 Object Tracking Investigation **43**
7.1 Tracking Dataset . . . 43
7.1.1 Software environment . . . 43
7.1.2 Preprocessing . . . 43
7.1.3 Data exploration . . . 44
7.2 Baseline Tracking . . . 46
7.2.1 Basic tracking . . . 46
7.2.2 Standard Kalman filtering . . . 49
7.2.3 Extended Kalman filtering . . . 52
7.3 Proposed Tracking . . . 54
7.3.1 Modified KalmanNet . . . 54
7.3.2 Model training . . . 55
7.3.3 Evaluation of results . . . 56

8 Conclusion **59**
8.1 Further Work . . . 60
8.1.1 Improving ball measurements . . . 60
8.1.2 Improving state estimation . . . 61
8.1.3 Extensions . . . 61

References **69**

List of Figures

2.1 Generalized tracker flowchart (Redrawn from Wang *et al.* [2015]) 4
2.2 KalmanNet block diagram [Revach *et al.* 2022] 7
2.3 KalmanNet RNN [Revach *et al.* 2022] 8
2.4 Haar features [Viola and Jones 2001] (Left) and strong responses (Right) 10
2.5 YOLO precision-recall curve (Picasso Dataset) [Redmon *et al.* 2015] . . . 11
2.6 YOLO object detection [Redmon *et al.* 2015] 12
2.7 Keypoint based object detection [Tian *et al.* 2019] 13
2.8 ONNX Model environments [Fehlner 2022] 17
2.9 Simulated RoboCup Soccer 18
2.10 Simulated image perceptor (Agent perspective in bottom-left subwindow with a ball in the distance) 18

6.1 RoboCup agent visual perception [SimSpark 2022] 26
6.2 Ideal wide angle lens transformation [Devernay and Faugeras 2001] . . 27
6.3 A selection of detection dataset ground-truths 28
6.4 Detection dataset ball statistics 29
6.5 Thresholded edge image 30
6.6 AP vs $numStages$ 31
6.7 Cascade classifier 4 layers of final stage 31
6.8 AP vs $minNeigbours$ 32
6.9 AP, latency vs $scaleFactor$ 32
6.10 YOLO v4 Tiny training curves 34
6.11 YOLO v4 Tiny variations compared 35
6.12 Pretrained model performance vs. size 36
6.13 NanoDet training curves 38
6.14 NanoDet performance visualised across variations 39
6.15 Evaluation of detectors 40
6.16 Trained Viola-Jones (6.16a cropped) & Trained NanoDet (6.16b cropped) 41
6.17 Pretrained NanoDet (6.17a cropped) & Trained NanoDet (6.17b cropped) 41
6.18 Trained YOLO v4 (6.18a cropped) & Trained NanoDet (6.18b cropped) . 42

7.1 Tracking dataset statistics 44
7.2 Tracking dataset ground-truths 45
7.3 Tracking dataset - Cherry picked to show error 45
7.4 Basic tracking error 46
7.5 A ball at 5m distance 47
7.6 Short ground kick – Raw detections 47

7.7 Long ground kick – Raw detections 48
7.8 Short lofted kick – Raw detections 48
7.9 Long lofted kick – Raw detections 49
7.10 Short ground kick – First order Kalman filter search 50
7.11 Long ground kick – First order Kalman filter search 50
7.12 First order Kalman filter error 51
7.13 Short ground kick – Extended Kalman Filter 52
7.14 Long ground kick – Extended Kalman Filter 53
7.15 Extended Kalman Filter error 53
7.16 Neural network aided Kalman filtering model training 55
7.17 Short ground kick – Tracker evaluation 56
7.18 Long ground kick – Tracker evaluation 56
7.19 Tracker error (Test set) 57
7.20 Tracker speeds 58

8.1 Path Aggregation Network [Liu *et al.* 2018] 60

List of Tables

6.1 Detection dataset 28

7.1 Tracking dataset 44

Chapter 1

Introduction

Object tracking is an important practical topic in the domain of robotics. The ability of an agent to identify, localize and capture the state of objects is an essential task required for perception of and interaction with a dynamic environment. A variety of methods are possible for capturing the state information of a target object – with some examples being: mechanical trackers which consist of sensorised linkages connecting an object to a frame of reference, inertial trackers such as accelerometers which measure the rate of change of velocity, as well as vision based systems that optically capture a scene and use processing to extract relevant information [Stamou *et al.* 2005]. Compared to techniques that measure the object state directly, vision can capture vast additional information such as multiple objects, position, pose, and context as well as being versatile in the sense that it does not intrinsically depend on material properties or the addition of sensors to a tracked object. However, extracting information accurately with real-time performance using a computationally and memory constrained device remains an inherent challenge in robotics [Gholami *et al.* 2021; Revach *et al.* 2022].

Modern computer vision is dominated by deep learning approaches. Traditional computer vision utilizes pipelines with hand-crafted feature extractors and shallow machine learning algorithms to perform tasks like classification and regression. The effectiveness of deep learning techniques over traditional techniques results from the ability to learn feature extraction from training data as well as introducing end-to-end learning which facilitates simultaneous optimization of the entire architecture [Mahony *et al.* 2019]. Traditional approaches are still useful in certain contexts, typically providing intuitive and interpretable solutions and often requiring less data, memory and training.

More formally, in visual object tracking, the aim is to use a sequence of image frames to robustly estimate the motion state of a target object [Li *et al.* 2013]. Many different visual object tracking paradigms are possible, with some popular traditional approaches being tracking-by-detection, optical flow, silhouette tracking and discriminative correlation filters to name a few [Zhang *et al.* 2022]. During initialization of any object tracker – unless manually targeted – objects must first be selected using an object detector [Yilmaz *et al.* 2006]. One might question then, why not always just apply object detection on each frame (as is done in tracking-by-detetion) to track an object? According to Bolme *et al.* [2010] tracking is less demanding than detection since tracking algorithms

can use fewer computational resources than running an object detector on every frame. Modern approaches to tracking include applying CNN feature extraction to traditional tracking frameworks and end-to-end deep learning where a variety of network architectures have been used including convolutional neural networks (CNN), siamese neural networks (SNN), recurrent neural networks (RNN) and generative adversial networks (GAN) [Li *et al.* 2017]. With the advancement of modern object detection methods for real-time operation, such as You Only Look Once (YOLO) [Redmon *et al.* 2015], it becomes interesting to apply these effective models to a tracking-by-detection framework. According to Wojke *et al.* [2017], this approach has become the leading paradigm for object tracking.

Object detection is a computer vision task where objects are identified (classification) and localised in an image (regression). From a single frame it is possible to determine position (and other states such as pose), however, without establishing correspondence within an image sequence time-dependent states such as velocity cannot be computed and the state estimate lacks robustness. Here robustness means that noise such as misclassification is attenuated as well as providing instance identification and resilience to occlusion. These can be achieved by linking the frames of the image sequence (also known as data association). Yilmaz *et al.* [2006] states that the nearest neighbour approach is the simplest method to perform correspondence.

A downside of deep learning approaches is that they demand a significant amount of training data in order to achieve useful results. Accurately labeled real world training images often come at a labour cost, which makes the use of a simulated environment an efficient choice. This approach is still practical for application in the real world as advances in domain adaptation have shown significant success in mapping a simulated environment to a physical environment [Zhang *et al.* 2018]. Additionally, photo realistic rendering can also be applied [Müller *et al.* 2018]. These topics will however not be investigated in this research. In this research, visual ball tracking is investigated in the context of 3D simulated RoboCup soccer, which is an international robotics competition that has been entered by the Robotics, Autonomous Intelligence and Learning (RAIL) Lab from the School of Computer Science and Applied Mathematics at the University of the Witwatersrand [Ingram *et al.* 2021]. Although the current focus of the team is the 3D simulation league, physical leagues are in consideration for the future.

The research report continues as follows. Firstly in the introductory chapters, a background section is presented (Chapter 2) which gives relevant context with regards to the research problem and the chosen techniques. This is followed by a related work section (Chapter 3) which presents a selection of published work, in a similar problem space, where some of the important methods and results that are considered in this work are summarized. Then, the research problem is outlined (Chapter 4) prior to presenting the research methodology (Chapter 5). Continuing to the central chapters, the investigations focusing on object detection (Chapter 6) and object tracking (Chapter 7) are presented respectively. Finally, a concluding chapter is presented (Chapter 8).

Chapter 2

Background

In this section, an overview is given of some of the underlying topics relevant to ball tracking. Firstly, the object tracking problem is considered (Chapter 2.1). It is found that a tracking-by-detection approach can be used to produce state-of-the-art results by taking advantage of the progress made in the popular object detection problem space. Further, the intuitive Kalman filtering algorithm is found to still be relevant in the modern tracking context and a recent approach to improve its results for complex systems by incorporating a neural-based step is discussed. Following on, the object detection problem is considered (Chapter 2.2) and modern approaches are addressed. The RoboCup Soccer background of the research is introduced (Chapter 2.3).

2.1 Object Tracking

Object tracking describes the problem of estimating the trajectory of an object as it moves around the scene. Object tracking can be divided into offline (also known as batch) and online problem spaces. In the offline problem a complete image sequence can be used, whereas in online problems only the images of the previous time steps are available [Zhang *et al.* 2022]. Real-time tracking is online by definition. Some of the challenges include object occlusion, objects going out of frame, complex object motion, background clutter, dynamic backgrounds, imaging noise, motion blurring, illumination changes, object deformation, multiple instances, monocular vision and real-time processing requirements [Yilmaz *et al.* 2006].

A visual object tracker can be understood using the general framework suggested by Wang *et al.* [2015] where the authors have decomposed and studied the object tracking problem in its constituent parts. Figure 2.1 provides a representation of a generalized tracking process. Here it is assumed that an ensemble model is not utilized, this is because an ensemble is often avoided in real-time applications due to an increased computational demand. The modeling of the object's motion and appearance aids with data association as well as improving trajectory estimation [Bewley *et al.* 2016].

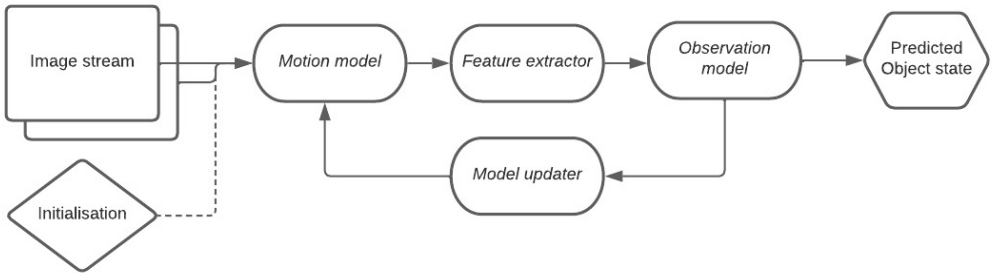

Figure 2.1: Generalized tracker flowchart (Redrawn from Wang *et al.* [2015])

The components are described as follows:

- Initialization – An object to be tracked is selected manually or by detection.
- Motion model – Generates candidates that may contain the tracked object.
- Feature extractor – Acquires a representation of the objects in the image.
- Observation model – Classify if is the tracked object. Assigns detection to target.
- Model Updater – Adaptation of the observation model.

It is understood that not all components have equal importance, which is useful insight when real-time performance is desired. Wang *et al.* [2015] report that contrary to popular belief, it turns out that the observation model (which is the focus of many papers on visual tracking) does not play as important role as often thought in a tracker. Feature extraction is found to affect the performance most, which is a task that CNNs excel at. This has been exploited in the popular "Simple Online and Real-time Tracking" (SORT) object detection algorithm of Bewley *et al.* [2016].

2.1.1 SORT

SORT is a visual object tracking technique that is based on the Kalman filter [Kálmán 1960] and Hungarian method [Kuhn 1955]. A Kalman filter is a recursive state estimator that fuses a motion model and measurement to predict the motion state of a linear system [Porikli and Yilmaz 2012].

Kalman filtering

In this model, object motion is described by a discrete linear time invariant (LTI) state-space process which defines the evolution of the state from time $t-1$ to time t as:

$$x_t = Fx_{t-1} + Bu_t + w_t \tag{2.1}$$

where x_t is the object state at time t, F is the state transition matrix (from a system dynamics model) which describes how the state evolves between time t and $t-1$, B is the control-input matrix and u is the control vector which models the effect of control signals into the system. w is the process noise vector which is normally distributed about a mean of zero with process error covariance Q which accounts for neglected and uncertain modeled behavior. By definition, the system's properties and behavior remain consistent over time.

An observation model describes the relationship between the object state and the measurement at the current time step:

$$z_t = Hx_t + \upsilon_t \tag{2.2}$$

where z_t is the measurement vector at time t, H is the measurement matrix which maps measurements to the object state and υ is the measurement noise vector which is normally distributed about a mean of zero with covariance R which accounts for measurement uncertainty.

The Kalman filtering algorithm is then performed for state estimation as follows. The "prediction step" is computed as:

$$\hat{x}_{t|t-1} = F\hat{x}_{t-1|t-1} + Bu_{t|t} \tag{2.3}$$

$$\hat{P}_{t|t-1} = F\hat{P}_{t-1|t-1}F^T + Q \tag{2.4}$$

where $\hat{x}_{t|t-1}$ and $\hat{P}_{t|t-1}$ are the state and the covariance predictions at time t.

The "correction step" is then computed as:

$$K = \hat{P}_{t|t-1}H^T(H\hat{P}_{t|t-1}H^T + R)^{-1} \tag{2.5}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K(z_t - H\hat{x}_{t|t-1}) \tag{2.6}$$

$$\hat{P}_{t|t} = (I - KH)\hat{P}_{t|t-1} \tag{2.7}$$

K is known as the Kalman gain which, for linear systems with Gaussian noise, can be optimally computed for mean-square error to apply a correction based on the relative uncertainty between the process and measurement. This can be understood as a trade-off between the model predicted state and the measurement. In typical usage, $\hat{P}_0$ is initialized to a relatively high value which has the impact of the system favoring the measured state while the state estimate is very uncertain.

In the case of a nonlinear state transition or measurement mapping, object motion can be modeled more generally by:

$$x_t = f(x_{t-1}, u_t) + w_t \tag{2.8}$$

$$z_t = h(x_t) + \upsilon_t \tag{2.9}$$

Systems with nonlinear state transition or measurement mappings can be linearized at each time step, using the Taylor series expansion, and used in the standard equations to obtain the Extended Kalman Filter (EKF) [Porikli and Yilmaz 2012]. The resulting matrices can each be computed as the Jacobian matrix (first-order partial derivatives):

$$F_t = \frac{\partial f(x_{t-1}, u_t)}{\partial x_{t-1}} \tag{2.10}$$

$$H_t = \frac{\partial h(x_t)}{\partial x_t} \tag{2.11}$$

The EKF is non-optimal however remains a powerful and popular tracking algorithm.

SORT algorithm

With real-time performance being a crucial component of online tracking, the light weight and minimalist nature of the SORT algorithm supports rapid update cycles. Despite its simplicity, it performs exceptionally well. Bewley *et al.* [2016] state that this approach achieves an accuracy comparable to state-of-the-art online trackers. Furthermore, due to the computational simplicity of their method, the algorithm achieves a rapid update rate of 260 Hz which is quoted to be over 20x faster than other state-of-the-art trackers and certainly exceeding real-time requirements.

SORT comprises of the following components:

- To model the states of a tracked object, a first order Kalman filter is used with a linear constant velocity assumption. Without a detection associated with the object, its state is simply predicted without correction using the linear velocity model. With an associated detection, the predicted bounding box (BB) can be used to update the object states using the Kalman filtering algorithm.
  The state of each object is modelled as:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \tag{2.12}$$

  With u and v representing the horizontal and vertical location of the center of the object and the area (s) and aspect ratio (r) of the target's bounding box.
- For feature extraction, the original implementation uses an object detector known as "Faster R-CNN", however, since SORT utilises only bounding box position and size for both motion estimation and data association it is possible to evaluate and choose any sufficiently performing object detector [Bewley *et al.* 2016].
- To assign detections, each object's expected bounding box is predicted for the current time step. The IoU score (see object detection) is then computed between each detection and all predicted bounding boxes. The association problem is then solved optimally using the Hungarian algorithm [Kuhn 1955].

Bewley *et al.* [2016] reiterate the strong influence of object detector performance on overall tracking results. Further, the relevance of the Kalman filter is reaffirmed in modern tracking applications. More recent work by Salscheider [2021] shows that a constant velocity model with a Kalman filter and Hungarian method is still a powerful tracking approach. For robotics applications, Pereira *et al.* [2022] states that SORT trackers are still achieving state-of-the-art results with a high frame rate.

2.1.2 Deep SORT

As an extension to the SORT algorithm, Deep SORT (Simple Online and Real-time Tracking With a Deep Association Metric) has been introduced by Wojke *et al.* [2017]. In this extension, appearance information is included in the observation model in an attempt to improve the performance for multiple objects. The addition is effective at reducing the number of object instance identity switches, which permits longer periods of occlusions, however, at a cost of real-time performance. Therefore in the case of real-time single object tracking, it does not provide sufficient benefits over SORT.

2.1.3 KalmanNet

The Kalman filter remains an effective and popular tracking algorithm that finds itself in common use in computer vision and robotics [Urrea and Agramonte 2021]. A downside of motion model based methods is that performance relies on accurate knowledge of the underlying dynamics – which typically requires domain knowledge and valid assumptions. Modern data driven approaches try to learn entirely from data, however, they have many parameters, require significant amounts of data and lack intuition, which ultimately prevents their application in embedded systems [Revach *et al.* 2022].

Revach *et al.* [2022] propose a hybrid data-driven and model based approach to real-time Kalman filtering, where the underlying state space model is non-linear and accurate knowledge of it is unavailable. In this approach a recurrent neural network (RNN) (more specifically a Gated Recurrent Unit (GRU) variant) is trained on representative data in order to predict the appropriate Kalman gain at each time step (see figure 2.2). Compared to "traditional" feed-forward neural networks where fixed length input data simply flows through the network to compute an output, in an RNN the input as well as a hidden state (which represents the history) are fed back into the network – which allows a series of temporal information to be processed (see figure 2.3).

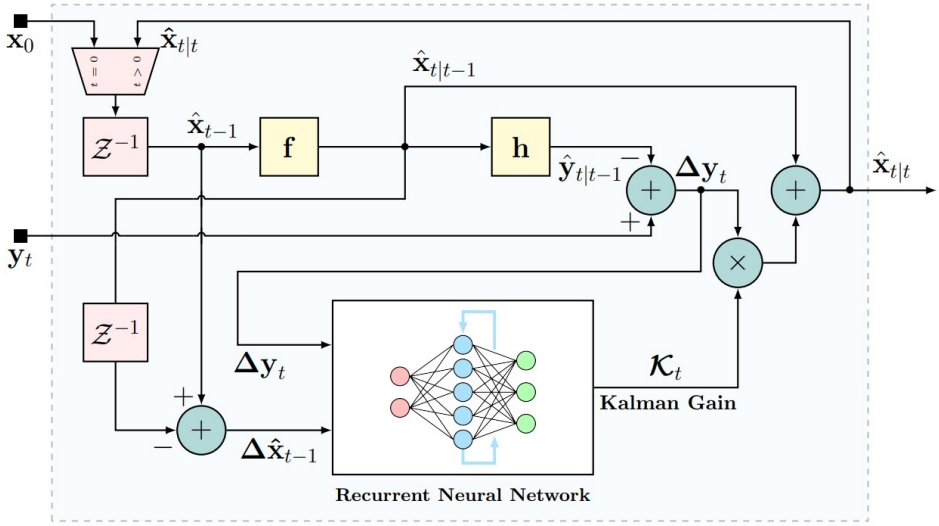

Figure 2.2: KalmanNet block diagram [Revach *et al.* 2022]

In this manner, KalmanNet aims to remove the dependency on accurate knowledge of the underlying process and measurement noise and facilitates a partially known or approximated underlying state space model from a system dynamics model. The model can be trained end-to-end which means that the Kalman filter is embedded in the network and a loss function can be formulated to evaluate the tracking performance directly, rather than by training intermediate subcomponents and creating a pipeline.

The interpretability of the original algorithm can be retained whilst potentially learning complex dynamics implicitly, reducing the need for domain knowledge and model assumptions that are needed to accurately characterize a complex system as a tractable state space model. According to Revach *et al.* [2022], KalmanNet is shown to converge

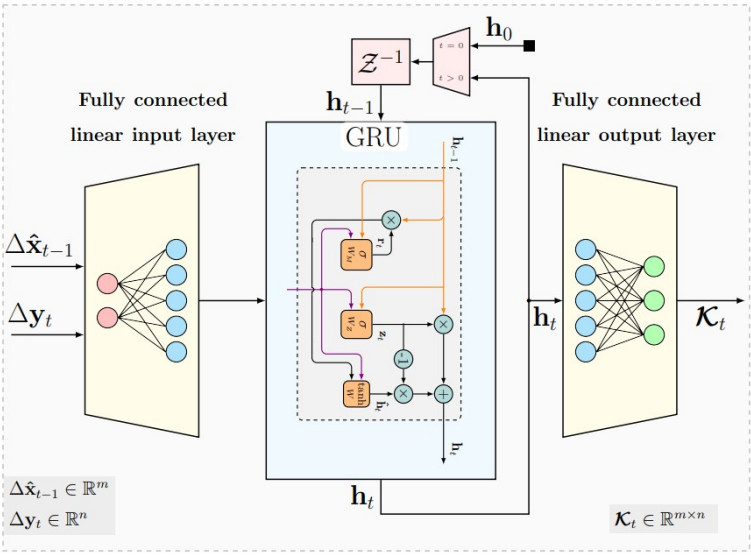

Figure 2.3: KalmanNet RNN [Revach *et al.* 2022]

much faster compared with purely data driven systems, while outperforming the model based Extended Kalman filter, Unscented Kalman filter and particle filter, when facing model mismatch and dominant non-linearities. This data driven approach to Kalman filtering has recently found other robotics applications – replacing the extended Kalman filter for online Simultaneous Localization And Mapping (SLAM) – a crucial task for robotic systems where a map of an environment is simultaneously constructed while localizing the position of an agent [Choi *et al.* 2022].

2.1.4 Performance measures

A useful performance measure used for evaluating state estimators is the Root Mean Square Error (RMSE) – which can be considered as the standard deviation of the difference between the predicted state and the ground truth [Havlík and Straka 2015].

RMSE can be computed as follows:

$$RMSE = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\epsilon_i^2} \tag{2.13}$$

where ϵ_i is the error for the ith sample which is the difference between the predicted and ground truth state. RMSE is often used as a statistical design criterion for noisy systems as it prescribes a soft bound based on probability (for true Gaussian noise it is by definition that 99.7% of samples will occur within three standard deviations).

2.2 Object Detection

Object detection is one of the oldest and most fundamental computer vision problems. Within this scope are facial recognition, pedestrian detection and multi-class image classification which have historically received significant attention in various competitions and the establishment of various datasets such as Caltech pedestrians [Dollar *et al.* 2009], PascalVOC [Everingham *et al.* 2010] and Microsoft COCO [Lin *et al.* 2014].

2.2.1 Traditional object detection

With the most successful approaches being based on discriminative models – a traditional object detector can be considered in general to have the following components: Region selection, feature extraction and classification [Zhao *et al.* 2019]:

- In region selection, different sub-windows are considered at various scales and are passed into the detector window. This may be by region proposal or by scanning the entire image at different scales. This approach is less complex than processing the entire image in a single pass and allows a single detector to be trained and applied throughout the image, but is computationally expensive.
- Feature extraction attempts to capture patterns in the image that represent an object or provide semantic information. Low-level features (textures, edges, etc.) may be combined with spatial information into a single feature descriptor.
- A classifier predicts the presence of an object class in the window given the extracted features. The decision boundary (which is an $n-1$ dimensional surface in an n dimensional domain) is learned using training examples. For a positive detection, the current window is returned as a bounding box. Since multiple windows will return positive for the same object, techniques such as non maximum suppression can be used to isolate a single prediction.

Some of the criticisms of traditional strategies are:

- Generating bounding boxes with a sliding window strategy is inefficient and the localization is often inaccurate (loose fitting bounding boxes).
- Using hand crafted low-level features it is difficult to adequately capture the semantics of the image.

Traditional object detectors that had relative success and some real-time applications include the Viola-Jones algorithm [Viola and Jones 2001] as well as the HOG features and SVM classification based D&T detector [Dalal and Triggs 2005].

Viola-Jones detection

The Viola-Jones (VJ) detector [Viola and Jones 2001] was a breakthrough algorithm that was considered to be the first viable real-time object detector still maintaining state-of-the-art accuracy. Although targeting face detection, it can be equally applied to different object classes. The VJ detector uses a sliding window detector with a fixed step size. A scaling factor can be applied to the window which increases the filter size proportionally after each complete image traversal in order to search for larger faces.

Firstly considering feature extraction, an observation can be made that classes of objects, such as faces, have a typical layout that can be approximated by patterns of adjacent lighter and darker regions. These "Haar-like" features (figure 2.4a) are applied to a gray-scale image to capture the differences in regions with a single number. Filters of pattern A and B respond to edges, filters of pattern C respond to lines and filters of pattern D respond to diagonal features. During training, an exhaustive search for features is performed where all possible scales, aspect ratios and positions of the filters in the detection window are considered for each training face example – in the case of a 24x24 window approximately 180k features exist in each detection window.

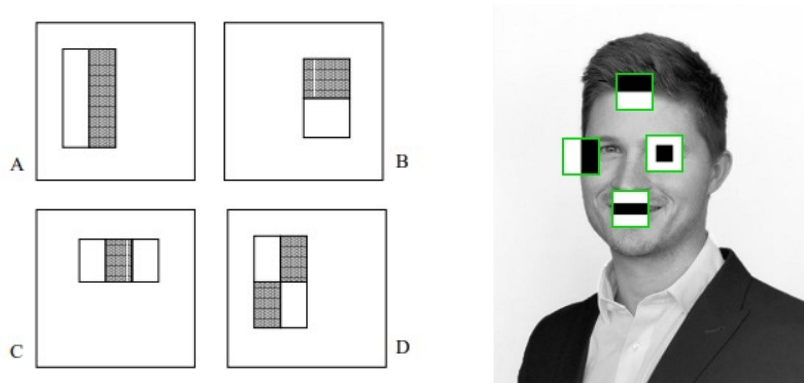

Figure 2.4: Haar features [Viola and Jones 2001] (Left) and strong responses (Right)

Viola and Jones (2001) introduce an elegant approach to evaluate these features rapidly by precomputing an intermediate "integral image" which allows the subsequent computation of a region in constant time as it only has to be evaluated once per image and can then be used for each feature.

In order to classify each window, there are a significant number of Haar-like features to consider at each window. An efficient approach to classification, based on the AdaBoost machine learning algorithm, is used as a "feature selection" process by training and selecting a subset of weak classifiers that can be combined into a strong classifier. The set of "cascading classifiers" is arranged as a degenerate decision tree, where each node is a stage of increasing complexity. Negative classifications cause an early exit from the cascade, therefore background regions are usually rejected quickly and more computation is spent on distinguishing complex regions. Multiple stages may be used, which are progressively more "difficult" to pass – candidates who pass all stages in the cascade are classified as the target object.

The speed of processing an image is therefore impacted by the complexity of the image (complex backgrounds can slow it down). Using fewer stages and features can result in faster computing times. Faster speeds can also be achieved by increasing the step sizes and scaling factor. Models that are under-trained tend to focus on background areas which can lead to impractical inference times.

2.2.2 Modern object detection

Before the dramatic growth and change of focus towards modern computer vision techniques, traditional object detectors were reaching saturation. Girshick *et al.* [2014] writes: *"Object detection performance, as measured on the canonical PASCAL VOC dataset, has plateaued in the last few years. The best-performing methods are complex ensemble systems that typically combine multiple low-level image features with high-level context"*.

Deep learning approaches are considered to have become the state of the art since the year 2012 with AlexNet becoming the first CNN to lead the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Krizhevsky *et al.* 2012]. The first widely popularized object detector to make use of CNNs is R-CNN (Regions with CNN features) [Girshick *et al.* 2014], which uses CNNs for feature extraction but otherwise retains a traditional detector pipeline with region proposal. CNN based detectors which utilize a region proposal stage are known as two-stage detectors [Arani *et al.* 2022]. Modern object detectors are able to achieve near perfect results on datasets where traditional architectures fail to achieve any useful performance. This can be seen in the benchmarking of the You Only Look Once (YOLO) object detector (see figure 2.5) [Redmon *et al.* 2015] versus the famous "D&T" detector of Dalal and Triggs [2005].

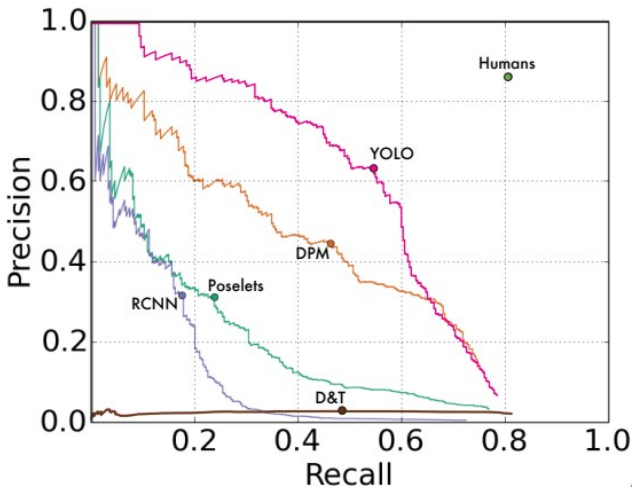

Figure 2.5: YOLO precision-recall curve (Picasso Dataset) [Redmon *et al.* 2015]

Modern object detectors utilize deep CNNs in order to learn feature extraction. CNNs are based upon the mathematical operation of convolution, where a matrix of trainable weights called a filter kernel is swept across an image. CNNs take advantage of the spatial context of an image and are able to extract both low-level and high-level contextual features due to a hierarchical network structure, often referred to as the backbone of the network. The networks can be trained by minimizing a loss function that combines the regression and classification performances by using an algorithm known as back propagation to search for optimum kernel weights. The object detection problem can be considered as the combination of both classification and regression tasks. A loss function is a measure of the error of the prediction for the considered task and the aim of training is to find a local minimum of this objective.

Least square error (L_2 loss) is a common choice for regression loss when considering continuous variables:

$$L_{regression} = \sum (y_i - \hat{y}_i)^2 \tag{2.14}$$

where y_i is the ground truth.

Categorical cross-entropy loss is a popular choice for classification tasks when categorical variables are considered, as it is readily differentiable for back-propagation:

$$L_{classification} = -\sum_{i=1}^{n} t_i log(p_i) \tag{2.15}$$

with t_i being the ground-truth and p_i being the softmax probability for the i_{th} class.

More recent architectures have seen great success by encouraging models to learn from more challenging examples, rather than focusing on "easy" examples, by adding an additional factor that reduces the weighting of easy cases. Such a loss function is known as a "focal loss" [Lin *et al.* 2018]:

$$L_{focal} = -\sum_{i=1}^{n} (1 - p_i)^{\gamma} log(p_i) \tag{2.16}$$

where γ is a tuneable variable known as the focusing parameter.

Various loss functions can be combined into a multipart loss function which is engineered to achieve a certain behavior.

YOLO

A breakthrough for real-time object detection was achieved with the introduction of the one-stage (AKA single-shot) detector YOLO by Redmon *et al.* [2015], which achieved real-time speeds whilst remaining competitive with state-of-the-art detectors. The speed of single stage detectors is in general considered to be faster than that of two-stage detectors – but at a cost of accuracy. According to Carranza-García *et al.* [2021] regarding two stage detection: *"This architecture can achieve more reliable detection, but increases the required computation, hence being less convenient for real-time applications"*.

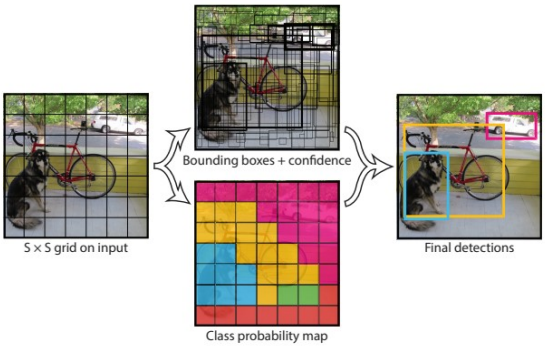

Figure 2.6: YOLO object detection [Redmon *et al.* 2015]

In the basic YOLO architecture by Redmon *et al.* [2015], the image is divided into an $S \times S$ grid. Each grid cell is used to predict B bounding boxes with associated confidence scores and C class probabilities which is demonstrated in figure 2.6. This is done by passing the output of the backbone into fully connected layers with output nodes representing each cell. Redmon *et al.* [2015] considers disadvantages of this approach to be a sensitivity to hyper-parameters, difficulty detecting objects much smaller than the anchors as well as suffering localization error: *"While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones"*.

To improve performance, the YOLOv2 architecture by Redmon and Farhadi [2016] introduces (besides other incremental improvements) the usage of predefined bounding boxes which are used as initial guesses to facilitate the model learning more diverse object sizes and aspect ratios. These are known as "anchor" boxes, with the technique dubbed an anchor-based approach. The box dimensions (dimensional priors) are learned by k-means clustering of the dataset. Further improvements have been made with YOLOv3 [Redmon and Farhadi 2018] and YOLO v4 [Bochkovskiy 2020b], with the most significant enhancements being a deeper and more optimized backbone, a loss function tailored to the evaluation metric as well including various proven pre- and post-processing tricks ("bag-of-freebies" and "bag-of-specials") to improve the accuracy with little overhead. These models are also released with a "tiny" version which further trades accuracy for speed and model size. Note that numerous unofficial "YOLO" detectors have also been created which have no affiliation to the original researchers.

NanoDet

Contemporary object detection advancements include anchor box free detectors – by reformulating the problem as "keypoint" detection. Arani *et al.* [2022] report: *"The keypoint-based methods generally outperform the anchor-based and two-stage methods in terms of accuracy and speed"*. Although CornerNet [Law and Deng 2018] was proposed as the first alternative to the single stage anchor-based approaches, more recent and successful models include Fully Convolutional One-stage Object Detection (FCOS) [Tian *et al.* 2019] and the subsequent NanoDet [RangiLyu 2021] which has been shown to be the current state-of-the-art for real-time object detection as demonstrated by the comprehensive study of Arani *et al.* [2022].



Figure 2.7: Keypoint based object detection [Tian *et al.* 2019]

In the "FCOS-style" key-point approach which NanoDet is based on, each object is modeled as a single center point of a bounding box. The network is used to predict heatmaps where the intensity peaks are representing keypoints. Regression of the bounding boxes can then be performed utilizing key-points rather than anchors. A benefit of this approach is that it eliminates the anchor-box hyper-parameters, which are dataset-dependant, as well as avoiding the problem of having to compute and reject redundant (non-maximal) box predictions. According to Zhou *et al.* [2019] *"Most successful object detectors enumerate a nearly exhaustive list of potential object locations and classify each. This is wasteful, inefficient, and requires additional post-processing."*

After passing the input image through the convolutional backbone for feature extraction, NanoDet utilizes a modified lightweight Path Aggregation network (PAN) [Liu *et al.* 2018; RangiLyu 2021] neck before passing the output to its detection head. PAN is an incremental improvement of a Feature Pyramid Network [Lin *et al.* 2017] which is a popular technique in the object detection space. In a feature pyramid network, the output feature maps taken from various shallow and deeper levels from the backbone network (representing spatial and semantic information respectively) are utilized by the model to better detect objects across a range of scales. This is a similar concept to the idea of an image pyramid which is popular with traditional approaches, however, it naturally leverages the pyramidal hierarchy of the backbone without the additional computational overhead.

2.2.3 Performance measures

In order to evaluate object detection algorithms, it is important to choose a measure that represents the desired performance in a comparable manner (not algorithm specific). In the context of classification, a tempting approach is to evaluate the accuracy:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.17}$$

However, in the case of class imbalance this measure gives misleading results. Consider an extreme of 0.5% of ground truth labels being positive in a dataset – a classifier that predicts all labels as negative will achieve 99.5% accuracy, however will not have learned any aspect of the data. Therefore precision is often used, which considers the correctness of the positive predictions that have been made:

$$Precision = \frac{TP}{TP + FP} \tag{2.18}$$

This metric however only considers the cases when a prediction has been made. Therefore it must be considered along with recall (sensitivity) which considers what proportion has been successfully predicted across the entire dataset:

$$Recall = \frac{TP}{TP + FN} \tag{2.19}$$

If using only recall, all examples could be classified as positive and no learning will have occurred. Therefore precision and recall must be considered together.

Precision and recall have a relationship (see figure 2.5) through the chosen threshold of the confidence value, as only making predictions when the confidence is high, means that less confident cases are missed. The relationship can be plotted as an ROC curve, however, comparing the performance of multiple algorithms using this approach makes it difficult to evaluate quantitatively. In order to represent this curve as a single value for comparison, there are a variety of approaches possible including F1 score, Area Under Curve and Average Precision (AP).

AP is a popular metric used in the context of modern object detection – traditionally calculated by averaging the sum of the precisions (p) across 11 evenly distributed levels in recall. A good score requires high precision and recall (r): [Everingham *et al.* 2010].

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, 0.2, \ldots, 1\}} p_{interp}(r) \tag{2.20}$$

where:

$$p_{interp}(r) = \max_{\widetilde{r} > r} p(\widetilde{r}) \tag{2.21}$$

AP_{small}, AP_{medium}, AP_{large} are variations used to denote differently sized detection objects (COCO [2023] considers images less than 32×32 as small and larger than 64×64 as large), to get an understanding of the performance over scale. In order to account for multiple classes, the mean average precision (mAP) is commonly computed which is nothing but the sum of the AP per class divided by the number of classes:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.22}$$

When detecting an object, a bounding box is placed around the target. It is possible to compare the quality of the localization prediction with the ground truth by evaluating the overlapping areas – called Intersection-over-union (IoU). A perfect overlap results in a score of 1 and a score of 0 for a complete miss. AP may be evaluated at a given threshold of IOU, for example an IoU of 0.5 may be specified (denoted as AP@.50). A series of IoU might be used by averaging the AP results over the range to evaluate the results in isolation of threshold (denoted as AP@[.5 : .95]) [Aidouni 2019].

A criticism of these commonly used metrics is that they do not consider run-time or memory usage – which are required for practical deployment. The real-time performance is difficult to compare between algorithms since most papers do not go into significant depth regarding the relationship between accuracy and speed [Huang and Rathod 2017] – with most vision applications focusing on quality of prediction. A further challenge is that the actual real-time performance is also largely dependent on the hardware and software infrastructure on which the algorithm is executed. An approach to understanding the run-time performance (sometimes called latency) of different architectures is to consider relative comparisons that are bench-marked under the same conditions. Typical metrics are frames per second (FPS) or Hz.

2.2.4 Optimizing models for real-time performance

In practice, a trade off exists between the real-time performance and accuracy achieved by a model when performing inference. Intuitively, when less data is taken into consideration – such as by the reduction of image resolution – inference can be faster as it requires less computation, however, the accuracy may decrease as the detail and structure of the data are lost. Therefore the performance of a computer vision algorithm should not be evaluated in isolation of latency for real-time application. According to Huang and Rathod [2017], many computer vision architectures tend to focus on accuracy and therefore may use model ensembling and multicrop methods which are too slow for practical real-time implementation.

Deep learning models in particular tend to be large in both memory and computational demands relative to the capacity of embedded computers, that are often used in robotics applications, which are typically space and power constrained. Gholami *et al.* [2021] write, *"This creates a problem for realizing pervasive deep learning, which requires real-time inference, with low energy consumption and high accuracy, in resource-constrained environments"*. Apart from selecting efficient network architectures, quantization offers possible means to reduce the memory consumption and computational costs of a model which can result in improved real-time performance. The aim of quantization is to represent the weights and activations of the model with reduced precision without a significant impact on accuracy.

2.2.5 Neural network development and deployment

A variety of software tools have been developed in order to abstract the low-level implementation details of neural networks from high-level model architecture for the convenience of the researcher. New deep learning frameworks have often coincided with leaps in network performance such as Caffe (Berkeley AI) [Jia *et al.* 2014] – utilised for the development of R-CNN [Girshick *et al.* 2014] – as well as Darknet – used for the development of YOLO [Redmon *et al.* 2015]. Popular contemporary frameworks such as Tensor Flow [Abadi *et al.* 2016] and PyTorch [Paszke *et al.* 2019] facilitate the development of a variety of model architectures such as CNN, RNN, and fully connected networks as well as providing support for CPU and GPU accelerated training and inference. These open-source frameworks, supported by major tech companies, facilitate the rapid advancement of machine learning technologies.

Researchers often implement model architectures using the paradigm of a particular framework and therefore models are often not easily interchangeable between frameworks. Further, the libraries and tooling provided by these frameworks are typically used in the context of development and training and are therefore not optimum environments for practical model deployment where only the forward pass is desired as they are large in memory and are not necessarily portable between computing architecture and platform. In the context of a real world multidisciplinary development environment with many interacting components, such as Robotics, it is good practice to consider the management of subsystems and the run-time environment as well as avoiding dependencies so that models from a variety of frameworks might be used.

Although some popular computer vision packages such as OpenCV do have an interface to load models from a limited set of frameworks – not all operations are supported, each framework has different configurations and run-time optimization is not possible. Empirically, the real-time performance is poor. Open Neural Network Exchange (ONNX) is a well accepted standard [Fehlner 2022] for machine learning interoperability that provides a neutral format that can be executed in the ONNX Runtime environment. This inference engine supports different neural network types, multiple platforms, hardware architectures, operations as well as execution and model optimizations such as quantization. Models compiled to be run on ONNX Runtime have shown to receive significant real-time performance improvements [Stefani *et al.* 2022].

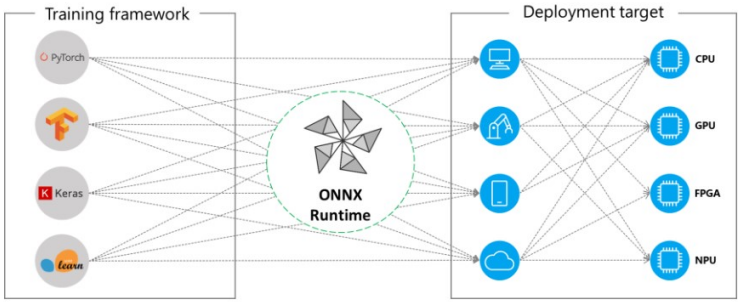

Figure 2.8: ONNX Model environments [Fehlner 2022]

2.3 RoboCup Soccer

RoboCup is an internationally coordinated robotics competition with a focus on advancing artificial intelligence and robotics research. The ultimate goal of the soccer competition is to support the development of a team of fully autonomous humanoid robot soccer players that are capable of winning against the winner of the most recent soccer World Cup, by the year 2050. The RoboCup challenge features a dynamic environment and distributed control performed in real-time [RoboCup 2022].

There are various leagues which each concentrate on different challenges that need to be addressed in order to achieve the ultimate objective of RoboCup. Of particular interest to the RAIL lab is the RoboCup 3D Soccer Simulation League (3DSSL) – which is demonstrated in figures 2.9 and 2.10 – as well as the Standard Platform League (SPL) as a possible future endeavor. SPL is a hardware based league that utilises the Nao v5, featuring an embedded Intel Atom Z530 CPU with 1GB RAM [Aldeberan 2023]. 3DSSL has a focus on the software programming aspects of the Nao and therefore serves as a possible bridge into the hardware league.

In RoboCup soccer, the position of the ball is crucial for determining appropriate agent behavior. At a close observation range the agent may need to kick the ball, in this case large errors will result in a miss-directed kick or perhaps a complete miss. At middle ranges the agent may attempt to block an opposition passing option and should position itself accordingly. At long ranges the agent may need to know what region of the field the ball is in so that an appropriate behavior is chosen.

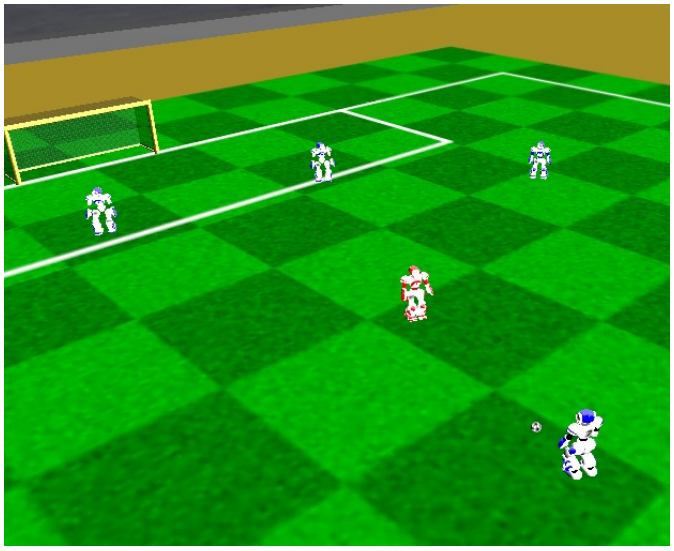

Figure 2.9: Simulated RoboCup Soccer

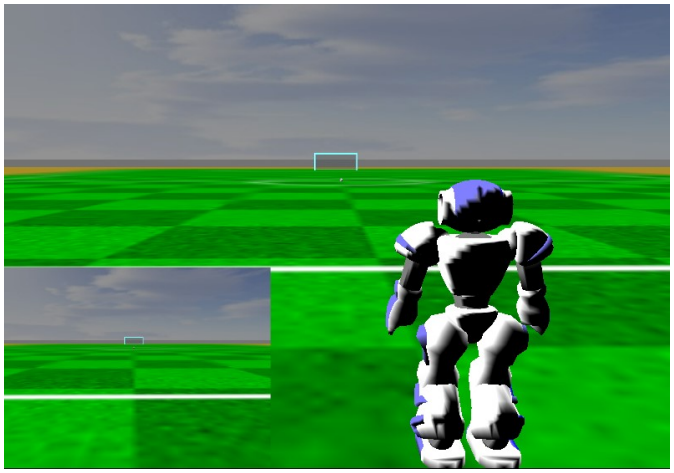

Figure 2.10: Simulated image perceptor (Agent perspective in bottom-left subwindow with a ball in the distance)

2.3.1 RoboCup soccer simulator

3DSSL runs on a simulator known as the RoboCup Soccer Simulator Server (RCSSServer3d) which is developed using plugins to a generic physical multi-agent simulation platform, SimSpark. In the architecture of SimSpark, agents receive sensor feedback from "perceptors" and take actions using "effectors" [Boedecker and Asada 2008]. In 3DSSL, a vision perceptor returns preprocessed visual data, in spherical coordinates, to each agent every two simulation cycles (40ms). Therefore in 3DSSL, computer vision is not currently within the scope of the challenge. The capabilities do however exist within the simulator to provide an image perceptor that produces rendered simulated camera images by enabling a plugin. The existing RAIL lab domain knowledge and infrastructure can therefore be utilised (and developed) to investigate perception algorithms that may be ported to a physical league. Some of this infrastructure includes "Fat-proxy" which is used to abstract agent actions from low-level effector commands.

Chapter 3

Related Work

In this section, a selection of published works that are related to ball detection (Chapter 3.1) and tracking (Chapter 3.2) in RoboCup are considered. This overview lists some important approaches and results whilst keeping in mind the historical context of traditional versus modern techniques by arranging them chronologically. It is shown that recent approaches take advantage of fine tuning state-of-the-art CNN object detector architectures for transferring to their own problem domain as well as Kalman filtering for tracking.

3.1 Ball Detection

The object detection space has significantly changed over time, with the most successful detectors being deep learning based. Using what is now considered a traditional approach, Mitri *et al.* [2004] implement a Viola-Jones based object detector that performs real-time ball detection with a focus on mobile robots in a RoboCup soccer environment. They perform an edge detection preprocessing step which allows them to achieve a satisfactory color-independent result.

Attempting to apply more modern deep learning techniques to improve real-time ball localization, Speck *et al.* [2016] propose a novel CNN to localize multicolored, patterned balls for RoboCup by formulating their approach as a regression problem. They develop an architecture that can process an entire image at once to avoid a sliding window approach by predicting Gaussians that correspond to probabilities of the ball location. They find their proposed model to be too large for their robot (over 2GB) but also find that reducing their model size leads to poor performance. Speck *et al.* [2016] conclude, *"Thus, either faster processors for robots are necessary (to run the bigger nets) or low classification rates at medium to high distances have to be accepted"*. Although the design was not successful, the single shot strategy for real-time inference is widely adopted by state-of-the-art detectors such as YOLO. This might also be seen as a step toward the keypoint based approach.

Basing their work on the then state-of-the-art single shot architecture, YOLO v3 Tiny, Szemenyei and Estivill-Castro [2019] propose a real-time object detector that is fine tuned for RoboCup ball detection for the Nao v5 platform. The maximum reported inference speed that is achieved is only 13 FPS. The authors also introduce a technique they describe as "synthetic transfer learning", which involves pretraining the network on a fully synthetic dataset to improve convergence.

Fernandes *et al.* [2021] evaluate several CNN models on an embedded RoboCup robotic system. They find that, for their RoboCup dataset, YOLO v4 Tiny performs the best in terms of the AP@75 metric (scoring 0.549), as well as for the small and medium objects, which points towards this model being the best for accurate ball detection. However, the detector fails to execute at real-time speed on their Small Size League (SSL) platform, hence, the selection of the Mobile SSD architecture.

3.2 Ball Tracking

In the ball tracking problem space, Kalman filtering has been successful and remains particularly popular. The approach of fusing measurements with a dynamic physics model allows noise attenuation as well as substep estimation for problems such as a parabolic trajectory or rolling with friction.

After winning the RoboCup-97 small-robot competition, Veloso *et al.* [1998] describe their development of an EKF model – for their team CMUnited – to track the motion of a kicked ball. The tracker includes a velocity discounting factor in the state model which is used to represent a non-linear rolling friction.

In order to establish the real-time capability of a non-linear Kalman filtering approach for use in RoboCup, Birbach *et al.* [2009] investigate tracking of ball trajectories using an unscented Kalman filter (UKF) and monocular vision. Although only the tracking problem is considered (manually processed images) the investigation confirms that, compared to a maximum likelihood estimator, UKF is feasible as a real-time tracker: *"after half the time of flight the error is about 0.3m which we judge to be enough for playing soccer"*.

With a focus on RoboCup Middle Size League, Silva *et al.* [2012] consider the use of an extended Kalman filter (EKF) for real-time 3D Ball trajectory estimation using a single camera. In their research they propose applying a maximum Likelihood method to provide an initial estimate of the ball trajectory for the initialization which improves tracking of the ball.

Zhang *et al.* [2020] investigates real-time golf ball tracking by taking advantage of state-of-the-art CNN based object detection and Kalman filter based tracking – with small object detection as a primary focus. They report: *"most object detectors do not perform well on small object detection"* where it is explained that since most detectors are pretrained on datasets, such as MS COCO, where the size of the objects tends to be larger. The detectors: Faster R-CNN, SSD, YOLO v3, YOLO v4 and RefineDet are investigated, however, significant GPU hardware is utilised – which at this stage is not practically accommodated on embedded robotics platforms.

Chapter 4

Problem Statement, Research Aims and Objectives

4.1 Problem Statement

In RoboCup soccer most of the decision-making involves the position of the ball on the field. This affects factors such as how the players should move, when and where they should kick and how they should behave. In order to facilitate decision making, it is important to have an accurate understanding of the ball's position. In this research project, the interest lies in determining if it is feasible to accurately capture the trajectory of a kicked ball in the RoboCup simulator using a computer vision based system, with a neural network based approach utilized for real-time tracking.

4.2 Research Hypothesis

Hybrid approaches to state estimation which fuse intuitive model based approaches with measured data are a promising technique. It is hypothesized that a neural network assisted approach can outperform the traditional EKF method at tracking the trajectories of a kicked RoboCup soccer ball as measured by RMSE for real-time inference speeds, using a tracking-by-detection framework.

Modern object detection has significantly improved upon traditional approaches, with contemporary models promising state-of-the-art detection with real-time performance. Tracking-by-detection is a paradigm that takes advantage of this technological leap and can be paired with the popular Kalman filter for real-time robotics applications. It is hypothesized that a more recent key-point based modern object detection approach (NanoDet) can be used to outperform established RoboCup approaches such as an anchor based method (Yolo v4 Tiny) as well as the traditional Viola-Jones detector for RoboCup soccer ball detection – measured by the quality of inference using AP scores, at a real-time speed.

4.3 Research Aims And Questions

The first aim of the research is to prepare a suitable real-time object detector trained on ball localisation. A traditional architecture (Viola-Jones) as well as a more modern anchor based detector (YOLO v4 Tiny) serve as a benchmark against a recent keypoint based approach (NanoDet). To motivate the selection of NanoDet, a variety of pre-trained detector models which use different approaches is compared on latency and memory usage. Inference speed is evaluated on a laptop CPU, which is a mobile and physically constrained computing device that is both accessible and more representative of a typical robotics platform than a high-performance computing cluster would be. An AMD Ryzen 5 4600H is available.

A significant challenge for object detection is the size of the ball which at a distance is small in the image (as defined by COCO [2023]). The ball used in RoboCup is also partially white in colour and may be difficult to distinguish against the field markings, goals and Nao robots. The algorithm further needs to act in real-time since slow responses due to heavy computational load can result in delayed agent responses or lead to control inaccuracies and instability. The memory footprint of the architecture should also be considered in the context of embedded computing capacities, which can be improved using model quantization.

The principal questions which arise are:

- Can an established modern CNN object detector run in real-time on a modern mobile laptop CPU?
- Can pretraining on real images transfer to a simulated RoboCup environment?
- Does the more recent keypoint based NanoDet detector outperform the established RoboCup anchor based Yolo v4 Tiny and traditional Viola-Jones detectors, as measured by AP in real-time on a RoboCup ball detection dataset?
- Can model quantization offer a worthwhile optimization versus performance degradation?

The second aim of the research project is to use the object detector to implement a tracking-by-detection framework based on neural aided Kalman filtering. Using ball telemetry from the RoboCup simulation as ground truth and processing the associated visualization with the object detector, a tracker based on the KalmanNet approach can be trained. As a baseline, an extended Kalman filter can be implemented in a similar fashion to the RoboCup trackers found in the related works (Chapter 3.2).

A significant challenge is the non-linearity of the trajectories due to rolling friction of the ball as well as the bouncing after the initial parabolic trajectory makes contact with the pitch. Such physics is not simply modeled in a standard linear Kalman filter.

The principal questions which arise are:

- In the context of the simulated RoboCup soccer environment, can a neural aided Kalman filter outperform a traditional EKF for tracking rolling and lofted kicks?
- Can a tracking-by-detection approach meet the real-time performance requirements?

Chapter 5

Research Methodology

The methods used to investigate the proposed research topic can be separated into two distinct phases where the former focuses on object detection (Chapter 5.1) and the latter on tracking-by-detection (Chapter 5.2) using the result. In-line with the approach taken in previous research (Chapters 3.1 and 3.2) the focus of the investigation is from the perspective of the kicker, and tested against a set of randomized rolling and lofted kicks in a control RoboCup soccer environment.

5.1 Phase One: Object Detector Investigation

In this phase, the training and performance of the Viola-Jones, Yolo v4 Tiny and NanoDet object detectors are considered. To justify the selection of NanoDet, an initial comparison of a variety of popular pretrained detectors is evaluated on inference speed and memory consumption. The chosen object detectors are then fine-tuned on a representative dataset, generated using the RoboCup simulator, and evaluated in order to determine which is the most promising as measured by AP, model size and latency. The AP score is evaluated on large, medium and small object sizes (as distinguished by the COCO [2023] guidelines) as well as against occlusion and background complexity. For the optimum architecture, the impact of model quantization is investigated.

5.1.1 Dataset generation

The RoboCup simulator has an image perceptor plugin that can be enabled to produce perspective images at 25Hz (real-time). In this manner actual gameplay can be used to create representative data. Although a ground truth ball position is available to be enabled, Fatproxy does not directly support the processing of the additional server message at this stage and will crash. Handling of this data needed to be added in order to enable extracting ground truths directly. With some changes to the source code, ground truth bounding boxes can be automatically computed using a coordinate transform from the image perceptor to the camera location followed by a correction from the wide angle image, in order to describe the ball center in image coordinates. The distance of the ball is used to compute the relative ground truth bounding box size.

An unoccupied test arena is used for the kick tracking but the inclusion of match images with details such as players, field markings and the goals in the background is prudent to evaluate object detector performance in a match environment and to ensure robustness. Since the server provides feedback on whether a ball is in the vision cone of an agent, actual match footage can be generated and programmatically labeled for inclusion in the dataset. It is noted however that a check for occlusion is not programmed into the server so the ball could be completely hidden, therefore some manual work is done to remove such images and to separate occluded images. The dataset is then prepared according to the requirements of the given model.

5.1.2 Model selection

The selection of the baseline Viola-Jones and Yolo v4 Tiny detectors follows on from the related works. NanoDet is proposed since it is a recent key-point based architecture with promising capabilities. To justify this choice, a set of various popular detectors is evaluated on inference speed and memory utilization. The following pretrained models are selected from the ONNX model zoo [ONNX 2023] as well as the investigation by Arani *et al.* [2022]: Faster-RCNN, YOLO v3, YOLO v3 Tiny, YOLO v4, SSD, Mobilenet SSD, YOLOX and NanoDet. The three investigated models are then fine-tuned to the RoboCup dataset and are evaluated and compared using AP, model size and latency (as measured in a single threaded application on a modern mobile laptop CPU).

5.1.3 Training

Often pretrained model weights can be used to significantly reduce training time – but the training of modern neural network based models still requires significant compute since typically it is desired to train using various hyper-parameter combinations in order to optimize via a grid search. In order to achieve this within practical time scales the Wits Mathematical Sciences Cluster is used such that multiple models can be trained in parallel as well as taking advantage of GPU resources to achieve a training speed up.

A possible risk exists in using pretrained models since they are typically trained on real image datasets – therefore domain transfer to a synthetic domain may fail to produce acceptable results. Further, most datasets used for pretraining feature few small scale object examples which may also be problematic for this application.

5.1.4 Evaluation

Models are all translated to a neutral ONNX format which serves two purposes here: Firstly, the runtime environment can be controlled with ONNX Runtime in order to limit the computational resources to single threaded execution. Secondly, it mimics practical model deployment as it is a platform, device and training framework independent with support for multiple languages. To evaluate speed, a laptop's CPU is used as a proxy for a mobile computing environment. This setup, combined with ONNX runtime, simulates a constrained computing device that is both accessible and more representative of a typical robotics platform than a high-performance computing cluster would be.

5.2 Phase Two: Object Tracking Investigation

In the next phase of the project, the tracking problem is considered by using the detector from phase one for tracking-by-detection, applied in a typical configuration as depicted by figure 2.1. An EKF with a simple RoboCup world model (similar to Veloso *et al.* [1998] and Silva *et al.* [2012]) is implemented as a baseline tracker. A tracking dataset is generated using the RoboCup simulator and used to train and optimize neural assisted Kalman tracking. The tracking performances are then compared on a test dataset with RMSE as a performance measure.

5.2.1 Dataset generation

To generate a tracking dataset, a series of randomized kicks are performed in order to capture image and ground truth position data that describes various trajectories. Kicks are performed from random locations on the training pitch in order to capture a variety of scenes. The tracking dataset is produced by applying the object detector for inference which can be transformed into coordinates relative to the robot frame of reference. The ground truth ball data is directly read from the server at the default sampling rate of 50hz. This rate is faster than the camera, which means that intermediate estimations (those without correction) are also evaluated.

5.2.2 Training

The neural assisted Kalman tracking is based on the KalmanNet architecture. By attempting to reproduce the results of the authors using the available implementation from KalmanNet [2023], it was found that the implementation suffered significantly in terms of training stability, initialization and GPU support – which was limiting when applied to the chosen problem. In order to utilize the model infrastructure, it was decided to apply some architecture improvements which required familiarization with the chosen deep learning framework, PyTorch. Some architectural decisions were noticed such as an unbounded Kalman gain as well as a particular choice of preprocessing which was thought to undermine spatial information of the input feature, were targeted as areas for improvement. Therefore some refactoring is performed for this project's purposes (detailed in section 7.3.1) although not targeting the architecture in general.

5.2.3 Evaluation

The tracking performance of the trackers is evaluated and compared using the RMSE performance measure on the test dataset. The performance target is to process images at a minimum of 25 FPS, which corresponds to the update rate of the simulation and is generally accepted as real-time execution.

Chapter 6

Object Detection Investigation

6.1 Detection Dataset

This dataset is generated using various open source programs in the RoboCup soccer ecosystem as well as the Wits agent with a bespoke behavior. Half the images are acquired in a simulated match, the remainder are collected in a practice environment.

6.1.1 Software environment

The ball detection images are generated using the RoboCup match server rcssserver3d-0.7.3 which is built on the simspark-0.3.2 simulator [SimSpark 2022]. In the simulator, the "agentSyncMode" variable is changed to true to enable synchronized execution despite additional rendering and file-writing. In the match server: "enableRealTimeMode" is set to false, "addNoise" is set to false and an "ImagePerceptor" is added to the Nao model to enable rendered perspective images to be appended to the server messages sent to each agent.

The Wits agent message parser is extended to include the image messages that are received in a base64 string encoded format. Each serialized image is stored in a .json file, for readability, which includes the image metadata such as the spherical coordinates of the ball (see figure 6.1) – if the ball is inside the agent vision cone and the image size.

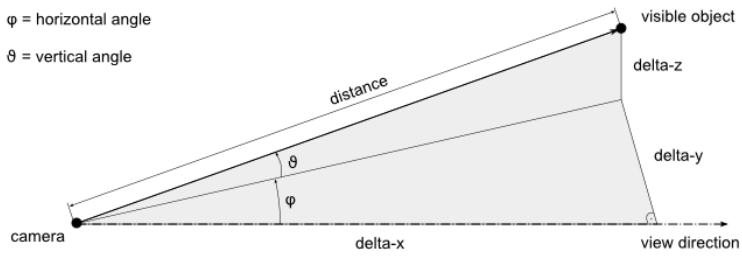

Figure 6.1: RoboCup agent visual perception [SimSpark 2022]

Low-level control of the agent is abstracted by using magmaFatProxy-1.2.2, which is forked and extended to include the handling of image messages.

6.1.2 Preprocessing

A script is written to decode the images into an RGB array format that can be displayed, manipulated and stored using the Numpy and OpenCV Python libraries. In order to better understand the performance of each detector in the context of the image content, images are sorted into categories that classify ball visibility, occlusion, size and play mode. Hidden and occluded ball images are manually separated, since the server vision messages do not include the breakage of line-of-sight (all objects are transparent to the agent).

From the spherical coordinates of the ball that are given by the match server relative to the center of the head of the Nao agent, one can transform them to the camera position by converting the coordinates to a Cartesian axis, performing a translation (given 6.5cm head radius) and then returning the coordinate system to spherical. From the spherical coordinates at the camera, with the known 58° field of view of the camera, it is possible to use an ideal wide angle lens transformation to determine the BB center in image coordinates. This transformation is based on the ideal fish-eye lens model (see figure 6.2) which is then extended to the two rotational axes using spherical geometry. The size of the bounding box is then determined based on the distance of the ball (with a known 8.4cm diameter) and the expected size on the optical projection plane.

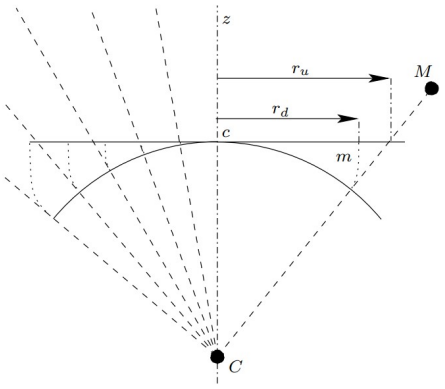

Figure 6.2: Ideal wide angle lens transformation [Devernay and Faugeras 2001]

Although the transformations can automatically be computed to get BBs for all ball containing images, using the simulator to produce perspective images for control purposes is outside of the typical usage of the simulator – it is found that in some agent initializations (approximately 1 in 5) that the synchronization of the produced images to the ground truth is between time steps and the temporal misalignment visually obvious. In these cases the data is dropped, however, it required a manual process of inspecting the batches, as a way of detecting bad initializations was not found.

Each object detector expects the dataset to be presented in a certain structure with annotations in a specified format in order to facilitate dataloading. NanoDet utilizes the COCO format [format 2023] which is commonly used by recent detectors, whereas the VJ detector expects a bespoke format [OpenCV 2023] and YOLO uses a “Darknet” format. Scripts are made to automate these conversions, respectively, from the initial generated data. A 60:20:20 training, validation and test split is used.

6.1.3 Data exploration

The dataset is interrogated in order to understand and sanity check the acquired data and the randomly sorted splits. Table 6.1 shows a breakdown of various image properties. The size of the object is classified according to COCO [2023], which declares small objects as those less than 32×32 pixels and large as greater than 64×64 pixels. Images are considered occluded if any of the pixels of the ball are hidden.

Dataset	Count	Split	Visible	Hidden	Occlu.	Match	Drill	Small	Med.	Large
Total	2563	100%	70.6%	20.1%	9.3%	50.6%	49.4%	54.6%	25.1%	0.2%
Train	1537	60%	70.4%	20.3%	9.3%	51.0%	49.0%	53.5%	26.2%	0.1%
Validation	513	20%	72.9%	18.9%	8.2%	51.1%	48.9%	58.3%	22.8%	0.0%
Test	513	20%	68.8%	20.9%	10.3%	70.6%	50.9%	54.2%	24.4%	0.6%

Table 6.1: Detection dataset

With the data acquired using the target environment, the dataset is considered representative. It is seen that the properties are consistent through each split. The majority of the images have a visible ball, which is expected since the agents have a priority to search for the ball. There are a non-negligible number of occluded ball images since a match environment features many obstructing players. Further, it is clear that small bounding boxes are the majority of the cases, with large being very infrequent.

In figure 6.3 some examples of the headings used in table 6.1 are portrayed:

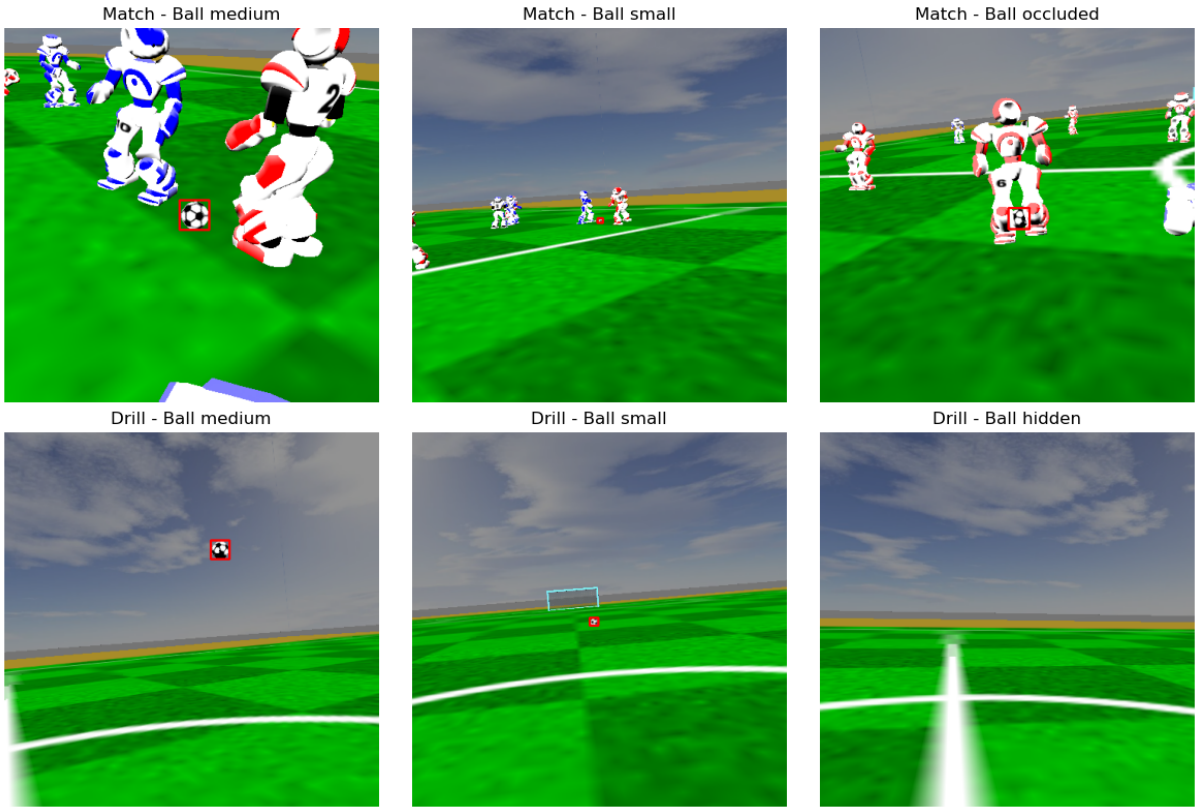

Figure 6.3: A selection of detection dataset ground-truths

The bounding box characteristics are plotted in figure 6.4. It is seen that the ball has a bias to be found in the center of the agent's perspective, this is caused by the agent which tends to focus on the ball. Data augmentation such as cropping can be used to avoid over-fitting on this trait.

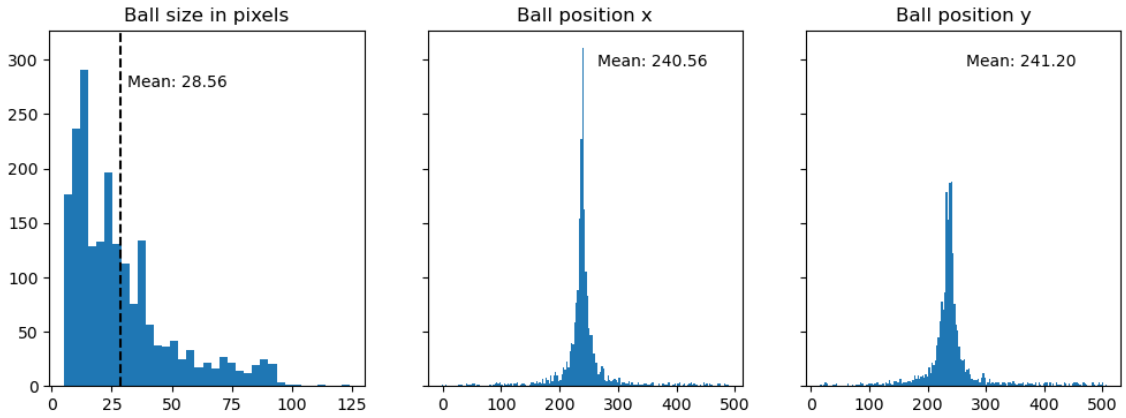

Figure 6.4: Detection dataset ball statistics

6.2 Baseline Detection

The baseline detectors, which have been chosen based on known prior usage in RoboCup as seen in the related works, are now investigated.

6.2.1 Viola-Jones

The Viola-Jones classifier (more generally known as the Haar Cascade Classifier) is implemented in the OpenCV libraries. It can be trained, stored and loaded for performing inference. A multi-scale implementation is utilized which, applies a scaling factor ($scaleFactor$) that reduces the image size to create an "image pyramid" to evaluate different object sizes with the same detector.

In the following section, the steps taken to prepare the software environment and data are described. Following this, the training process – which aims to maximize the quality of the inference – is discussed. Finally, an ablation study is performed to maximize the real-time performance of the detector whilst preserving sufficient accuracy.

Software environment

It is found that the latest OpenCV releases no longer support the training of Viola-Jones style detectors from version 4 onwards due to dropping support of the legacy C API (although, inference is still supported). Therefore it is chosen to roll back and build the OpenCV 3.4 release. The official documentation by OpenCV [2023] is used.

Data preparation

In the spirit of the related works of Mitri *et al.* [2004], for soccer ball detection, a pipeline is created where the input image is edge filtered and a hand crafted threshold applied to create a crisp binary image as an input feature.

With the detection data prepared and structured accordingly, the $opencv_createsamples$ application is used to create positive examples by cropping the objects using the ground truth bounding boxes and storing them in a binary format (named "vec-file"). The size $(w \times h)$ of the positive examples are specified and must match the window size of the classifier to be trained. The OpenCV [2023] recommendation is to use the size of the average object as the window dimensions – which in this case is 28×28. The output is shown in figure 6.5 (gray pixels caused by interpolation during rescaling).

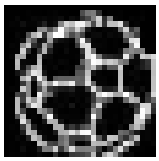

Figure 6.5: Thresholded edge image

Model training

The training of the detector is done as per Viola and Jones [2001]. In this approach, rather than trying to perform a complete optimization of the number of stages, features per stage and threshold per stage – which is considered to be difficult – a goal is chosen for both the allowed minimum hit rate and the desired maximum false alarm rate for each stage of the classifier. Each additional layer reduces the false alarm rate but also the hit rate, layers of features are added until the goal is met or exit conditions are reached.

The hit rate defines the amount of positive object training data that must be correctly classified. A higher hit rate prevents overly confident features that can cause positives to be incorrectly classified. The overall hit rate is computed as:

$$Overall_hitRate = hitRate^{numStages} \tag{6.1}$$

False alarm rate sets the rate at which false positives may occur. A lower number requires more weak classifiers to reject negatives correctly which therefore dictates how many features (also known as layers) are included in each stage.

$$Overall_falseAlarmRate = falseAlarmRate^{Num_Stages} \tag{6.2}$$

The number of features used is then increased until the target rates are met for this stage. Training further stages are usually stopped early if $acceptanceRatio$ (negative samples wrongly classified as positive) is reached, which prevents over-fitting of training data. In this study, a search is performed over $numStages$, and evaluated for the AP score against tested and validation sets, in order to ensure that the model is both sufficiently trained but also not over-fitting, for the actual targeted detection task.

In line with Mitri *et al.* [2004] $minHitRate = 0.999$ is chosen. A search for $falseAlarmRate$ (FA) is performed, which when coupled with $numStages$ allows for a variety of $Overall_hitRate$ and $Overall_falseAlarmRate$ combinations to be evaluated. $numPos$ is chosen to be 90% the positive samples, providing a conservative buffer of images to compensate for cases falsely not detected. A common choice for $numNeg$ is twice $numPos$ [Stackoverflow 2017]. The remaining parameters are recommended values of OpenCV [2023].

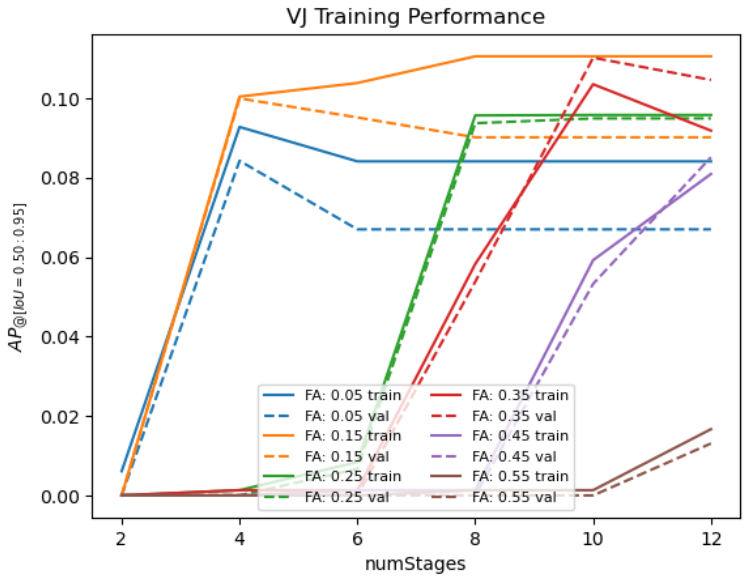

Figure 6.6: AP vs $numStages$

With some difficulties obtaining GPU support, CPU based training was performed using the Wits Mathematical Sciences computing cluster hosting Intel Xeon E5-2680 processors. The speed up achieved using OpenMP was sufficient to allow reasonable training times. Adding additional stages coincided with an increased training time, with the longest for 12 stages being roughly 6 hours. Initially, it can be seen on the training curves (figure 6.6) that the addition of stages allows the model to better fit the data, which leads to convergence – a show of sufficient training. However, the further addition of stages can lead to over-fitting which results in a decrease in validation performance. In general, the validation and training scores are not significantly different, which indicates that sufficient training examples are provided and that the datasets are reflective. The best performer on the validation set ($FA = 0.35$) is chosen for further optimization.

The first layers of the final stage can be visualized in figure 6.7, and is then discussed:

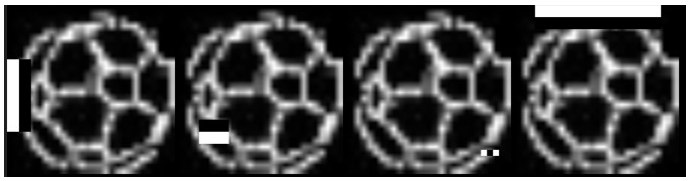

Figure 6.7: Cascade classifier 4 layers of final stage

In the layers investigated, it can be observed that the model seems to learn: the outer boundary of the ball, the thickness of the edges and the patches of the ball. These make intuitive sense and provide confidence in the results that have been obtained.

During inference, $minNeighbors$ provides a measure of confidence to retain the candidate, which is fine-tuned on the validation set to maximize the AP score (figure 6.8).

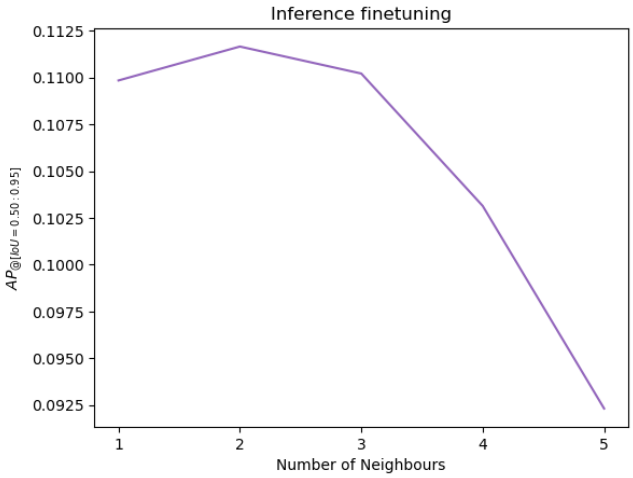

Figure 6.8: AP vs $minNeigbours$

Model tuning

An ablation study is now performed in order to optimize accuracy in the context of real-time performance. A search is performed to optimize the $scaleFactor$ (figure 6.9). It is found in general that as the scale factor is increased the inference speed increases linearly – this is since the steps between reduced images are larger therefore less computation is performed as the image is being considered in less detail. With same reasoning, as the scale factor is increased the average precision scores also decreases, however, a local maximum is achieved which can be explained by reduced false positives.

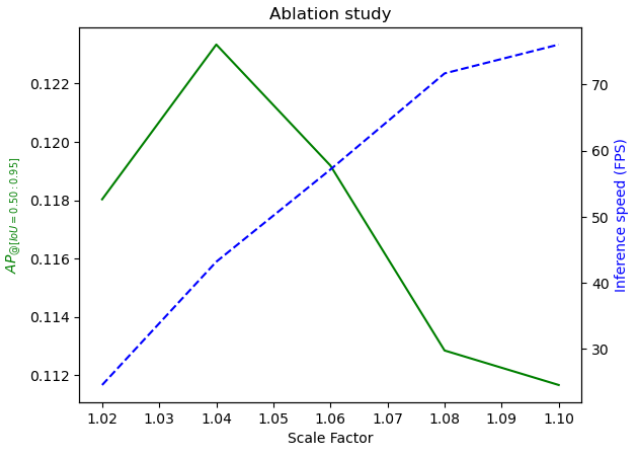

Figure 6.9: AP, latency vs $scaleFactor$

6.2.2 YOLO v4 Tiny

YOLO object detection is based on the Darknet deep learning framework, which is purpose built for training and evaluating the YOLO architecture in the object detection task. Alongside the framework, the pretrained YOLO v4 Tiny [Wang *et al.* 2021] model from the author is made available, which has been pretrained on the COCO dataset, as well as the configuration and pretrained backbone that have been used to get the reported results. Since the COCO dataset features a "sports ball" category, it is possible to apply the pretrained model to the RoboCup ball dataset in order to evaluate how well the models generalize between the domains as well as get a feel for the performance gain that can be achieved at the cost of additional training.

In the following section the steps taken to prepare the software environment and data are described in order to train YOLO v4 Tiny on the RoboCup dataset. Trained models of this framework are represented and stored by a configuration file and weights file. External tooling can be used to convert the results into the desired ONNX format.

Software environment

The "YOLO V4" release is cloned from the official repository hosted on Github [Bochkovskiy 2020a]. Before compiling the framework on the target training hardware, CUDA 11.8 is linked in the makefile – in order to benefit from GPU acceleration. The training dataset needs to be prepared in a "Darknet style" format, therefore a script is created to do this. The web service "Roboflow" [Roboflow 2023] is utilized as a means of validating the produced annotations by importing and visualizing the data.

Model training

The release configuration for YOLO v4 Tiny was utilized for model training which uses the CSPDarknet53 backbone (pretrained on ImageNet). Links to these can be found on the Github repository [Bochkovskiy 2020a]. As part of the "bag-of-freebies" training approach which can boost accuracy without an increase of the inference cost, a set of augmentations such as photometric and geometric distortions as well as simulated object occlusion are applied. In order to train the network, a multipart loss is minimized [Bochkovskiy 2020b]. The classic YOLO objective function has the following high-level form:

$$L_{YOLO} = L_{cls} + \lambda L_{reg} + \lambda' L_{confi} \tag{6.3}$$

where λ and λ' are balancing parameters used to weigh each term (giving importance to bounding box regression and decreasing the importance of boxes not containing objects (AKA "noobj") respectively), L_{cls} refers to the class prediction which is represented by the focal loss, L_{confi} refers to the objectness score (which is the probability that an object is present) which is represented by the L_2 loss and L_{reg} refers to the bounding box regression loss. YOLO v4 applies a loss known as "Complete-IOU" (CIoU) [Zheng *et al.* 2019] (rather than L_2 loss) which offers an objective that is made to suit the IoU evaluation metric. CIoU is an incremental improvement over "Generalized-IoU" (GIoU) [Rezatofighi *et al.* 2019] which solves the "gradient-vanishing" problem resulting from a basic IoU loss which would only "work" when the bounding boxes have overlap.

Initial trials showed that compiling the framework without any of the performance enhancing dependencies led to model training being estimated at over 200 hours to achieve completion, which was unacceptable. This was reduced by an order of magnitude if OpenMP was included for parallelizing and then by a further order of magnitude if CUDA was included for GPU acceleration of the training passes. The Wits Mathematical Sciences computing cluster hosting NVIDIA GeForce GTX 1060 processors was utilized, resulting in a model training time of approximately 2 hours which is a significantly more manageable duration. The recommended number of cycles by the author is applied for training, which is 2000 epochs per class but no less than 6000 overall.

The models are trained across various resolutions in order to investigate the speed/accuracy trade off and make a sensible selection of a real-time solution. It should be noted that model image resolutions need to be in multiples of 32 due to the model architecture.

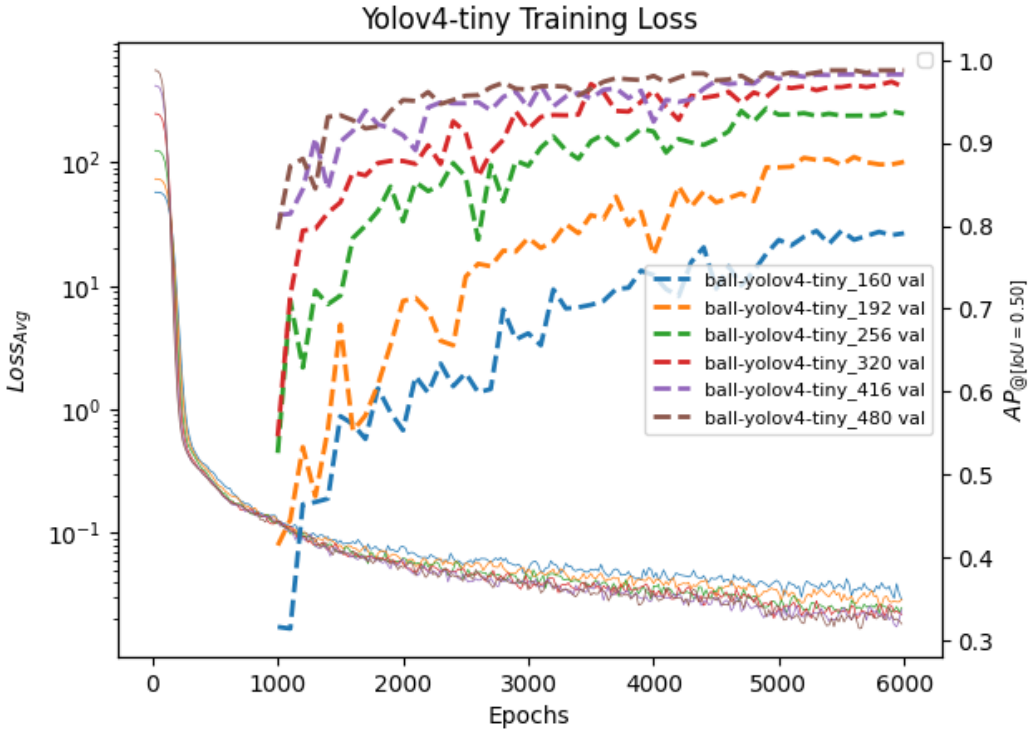

Figure 6.10: YOLO v4 Tiny training curves

In figure 6.10 it is seen that for each model, as the number of training epochs is increased the loss score decreases. This corresponds to the model learning the structure of the training dataset and performing better inference on each training example. Initially a very high rate of learning is achieved which reduces over time, however, it does not yet plateau. Too much training can lead to the model over-fitting the training dataset which can harm the ability of the model to generalize to the task it is training for. The model is also evaluated against the validation dataset every 10 epochs, which provides a measure of how well the model captures the general relationships in the data. After 5000 Epochs, it is clear that a plateau in the validation performance has occurred which is a good indicator that sufficient training has been performed.

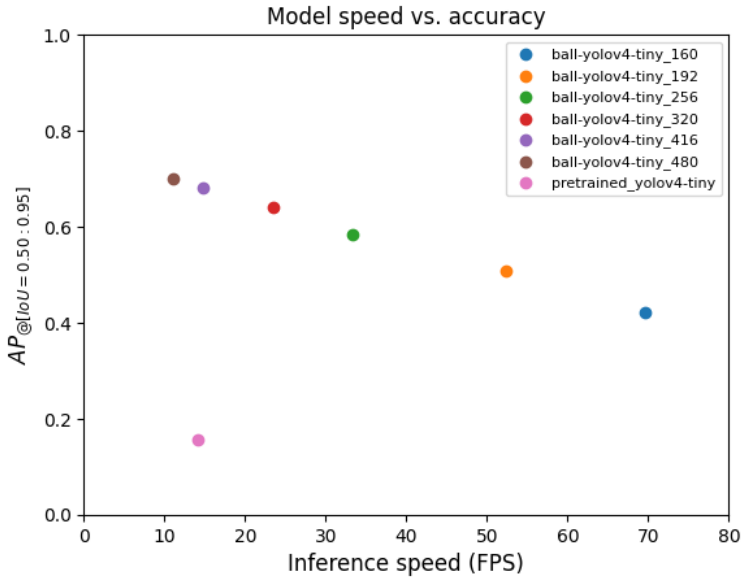

Figure 6.11: YOLO v4 Tiny variations compared

Also in figure 6.10 it can be noted that the $AP_{@[IoU=0.5]}$ score of the 480×480 resolution model achieves a near perfect validation result. This gives confidence that although the architecture has been prepared for the COCO dataset, it is capable of capturing the patterns within the data without the cost of a new and comprehensive search for hyperparameters. As the model resolution is decreased, the quality of the inference decreases. The models are also evaluated for inference speed, shown in figure 6.11. A reduced resolution allows faster inference. In order to achieve real-time inference, a significant drop in resolution from 480×480 down to 256×256 is required. This results in a drop in achievable inference quality, however, it does confirm that established modern CNN detectors can run in real-time on a mobile laptop CPU.

The additional data point of the pretrained model is also included. Although this model evaluates 80 COCO classes at 416×416, there is no significant difference in inference speed when compared to the trained model which only searches for the ball class. In principle this means that the detector can be extended to include limited additional classes such as the goal posts and other agents without significant time consequence.

6.3 Proposed Detection

Initially, a simple naive study is performed to justify the selection of the NanoDet object detector. Then, the detector is trained in order to find the most suitable configuration.

6.3.1 Naive latency evaluation

In order to justify the selection of a modern real-time object detection architecture, a naive approach is taken to simply evaluate pretrained models on the target device. In this approach, only the latency of the forward pass is considered and the accuracy of the detections is ignored. In this manner, a coarse approximation of the performance can be acquired. Further, the size of the model is also considered as a secondary factor.

The models are gathered in the following manner. All detector models from the ONNX model zoo [ONNX 2023] are used. NanoDet was found to be the most promising model, therefore more pretrained NanoDet models of different configurations were gathered and converted to ONNX format from the official repository for exploratory purposes. YOLOX [Ge *et al.* 2021a] is also a recent anchor free approach that may be a suitable alternative to NanoDet – therefore a pretrained detector is taken from its official repository [Ge *et al.* 2021b] and also converted to the ONNX format. YOLOv5 is readily accessible in the PyTorch library and is included due to its ease of access.

In this experiment, a 15 second perspective sequence (375 frames at 25hz) of RoboCup gameplay is preloaded as an array of $m \times n \times c \times frames$. The average time to complete inference of all frames is recorded and plotted alongside model size in figure 6.12. The detectors have been evaluated using onnxruntime 1.12.1 on a single core of an AMD Ryzen 5 4600H CPU, limited to single threaded execution using the session option $intra_op_num_threads = 1$.

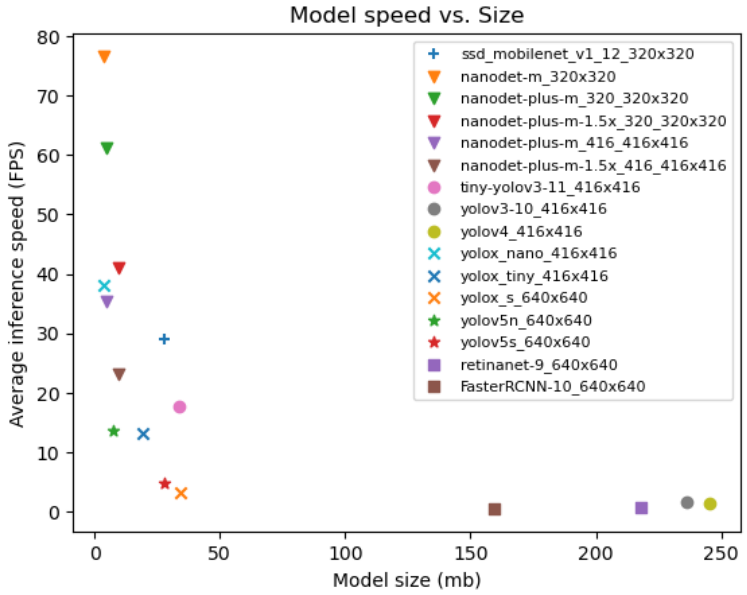

Figure 6.12: Pretrained model performance vs. size

The faster models tend to have fewer mathematical operations, and therefore also have fewer weights which results in smaller model sizes. Compared to the 1GB of RAM available on the Nao v5, the models larger than 50mb are certainly difficult to accommodate and larger than ONNXruntime Linux binary itself ($\approx 13mb$).

It is found that most of the detectors do not execute in real-time, on average, on this hardware. Although, this does not definitively mean that they cannot (after model tuning) but rather demonstrates that one will likely sacrifice a significant amount of inference quality to achieve this. Therefore these are not ideal choices to pursue. It is found that NanoDet executes in real-time for many of its configurations and therefore is a good choice to investigate further.

6.3.2 NanoDet

The NanoDet object detector is built using PyTorch which is a popular general purpose deep learning framework. Models can be trained, stored and loaded for performing inference in Python. The NanoDet repository can be found on Github with tested pretrained models for the COCO dataset which are shared by the authors [RangiLyu 2021]. Some variations of the model exist such as the "Plus" architecture which is the most recent and best performing architecture.

Software environment

NanoDet-Plus v1.0.0 is cloned from RangiLyu [2021], which is the latest release of the repository at this time. The PyTorch deep-learning framework is used however only versions $<$ 2.0.0 are supported which must therefore be manually installed, rather than using the currently outdated instructions found on the repository, before installing the required packages to get a suitable environment. It should also be noted (since it is not documented) that the "Plus" architecture has a different output shape, which may cause frustration for unknowing researchers who are using previous releases.

Model training

During an initial study using the pretrained model, it is found that the model achieves a useful performance on the target dataset (see figure 6.15) – even though it is synthetic. This confirms that real-image training can generalize to the synthetic RoboCup environment and supports an approach of fine-tuning the pretrained model onto the RoboCup ball dataset. The NanoDet model expects the data to be structured and annotated in a COCO style format, therefore a script is created to do this. The web service "Roboflow" [Roboflow 2023] is utilized as a means of validating the produced annotations by importing and visualizing the data.

In order to train the network, a multipart loss is to be minimized. The NanoDet objective function can be understood by the following high-level form:

$$L_{NanoDet} = L_{GFL}(L_{QFL}, \lambda L_{DFL}, \lambda' L_{GIoU}) \tag{6.4}$$

Generalized Focal Loss (GFL) [Li *et al.* 2020] generalizes Focal Loss from a discrete form to a continuous form – which results in training improvements. GFL unifies Quality Focal Loss (QFL) and Distributed Focal Loss (DFL). QFL is a joint representation of localization quality and classification – which softens a typical one-hot category label to a float representation of each category. DFL softens the representation of bounding box regression by viewing the offsets from the predicted center to the bounding box boundaries as a continuous arbitrary distribution rather than deterministically.

A variety of typical NanoDet architectures – which were evaluated and shared by the authors – are chosen and the default parameters and augmentations are utilized. These models have different number of weights ($1\times$ or $1.5\times$) and operate on different image sizes (320×320 or 416×416 images). Again, NVIDIA GeForce GTX 1060 processors are available and utilized for training. The training curves shown in figure 6.13 compare the training and validation results.

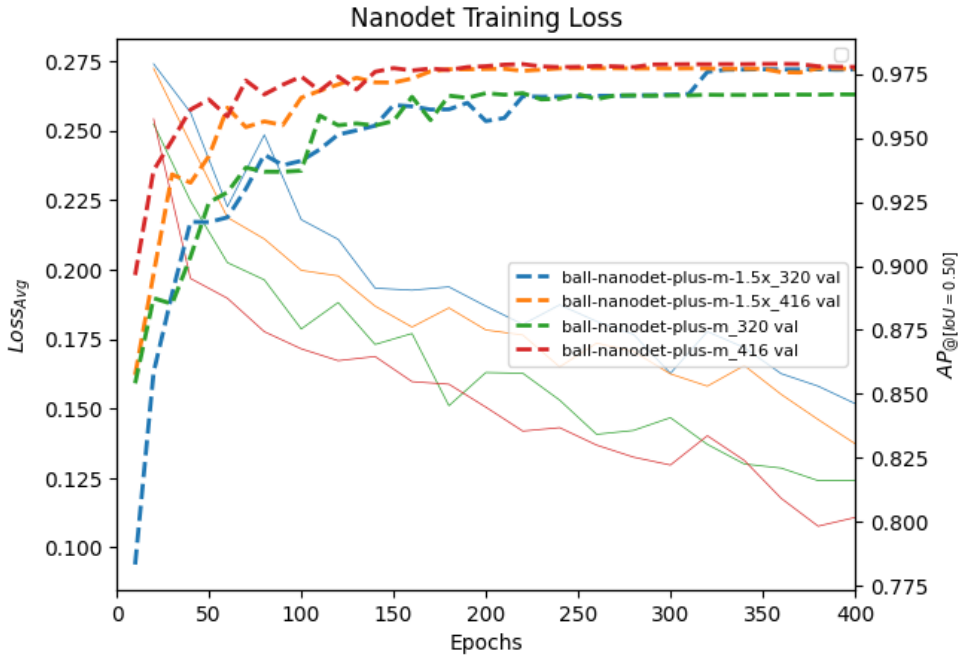

Figure 6.13: NanoDet training curves

In general, it can be seen that as more epochs are performed, the loss scores are reduced. This indicates that the model is learning the training data and improving its ability to perform object detection. Further, it can be seen that as training epochs are performed the validation scores of each of the models increases. This indicates that the model is indeed learning the general structure of the data. After a number of epochs have been performed, the validation scores reach a plateau. This indicates that sufficient training iterations have been performed and that further training may lead to over-fitting of the training data. The best iteration of each model is chosen based on the validation AP scores.

Post training, a script is used to convert the model to an ONNX format. A crucial step not documented by RangiLyu [2021] is to post-process the ONNX model with

the ONNX runtime $remove_initializer_from_input$ function. Without this, a significant degradation in inference speed occurs, preventing the detector from running in real-time. The different NanoDet variations are evaluated for their inference speeds and their performances are plotted in figure 6.14.

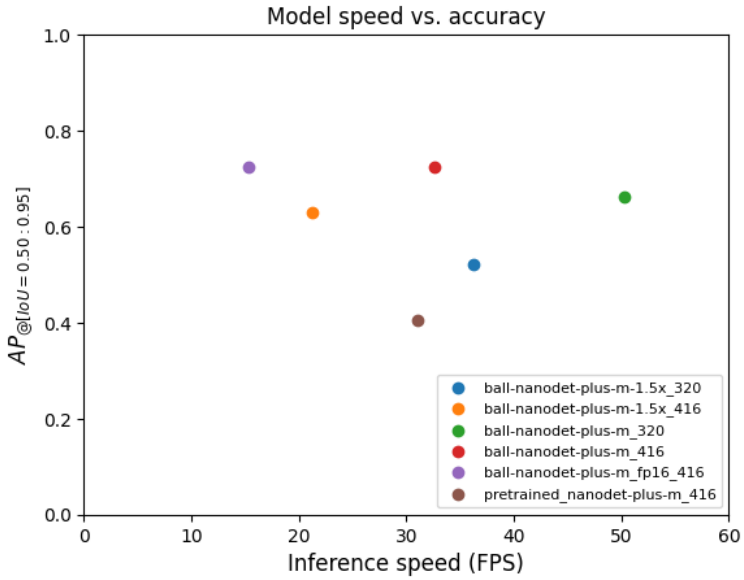

Figure 6.14: NanoDet performance visualised across variations

The validation scores for NanoDet are the highest among the considered models, without even performing a search for hyperparameters. For this reason, the additional efforts of a search are not performed (beyond the included basic architecture variations) as the results are satisfactory and sufficient to prove the respective hypothesis.

Similar to the results of all models considered thus far – it is seen in general, that the higher resolution architectures (416×416 vs. 320×320) perform inference at higher accuracy, however, at a cost of inference speed. The $1.5\times$ models also perform inference slower (they have a deeper backbone), however, in this case they do not perform at a higher inference quality. A reason for this could be that more complex models are prone to overfitting the pretraining data as they are able to capture the complex characteristics that are only present in the pretraining data. The "NanoDet-plus-m_416" instance is chosen as it provides the maximum performance at a speed that provides real-time inference.

In order to investigate model optimization for deployment, the chosen model is quantized from a float32 to a float16 representation – which gives a resulting model size reduction from 5.5mb to 3.2mb. A negligible loss of inference quality is realized but a significant reduction in speed has occurred. The reason for this appears to be that float16 operations are not well supported by all hardware (x86 CPUs) and thus require conversion to a float32 representation which adds an overhead. Since the model is already of acceptable size compared to the size of the ONNX Runtime binary, the quantized model is not utilized and further quantization is not considered worthwhile.

6.4 Evaluation Of Detection

The selected pretrained, trained and fine-tuned networks of the various object detection models are now compared relative to each other using the test dataset which has been isolated and remained unseen thus far. The dataset is decomposed into object sizes, scenario and occlusion for evaluation which helps to provide an insight into the performance of the detectors across various circumstances.

The object detection data is evaluated in three stages. Firstly, the "easiest" data is used, which does not feature match environment images or occluded images. This is most representative of the tracking portion of the research. Then, the dataset is extended to include match images (but not occluded images). It is thought that such a set would expose the model performance to a more challenging and dynamic background which is reflective of a RoboCup soccer environment and more likely to result in false positives. Finally, occluded images are added in, which helps to provide an indication of the performance against complex images with only partial object information. This provides the most realistic performance evaluation in a typical simulated soccer scenario.

The results are plotted in figure 6.15. Various conclusions can be drawn from these results, which are discussed below:

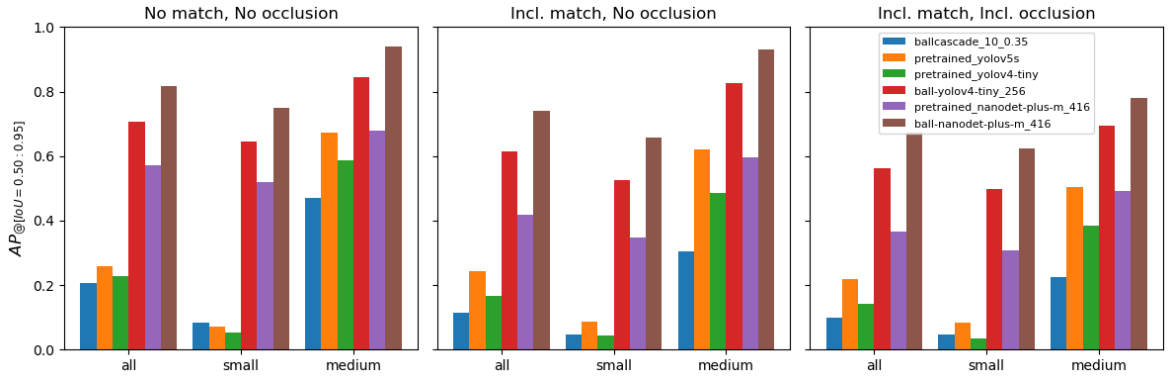

Figure 6.15: Evaluation of detectors

Performance against increasing difficulty

As a whole – it is observed that the models perform worse on the datasets of increased complexity, which is expected. Match images cause a decline in performance as the ball is more difficult to identify among the agents. The distance of the ball also corresponds to decreased performance as smaller objects are more difficult to classify and regress accurately. Additionally, occluded images cause even further performance decline as they provide only partial information for inference. A contributing factor to this is the challenge of accurately predicting the bounding box. In particular, object detection models tend to predict boundaries tightly around the visible part of the object, whereas for tracking the bounding box should also include the parts of the ball that are unseen in order to estimate distance.

Traditional detection

Relatively, the VJ detector performs the worst among the detectors across the data. Despite the effort of searching for ideal parameters, performing tuning and the trouble of training, it does not outperform any trained detector. In fact, the pretrained detectors provide better inference quality without any training effort and framework complexity. With the ease of the modern frameworks and performance of the models versus the minimal support and bespoke OpenCV tooling needed for training the VJ detector, it is obvious why utilizing the older models for new applications has fallen out of favor. When a ball is correctly identified (as in figure 6.16a), a further challenge appears to be the tightness of the fit of the bounding box due to the sliding window approach.

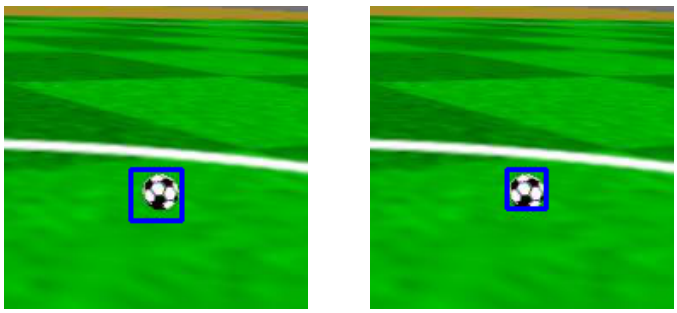

Figure 6.16: Trained Viola-Jones (6.16a cropped) & Trained NanoDet (6.16b cropped)

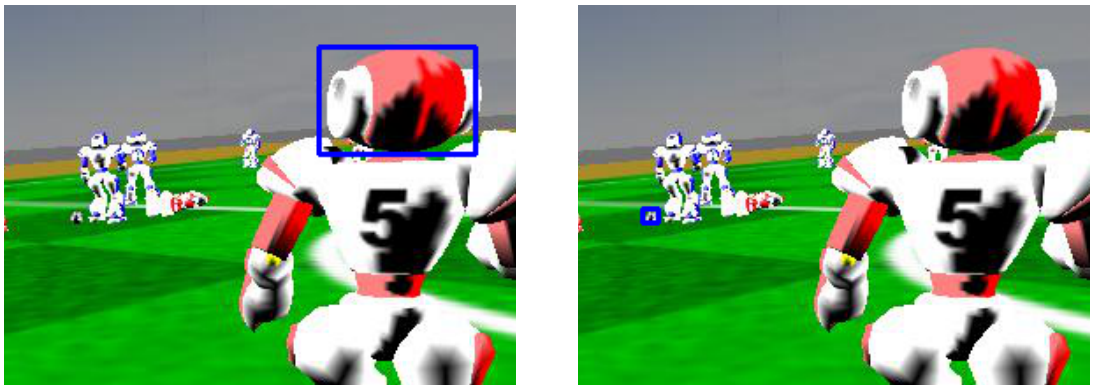

Figure 6.17: Pretrained NanoDet (6.17a cropped) & Trained NanoDet (6.17b cropped)

Pretrained modern detection

One of the interesting results that can be seen in figure 6.15 is that the pretrained detectors chosen actually achieve a useful performance on the unseen and synthetic RoboCup ball tracking dataset – although only for medium size objects (likely due to the COCO dataset under-representing small objects [Tong and Wu 2023]). This is not a generalization that one can necessarily rely on for all synthetic datasets, however, it is a useful result for expressing the domain adaptive property of the models and may be helpful for informing an initial selection of suitable algorithms that are susceptible to a task. In figure 6.17a a Nao head is misclassified as a ball and in figure 6.17b an improved result is seen after training.

YOLO v4 Tiny

The YOLO v4 Tiny model shows significant improvements in performance and surpasses the VJ detector in all categories in terms of inference quality. Initially the pretrained model suffered from poor inference of small objects but after training a significant boost in performance was realized. However, in order to achieve a real-time performance the quality of inference had to be significantly reduced. This result corresponds to the results seen in Chapter 3.1, where it was found that the YOLO v4 Tiny model does perform well in the task of RoboCup ball detection, however, these results do not carry over to real-time speeds. An unfortunate result is that the performance of the model against distance balls significantly drops in a match scenario. This reflects that the model has a tendency toward false positives, confusing the ball with the agents. This is seen in figure 6.18a where the boundary of the prediction slightly extends to include the agent behind the ball. Storage of the model weights is approximately 30mb which is larger than the binary of ONNX Runtime but still manageable with the given memory of a typical Nao RoboCup platform.

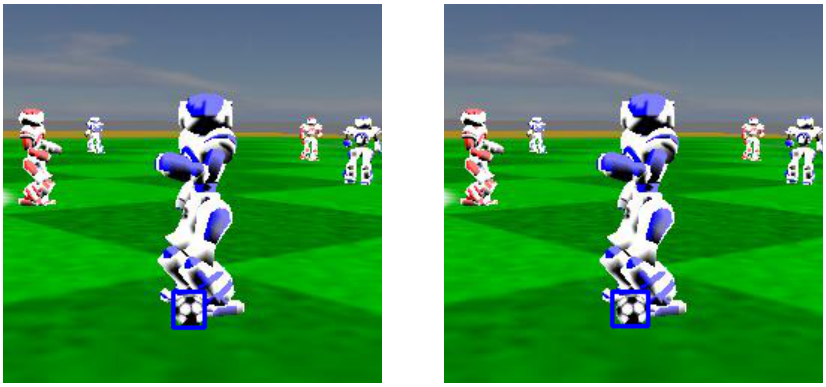

Figure 6.18: Trained YOLO v4 (6.18a cropped) & Trained NanoDet (6.18b cropped)

NanoDet

The NanoDet model shows itself to be the best network across all scenarios by outperforming all other architectures at real-time speed. This is true across each dataset characteristic and by a significant margin. A key insight that can be observed is that the pretrained Nanodet is the only pretrained model which has a useful performance for identifying the small size balls (long range) before fine tuning. This is interesting since the pretrained YOLO v4 Tiny model has also been trained for optimum performance against COCO but has negligible comprehension before training. Further, for the easiest scenario, the trained NanoDet scores a near perfect result which is impressive. This model also has the smallest memory footprint, only utilizing 5mb of storage capacity. For limited capacity machines this allows the memory to be used for other tasks, helping to improve the overall performance of the robotic platform. Therefore, for a ball detection system that may be used in a match scenario or for the proposed tracking, NanoDet is chosen. The key-point based NanoDet outperforms the baseline detectors – which proves the object detector hypothesis.

Chapter 7

Object Tracking Investigation

7.1 Tracking Dataset

As before, the dataset is generated using various open source programs in the RoboCup soccer ecosystem as well as the Wits agent with a bespoke behavior. Each kick is randomized and then captured in a practice environment as a sequence of images (with metadata) at 25hz with respective ground-truth positions at 50Hz.

7.1.1 Software environment

The kick tracking data is also generated using the RoboCup match server rcssserver3d-0.7.3 and simspark-0.3.2 simulator using the same settings as used for the detection dataset. Additionally, in the match server: a ball and agent ground truth perceptor is added to append global ball and agent positions to the server messages. The server parameters are modified to allow the agent to beam to any area of the pitch, ensuring the dataset has full coverage of the arena.

Alongside producing .json files for images with metadata as before, the agent stores a .csv file for each kick sequence which captures, for each time step, ground-truth positions as well as the agent transformation matrices from the Nao head coordinate system through the agent torso to the global coordinate system. The camera transformation used is the inverse ideal wide angle from the detection dataset – taking a bounding box at the camera position to a spherical coordinate located at the center of the agent's head. The data is synchronized by the server time. low-level control of the agent is also abstracted by using magmaFatProxy-1.2.2, but the code needs to be extended to enable the server to handle ground truth ball and agent position messages without crashing.

7.1.2 Preprocessing

A small portion of the tracking sequences need to be discarded due to abnormalities such as: agent instability, kicking outside of the play area and ball rebounding off the goal posts. These edge cases occur infrequently enough to be ignored for training, however, are manually excluded by watching the sequence.

In order to conserve memory, the tracking sequence images are taken from their .json base64 encoded string and compressed into an .avi video format – which results in a 99% reduction in size from 100mb per sequence. However, a dataloader is required to be constructed which also ensures synchronization between image and metadata.

7.1.3 Data exploration

The dataset is interrogated in order to understand and sanity check the acquired data. Table 7.1 shows a breakdown of various sequence properties. Kicks that have a peak height above the height of the diameter of the ball are deemed to be lofted, with the distribution of these kicks plotted in figure 7.1. As seen in figure 7.2, lofted kicks have a trajectory dominated by bouncing behavior compared to ground kicks which roll.

Dataset	Kicks	Split	Mean steps	Grounded	Lofted
Total	370	100%	295.4	20.3%	79.7%
Train	222	60%	294.1	21.2%	78.8%
Validation	74	20%	296.5	17.8%	82.2%
Test	74	20%	297.9	20.0%	80.0%

Table 7.1: Tracking dataset

It is seen in table 7.1 that the properties of the tracking data are consistent between the datasets. A bias towards lofted kicks is evident. The distances are plotted in figure 7.4.

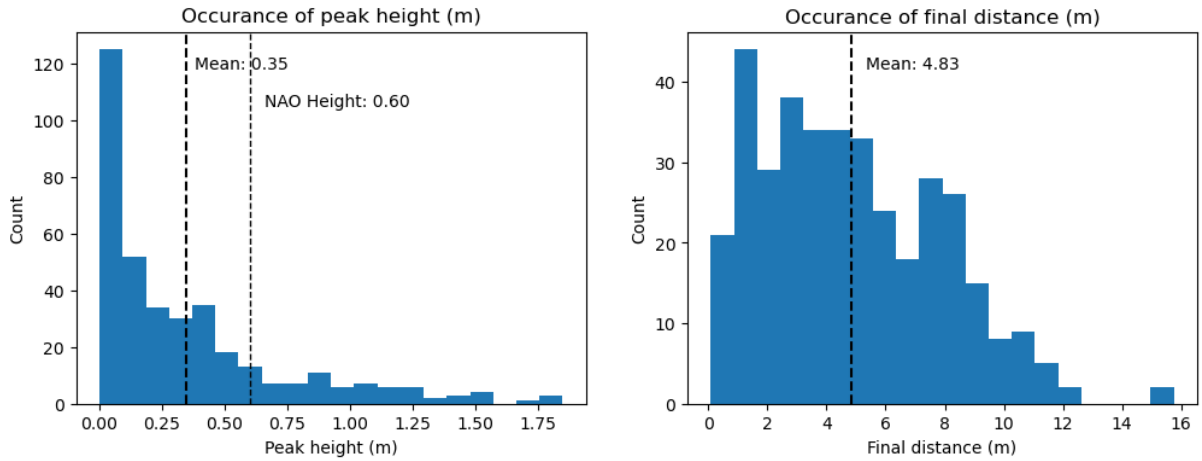

Figure 7.1: Tracking dataset statistics

The ground truth ball trajectories are plotted together in figure 7.2 – which show a spectrum of different kicks which have been captured. Two peculiar behaviors of the simulator's handling of the ball are noticed in the tracking data - a cherry picked example is plotted in figure 7.3. Firstly, it is noted that at times the ball passes slightly through the ground of the pitch when it bounces. This is verified by comparing the rendered images to the logging and indeed seems to be the case although it is not understood why the simulation has this behavior – since it is run with synchronized execution. Secondly, the initial ball trajectory has a "kink" resulting from a slight non-physical change in position – which is caused by the server that does not execute a true

kick, but rather instantaneously places the ball by the feet of the agent and then applies a velocity rather than performing a realistic kick, described by magmaOffenburg [2023]. This behavior only affects a minority of time-steps and is therefore accepted.

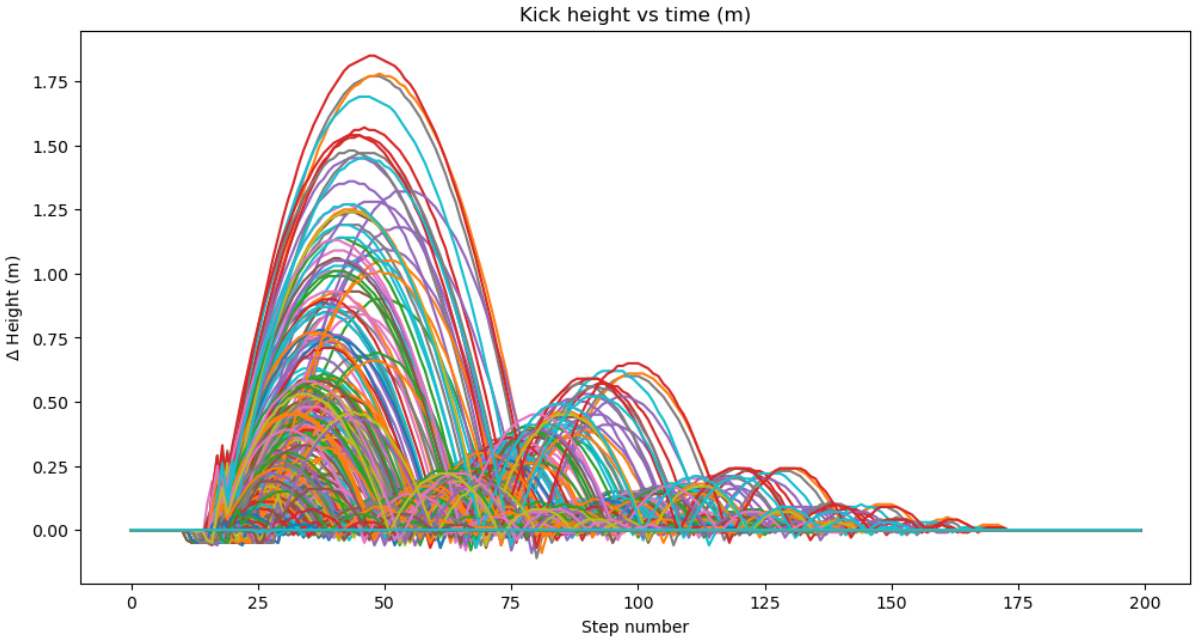

Figure 7.2: Tracking dataset ground-truths

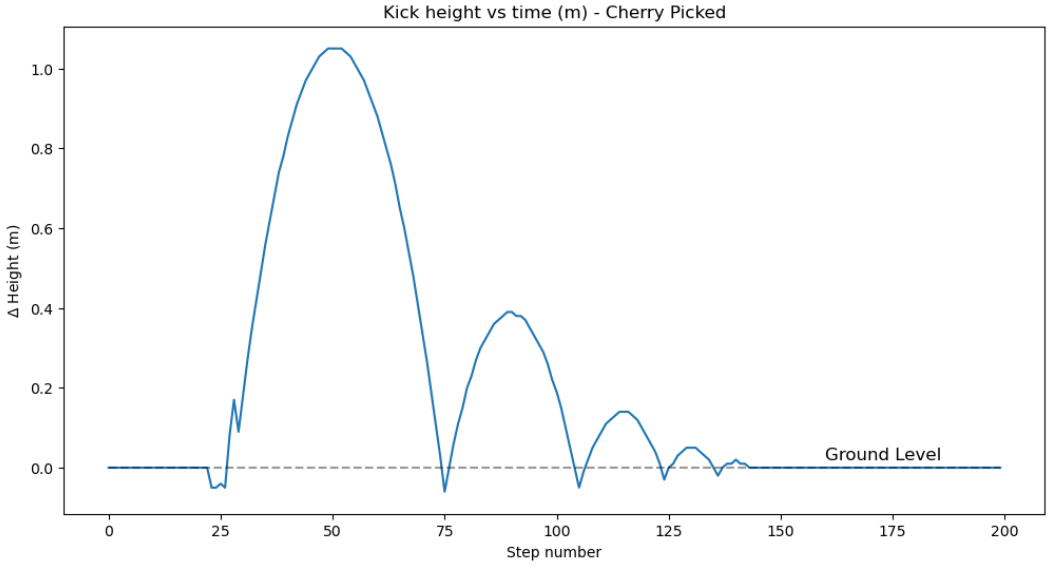

Figure 7.3: Tracking dataset - Cherry picked to show error

7.2 Baseline Tracking

The baseline trackers, which have been chosen based on traditional methods popular in RoboCup as seen identified in the related works, are now investigated.

7.2.1 Basic tracking

Initially, the raw detections of the tracking dataset are to be studied alongside the ground truth ball position as well as the ball position inferred by a perfect object detector. To do this, the ball detections are forward transformed to field ball positions using the same transformation as is used by the Wits team, which has been extracted and captured in the metadata. A simple latch is then used for the most basic tracking, which means that missed and intermediate steps are filled by the previous positions.

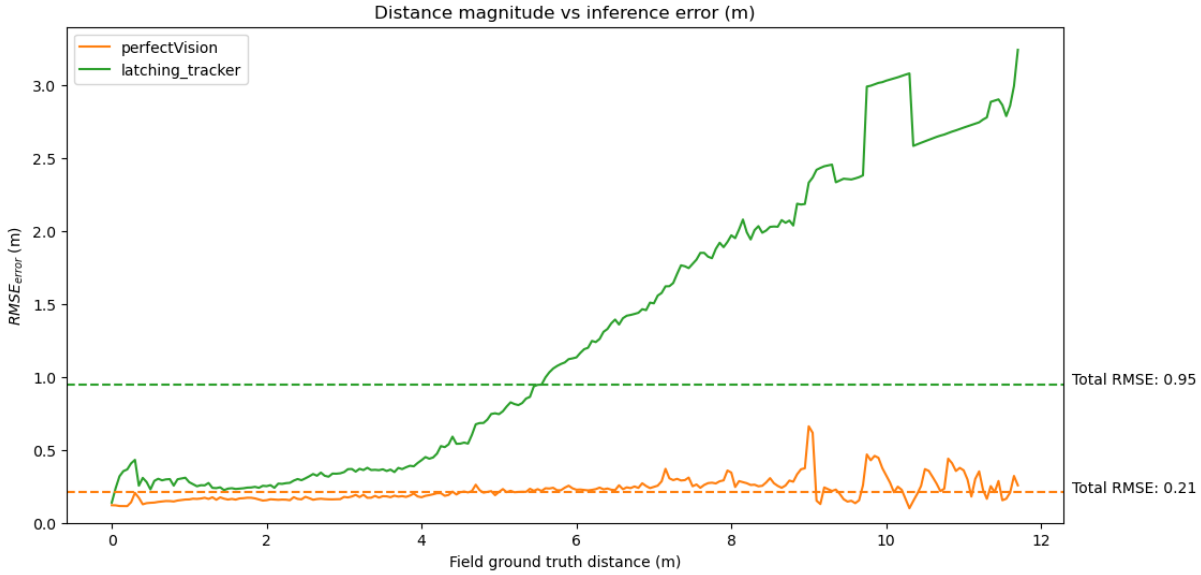

Figure 7.4: Basic tracking error

In figure 7.4, after forward transforming the bounding box to the field coordinates using the agent's world model, the root mean square error is computed over binned kick distance. It can be seen from the error in the "perfect vision" that a "noise floor" exists due to the presence of inaccuracies in the forward transformation of the agent. A reason for this is that a simplified representation of the the agent model is presently used in the code base of the team which does not compensate for offsets in the angular uprightness of the agent, which have generally been seen as providing an acceptable level of error compared to the complexity of a more accurate computation. Since the Fatproxy agent walks on the spot when not commanded, the agent is constantly in motion (bobbing side to side) and error is continuously introduced.

Near zero distance (before the Fatproxy kick causes the ball to beam) the position error is dominated by the error in the transformation. This shows that the detector that has been trained is as effective as perfect vision up until the time of kicking. Up until approximately 4m distance from the agent, a portion of error can be assigned

to both the ball detector as well as the transformation model. Here the system works comparably to the performance suggested in the related works (tracking error of 0.3m) – however the system has room for improvement although the improvement could equally be achieved by improving the transformation. However, beyond this point, it is evident that the error due to the detector explodes. Individual trajectories are investigated further in order to better understand this.

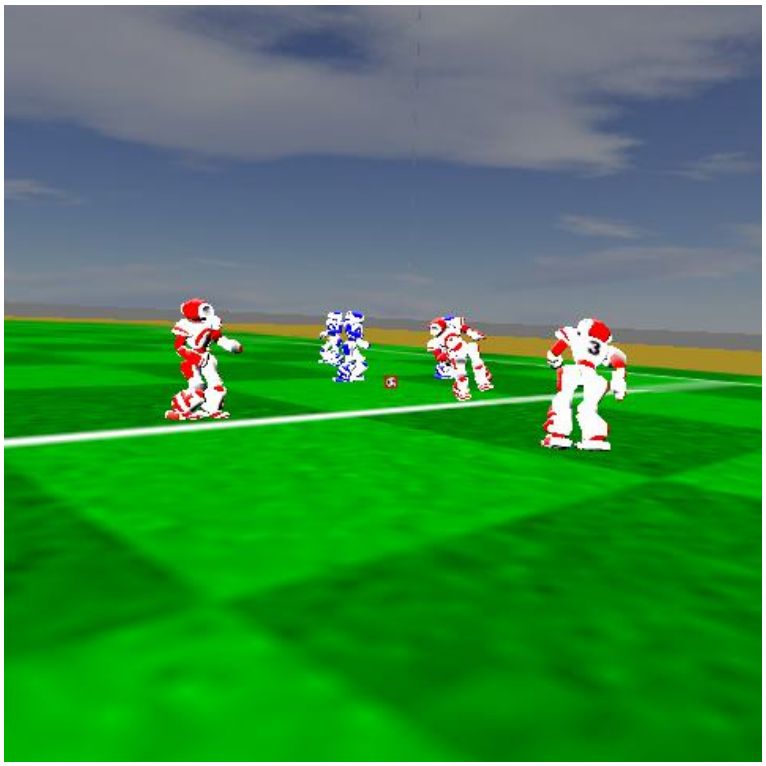

Figure 7.5: A ball at 5m distance

A short and long ground based kick are analyzed respectively in figures 7.6 and 7.7:

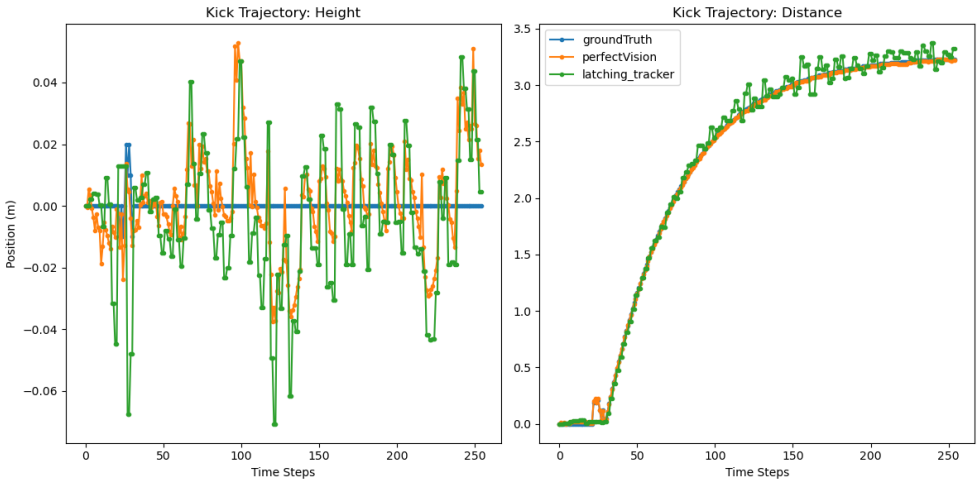

Figure 7.6: Short ground kick – Raw detections

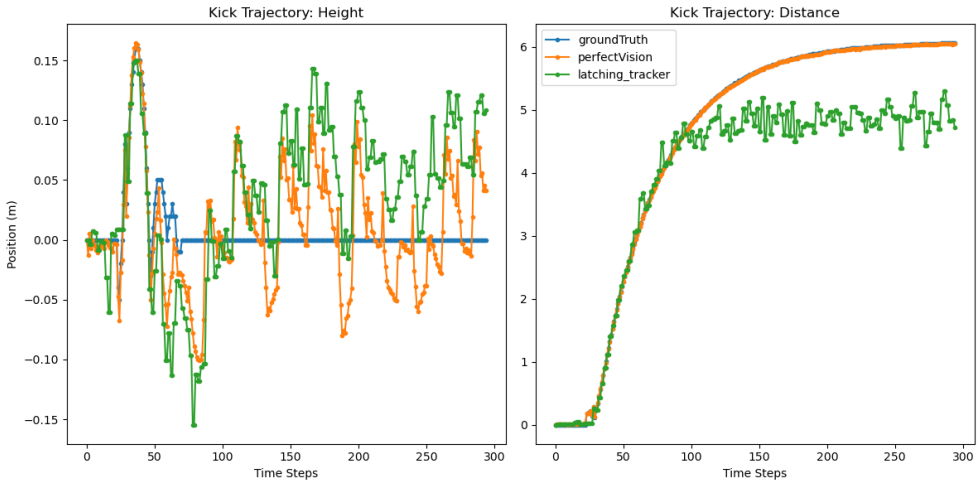

Figure 7.7: Long ground kick – Raw detections

In these kicks the following is noted. Firstly, considering the ground truth trajectory for a kick along the ground, the height of the ball remains zero as expected. The distance covered by the ball per unit of time decreases in a non-linear manner after the kick takes place, due to rolling friction. Secondly, considering the measurements of the trajectory, there is noise introduced by the detector and transformation which can be observed in the time domain where it is apparent that the cyclic nature of the agent motion results in error in the perceived ball height. The correctness of the height has a dependency on the placement of the bounding box, which appears to be comparable to the perfect object detector. Until 4m the inference of distance appears to be effective, although noisy. Beyond this distance, inference of the ball position appears to be problematic. Since the accuracy of the distance is dependent on the size of the bounding box, it seems that a limit exists. The object detector is impressive within its reliable range.

A short and long lofted kick are analyzed respectively in figures 7.8 and 7.9:

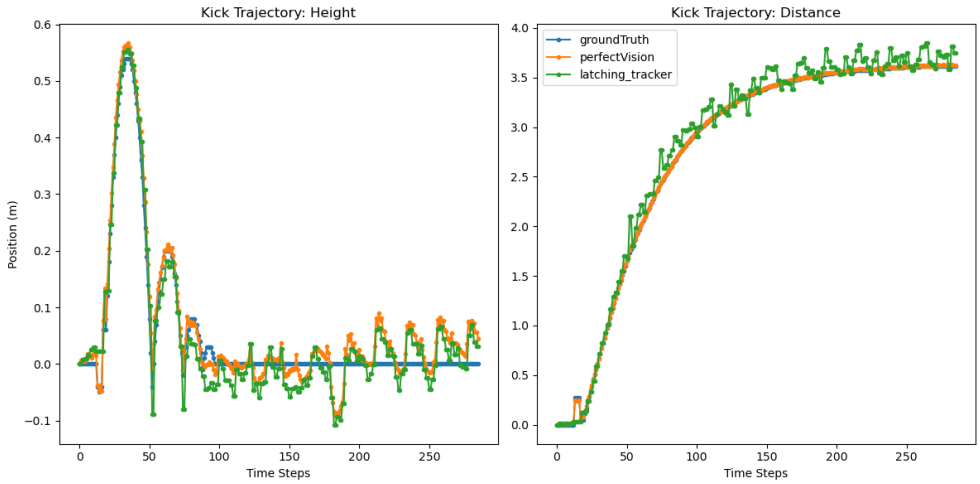

Figure 7.8: Short lofted kick – Raw detections

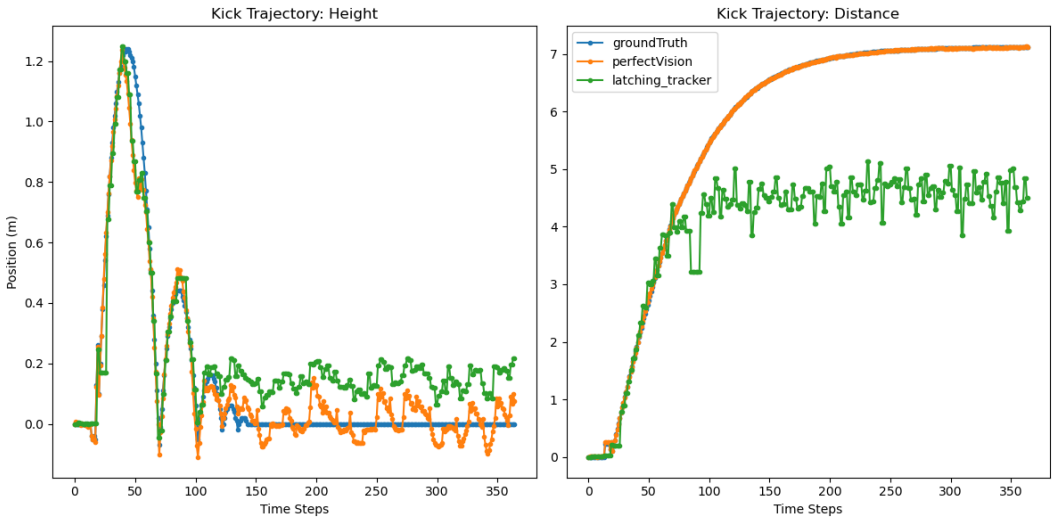

Figure 7.9: Long lofted kick – Raw detections

For lofted kicks, it is firstly noted that the trajectory follows a parabolic path as expected for projectile motion. With each bounce, energy losses cause rebounds of diminishing height. However, the distance follows a similar pattern to a kick along the ground. Secondly it is noted that within 4m the detector is comparable to perfect vision. Beyond 4m a limit on the ball distance is reached which causes the error to explode.

The distance measurement at long range is the limiting factor for the overall performance of the system – in figure 7.5 it is understandably so since the ball becomes tiny, therefore any changes in distance are difficult to perceive. However, early detections are reliable, therefore applying a tracking algorithm has the potential to improve the tracking of the ball trajectory – which is now investigated.

7.2.2 Standard Kalman filtering

Initially, a simplistic approach is taken by assuming the process (the ground motions of the ball) can be described by Newton's equations of motion – as a constant velocity model in equation 7.1 below. This dynamics model can then be expressed as a linear set of recursive equations as follows in equation 7.2:

$$x_k = v\Delta t + x_{k-1} \tag{7.1}$$

$$\boldsymbol{x}_k = F(t)\boldsymbol{x}_{k-1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \boldsymbol{x}_{k-1} \tag{7.2}$$

Applying the constant velocity model in the prediction step of a Kalman filter results in what is known as a "first order" Kalman filter, where a zero order filter assumes a constant position. The measurement noise R must be assumed to be of Gaussian nature and is quantified by comparing it to the error of all the measurements taken to the respective ground truths. The process noise model Q assumes zero-mean white noise of a Gaussian nature, originating as fluctuations in the velocity. A continuous

velocity noise model Q_c (equation 7.3) can be discretized for the desired time step and propagated through the system dynamics model F using equation 7.4 [Labbe 2022].

$$Q_c = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \Phi \tag{7.3}$$

$$Q = \int_0^{\Delta t} F(t) Q_c F^T(t) dt = \begin{bmatrix} \Delta t^3/3 & \Delta t^2/2 \\ \Delta t^2/2 & \Delta t \end{bmatrix} \Phi \tag{7.4}$$

Φ is the spectral density of the white noise – which is often difficult to compute or measure, especially when the process model differs from reality and is therefore often applied as an engineering factor or found by tuning. In order to find an appropriate Φ for the ball tracking task, a search is used. The results are shown in figures 7.10 and 7.11.

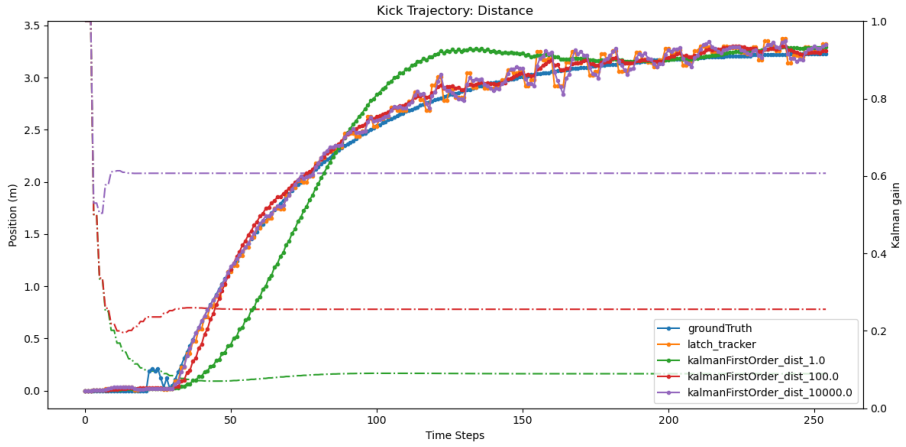

Figure 7.10: Short ground kick – First order Kalman filter search

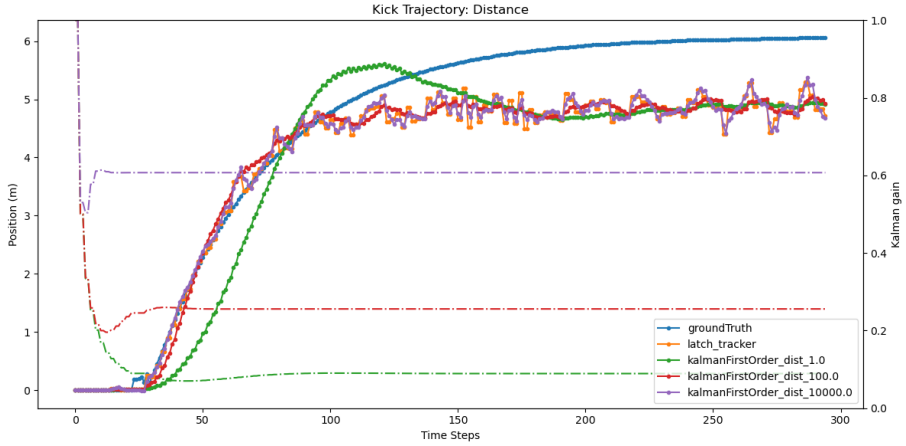

Figure 7.11: Long ground kick – First order Kalman filter search

By inspecting a set of kicks, one can get an understanding of the characteristics of the first order Kalman tracker. Initially, at the start of each kick, the model uncertainty is

initialized high which results in a Kalman gain close to 1. This corresponds to a strong dependency on the sensor measurement (rather than the prediction model) which helps to obtain a better initial state estimation. This makes sense as initially there is an assumption of no knowledge of the object states. As time steps are performed the Kalman gain reduces and the model relies less on the measurement and more on the dynamics model, which can result in noise attenuation. The rate of convergence is an intrinsic property of the model – it is not "intelligent" and relies on timing – and in this case convergence happens before the ball is kicked which means that the constant velocity assumption causes a significant lag when it dominates the estimation.

The ultimate steady state of the Kalman gain depends on the balance of the magnitudes of the measurement noise and process uncertainty. When the process is very uncertain (high Φ) then the filter tends to ignore the dynamics model and a state estimation with a negligible difference from a no filtering approach is achieved. As Φ is reduced, the dynamics model plays a larger role. With prior knowledge that the object is moving at constant velocity, it is possible to attenuate noise. However, if the dynamics model is overly dominant it causes a lag in the cases where the ball does not behave as is modeled by the function. This can lead to significant error, much worse than if no filtering is applied. There exists a trade off between responsiveness and noise attenuation.

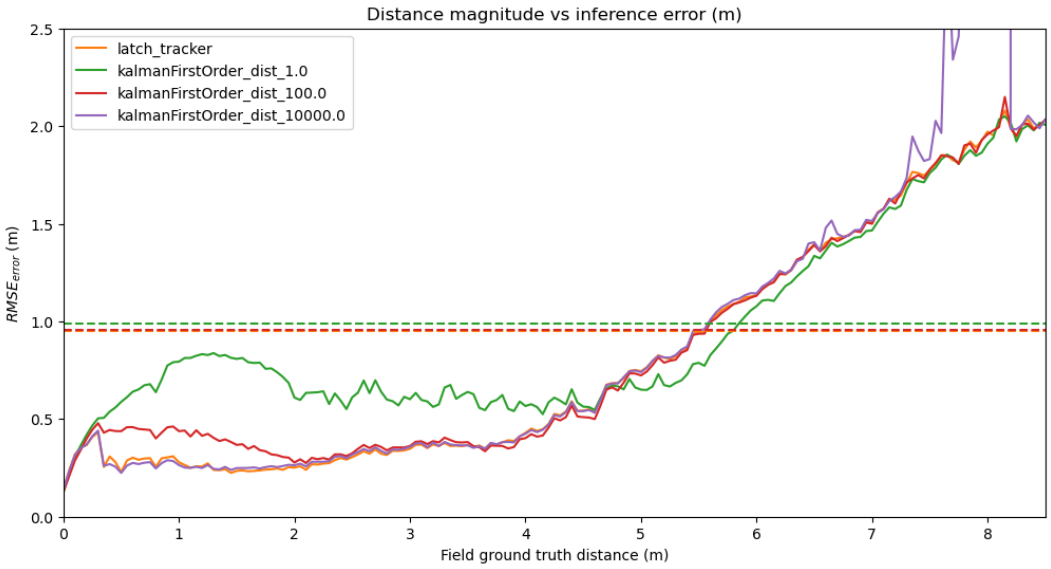

Figure 7.12: First order Kalman filter error

The tracking error has been plotted in figure 7.12. Although the filter can be tuned in order to achieve some noise attenuation, the Kalman filter assumes Gaussian noise with a zero mean and therefore is not able to capture the heavily non-linear characteristics of the distance measurement. Therefore, the standard Kalman filter is not able to learn and compensate for the large non-linearity which in this case dominates the error significantly more than the position jitter. It can be seen in figure 7.12 that for such a complex case, the application of a standard first order Kalman filter, across a variety of Φ, is either detrimental or causes negligible improvement to the performance compared to no tracking being used.

7.2.3 Extended Kalman filtering

A more sophisticated model describing the ground motion of the ball, which has been successful in the RoboCup domain, is described by Veloso *et al.* [1998] where it was first applied to their team (CMUnited) to win the RoboCup-97 small-robot competition. As per Veloso *et al.* [1998], the rolling resistance of a ball can be expressed by a constant discounting factor λ which is applied each time step, resulting in a reducing velocity. In this model the unknown factor is defined as a state variable. This dynamics model can then be expressed as a non-linear set of recursive equations as follows in equation 7.5 – which can be applied in an Extended Kalman filter:

$$\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k \cdot \Delta t \\ \dot{x}_k \cdot \lambda_k \\ \lambda_k \end{bmatrix} \tag{7.5}$$

The same measurement noise R is applied, as for the first order Kalman filter, as is the same process noise model Q. Again Φ is to be found by doing a search (figures 7.13 and 7.14).

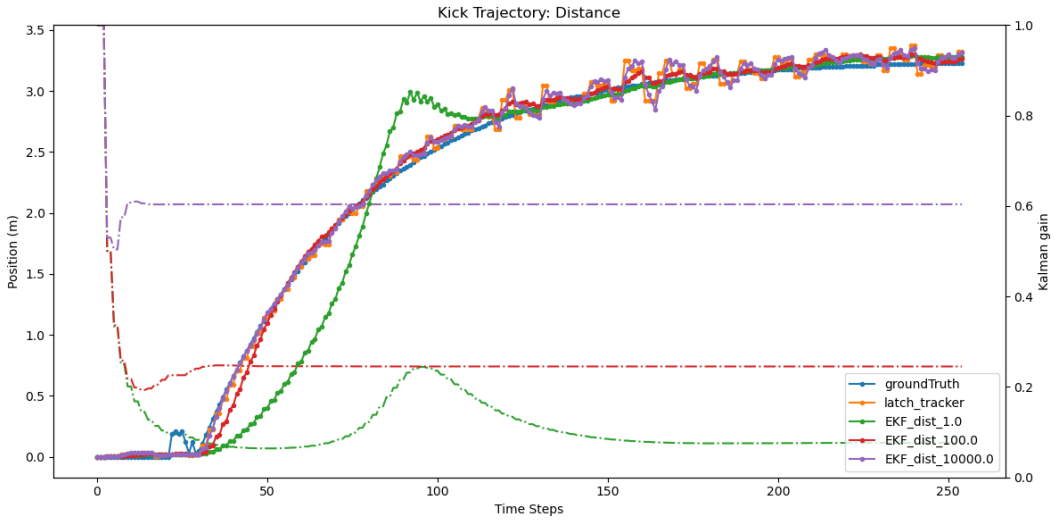

Figure 7.13: Short ground kick – Extended Kalman Filter

The EKF operates in a similar fashion to the standard Kalman filter. Initially the Kalman gain is set close to 1 and it converges (unintelligently) towards a steady state which is dependent on the balance between measurement and process noise. In the case of an EKF however there is some instability, due to the nonlinearity of the process model, which causes the Kalman gain to oscillate in some cases. Compared to the first order Kalman filter, it is seen after the initial kick that the tracker has a reduction in overshoot. The model is therefore more responsive and can in principle be more aggressively tuned for noise rejection.

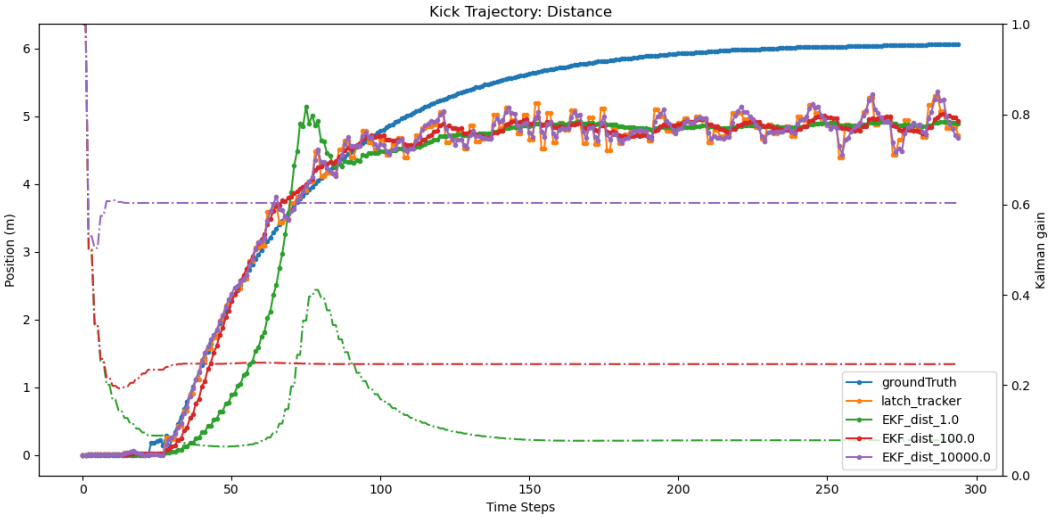

Figure 7.14: Long ground kick – Extended Kalman Filter

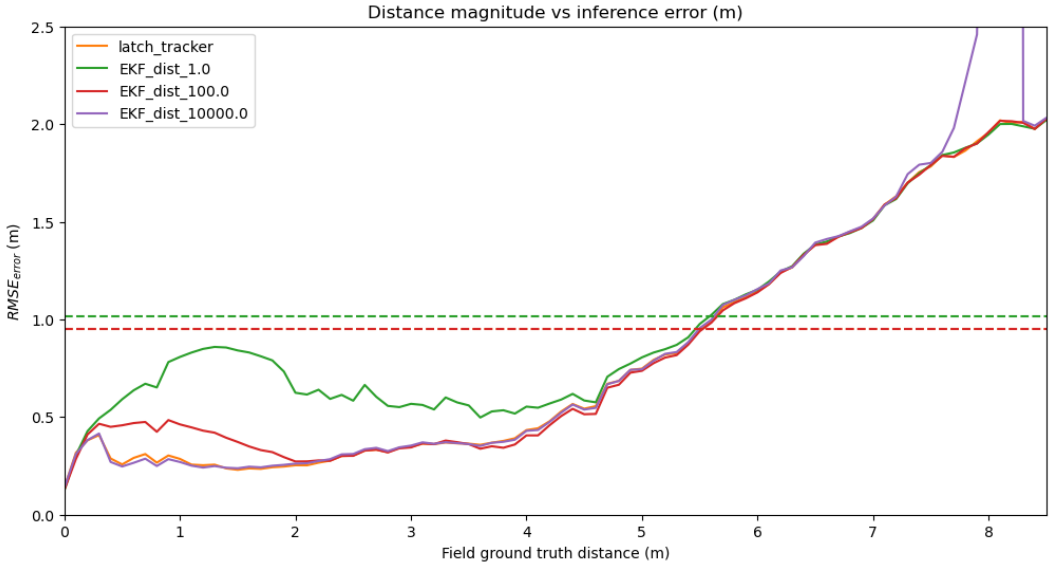

Figure 7.15: Extended Kalman Filter error

Although a more sophisticated approach, in figure 7.15 the EKF still suffers from some of the same simplistic assumptions as used for the first order Kalman filter. The EKF is not an optimal tracker for this case and therefore is either detrimental or causes negligible improvement to the performance compared to no tracking being used.

7.3 Proposed Tracking

For the tracking cases considered it is apparent that the performance is sub optimal. Neural Kalman filtering is an approach to Kalman filtering that can be used when the underlying state space model is non-linear and accurate knowledge of it is unknown. In this section the application of neural Kalman filtering to the tracking problem is investigated.

7.3.1 Modified KalmanNet

Initially the KalmanNet Github repository KalmanNet [2023] is cloned and it is attempted to reproduce their training results and then begin training for the ball tracking problem. Various issues were encountered which made training impractical therefore some changes were applied. The targeted major issues were: frequent cases of overflow in training following initialization (models that diverge immediately), impractically slow convergence (across learning rates) as well as high ultimate loss scores. Therefore the useful basic structure of the KalmanNet repository and training routine is utilized but some changes that significantly improved performance for this research have been implemented to achieve practical neural Kalman filtering.

Although some of the KalmanNet architecture is peculiar, some of the choices are not necessarily improper but they simply seem to just do not work well for the research problem being considered here. They may still have useful results in other domains:

- In figure 2.3 it is shown that the predicted Kalman gain results from a fully connected linear output layer. Although it's not necessarily an invalid approach, it does however not conform to the essence of Kalman filtering where the Kalman gain must lie between 0 and 1 – which has an intuition of describing a level of confidence in the measurements. Therefore it is chosen to replace this layer with a softmax layer, which has the effect of clamping the range. It was found that this change resulted in a stable model with a significantly increased rate of convergence.
- In both the Revach *et al.* [2022] paper as well as in figure 2.2, the data preprocessing step is not fully described. It is found that the input features are scaled by the magnitude of the state vector. This introduces significant non-linearity to the problem since with this processing, for example, at zero velocity the model would have the same input feature (state estimate and measurement) at a position of 0m as it would as a position of 1000m. This has the effect of destroying necessary information for this research problem being considered. A minimalistic approach was applied by simply applying an identity transformation to the input features and relying on the linear input layer with applied bias – which was found to give sufficient results for this dataset without relying on prior knowledge – and notably boosted the ultimate performance of the model by achieving a lower loss than using the existing feature. Empirically this gave better training results compared to other preprocessing approaches taken such as standardization and normalization.

7.3.2 Model training

In order to use the KalmanNet model training routine, a data loader has been constructed which places the training data in the expected tensor format. The duration of each series should be uniform, which is achieved here by clipping. The GRU is reset only per epoch, which aims to prevent overfitting on the moment of starting the kick as well as reducing the importance of timing initialization. To train the network, the mean square error is minimized with cross validation applied to store the best model weights found.

Compared to the CNN models trained previously, significantly slower processing of the data was realized – which is most likely due to the sequential nature of the problem and data. A speed up was achieved (more than an order of magnitude over CPU) by training utilizing NVIDIA GeForce RTX 3090 processors, which still resulted in a training time of nearly 12 hours per model for 1000 Epochs. Training results were found to be highly variable, at times converging on significantly different results which can be explained by variations in initialisation. By training numerous models with the same parameters, it was possible to select the best one and therefore it is believed that more training would not result in significantly better results. The results of two trained models (best and cherry picked) are plotted in figure 7.16 in order to convey the variety in training seen, where one might wrongly have considered convergence to have been achieved.

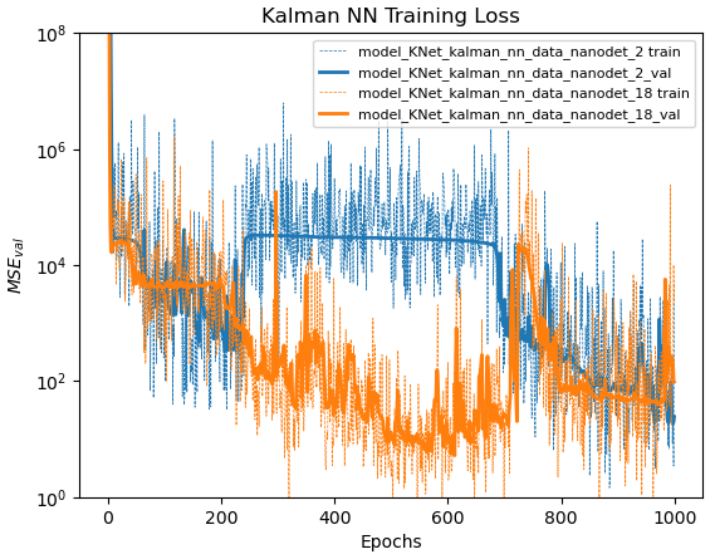

Figure 7.16: Neural network aided Kalman filtering model training

Considering figure 7.16 – at initialization the model scores extremely poorly, however, rapidly converges on a lower loss score against both the training and cross validation score. This is a good indication that the model is learning and capturing patterns within the data. The fact that both training and validation scores are indiscernibly close provides confidence that the model is not over-fitting on the training data. Between the two models shown, further epochs lead to either further convergence or even divergence depending on the learning of the model. The best model achieves its performance after 612 epochs with further training resulting in diminishing performance.

7.3.3 Evaluation of results

The trackers that have been considered thus far are now evaluated against the unseen test training set. In order to have a feel for the results – a representative short and long grounded kick are interrogated in figures 7.17 and 7.18. In particular the focus is on understanding the performance of the neural aided tracking method as well as how it compares to the traditional approaches.

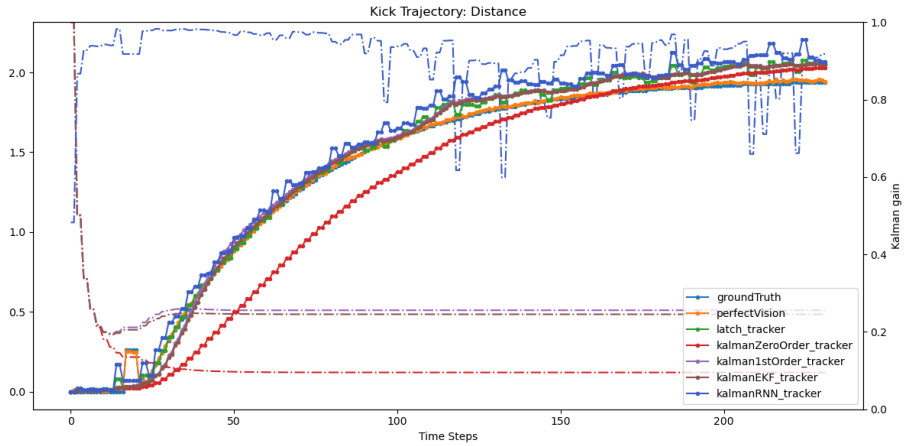

Figure 7.17: Short ground kick – Tracker evaluation

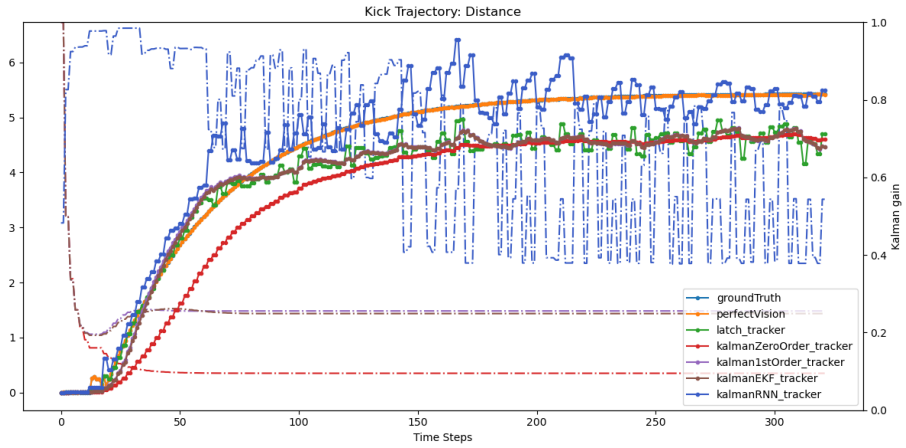

Figure 7.18: Long ground kick – Tracker evaluation

Initially, at the start of the kick routine, the neural network predicts Kalman gains close to 1 – which is a desirable initialization behavior. This is also shared and considered to be a highly effective feature of the standard Kalman and EKF trackers. This also corresponds to the knowledge that the early measurements are reliable and shows that meaningful learning has occurred. Compared to the standard and EKF trackers however, the neural assisted method does not show an initial convergence of the gain. Due to this it can be recognized that when the kick is performed the neural Kalman filter does not show any lag in the same manner as the traditional methods do, which have already converged. Since the traditional trackers have a portion of a mismatching physics model as part of the update, these models will always experience a lag.

At close range, following kick execution, the Kalman gain remains consistent but is not static. In this manner the state estimation benefits from accurate measurements and retains responsiveness. As the ball travels further in distance within the region of reliable detection the rolling average Kalman gain drops only slightly. With a gain near 1, the neural Kalman filter tracks the trajectory similarly (but slightly ahead of) the latch tracker (a reflection of the raw detections) rather than the ground truth. An explanation for a lack of learning here is likely that the loss resulting from such a case is very small compared to the loss from range limited detections. This may also explain why there seems to be no noise attenuation – the error reduction is imperceptible.

In figure 7.18 the impacts of the ball moving into the longer range region is seen. It is apparent that as the ball moves into the region where the detection becomes range limited that the Kalman gain predictions start to become interesting. A sudden onset of high frequency jumps in the Kalman gain appears, which has the effect of increasing the state estimate beyond the measured distance. The result of this is that the state estimate diverges from the measurements. The mechanism for this appears to be that certain measurements are "extrapolated" upwards by the first order Kalman filter by dropping the gain. A high Kalman gain then corrects for an overshoot by using the measurement to drop the state estimation. In this manner, the amplitude of the fluctuations might be understood as a measure of uncertainty. A side effect of this is that it results in an amplification of the jitter resulting from the measurement noise.

As more time steps are elapsed and velocity of the ball starts to reduce the rolling average Kalman gain drops further – this places a greater emphasis on the dynamics model. The effect of this is that despite more range limited measurements arriving, the state estimations are able to track the ground truth rather than converging on the measurements as with the traditional tracking methods. A complex behavior appears been learned where the tracker has been able to capture the trajectory of the ball despite the range limited measurements. In this manner it does appear that the network has indeed implicitly learned to capture the underlying system dynamics.

The error of the trackers against the unseen test data is plotted in figure 7.19:

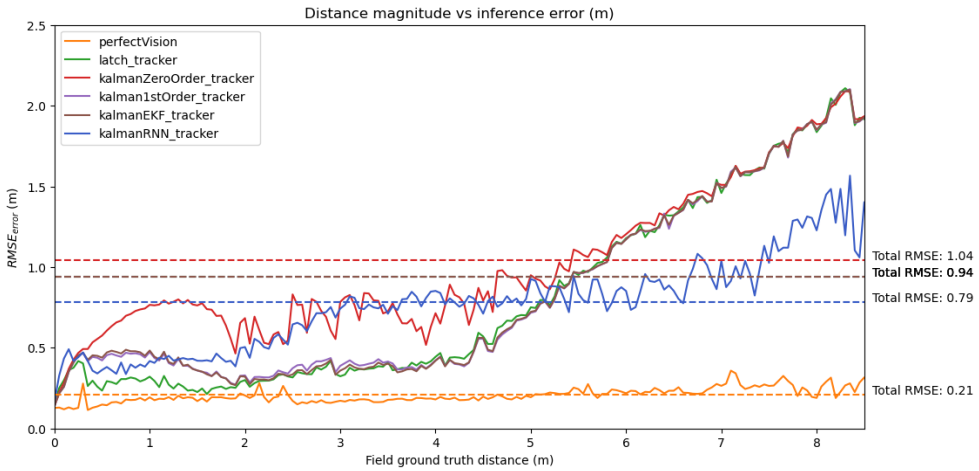

Figure 7.19: Tracker error (Test set)

Overall it can be seen that of all the trackers considered, neural Kalman filtering has achieved the lowest root mean square error against the test dataset. These results prove that neural Kalman filtering can outperform the traditional EKF method at tracking the trajectories of a kicked RoboCup soccer ball as measured by RMSE, using a tracking-by-detection framework. The majority of the performance gain arises from improved tracking from the long kicks which travel beyond the range where the object detector is reliable. In this region the neural network has implicitly captured the underlying system dynamics which facilitates divergence of the state estimation from the measurements accordingly. Neural Kalman filtering is also the most responsive tracking approach, having a reduced error in the region up to 1m after the kick due to a reduction in latency when the process diverges from the dynamics model.

With the model that has been utilized there are however aspects that come at a cost of the previously mentioned performance. In the region where the detections are accurate, the traditional approaches track better. A possible explanation is that the loss scores due to this error are small in comparison to the error from the range limited measurements. Therefore, during training, the network does not learn to optimize Kalman gain predictions in the middle range regions. Another cost is noise amplification, which seems to manifest due to the fluctuations in the Kalman gain predictions as the network has some level of uncertainty.

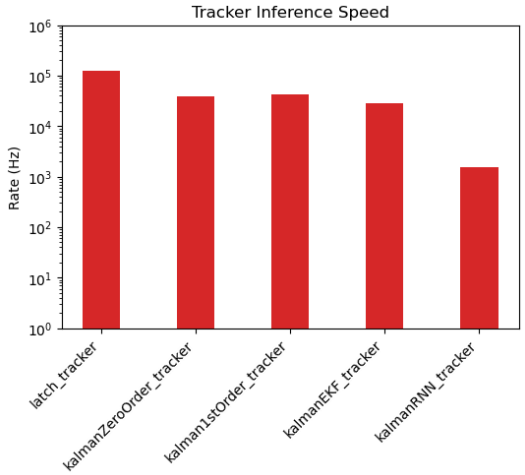

Figure 7.20: Tracker speeds

In order to evaluate the real-time performance of the tracking algorithms, a benchmarking of the trackers is performed and plotted in figure 7.20. All algorithms are have been implemented in Python and tested on a mobile laptop CPU as done with the evaluation of object detectors. In figure 7.20 it is seen that the simple latch tracker is the best performing approach in terms of computational speed. At such frequencies, the measurement error is likely significant however is useful for relative comparison. The standard Kalman filter and EKF tracker have a diminished speed respectively due to increased complexity. Neural network aided Kalman filtering is two orders of magnitude slower than the latch but still completes multiple orders of magnitude faster than the object detectors. Therefore the tracking step is not limiting and completes in real time despite the addition of further complexity.

Chapter 8

Conclusion

The aim of visual object tracking is to use a sequence of image frames to robustly estimate the motion state of a target object. This offers a versatile approach that is capable of acquiring an abundance of environmental information that can be used by robotic agents to perceive and interact with a dynamic environment in real-time. In this research report, the context of ball tracking in a simulated real-time RoboCup soccer environment is considered. It has been proposed to take advantage of the progress of modern neural approaches to computer vision by utilizing models from the popular problem space of object detection by applying a tracking-by-detection framework. Correspondence was then achieved through the use of Kalman filtering which is a popular model based tracking approach commonly used for state-estimation in the robotics domain. It was proposed to apply neural assisted Kalman filtering, which balances the ease of interpretation of a model-based approach and the ability to learn complex dynamics using neural networks.

For the object detection task, initially a detection dataset needed to be crafted using the RoboCup simulation environment. Following this, the benchmark YOLO v4 Tiny and Viola-Jones detector models were investigated and trained on the training and validation sets. A naive benchmarking of modern object detectors was then performed where the selection of NanoDet was justified. Finally, the NanoDet model was also trained and then compared against the benchmark detectors using the test set. The findings of this work showed that it was indeed possible to execute established modern CNN object detector architectures in real-time on a mobile laptop CPU. It was also found that pretraining on real-images was transferable to the simulated RoboCup domain – with the pretrained object detectors, pretrained on COCO, managing to achieve useful scores and even outperforming trained VJ detection. The object detector hypothesis was proven which hypothesized that a more recent key-point based modern object detection approach (NanoDet) can be used to outperform established RoboCup approaches such as an anchor based method (Yolo v4 Tiny) as well as the traditional Viola-Jones detector for RoboCup soccer ball detection – measured by the quality of inference using AP scores, at a real-time speed. Further, it was seen that model quantization can indeed cause a noticeable performance degradation, with the reduction in model size not yielding a worthwhile return to justify its inclusion.

The object tracking task was then considered. Initially a tracking dataset needed to be generated using the RoboCup simulation environment. The results of the object detector task were then applied in a tracking pipeline. It was found that the detector was effective at detection but was range limited in distance, which introduces significant complexity for tracking the trajectory of a kicked ball. Following this, the benchmark linear and extended Kalman filtering trackers were then implemented and configured using the training and validation datasets. The KalmanNet model was then cloned and the architecture was adapted to suit before using the training and validation sets to find optimized model parameters. The traditional and modern trackers were then compared, where it was found that a neural network assisted approach to Kalman filtering can outperform the traditional EKF method at tracking the trajectories of a kicked RoboCup soccer ball as measured by RMSE at real-time, using a tracking-by-detection framework – therefore proving the object tracking hypothesis. Further, it was shown that all the trackers investigated can meet real-time performance requirements.

8.1 Further Work

In this subsection some potential areas for further investigation are discussed which follow the observations and results of the research.

8.1.1 Improving ball measurements

Although the performance of the object detectors in detecting a distant ball was more impressive than expected, there is room for improvement of the bounding box regression which appears to have a minimum size limit despite still being able to make successful ball classifications. A reason for this may be found in the architecture of the Path Aggregation Network, which leverages the pyramidal hierarchy of the network backbone in order to combine higher and lower level feature maps that represent semantic and spatial information accordingly as seen in figure 8.1.

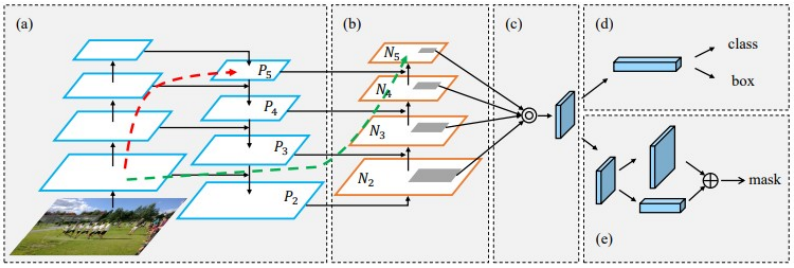

Figure 8.1: Path Aggregation Network [Liu *et al.* 2018]

The largest feature map that is laterally connected has undergone down-scaling due to the stride of the CNN backbone. Therefore, in order to improve the bounding box regression, a higher resolution feature map may boost regression by supplementing additional spatial information. In NanoDet a stride of 8 is used, however, reducing the stride is undesirable for real-time performance since – due to the arithmetic complexity of the CNN – in order to achieve a bottleneck of the same ultimate feature map size with

a reduced stride by factor a, an expansion of the number of computations proportional to a^3 would result which would impact the detector real-time performance. In the simplest approach one might resize the image by a factor b, however this creates b^2 more computations in the backbone. A potential approach that adds only a small overhead would be the addition of an additional up-sampling layer, with a lateral connection to the input layer.

Constrained to monocular vision, in order to further improve the distance measurement after having computed a bounding box it may be possible to use further clues such as optical methods or environmental information. One optical approach may be to utilize the phenomenon of motion parallax where an observation from two different points leads to an apparent displacement of an object. In this manner the natural sideways bobbing of the agent can give depth information – which is a mechanism that is based on observations in nature [Hataji *et al.* 2021]. Further environmental observations such as the position of the ball against field markings as well as the perceived gravitational drop of a ball, if it is a lofted kick, can give additional information in the correct circumstances.

8.1.2 Improving state estimation

Although neural Kalman filtering was able to outperform the traditional tracking methods as measured by the mean square error – there are some tracking issues that should be addressed.

One issue is that, within the reliable range of the object detector, the neural based approach underperforms versus the traditional methods of Kalman filtering. It is more desirable to have a tracker which is best across the range rather than specializing for certain circumstances. The current issue appears to be that the region where the detector is reliable does not contribute significantly to the loss score. Therefore, applying a focal loss inversely based on distance may encourage better performance in this region (possibly at a cost of the long range performance, which might be tuned using a balancing hyper-parameter). Alternatively, a loss based on integrated error (similar in nature to the integral gain of a PID controller) may help to reduce the steady state error which will also boost the performance in this region.

A further issue is the impact of jitter, which is undesirable for control purposes. Traditional Kalman filters are often used to provide some noise attenuation in a signal. The neural Kalman filter used in this research however has a tendency to add additional noise. One approach may be to punish volatility using a loss metric of sorts, however, this may have ill consequences since punishing rapid change is likely to result in a loss of responsiveness. The fluctuations also seemed to be related to an expression of uncertainty, punishing this may result in a degradation of performance. Therefore a better way may be to add a component dedicated to smoothing within the end-to-end model. This might be a traditional Kalman filter (or more sophisticated filter).

8.1.3 Extensions

Some additional further work is considered below:

An aspect that has not been considered in this research is the ability of the model to predict future time steps. This can be useful for agent positioning. Forward predictions from a Kalman filter are usually performed by skipping the correction step. In the case where the dynamics model is significantly different from the process, the prediction will diverge significantly from the real process which can be significantly more complex in the case of neural Kalman filtering. In order to get useful future predictions for ball tracking, replacing the embedded first order Kalman filter with an EKF model is an approach that will yield better predictions but should also improve the tracking accuracy as the model better represents the underlying dynamics. However, the additional complexity may add further complexity to training. An alternative approach would be to create a recurrent model to predict future measurements and to feed these outputs into the tracking model.

A limitation of the approach used in this research is that a pipeline between detection and tracking is utilized rather than end-to-end training. Due to this, the detection loss is not the same as the ultimate metric that is used to evaluate the tracker. A benefit of this approach is that there is modularity in the system – a newly trained object detector model could be substituted without requiring retraining of the tracker. Further, other visual systems (such as for agent localization) can plug in to use the results without further complicating training. A consequence of this is that the detector is not actually optimized specifically for the tracking task, so there is potentially additional performance that can be gained here. Therefore, an approach might be taken to create an architecture to solve the problem in an end-to-end manner, utilizing a common backbone with different heads for each of the desired tasks, would could result in better performances but at a cost of complexity and further loss of intuition into the workings of the system.

The natural continuation of this work would be to apply the detection and tracking to a physical robotic system in a real world setting. For detections it is interesting to investigate how well the detectors generalize to the natural world with various physical sources of optical noise as well as scene variation in both the foreground (different balls) and the background. For tracking, there is significantly more variation possible especially on a natural pitch where wind, spin and an imperfect surface can result in changes to the ball trajectory. Transfer learning might be investigated in order to reduce the effort required to create a sufficiently descriptive dataset.

References

[Abadi *et al.* 2016] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: A system for large-scale machine learning*, 2016.

[Aidouni 2019] Manal El Aidouni. *Evaluating Object Detection Models: Guide to Performance Metrics*, 2019. `https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html`, Accessed: 2022-10-26.

[Aldeberan 2023] Aldeberan. *Aldeberan Documentation*, 2023. `http://doc.aldebaran.com/2-1/family/robots/motherboard_robot.html`, Accessed: 24-01-2023.

[Arani *et al.* 2022] Elahe Arani, Shruthi Gowda, Ratnajit Mukherjee, Omar Magdy, Senthilkumar Kathiresan, and Bahram Zonooz. A comprehensive study of real-time object detection networks across multiple domains: A survey. *Transactions on Machine Learning Research*, 2022.

[Bewley *et al.* 2016] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468. IEEE, sep 2016.

[Birbach *et al.* 2009] Oliver Birbach, Jörg Kurlbaum, Tim Laue, and Udo Frese. Tracking of ball trajectories with a free moving camera-inertial sensor. In *RoboCup 2008: Robot Soccer World Cup XII*, volume 5399, pages 49–60, 06 2009.

[Bochkovskiy 2020a] Alexey Bochkovskiy. *Yolo v4, v3 and v2 for Windows and Linux*, 2020. `https://github.com/AlexeyAB/darknet`, Accessed: 04-04-2023.

[Bochkovskiy 2020b] Alexey Bochkovskiy. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv preprint arXiv:2004.10934, 2020.

[Boedecker and Asada 2008] J. Boedecker and M. Asada. Simspark – concepts and application in the robocup 3d soccer simulation league. In *Autonomous Robots*, Jan. 2008.

[Bolme *et al.* 2010] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Com-*

puter Society Conference on Computer Vision and Pattern Recognition, pages 2544–2550, 2010.

[Carranza-García *et al.* 2021] Manuel Carranza-García, Jesús Torres-Mateo, Pedro Lara-Benítez, and Jorge García-Gutiérrez. On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, 13(1), 2021.

[Choi *et al.* 2022] Geon Choi, Jeonghun Park, Nir Shlezinger, Yonina C. Eldar, and Namyoon Lee. *Split-KalmanNet: A Robust Model-Based Deep Learning Approach for SLAM*. arXiv preprint arXiv:2210.09636, 2022.

[COCO 2023] Detection Evaluation MS COCO. *Detection evaluation*, 2023. `https://cocodataset.org/#detection-eval`, Accessed: 24-01-2023.

[Dalal and Triggs 2005] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:886–893 vol. 1, 2005.

[Devernay and Faugeras 2001] Frédéric Devernay and Olivier Faugeras. Straight lines have to be straight automatic calibration and removal of distortion from scenes of structured environments. *Mach Vis Appl*, 13, 08 2001.

[Dollar *et al.* 2009] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 0:304–311, 06 2009.

[Everingham *et al.* 2010] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 06 2010.

[Fehlner 2022] Andreas Fehlner. The open standard for machine learning interoperability. In *Workshop: Conceptual Advances in Deep Learning for Research on Universe and Matter*, 2022.

[Fernandes *et al.* 2021] Roberto Fernandes, Walber M. Rodrigues, and Edna Barros. *Dataset and Benchmarking of Real-Time Embedded Object Detection for RoboCup SSL*. arXiv preprint arXiv:2106.14597, 2021.

[format 2023] COCO: Data format. *Data format*, 2023. `https://cocodataset.org/#format-data`, Accessed: 24-01-2023.

[Ge *et al.* 2021a] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. *YOLOX: Exceeding YOLO Series in 2021*. arXiv preprint arXiv:2107.08430, 2021.

[Ge *et al.* 2021b] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. *YOLOX repository*, 2021. `https://github.com/Megvii-BaseDetection/YOLOX.git`, Accessed: 04-04-2023.

[Gholami *et al.* 2021] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. *A Survey of Quantization Methods for Efficient Neural Network Inference*. arXiv preprint 2103.13630, 2021.

[Girshick *et al.* 2014] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2014.

[Hataji *et al.* 2021] Yuya Hataji, Hika Kuroshima, and Kazuo Fujita. Motion parallax via head movements modulates visuo-motor control in pigeons. *Journal of Experimental Biology*, 224(3):jeb236547, 02 2021.

[Havlík and Straka 2015] Jindřich Havlík and Ondřej Straka. Performance evaluation of iterated extended kalman filter with variable step-length. In *Journal of Physics: Conference Series*, volume 659, 11 2015.

[Huang and Rathod 2017] J. Huang and V. Rathod. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, Apr. 2017.

[Ingram *et al.* 2021] B. Ingram, A. Bingani, P. Ranchod, and B. Rosman. Wits fc team description paper. In *Robocup.org*, 2021.

[Jia *et al.* 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. *Caffe: Convolutional Architecture for Fast Feature Embedding*, 2014.

[Kálmán 1960] Rudolf E. Kálmán. A new approach to linear filtering and prediction problems" transaction of the asme journal of basic. 1960.

[KalmanNet 2023] KalmanNet. *KalmanNet TSP Repository*, 2023. `https://github.com/KalmanNet/KalmanNet_TSP`, Accessed: 14-02-2023".

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[Kuhn 1955] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52, 1955.

[Labbe 2022] Roger R. Labbe. *Kalman-and-Bayesian-Filters-in-Python*, 2022. `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/tree/master`, Accessed: 12-08-2023.

[Law and Deng 2018] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. *International Journal of Computer Vision*, 128, 2018.

[Li *et al.* 2013] Xi Li, Weiming Hu, Chunhua Shen, Zhongfei Zhang, Anthony R. Dick, and Anton van den Hengel. A survey of appearance models in visual object tracking. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4:1 – 48, 2013.

[Li *et al.* 2017] Peixia Li, Dong Wang, Lijun Wang, and Huchuan Lu. Deep visual tracking: Review and experimental comparison. *Pattern Recognition*, 76, 11 2017.

[Li *et al.* 2020] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. *Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection*. arXiv preprint arXiv:2006.04388, 2020.

[Lin *et al.* 2014] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Zitnick. Microsoft coco: Common objects in context. 05 2014.

[Lin *et al.* 2017] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*, 2017.

[Lin *et al.* 2018] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*, 2018.

[Liu *et al.* 2018] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[magmaOffenburg 2023] magmaOffenburg. *magmaFatProxy*, 2023. `https://github.com/magmaOffenburg/magmaFatProxy`, Accessed: 18-01-2023.

[Mahony *et al.* 2019] Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Adolfo Velasco-Hernández, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. *CoRR*, abs/1910.13796, 2019.

[Mitri *et al.* 2004] Sara Mitri, Kai Pervölz, Hartmut Surmann, and Andreas Nuchter. Fast color-independent ball detection for mobile robots. 01 2004.

[Müller *et al.* 2018] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Sim4cv: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision*, 126(9):902–919, mar 2018.

[ONNX 2023] ONNX. *Model Zoo*, 2023. `https://github.com/onnx/models`, Accessed: 04-04-2023.

[OpenCV 2023] OpenCV. *Cascade Classifier Training*, 2023. `https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html`, Accessed: 24-01-2023.

[Paszke *et al.* 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019.

[Pereira *et al.* 2022] Ricardo Pereira, Guilherme Carvalho, Luís Garrote, and Urbano J. Nunes. Sort and deep-sort based multi-object tracking for mobile robotics: Evaluation with new data association metrics. *Applied Sciences*, 12(3), 2022.

[Porikli and Yilmaz 2012] Fatih Porikli and Alper Yilmaz. Object detection and tracking. *Studies in Computational Intelligence*, 409, 01 2012.

[RangiLyu 2021] RangiLyu. *NanoDet-Plus: Super fast and high accuracy lightweight anchor-free object detection model.*, 2021. https://github.com/RangiLyu/nanodet, Accessed: 04-04-2023.

[Redmon and Farhadi 2016] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2016.

[Redmon and Farhadi 2018] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*, 2018.

[Redmon *et al.* 2015] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Autonomous Robots*, Jan. 2015.

[Revach *et al.* 2022] Guy Revach, Nir Shlezinger, Xiaoyong Ni, Adria Lopez Escoriza, Ruud J. G. van Sloun, and Yonina C. Eldar. KalmanNet: Neural network aided kalman filtering for partially known dynamics. *IEEE Transactions on Signal Processing*, 70:1532–1547, 2022.

[Rezatofighi *et al.* 2019] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*, 2019.

[RoboCup 2022] RoboCup. *RoboCup Objective*, 2022. https://www.robocup.org/objective/, Accessed: 25-06-2022.

[Roboflow 2023] Roboflow. *Roboflow*, 2023. https://roboflow.com/, Accessed: 04-04-2023.

[Salscheider 2021] Niels Ole Salscheider. *Object Tracking by Detection with Visual and Motion Cues*, 2021.

[Silva *et al.* 2012] Hugo Silva, A. Dias, J.M. Almeida, Alfredo Martins, and E. Silva. Real-time 3d ball trajectory estimation for robocup middle size league using a single camera. volume 7416, pages 586–597, 06 2012.

[SimSpark 2022] SimSpark. *Perceptors*, 2022. https://gitlab.com/robocup-sim/SimSpark/-/wikis/Perceptors, Accessed: 25-06-2022".

[Speck *et al.* 2016] Daniel Speck, Pablo Barros, Cornelius Weber, and Stefan Wermter. Ball localization for robocup soccer using convolutional neural networks. pages 19–30, 11 2016.

[Stackoverflow 2017] Stackoverflow. *Can numPos be a Negative Number in Haar Cascade Training?*, 2017. https://stackoverflow.com/questions/44337461/can-numpos-be-a-negative-number-in-haar-cascade-training, Accessed: 12-08-2023.

[Stamou *et al.* 2005] Georgios Stamou, Michail Krinidis, Evangelos Loutas, Nikos Nikolaidis, and Ioannis Pitas. 4.11 - 2d and 3d motion tracking in digital video. In AL BOVIK, editor, *Handbook of Image and Video Processing (Second Edition)*,

Communications, Networking and Multimedia, pages 491–XVIII. Academic Press, Burlington, second edition edition, 2005.

[Stefani *et al.* 2022] Domenico Stefani, Simone Peroni, and Luca Turchet. A comparison of deep learning inference engines for embedded real-time audio classification. 09 2022.

[Szemenyei and Estivill-Castro 2019] Marton Szemenyei and Vladimir Estivill-Castro. *ROBO: Robust, Fully Neural Object Detection for Robot Soccer*, 2019.

[Tian *et al.* 2019] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. *FCOS: Fully Convolutional One-Stage Object Detection*, 2019.

[Tong and Wu 2023] Kang Tong and Yiquan Wu. Rethinking pascal-voc and ms-coco dataset for small object detection. *Journal of Visual Communication and Image Representation*, 93:103830, 2023.

[Urrea and Agramonte 2021] Claudio Urrea and Rayko Agramonte. Kalman filter: Historical overview and review of its use in robotics 60 years after its creation. In *Journal of Sensors, vol. 2021*, volume 659, 9 2021.

[Veloso *et al.* 1998] Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim. The cmunited-97 small robot team. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 242–256, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[Viola and Jones 2001] Paul Viola and Michael Jones. Robust real-time object detection. volume 57, 01 2001.

[Wang *et al.* 2015] Naiyan Wang, Jianping Shi, Dit-Yan Yeung, and Jiaya Jia. Understanding and diagnosing visual tracking systems. *ArXiv*, pages 3101–3109, 12 2015.

[Wang *et al.* 2021] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-yuan Liao. Scaled-yolov4: Scaling cross stage partial network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13024–13033, 06 2021.

[Wojke *et al.* 2017] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*, 2017.

[Yilmaz *et al.* 2006] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13–es, dec 2006.

[Zhang *et al.* 2018] Jingwei Zhang, Lei Tai, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. VR goggles for robots: Real-to-sim domain adaptation for visual control. *CoRR*, abs/1802.00265, 2018.

[Zhang *et al.* 2020] Tianxiao Zhang, Xiaohan Zhang, Yiju Yang, Zongbo Wang, and Guanghui Wang. *Efficient Golf Ball Detection and Tracking Based on Convolutional Neural Networks and Kalman Filter*. arXiv preprint arXiv:2012.09393, 2020.

[Zhang *et al.* 2022] Z. Zhang, C. Wang, J. Song, and Y. Xu. Object tracking based on satellite videos: A literature review. *Remote Sensing*, 14(15), July 2022. This is an

open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

[Zhao *et al.* 2019] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–21, 01 2019.

[Zheng *et al.* 2019] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. arXiv preprint arXiv:1911.08287, 2019.

[Zhou *et al.* 2019] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. *Objects as Points*. arXiv preprint arXiv:1904.07850, 2019.