DISSERTATION

PERCEPTION SYSTEMS FOR ROBUST AUTONOMOUS NAVIGATION IN NATURAL ENVIRONMENTS

Submitted by

Ameni Trabelsi

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2022

Doctoral Committee:

Advisor: Ross J. Beveridge

Nathaniel Blanchard
Chuck Anderson
Emily King

ABSTRACT

PERCEPTION SYSTEMS FOR ROBUST AUTONOMOUS NAVIGATION IN NATURAL ENVIRONMENTS

As assistive robotics continues to develop thanks to the rapid advances of artificial intelligence, smart sensors, Internet of Things, and robotics, the industry began introducing robots to perform various functions that make humans' lives more comfortable and enjoyable. While the principal purpose of deploying robots has been productivity enhancement, their usability has widely expanded. Examples include assisting people with disabilities (e.g., Toyota's Human Support Robot), providing driver-less transportation (e.g., Waymo's driver-less cars), and helping with tedious house chores (e.g., iRobot). The challenge in these applications is that the robots have to function appropriately under continuously changing environments, harsh real-world conditions, deal with significant amounts of noise and uncertainty, and operate autonomously without the intervention or supervision of an expert. To meet these challenges, a robust perception system is vital. This dissertation casts light on the perception component of autonomous mobile robots and highlights their major capabilities, and analyzes the factors that affect their performance. In short, the developed approaches in this dissertation cover the following four topics: (1) learning the detection and identification of objects in the environment in which the robot is operating, (2) estimating the 6D pose of objects of interest to the robot, (3) studying the importance of the tracking information in the motion prediction module, and (4) analyzing the performance of three motion prediction methods, comparing their performances, and highlighting their strengths and weaknesses. All techniques developed in this dissertation have been implemented and evaluated on popular public benchmarks. Extensive experiments have been conducted to analyze and validate the properties of the developed methods and demonstrate this dissertation's conclusions on the robustness, performance, and utility of the proposed approaches for intelligent mobile robots.

ACKNOWLEDGEMENTS

I am grateful to many people who made it possible to conclude my Ph.D. studies at CSU. I would first like to thank my Ph.D. advisor, Professor Ross J. Beveridge, for admitting me to join the CwC research group and always being supportive. Ross provided me with all the guidance, assistance, and expertise that I needed in the beginning of my journey; then, when I felt ready to dive into research on my own, he gave me the flexibility and freedom to follow my own research interests, while continuing to provide me with constructive feedback and encouragement. This dissertation quite simply would not have been possible without his mentoring and guidance. I would like to express my gratitude to my Ph.D. committee members: Professor Nathaniel Blanchard, Professor Chuck Anderson, and Professor Emily King, for their insightful and valuable suggestions and professional guidance. I am very grateful to them for their constructive input on how to improve my research skills and work. I would particularly like to acknowledge Professor Nathaniel Blanchard for all his help, advice, and encouragement. I had the chance to work with him on research projects and co-author papers during my Ph.D., and I really enjoyed working with him. I greatly appreciated his continued guidance and support throughout my journey. I extend my gratitude to Dr. Lionel Gueguen, who has been a great inspiration to me. I was lucky to work with Lionel on some projects during my Ph.D., which helped me significantly to grow as a scientist. He has been and always will be a great source of inspiration and insight. Thank you, Lionel.

Last and foremost, I am deeply thankful to my parents, Habib and Sana, and my one and only brother, Aymen, for their love, support, and sacrifices. Without them, this dissertation would never have been written. I have saved this last word of acknowledgment for my dear husband, Mohamed Chaabane, who has been with me all these years and has made them the best years of my life. He is the one who encouraged me to follow my dreams, held my hand through this journey, and continues to be my greatest source of strength. Therefore, I would like to dedicate this dissertation to him.

DEDICATION

I would like to dedicate this dissertation to my dear husband and family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The world of Robotics is one of the most exciting areas that has been through constant innovation and evolution. Robotics are interdisciplinary and have become more and more a part of our lives. They are no longer a vision for the future but a reality of the present. In recent years we have witnessed a significant increase in intelligent robots. Nowadays, they are the subject of significant research projects in industry, military, security, and even home environments as consumer products for entertainment and home aid tasks. The ultimate goal is to achieve a fully autonomous robot that can independently fulfill its tasks in continuously changing, unconstrained environments without the need for human intervention. Such autonomy has only become possible thanks to the emergence of "Artificial Intelligence" (AI), the primary technology behind most breakthroughs in fields like self-driving cars. AI is "the study of agents that receive percepts from the environment and perform actions" as defined by Stuart J. Russell and Peter Norvig [4]. Its main challenge is to conceive and implement complex functions that can efficiently map the environmental percepts (such as images, sounds, point clouds) captured by the robot's sensors into appropriate actions. These functions are typically associated with human intelligence, such as learning, reasoning, problem-solving, and perception.

The project presented in this dissertation focuses on the perception component of autonomous mobile robots. More specifically, it aims to shed light on two main perception topics: Object detection/pose estimation, and motion prediction. These tasks serve to identify objects/actors in the scene, estimate their poses, and predict their motion in the near future. Such knowledge is crucial as it allows the robot to efficiently identify its surroundings and navigate without the risk of collision with the other actors in the scene.

1.1 Motivation

"Autonomous robot" is a vague term that refers to any system that can achieve a task solely without human interventions. For instance, it can refer to a manipulator's arm, an automation system for vehicle construction, or a financial analysis software. However, most people tend to use the term "robot" to refer to a particular category of robots commonly known, among technicians and people from the field, as "service robots" whose purpose is to help humans achieve recurrent and tedious tasks.

Though as straightforward as the problem statement might sound, autonomous robots nowadays are still far from being part of our everyday life. Current technologies only allow robots to operate perfectly in constrained environments such as automation industries where complete knowledge of the operating environment is required. In controlled space settings, robots can achieve remarkable speed, accuracy, and reliability that by far exceeds human capabilities. Their performance, however, decreases considerably in more natural unconstrained settings. This drop in performance leads us to the question: Why do robots fail to perform as effectively in natural environments as in controlled spaces? One reason behind this failure is the lack of adaptability of robots behaviors. Many practitioners focus on improving the performance of specific robot tasks and building systems to master certain applications. However, only a few researchers investigate the adaptability of robots and study their performance on multi-tasking, especially taking advantage of the correlation between multiple tasks and leveraging the knowledge from one task into another.

Another reason for the ineffectiveness of intelligent robots in natural environments is the lack of cognitive competence and adaptability to complex and rapidly changing environments. The ability to reason about tasks and problems in highly uncertain, continuously evolving, and interactive environments shared with other dynamic agents is a critical property for applying mobile robots in natural environments. Hence, AI is a crucial technique that can be leveraged to solve these challenges. More specifically, AI can be used to keep track of the evolution of the surrounding

environment by recognizing its different components and studying the motion of the dynamic ones.

The project presented in this dissertation contributes to the ambitious target of developing mobile robots that autonomously achieve complex tasks in unrestricted natural environments. Such a goal requires equipping the robot with strong reasoning capabilities, which allows it to adapt to a new unseen environment on the one hand and to reliably perceive and manipulate its environment on the other hand.

The following three complex robot tasks are examples of applications to illustrate the importance of the perception systems in intelligent mobile robots:

- **Pick-up and delivery services:** A robot should be able to deliver objects from and to different places. This task includes the perception of the object of interest, grasping capabilities, and placing the object on a surface. This dissertation focuses on perception capabilities, including the detection, recognition, and pose estimation of the object of interest. Issues on grasping and placing capabilities of objects will be excluded since this is an area of active research on its own.

- **Navigation in challenging environments:** The robot must be able to navigate safely in a dynamic environment, possibly with the presence of humans and other dynamic entities (such as other mobile robots). Navigation includes motion prediction of moving entities and path planning. In this dissertation, the motion prediction component is studied.

- **Inventory taking:** The robot should be able to detect objects, identify them, and keep track of their number, location and pose. This task targets objects of particular interest to the robot, i.e., objects that can be manipulated or influence the robot's navigation.

1.2 Basic Capabilities of Autonomous Robots

Autonomy is one of the most exciting problems of modern AI that has drawn the attention of many companies and researchers. It has the potential to change our everyday life as it allows robots

to acquire the necessary skills and intelligence to achieve complex tasks (such as safe and reliable navigation in unconstrained areas) that were once only achievable by humans.

Autonomous robots interact with the physical world by collecting percepts from the surrounding environment through their sensors. This capability is a crucial aspect of intelligent robots. Several sensory inputs such as cameras, microphones, wireless signals, active LiDAR, sonar, and radar are used to deduce the different aspects of the world.

Perception is one of the most critical capabilities for intelligent robots. It serves to translate the percepts of the environment into spatio-temporal cues helpful in planning their future actions. First, the environment is scanned using the robot's sensors. The obtained percepts are then collected and analyzed in order to decompose the scene into various entities, which are then utilized to learn complex spatio-temporal relationships. Specifically, the perception module can help answer questions like what actors are in the scene? Where are they located? How do they interact with each other? And how do they intend to move in the near future? Answering these questions helps the robot better plan their future actions, such as navigating in the surrounding without the risk of colliding with other entities.

Since it is impossible to predict, at the development phase, all situations the robot might encounter during deployment, a robot needs the *reasoning* capability to draw inferences appropriate to the situation flexibly. Given complex tasks assigned by their human instructor, a robot has to decompose its given tasks into achievable sub-tasks. Then, goal-directed decisions on its future actions have to be made.

Learning is one of the fundamental abilities as it guarantees the intelligent aspect of the system. It may be generally defined as the process of improving behavior based on experiences. Two significant learning capabilities are memorization and generalization. First, memorization is the ability to recall past experiences by storing the solutions to the encountered problems and using them in similar situations. Second, generalization involves the application of past experiences to new unseen situations. Like reasoning capabilities, learning enables a system to form behavior that the programmer did not foresee explicitly.

Planning is the capability responsible for translating the set of decisions taken by the robot into action. Planning is a general capability that depends on features that characterize the robot, such as object grasping/placing planning and path planning. For instance, to navigate in the environment, a robot needs to decide from all possible trajectories on what path to take. This path must be dynamically feasible, avoid collisions with all present obstacles, and ideally, it must be optimal to reach the target location with less energy consumption. The optimal path search capability is known as "motion planning". Motion planning is mainly a non-linear optimization task. Generally, numerical approximation methods solve the optimal trajectory task since exact solutions are usually computationally intensive. Multiple techniques have been used to solve this task, such as variational methods, graph-search approaches, and incremental tree-based approaches [5].

1.3 Dissertation Contributions

It goes without saying that one dissertation cannot solve all challenges of intelligent robot research. However, the perception capability is one of the most crucial components of an intelligent robot, and dealing with its challenges is inevitable. The project described in this dissertation aims to tackle some of these crucial challenges to enable a robot to achieve a variety of complex and abstract perception tasks autonomously. More specifically, it introduces a practical approach for solving the neighboring actors' recognition, poses estimation, and motion prediction. Ultimately, this dissertation fulfills the following targets:

1.3.1 Detection and Pose Estimation of Objects

The first step towards a functioning intelligent robot is to assimilate its surrounding environment by analyzing and identifying its different components. The robot needs to be able to detect objects, identify them and recover their 6D poses.

1. The detection and identification allow the robot to recognize the objects of special interest, such as the objects that can be manipulated by the robot or the dynamic objects that may influence its navigation.

2. The 6D pose estimation of objects helps the robot learn the location and rotation of objects with respect to the world. Such information is crucial for tasks such as object grasping. Furthermore, it can be used to deduce additional information about dynamic objects, such as their orientation, which helps predict their future movements.

In addition to accurate object detection, identification, and pose estimation, the project described in this dissertation aims to build a fast approach that is deployable in real-world scenarios. Multiple experimentation on public benchmarks will be used to analyze the performance of the proposed approach.

A practical contribution of this work is the enhancement of the capabilities of the embodied agent Diana [6–8]. Initially, Diana recognized the gestures and audio she received from the user and followed their instructions to manipulate virtual objects by grasping, lifting, moving, and sliding them in her virtual world. However, Diana could not interact with the real world as she only interacts with her virtual world based on the commands she receives from the user. Thus, this embodied agent will be used as a testbed to demonstrate the effectiveness of the proposed perception modules. The new skills will boost Diana's ability to interact with her surroundings by allowing her to "see" the real world, recognize the real objects present in the table in front of the user, and estimate their poses.

1.3.2 Motion Prediction of Moving Actors

Autonomous robots are rapidly advancing with various applications which require an effective motion planning that can handle high uncertainties in the environment and motion constraints due to static or dynamic obstacles. While dealing with these challenges in various capacities, this work tackles two fundamental motion planning problems:

1. First, this project describes three motion prediction models to predict the movements of the detected moving actors in the near future. This module is essential for planning safe and comfortable navigation.

2. Second, this dissertation highlights a crucial research question: What is the importance of the tracking module, and to what extent can it influence the performance of the motion prediction component? Generally, tracking, motion prediction, and planning components are cascaded. The actor tracking output feeds into the motion prediction, whose output, in turn, feeds into motion planning. However, such cascaded approaches are usually highly affected by errors propagating from noisy components. For instance, errors propagated from a noisy tracking module can hinder the performance of the motion prediction and, therefore, the planning module. Thus the study described in this dissertation aims to evaluate the influence of tracking on the performance of motion planning.

1.4 Dissertation Structure

This dissertation is organized as follows. Chapter 2 reviews the necessary background of the two main topics: object detection and pose estimation (see § 2.1) and motion prediction (see § 2.2). Existing end-to-end approaches that solve the two problems in an end-to-end fashion are also covered (see § 2.3). Chapter 3 presents the proposed approach for object detection and pose estimation. The approach can be dissected into two main components: a Pose Proposal Network (in § 3.1) and a Multi-Attentional Refinement Network (in§ 3.2). A set of experiments conducted on pose estimation are then covered and the obtained results are reported in § 3.4. Chapter 4 covers a comprehensive analysis of the motion prediction component. The set of experiments conducted in this analysis as well as the obtained results are reported in § 4.6.

Chapter 2

Related Work

This chapter aims to explore the intricacies of perception tasks in the context of autonomous navigation. First two perceptual tasks: object pose estimation, and motion prediction, are explored in their general context. Then, end-to-end methods that jointly solve the two perception tasks are covered. Throughout this manuscript, the term 'actor' or 'agent' are used to refer to any moving entity we are interested to identify and analyze its motion in the scene.

2.1 Object Detection and Pose Estimation

Object detection and pose estimation are two interconnected problems. Detection is the task of locating and identifying instances of real-world objects in images or videos, while pose estimation consists of determining the exact position and orientation of these objects relative to some coordinate system (Figure 2.1 **(A)**). There are many works that treat object detection and pose estimation as two separate problems [9, 10]; however, in this dissertation, the focus is on works that treat the two problems jointly. In the rest of this dissertation, this joint task is referred to as object pose estimation. Solving this problem has direct usage in multiple real-world applications such as robot manipulation, self-driving vehicles, and augmented reality. For instance, autonomous navigation requires precise object pose estimation to ensure accuracy and practicality. Though speed is a crucial factor to consider when comparing different approaches, a little delay can be tolerated in situations where time does not affect performance much. For example, self-driving and augmented reality systems need to use real-time approaches to be operable. However, robot manipulation can tolerate a little time delay without affecting its performance.

Though the task of object pose estimation can be defined with different types of sensor data such as LiDAR, RADAR... [11–13], Image-based approaches are by far the most popular [14]. Image inputs can be further categorized into two forms: Monocular images which include RGB and Gray-scale images, and RGB-D images which integrate depth information alongside RGB

information. Theoretically, the addition of depth information boosts the reliability and accuracy of pose estimation. Yet, in the real world, depth sensors suffer a variety of failure cases, have high energy and monetary costs, and are less ubiquitous than their non-depth counterparts. Ultimately, pose estimation from RGB alone is a more challenging problem, but also a far more attractive one. This manuscript is inclusive to the different image-based methods and approaches with a more focus on RGB based approaches.

The remainder of this section will cover the necessary background to explore the task of object pose estimation. Similar to many other fields, the emergence of deep learning has been a turning point in the evolution of this technology. To this end, the chronological order is followed in the following sections. Starting with an overview of the classical deep learning-free methods in § 2.1.1, deep learning-based methods will then be discussed in § 2.1.2. Furthermore, the refinement methods which were introduced to improve the performance of pose estimation models will be explored in § 2.1.3. This section will further highlight their advantages and drawbacks and discuss their importance for an accurate 6D pose.

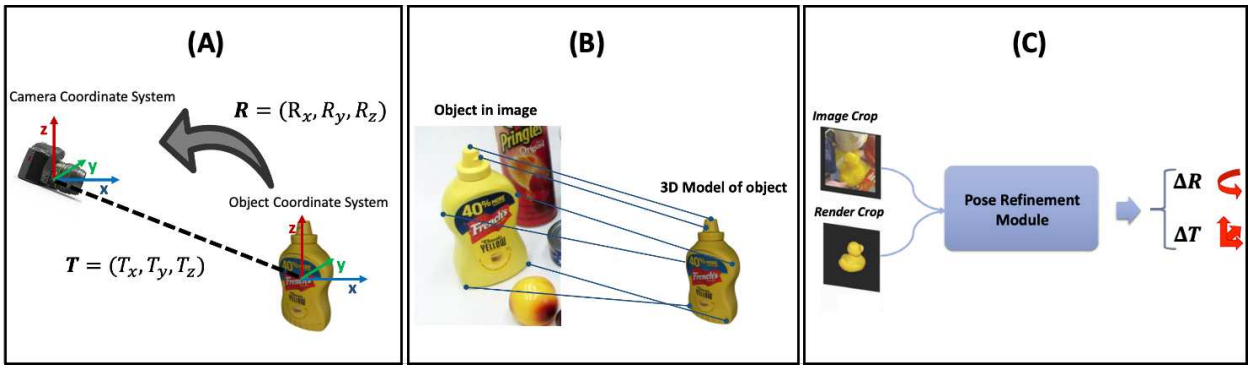

Figure 2.1: **(A)** Illustration of the 6D object pose estimation task. It is the estimation of the rigid transformation (R|T) from the object coordinate system to the camera coordinate system. **(B)** Illustration of key-points correspondences between a 2D image and the 3D model of the object of interest. **(C)** Overview of the task of iterative pose refinement using two inputs: an image crop and a rendered crop of the 3D model of the object of interest using an initial pose estimate. The pose refinement output is an estimation of the relative transformation (pose residual) between the ground-truth pose represented in the image crop and the initial pose estimate represented in the render crop.

2.1.1 Classical Object Pose Estimation

Object pose estimation has a long and storied history. In robotics, registration algorithms such as RANSAC [15] and ICP [16,17] were widely used. The idea of these algorithms is to estimate the 6D pose estimation of an object of interest by fitting the object's 3D model to images or 3D point clouds of the same object. In order to properly converge, these methods require an accurate pose initialization algorithm since failing to satisfy this condition can lead to sub-optimal pose estimates. Therefore, many works rely on additional processes to generate accurate pose initialization which can be cumbersome and time-consuming. For instance, one common process is a brute force multi-hypothesis generation where a set of predetermined pose initializations are evaluated, and the best fit is selected [18].

Many vision-based approaches were also introduced to solve the object pose estimation problem. Common classical approaches include template-based methods [19–21] and feature-based methods [22, 23]. In template-based methods, templates are first created by rendering the 3D model of the object of interest. Then, the templates are scanned over all locations of the input image where a similarity score is measured. Finally, the best match is recovered by selecting the location with the highest similarity score. Template-based methods are known to work well with texture-less objects but fail to handle occluded objects as this will lead to a low similarity score with the created occlusion-free templates. Feature-based methods consist in extracting local features from the input image whether from every pixel or from selected points of interest. The extracted features are then matched with features extracted from the object's 3D model. The obtained 2D – 3D correspondences can then be used to recover the object pose. While feature-based methods can overcome occlusion, they generally fail with low textured objects.

2.1.2 Modern Object Pose Estimation

With the emergence of deep learning in many fields, the object pose estimation domain was also revolutionized as deep neural network architectures have shown to be more effective than handcrafted descriptors and classical algorithms. The switch to deep learning has come about

gradually with time. With the success of deep learning techniques in other tasks such as object detection and segmentation, researchers started to integrate those techniques in pose estimation by introducing hybrid methods where only a part of the pipeline was deep learning-based, such as the feature extractor or the classifier/regressor. Some other researchers chose to use deep neural networks to extract local object key-points to learn correspondences between the object in the 2D input and the corresponding 3D model. These correspondences are then used to recover the object's 6D pose with a PnP algorithm [15]. PnP algorithm is a method to estimate the pose of a calibrated camera given a 2D-3D (image pixels to 3D points in the world) correspondence of a set of n points. This two-stage pipeline has witnessed great success given that 2D key-point detection is relatively easier than 3D localization and rotation regression directly from the input image. Assuming we obtain rich and noise-free local information from the detected key-points, PnP has shown to be effective for 6D pose recovery. More recently, fully deep learning-based methods were introduced. These methods estimate 6D poses directly from the input image in a single shot and have shown to give competitive results. In the remainder of this section, hybrid methods will be covered in § 2.1.2. Then, § 2.1.2 will briefly dwell in 2D-3D correspondences based methods. Finally, end-to-end methods and their effectiveness as compared to 2D-3D correspondence-based methods will be discussed in § 2.1.2.

Hybrid Methods

Early works on object pose estimation with deep learning tried to benefit from the advantages of both classical and deep learning techniques by adopting hybrid approaches. In some works, hand-crafted feature representations were used to work with neural networks based classifier/regressor. The work of Mousavian *et al.* [10] started with 2D bounding boxes of objects in the input image and, used projective geometry to recover translation components and add geometric constraints to the object's orientation and size estimates. The object orientation was then regressed by discretizing the orientation angle and dividing it into overlapping bins. For each bin, CNN networks were used to estimate a confidence value for each bin and a residual rotation. Others chose to use CNNs to extract feature representations from the input image and then applied classical algorithms

to learn the objects' poses. For instance, a more recent work of Kehl *et al.* [24], used CNNs to learn descriptors of locally sampled RGB-D patches. The learned features were then matched to a codebook of pre-computed synthetic object patches using k Nearest Neighbors (kNN) [25] to extract k nearest synthetic patches which were then used to cast votes for the final 6D pose.

With the great success of deep neural networks in object detection, other hybrid approaches [26, 27] tried to leverage popular deep learning-based 2D object detectors [28–30] and augment them for 6D pose estimation. Kehl *et al.* [27] extended the "Single Shot Detector" (SSD) [29] to determine 2D bounding boxes as well as possible viewpoints and in-plane rotations. The rotation estimation was then treated as a classification problem based on an appropriate sampling of the rotation space. The translation estimation was processed offline and used to pre-compute bounding boxes for all possible sample rotations and selecting the best fit as the final estimation. Although hybrid methods try to achieve the best of both worlds, the following sections will show that extending the network to the full pipeline has shown to achieve better performance.

Two-Stage Methods Based on 2D – 3D Correspondences

Despite the many successes, many researchers have argued that deep neural networks have a major limitation [31, 32]: Though deep networks have proven their effectiveness in the task of object detection and classification, their effectiveness in regression tasks is still controversial due to the large output space which can grow to infinity when regressing real-valued outputs. To overcome this problem, many works attempted to convert the pose regression problem into key-points detection. A two-stage pipeline was adopted where the first stage detects local 2D key-points of the object of interest from the input image (See Figure 2.1 **(B)**). The second stage uses the obtained 2D-3D correspondences between the detected key-points and their matches in the object's 3D model to compute the 6D pose using a classical algorithm such as the PnP algorithm [33]. Thanks to the robustness of CNNs in key-point localization, these methods managed to overcome the difficulty in handling texture-less objects and low-resolution images [32]. For instance, Rad *et al.* [34] introduced Bb8 which leveraged CNN based segmentation techniques to localize objects of interest in the 2D input image as well as their classes. It then applied CNN-based architecture

on the detected region of interest to detect the vertices of the object 3D bounding box projected in the image space. The 6D pose was recovered using PnP.

Tekin *et al.* [31] opted for 2D detection techniques instead of segmentation. It used the popular YOLO detector [35] and extended it to detect the vertices of the object's 3D bounding box in the input image. The object's 6D pose was then similarly estimated using a PnP algorithm. Tekin *et al.*'s approach is now known as YOLO6D. Since its introduction, YOLO6D has witnessed great success not only thanks to its relatively simple one-block architecture followed by PnP but also thanks to its fast runtime which was estimated by 50fps on a Titan X GPU. That being said, due to the competitiveness of the field, the performance of YOLO6D was quickly beaten by other 2D-3D correspondences-based approaches which were later introduced.

The novelty of HMap [36] resides in the use of independently predicted heat-maps extracted from multiple small patches of the image to predict the 2D projections of 3D points related to the target object. The authors argued that predicting heat-maps is more robust than regular key-point detection which can be highly sensitive to partial occlusions. The main focus of their paper was on predicting heat-maps and accumulating the results to obtain accurate and robust predictions. However, since heat-maps have a fixed size, these methods have difficulty in handling truncated objects where some of the key-points lay outside of the input image.

Further attempts to better model the 2D-3D correspondences aimed to find a more flexible representation for key-point prediction such as vector fields. Vector fields fall under the category of dense methods where every pixel outputs a prediction and casts a vote for the final output following a voting scheme [37, 38]. More specifically, every pixel predicts a direction to the estimated key-point location. The final key-point location is then determined by voting from the predicted directions of all pixels. In 2018, Peng *et al.* [32] introduced PVNet, which stands for "Pixel-wise Voting Network". PVNet regresses pixel-wise unit vectors that point to the predicted key-point directions. These vectors are then aggregated to vote for the final key-point locations using a RANSAC [37] based voting scheme. Final poses are then recovered using PnP.

Despite the success of these methods which achieved state-of-the-art performance when first introduced, this two-stage process is still considered, by many, sub-optimal. First, this process is hybrid since it still uses a geometric algorithm to recover the final 6D object poses. As a result, such methods cannot be trained in an end-to-end manner, something that is known to help improve the performance of the model [39, 40]. Second, training the network for 2D-3D correspondences relies on a loss function that does not reflect the final 6D pose estimation task. Furthermore, these methods can still fail when dealing with texture-less objects. In the following section, the alternative of end-to-end methods will discussed and their performances will be compared with the above-mentioned methods.

End-to-End Methods

Despite the popularity and success of end-to-end methods in other regression tasks such as disparity estimation and flow estimation, most of the recent 6D object pose estimation methods are still based on the two-stage pipeline described in § 2.1.2. That being said, early attempts to apply end-to-end approaches for pose estimation were suggested. PoseNet [41], introduced in 2016, is one of the earliest fully deep learning-based methods for pose estimation. Unlike other methods introduced at the time, PoseNet adopts an end-to-end approach that relies on a modified version of Inception architecture [42] to directly regress a 6D camera pose from an input RGB image. At inference time, the network operates in real-time and is able to generalize to unseen scenes. However, regressing the 3D translation vector directly from the input image was a little challenging especially due to the lack of depth information since it only relies on RGB inputs, and also the large search space. To address this problem, PoseCNN [39], introduced two years after PoseNet, operates by localizing the object center in the 2D RGB input and then estimates the center's distance from the camera to finally obtain its translation components.

The PoseCNN network [39] consists of two blocks: The first block extracts feature maps at multiple resolutions from the RGB input, and the second block embeds the high dimensional feature maps extracted from the first block into low-dimensional task-specific features. The second block consists of three sub-networks that perform three different tasks: semantic labeling, 3D

translation estimation, and 3D rotation regression. It is important here to highlight the decoupling of the estimation of 3D translation and 3D rotation as these two components are considered as two separate tasks. The regression of the object 2D center for translation estimation is obtained by a pixel-wise regression of a unit vector pointing towards the object center. Each pixel casts a vote to the direction it points to, then the center is selected after aggregating all votes using Hough Voting algorithm [38]. This approach allows the object center detection to be robust to occlusion and truncation. The rotation is estimated by regressing a 4-dimensional quaternion vector. In addition to the novelty of its architecture, PoseCNN uses two novel loss functions for rotation estimation, "PLoss" and "SLoss", to handle both asymmetric and symmetric objects, respectively. PLoss measures the average squared distance between points on the ground-truth model pose and their corresponding points on the predicted model pose. SLoss measures the offset between each point on the predicted model pose and their closest point on the ground-truth model pose.

Despite the success of PoseCNN in industry, thanks to its meticulously established network and loss function, end-to-end approaches were still not popular for pose estimation and most contemporaneous works, such as HMap and PVNet, still opted for two-stage approaches where deep networks are only used to establish the 2D-3D correspondences. Recently, in 2020, a resurgence of end-to-end methods for pose estimation has been taking place [40, 43]. Hu *et al.* [40], for instance, introduced a novel network that extends the 2D-3D correspondence networks to directly regress the 6D poses. The proposed network can be exploited in conjunction with any existing 2D-3D correspondence network. Experiments with existing methods such as PVNet and their novel single-stage variants showed that the single-stage approach consistently outperformed its two-stage counterpart in both accuracy and speed.

Overall, these methods can handle texture-less objects very well unlike two-stage approaches. Nonetheless, there is still room for improvements in terms of pose estimation accuracy, especially because small errors in the pose regression can easily lead to pose mismatches. To further improve the pose estimation performance, many works tend to use an additional component known as pose refinement which will be discussed, in the following section.

2.1.3 Pose Refinement

Regression of object poses in a single shot straight from an input image can have limited accuracy. A common way to improve the pose estimation performance is to include an additional component to further refine the initial estimates. Figure 2.1 **(C)** shows an overview of the refinement step. Given an initial pose estimate, a new synthetic image can be created by rendering the 3D model of the object of interest. The obtained render image is then matched with the original input image to learn a better pose estimate. A common traditional refinement method is the Iterative Closest Point (ICP) algorithm with projective data association and a point-plane residual term. First, a point cloud of the object 3D model is rendered using the initially estimated pose. The depth values of the observed points are obtained by associating them with the depth values of the rendered points at the same pixel locations. Then, a residual distance is measured as the 3D distance from the observed point to the plane defined by the rendered point and its normal. Finally, points are rejected if their residuals lie above the selected threshold and the residuals of the remaining points are minimized using gradient descent. ICP refiner has been adopted by many pose estimation methods that have been mentioned in previous sections, notably PoseCNN and SSD-6D. However, ICP is highly slow for real-time applications and is not differentiable so it cannot be trained jointly with the pose estimation network. Therefore, many works switched to deep learning-based refiners such as Manhardt *et al.*'s refiner [44] and DeepIM [45].

The work of Manhardt *et al.* [44] presented a novel CNN network that iteratively refines an initial pose estimate, given an input image and an object's 3D model. Their approach operates by aligning object contours between object rendering, using the initial pose estimate and the observed image. Furthermore, this work introduced a novel formulation of a visual alignment loss that implicitly optimizes for metric translation and rotation. Architecture wise, the network consists of 3 main steps. First, a subset of Inception V4 network [46] is used to extract deep feature maps from both inputs. The extracted features of the first step are then concatenated and fed to another sub-network to extract higher-level features. Finally, the obtained features are fed into two separate branches for rotation regression and translation estimation. In summary, even though Manhardt *et*

al.'s method does not need depth information to operate unlike ICP, it has shown to be robust to occlusion and accurate even with rough initialization.

Generally, refinement methods proceed by extracting appearance features from both inputs using networks pretrained on image classification such as VGG16 [47] or Inception [46]. The obtained features are then concatenated and fed to the next steps of the process for pose residual estimation. In 2019, DeepIM [45] proposed a novel feature extraction approach that relies on optical flow estimation. DeepIM leveraged the FlowNet Simple architecture [48], which is an encoder-decoder network that estimates the optical flow between two input images. The extracted features from the FlowNet encoder are used to learn the transformation between the predicted and the target pose. The authors confirmed their intuition about the usefulness of optical flow-related representation in the task of pose matching. To show the effectiveness of the proposed approach, they compared the performance of their network with its variant using VGG16 [47] and showed that their network, by accounting for flow features, outperformed its counterpart variant. DeepIM also proposed a novel disentangled representation of pose residuals. The proposed pose representation helped the network to learn a pose transformation independent from the camera parameters and actual size of objects in addition to any complex geometric relationships between translations and rotations. According to the authors, such representation helped achieve accurate pose estimates and enabled the network to handle unseen objects.

2.1.4 Performance Comparison and Summary

In Table 2.1, a performance summary of some of the methods described in this chapter on two benchmarks LINEMOD [20] and Occlusion [49] is given. The methods are divided into three categories representing the three general approaches noted in this chapter: Hybrid, Two-stage based on 2D-3D correspondences, and end-to-end methods. We also report the results of the methods before and after the refinement step if available. The results are reported in ADD metric [20] with a threshold of 10% of the object's diameter.

Table 2.1: The reported results of the discussed methods on two common datasets LINEMOD and Occlusion. We divide the methods into three categories: Hybrid, Two-stage (based on 2D-3D correspondences), and end-to-end methods. We also report results of some methods after adding a refinement step, used to improve the performance. The reported metric is ADD [20] with a threshold of 10% of the object's diameter. The best results are highlighted in bold.

Category	Input	Approach	LINEMOD [20]	Occlusion [49]
Hybrid	RGB	SSD-6D [27]	2.42	-
+ Ref	RGB-D	SSD-6D+ref [27]	79	-
Two-stage	RGB	YOLO6D [31]	55.95	6.42
		PVNet [32]	86.27	40.77
		HMap [36]	-	30.4
		BB8 [34]	43.6	33.8
+ Ref	RGB	BB8+ref [34]	62.7	-
End-to-end	RGB	PVNet [32] + [40]	-	43.3
		PoseCNN [39]	-	24.9
+ Ref		PoseCNN [39] + DeepIM [45]	**88.6**	55.5
	RGB-D	PoseCNN [39] + ICP	-	**78.0**

First, the Hybrid approaches perform the poorest among the three categories which proves the superiority of pure deep learning based approaches. YOLO6D, for instance, shows a huge improvement over SSD-6D on the LINEMOD dataset with an ADD of 55.95% compared to 2.42% for the SSD-6D. Second, both the two-stage and end-to-end methods show competitive results with end-to-end methods showing some improvements compared to the two-stage approaches. For instance, the end-to-end extension of PVNet ([32] + [40]) outperforms its two-stage version ([32]) by 6.2% in ADD on the Occlusion dataset. Furthermore, the integration of a refinement step considerably enhances the model's performance. The addition of a refinement step in BB8 has increased its performance by 43.8% on the LINEMOD dataset. Similarly, the integration of the refiner DeepIM in PoseCNN has boosted its performance by 122.8% on the Occlusion dataset. Finally, the integration of depth information in the model's input have shown to considerably increase its performance. The use of ICP refiner which takes advantage of the depth information with PoseCNN have shown to increase its performance over DeepIM refiner by 40.5% in ADD on the Occlusion dataset. Though, the integration of depth information has shown to be advantageous

in terms of performance, in this dissertation, the focus is on methods with RGB only inputs since it is a cheaper and more adaptable approach to tackle the 6D pose estimation problem.

2.2 Future Motion Prediction

Understanding actors' motion is essential for intelligent systems to coexist and interact with surrounding actors. It has a direct effect on many aspects such as perception and motion analysis. Depending on the application, actors can be humans, vehicles, or any other moving entities. Motion prediction is a crucial task as it allows to predict the evolution over time of a scene involving multiple actors. Such knowledge can be incorporated for an enhanced perception, human-robot interaction, and planning.

Many important applications exist including autonomous driving systems, advanced surveillance systems, and service robots. Given the growth of this research topic in multiple domains, this dissertation will mainly consider works that focus on motion prediction in the context of autonomous driving systems. In an autonomous driving scenario, surrounding actors include vehicles, cyclists, and pedestrians. The challenge of making accurate motion predictions arises from the complexity of the highly uncertain, rapidly changing, and interactive environment. As a matter of fact, an actor movement can be directly influenced by their own intent, presence or actions of surrounding actors, and the environmental constraints such as lane boundaries, walls, crossroads... Most of these factors cannot be extracted directly from the system sensors and require additional processing by extracting them from noisy perceptual cues or modeling them from context information.

This section will cover methods that rely on a subset of these factors to predict the future motion of actors, as well as methods that are exhaustive to all factors. The following topics in this section are divided as follows: First, in § 2.2.1, classical engineered methods for motion prediction that are still widely used in industry are revisited. Then, more recent deep-learning-based approaches are covered in § 2.2.2 which will be categorized based on their proposed architecture. First, sequence prediction models that rely on actors' historical dynamics in order to predict their future behaviors

are explored (See § 2.2.2). Second, CNN-based approaches that operate on Bird's Eye View (BEV) raster representations of the scene to predict future movements of actors are covered (See § 2.2.2). Finally, Graph Neural Network (GNN) based approaches where each entity of the scene is treated as a node in an interconnected graph as well as other hybrid approaches are presented (See § 2.2.2).

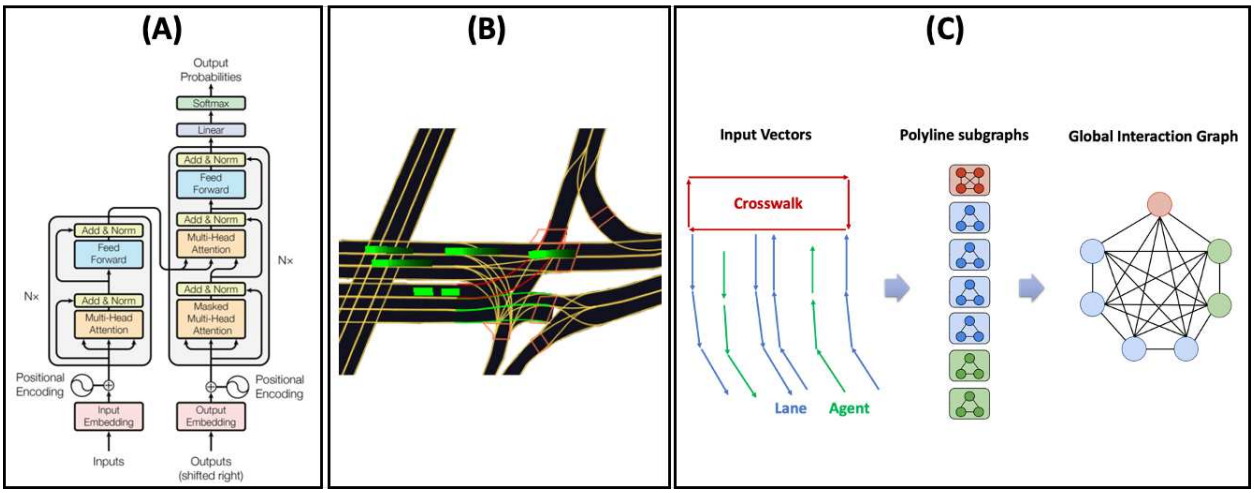

Figure 2.2: **(A)** The Transformer model architecture taken from [50]. **(B)** Example of RGB rasterization results of a Bird's Eye View scene extracted from the self-driving HD map of the Lyft Dataset [51]. Components of the scene are represented with different colors such as roads in black, lanes in yellow, traffic lights in green and red, crosswalks in orange, and vehicles in light green. The historical polygons of the vehicles are also shown with the same color but with a reduced brightness. **(C)** Polyline representation for GNN based methods. The illustrated approach follows the work of Gao *et al.* [52] where each map entity is represented by a polyline of vectors. Each polyline forms a separate subgraph and all subgraphs are connected through a Global Interaction Graph. This approach allows to integrate both social and context interaction in a Graph representation.

2.2.1 Classical Methods

Most of the classical methods rely on kinematic models to predict the future movements of actors. In this section, a subset of these methods are briefly examined. Despite the unfolding of deep learning in all domains including motion analysis, well-established engineered motion prediction methods are still witnessing great success in industry thanks to their simplicity and acceptable performance.

One standard technique for motion prediction is Kalman-filter-based (KF) prediction of dynamic obstacles using a constant acceleration model [53]. KF is used to recursively estimate and propagate the actor's state in the future, given noisy sensor measurements. Generally, an actor's state at a given time-step comprises its position, speed, acceleration, and heading. In the standard KF based methods, the uncertainty of the actor's current state and its evolution are modeled by a normal distribution. Overall, KF based methods consist of two main steps: a prediction step and an update step. In the prediction step, the actor's state at the current time-step is fed into the dynamic model which projects it into the next predicted state in the form of a gaussian distribution. In the update step, the obtained sensor measurements at the following time-step are combined with the predicted state to form an estimated state, at that time-step, which also follows a gaussian distribution. These two steps are repeated recursively every time a new measurement is available. In the case of motion prediction, the goal is to predict multiple states in the future where sensor measurements are not available. To this end, the prediction step is looped over multiple times to obtain multiple future states prediction. The mean and covariance matrix of each predicted state can be transformed into mean location and its associated uncertainty. Multiple extensions to KF were developed such as the Extended KF [54] and the Unscented KF [55] which work on nonlinear systems.

Other classical methods that are based on machine learning were introduced in the context of motion prediction. Hidden Markov Models [56], Gaussian Mixture Models [57] and Processes [58] are some of the techniques that were leveraged to solve this task. While these methods are relatively simple to use in deployed systems, they usually suffer from generalization issues when applied to real-world scenarios, especially with noisy detections. Furthermore, they can fail in modeling complex actor/environment interactions. More sophisticated deep learning methods were introduced to tackle these problems which will be covered in the remainder of this section.

2.2.2 Deep Learning Methods

Solving the motion prediction task using deep learning techniques is an appealing option since deep neural networks are flexible and can be designed to capture complex patterns. Furthermore, if properly designed and trained with rich datasets, deep learning models can be generalizable to unseen scenarios. In this section, deep learning methods are categorized into three classes based on their input representations and architectures. The first class comprises the majority of the proposed methods which are sequence-based models that operate on a sequence of actor's historical states to predict their future trajectory. The second class represents CNN-based methods that operate on a BEV raster representation of the scene. Finally, the third class includes GNN-based methods as well as hybrid methods which combine different techniques and types of networks.

Memory Based Models

It is straightforward to address the motion prediction task using sequence-based models, where the actor's history of movements is modeled as a sequence of states, mainly because these models are known to best capture the sequence's dependencies and dynamics. One common technique for sequence learning is Recurrent Neural Networks (RNN) and specifically Long Short-term Memory Networks (LSTM) which has become a widely popular modeling approach for predicting human [59,60], vehicle [61], and cyclist [62] motion. However, treating the actors as separate independent entities often leads to sub-optimal performance since they live in an interactive inter-dependent environment where each entity can be directly affected by its surrounding. As a result, recent works proposed various approaches to handle the task at hand while accounting for the different factors that can affect their predictions.

As a pioneer, Alahi *et al.* [59] introduced the Social LSTM model to predict pedestrians' trajectories while accounting for the interactions among them. Each person was modeled by a single LSTM and the interaction between people was modeled using a social pooling system where each LSTM shared its hidden state with other spatially neighboring LSTMs. Further attempts to improve the performance of socially aware sequence models focused on including the interaction with other entities such as static objects. As an example, the work of Bartoli *et al.* [60] extended

Social-LSTM by adding a "context-aware" pooling layer which accounted for neighboring static entities when predicting future movements of pedestrians. Despite their success, RNNs have been target to many critics since the sequential nature of RNNs prevents parallel processing of the input sequence. Unlike CNNs, the predictions for later time-steps using RNNs must wait for their predecessors to complete which makes their running time relatively slow. Further, with long input sequences, RNNs have a major issue of exploding/vanishing gradients. Furthermore, RNNs suffer from high memory requirements to store partial results especially in the case of long input sequences.

While most recent memory-based successes on motion prediction are based on LSTM models and the main focus of researchers in this field is on better modeling the interaction of the actors with their surroundings, some researchers [63–66] chose to side-step sequential learning and proposed only-attention-based memory mechanisms of Transformers (TF). Transformer networks were first introduced in the field of Natural Language Processing (NLP) in 2017 to model word sequences [50]. They mainly rely on their attention mechanism that learns to focus on the more important parts of the input sequence for a better prediction. Besides, TFs operate on the full sequence at the same time, while RNNs operate sequentially which boosts the running time optimality. Furthermore, Transformers allow for more parallelization during training compared to RNNs which enabled training on larger datasets. Quickly after its introduction, Transformer based methods became state-of-the-art in NLP field which proved its effectiveness. The architectural details are given in Figure 2.2 **(A)**. The building blocks of a TF architecture mainly are embedding layers and positional encoding layers, at the start of both the encoder and the decoder, multi-head attention mechanism and fully connected layers. The embedding layers serve for learning an embedding representation for their input while positional encoding layers are responsible for the notion of order in the input/output sequential data of the TF. Multiple variants of Transformers were also introduced such as the Bidirectional Encoder Representations from Transformers (BERT) [67] which is a bidirectional architecture consisting of a sequence of stacked TF encoders.

Giuliari *et al.* [63] used TFs to model each actor separately without considering the different interactions with the environment. More specifically, they assessed the performance of the original TF as well as BERT on common benchmarks and showed that TF outperformed all existing techniques, even those including social mechanisms. The novel application of TFs in motion prediction has opened a new research direction in this field. Many recent research works revolved around leveraging TF techniques while taking the social interaction into account [64–66]. Saleh *et al.* [64] for instance, extended the original TF architecture by proposing a novel context-augmented TF architecture focusing on the embedding layers and positional encoding layers to account for contextual information around moving agents i.e. their interaction with other agents and the scene.

Raster Based CNN Models

Unlike in RNNs where the input sequence is processed in sequential order, convolutions can be done in parallel since the same filter is used in each layer and the input sequence is processed as a whole. CNNs can process a multi-dimensional input, where the spatial and temporal information are represented in separate dimensions while being efficient in terms of running time. As a result, many researchers [68–70] worked on using a Bird's Eye View (BEV) representation of the scene and applied rasterization techniques to provide complete context information through time necessary for accurate motion prediction.

To better illustrate the rasterization technique, it is necessary to cover the concept of vector graphics. The vector graphics representation consists of a stack of multiple vector layers where each layer is formed by a collection of polygons and lines that represent a specific object class. For example, in the context of self-driving vehicles, layers can represent lanes, boundaries, crosswalks, and other elements present in the scene. Rasterization is the task of converting vector graphics into a raster image. For instance, to rasterize into RGB format, each vector layer is represented by a different color. Besides, to capture the past movements of actors, their bounding boxes at historical time-steps are rasterized on top of the vector layers. Each historical actor polygon is rasterized with the same color as its corresponding current actor but with a reduced level of brightness resulting in a fading effect (See Figure 2.2 **(B)**).

Raster representation has witnessed great success especially in the domain of self-driving vehicles. Djuric *et al.* [68] proposed to use RGB raster representation of the BEV of the autonomous vehicle (AV) and its vicinity to account for complete context and actor-scene interactions. The authors suggested a deep CNN model that takes the BEV raster centered around the actor of interest as input. The extracted features are then combined with the actor's state information, including velocity, acceleration, and heading, and fed to fully connected layers. The model finally predicts the actor's future trajectory as well as the inherent uncertainty of their motion in road traffic which they assumed to follow a half-normal distribution. The intuition behind uncertainty estimation is to reflect the situation's peculiarity by estimating all possible directions an actor might take as well as their corresponding probabilities. For example, if an actor decided to unfollow the traffic flow and take an unexpected turn or cut the road and change lanes, the uncertainty distribution should account for all possibilities with their relative probabilities. In addition to their first model, Djuric *et al.* [68] experimented with decoding the actor trajectory using an LSTM layer which they stacked after the first fully connected layer. Experiments have shown that their second variant outperformed the first LSTM free variant.

Just like in pose estimation domain, some approaches tried to convert the motion prediction problem, which is originally a regression task, to a multi-modal probabilistic classification task by spatially discretizing the prediction space into a set of possible trajectories or modes [69–71]. CoverNet [69] followed this path and extended the work of Djuric *et al.* [68] by proposing a multi-modal probabilistic trajectory prediction where engineered algorithms were used to dynamically structure the trajectory set based on the actor's current state. This aimed to ensure the coverage of all possible scenarios while eliminating the physically impossible ones. TPNet [72] followed a similar idea to CoverNet but solved the problem in an end-to-end fashion. TPNet first generated a set of trajectory proposals as candidates of possible hypotheses, then made the final predictions by classifying and refining the proposals following the underlined physical constraints.

Further amelioration of motion trajectory prediction methods from raster inputs aimed to use a joint input representation by considering both raster representation of the scene and specific feature

representation of the actors' historical states. Messaoud *et al.* [70], for instance, used LSTMs to learn feature embeddings of the actors' recent movements and combined it with context features learned from the scene information. The combined representation was used as input to a multi-attentional CNN based network where each attention head generated a possible distribution over a possible predicted trajectory.

Other Methods: GNN and Hybrid Methods

Social interactions can be naturally modeled using connected graphs where each node represents a single actor, and each edge connecting two nodes represents a direct social interaction between the two actors. Further, thanks to the tremendous increase in the available computational power in recent times, a lot of attention has been directed towards Graph Neural Networks (GNN) to solve the motion prediction task.

An example of GNN-based methods is Social-STGCNN [73]. Social-STGCNN modeled the pedestrian trajectories as a spatio-temporal graph and used Graph Convolutional Neural Networks (GCNN) followed by Time-Extrapolator Convolution Neural Network (TXP-CNN) to operate on the temporal dimension. Though these methods used graph representation which is a natural and straightforward tool to capture the social interactions, they failed to model the interactions of actors with other environment entities and constraints. VectorNet [52] addressed this problem by adopting a polyline representation where each entity was represented by its vectorized form (See Figure 2.2 **(C)**) in a separate subgraph. The formed subgraphs were then connected through a global interaction graph to capture the different interactions between the environment entities. The authors of VectorNet suggested that their method could capture the different social interactions, outperformed convolution-based methods, and also reduced the computational cost. Nevertheless, it is known that polyline representation is often hard to extract automatically from sensors and requires human annotations.

As many techniques have been adopted to solve the motion prediction task, some works tried to achieve the best of all worlds and proposed to use hybrid methods where two or more techniques are jointly used to solve the task at hand. Social-BiGAT [74], for instance, presented a graph-

attention-based generative adversarial network that generated multi-modal trajectory predictions. It first used a graph attention network (GAT) to learn feature representations modeling social interactions. Then, it relied on a recurrent encoder-decoder that was trained adversarially to predict the future trajectories.

Trajectron [75] and Trajectron++ [76] were two consecutive works for motion prediction. Trajectron was first introduced as a graph-structured model that combined tools from recurrent sequence modeling and variational deep generative modeling for multi-modal trajectory predictions of multiple agents in the scene. Trajectron++ was built upon Trajectron to produce a dynamically feasible trajectory hypothesis for multiple agents in the scene and include environment context information by extracting map features using CNNs.

2.2.3 Performance Evaluation and Summary

Table 2.2 summarizes the described methods in this section and provides a comparative analysis based on the different characteristics of their approaches. First, we can see the advantage of using TFs against LSTMs in the task of motion prediction. The TF model outperforms S-LSTM by 0.48m on the ETH dataset. Similarly, the integration of CNNs operating on raster input representation in addition to the sequence information improves the model's performance over sequence-only based methods. For instance, MHA-JAM outperforms S-STGCNN by 0.10m in ADE on ETH dataset. Comparing MHA-JAM to Trajectron++ shows that the addition of GNNs does not necessarily improve the model's predictive performance. Comparing TPNet and Trajectron++, on the other hand, proves the advantage of accounting for social interactions. In fact, Trajectron++, by accounting for social interactions, outperforms TPNet by 0.19m in ADE on ETH dataset. Furthermore, looking at the two best performing models (Multipath on nuScenes dataset and Trajectron++ on ETH dataset) show the importance of all of the factors social interactions, context and multimodality in improving the performance of the model.

Table 2.2: Overview of the described methods and their characteristics. The input type and the main approach/architecture are described. Other factors such as whether the method accounts for social interaction and context awareness and whether it outputs a single unimodal possible trajectory or probabilistic multimodal trajectories are also considered. The performance of the models is reported in ADE on two datasets nuScenes [2] (on the left) and ETH (on the right) [77]. Results on nuScenes are reported over a prediction horizon of 6 seconds and $k = 5$ modes. Results on ETH are over the next 12 frames.

Method	Input	Approach	Social	Context	Multimodal	Performance
UKF [55]	seq	UKF	No	No	No	-/-
Social LSTM [59]	seq	LSTM	Yes	No	No	-/1.09
Multipath [71]	raster	CNN	Yes	Yes	Yes	**1.78**/-
TF [63]	seq	TF	No	No	No	-/0.61
MTP [78]	raster+seq	CNN	Yes	Yes	Yes	2.22/-
CoverNet [69]	raster+seq	CNN	Yes	Yes	Yes	2.62/-
TPNet [72]	raster+seq	CNN	No	Yes	Yes	-/0.54
MHA-JAM [70]	raster+seq	CNN+LSTM	Yes	Yes	Yes	1.85/-
S-STGCNN [73]	seq	GCNN+CNN	Yes	No	Yes	-/0.64
VectorNet [52]	polyline	GNN	Yes	Yes	No	-/-
S-BiGAT [74]	seq+RGB	GAT + LSTM	Yes	Yes	Yes	-/0.69
Trajectron++ [76]	seq+raster	CNN+LSTM+GNN	Yes	Yes	Yes	1.88/**0.35**

2.3 End-to-end Object Detection and Motion Prediction

Identifying the environmental states is crucial for several applications such as human-robot interaction and motion planning for self-driving systems. The full understanding of the environmental states generally includes the identification of the objects of interest from the background and the prediction of their motion in the near future. Many researchers have focused on these two problems especially in the domain of self-driving systems and multiple milestones have been achieved in this domain.

Modern approaches to self-driving systems heavily rely on end-to-end networks since this technique has shown to be effective in translating the raw sensor data into useful information for the system's planning module. These approaches can be divided into two categories: The first category relies on three main tasks: Object detection, object tracking, and object motion prediction. The second category removes the need for the object tracking component and solves the problem at hand using only object detection and motion prediction. In this dissertation, end-to-end methods are covered in § 2.3.1 and § 2.3.2 for first and second category respectively.

Different types of input representations are commonly used in end-to-end methods namely the voxelized representation and the HD maps. To encode the spatio-temporal sensor information, voxelization is a quantized representation of the 3D world which is easier to process with convolutions. It is a binary 3D voxel grid that indicates the occupancy of each voxel. Typically, the voxel tensor is four-dimensional where the first three dimensions represent the 3D spatial dimensions, and the fourth dimension encodes the temporal information. In addition to the voxelized representation, some approaches also used HD map information. "HD maps" or "High-Definition maps" is a term that defines the maps that are built for self-driving systems. They are highly precise inventories of all static assets including road lanes, boundaries, traffic signals...

2.3.1 End-to-End Detection, Tracking and Motion Prediction

End-to-End Detection, Tracking and Motion Prediction approaches are designed to perform simultaneous 3D detection, tracking, and motion prediction by exploiting raw spatio-temporal data

captured from the system's 3D sensors. Since the object detection and motion prediction tasks have already been tackled in the previous sections, this section will also cover a brief overview of the object tracking task. Object tracking consists of assigning unique IDs to the objects present in the scene throughout a sequence of frames. Starting with the initial frame, a unique ID is first created for each of the initial detections. Then, as the objects move, the assignments of unique IDs are maintained. This last step is called the association of the current object detections with the previous tracks. The advantage of including a tracking step into the pipeline is that it allows identifying the specific past trajectory of each current actor which helps to study the dynamics of their past movements to better predict their future trajectories.

Two variants of these methods exist, depending on the position of the tracking step in the pipeline. In the first variant, the tracking step is placed at the end of the pipeline where the outputs of both object detection and motion prediction are fed to the tracking to aggregate the information of current detections and future predictions of previous tracks (or also called past future predictions). For instance, when there is an overlap between a current detection and past future predictions, they are considered to be the same object. The goal of tracking in this pipeline is to help in scenarios where we have strong past predictions and no current evidence (in the case of occlusion or a false negative from detection for instance) or in the opposite situation, where we have strong current detection and no past predictions which is evidence for a new object. Fast and Furious (FaF) [79] is an example of the first variant. FaF proposed a one-stage detector which took as input a voxelized representation of the 3D space over several time frames and produced detections and short-term motion forecasting of the objects' trajectories into the future followed by a simple tracker to improve the performance of the two other tasks.

The second variant represents the methods that put the prediction module after explicit object tracking. The tracking module uses the detection outputs to track the objects. After identifying the different objects in the scene, each object trajectory representation is fed to the prediction module to estimate their future movements. PnPNet [80] follows this direction. It takes advantage of both the HD map and voxelized representation and outputs at each time step object tracks and their

future trajectories. They specifically introduced a novel multi-object tracker that performed online tracking directly from the detection output and suggested using trajectory level features for motion prediction. To store extracted features at previous time steps, they suggested using two explicit memories for global features maps and past trajectories which were updated dynamically at each time step. It is also worth to note that, in addition to CNNs, PnPNet also integrated an LSTM layer which was used to model the temporal dynamics of actor trajectories.

2.3.2 End-to-End Detection and Motion Prediction

These approaches are designed to perform simultaneous object detection and motion prediction. Specifically, they remove the need for an object tracking step without loss in performance. To achieve that goal, most of these methods paid a lot of attention to the meticulous design of their networks as the very architectural detail can play a crucial role in the effectiveness and robustness of their methods.

A key component for an effective multi-task learning method is the design of the backbone network that is needed to extract rich feature representation from the inputs. MotionNet [81] introduced a novel spatio-temporal pyramid network, which extracted deep spatial and temporal features through a series of hierarchical spatio-temporal convolutions. Following the spirit of recent studies, the suggested pyramid network replaced the bulky 3D convolutions with blocks of 2D convolutions (to operate on the spatial dimensions) followed by pseudo-1D convolution (3D convolutions of kernel size $k \times 1 \times 1$ operating on the temporal dimension) which allowed reducing the complexity of the model. The features extracted from the pyramid network were fed to three heads performing cell classification, motion prediction, and state estimation. It is worth noting that in MotionNet, all tasks were performed simultaneously, and they were only connected using the loss function during training where the outputs of cell classification and state estimation heads were used to regularize the motion prediction head output.

Furthermore, in addition to the backbone network, some approaches included a jointly trained trajectory refinement step that helped improve the trajectory prediction. MultiXNet [82] is a multi-

class multimodal end-to-end network that adopted a refinement step. MultiXNet was built upon IntentNet [83], prior work on actor detections and motion prediction. IntentNet was an anchor-based object detection and motion prediction that output uni-modal future trajectories. MultiXNet extended it by transforming the uni-modal motion prediction component into a multi-modal multi-class motion prediction step.

With the new trend of transformers and their applications in multiple research areas, some approaches tried to leverage these techniques for motion prediction (see § 2.2.2 for more details about transformers). Li *et al.* [84] adopted a classical deep CNN detector and tackled the task of motion forecasting by introducing a novel transformer architecture called "Interaction Transformer". The latter was an adaptation of the original architecture, which was designed for sequence modeling, to the task at hand. First, the Interaction Transformer adopted a pairwise attention mechanism to capture the spatial relative positions of actors. Furthermore, it integrated the temporal information captured using a recurrent structure that predicted the actors' motion in an auto-regressive fashion. In addition to the initial motion prediction, a per-time-step refinement step was adopted to further refine the initial estimates.

2.4 Conclusion

For decades, perception tasks such as object pose estimation and motion prediction have been the center of interest of many researchers as these tasks play a crucial role in AI systems by helping them translate the received percepts of the environment into useful cues for their planning and control modules. More recently, the researchers' interest in these tasks has been buoyed by the success of deep learning technologies in many fields. In object pose estimation, researchers have tried different directions to solve the task at hand: Some tried to convert the pose regression task into a classification to a set of possible poses. Others tried to learn a key-point correspondence between the 2D image of the object of interest and its corresponding 3D model, and, some others tried to solve the pose regression directly in one shot from the input image. In addition to the main object pose estimation module, many works suggested a refinement step to further improve

the pose estimation performance. In this chapter, a brief overview of the different approaches and techniques used to solve object pose estimation and refinement was given.

In motion prediction, multiple deep learning techniques were experimented with such as CNNs, RNNs, TFs, and GNNs. One major key for an effective motion prediction module is the understanding of the dynamics of the agent's motion history. However, with the advance of research in this topic, many studies have shown that treating the agent as an independent entity and only focusing on its history is not sufficient and generally yields to sub-optimal performance. To overcome this issue, many works have studied the effect of social interaction on the future movements of agents. Other works further included the effect of the environmental context on the future movements of agents. This chapter comprehensively analyzed the evolution of motion prediction approaches and the different factors that they highlighted as major keys for better performance. The absence of a common benchmark, to evaluate and compare the performances of the different methods, is still a problem in the motion prediction domain that needs to be overcome in the near future.

For a long time, these perception tasks were treated separately and a cascaded approach was used where the output of the first task was fed to the other. However, recent studies have shown that the cascaded approaches generally suffer from a major error propagation problem where errors from upstream processes propagated to the end of the pipeline causing, in some cases, catastrophic failures. As an alternative, end-to-end approaches have been proposed where all the tasks are jointly learned in a single network. Such approaches have shown to lead to a great performance improvement compared to their counterparts. In the third section of this chapter, existing works that jointly solved object detection and motion prediction were examined by considering two categories for those that account for a tracking step and those that do not. However, a comprehensive analysis of these methods and a thorough experimentation to measure the importance of the tracking module in the pipeline is still needed.

Chapter 3

Pose Estimation and Refinement from RGB Inputs

Accurate 6D object pose estimation is crucial for many real-world perception based applications, such as autonomous navigation systems, robot grasping, and augmented reality. This chapter will cover a novel end-to-end 6D pose estimation approach from RGB inputs that my colleagues and I have proposed. This work was published in WACV2021 [85] and won the "Best Student Paper Award". In Figure 3.1, an overview of the proposed approach is shown.

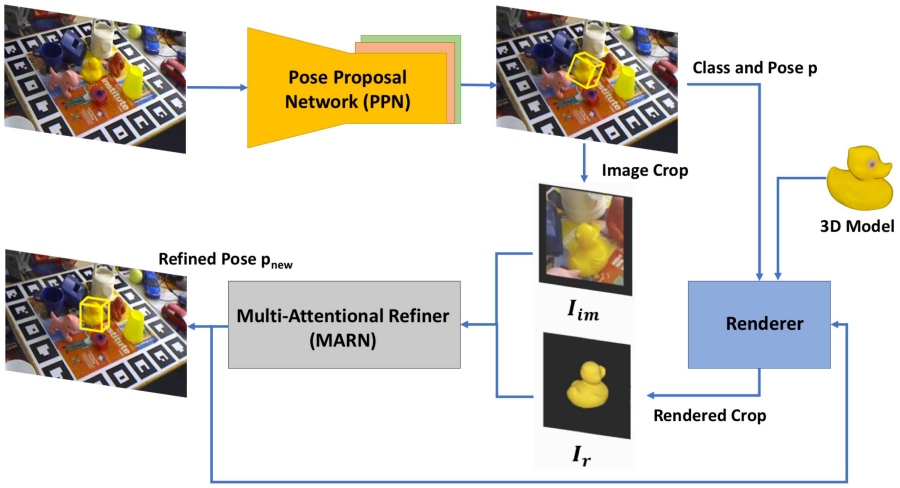

Figure 3.1: An overview of the proposed approach, consisting of a pose proposal module and a pose refinement module. The pose proposal module (PPN), outputs an object classification and an initial pose estimation from RGB inputs. The pose refinement module consists of a differentiable renderer and an iterative refiner called MARN. The renderer initializes the rendered crop of the detected object using its initial pose estimate and its 3D model. The refinement step utilizes a hybrid representation of the initial render and the input image, combining visual and flow features, and integrates a multi-attentional block to highlight important features, to learn an accurate transformation between the predicted pose and the actual, observed pose.

First, the Pose Proposal Network (PPN) extends the region proposal framework to classify and regress initial estimates of the rotations and translations of objects present in the RGB input.

Notably, the proposed PPN method requires no additional steps, unlike methods that use PnP [15] or matching with pre-engineered codebook.

Second, the pose refinement module consists of a differentiable renderer and a Multi-Attentional Refinement Network (MARN). MARN can be depicted as two main components: first, visual features from both the input crop and the rendered crop are fused using the flow vectors to learn better object representations. Second, a spatial multi-attention block highlights discriminative feature parts, insulating the network from adverse noise and occlusion effects. MARN is designed to allow iterative refinement; MARN outputs an estimated pose that directly maps to MARN's input. In the conducted experiments, the greatest performance gains typically occurred within a couple of refinement iterations. The entire pipeline is trained end-to-end and in Chapter 3.4, we will show that the full end-to-end model achieves state-of-the-art results on three separate datasets.

In the following, the 6D pose is represented as a homogeneous transformation matrix, $p = [R|t] \in \mathcal{SE}(3)$, composed of a rotation matrix $R \in \mathcal{SO}(3)$ and a translation $t \in \mathbb{R}^3$. R can also be represented by a quaternion $q \in \mathbb{R}^4$.

3.1 Pose Proposal Network

In this approach, the object pose estimation is reframed as a combined object classification and pose estimation problem, regressing from image pixels to region proposals of object centers and poses. Figure 3.2 illustrates the 6D object pose proposal network architecture. The architecture has two stages: first, a backbone encoder, modeled on the YOLOv2 framework [35], extracts high-dimensional region feature representations from the input image. Second, the obtained feature representations are embedded into low-dimensional, task-specific features extracted from three decoders which output three sets of region proposals for translations, rotations, and object centers and classes. It is worth noting that, similar to [31], the YOLOv2 framework (§ 2) was adopted to extract feature representations from the input image. However, the application of the YOLOv2 network in this work is fundamentally different in the sense that it serves only the purpose of extracting appearance features from the input image which will be used as input to the second stage

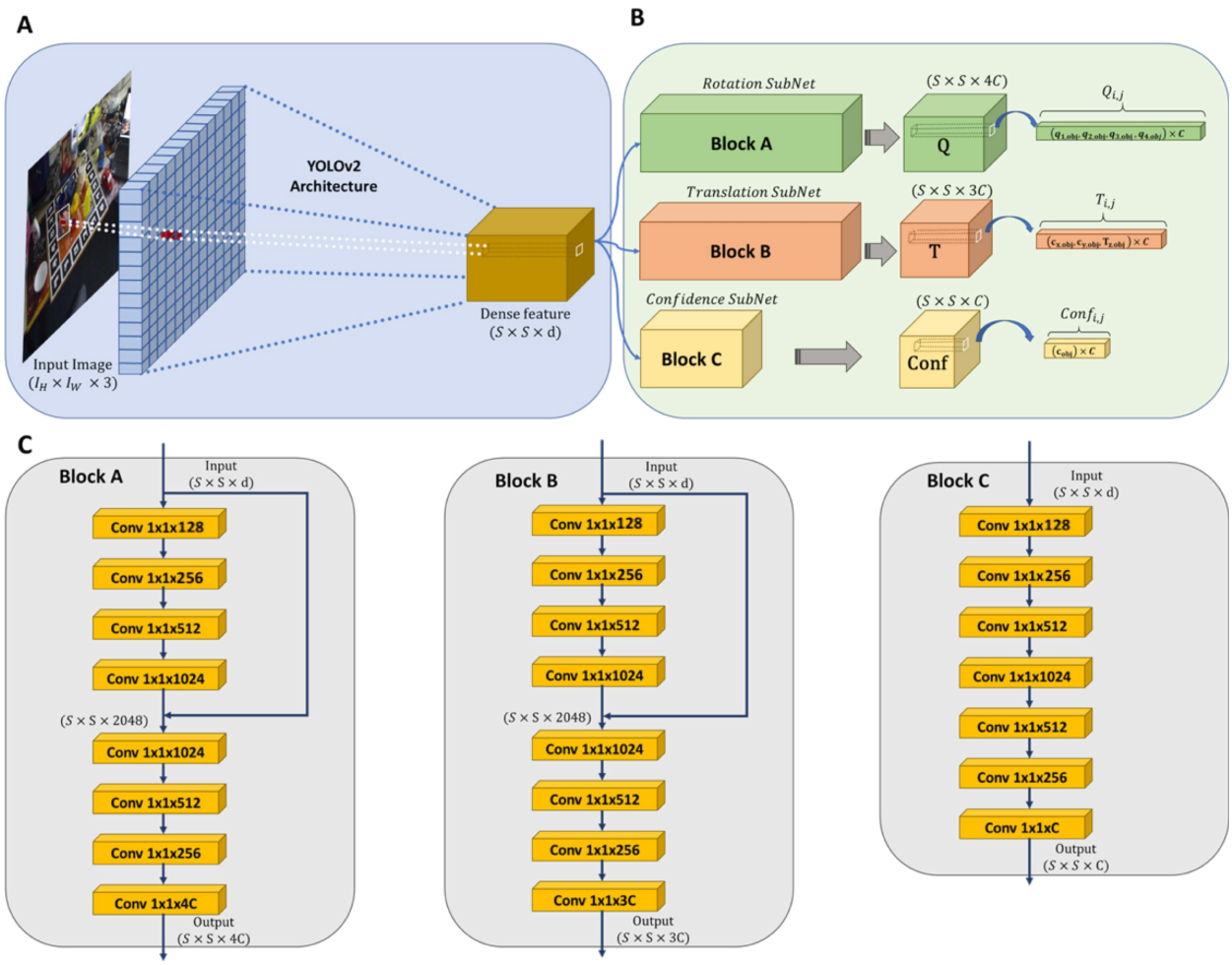

Figure 3.2: The Pose Proposal Network (PPN) Architecture. The encoder/multi-decoder network takes an RGB image, **A**. encodes it into high dimensional feature embedding of size d dividing the input image into a $S \times S$ grid, and **B**. decodes it into 3 task-specific outputs, which correspond to the rotation, translation, and confidence in the presence of the detected object. **C**. Architectural details of blocks A, B, and C in PPN

consisting of three decoders to ultimately estimate the objects poses. Specifically, the backbone encoder (Figure 3.2**A**) produces a dense feature representation $\mathcal{F}$ by dividing the input image into a $S \times S$ grid, each cell of which corresponds to an image block, that produces a set of high dimensional feature embeddings $\{\mathcal{F}_{i,j}\}$, with $\mathcal{F}_{i,j} \in \mathbb{R}^d$ for each grid cell $(i,j) \in \mathcal{G}^2$ s.t. $\mathcal{G} = \{1,\ldots,S\}$ and d is the embedding size. $\mathcal{F}$ is decoded by 3 parallel convolutional blocks (as shown in Figure 3.2**B**) that produce a fixed-size collection of region proposals $\{(Conf_{i,j}^{o_k}, T_{i,j}^{o_k}, Q_{i,j}^{o_k})\}$ for each object in the set of target objects $o_k \in \{o_1,\ldots,o_C\}$, where C is the number of target objects.

Block A: This block is a rotation proposal network that regresses a 4-dimensional quaternion vector $Q_{i,j}^{o_k}$ for each image region and object class.

Block B: This block is a translation proposal network that regresses a 3-dimensional translation vector $T_{i,j}^{o_k}$ for each image region and object class. Rather than predicting the full translation vector $T = [t_x, t_y, t_z]^T$, which can be cumbersome for training as discussed in [39], the object center coordinates in the image space $c = (c_x, c_y)^T$ is regressed and the depth component t_z. The two remaining components of the translation vector are then easily computed with the camera intrinsics and the predicted information:

$$\begin{aligned} t_x &= \frac{(c_x - p_x)t_z}{f_x}, \\ t_y &= \frac{(c_y - p_y)t_z}{f_y} \end{aligned} \tag{3.1}$$

where f_x and f_y denote the focal lengths of the camera, and (p_x, p_y) is the principal point offset. To regress the object's center coordinate, offsets for the 2D coordinates are predicted with respect to $(g_x, g_y) \in \mathcal{G}^2$, the top-left corner of the associated grid cell. this offset is constrained to lie between 0 and 1. The predicted center point (c_x, c_y) is defined as: $c_x = f(x) + g_x$ and $c_y = f(y) + g_y$ where $f(\cdot)$ is a 1-D sigmoid function.

Block C: This block is a confidence proposal network, which should have high confidence in regions where the object is present and low confidence in regions where it is not. Specifically, for each image region, Block C predicts a confidence value for each object class corresponding to the presence or absence of that object's *center* in the corresponding region in the input image.

Duplication Removal: After the inference of object detection and pose estimation, which is done by one pass through PPN, non-maximal suppression is applied to eliminate duplicated predictions when multiple cells have high confidence scores for the same object. Specifically, the inference step provides class-specific confidence scores, referring to the presence or absence of the class in the corresponding grid cell. Each grid cell produces predictions in one network evaluation, and cells with low confidence predictions are pruned using a confidence threshold.

Non-maximal suppression is then applied to eliminate duplicated predictions when multiple cells have high confidence scores for the same object and only consider the predictions with the highest confidence score, assuming either the object center lies at the intersection of two cells or the object is large enough to occupy multiple cells. The similarity of the projected bounding boxes of the 3D models given the predicted poses is specifically measured by computing the overlap score using intersection over union (IoU). Given two bounding boxes with high overlap score, the bounding box that has the lower confidence score is removed. This step is repeated until all of the non-maximal bounding boxes has been removed for every class. Two projections are considered to be overlapping if the IoU score is larger than 0.3.

3.2 Multi-Attentional Refinement Network

The proposed multi-attentional refinement network (MARN) iteratively corrects the 6D pose estimation error. Given the success of end-to-end trainable models [86, 87], my colleagues and I opted for an end-to-end refinement pipeline. Figure 3.3 depicts the MARN architecture and illustrates a typical refinement scenario. Two color crops (I_{im} and I_r), corresponding to an observed image and an initial pose estimate of the object in the image, are input into MARN, which outputs a pose residual estimate to update the initial predicted pose. This procedure can be applied iteratively, potentially generating finer pose estimation at each iteration.

3.2.1 Input Crops

Input Crops are sampled from a given predicted 6D pose p. Crops circumvent the difficulty of extracting visual features from small objects. Two crops, a rendered and an RGB, are generated. Images are cropped under the assumption that only minor refinements are needed. Both crops will be used as input to the refinement network. The rendered crop is generated by rendering the 3D object model viewed according to the predicted pose p. The RGB crop is generated from the original input image. A bounding box, that bounds the object's 3D model, projected on the image space using the predicted pose p is computed. The bounding box is then padded by $epsilon$

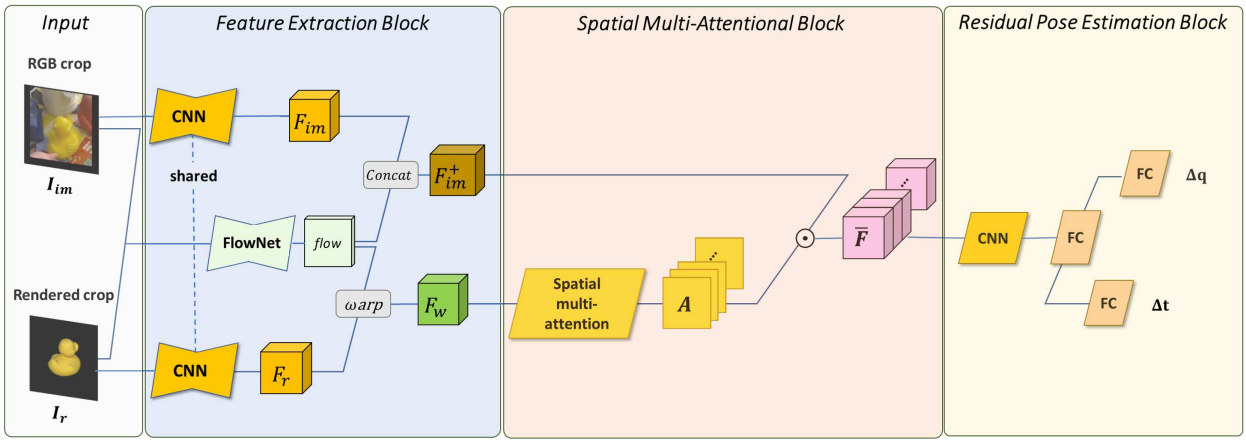

Figure 3.3: The proposed multi-attentional refinement network (MARN) takes a proposed pose and iteratively refines it. In the context of the proposed pipeline, the initial pose estimate, represented as a render image crop and a real image crop, are input into MARN. First, the network extracts visual feature representations from the inputs and an optical flow estimation between the two inputs (the Feature Extraction Block). Then, multiple attention maps, which correspond to different parts of the target object, are extracted from the flow and render crop features and applied to the feature representation of the real image crop, highlighting the important feature parts (the Spatial Multi-Attentional Block). Subsequently, the highlighted features are used to refine the pose estimate (the Residual Pose Estimation Block). The output refined pose estimate can be input into MARN for iterative refinement

pixels for each side to take into account the error introduced by the pose prediction. The enlarged bounding box is then used as a mask applied to the RGB image. Note that the mask cancels out the background, it does not crop the images. The images are cropped with a fixed size window $H \times W$, where the crop center corresponds to the object center, as defined by the 2D projection of the predicted pose p. Predicting $(\Delta c_x, \Delta c_y)$ consists of estimating how far the object center is from the image center.

3.2.2 Feature Extraction Block

MARN refines the estimated pose by predicting the relative transformation to match the rendered view of the object to the observed view in the original image. To this end, MARN's feature extraction block is composed of two different networks: 1) a visual feature embedding network that captures visual features of the object, and 2) a flow estimation network that estimates the object "motion" between the rendered image and the observed image. The network takes two input crops: $I_r \in \mathbb{R}^{H \times W \times 3}$ and $I_{im} \in \mathbb{R}^{H \times W \times 3}$. Both crops are processed through the shared visual

feature embedding network to extract visual feature representations $F_{im} \in \mathbb{R}^{H \times W \times d_{em}}$ for the image crop and $F_r \in \mathbb{R}^{H \times W \times d_{em}}$ for the render crop. Each pixel location of the embedding is a d_{em}-dimensional vector that represents the appearance information of the input image at the corresponding location. Simultaneously, the flow estimation network, based on the FlowNetSimple architecture [48], produces the optical flow between the rendered image and the observed image.

Subsequently, the visual feature map F_r, extracted from the render crop, is warped toward the visual feature map of the image crop F_{im}, guided by the flow information. Specifically, the warping function $\mathcal{W}$, extracted from the Flow estimation network, computes a new warped feature map F_w from the input F_r following the flow vectors $\text{flow}_{r \rightarrow im} \in \mathbb{R}^{H \times W \times 2}$:

$$F_w = \mathcal{W}(F_r, \text{flow}_{r \rightarrow im}) \tag{3.2}$$

Following [88], the warping operation is a bilinear function applied on all locations for each channel in the feature map. The warping in one channel l is performed as:

$$F_w^l(\boldsymbol{x_w}) = \sum_{\boldsymbol{x_r}} \mathcal{I}(\boldsymbol{x_r}, \boldsymbol{x_w} + \delta \boldsymbol{x_w}) F_r^l(\boldsymbol{x_r}) \tag{3.3}$$

where $\mathcal{I}$ is the bilinear interpolation kernel, $\boldsymbol{x_r} = (x_r, y_r)^T$ is the 2D coordinates in the visual feature embedding F_r, and $\boldsymbol{x_w} = (x_w, y_w)^T$ is the 2D coordinates in the visual feature embedding F_w. For backpropagation, gradients to the input CNN and flow features are computed as in [88]. Furthermore, the estimated optical flow $\text{flow}_{r \rightarrow im}$ is concatenated with the feature map extracted from the image crop F_{im} to produce $F_{im}^+ \in \mathbb{R}^{H \times W \times (d_{em}+2)}$.

3.2.3 Spatial Multi-Attention Block

Estimating an object's relative transformation between two images first requires successful localization of the target object within the two inputs. MARN handles this in the spatial multi-

attention block by localizing discriminative parts of the target object with spatial multi-attention maps, which robustly localize discriminative parts of the target. Therefore when the target is partially occluded, the multiple attention module can adaptively detect the visible parts while ignoring the occluded parts. Attention maps $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, where $a_i \in \mathbb{R}^{H \times W}$ for $i \in \{1, \ldots, N\}$ and N is the number of attention maps, are extracted by generating summarized feature maps $s_i \in \mathbb{R}^{H \times W}$ for $i \in \{1, \ldots, N\}$ by applying two 1×1 convolutional operations to feature map F_w, extracted by the feature extraction block. Each attention map $a_i \in \mathcal{A}$, corresponding to a discriminative object part, is obtained by normalizing the summarized feature map s_i using softmax:

$$a_i = \frac{\exp\left(s_i\right)}{\sum_{h=1}^{H}\sum_{w=1}^{W}\exp\left(s_{i,h,w}\right)}, \quad i = 1, \ldots, N \tag{3.4}$$

Finally, the attention map a_i and the feature map F_{im}^{+} are element-wisely multiplied to extract the attentional feature map $\bar{F}_i$:

$$\bar{F}_i = A_i \cdot F_{im}^{+}, \quad i = 1, \ldots, N \tag{3.5}$$

where $A_i \in \mathbb{R}^{H \times W \times (d_{em}+2)}$ is the replication of the attention map a_i, $(d_{em} + 2)$ times to match the dimensions of F_{im}^{+}. $\bar{F} \in \mathbb{R}^{H \times W \times (d_{em}+2)N}$ is the final extracted multi-attentional feature representation obtained by concatenating the attentional feature maps $\{\bar{F}_i\}_{i=1,\ldots,N}$. Inspired by [89], a regularization term is added to the total loss function to discourage multiple attention maps locating the same discriminative object part. The regularization emphasizes orthogonality among the attention maps:

$$\mathcal{L}_{orth} = \left\| \tilde{A}^T \tilde{A} - I \right\|_2 \tag{3.6}$$

where $\tilde{A} = [\tilde{a}_1, \ldots, \tilde{a}_N] \in \mathbb{R}^{HW \times N}$ and $\tilde{a}_i \in \mathbb{R}^{HW}$ is the vectorized attention map of a_i. By minimizing the orthogonal loss, the attention maps are constrained to focus on diverse object parts.

3.2.4 Residual Pose Estimation Block

This block processes the residual pose estimation. First, the embedding space of the extracted feature map $\bar{F}$ is reduced the from $(d_{em}+2)N$ to 8 with three 3×3 convolutional operations. The resulting feature map is then fed into one fully connected layer, whose output is then fed into two separate fully connected and final output layers, one corresponding to the regressed rotation and the other corresponding to the translation. As explained in §3.2.1, MARN outputs an estimated relative rotation quaternion $\Delta q \in \mathbb{R}^4$ and a relative translation $[\Delta c_x, \Delta c_y, \Delta t_z]^T$. The refined pose prediction is then computed with regard to the the initial pose prediction $\hat{p} = [\hat{R}|\hat{t}]$ using $c_{x,new} = c_x + \Delta c_x$, $c_{y,new} = c_y + \Delta c_y$, $\hat{t}_{z,new} = \hat{t}_z + \Delta t_z$, and $\hat{R}_{new} = \Delta R * \hat{R}$, where (c_x, c_y) is the center of the object in the image space using $\hat{p}$, $*$ is the matrix multiplication and ΔR is the relative rotation matrix obtained from Δq. $\hat{t}_{x,new}$ and $\hat{t}_{y,new}$ are then computed using (3.1).

3.3 Losses

In order to achieve accurate pose estimation, it is crucial to provide a criterion which quantifies the quality of the predicted pose. The different components of the proposed approach are trained jointly in an end-to-end fashion with a multi-task learning objective:

$$\begin{aligned} \mathcal{L}_{total} &= \mathcal{L}_{PPN} + \mathcal{L}_{MARN} \\ &= \alpha\mathcal{L}_{pose} + \beta\mathcal{L}_{conf} + \gamma\mathcal{L}_{ref} + \kappa\mathcal{L}_{orth} \end{aligned} \tag{3.7}$$

where α, β, γ and κ are weight factors. The proposed multi-task learning objective is composed of four loss functions. First, a composite $\mathcal{L}_2$ loss function to optimize the PPN pose and center detection parameters:

$$
\begin{aligned}
\mathcal{L}_{PPN} &= \alpha \mathcal{L}_{pose} + \beta \mathcal{L}_{conf} \\
\text{where} \quad \mathcal{L}_{pose} &= \underset{x \in \mathcal{M}_s}{\text{avg}} \left\| (Rx + t) - (\hat{R}x + \hat{t}) \right\|_2 \\
\text{and} \quad \mathcal{L}_{conf} &= \left\| \text{conf}_{gt} - \text{conf}_{pr} \right\|_2
\end{aligned}
\tag{3.8}
$$

where $\|\cdot\|_2$ is the L_2 norm. $\mathcal{L}_{conf}$ is the loss term used to train the confidence block. $\mathcal{L}_{pose}$ is the loss term used to train the pose regression. $\mathcal{L}_{pose}$ is similar to the average distance (ADD) measure (further discussed in § 3.4.2). $p = [R|t]$ is the ground truth pose and $\hat{p} = [\hat{R}|\hat{t}]$ is the estimated pose. $\hat{R}$ and R are the rotation matrices computed from the predicted quaternion $\hat{q}$ and the ground truth quaternion q, respectively. $conf_{gt}$ and $conf_{pr}$ are the ground-truth and the predicted confidence matrix, respectively. $\mathcal{M}_s \in \mathbb{R}^{M \times 3}$ is a set of points sampled from the CAD model. $\mathcal{L}_{pose}$ is only used for asymmetric objects. To handle symmetric objects, a modified loss function is instead used:

$$
\mathcal{L}_{pose,sym} = \underset{x_1 \in \mathcal{M}}{\text{avg}} \underset{x_2 \in M}{\min} \left\| (Rx_1 + t) - (\hat{R}x_2 + \hat{t}) \right\|_2 \tag{3.9}
$$

Second, MARN's loss function is defined as:

$$
\begin{aligned}
\mathcal{L}_{MARN} &= \gamma \mathcal{L}_{ref} + \kappa \mathcal{L}_{orth} \\
\text{where} \quad \mathcal{L}_{ref} &= \underset{x \in \mathcal{M}_s}{\text{avg}} \left\| (Rx + t) - (\hat{R}_{new}x + \hat{t}_{new}) \right\|_2
\end{aligned}
\tag{3.10}
$$

$\mathcal{L}_{ref}$ is the same loss term used in PPN. Symmetric objects are handled similarly to PPN. $\hat{R}_{new}$ and $\hat{t}_{new}$ are the refined rotation and translation estimates.

$\mathcal{L}_{orth}$ is a regularization term used to discourage multiple attention maps locating the same discriminative object part. The regularization emphasizes orthogonality among the attention maps as proposed by [89]:

$$\mathcal{L}_{orth} = \left\| \tilde{A}^T \tilde{A} - I \right\|_2 \tag{3.11}$$

where $\tilde{A} = [\tilde{a}_1, \ldots, \tilde{a}_N] \in \mathbb{R}^{HW \times N}$ and $\tilde{a}_i \in \mathbb{R}^{HW}$ is the vectorized attention map of a_i.

3.4 Experiments and Results

In this section, the introduced pose estimation approach (§ 3) is compared against state-of-the-art RGB-based methods across three datasets: YCB-Video, LINEMOD, and LINEMOD Occlusion. Results indicate that the proposed method outperformed other state-of-the-art methods on all datasets, with competitive runtimes. The remainder of this section will first briefly cover the benchmarks used to compare the proposed approach with state-of-the-art methods (in § 3.4.1) as well as the evaluation metrics (in § 3.4.2). Then, § 3.4.3 will provide details about the proposed architecture as well as the training process. Finally, § 3.4.4 will depict and analyse the obtained experimental results.

3.4.1 Datasets

This section will cover the different benchmarks that were used to run the experiments in order to objectively compare the performance of the different methods.

YCB-Video Dataset: [39] has 21 objects [90] across 92 video sequences. CAD models for all the objects are provided. YCB objects have varying shapes, sizes, and symmetries, levels of occlusion and lighting conditions of the scenes. In the conducted experiments, the data is divided as in [39], using 80 sequences for training and 20 sequences for testing. The training is augmented with 80k synthetically rendered images released by [39]. Pose predictions on the test set was refined with four MARN iterations.

LINEMOD Dataset: [20] contains 15,783 images of 13 objects, and includes CAD models of the different objects. Each image is associated with a ground truth pose for a single object of interest. The objects of interest are considered as texture-less objects, which makes the task

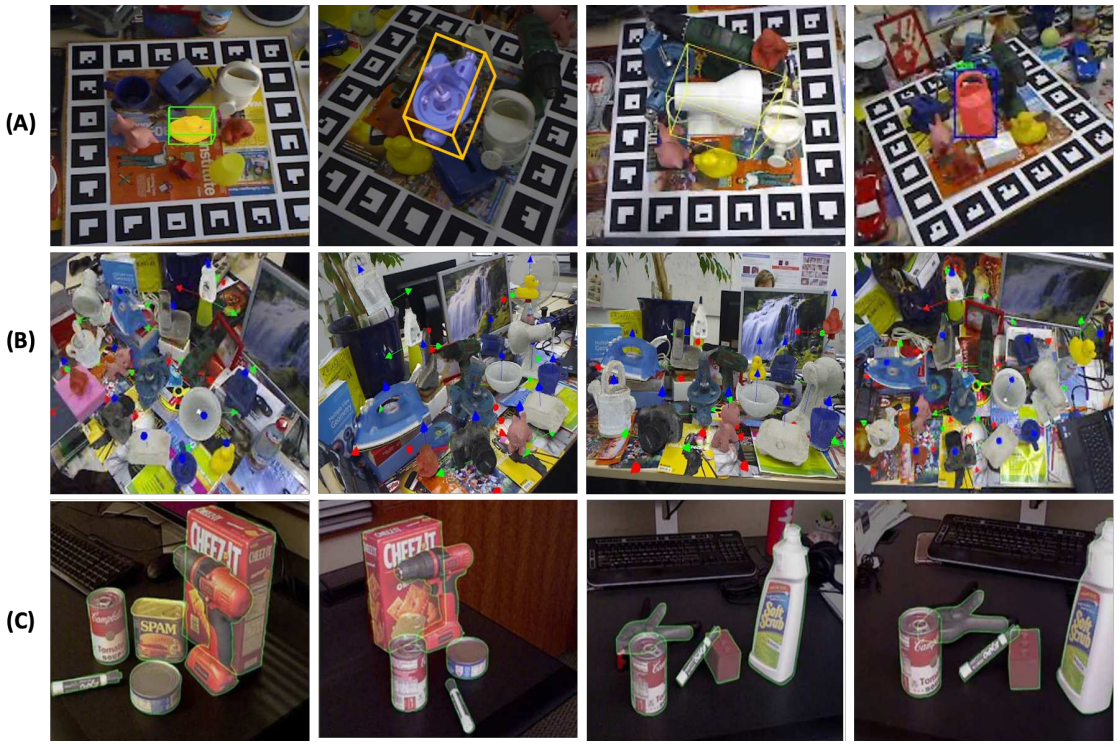

Figure 3.4: Popular public datasets for 6D pose estimation task that will be used in the experiments of this dissertation: **(A)** the LINEMOD dataset [20]. **(B)** the Occlusion dataset [49]. And **(C)** the YCB-Video dataset [39].

of pose estimation challenging. The train/test split is chosen following [91] — 200 images per object are used in the training set and $1,000$ images per object in the testing set. When using the LINEMOD dataset, online data augmentation is applied during training, to avoid over-fitting. Using this method, random in-plane translations and rotations are applied to the image along with random hues, saturations, and exposures. Finally, the images are modified by replacing the background with random images from the PASCAL VOC dataset [3]. Note that for testing on the LINEMOD dataset, two MARN iterations were used for refinement.

Occlusion Dataset: [49] is an extension of the LINEMOD dataset. Unlike LINEMOD, the dataset is multi-object — 8 different objects are annotated in each single image, with objects occluded by each other. The models are trained with the same online data augmentation procedure described in the LINEMOD dataset, further augmented by adding in image objects extracted from the LINEMOD dataset. Four MARN iterations were used for refinement on the Occlusion dataset. The Occlusion Dataset is particularly important because it tests the robustness of the pipeline to

occlusion, something the spatial multi-attentional block of MARN was explicitly designed to be robust to.

3.4.2 Evaluation Metrics

Two standard performance metrics are used for evaluating pose estimation models. First, the 2D-projection error, analogously to [32], measures the average distance between the 2D projections in the image space of the 3D model points, transformed using the ground-truth pose and the predicted pose. The pose estimate is considered to be correct if it is within a selected threshold. 2D-Proj denotes the percentage of correctly estimated poses using a 2D Projection Error threshold set to 5 pixels. For symmetric objects, the 2D projection error is computed against all possible ground truth poses, and the lowest value is used. The second metric, Average 3D distance (ADD) [20], measures the average distance between the 3D model points transformed using the ground-truth pose and the predicted pose. For symmetric objects, the closet point distance is used, referred to as ADD-S in [39]. In the conducted experiments, ADD(-S) is, following [39], the metric that measures the percentage of correctly estimated poses using a ADD(-S) threshold. Unless specified, in the experiments the threshold is set to 10% of the 3D model diameter. When evaluating on the YCB-Video dataset, the ADD(-S) AUC is also reported as proposed in [39].

3.4.3 Architectural and Training Details:

Below details about both the training procedures and system architecture are presented. These details specifically pertain to experiments which follow. The model was optimized with Adam optimizer with weight factors (α, β , γ, κ) set to (0.1, 0.05, 0.1, 0.01). These weights, as well as the other hyper-parameters of the model, were selected during the hyper-parameter selection step where a set of hyper-parameters were experimented with and the values yielding the best results in terms of the 2D-projection error metric were selected based on a 3-fold cross-validation. Given a 480×640 input image, PPN alone runs at 50 fps and the full model runs at 10 fps, with two refinement iterations, which is efficient for real-time pose estimation.

PPN:

The backbone encoder in PPN consisted of 23 convolution layers and 5 max-pooling layers, following the YOLOv2 architecture [35]. Additionally, a pass-through layer was added to transfer fine-grained features to higher layers. The model was initialized with pre-trained weights from YOLOv2, with the remaining weights being randomly initialized. Input images were resized to 416×416 and split into 13×13 grids ($S = 13$). The feature embedding size of the backbone network, d, was set to be equal to 1024.

The main architecture components were experimentally manipulated in order to test multiple architectural designs. Across all variations, the encoder remained unchanged. In the decoders, the number of layers and the number of channels were varied. We varied the number of layers for the translation and rotation decoders (block A and block B in Figure 3.2) with 4 layers (with decreasing number of channels from 1024 to the number of output channels), 8 (selected architecture) and 10 layers (with increasing number of channels from 128 to 2048 and then decreasing to the number of output channels). We also experimented with adding and removing the residual connections. Similarly, for the class confidence decoder (block C in Figure 3.2), we varied the number of layers with 4 (with decreasing number of channels from 1024 to the number of output channels), 7 (selected architecture) and 9 layers (with increasing number of channels from 128 to 2048 and then decreasing to the number of output channels C).

Initially, an additional weight factor is used, λ, applied to the confidence block output. Specifically, PPN was trained with λ set to 5 for the cells that contain target objects and 0.5 otherwise. This circumvents convergence issues with the confidence values because otherwise the early stages of training tend to converge on all zeros (since the number of cells that contain objects is likely to be much smaller than the cells that do not). In later training stages, λ was updated to penalize false negatives and false positives equally ($\lambda = 1$ for all cells). The number of points M, in the set of 3D model points $\mathcal{M}_s$, was set to $10,000$ points.

MARN:

For the visual feature embedding network, a Resnet18 encoder pre-trained on ImageNet is used followed by 4 up-sampling layers as the decoder. During training, the two networks were fine-tuned with shared weight parameters. The embedding size of the extracted features from the visual feature embedding network, d_{em}, is set to be equal to 32. The flow estimation network was the FlowNetS architecture populated with pre-trained weights following [48]. The network weights were frozen for the first two training epochs and unfrozen in later epochs. Once the weights were unfrozen, the component was trained in an end-to-end manner along with the other MARN components. The initial weight freeze increased training stability and ensured the output of the flow estimation network was meaningful. FlowNet output was up-sampled to match the input image crops. After a hyperparameter search, the padding offset for the mask ϵ was set to 10 pixels and the cropping window size was set to $H \times W = 256 \times 256$ applied to the original input image. Pose perturbations were used to create training data by adding angular perturbations ($5\,\mathrm{deg}$ to $45\,\mathrm{deg}$) and/or translational perturbations (0 to 1 relative to the object's diameter) to obtain a new noisy pose and rendering an image. The network was then trained to estimate the target output which was the relative transformation between the perturbed pose and the ground-truth pose.

3.4.4 Results on YCB-Video Dataset

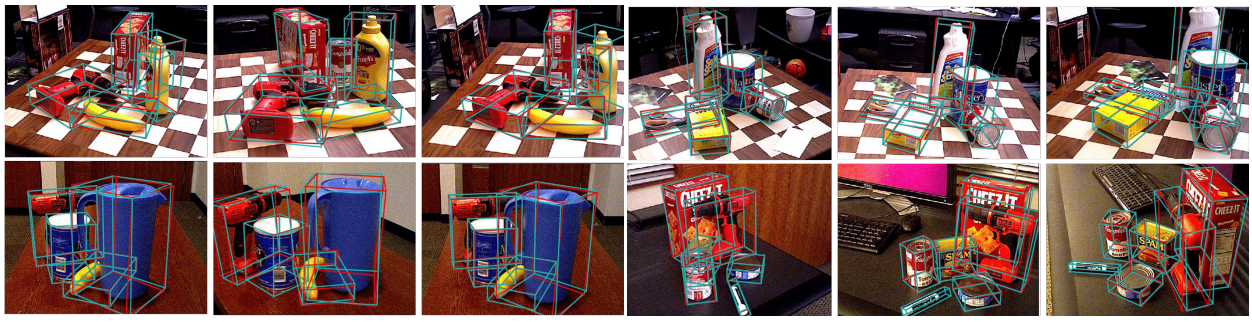

Figure 3.5: Pose estimation results using the proposed method on the YCB-Video Dataset. Cyan bounding boxes correspond to predicted poses and red bounding boxes correspond to ground-truth poses

Table 3.1: Comparison of the proposed approach with state-of-the-art RGB-based methods on **YCB-Video dataset** in terms of 2D-Proj, ADD AUC and ADD(-S) metrics, averaged over all object classes for each method. A threshold of 2 cm for the ADD(-S) metric is used.

Methods	HMap [36]	PVNet [32]	DeepIM† [45]	OURS†
2D-Proj	39.4	47.4	-	**55.6**
ADD AUC	72.8	73.4	81.9	**83.1**
ADD(-S) ($< 2cm$)	-	-	71.5	**73.6**

† denotes methods that deploy refinement steps.

Overall Results:

The results in Table 3.1 summarize the performance of existing RGB based methods compared to the proposed approach. The results of our approach represent the average of the 3 performance values reported from a 3-fold cross-validation. During cross-validation, we perform the train/test split on the sequence level so that the evaluation is performed on newly previously unseen sequences. The results suggest that the proposed approach significantly outperforms state-of-the-art RGB-based methods with an average 2D-Proj accuracy of 55.6% (average of the 3 values: 55.3%, 56.4%, 55.1%). The results in ADD AUC were averaged over the three values 82.8%, 83.4%, 83.1% and ADD(-S) averaged over the three values 73.1%, 74.2%, 73.5%. Compared to DeepIM [45], which also deploys refinement steps, the proposed approach achieves better performance by a margin of 1.2% and 2.1% in terms of ADD AUC and ADD(-S) respectively. Some examples of pose estimation results using the proposed approach on the YCB-Video dataset are provided in Figure 3.5.

Detailed Results on the YCB-Video Dataset:

Table 3.2 shows the detailed pose estimation results on the YCB-Video dataset [39] in terms of ADD AUC. The proposed approach achieves the best results in 12 object classes out of 21 compared to other methods. DeepIM, surpasses other methods on 6 object classes out of 21, and HMap outperforms other methods on 4 object classes.

Table 3.2: Detailed results of the proposed approach and other existing RGB-based methods on the different objects of the YCB-Video dataset in terms of ADD AUC

Methods	HMap [36]	PVNet [32]	DeepIM† [45]	OURS†
002-master-chef-can	**81.6**	-	71.2	72.1
003-cracker-box	**83.6**	-	**83.6**	81.7
004-sugar-box	82.0	-	94.1	**95.7**
005-tomato-soup-can	79.7	-	86.1	**88.2**
006-mustard-bottle	91.4	-	91.5	**94.8**
007-tuna-fish-can	49.2	-	87.7	**88.2**
008-pudding-box	**90.1**	-	82.7	80.2
009-gelatin-box	93.6	-	91.9	**94.5**
010-potted-meat-can	79.0	-	76.2	**82.6**
011-banana	51.9	-	**81.2**	78.7
019-pitcher-base	69.4	-	**90.1**	87.7
021-bleach-cleanser	76.1	-	**81.2**	78.1
024-bowl*	76.9	-	81.4	**83.4**
025-mug	53.7	-	81.4	**81.7**
035-power-drill	82.7	-	85.5	**87.8**
036-wood-block*	55.0	-	81.9	**83.7**
037-scissors	65.9	-	60.9	**67.4**
040-large-marker	56.4	-	**75.6**	71.1
051-large-clamp*	67.5	-	74.3	**75.2**
052-extra-large-clamp*	53.9	-	**73.3**	71.3
061-foam-brick*	**89.0**	-	81.9	82.2
MEAN	72.8	73.4	81.9	**83.1**

† denotes methods that deploy refinement steps.
* denotes symmetric objects.

Table 3.3: Results of the ablation study on different components of MARN on **YCB-Video dataset**. The same 2cm threshold for ADD(-S) is used. AUC means ADD(-S) AUC. Each variant was refined with 4 iterations

Experiments	flow vectors	visual features	Attention maps	ADD(-S)	AUC
Variant 1	None	✓	None	63.7	77.2
Variant 2	✓	✓	None	68.9	79.8
Variant 3	✓	✓	single	71.2	81.9
Variant 4	✓	✓	multiple	**73.6**	**83.1**

Ablation Study of The Refiner on YCB-Video Dataset:

This section covers an ablation study that was performed on MARN's components (detailed in § 3.2) to measure the effect of each of its components. In all, four variants are tested: In variant 1, MARN only uses visual features extracted from the two input crops. In variant 2, MARN uses the flow estimation features but not the attention component, instead fusing the extracted feature map F_{im}^{+} and the warped feature map F_w with simple concatenation. In variant 3, spatial attention is added, but only a single attention map is used. Variant 4 is the production variant of MARN. Each variant refined the pose 4 times. The results of the ablation study are presented in Table 3.3. First, these results demonstrate that variant 1 refinement, though the simplest, still improves the pipeline performance significantly by a margin ADD(-S) of 5.2%. This finding proves that visual features help in capturing the relative transformation between two inputs, and thus helps refine the pose. Variant 2, which adds in optical flow estimation improves the performance of the refiner by 2.3% over variant 1. This can be explained by the fact that the predicted flow ensures that the network learns to exploit the relationship between both crops and thus capture the relative transformation of the object between them. Variants 3 and 4 show that the addition of attention maps helps to improve the performance of the refiner. The improvement of variant 4 over variant 3 demonstrates that multiple attention maps help achieve better performance than a single attention map. Thus, these results prove the ability of multiple attention maps to capture various salient parts of the objects helps the model highlight important features, and makes the refinement process robust to various degrees of occlusion in the dataset.

3.4.5 Results on LINEMOD Dataset

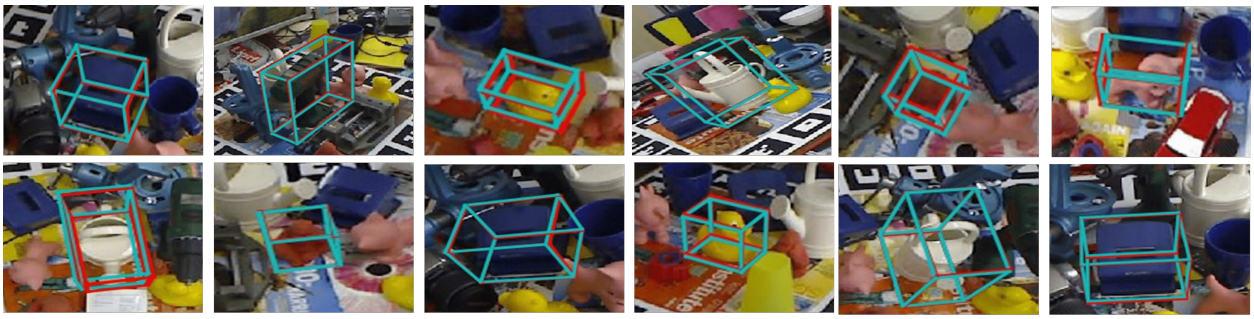

Figure 3.6: Pose estimation results using the proposed method on the LINEMOD Dataset. Cyan bounding boxes correspond to predicted poses and red bounding boxes correspond to ground-truth poses

Overall Results:

As presented in Table 3.4, the proposed approach achieves better results than other RGB-based methods in terms of ADD(-S), with an average accuracy of 93.87% accuracy compared to an average accuracy of 88.6% for DeepIM, the second best performing method. The results of our approach represent the average of the 3 performance values reported from a 3-fold cross-validation. In terms of ADD(-S) the reported values of our approach are 93.2, 93.69 and, 94.72. For the 2D-proj metric, the values are 99.32, 98.89 and, 99.36.

Some examples of pose estimation results using the proposed approach on the LINEMOD dataset are provided in Figure 3.6.

Table 3.4: Results of the proposed approach compared with state-of-the-art RGB-based methods on the **LINEMOD dataset** in terms of ADD(-S) and 2D-Proj metrics. Percentages of correctly estimated poses averaged over all object classes are reported.

Method	Tekin [31]	PVNet [32]	SSD6D† [27]	DeepIM† [45]	OURS†
ADD(-S)	55.95	86.27	79	88.6	**93.87**
2D-Proj	90.37	99.0	-	97.5	**99.19**

† denotes methods that deploy refinement steps.

Table 3.5: Detailed Results of the proposed approach and other existing RGB-based methods on the different objects of the **LINEMOD dataset** in terms of ADD metric

Method	Tekin [31]	PVNet [32]	BB8† [34]	SSD6D† [27]	DeepIM† [45]	OURS†
ape	21.62	43.62	40.4	65	77	**84.47**
benchvise	81.80	**99.90**	91.8	80	97.5	98.71
cam	36.57	86.86	55.7	78	93.5	**93.73**
can	68.80	95.47	64.1	86	96.5	**97.84**
cat	41.82	79.34	62.6	70	82.1	**87.33**
driller	63.51	96.43	74.4	73	95	**96.91**
duck	27.23	52.58	44.30	66	77.7	**88.45**
eggbox*	69.58	99.15	57.8	**100**	97.1	98.49
glue*	80.02	95.66	41.2	**100**	99.4	99.5
holepuncher	42.63	81.92	67.20	49	52.8	**84.53**
iron	74.97	98.88	84.7	78	98.3	**99.10**
lamp	71.11	**99.33**	76.5	73	97.5	98.74
phone	47.74	92.41	54.0	79	87.7	**92.53**
MEAN	55.95	86.27	62.7	79	88.6	**93.87**

† denotes methods that deploy refinement steps.
* denotes symmetric objects.

Detailed Results on the LINIEMOD Dataset:

In Table 3.5, the proposed approach is compared with existing state-of-the-art methods: Tekin [31], PVNet [32], BB8 [34], SS6D [27] and DeepIM [45] on LINEMOD dataset [20]. Compared with other methods, the proposed approach had the highest performance on 9 of the 13 object classes, PVNet had the best performance on 2 object classes, and SSD6D had the best performance on 2 object classes.

Ablation Study of the refiner on the LINEMOD Dataset

In Table 3.6, results of an ablation study on LINEMOD dataset are reported. The ablation study is similar to the one conducted on YCB-Video dataset. The results in Table 3.6 suggest that each component iteratively improves the refinement results, highlighting their effectiveness, but the full importance of each method may be somewhat muted, compared to the results on the YCB-Video dataset, since the experiment took place on the LINEMOD dataset, where accuracy is near the dataset ceiling.

Table 3.6: Results of the ablation study on different components of the refinement network MARN on the **LINEMOD dataset**

Experiments	flow features	CNN features	Attention maps	ADD	2D-Reproj
Variant 1	None	✓	None	87.32	96.59
Variant 2	✓	✓	None	89.17	97.99
Variant 3	✓	✓	single	91.28	98.56
Variant 4	✓	✓	multiple	**93.87**	**99.19**

Table 3.7: Comparison of the proposed approach with state-of-the-art RGB-based algorithms on **Occlusion dataset** in terms of ADD(-S) and 2D-Proj metrics. Percentages of correctly estimated poses averaged over all object classes are reported.

Method	HMap [36]	PVNet [32]	BB8† [34]	DeepIM† [45]	OURS†
ADD(-S)	30.4	40.77	33.88	55.5	**58.37**
2D-Proj	60.9	61.06	-	56.6	**65.46**

† denotes methods that deploy refinement steps.

3.4.6 Results on Occlusion Dataset

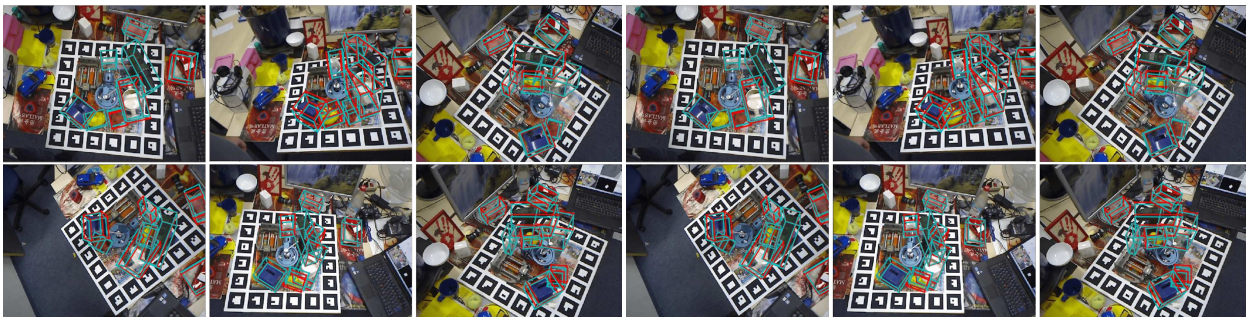

Figure 3.7: Qualitative pose estimation results using the proposed method on the Occlusion Dataset. Cyan bounding boxes correspond to predicted poses and red bounding boxes correspond to ground-truth poses

Overall Results:

Results in Table 3.7 indicate that, the proposed approach achieves significant improvements over all state-of-the-art RGB-based methods. Specifically, the proposed approach surpasses DeepIM by an ADD(-S) margin of 2.87% and PVNet by 17.6%. Furthermore, it significantly outperforms HMap, which was explicitly designed to handle occlusion, by an ADD(-S) margin of 27.97%. The

Table 3.8: Evaluation Results of PPN compared to other state-of-the-art RGB-based methods that do not use refinement on three datasets: **YCB-Video**, **LINEMOD** and **Occlusion** using the 2D-Proj metric

Methods	PoseCNN [39]	HMap [36]	PVNet [32]	PPN(ours)
YCB-Video	3.72	39.4	47.4	**49.3**
LINEMOD	62.7	-	**99.0**	96.12
Occlusion	17.2	60.9	61.06	**61.10**

significant improvement in performance on the Occlusion dataset, shows the importance of the different components of MARN, and mainly the spatial multi-attentional block, in robustly recovering the poses of objects under severe occlusion. It is worth noting that the performance of our approach is averaged over a 3-fold cross-validation process. In terms of ADD(-S) the reported values of our approach are 58.22, 58.31 and, 58.58. For the 2D-proj metric, the values are 64.87, 65.71 and, 65.80.

In Figure 3.7, examples of pose estimation results using the proposed approach on Occlusion dataset are given. Even when most objects are heavily occluded, the approach robustly recovers their poses.

3.4.7 PPN Only: An Efficient Pose Estimator for Real Time Applications

In this experiment, the performance of PPN, the proposed pose estimation network without refinement, is evaluated and compared with state-of-the-art methods that do not use refinement. Results in Table 3.8 on three benchmarks suggest that PPN alone performs better than HMap and PoseCNN on all three datasets, and performs comparably to PVNet.

Unlike these approaches, PPN has the highest speed (50 fps), is completely end-to-end, and does not require any additional steps such as the PnP algorithm. Thus, this suggests that PPN alone is fast and robust enough to be deployed in real-world applications.

3.5 New Capabilities for the Embodied Agent Diana: Enhanced Awareness of the Real World

This section aims to put the work on object detection and pose estimation described in the previous sections of this chapter together in a practical experiment. While previous sections summarize findings that are of general applicability, this section may be understood as an application summary to validate and demonstrate the proposed approach. The proposed object pose estimation approach has achieved great results and was able to beat the existing state-of-the-art methods on three commonly used benchmarks while running in real-time. For application purposes, the embodied agent Diana is used as a testbed where the new integrated features are expected to significantly enhance its capabilities to interact with the real world. Specifically, the integration of object detection and pose estimation component will allow Diana to be aware of the real world as she will now be able to “see" her surroundings, recognize the real objects present in the table in front of the user and estimate their poses. This capability can open up new areas of applications for Diana that have not been possible before.

3.5.1 The Embodied Agent Diana

The embodied agent Diana is a multi-modal interactive agent who exists in a virtual world built on the Unity game engine (see Figure 3.8). The agent can interpret multi-channel inputs including language, gesture, and emotion in real-time, and engage in collaborative interactions with a human. [6–8] Diana can recognize the gestures and audio she receives from the user and follows the instructions to manipulate virtual objects by grasping, lifting, moving, and sliding them in her virtual world. This world is accessible through a computer connected to a Kinect V2 camera providing both RGB and depth information and a microphone to allow audio inputs from the user. The embodied agent can assimilate up to 34 individual human gestures translating the user commands on manipulating the virtual objects and her ability to understand spoken language.

Diana integrates a multi-modal model of semantics on the software side, allowing her to identify human gestures through its real-time visual recognition system. In order to recognize the

Figure 3.8: A screenshot of Diana's world in Unity. Diana manipulates a set of virtual objects in front of her following the instructions of the user. We also show a human inset in upper right. The user gives Diana instructions whether through gestures or through vocal dialogue.

user's gestures in real-time, a convolutional neural network-based machine vision system is used to translate the sensors' inputs (RGB + depth) into helpful cues networked to the simulation environment for joint activity and human-agent communication. For speech recognition, a Visual Object Concept Modeling Language (VoxML [92]) is used as the platform for multi-modal semantic simulations in the context of human-computer discourse. Overall, agent-human communication involves integrating inputs from speech, gesture, and action, mediated through a dialogue manager (DM) that tracks communication and situation-based context variables embodied in a shared situated simulation. Hence, the obtained human-computer interaction consists more of a dynamic peer-to-peer conversation than giving and receiving orders.

A practical contribution of the work described in this dissertation is to experiment with integrating the proposed object detection and pose estimation model into Diana's system. With the new feature, Diana will not only receive commands from the user, whether through gestures or audio, but she will now be able to "see" the real-world, recognize the real objects present in the table in front of the user and estimate their poses.

3.5.2 Synthetic Dataset Generation

Motivation: A large dataset is a crucial component for training deep neural networks. Computer vision-related networks, in particular, require millions if not billions of images to reach their full potential. This constraint exists mainly because the learned features are automatically inferred from the input data, and thus, using small datasets will most likely lead to over-fitting problems. That being said, collecting and annotating enough data for generalization performance has always been a tedious and impractical task.

On the one hand, collecting a large dataset may be prohibitively expensive. Nowadays, the sizes of the available public datasets can give us clues on how challenging real dataset collection can be. For instance, popular public datasets in image recognition such as ImageNet dataset [93], where only a single label per image is needed, contains around 1.3 million images. Public datasets are far smaller for other tasks requiring more complex annotation, such as semantic segmentation. Microsoft COCO dataset [94] for instance, only contains 328 thousand annotated images. The more complex the annotation process is, the more challenging the real dataset collection can be. With tasks in specialized domains such as cancer segmentation in microscopic data, the number of images per dataset can drop to the order of thousands (example: the CAMELYON dataset [95] with 1,399 samples). All these examples can illustrate the difficulty in collecting enough valuable datasets for deep neural networks and machine learning in general.

On the other hand, dataset collection with precise annotation can be challenging and sometimes impossible for some tasks. Consider the object pose estimation problem, the main topic of this chapter, where the goal is to annotate a precise 6D pose of objects in the real world. Collecting real data for this task would require a particular setup such that the object position with respect to the camera is very well determined. One possible way is to stick the object to the center of a planar board with markers to provide the corresponding ground-truth poses. The camera moves within a hemisphere with precise distance from the center of the object and regularly samples viewpoints of the object. Such setup is challenging to establish and can also introduce inaccuracy to the data annotation with errors coming from the camera's movement or the ground-truth poses recovery.

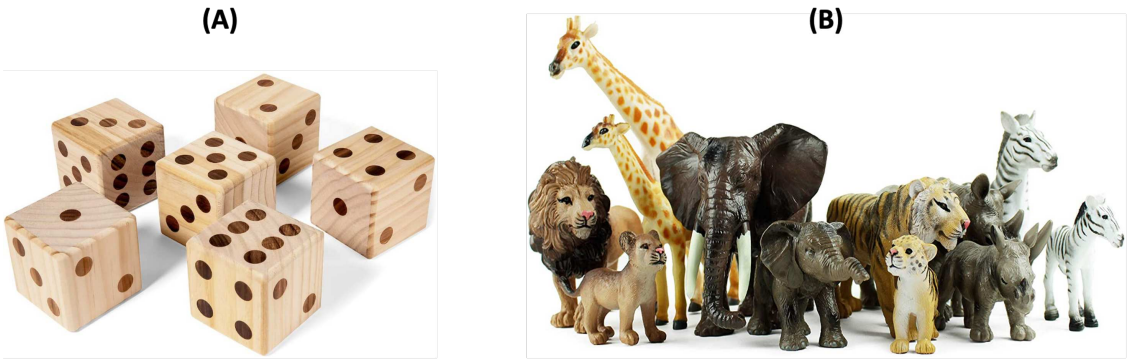

Figure 3.9: Two types of objects that we used to train our models. The first is a set of wooden dices that are identical. Though the dices are fairly simple objects, the network is expected to differentiate among the different faces of the cube given the number of dots they contain. The second set of objects consist of a set of animal toys. We have twelve different animal including giraffe, elephant, lion, tiger and others. The challenge with these objects is their relatively small size (the average height of these toys is 4 inches) and the absence of texture in some of them (such as the rhino and elephant).

Fortunately, many problems in computer vision, such as 6D pose estimation, have well-understood physical models. These physical models can be readily used to simulate the problem and generate synthetic datasets for training a machine learning model. Generating synthetic data offers a much safer and more efficient alternative to real data collection. Furthermore, with proper randomization, synthetic data can be generated in large quantities to satisfy the data requirements of modern machine learning algorithms. As a matter of fact, synthetic data has been used in many works and various disciplines, including the work described in this dissertation. In Figure 3.9, we show examples of objects that we have used to train our models.

Approach: Given an object of interest, a synthetic image can be generated by rendering its 3D model following a pre-generated random 6D pose. For this purpose, Pytorch3D [96] is selected as the tool to render the objects (this library has also been used in the pose refinement process in § 3.2). Pytorch3d is a python-scripted library that is fast and practical to use and can be easily integrated into the training pipeline. The integration of Pytorch3d allows us to have an online data generation that can be processed in real-time and online with the training process. As a result, training the model can consume as many training samples as it requires to converge as the training process allows an infinite number of training samples.

For the random pose generation, a set of conditions has to be satisfied. First, the poses are generated such that the objects are visible to the camera. In order to achieve that, the translation space has to be limited to the spatial area visible to the camera, which can be defined using the following conditions given a pose p=[R|t] = [R|(x,y,z)]:

$$\begin{aligned} &d_{zmin} \leq z \leq d_{zmax} \text{ where } d_{zmin} > 0 \text{ and } d_{zmax} > 0, \\ &-d_x \leq x \leq d_x \text{ where } d_x \geq 0, \text{ and} \\ &-d_y \leq y \leq d_y \text{ where } d_y \geq 0 \end{aligned} \tag{3.12}$$

where d_{zmin} is the minimum positive distance from the center of the camera on the z-direction at which the object is fully visible (in the field of view) to the camera. d_{zmax} is the maximum positive distance from the center of the camera in the z-direction. d_x (d_y respectively) is the maximum distance from the center of the camera on the x-axis (y-axis respectively) at which 80% of the object is visible in the field of view of the camera. We note that we did not restrain the rotation space as this allows the model to learn more details about the objects of interest and recover accurate 6D poses from any point of view.

In addition, the obtained rendered images are augmented with random shifting, flipping, and lighting (contrast, hue, saturation) variations, all while making sure the conditions in equations (3.12) are met. Furthermore, the background of the rendered images, which are initially dark backgrounds, are replaced with random real images from the PASCAL VOC dataset [3]. This augmentation technique will boost the robustness of the network to the noisy background and help it focus only on the objects of interest and ignore the background. In Figure 3.10 we give examples of training images where the objects of interest are wooden dices.

Despite the usefulness of synthetic datasets, making sure that the generated data is representative of the real world remains challenging. For example, if the generated training images are pixelated, the machine learning model will not generalize well to real world images. Hence, building high-quality and detailed 3D models of objects is a sensitive and crucial step in data generation.

Figure 3.10: Examples of training images using the Dice objects. A random pose is first generated such that the object lies within the field of view of the camera. The object is then rendered using the genrated pose. The process is repeated for each object. Finally the background is replaced with a random image from the the PASCAL VOC dataset [3]. Furthrmore, an image augmentation is applied where a variation of spatial offset, lighting, contrast and saturation is applied. The number of objects per image can vary from 1 up to 5 objects.

The higher quality the 3D models are, the more realistic the training images can be, and the less domain gap between natural (inference) and synthetic (training) images the model has to handle. Different modeling tools have been used depending on the complexity of the 3D reconstruction of the objects of interest.

To build the 3D models of objects with relatively simple geometries (such as dices in Figure 3.9.**A.**), Blender [97] was used as the modelling tool. First, the 3D model geometry is constructed by specifying the parameters and dimensions of the object. Then, real objects' textures are applied by painting the 3D models using texture images obtained from the real objects. For more complex objects (such as animal toys in Figure 3.9.**B.**), manual reconstruction can be difficult and sometimes impossible. In order to build realistic high definition 3D models, it is important to capture even the small details of the object of interest. Thus manual reconstruction may prove impractical for complex objects. Consequently, we opted for Qlone [98], which is an online 3D reconstruction tool that builds 3D models by scanning the actual object from different angles. Qlone is an all-in-one 3D-scanning application that provides a functional interface that allows the user to scan real objects using a phone's camera, modify them, and export the result to 3D file formats.

Ultimately, the goal is to run the new pose recovery capability in real-time alongside the other capabilities of the embodied agent. Thus, additional work has been conducted on integrating the pose estimation script into Diana's system. First, the pose estimation models receive image streams

from the agent's RGB sensor (Kinect V2) of size 1920×1080 and process them to output the classes and estimated poses of the objects present in the scene. Then the output is transferred to Diana's interactive module.

3.5.3 Experiments and Results

The generated dataset is used to train the network to estimate real object poses. The data generation process is integrated and runs online within the training scheme, and thus we have unlimited training data. Additional 5000 samples are also generated for the validation process. As for the testing phase, the models are tested on real images where a user manipulates a set of objects. If trained properly, the network is expected to recognize the actual objects and accurately recover their poses.

Experimental Details

Similar to the initial pose estimation experiments, the models were optimized with Adam optimizer with weight factors (α, β , γ, κ) set to (0.1, 0.05, 0.1, 0.01). The model takes as input image resized to 416×416 and then split it into 13×13 grids. The original image size of the input stream is 1920×1080, and resizing it to 416×416 might lead to the loss of important appearance information. Thus, we crop the input image by removing the border pixels which do not contain useful information for our task. The cropping step reduces the input size to 1400×925, then further resized to 416×416. An additional weight factor is used, λ, applied to the confidence block output. Specifically, PPN was trained with λ set to 5 for the cells that contain target objects and 0.5 otherwise. This setting circumvents convergence issues with the confidence values because otherwise, the early stages of training tend to converge on all zeros (since the number of cells that contain objects is likely to be much smaller than the cells that do not). In later training stages, λ was updated to equally penalize false negatives and false positives ($\lambda = 1$ for all cells). The number of points M, in the set of 3D model points $\mathcal{M}_s$, was set to $10,000$ points.

Similar to our previous experiments, three standard performance metrics are used for evaluating pose estimation models: the 2D-projection error, which denotes the percentage of correctly

Table 3.9: Overall performance of the pose estimation approach on the validation set of the two synthetic datasets: Dice dataset and Animal toys dataset. We report the same evaluation metrics as in previous experiments (ADD(-S), ADD AUC, and 2D-Proj). We also report the classification accuracy on the Animal toys dataset to assess the model's capacity to differentiate among different object classes.

Dataset	ADD(-S)	ADD AUC	2D-Proj	Accuracy
Dice Dataset	95.6	87.2	99.3	-
Animal Toys Dataset	89.36	85.18	90.01	81.2

estimated poses using a 2D Projection Error threshold set to 5 pixels, the Average 3D distance (ADD), which measures the percentage of correctly estimated poses using the average distance threshold between the 3D model points transformed using the ground-truth pose and the predicted pose (set to 10% of the 3D model diameter), and the ADD(-S) AUC. Further details about the different metrics were treated in § 3.4.2.

Synthetic Datasets Results

The results in Table 3.9 suggest that the proposed approach accurately predicts the 6D poses of objects in both experiments. The model performance is higher in the first experiment since it includes only a single type of objects (Dices), achieving an ADD(-S) of 95.6% and a 2D-Proj of 99.3%. Furthermore, though the dices have similar looking faces, the network was able to extract discriminative patterns (e.g. dots, wood pattern) and accurately estimate the 6 degrees of freedom related to the dice's pose.

In the second experiment, the network is expected to differentiate among the different objects (i.e., animal toys) and extract helpful appearance information for each of the dataset's object classes to recover the poses of the different objects. Hence, we see a decrease in performance of 9.29% in 2D-Proj compared to the first experiment. Additionally, we report the classification accuracy to evaluate the ability of the network to differentiate among the different classes. We obtain an accuracy of 81.2%, which proves the effectiveness of the network in the task of object classification.

We give some qualitative examples of pose estimation results using the proposed approach on real images in Figure 3.11. Examples **3-B** and **3-C** show the effectiveness of the model in

Figure 3.11: Qualitative pose estimation results using the proposed method on the Dice Dataset and the Animal Toys Dataset. First row shows qualitative results on the validation synthetic set. Cyan bounding boxes correspond to predicted poses. Row 2 and 3 show results on the test set of the Dice Dataset which contains images with real dices. Red bounding boxes correspond to predicted poses. Row 2 show images with only one dice while row 3 shows images with 2 dices. Row4 shows results on the Animal Toys Dataset with 2 objects. The color in row 4 designs a different object class and predicted class labels are annotated on the red squares. Row 3 and 4 show the effectiveness of our model in handling multiple objects detection and pose estimation.

accurately estimating the poses of objects with partial occlusion caused by the neighboring objects. Examples **2-A**, **2-B** and **4-A** show the effectiveness of the model in handling partial occlusions caused by the user holding the objects.

It is also essential to highlight the decrease in the model's performance when qualitatively comparing the synthetic images to the real world images. Though the proposed simulation process can provide an infinite amount of data samples, it does not perfectly represent the natural world in both visual and physical properties, which causes this gap between both environments known as the "reality gap". Domain randomization techniques are popular solutions used in this work to bridge the gap between simulation and reality, such as lightning variations, background randomization, pose randomization. Though such techniques have helped decrease the gap, the qualitative results suggest that further efforts to improve our simulation process are needed to reduce this gap. Thus, future work can include experimenting with gap reduction techniques to improve our simulation process.

Detailed Results on the Animal Toys Dataset:

In Table 3.10, we summarize the detailed pose estimation results on the Animal toys dataset dissected by object class in terms of ADD(-S), ADD AUC, and 2D-Proj. Results are collected on the validation set. The proposed approach achieves the best results on the Parent Giraffe class with an ADD of 97.5%, followed by the Parent zebra class with 97.3%. The performance increase on these two classes can be explained by the rich textures of both the Giraffe and Zebra objects. The long neck and orange dotted skin of the giraffe class, or the zebra pattern of the zebra class are powerful appearance features the network relies on to detect and estimate the object's pose.

We note that objects in this dataset are relatively small in size, which explains the challenge in this dataset. This challenge is further highlighted when comparing the parents classes to the children classes, where the objects are even smaller. We can see a considerable decrease in performance of 10% on average. Thus we conclude that the network can have a decreased performance with small-sized objects as the object might not cover enough pixels in the input image. Hence, the network fails to extract useful appearance features for the pose estimation task.

Table 3.10: Detailed results of the proposed pose estimation approach on the validation set of the generated synthetic dataset Animal toys in terms of ADD, ADD AUC and 2D-Proj. The results are dissected by object class. Overall we have 12 different object classes

Objects	ADD(-S)	ADD AUC	2D-Proj
Parent Giraffe	**97.5**	**96.4**	**98.3**
Child Giraffe	95.2	93.8	95.8
Parent Zebra	97.3	95.9	98.7
Child Zebra	95.2	92.1	95.5
Parent Lion	92.7	87.9	93.4
Child Lion	78.5	69.2	78.2
Parent Elephant	93.5	88.7	94.1
Child Elephant	81.0	79.3	81.6
Parent Tiger	96.0	95.2	97.1
Child Tiger	79.9	72.6	80.7
Parent Rhino	89.4	83.8	89.9
Child Rhino	76.1	67.2	76.8
MEAN	89.36	85.18	90.01

3.6 Conclusion

In this chapter, we covered a novel end-to-end method for RGB-based 6D pose estimation. Specifically, the proposed end-to-end approach was mainly composed of two modules. First, PPN was a fully-CNN-based architecture that produced one-pass pose estimates. Second, MARN was a pose refinement network that combined visual and optical flow features to estimate accurate transformations between the predicted and actual object pose. Further, MARN utilized a spatial multi-attentional block to emphasize important feature parts, making the method more robust. The proposed full end-to-end model achieved state-of-the-art results on three popular benchmarks.

This chapter also included a practical application of the approach described in this dissertation, consisted of integrating the proposed object detection and pose estimation model into Diana's system. Diana is a multi-modal interactive agent who can conduct a realistic conversation with users thanks to her capabilities, including gestures and audio recognition, and the ability to reason and intelligently react. With the new feature, Diana not only receives commands from the user, whether through gestures or audio, but she is now able to "see" the real world, recognize the natural objects present in the table in front of the user and estimate their poses.

The model was trained on a synthetic dataset generated by randomly rendering 3D models of objects reconstructed with adequate 3D modeling tools. Results demonstrated the effectiveness of our model on accurately recovering the detection and pose estimation of objects in the real world in front of Diana. They also suggested a slight decrease in performance when switching from synthetic to real-world images, explained by the reality gap existing between simulation and reality. Thus, future work can improve our simulation process and experiment with multiple techniques to reduce this gap.

Chapter 4

Future Motion Prediction of Moving Actors for Autonomous Navigation Systems

Intelligent navigation agents depend upon a mixture of perception modules to achieve safe motion planning. Perception must unfold in highly uncertain, rapidly changing, and interactive environments shared with other dynamic agents. Planning focuses on the real-time, safe navigation of such an environment. In this chapter, the focus is on the two perceptive tasks of agent tracking and motion prediction. Typically, these two tasks are cascaded; agent tracking output feeds into motion prediction. Such cascaded approaches are usually highly affected by errors propagating from noisy components. For instance, errors propagated from a noisy tracking module can hinder the performance of the motion prediction and planning modules. Such problems can result in catastrophic failures as the system fails to recover from errors accumulated through the pipeline.

Despite the complexities of cascaded interactions, most works on these topics do not examine how errors propagate and affect downstream modules. This chapter will cover a study that asks a novel question: does the tracking system, a sub-component of the motion prediction, contribute to overall accuracy improvements in real-world settings. The goal of this study is to focus on the tracking module due to the propensity of noise in real-world environments, a reality of several common autonomous driving issues like heavy occlusion, crowded scenes, high inter-frame motion, and camera motion. Thus, this study sheds light on the effectiveness of three motion prediction modules under challenging conditions and evaluates the importance of tracking. Results prove that, tracking noise can have considerable impact on the performance of motion prediction models. In real world settings, we might face tracking noise due challenging conditions that is significant enough to hinder tracking-dependent motion prediction models. In that case, tracking-free motion prediction methods can achieve better performance than the models that use tracking information.

Motion prediction is an indispensable task for planning safe and comfortable maneuvers [99]. Recent works [70, 71] have highlighted two main factors that directly affect the agents future motion: The short term history of the agents movements and their interactions, and the scene context including road and crosswalk polygons, lane directions and boundaries, traffic lights, and other relevant map information. This task is specifically challenging due to the uncertainty of the future decisions of the agents, and it is seemingly intuitive predicting the future trajectories said agents is important. However under certain circumstances, the agent tracking can overly complicates the motion prediction task, and actually decreases performance in the substantial presence of real-world noise.

Three models are described in this work (in Figure 4.1) that utilize a Bird's Eye View (BEV) multi-channel input image representation that integrates both scene context, from a high definition map, and agents' motion history, obtained from a working object pose estimation module. All three models produce both multi-agent trajectory predictions and spatial uncertainty estimations. The baseline model is the tracking free model (§ 4.2). In order to evaluate the effect of the tracking module, the tracking information is integrated into two of the models. In one model, an LSTM embedding is integrated to represent the agent's past states based on its tracking information (§ 4.3). In the second model (further explained in § 4.4), the identity information obtained from a tracking module in the BEV input image is integrated using displacement fields (to the best of my knowledge, this input representation is novel in the task of motion prediction). The performance of the three models is evaluated by using a real-world tracker and conclude that (see § 4.6), in real-world settings, it is important to study the effect of tracking module on the motion prediction performance to avoid situations where the tracking module is a hindrance to the system's performance.

For the following experiments, a few assumptions were made. First, for all models, a high definition map $\mathcal{M}$ of an operating area, comprising road and crosswalk polygons, lane directions and boundaries and other relevant map information is provided. Second, all models also have a functioning pose estimation system ingesting the sensor data to detect and pose traffic actors.

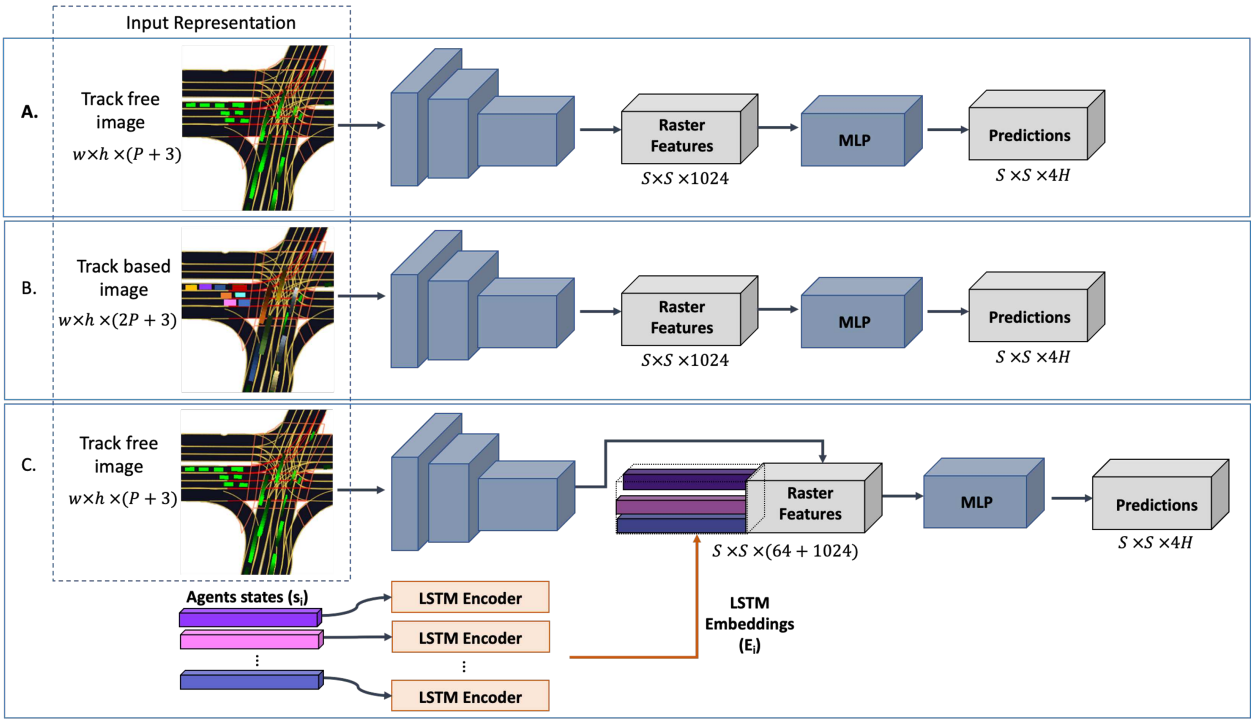

Figure 4.1: Overview of the three described architectures. The input representation is shown as rasterized into color-coded RGB image for visualization purposes. Each historical agent polygon is rasterized with the same color as the current polygon but with reduced level of brightness, resulting in the fading effect. **A.** represents the Track-free CNN method that relies on the tracking free input to predict the agents future trajectories. **B.** is the Track-based CNN method that integrates the tracking information in the input representation using displacement vector fields. **C.** shows the Hybrid method that extends **A.** by adding an LSTM encoding to represent each agent history. All agents in **A.** and **C.** inputs are represented with the same green color to show that no identity information was used to differentiate among agents. In **B.**, each agent is represented with different color to infer their identity information. The actual size of the input representation is $w \times h \times (nP + 3)$ where P is the number of past frames and n is equal to 1 or 2 depending on the architecture.

Lastly, unless specified differently, a perfect tracking system is available for the tracking-based models, providing ground-truth tracking of the detected traffic actors.

4.1 Input representation

The static map elements from the high definition map $\mathcal{M}$ are encoded in a bird's eye view (BEV) image centered on the self-driving vehicle (SDV) where each element of the map, including driving lanes, crosswalks and traffic lights, is encoded as a binary mask in its own separate channel. These channels are then rasterized into an RGB image where each element is assigned a different color, as described in [68]. Furthermore, P additional channels stacked with the map raster are considered where each channel represents the agents locations at each timestep of the history and present. Each of these channels is a binary mask encoding the agents top down positions in the same BEV frame as introduced above. The final input is then formed of $P+3$ channels representing map information and agent's history and present locations. It is important to note here that no identity information is inferred as all detections of agents at each timestep are treated similarly.

4.2 Tracking free CNN model

In this model, the input multi-channel image is processed, following [82], using a sequence of 2-D convolutions to produce a dense feature representation for each grid cell of the input. Three 1×1 convolutional layers are further added to finally output a 3-D tensor of size $S \times S \times 4H$ representing the predicted future movements of the agents present in the scene, where $S \times S$ is the size of the grid and H is the number of future predictions. For each grid cell containing an agent center at the present timestep, the 2-D centers offsets $(\Delta c_x, \Delta c_y)$ are predicted in H future time horizons. In addition to predicting the future trajectories, the spatial uncertainties of the predictions are also estimated. Similarly to [82],the centers location uncertainty is decomposed in the along-track (AT) and cross-track (CT) directions [100] and the uncertainty in each direction is assumed to follow an increasing linear function with time where the function parameters are model hyper-

parameters (see § 4.5 for more details). Note that this model utilizes no prior identity information nor tracking step.

4.3 Hybrid Model

The hybrid model is an extension of the first model (§ 4.2) which further integrates an LSTM sequence model [70]. The LSTM encodes each agent's states across past and present timesteps into a single embedding (E_i) for agent i. In the conducted experiments, an agent state s_t comprises position displacements with respect to the present, relative position changes, and speed at each timestep t where $t \in 1, ..., P$ and s_1 represents the present state. For each agent, the LSTM embedding is concatenated with CNN features extracted from the CNN network (as in first model) at the grid cell containing the agent's center at the present timestep. The grid cells that do not contain agent centers are padded with zeros. The obtained feature block is then processed, similarly with the first model, with three 1×1 convolutional layers and output a tensor of size $S \times S \times 4H$ representing the future trajectories and the corresponding uncertainties.

Note that the use of LSTM to encode an agent's past trajectory relies on the assumption that the identity of the agent is well known through time.

4.4 Tracking based CNN Model

This model follows the same architecture as the first model (§ 4.2). The main difference resides in the input representation; in this model, the identity information is integrated in the input image. Specifically, instead of representing each timestep from the past with a binary mask to indicate the presence/absence of a detection at each pixel, a spatio-temporal displacement vector field $D \in \mathbf{R}^{w \times h \times 2}$ is considered at each timestep, where a 2-D vector at each pixel parallels the vector from the agent center at that timestep, to the center of the agent at the present timestep. w and h are the width and height of the input image. At the present timestep, a simple binary mask is used, similar to the initial input representation. The final input image then has $2P + 3$ channels.

Like first model, this model relies on CNNs to operate on the spatial and the temporal dimension simultaneously and thus it is smaller in size compared to the hybrid model that uses both CNNs and LSTMs (§ 4.3). Though displacement vector fields are a common representation in the segmentation task [101, 102], to the best of my knowledge, the application of this technique in the motion prediction task is novel.

4.5 Loss Function

For the three models, both trajectory prediction and uncertainty estimation are trained jointly. The prediction errors are projected on the along-track (AT) and cross-track (CT) directions using the ground-truth heading of agent, and each projected error in one of the two directions is assumed to be independent from the other and follows a Laplace distribution $Laplace(\mu, b)$ with a PDF of a random Laplacian variable v computed as:

$$\frac{1}{2b}\exp(-\frac{|v-\mu|}{b}) \tag{4.1}$$

where mean μ and diversity b are the Laplace parameters. Ideally the AT and CT errors would follow a ground-truth distributions of mean $\mu = 0$ and diversities b_{AT} and b_{CT}, respectively. Since, the uncertainty is expected to increase with time, the diversity is defined as a linearly increasing function:

$$b_i = \alpha_i t + \beta_i \tag{4.2}$$

where α_i and β_i are model hyper-parameters defined separately for AT and CT. To train the model, the Kullback-Leibler (KL) divergence between the ground-truth distribution $Laplace(0, b_i)$ and the predicted distribution $Laplace(\hat{e}_i, \hat{b}_i)$ is minimized as in [82] defined as:

$$KL_i = \log(\frac{\hat{b}_i}{b_i}) + \frac{b_i \exp(-\frac{|\hat{e}_i|}{b_i}) + |\hat{e}_i|}{\hat{b}_i} - 1 \tag{4.3}$$

where i is whether AT or CT.

4.6 Experiments and Results

It is a common practice in the field of motion prediction to rely on the agent's past motion information to predict their future trajectory. Such approach makes a major assumption on the availability of a robust tracking system that provides little-to-no-noise identity information to the agents in the scene. However, this assumption does not always hold true in real world settings as the tracking system is always prone to noise. Numerous factors are directly related to the tracking system and might considerably affect its performance namely occlusion, crowded scenes, high inter-frame motion... In order to quantify the effect of such conditions on the performance of the tracking and thus motion prediction components, we conducted a set of exhaustive experiments on the three motion prediction models described in § 4.2, § 4.4, and § 4.3. These experiments aim to measure the effect of noise in the tracking information on the motion prediction models and compare their performances to tracking free methods under these settings. Ultimately, we aim to explore the importance of the tracking component for the performance prediction module in challenging scenarios.

For this purpose, this section describes the set of experiments conducted in the above-mentioned analysis. First, a summary of the set of public datasets used in these experiments is given (see § 4.6.1). Then a brief overview of the evaluation metrics that were used to evaluate the performance of the three described models is provided (see § 4.6.2). Further, the performance of the motion prediction methods described in § 4 is evaluated on both datasets in § 4.6.3 and § 4.6.4 where an exhaustive comparison of the models that integrate the identity information of agents to the models that do not is conducted. In § 4.6.3 (§ 4.6.4 respectively), an overall comparison of the models is conducted on the Lyft Dataset (Nuscenes dataset respectively). Section § 4.6.3 (§ 4.6.4 respectively) will further depict these results by considering scenarios where the knowledge of the agent identity may play a crucial role in the performance of the model prediction such as the case of crowded scenes on the lyft dataset (Nuscenes dataset respectively). Furthermore, the effect of noise in tracking on the performance of the models is evaluated by applying synthetic noise (§ 4.6.3, § 4.6.4) as well as realistic noise coming from real-world tracker (§ 4.6.3, § 4.6.4).

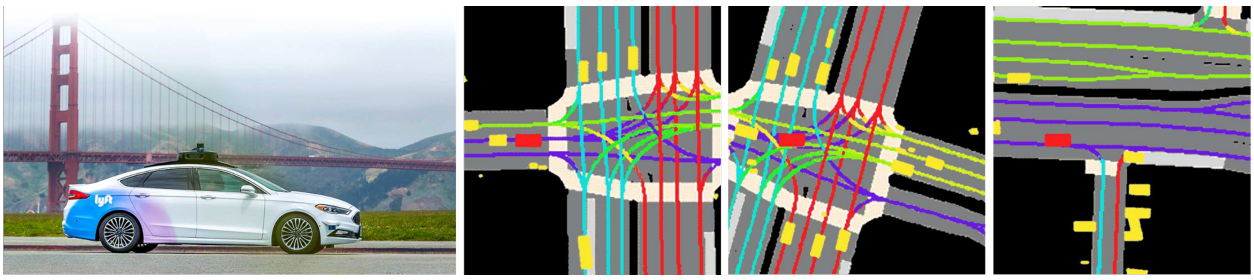

Figure 4.2: The Lyft Motion Prediction Dataset [51]

4.6.1 Datasets

It is important to note that the goal of the proposed work is to shed light on the different aspects of motion prediction models and factors influencing their performance, rather than comparing the performance of the proposed models with state-of-the-art methods. Two datasets are used in this work, the Lyft Prediction Dataset [51] and the Nuscenes Dataset [2].

The Lyft Prediction Dataset (Figure 4.2) is the largest public self-driving dataset for motion prediction to date, with 1,118 hours of recorded self-driving perception data. It was collected by a fleet of 20 autonomous vehicles along a fixed route in Palo Alto, California over a four-month period. It consists of 170,000 scenes, 25 seconds long each capturing the positions and motions of the surrounding agents including vehicles, cyclists and pedestrians. The dataset also comprises a high-definition semantic map with 15,242 labelled elements and a high-definition aerial view over the area.

The nuScenes prediction dataset (Figure 4.3) is an autonomous driving large-scale dataset that was collected within two distinct regions on different continents (Boston and Singapore), featuring trajectories from both left-hand and right-hand drive locales. The dataset contains a collection of around 40,000 scenarios that were extracted from 1,000 scenes in the two locations. Each scene has a length of 20 seconds captured with a frequency of 2Hz ($\Delta\, t = 0.5s$). The dataset provides annotations of up to 23 semantic object classes, as well as high definition maps with 11 annotated layers. We follow the official benchmark for the nuScenes prediction challenge to split the dataset. The training set contains 32,186 prediction scenes and the validation set contains 8,560 scenes.

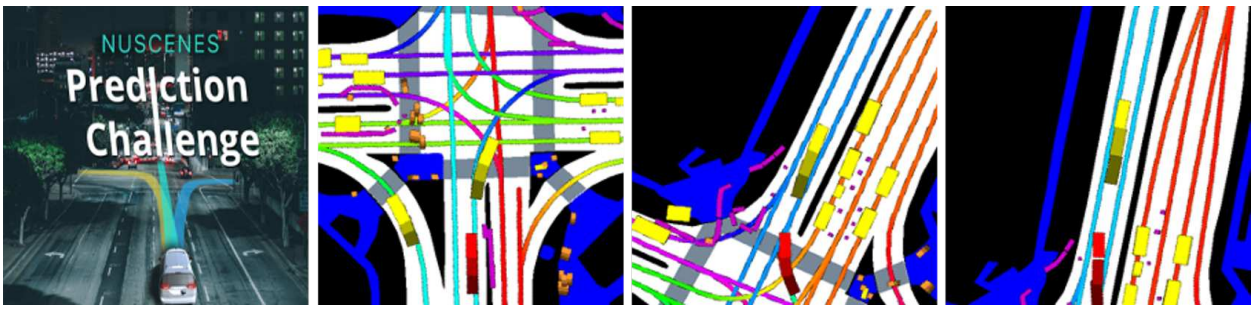

Figure 4.3: The nuScenes Prediction Dataset [2]

Due to the inaccessible ground truth of the test set of the prediction challenge, we rely on the validation set to evaluate the models for motion prediction.

4.6.2 Experimental Settings and Evaluation metrics

Throughout the conducted experiments in this chapter, we use a BEV image as the input representation with spatial horizontal dimensions 512×512, where each grid cell is $0.25m \times 0.25m$. For the temporal information, we consider a history of h seconds resulting in an input of size of $w \times h \times nP + 3$ where n is the number of channels per timestamp ($n = 1$ for both tracking-free CNN model and hybrid model and $n = 2$ for tracking based CNN model) and P is the number of frames considered in the input considering the historical frames as well as the present frame. The output tensor is of size $128 \times 128 \times d$ where $d = 4H$ channels.

For the backbone network, we use ResNet-50 [103] to extract deep features of size $S \times S \times 1024$. The models were implemented in PyTorch [104] and trained from scratch with a batch size of 4 with Adam optimizer [105], setting the initial learning rate to 10^{-4} that was further decreased by a factor of 0.9 every 200 thousand iterations. We ran our experiments on a Ubuntu server with a TITAN X GPU with 12 GB of memory.

In the case of the lyft prediction dataset, we consider $h = 1s$ of history which is equivalent to 10 past frames (the scenes were collected at a frequency of 10Hz) and thus $P = 11$ (10 past frames and 1 present frame). We chose to use 1s of history for real-time efficiency following [80, 82]. We aim to predict 5s into the future so we select a horizon $H = 50$. For the nuScenes prediction

Table 4.1: Overall comparison of the three described methods on the Lyft Prediction Dataset using four metrics in meters. Given a noise free tracking system, the Hybrid model performs the best in AT and FDE metrics, while the Track-based model performs the best in CT and ADE metrics.

Method	AT	CT	ADE	FDE
Track-free CNN	1.241	0.571	1.379	2.577
Track-based CNN	1.232	**0.549**	**1.328**	2.556
Hybrid	**1.229**	0.567	1.345	**2.552**

dataset, we consider $h = 3s$ of history which is equivalent to 6 past frames (the scenes were collected at a frequency of 2Hz) and thus $P = 7$ (6 past frames and 1 present frame). We chose to use 3s of history because the frames were collected at a low frequency and thus considering less history (1s for instance) will provide us with very less information that is not sufficient to give accurate predictions. Similar to the Lyft prediction dataset experiments, we predict 5s into the future so we select a horizon $H = 10$.

For our experiments, we report the along-track (AT) error metric and cross-track (CT) error metric [100] to measure the difference in the position of the predicted location and the actual location of the agent, projected onto the actual course at the present time. We also report the average displacement error (ADE) which is simply the mean l_2 distance between the ground truth and predicted trajectories, and the final displacement error (FDE) [59] which is defined as the l_2 distance between the predicted final position and the ground truth final position at the prediction horizon h. All metrics are reported on the validation dataset as specified in [51] and [2].

4.6.3 Experiments and Results on the Lyft Prediction Dataset

Overall Performance Evaluation

Results of the three models on the Lyft Prediction Dataset [51] are summarized in Table 4.1 with best prediction results highlighted in bold. The performance of the models is compared using 4 different metrics AT, CT, ADE and FDE (as introduced in § 4.6.2) averaged over a prediction horizon of 5s. It is worth noting that even small metric improvements can make a significant difference in the performance and safety of the real-world system.

Using the tracking information in both the Track-based CNN and Hybrid models improves the performance by 3% and 2.5% respectively, compared to the Track-free CNN model. Comparing the Track-based CNN and the Hybrid, the latter obtains better prediction accuracy on the AT and FDE metrics. This is unsurprising, as LSTMs are efficient in learning long-term temporal dependencies and thus can better capture the agent dynamics such as velocity and acceleration. Furthermore, the Track-based CNN model performs better than the Hybrid model in terms of CT and FDE. Thus, such model would perform better in lane association or in passing scenarios. In Figure 4.4 qualitative examples of 2 success cases and 2 failure cases for each of the three models described in this work are given.

Model Performance Depends on Agent Velocity and Traffic Density

Since the dataset encloses a variety of scenarios with large amounts of behavioural observations and interactions, it is hard to depict the effect of the tracking module by evaluating the full testing data. Based on preliminary studies, all three models have shown to perform equally well in the scenarios where agents are moving slowly or are stationary. The scenes are then categorized using the agent's velocity and the density of the scene and only scenes with agent velocities v larger than $3m/s$ will be considered in the future comparisons. Furthermore, the selected agents are categorized based on the density of their surrounding environment. The density is measured by calculating the radius between the agent and their nearest neighbour. A scenario is considered "dense" if the agent's radius is less than 4 meters ($r < 4$) and a scenario "non dense" if the radius is larger than 10 meters ($r > 10$).

The results are summarized in the Figure 4.5 which represents a bar plot with the performance of each scenario in ADE metric for each model. For dense and non-dense scenarios the performance change (in percentages) is specified with respect to all agents with $v > 3m/s$. First, as expected, the performance decreased in all three models since the selected scenarios are relatively challenging due to the high inter-frame motion and density of the scenes. Second, the track-free model significantly decreased in the performance compared to the tracking based models. Comparing to all moving agents ($v > 3m/s$), the performance of the track-free CNN model has decreased

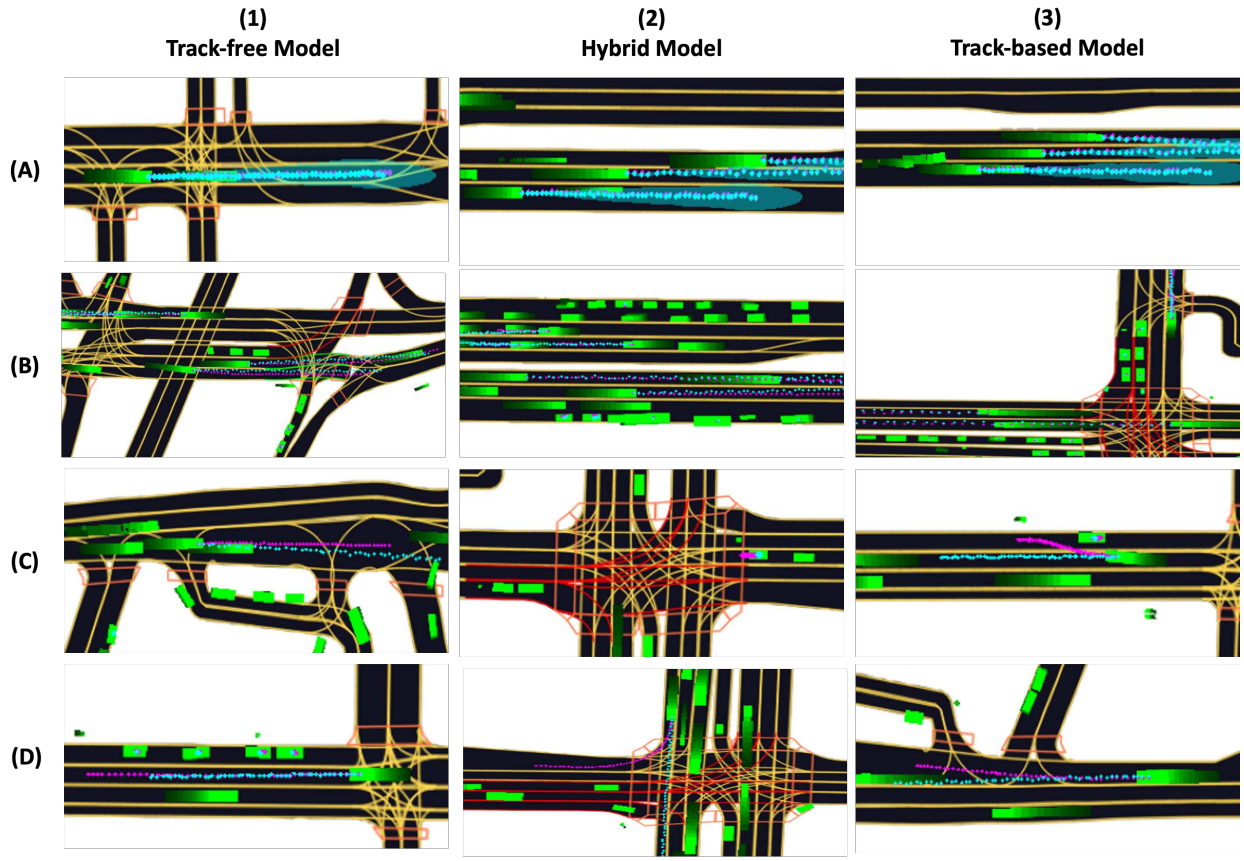

Figure 4.4: Examples of qualitative results of the described methods on the Lyft Prediction Dataset. The target trajectories are plotted in Magenta and the predicted trajectories in Cyan. For clearer visualization, The scenes are zoomed in and only the trajectories of a subset of the agents are shown. Columns **(1)**, **(2)**, and **(3)** show examples using the Track-free CNN model, Hybrid model and Track-based CNN model respectively. Rows **(A)** and **(B)** show success cases. In **(A)**, the uncertainties of the predicted trajectories are also plotted in light Cyan. Rows **(C)** and **(D)** show failure cases. Examples (1)-(C), (2)-(D), (3)-(C) and (3)-(D) show failure in the estimation of the future direction of the agents. High error is reported in the cross-track direction. (1)-(C), (1)-(D), (2)-(C) and (3)-(C) show failure in the estimation of the velocity of the agents. Thus high error is reported in the along-track direction.

by 12.6% on dense scenarios ($r < 4$) as compared to a more attenuated decrease of 8.8% and 8.5% for track-based CNN and hybrid models, respectively. On non dense scenarios ($r > 10$), the three models perform more comparably. These findings rightly demonstrate that the tracking information plays a crucial role in overcoming challenging scenarios, such as a very crowded scene where the input representation of the agents can become less effective. However, for other scenarios, such as non dense scenarios, the three models seem to perform comparably well. This is the noise-free condition — in the following sections, the three models are reevaluated in the context of tracking noise.

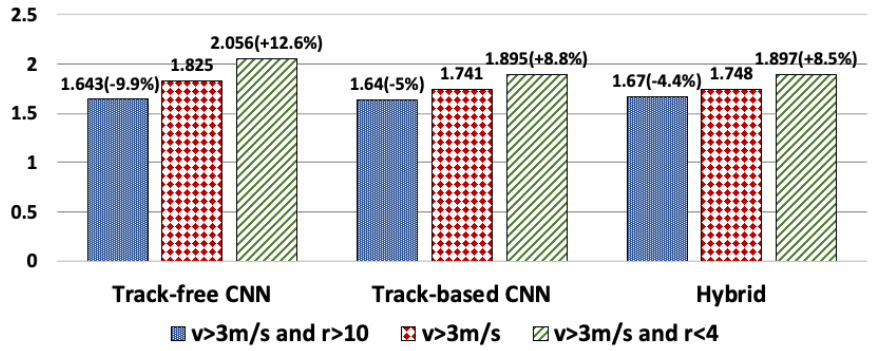

Figure 4.5: Performance evaluation of the described methods in ADE (m) on agents moving with a velocity larger than $3m/s$ on the Lyft Prediction Dataset [51]. The scene density factor was further considered. The radius r between a given agent and their closest neighbor was calculated and those with $r < 4m$ in one experiment (dense scenarios) and $r > 10m$ in a second experiment (non dense scenarios) were selected. The performance of the three methods degrade in dense scenarios. The decrease is most pronounced with Track-free model which shows the importance of tracking information under these conditions.

Performance Evaluation with Noisy Tracker

In these experiments, the effect of tracking noise on the performance of the three models is evaluated. Being independent from the tracking information, the performance of the track-free CNN model remains constant in these experiments. Synthetic noise was applied to the tracking system and its effect was evaluated on the performance of the two tracking based models. In the first set of experiments, summarized in Figure 4.6, random identity switches with varying chances per track were performed. The probability of an identity switch was varied from 0% to 20%

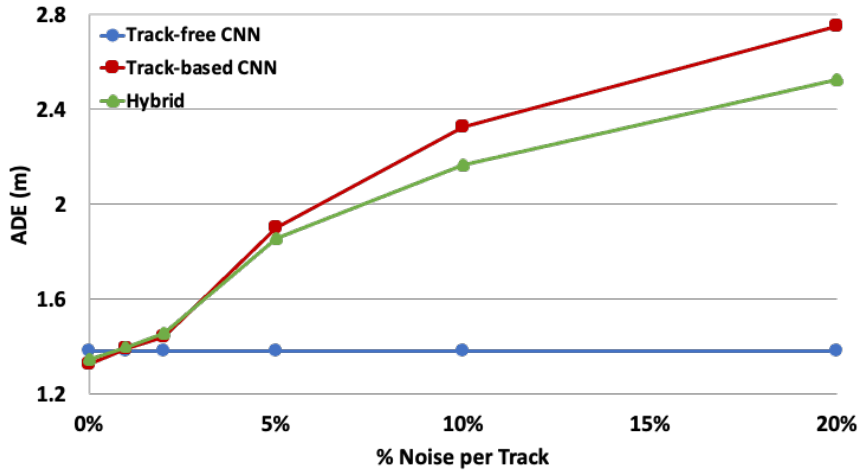

Figure 4.6: Performance Evaluation of the described methods in ADE (m) with synthetic noise applied to the tracking information on the Lyft Prediction Dataset [51]. An increasing chance of 1 identity switch per track was experimented with. The performance of the tracking based methods (track-based CNN and Hybrid) decreases with increasing tracking noise. The track-free model is not affected by tracking noise.

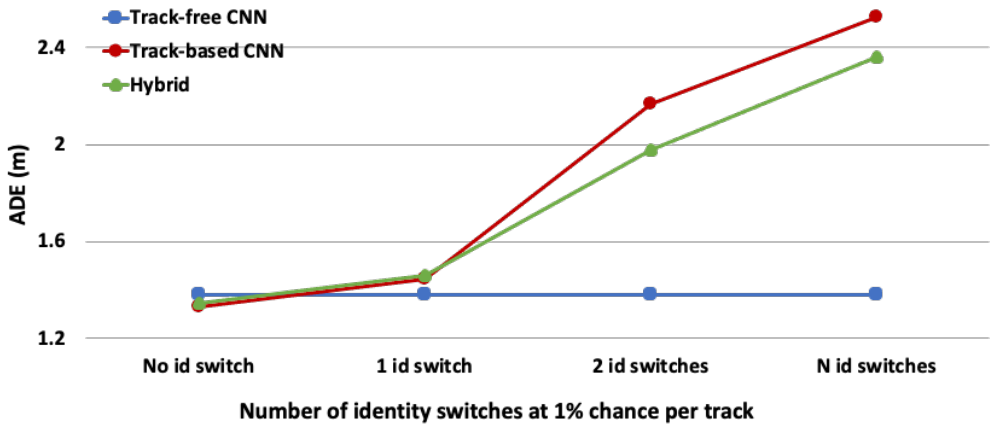

Figure 4.7: Performance Evaluation of the described methods in ADE (m) with synthetic noise applied to the tracking information on the Lyft Prediction Dataset [51]. Synthetic tracking noise of 1% chance and an increasing number of identity switches per track were experimented with. 1 id switch represent identity switch at only 1 timestamp. 2 id switches represent an identity switch for 2 consecutive timestamps and N id switch represent 1 identity switch that started at a random timestamp and continued until the end of the scene. The performance of the tracking based methods (track-based CNN and Hybrid) decrease with increasing tracking noise. The track-free model is not affected by tracking noise.

per track. The performance of both the track-based CNN and hybrid model has decreased with increasing noise, degrading slightly around 1% chance and then drastically after 2% chance of identity switch per track. Comparing the tracking based models to the track-free CNN model, the latter obtains better performance on all experiments with noise larger than 0.8%. The drastic decrease in performance of both the track-based CNN and Hybrid model reveals that they both rely heavily on the tracking information to capture the agents' past movements and thus at a certain level (around 0.8% noise chance), the cascaded tracking noise starts to negatively affect the performance of these models. With noise chance larger than 1%, the tracking noise has a significant negative effect on the performance of these models. For experiments with noise chance larger than 2%, the Hybrid model is clearly more robust than the Track-based CNN model. Though the Hybrid model is highly dependent on the LSTM input enclosing the tracking information, it also relies on the input 2D representation which is independent from the tracking module, which explains its relative robustness to noise compared to track-based CNN model.

The second set of experiments presented in Figure 4.7, comprises common identity switch scenarios. The identity switch chance was set to 1% per track throughout the experiments and three scenarios were considered: an identity switch happening at a single random timestamp (1 id switch), an identity switch happening for two consecutive timestamps (2 id switches), and an identity switch happening at a random timestamp and continuing until the end of the scene (N id switches). Similarly to the first set of experiments, the performance of both the track-based CNN and hybrid model declined in all three scenarios and fell behind the performance of the track-free model. This drop proves their high dependency to tracking. The Hybrid model, for instance, has decreased from 1.345 to 1.485 in ADE when applying 1 id switch with a 1% chance. It then fell by 35.5% and 59.1% when applying 2 id switches and N id switches respectively. Similar to the findings in the previous experiments, the Hybrid model is more robust to noise compared to Track-based CNN model when dealing with 2 id switches and N id switches.

Table 4.2: Overall comparison of the described methods using four metrics (m) using StanfordIPRL-TRI tracker [1] on the Lyft Prediction Dataset [51]. Real-world trackers introduce noise to the tracking information which affects the performance of track-based methods (Track-based CNN and Hybrid methods).

Method	AT	CT	ADE	FDE
Track-free CNN	**1.241**	**0.571**	**1.379**	**2.577**
Track-based CNN	1.268	0.607	1.478	3.013
Hybrid	1.263	0.611	1.485	2.987

Peformance Evaluation with Real-world Tracker:

In this section the performance of the tracking based models using a real-world tracker is evaluated. To this end, the tracking information provided in the dataset was replaced with the output of a real-world, popular tracker. The experiments conducted in the sections § 4.6.3 and § 4.6.3 on the tracking based models are then reproduced in this section. The StanfordIPRL-TRI tracker introduced in Chiu *et al.* [1] was used to run the experiments based on their publicly available code and parameters suggested in their work. The StanfordIPRL-TRI tracker won the nuscenes challenge competition [2] by achieving state-of-the-art results on the nuscenes dataset.

The results of the Track-based CNN and Hybrid models using the StanfordIPRL-TRI tracker are summarized in Table 4.2. The track-free CNN model does not depend on the tracking module so its performance remains the same as in Table 4.1. The results suggest that there is a slight drop in performance of the two tracking based models when using the StanfordIPRL-TRI tracker. Compared to the track-free CNN model, the track-based CNN model fell behind by 7.17% in ADE, 2.17% in AT and 6.3% in CT. Similarly, the Hybrid model dropped back by 7.68% in ADE, 1.77% in AT and 7% in CT with respect to the track-free CNN model. These findings suggest that state-of-the-art trackers can introduce noise that will be cascaded to the motion prediction module. If practitioners do not take preventive measures, the introduced noise can ultimately affect the motion prediction performance. These results also indicate that, in the case of noisy tracking information, the tracking free model performs better than the tracking based models which makes it a potential option to avoid cascaded noise. Alternatively, the motion prediction model can

be trained to overcome noisy inputs from the tracking step and thus robustly recovers accurate predictions even under challenging conditions.

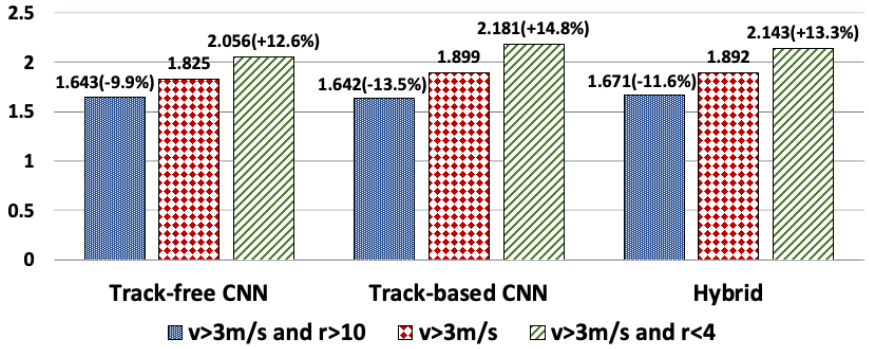

Figure 4.8: Performance evaluation of the described methods using StanfordIPRL-TRI tracker [1] in ADE (m) on agents moving with a velocity larger than $3m/s$ on the Lyft Prediction Dataset [51]. The scene density factor was further considered. The radius r between a given agent and their closest neighbor was calculated and those with $r < 4m$ in one experiment (dense scenarios) and $r > 10m$ in a second experiment (non dense scenarios) were selected. The performance decrease of the track-based methods is more pronounced using the real-world tracker due to the noise introduced to the tracking information.

Further experiments are conducted to evaluate the performance of models using the StanfordIPRL-TRI tracker on challenging scenarios, as described in § 4.6.3, where the agents that moved at a speed higher than $3m/s$ were selected. The dense-versus-non-dense scenarios were also considered where the closest neighbor to the agent was located at a radius less than 4 meters and larger than 10 meters, for dense and non dense respectively. Results of this experiment are outlined in Figure 4.8.

Results reveal that, similarly to the track-free CNN model, the tracking based models performance has dropped. The performance of the Track-based CNN model and Hybrid model dropped by 4% and 3.6%, respectively, compared to track-free CNN model. For the dense scenarios, the track-based CNN performance drops by 14.8% compared to the "all moving agents scenario" ($v > 3m/s$) (§ 4.6.3), while the track-free model has dropped by 12.6%. These results suggest that though the performance has decreased across the three models due to the complexity of the scenarios, the track-free model performs the best among the three models. Furthermore, the

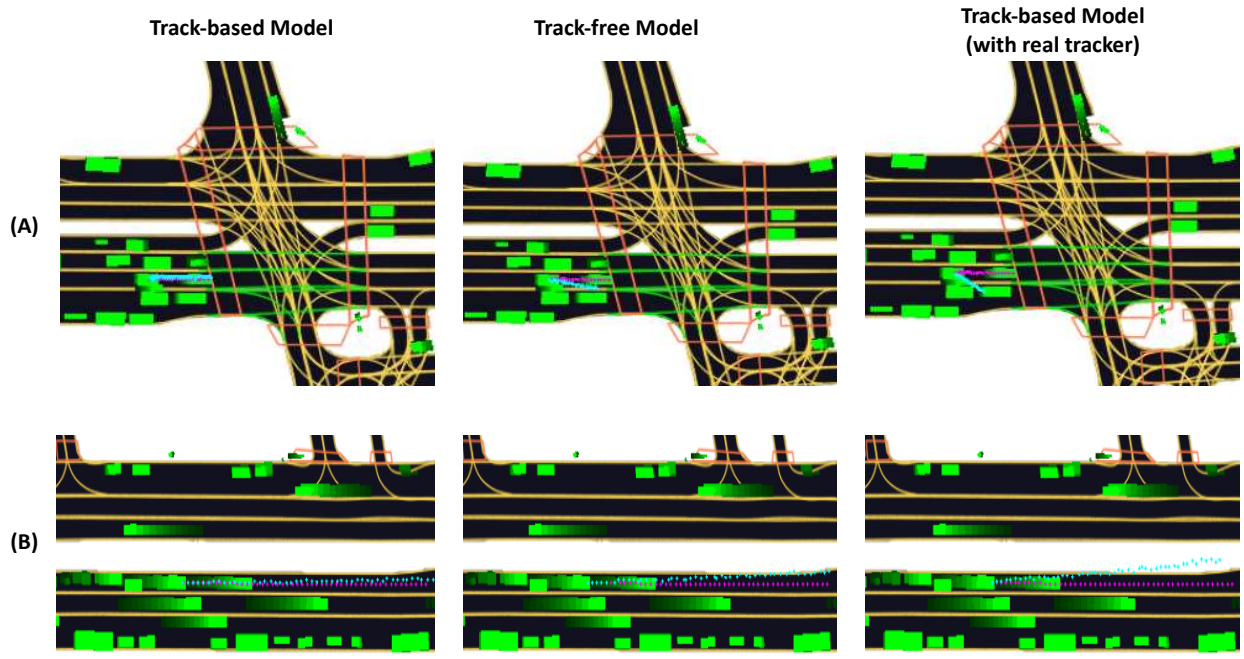

Figure 4.9: Qualitative evaluation on the Lyft Prediction Dataset [51] in the case of a crowded scene where an identity switch happened. The performance of the track-based CNN model using ground-truth tracking **(1)**, the track-free CNN model **(2)**, and the track-based CNN model using StanfordIPRL-TRI tracker **(3)**, in 2 examples **(A)** and **(B)** is shown.

Table 4.3: Overall performance comparison of the three described methods on the nuScenes Prediction Dataset using four metrics in meters (AT, CT, ADE, and FDE). Best prediction results shown in bold. Given a noise free tracking system, the Track-based model performs the best in all metrics, the Hybrid model performs the second best and the Track-free model performs the worst. Thus, tracking information does help improve the performance of the motion prediction module. The performance of the three models decreased on the nuScenes prediction dataset compared to the Lyft Prediction Dataset (see Table 4.1) which indicates that the scenes are more challenging in the nuScenes prediction dataset.

Method	AT	CT	ADE	FDE
Track-free CNN	2.879	1.684	3.798	6.377
Track-based CNN	**2.215**	**1.114**	**3.187**	**5.219**
Hybrid	2.435	1.283	3.316	5.447

decrease in performance is more highlighted in the more challenging scenarios ($v > 3m/s$ and $r < 4$) where the Track-based CNN model's ADE decreases to 2.181m as opposed to 2.056m for the track-free CNN model. This finding is unsurprising since the tracking based models are more affected by the tracking noise in more challenging scenarios where they heavily rely on tracking information.

In Figure 4.9, examples of challenging scenarios are highlighted, with crowded scenes where identity switches happened, and the performance of the Track-based CNN model, using real-world tracker [1], is compared with the Track-free CNN model and Track-based CNN model using the ground-truth tracking information (no identity switch for this model). Comparing the first and third row, the performance of the track-based model, using the real-world tracker, (third row) degrades compared to the track-based model using the ground-truth tracker (first row) in the presence of identity switches. Comparing the second and third row, both models do not perform well in the two proposed scenarios. However, the track-free model is more robust to crowded scenes.

4.6.4 Experiments and Results on the nuScenes Prediction Dataset

Overall Performance Evaluation

Results of the three models' performance on the nuScenes prediction dataset are summarized in Table 4.3 with best prediction results shown in bold. The models' performances are compared using four different metrics AT, CT, ADE, and FDE (as introduced in § 4.6.2) averaged over a

prediction horizon of 5s. It is worth noting again that even minor metric improvements in these experiments can make a significant difference in the performance and safety of the real-world system. Based on these results, similar conclusions to those on the Lyft Prediction Dataset can be deduced. Notably, both the Track-based CNN and Hybrid models outperform the Track-free CNN model on the nuScenes Prediction Dataset by 0.664m and 0.444m, respectively. These results further prove that accurate tracking is an essential factor for improving motion prediction performance. Comparing the models' results on the two datasets, all three models have decreased in performance on the nuScenes Prediction dataset compared to the Lyft Prediction dataset. This decrease can be explained by the low inter-frame rate and the more challenging scenarios (crowdedness, occlusion, variety of scenes) in the nuScenes dataset. Considering the tracking-based methods (Track-based and Hybrid), the Track-based CNN obtains better predictions by considering all metrics. This finding varies from the results on the Lyft Prediction Dataset. Though LSTMs are efficient in capturing the long-term temporal dependencies of the agents' motion, their integration in the Hybrid model has indicated minor performance improvement compared to the model integrating displacement fields (the Hybrid model improved the performance of the baseline by only 0.444m while the track-based model improved by 0.664m). Thus, this latter would perform better in more challenging conditions like those presented in nuScenes dataset. In Figure 4.10 qualitative examples on the nuScenes dataset of one success case and one failure case for each of the three models described in this work are given.

Model Performance Depends on Agent Velocity and Traffic Density

NuScenes Prediction is an exhaustive dataset that encloses a variety of scenarios and situations. Thus, further categorization of the different scenes is needed to thoroughly understand the effect of the tracking module on the performance. This section describes similar experiments to those detailed in § 4.6.3 on the Lyft Prediction Dataset where a categorization of the scenes is applied based on the agents' velocities and the crowdedness of the scene. First, all models have shown to perform considerably well when agents are moving slowly or are stationary, based on preliminary studies. Thus we will only focus on the cases where agents move faster than $3m/s$. Furthermore,

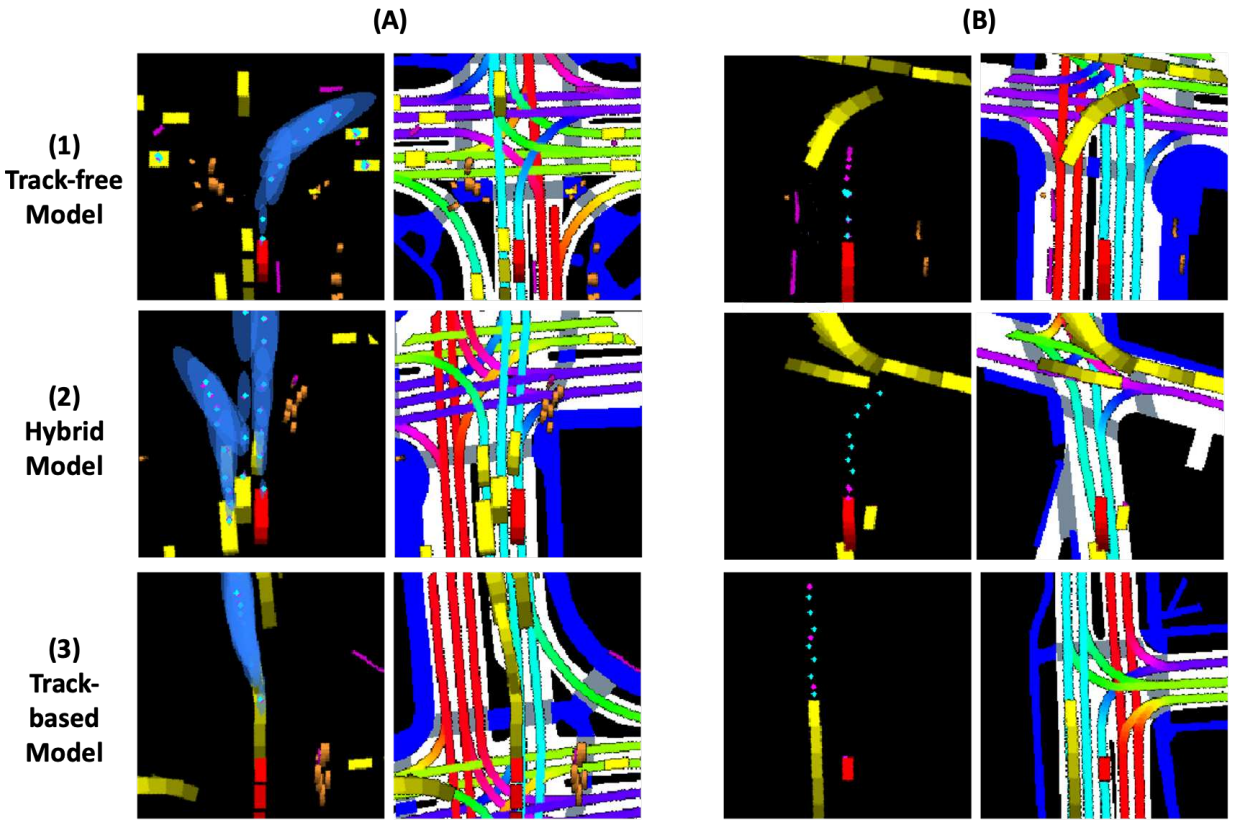

Figure 4.10: Results examples of the described methods on the nuScenes dataset. The target trajectories are plotted in Magenta and the predicted trajectories in Cyan. For clearer visualization, The scenes are zoomed in and only the trajectories of a subset of the agents are drawn. Rows **(1)**, **(2)**, and **(3)** are examples using the Track-free CNN model, Hybrid model and Track-based CNN model respectively. Column **(A)** shows success cases. The uncertainties of the predicted trajectories are also plotted in light Blue. Column **(B)** shows failure cases. (1)-(B), and (3)-(B) show failure in the estimation of the velocity of the agents. Thus high error is reported in the along-track direction. Example (2)-(B) shows failure in the estimation of the future direction of the agents. High error is reported in the cross-track direction.

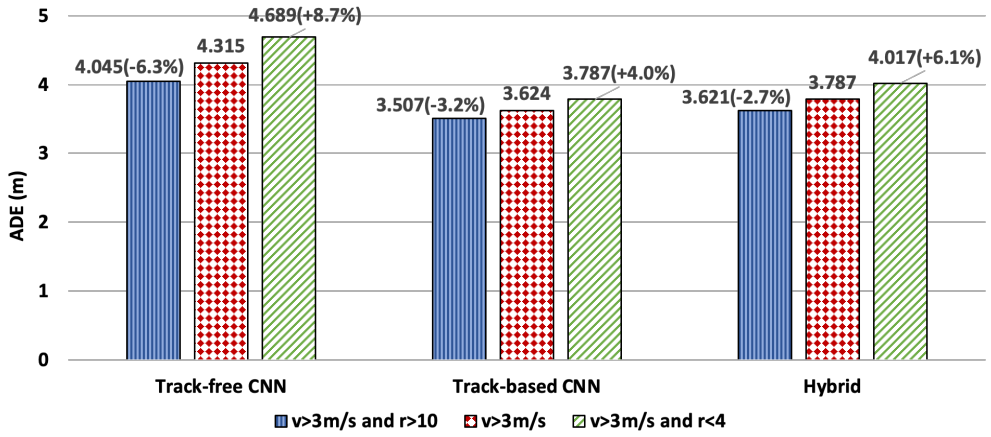

Figure 4.11: Performance evaluation of the described motion prediction methods on the nuScenes dataset [2] using ground-truth tracking information on agents moving with a velocity larger than $3m/s$. The scene density factor was also considered to further categorize the scenarios. The radius r between a given agent and their closest neighbor was calculated and those with $r < 4m$ in one experiment (dense scenarios) and $r > 10m$ in a second experiment (non dense scenarios) were selected. The performance of the three methods degrade in dense scenarios. The decrease is most pronounced with Track-free model which consolidates the importance of tracking information under these conditions.

the selected agents are categorized based on the density of their surrounding environment. The density is measured by calculating the radius between the agent and their nearest neighbor. A scenario is considered "dense" if the agent's radius is less than 4 meters ($r < 4$) and a scenario "non dense" if the radius is larger than 10 meters ($r > 10$).

The results are summarized in Figure 4.11. The figure represents a bar plot with the models' performances on each scenes category in ADE metric. For dense and non-dense scenarios, the performance change (in percentages) is specified for all agents with $v > 3m/s$ (red bars). First, focusing on the set of moving agents with a velocity larger than 3m/s represented in red bars, the performance decreased in all three models since the selected scenarios are relatively challenging due to the high inter-frame motion and density of the scenes. The decrease in performance when considering the moving agents only as opposed to the overall performance is more pronounced in the nuScenes dataset (with a decrease of 0.517 m in ADE) than in the Lyft dataset (with a decrease in ADE of 0.446 m). This drop can be explained by the relatively low frame rate of the nuScenes

dataset that, overlapped with the high inter-frame motion, has led to an input with temporally more distant states to represent the past motion of the agents compared to the Lyft dataset.

Second, the track-free model significantly decreased in performance compared to the tracking-based models. Compared to all moving agents ($v > 3m/s$, red bar), the performance of the track-free CNN model has decreased by 8.7% on dense scenarios ($r < 4$, green bar) as opposed to a more attenuated decrease of 4% and 6.1% for track-based CNN and hybrid models, respectively. On non-dense scenarios ($r > 10$, blue bar), the three models perform better than the average performance of moving agents (red bars) with a more attenuated improvement for the track-free CNN model (4.045 m compared to 3.507 m and 3.621 m for the track-based models). These findings rightly demonstrate that the tracking information does improve the motion prediction model's performance in challenging scenarios, such as a very crowded scene where the input 2D BEV representation of the detected agents can become less effective. However, for other scenarios, such as non-dense scenarios, the 2D representation is solely a powerful input representation for effective motion prediction models.

In the non-dense experiments (blue bars), the track-free model performs worse than both the tracking-based models. This finding is different in the case of the Lyft Dataset, where all three models performed comparably. We conclude that the non-dense scenarios with moving agents at $v > 3m/s$ in the nuScenes dataset are more challenging than those in the Lyft dataset and that the importance of the tracking information is more highlighted in the case of the nuScenes dataset.

We note that the described experiments were carried out with the ground-truth tracking information provided in the dataset, which we assume is perfect and noise-free. In the following experiments, the performance of the three models will be re-evaluated with noisy trackers.

Performance Evaluation with Noisy Tracker

These experiments aim to study the effect of synthetic tracking noise on the performance of motion prediction models on the nuScenes dataset. First, as mentioned in the Lyft dataset experiments, the track-free CNN model is not affected by the noise applied to the tracking information. It treats each past frame as independent detections and assumes no direct correlation between

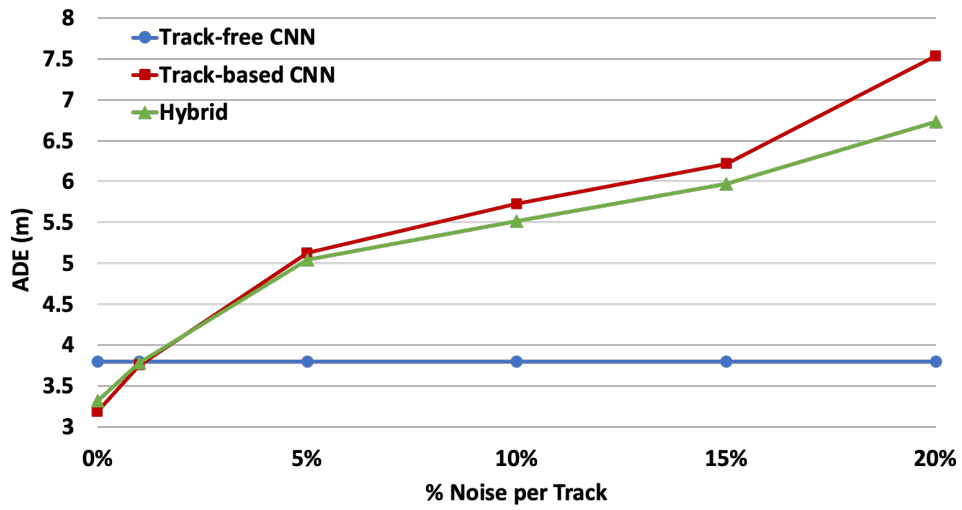

Figure 4.12: Performance Evaluation of the described methods the nuScenes dataset with synthetic noise applied to the tracking information. Results reported in ADE (m). An increasing chance of 1 identity switch per track was experimented with. The performance of the tracking based methods (track-based CNN and Hybrid) decreases with increasing tracking noise. The track-free model is not affected by tracking noise.

detections across frames. Thus, the performance of this model remains constant across these experiments. Second, the effect of synthetic noise is evaluated on the two tracking-based models (track-based CNN and Hybrid) by conducting two sets of experiments (described in § 4.6.3). The first set of experiments varies the chances of random identity switches per track from 0% to 20%. Results of these experiments are presented in Figure 4.12. A similar trend is observed to that on the Lyft dataset (Figure 4.6). The performance of both the track-based CNN and hybrid model is highly affected by the tracking noise. Around an identity switch chance of 1%, the performances of the three models become comparable. Then the tracking-based models decrease in performance with higher tracking noise chances, proving the track-based models' high sensitivity to the tracking noise. Furthermore, we compare the performance of the tracking-based models. Though the track-based CNN model performs better than the hybrid model when no noise is applied to the tracking information, its performance quickly falls behind the Hybrid model when we apply tracking noise. Thus we conclude that the two tracking-based models are both sensitive to the tracking noise, with the track-based CNN having a higher sensitivity to the tracking noise with chances higher than 2%. Though the Hybrid model is highly dependent on the LSTM input enclosing the tracking

information, it also relies on the input image independent of the tracking module, which explains its relative robustness to noise compared to the track-based CNN model.

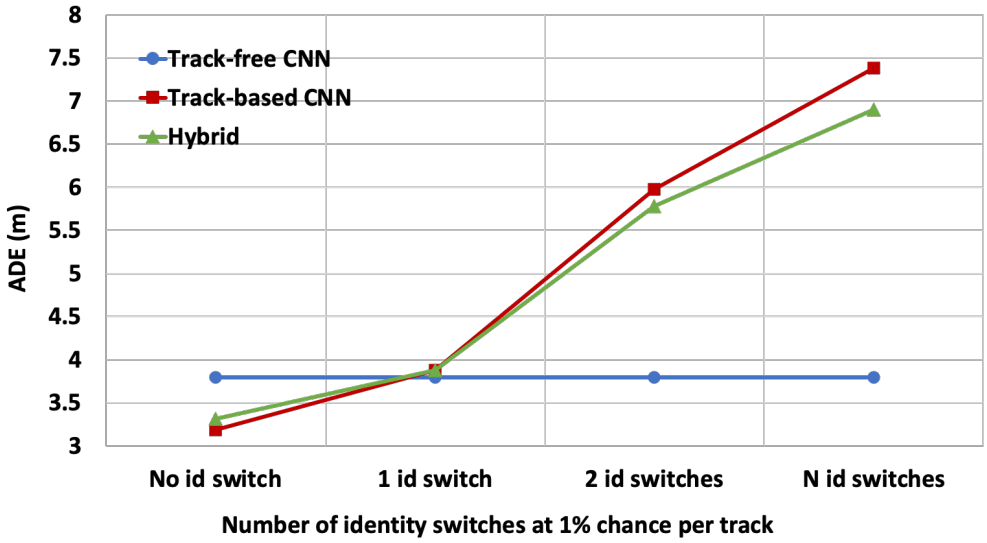

Figure 4.13: Performance Evaluation of the described methods on the nuScenes dataset with synthetic noise applied to the tracking information. Results reported in ADE (m). Synthetic tracking noise of 1% chance and an increasing number of identity switches per track were experimented with. 1 id switch represent identity switch at only 1 timestamp. 2 id switches represent an identity switch for 2 consecutive timestamps and N id switch represent 1 identity switch that started at a random timestamp and continued until the end of the scene. The performance of the tracking based methods (track-based CNN and Hybrid) decreases with increasing tracking noise. The track-free model is not affected by tracking noise.

The second set of experiments reported in Figure 4.13 varies the number of identity switches per track at a constant chance of 1%. Further details of these experiments are given on the Lyft dataset tracking noise experiments in § 4.6.3. A similar trend to the first set of experiments is seen. Both tracking-based models decrease with an increasing number of identity switches translating the increasing level of noise in the tracking information. Similarly, the hybrid model indicates slightly higher robustness to tracking noise, as highlighted in the first set of experiments. The track-based CNN model falls behind the hybrid model with tracking noise higher or equal to 1 identity switch per track. In the scenario of one identity switch continuing until the end of the scene (N id switch), the hybrid model's performance outperforms the track-based CNN model by 0.5 meters in ADE.

Table 4.4: Overall comparison of the described methods using four metrics (m) using StanfordIPRL-TRI tracker [1] on the nuScenes dataset [2]. Real-world trackers introduce noise to the tracking information which affects the performance of track-based methods (Track-based CNN and Hybrid methods).

Method	AT	CT	ADE	FDE
Track-free CNN	**2.879**	**1.684**	**3.798**	**6.377**
Track-based CNN	3.343	1.927	4.292	6.834
Hybrid	3.047	1.858	3.997	6.739

In contrast, the track-free model maintains its higher performance of 3.6 meters gap in ADE with the track-based CNN model.

Peformance Evaluation with Real-world Tracker:

In this section, we reproduce the experiments described in § 4.6.3 on the nuScenes prediction dataset. Similarly, the ground-truth tracking information is replaced with the output of the real-world tracker StanfordIPRL-TRI introduced in [1] and fed to the motion prediction tracking based models. The StanfordIPRL-TRI tracker was used to run the experiments based on their publicly available code, and parameters suggested in their work. We recall that The StanfordIPRL-TRI tracker won the nuScenes challenge competition [2] by achieving state-of-the-art results in the task of tracking on the nuscenes dataset.

The results of the Track-based CNN and Hybrid models using the StanfordIPRL-TRI tracker on the nuScenes dataset are summarized in Table 4.4. Again, the track-free CNN model does not depend on the tracking module, so its performance remains constant and equal to the results presented in Table 4.3. Results in Table 4.4 reveal a decrease in performance of the tracking-based models using the StanfordIPRL-TRI tracker. Tracking-based models fell behind the track-free model by a drop of 0.168m and 0.464m in AT for the hybrid model and the track-based CNN model, respectively. These findings further consolidate the findings of the experiments on the Lyft dataset. Even real-world trackers can introduce noise that affects the performance of motion prediction models. Thus, a preliminary study on the tracking noise effect is recommended. Furthermore, these results suggest that, in the case of tracking noise, tracking-free motion prediction

models are preferred. Alternatively, the motion prediction model should be trained to ignore the noise coming from upstream processes and thus be robust to noise propagation.

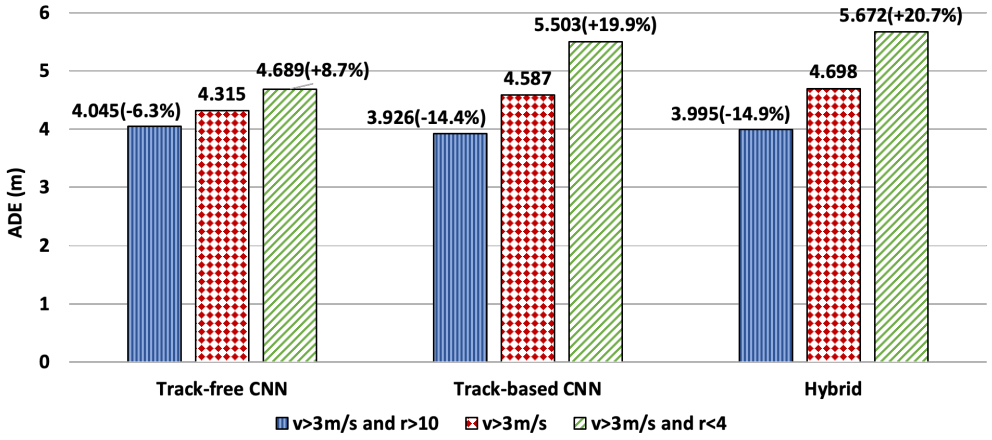

Figure 4.14: Performance evaluation of the described methods in ADE (m) on agents moving with a velocity larger than $3m/s$. The scene density factor was further considered. The radius r between a given agent and their closest neighbor was calculated and those with $r < 4m$ in one experiment (dense scenarios) and $r > 10m$ in a second experiment (non dense scenarios) were selected. The performance decrease of the track-based methods is more pronounced using the real-world tracker due to the noise introduced to the tracking information.

The experiments described in § 4.6.3 are reproduced with a real-world tracker on the nuScenes dataset to analyze further the performance of the described models under different conditions and scenarios. The different scenes are grouped by the density factor (dense with the closest distance between any two actors is less than 4m vs. non-dense with the closest distance is larger than 10m) and actors speed (considering only actors with speed higher than 3m/s). Results are summarized in Figure 4.14.

Results demonstrate that, when considering all moving agents (red bars in Figure 4.14), the tracking-based models have decreased in performance when using real-world tracker with an ADE drop of 0.963 m and 0.911 m for the track-based CNN and hybrid models, respectively. The decrease in performance has led to the tracking-based models falling behind the track-free model with an improvement of 0.272m in ADE compared to the track-based CNN model on all moving agents. For the dense scenarios (green bars in Figure 4.14) where the tracker is more prone to mistakes,

the decrease in performance compared to all moving agents was noticed on all models with a more pronounced gap for the tracking-based models (19.9% and 20.7% for track-based CNN and hybrid models, respectively). For the non-dense scenarios (blue bars in Figure 4.14), all models perform comparably with a slight improvement for the track-based CNN model. In conclusion, these results suggest that the tracking noise has more effect on the models' performances in challenging scenarios where tracking is more important.

4.7 Conclusion

This chapter shed light on two major perception components of the autonomous navigation systems known as tracking and motion prediction. The two components are typically cascaded and are used to establish safe and reliable motion planning. However, such cascaded approaches are typically highly sensitive to error propagation from one component to another. The study described in this chapter aimed to analyze and highlight the effect of noise propagation in motion prediction which can lead to catastrophic failures if not studied and addressed thoroughly.

To this end, this chapter described a comprehensive evaluation of three motion prediction models. These models were used as testbeds to evaluate the effect of tracking noise on the motion prediction performance and to compare the tracking-based to the tracking-free alternatives. The first, the Track-free CNN model, operated on a BEV input created based on a high definition map and agent detections, with no tracking information included. The second, the Hybrid model, extended the first model by integrating the tracking information in LSTM embeddings of the agent's past states. The third, the Track-based CNN model, integrated the tracking information in the BEV input using spatio-temporal displacement fields. The three models were experimentally compared across different conditions: no-noise using the ground-truth tracking information provided by the datasets, synthetic-noise introduced by applying random identity switches to the ground-truth tracking information, and real-noise conditions using a real-world challenge winning tracker. These experiments were conducted on the two largest motion prediction datasets: The Lyft prediction dataset [51] and the nuScenes prediction dataset [2].

Results indicated that, while the tracking-based models performed better than the track-free model in the noise-free condition, their performance rapidly degraded when the tracking system produced noise — resulting in them falling behind the tracking-free system performance. The takeaway from this study is that practitioners should be aware of the effect of tracking noise on the motion prediction performance and should consider it when selecting their approach. Furthermore, the tracking-free models can be, in the case of an inevitable tracking noise, a potential option that is more robust when creating real-world applications. As a matter of fact, preliminary experiments could be conducted before deciding on the final approach. For instance, a comparative study on the motion prediction models with and without tracking step can be done. Alternatively, an end-to-end approach could be adopted with a thorough performance analysis to make sure that the motion prediction model learns to ignore noisy inputs from the tracker and robustly recovers accurate predictions under various conditions.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

As the robots industry continues to grow considerably, so does the number of challenges the companies face as clients have higher expectations for their products' performance. One of these challenges is the robot's adaptability to the dynamic, continuously changing environment they are operating in and their ability to handle the incredible amounts of noise and uncertainty. Thus, one key component, towards which companies are putting more effort, is the perception module which is responsible for translating the world's percepts into useful cues understandable by the robot. The work described in this dissertation shed light on this essential module and treated a subset of its components.

As a first step, this work studied the robot's ability to assimilate its surroundings by analyzing and identifying its different components. In short, the first goal was to allow the robot to detect objects, identify them and recover their 6D poses. For this purpose, a novel end-to-end method for RGB-only 6D pose estimation was introduced. Specifically, the proposed approach was composed of two modules: First, PPN, a fully-CNN-based architecture that produced initial pose estimates. Second, MARN, a pose refinement network that combined visual and flow features to estimate accurate transformations between the predicted and actual object pose. MARN utilized a spatial multi-attentional block to emphasize important feature parts, making the method more robust. The full end-to-end model achieved state-of-the-art results on three popular benchmarks. It is worth noting that the reported results of our approach were based on a 3-fold cross-validation procedure which can be different than the procedure adopted by the other existing methods.

This work also included a practical contribution, which integrated the proposed object detection and pose estimation model into the multi-modal interactive agent Diana. The agent conducted interactive conversations with users, made human-like reasoning and intelligently reacted with

them. With the new feature, Diana not only received commands from the user, whether through gestures or audio, but she could also "see" the real world, recognized the natural objects present in the table in front of the user and estimated their poses. The model was trained on a synthetic dataset generated by randomly rendering 3D models of objects reconstructed with adequate 3D modeling tools. Results confirmed the effectiveness of the proposed model on accurately recovering the detection and pose estimation of objects in the real world in front of Diana.

The second topic described in this dissertation revolved around the navigation capability of robots. Specifically, it tackles two essential components known as tracking and motion prediction. The study aimed to analyze the performance of navigation systems under various conditions and highlight the effect of tracking noise propagation on motion prediction, which can lead to catastrophic failures if not studied and addressed thoroughly. For this purpose, the study covered a comprehensive evaluation of three motion prediction models. These models were used as test-beds to evaluate the effect of tracking noise on the motion prediction performance and to compare the tracking-based to the tracking-free alternatives. The first, the Track-free CNN model, operated on a BEV input created based on a high definition map and agent detections, with no tracking information included. The second, the Hybrid model, extended the first model by integrating the tracking information in LSTM embeddings of the agent's past states. The third, the Track-based CNN model, integrated the tracking information in the BEV input using spatio-temporal displacement fields. The three models were experimentally compared across different noise levels: no-noise using ground-truth tracking information, synthetic noise by applying random identity switches to the ground-truth tracking information, and real-noise condition using a real-world tracker. These experiments were conducted on the two largest motion prediction datasets to date: The Lyft prediction dataset [51] and the nuScenes prediction dataset [2].

The obtained results showed that tracking-based models performed better in our experiments than the track-free model in noise-free conditions with higher performance gap in the more challenging conditions, proving that accurate tracking is an important factor for improving motion prediction performance. With the introduction of tracking noise, the performance of the three models

became comparable around an identity switch chance of 1%, then the performance of tracking-based models degraded and fell behind that of the tracking-free model. The tracking-based models' decrease in performance with higher tracking noise chances, proved the high sensitivity of the track-based models to the tracking noise. Further results of the tracking-based models using the StanfordIPRL-TRI tracker, the nuScenes challenge winner, revealed that even state-of-the-art trackers could introduce tracking noise which affected the performance of motion prediction models.

Considering the tracking-based methods (Track-based CNN and Hybrid), the results indicated that though LSTMs efficiently capture the long-term temporal dependencies of the agent's motion, the model integrating displacement fields would perform better in more challenging scenarios. This conclusion quickly became invalid with the introduction of noise, where hybrid models relying on both 2d representations inputs and LSTMs revealed higher robustness to tracking noise compared to those integrating displacement fields. These results indicate that though the Hybrid model is highly dependent on the LSTM input enclosing the tracking information, it also relies on the input image, independent of the tracking module, which explains its relative robustness to noise compared to the track-based CNN model.

In conclusion, the takeaway from this study is that effective and robust trackers introducing little to no noise are essential for effective motion prediction models that use the tracking information. Furthermore, practitioners should be aware of the effect of tracking noise on the motion prediction performance and should consider that when selecting their approach. Finally, the tracking-free models can be a potential alternative, in the case of an inevitable tracking noise, which can be a more robust option when creating real-world applications. Potentially, it is recommended to conduct preliminary experiments before deciding on the final system. For instance, a thorough analysis on the noise level of the tracker in challenging conditions is encouraged. Furthermore, a comparative study on the motion prediction models with and without tracking step can be done. Alternatively, an end-to-end approach could be adopted with a thorough performance analysis to

make sure that the motion prediction model learns to ignore noisy inputs from the tracker and robustly recovers accurate predictions under various conditions.

5.2 Future Work

This dissertation aimed to tackle crucial aspects of the perception capability for intelligent mobile robots. The first contribution consisted of an effective object detector and pose estimator that achieved state-of-the-art results on three public benchmarks. The proposed approach had two main components: PPN, a fast object detector and pose estimator that takes only a single RGB as input and does not require any additional information such as the 3D models of objects, and MARN, a pose refiner that improves the initial pose estimates by reducing the residual poses. In contrast to PPN, MARN requires the knowledge of the 3D models of objects of interest to operate. This constraint can affect the adaptability of MARN to different pose estimation tasks. For instance, if the target objects had no 3D models due to the unavailability of the real object. Similarly, when the goal is to estimate the pose of similar-looking objects that do not necessarily have the same 3D models, such as in the context of self-driving cars (including cars, buses and motorcycles). Thus, future work can include ways to improve the refinement process by removing the need for prior knowledge of 3D models while maintaining similar or better performance. Furthermore, the iterative nature of the refiner can considerably increase time consumption if multiple iterations are needed to reach the desired performance. Thus, time optimization is an important topic that can be an extension to this work. Besides, the reported results of our approach were based on a 3-fold cross-validation procedure which can be different than the procedure adopted by the other existing methods. Thus, future work can reproduce the same experiments for the existing methods and perform an exhaustive comparison of the different performances. Finally, future work can also include the experimentation with more variations in architecture and training parameters.

The second contribution of this work consisted in integrating the proposed detector and pose estimator into the CwC interactive embodied agent Diana [7]. The new capability enhanced the interactive power of Diana with the surrounding real world by allowing her to "see" the real world,

recognize the natural objects present in the table in front of the user and estimate their poses. Such capability can open up the door to many improvements in Diana's system. One potential extension to this work can include Diana interactively playing games with the user telling them if one object is missing from the table and identifying it. Another potential improvement could be communication through imitation with the user. Diana can imitate the user's layout of real objects in the table in front of them using her virtual objects without addressing verbal or gesture commands.

This dissertation's third and final contribution consisted of a comprehensive analysis of noise propagation in the context of motion prediction, where the effect of tracking noise on the performance is thoroughly studied and evaluated. The experiments were conducted on three motion prediction models where one model was used as a tracking-free baseline, and the two other models used the tracking information. The Track-based CNN is one of the models that used the tracking information encoded as displacement fields in the 2D input representation. Though displacement fields are known in other tasks, we believe their application to motion prediction is novel. Thus, future research can include ameliorating the performance of the Track-based CNN as a novel approach for motion prediction.

Finally, this dissertation focused on the study of subsets of an efficient and effective perception module. For this purpose, this work included: An object recognition and pose estimation model which allowed to detect objects of interest in the input frame and recover their poses. Second, three motion prediction models which were used to conduct an analysis on the inter-dependency between the tracking and the motion prediction task leads to the fact that we can achieve great results on the motion prediction task directly from the detection and pose estimation results without the need for an additional, time-consuming tracking module. Recently, many researchers have been interested in Multi-task Learning (MTL) [106]. MTL has shown that learning multiple tasks jointly can lead to greater performance improvement than learning them individually since the other tasks can leverage the knowledge contained in one task. For example, leveraging motion prediction information can help reduce the false negatives of the detection task when dealing with occluded objects. Furthermore, MTL allows reducing system latency and memory by sharing computations

across all tasks. Thus, future work points towards gathering the components mentioned above into a single end-to-end detection and motion prediction model trained jointly and used in conjunction with the motion planning module for autonomous navigation systems. The goal is to achieve good performance compared to existing perception modules while efficiently managing time and resources.

Bibliography

[1] Hsu-kuang Chiu, Antonio Prioletti, Jie Li, and Jeannette Bohg. Probabilistic 3D Multi-Object Tracking for Autonomous Driving. *arXiv preprint arXiv:2001.05673*, 2020.

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. NuScenes: A Multimodal Dataset for Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11621–11631, 2020.

[3] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International journal of computer vision (IJCV)*, 88(2):303–338, 2010.

[4] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2002.

[5] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles. *IEEE Transactions on Intelligent Vehicles (T-IV)*, 1(1):33–55, 2016.

[6] Nikhil Krishnaswamy, Pradyumna Narayana, Isaac Wang, Kyeongmin Rim, Rahul Bangar, Dhruva Patil, Gururaj Mulay, Ross Beveridge, Jaime Ruiz, Bruce Draper, et al. Communicating and Acting: Understanding Gesture in Simulation Semantics. In *12th International Conference on Computational Semantics—Short papers (IWCS)*, 2017.

[7] Nikhil Krishnaswamy, Pradyumna Narayana, Rahul Bangar, Kyeongmin Rim, Dhruva Patil, David McNeely-White, Jaime Ruiz, Bruce Draper, Ross Beveridge, and James Pustejovsky. Diana's World: A Situated Multimodal Interactive Agent. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 13618–13619, 2020.

[8] Pradyumna Narayana, Nikhil Krishnaswamy, Isaac Wang, Rahul Bangar, Dhruva Patil, Gururaj Mulay, Kyeongmin Rim, Ross Beveridge, Jaime Ruiz, James Pustejovsky, et al. Cooperating with Avatars through Gesture, Language and Action. In *Proceedings of SAI Intelligent Systems Conference (IntelliSys)*, pages 272–293. Springer, 2018.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[10] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3D Bounding Box Estimation using Deep Learning and Geometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7074–7082, 2017.

[11] Dominic Zeng Wang and Ingmar Posner. Voting for Voting in Online Point Cloud Object Detection. In *Robotics: Science and Systems (RSS)*, volume 1, pages 10–15607, 2015.

[12] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks. In *International Conference on Robotics and Automation (ICRA)*, pages 1355–1361. IEEE, 2017.

[13] Aniruddha V Patil and Pankaj Rabha. A Survey on Joint Object Detection and Pose Estimation using Monocular Vision. In *MATEC Web of Conferences*, volume 277, page 02029. EDP Sciences, 2019.

[14] Caner Sahin, Guillermo Garcia-Hernando, Juil Sock, and Tae-Kyun Kim. A Review on Object Pose Recovery: from 3D Bounding Box Detectors to Full 6D Pose Estimators. *Image and Vision Computing (IVC)*, page 103898, 2020.

[15] Martin A Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM (CACM)*, 24(6):381–395, 1981.

[16] Paul J Besl and Neil D McKay. Method for Registration of 3-D Shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures (SPIE)*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[17] Szymon Rusinkiewicz and Marc Levoy. Efficient Variants of the ICP Algorithm. In *Proceedings of the IEEE International Conference on 3-D Digital Imaging and Modeling (3DIM)*, pages 145–152. IEEE, 2001.

[18] Jay M Wong, Vincent Kee, Tiffany Le, Syler Wagner, Gian-Luca Mariottini, Abraham Schneider, Lei Hamilton, Rahul Chipalkatty, Mitchell Hebert, David MS Johnson, et al. SegICP: Integrated Deep Semantic Segmentation and Pose Estimation. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5784–5789. IEEE, 2017.

[19] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient Response Maps for Real-time Detection of Texture-less Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(5):876–888, 2011.

[20] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model Based Training, Detection and Pose Estimation of Texture-less 3D Objects in Heavily Cluttered Scenes. In *Asian Conference on Computer Vision (ACCV)*, pages 548–562. Springer, 2012.

[21] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. Comparing Images using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 15(9):850–863, 1993.

[22] David G Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[23] Fred Rothganger, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 3D Object Modeling and Recognition using Local Affine-Invariant Image Descriptors and Multi-View Spatial Constraints. *International Journal of Computer Vision (IJCV)*, 66(3):231–259, 2006.

[24] Wadim Kehl, Fausto Milletari, Federico Tombari, Slobodan Ilic, and Nassir Navab. Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 205–220, 2016.

[25] Marius Muja. *Scalable Nearest Neighbour Methods for High Dimensional Data*. PhD thesis, University of British Columbia, 2013.

[26] Shubham Tulsiani and Jitendra Malik. Viewpoints and Keypoints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1510–1519, 2015.

[27] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-based 3D Detection and 6D Pose Estimation Great Again. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1521–1529, 2017.

[28] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection And Semantic Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot Multibox Detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-time Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[31] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time Seamless Single Shot 6D Object Pose Prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 292–301, 2018.

[32] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4561–4570, 2019.

[33] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal of Computer Vision (IJCV)*, 81(2):155, 2009.

[34] Mahdi Rad and Vincent Lepetit. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3828–3836, 2017.

[35] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.

[36] Markus Oberweger, Mahdi Rad, and Vincent Lepetit. Making Deep Heatmaps Robust to Partial Occlusions for 3D Object Pose Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–134, 2018.

[37] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[38] Min Sun, Gary Bradski, Bing-Xin Xu, and Silvio Savarese. Depth-Encoded Hough Voting for Joint Object Detection and Shape Recovery. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 658–671, 2010.

[39] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *Robotics: Science and Systems (RSS)*, 2018.

[40] Yinlin Hu, Pascal Fua, Wei Wang, and Mathieu Salzmann. Single-Stage 6D Object Pose Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2930–2939, 2020.

[41] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A Convolutional Network for Real-time 6-DoF Camera Relocalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2938–2946, 2015.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[43] Tingbo Hou, Adel Ahmadyan, Liangkai Zhang, Jianing Wei, and Matthias Grundmann. MobilePose: Real-Time Pose Estimation for Unseen Objects with Weak Shape Supervision. *arXiv preprint arXiv:2003.03522*, 2020.

[44] Fabian Manhardt, Wadim Kehl, Nassir Navab, and Federico Tombari. Deep Model-Based 6D Pose Refinement in RGB. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 800–815, 2018.

[45] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep Iterative Matching for 6D Pose Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698, 2018.

[46] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, Inception-Resnet and the Impact of Residual Connections on Learning. *arXiv preprint arXiv:1602.07261*, 2016.

[47] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning Optical Flow with Convolutional Networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[49] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D object pose estimation using 3D object coordinates. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 536–551, 2014.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems (NIPS)*, 30:5998–6008, 2017.

[51] John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One Thousand and One Hours: Self-driving Motion Prediction Dataset. *arXiv preprint arXiv:2006.14480*, 2020.

[52] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. VectorNet: Encoding HD Maps and Agent Dynamics from Vectorized Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11525–11533, 2020.

[53] Ashraf Elnagar. Prediction of Moving Objects in Dynamic Environments using Kalman Filters. In *Proceedings 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (Cat. No. 01EX515)*, pages 414–419. IEEE, 2001.

[54] Andrew H Jazwinski. *Stochastic Processes and Filtering Theory*. Courier Corporation, 2007.

[55] Simon J Julier and Jeffrey K Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[56] Thomas Streubel and Karl Heinz Hoffmann. Prediction of Driver Intended Path at Intersections. In *IEEE Intelligent Vehicles Symposium Proceedings (IV)*, pages 134–139. IEEE, 2014.

[57] YoungJoon Yoo, Kimin Yun, Sangdoo Yun, JongHee Hong, Hawook Jeong, and Jin Young Choi. Visual Path Prediction in Complex Scenes with Crowded Moving Objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2668–2677, 2016.

[58] Andreas Møgelmose, Mohan M Trivedi, and Thomas B Moeslund. Trajectory Analysis and Prediction for Improved Pedestrian Safety: Integrated Framework and Evaluations. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 330–335. IEEE, 2015.

[59] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, 2016.

[60] Federico Bartoli, Giuseppe Lisanti, Lamberto Ballan, and Alberto Del Bimbo. Context-Aware Trajectory Prediction. In *24th International Conference on Pattern Recognition (ICPR)*, pages 1941–1946. IEEE, 2018.

[61] Seong Hyeon Park, ByeongDo Kim, Chang Mook Kang, Chung Choo Chung, and Jun Won Choi. Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1672–1678. IEEE, 2018.

[62] Ewoud AI Pool, Julian FP Kooij, and Dariu M Gavrila. Context-Based Cyclist Path Prediction using Recurrent Neural Networks. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 824–830. IEEE, 2019.

[63] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Transformer Networks for Trajectory Forecasting. *arXiv preprint arXiv:2003.08111*, 2020.

[64] Khaled Saleh. Pedestrian Trajectory Prediction using Context-Augmented Transformer Networks. *arXiv preprint arXiv:2012.01757*, 2020.

[65] Hao Cheng, Wentong Liao, Michael Ying Yang, Bodo Rosenhahn, and Monika Sester. AMENet: Attentive Maps Encoder Network for Trajectory Prediction. *arXiv preprint arXiv:2006.08264*, 2020.

[66] Hao Cheng, Wentong Liao, Xuejiao Tang, Michael Ying Yang, Monika Sester, and Bodo Rosenhahn. Exploring Dynamic Context for Multi-path Trajectory Prediction. *arXiv preprint arXiv:2010.16267*, 2020.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[68] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Nitin Singh, and Jeff Schneider. Uncertainty-Aware Short-Term Motion Prediction of Traffic Actors for Autonomous Driving. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2095–2104, 2020.

[69] Tung Phan-Minh, Elena Corina Grigore, Freddy A Boulton, Oscar Beijbom, and Eric M Wolff. CoverNet: Multimodal Behavior Prediction using Trajectory Sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14074–14083, 2020.

[70] Kaouther Messaoud, Nachiket Deo, Mohan M Trivedi, and Fawzi Nashashibi. Trajectory Prediction for Autonomous Driving based on Multi-Head Attention with Joint Agent-Map Representation. *arXiv preprint arXiv:2005.02545*, 2020.

[71] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. MultiPath: Multiple Probabilistic Anchor Trajectory Hypotheses for Behavior Prediction. In *Conference on Robot Learning (CoRL)*, pages 86–99, 2020.

[72] Liangji Fang, Qinhong Jiang, Jianping Shi, and Bolei Zhou. TPNet: Trajectory Proposal Network for Motion Prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6797–6806, 2020.

[73] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. Social-STGCNN: A Social Spatio-Temporal Graph Convolutional Neural Network for Human Trajectory Prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14424–14432, 2020.

[74] Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, Hamid Rezatofighi, and Silvio Savarese. Social-BiGAT: Multimodal Trajectory Forecasting using Bicycle-GAN and Graph Attention Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 137–146, 2019.

[75] Boris Ivanovic and Marco Pavone. The Trajectron: Probabilistic Multi-Agent Trajectory Modeling with Dynamic Spatiotemporal Graphs. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2375–2384, 2019.

[76] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Multi-Agent Generative Trajectory Forecasting with Heterogeneous Data for Control. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[77] Stefano Pellegrini, Andreas Ess, and Luc Van Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 452–465, 2010.

[78] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal Trajectory Predictions for

Autonomous Driving using Deep Convolutional Networks. In *International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019.

[79] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3569–3577, 2018.

[80] Ming Liang, Bin Yang, Wenyuan Zeng, Yun Chen, Rui Hu, Sergio Casas, and Raquel Urtasun. PnPNet: End-to-End Perception and Prediction with Tracking in the Loop. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11553–11562, 2020.

[81] Pengxiang Wu, Siheng Chen, and Dimitris N Metaxas. MotionNet: Joint Perception and Motion Prediction for Autonomous Driving Based on Bird's Eye View Maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11385–11395, 2020.

[82] Nemanja Djuric, Henggang Cui, Zhaoen Su, Shangxuan Wu, Huahua Wang, Fang-Chieh Chou, Luisa San Martin, Song Feng, Rui Hu, Yang Xu, et al. MultiNet: Multiclass Multistage Multimodal Motion Prediction. *arXiv preprint arXiv:2006.02000*, 2020.

[83] Sergio Casas, Wenjie Luo, and Raquel Urtasun. IntentNet: Learning to Predict Intention from Raw Sensor Data. In *Conference on Robot Learning (CoRL)*, pages 947–956, 2018.

[84] Lingyun Luke Li, Bin Yang, Ming Liang, Wenyuan Zeng, Mengye Ren, Sean Segal, and Raquel Urtasun. End-to-End Contextual Perception and Prediction with Interaction Transformer. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

[85] Ameni Trabelsi, Mohamed Chaabane, Nathaniel Blanchard, and Ross Beveridge. A Pose Proposal and Refinement Network for Better 6D Object Pose Estimation. In *Proceedings*

of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pages 2382–2391, 2021.

[86] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv preprint arXiv:1706.05098*, 2017.

[87] Mohamed Chaabane, Ameni Trabelsi, Nathaniel Blanchard, and Ross Beveridge. Looking ahead: Anticipating Pedestrians Crossing with Future Frames Prediction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[88] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2462–2470, 2017.

[89] Aaditya Prakash, James Storer, Dinei Florencio, and Cha Zhang. Repr: Improved Training of Convolutional Filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10666–10675, 2019.

[90] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The YCB Object and Model Set: Towards Common Benchmarks for Manipulation Research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.

[91] Eric Brachmann, Frank Michel, Alexander Krull, Michael Ying Yang, Stefan Gumhold, et al. Uncertainty-driven 6D pose estimation of objects and scenes from a single rgb image. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 3364–3372, 2016.

[92] James Pustejovsky and Nikhil Krishnaswamy. VoxML: A visualization modeling language. *arXiv preprint arXiv:1610.01508*, 2016.

[93] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. Ieee, 2009.

[94] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[95] Geert Litjens, Peter Bandi, Babak Ehteshami Bejnordi, Oscar Geessink, Maschenka Balkenhol, Peter Bult, Altuna Halilovic, Meyke Hermsen, Rob van de Loo, Rob Vogels, et al. 1399 H&E-Stained Sentinel Lymph Node Sections of Breast Cancer Patients: the CAMELYON Dataset. *GigaScience*, 7(6):giy065, 2018.

[96] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D. *arXiv:2007.08501*, 2020.

[97] Tony Mullen. *Mastering Blender*. John Wiley & Sons, 2011.

[98] M Lewis and C Oswald. Can an Inexpensive Phone App Compare to Other Methods When It Comes to 3D Digitization of Ship Models. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences (ISPRS)*, 2019.

[99] Mohamed Chaabane, Ameni Trabelsi, Nathaniel Blanchard, and Ross Beveridge. Looking ahead: Anticipating pedestrians crossing with future frames prediction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2297–2306, 2020.

[100] Chester Gong and Dave McNally. A Methodology for Automated Trajectory Prediction Analysis. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 4788, 2004.

[101] Jiwoon Ahn, Sunghyun Cho, and Suha Kwak. Weakly Supervised Learning of Instance Segmentation with Inter-Pixel Relations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2209–2218, 2019.

[102] Davy Neven, Bert De Brabandere, Marc Proesmans, and Luc Van Gool. Instance Segmentation by Jointly Optimizing Spatial Embeddings and Clustering Bandwidth. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8837–8845, 2019.

[103] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[104] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8026–8037, 2019.

[105] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[106] Yu Zhang and Qiang Yang. A Survey on Multi-Task Learning. *arXiv preprint arXiv:1707.08114*, 2017.