

# Applications of Computer Vision and Machine Learning to Three Engineering Problems

by

Bowen Xie

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

University of Alberta

© Bowen Xie, 2021

# Abstract

This thesis applies computer vision and machine learning techniques to three engineering projects: a self-driving vehicle, a predictive display system, and a vision-based robot manipulator joint detector. In the first project, we build a remote-controlled car and implement three core self-driving features: lane-keeping control, traffic signs/signals detection and distance estimation, and obstacle avoidance. The first two features are benchmarked in a lab environment. We employ a novel end-to-end learning method which directly controls the vehicle based on the image perceived, instead of a traditional model-based control design. The YOLO object detector is used to identify different traffic signs and its bounding boxes are utilized to estimate their distance to the vehicle. The proposed system demonstrates satisfactory results in both qualitative and quantitative evaluations, and it outperforms human drivers in terms of control consistency and smoothness. For the predictive display project, we propose a new generative model-based predictive display for robotic teleoperation over high-latency communication links. Our method is capable of rendering photo-realistic images of the scene to the human operator in real-time from RGB-D images acquired by the remote robot. A preliminary exploration stage is used to build a coarse 3D map of the remote environment and to train a generative model, both of which are then used to generate photo-realistic images for the human operator based on the commanded pose of the robot. Data captured by the remote robot is used to dynamically update the 3D map, enabling teleoperation in the presence of new and relocated objects. Various

experiments validate our proposed method’s performance and benefits over alternative methods. The third project considers vision-based estimation of robot arm joint locations. Automatic robot arm manipulation is well developed for small robot arms with precise joint feedback, but still underdeveloped for inexpensive robots or human-operated equipment due to lacking precise joint feedback. Manual training data labelling for neural networks for robotic objects is not economic, so the simulator is now a popular tool to generate training data. The problem is the gap between real-world images and simulation images. Hence we propose a vision-based system with domain adaption for joint state estimation. The resulting system is implemented and benchmarked against a state-of-the-art approach with favourable results.

# Preface

Some of the research conducted for this thesis forms parts of three collaborative research projects. The software packages referred to in Chapter 3 were designed and programmed by myself, with the assistance of Mingjie Han and Linjian Xiang. The software pipeline, literature review, and data analysis in Chapter 4 are my original work. Additionally, part of the software modules, data collection and data analysis in Chapter 5 are collaborative work with Mingjie Han.

Chapter 4 of this thesis has been submitted to IEEE RA-L with ICRA 2021 option as “A Generative Model-Based Predictive Display for Robotic Teleoperation” by Bowen Xie, Mingjie Han, Jun Jin, Martin Barczyk and Martin Jagersänd. I was responsible for software design, data analysis and manuscript composition. Mingjie Han assisted with data collection and data analysis. Jun Jin provided high-level suggestions and contributed to manuscript edits. Martin Barczyk and Martin Jagersänd were the supervisors and were involved with concept formation and manuscript edits.

Chapter 5 of this thesis will be submitted to IEEE RA-L with IROS 2021 option as “Image-Based Joint State Estimation Pipeline for Low-cost or Feedback-less Robotic Manipulators” by Mingjie Han, Bowen Xie, Martin Barczyk and Alireza Bayat. I was responsible for training and testing the generative adversarial network (GAN), and composing the software pipeline. Mingjie Han was responsible for data collection and testing the performance of the full system, as well as composing the manuscript. Martin Barczyk and Alireza Bayat were the supervisors and were involved with concept formation and manuscript edits.



# Acknowledgements

I would like to thank Dr. Martin Barczyk and Dr. Martin Jagersänd for all the guidance, enlightenment and supervision through the course of my graduate study; Mingjie Han for the collaboration on the research projects; Jun Jin for inspiring me and giving me information about the predictive display. Finally, I would like to thank my parents for their support and love.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | The motivation of research . . . . .                          | 1         |
| 1.1.1    | Statement of contributions . . . . .                          | 3         |
| 1.2      | Literature review . . . . .                                   | 3         |
| 1.2.1    | Sub-domains of computer vision . . . . .                      | 3         |
| 1.2.2    | Applications of machine learning to engineering problems      | 4         |
| 1.3      | Outline of the thesis . . . . .                               | 5         |
| <b>2</b> | <b>Hardware Background</b>                                    | <b>7</b>  |
| 2.1      | Baxter robot . . . . .  | 7         |
| 2.2      | Computer vision preliminaries . . . . .                       | 9         |
| 2.2.1    | Monocular camera model . . . . .                              | 9         |
| 2.2.2    | Camera calibration . . . . .                                  | 13        |
| 2.3      | Imaging sensors . . . . .                                     | 15        |
| 2.3.1    | RGB camera . . . . .  | 15        |
| 2.3.2    | RGB-D camera . . . . .  | 16        |
| 2.3.3    | Vicon tracking system . . . . .                               | 16        |
| 2.4      | On-board processor . . . . .                                  | 17        |
| 2.5      | Off-board processing unit . . . . .                           | 18        |
| <b>3</b> | <b>Learned Self-Driving Vehicle</b>                           | <b>19</b> |
| 3.1      | Introduction . . . . .  | 19        |
| 3.2      | Related works . . . . .                                       | 20        |
| 3.2.1    | Vehicle automation level standard . . . . .                   | 20        |
| 3.2.2    | Image-based driving automation . . . . .                      | 20        |
| 3.2.3    | Traffic sign recognition . . . . .                            | 22        |
| 3.2.4    | Decision tree . . . . .                                       | 23        |
| 3.3      | DIY remote control (RC) car under lab settings . . . . .      | 24        |
| 3.3.1    | Dynamic model of the differentially-steered vehicle . . . . . | 24        |
| 3.3.2    | Hardware setup . . . . .                                      | 25        |
| 3.3.3    | Software setup . . . . .                                      | 27        |
| 3.4      | Experimental results . . . . .                                | 39        |
| 3.4.1    | Qualitative evaluation . . . . .                              | 41        |
| 3.4.2    | Quantitative evaluation . . . . .                             | 42        |
| 3.5      | Conclusion . . . . .  | 47        |
| <b>4</b> | <b>Predictive Display</b>                                     | <b>49</b> |
| 4.1      | Introduction . . . . .  | 49        |
| 4.2      | Related works . . . . .                                       | 51        |
| 4.2.1    | Predictive display . . . . .                                  | 51        |
| 4.2.2    | 3D reconstruction . . . . .                                   | 51        |
| 4.2.3    | Deep learning based images synthesis . . . . .                | 52        |
| 4.3      | Methodology . . . . .   | 53        |

|          |   |           |
|----------|---|-----------|
| 4.3.1    | Overview of the system . . . . .                      | 53        |
| 4.3.2    | Preprocessing . . . . .                               | 54        |
| 4.3.3    | Generative model learning . . . . .                   | 55        |
| 4.3.4    | Online model updating . . . . .                       | 56        |
| 4.3.5    | Robot pose correction . . . . .                       | 56        |
| 4.4      | Experimental results . . . . .                        | 57        |
| 4.4.1    | Experiment setup . . . . .                            | 57        |
| 4.4.2    | Qualitative evaluation . . . . .                      | 58        |
| 4.4.3    | Quantitative evaluation . . . . .                     | 61        |
| 4.5      | Conclusion . . . . .                                  | 63        |
| <b>5</b> | <b>External Vision-based Joint Detection System</b>   | <b>64</b> |
| 5.1      | Introduction . . . . .                                | 64        |
| 5.2      | Related works . . . . .                               | 66        |
| 5.2.1    | Robot joint angle measurements . . . . .              | 66        |
| 5.2.2    | Marker-based robot joint tracking . . . . .           | 67        |
| 5.2.3    | Articulated human body pose estimation . . . . .      | 67        |
| 5.3      | Methodology . . . . .                                 | 68        |
| 5.3.1    | System overview . . . . .                             | 68        |
| 5.3.2    | Instance segmentation . . . . .                       | 69        |
| 5.3.3    | Domain adaptation . . . . .                           | 70        |
| 5.3.4    | Joint detection . . . . .                             | 70        |
| 5.3.5    | Training dataset generation . . . . .                 | 71        |
| 5.4      | Experimental results . . . . .                        | 74        |
| 5.4.1    | Experimental datasets . . . . .                       | 74        |
| 5.4.2    | Groundtruth data generation . . . . .                 | 75        |
| 5.4.3    | Quantitative evaluation metrics . . . . .             | 76        |
| 5.4.4    | Baseline methods comparison . . . . .                 | 76        |
| 5.5      | Conclusion . . . . .                                  | 80        |
| <b>6</b> | <b>Conclusion and Future Work</b>                     | <b>82</b> |
| 6.1      | Conclusion . . . . .                                  | 82        |
| 6.2      | Limitations of work . . . . .                         | 83        |
| 6.3      | Future work . . . . .                                 | 84        |
|          | <b>References</b>                                     | <b>86</b> |
|          | <b>Appendix A Carla simulator setup</b>               | <b>96</b> |
| A.1      | Self-driving car simulator . . . . .                  | 96        |
| A.1.1    | An overview of Carla simulator . . . . .              | 96        |
| A.1.2    | Self-driving features to be tested in Carla . . . . . | 97        |
| A.1.3    | Simulated sensors setup . . . . .                     | 98        |

# List of Tables

|     |   |    |
|-----|---|----|
| 3.1 | End-to-end learning datasets . . . . .                            | 42 |
| 3.2 | Average errors and standard deviations in distance estimations    | 47 |
| 4.1 | PD datasets details . . . . .                                     | 57 |
| 4.2 | Quantitative evaluation of different 3D reconstruction methods    | 62 |
| 5.1 | Testing datasets for External vision-based joint detection system | 75 |
| 5.2 | PCK@0.2 scores for joint detection . . . . .                      | 76 |
| 5.3 | Confident detection rate for DREAM . . . . .                      | 79 |
| 5.4 | RMS scores for joint detection in two background settings . .     | 79 |

# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | Overview of Baxter robot . . . . .   | 8  |
| 2.2  | Switch for Baxter . . . . .  | 8  |
| 2.3  | Camera module on the cuff of the Baxter . . . . .  | 9  |
| 2.4  | Camera thin lens model . . . . .   | 10 |
| 2.5  | Pinhole camera model . . . . .   | 10 |
| 2.6  | Inverted Pinhole camera model . . . . .  | 11 |
| 2.7  | The chessboard for camera calibration . . . . .  | 14 |
| 2.8  | Corners extracted from the chessboard for camera calibration .                             | 14 |
| 2.9  | Logitech C920 webcam . . . . .   | 15 |
| 2.10 | Raspberry Pi Camera Module V2 . . . . .  | 16 |
| 2.11 | Vicon Vero capture camera . . . . .  | 17 |
| 2.12 | Vicon tracking markers . . . . .   | 17 |
| 3.1  | The schematic diagram of the kinematic model of the RC car .                               | 25 |
| 3.2  | System overview of the self-driving RC car . . . . .                                       | 27 |
| 3.3  | An overview of the system of the self-driving RC car . . . . .                             | 28 |
| 3.4  | Lane markings on the floor of the lab . . . . .  | 29 |
| 3.5  | Lane markings in HSV space . . . . .   | 30 |
| 3.6  | Binary image masking for lane markings . . . . .   | 30 |
| 3.7  | Output image of the lane markings . . . . .  | 31 |
| 3.8  | Sliding window search of the lane markings . . . . .                                       | 33 |
| 3.9  | The structure of the CNN used in self-driving RC project . . .                             | 34 |
| 3.10 | The stop sign detection test scene . . . . .   | 37 |
| 3.11 | The traffic light detector . . . . .   | 38 |
| 3.12 | The flowchart of the logical tree for the self-driving DIY RC car                          | 40 |
| 3.13 | A screenshot of the demonstration video for self-driving RC<br>car's performance . . . . . | 41 |
| 3.14 | An overview of the track used for testing lane-keep functionality                          | 42 |
| 3.15 | Subsampled trajectory of the car in different datasets . . . . .                           | 43 |
| 3.16 | Driving states along the trajectory of the car in different datasets                       | 44 |
| 3.17 | Inconsistent bounding box heights . . . . .  | 45 |
| 3.18 | The sensitivity of distance estimation to pixel errors . . . . .                           | 46 |
| 3.19 | The probability density function (PDF) of the errors in distance<br>estimations . . . . .  | 47 |
| 4.1  | An overview of the core blocks of our predictive display . . . . .                         | 50 |
| 4.2  | Flowchart of our proposed PD system . . . . .  | 53 |
| 4.3  | Experimental setup for our proposed PD system . . . . .                                    | 58 |
| 4.4  | Comparisons between other 3D reconstruction methods and our<br>PD system . . . . .         | 59 |
| 4.5  | Qualitative comparison between NeRF and our PD method . .                                  | 61 |
| 4.6  | LPIPS scores of our PD system on different datasets . . . . .                              | 63 |
| 5.1  | Workflow overview of the proposed system . . . . .   | 69 |

|     |   |     |
|-----|---|-----|
| 5.2 | Training flow for the external vision-based joint detection system                                  | 71  |
| 5.3 | PCK scores for robot joint detection over natural lab background                                    | 77  |
| 5.4 | PCK result of white background datasets . . . . .   | 78  |
| 5.5 | Examples of some common failure cases . . . . .   | 80  |
| A.1 | A bird's eye view of the Carla world . . . . .  | 97  |
| A.2 | An example of Carla client manual control interface . . . . .                                       | 98  |
| A.3 | An image captured by the front-facing camera mounted on the<br>vehicle in Carla simulator . . . . . | 99  |
| A.4 | Radar measured points drawn in front of the vehicle . . . . .                                       | 100 |

# Chapter 1

## Introduction

### 1.1 The motivation of research

Since the start of computer vision research in the 1960s, computer vision has been employed in many domains of engineering, with the goal of extracting information from images (or videos) [72]. Thanks to progress in machine learning (e.g. Convolutional Neural Networks [40], Generative Adversarial Network [24]), learning-based computer vision techniques have vastly extended the range of its applications. Machine learning and its integration with computer vision has solved a lot of engineering problems that were previously intractable. For example, [74] proposed “Stanley”, a self-driving vehicle that utilizes computer vision and machine learning techniques to drive in desert at high speed without any human interventions. Therefore, the objective of this thesis is to implement machine learning and computer vision techniques to three engineering problems which would be very difficult or impossible to solve by other approaches. In particular, the three projects are:

1. A Self-driving System for a DIY Raspberry Pi-controlled Car
2. A Generative Model-Based Predictive Display for Robotic Teleoperation
3. An Image-Based Joint State Estimation Pipeline for Low-cost or Feedback-less Robotic Manipulators

The first project, developing a self-driving vehicle, serves as a foundation for the next two, since it involves many techniques from classic (stand-alone)

computer vision, such as color thresholding and homography transformations. Due to time constraints, it was not practical to develop a complete self-driving system capable of performing in any real-world scenario. Instead, we focused on a few core features (i.e. lane-keep, traffic signs/signals detection) needed in self-driving applications, and tested them in a lab environment. Unlike conventional methodologies which rely on model-based nonlinear control [47], [81], we implemented a novel end-to-end learning-based approach to control the car and meet its goals.

The experience and skills in computer vision gained from the first project were carried over and led to the next projects described in this thesis. The second project was to develop a generative model-based predictive display for robotic teleoperation. Teleoperation tasks are often faced with high latency and low bandwidth issues, and much research has been conducted to solve these issues by optimizing the communication network [1] or using more advanced data communication technologies (e.g. 5G [66]). On the other hand, the Predictive Display approach uses image priors to construct scene models (e.g. [60], [85]), and provide instant visual feedback to the operator. However, it also faces challenges with updating the models on-line to reflect changes in the remote environment, and providing photo-realistic images to the human operator. To address these issues, we implemented a generative model-based predictive display system which outputs photo-realistic images while simultaneously being able to update the remote scene model.

The third project reported in this thesis was to design an image-based joint detector, aimed at low-cost or feedback-less robotic manipulators. High-end robotic manipulators (e.g. Franka Emika Panda, Barrett WAM Arm) are equipped with highly precise sensors which can accurately measure joint states. However, low-cost or feedback-less manipulators (e.g. loading crane) are not equipped with such feedback sensors. Hence we developed a joint state estimator which uses external cameras to capture images of the manipulator and uses learning-based masking (Mask-RCNN), generative modeling, and object detection (ResNet) to estimate joint locations. This method enables low-cost and feedback-less manipulators the capability to achieve autonomous opera-



tion.

### **1.1.1 Statement of contributions**

This thesis claims the following research contributions:

- An end-to-end learning lane-keeping driving control algorithm, which outperforms human drivers in terms of consistency and smoothness, as shown by extensive lab experiments.
- A novel approach to predictive displays which combines a coarse 3D map with a generative model to output photo-realistic images of the scene, and the system shows the ability to fill in missing features and upgrade texture details of images rendered from the 3D map. The system also features the continuous updates of the 3D map of the remote environment to handle dynamic events such as object relocation and new objects entering the scene while maintaining real-time performance regardless of communication delays.
- A joint state estimation pipeline built from a combination of image segmentation and domain transfer methods which were found to provide the best performance in experiments. We also provide extensive validation of the joint estimation system’s experimental performance in a variety of settings including different background types and illumination conditions, and assessing the resulting performance qualitatively against a ground truth. The proposed system, upon demonstration, performs as well or better than the very recent state-of-the-art work [43] which includes a trained network for the Baxter robot.

## **1.2 Literature review**

### **1.2.1 Sub-domains of computer vision**

The high-level goal of computer vision is to mimic the human brain’s ability to understand the scene in order to take appropriate actions [6]. Many sub-domains of computer vision have been developed in order to fulfill parts of this

high-level goal. For instance, 3D reconstruction (e.g. Kinect Fusion [55]) is used to reconstruct a 3D scene model from depth imaging sensor data.

Thanks to ongoing progress in hardware (GPU) and its ability to handle massively parallel calculations, machine learning has greatly expanded the domain of computer vision, such that high-dimensional information can be extracted from images. For instance, object detection (e.g. YOLO [62]) performs localization and classification of different objects appearing in an image. Machine learning has also been used for image processing. Generative adversarial neural networks (GANs) [24] have the ability to learn the representation of images and apply these learned models to new images. Image style transfer [33] is an example of the application of GAN. It learns characteristics from a set of images (e.g. paintings from artists) and given new images from a digital camera, these painting styles can be applied to the photos taken. In this thesis, we adopt many computer vision techniques to complete various tasks such as image preprocessing, object detection, domain adaptation, etc.

### **1.2.2 Applications of machine learning to engineering problems**

Early applications of machine learning are almost as old as the modern personal computer. MADALINE (Many ADALINE) [80] is an early example of machine learning in signal processing. MADALINE is a multi-layer feed-forward neural network, and each neuron is an ADALINE (Adaptive Linear Neuron) with multiple input nodes and one output node. MADALINE was used for echo elimination in phone lines [79]. Rumelhart et al. [65] first proposed back-propagation (BP) for training of feed-forward neural networks, which computes the gradient of the loss function of weights of the network. BP enables the efficient training of multi-layer neural networks and thus made larger and deeper neural networks feasible. As machine learning and its integration with computer vision continued to grow, more and more engineers and researchers began to adopt these techniques and solve problems in their fields. For example, [15] demonstrates a learning-based controller for an autonomous helicopter which can learn from human expert pilots.

Machine learning is now heavily involved with the automation of machines (e.g. passenger vehicles, heavy-duty construction equipment), where many key features are now handled using machine learning. For example, prior to learning-based methods, objects were usually recognized by their shapes or colors [61], which can be ambiguous in some cases. By contrast, YOLO [62] can detect up to 49 objects, or even 9000 objects in YOLO-9000 [63] precisely in real-time. For driving control, conventional control algorithms rely on a dynamic model of the vehicle. However, end-to-end learning (e.g. [8], [9]) enables creating a model-free controller, which inputs perceived images and directly outputs the control commands. For manipulator automation, machine learning has been employed to estimate joint states [43], which is otherwise impossible without high-precision sensors.

These applications have inspired us to solve other engineering problems using machine learning.

## 1.3 Outline of the thesis

This chapter discussed the motivations of this thesis, its contributions as three specific projects, and a brief review of related literature.

Chapter 2 provides a background about the hardware used throughout this thesis, including the imaging sensors and the Baxter two-armed industrial robot. We cover the mathematical details of camera models and camera calibration procedures, and use these in subsequent chapters.

Chapter 3 covers the details of the learned self-driving vehicle project, including a survey of related works in self-driving vehicles, the hardware and software setup of the vehicle, and experimental results. We focus on evaluating the lane-keep and traffic sign detection modules of the proposed system and show that it outperforms human drivers through both qualitative and quantitative experiments.

Chapter 4 describes the background of robot teleoperation and early adoptions of predictive display techniques. We then cover the details of our proposed software pipeline, including data preprocessing, generative model learn-

ing, online model updating and robot pose correction. Finally, we evaluate the performance and repeatability of the proposed predictive display system in different scenes.

Chapter 5 details the implementation of a vision-based joint state estimation pipeline, which uses external imaging sensors. The composition and setup of the system are discussed, including instance segmentation, domain adaptation and joint detection. The procedures to generate training and ground-truth datasets is covered. Finally, the performance of the proposed toolchain is assessed through experiments.

Chapter 6 summarizes all the results of this thesis. Limitations identified in the three projects and potential future research directions are described.

# Chapter 2

## Hardware Background

This section focuses on the hardware used in the three projects, including the overview of the hardware, the specification, etc. The detailed setup, however, is covered in each project section respectively.

### 2.1 Baxter robot

Baxter is a robot built by Rethink Robotics (shown in Figure 2.1), which is mainly used for educational purposes. The Baxter robot has two arms, each has 7-DOF (degrees of freedom). There is one RGB camera and one infrared camera (IR) at each cuff as demonstrated in Figure 2.3. However, in both projects that utilized the Baxter robot (External vision-based joint detection and predictive display projects), we do not use the onboard cameras from the OEM. Instead, we use third-party RGB cameras and RGB-D cameras which are covered in the next two sections.

Baxter robot is packaged with a customized Linux operating system installed on the onboard PC. The data is transmitted/exchanged via a dedicated switch (Figure 2.2). Baxter has ROS running by default, and it publishes all the topics for camera feeds, IMU data, joint poses and other sensor readings. External PC can get access to all these data when plugged into the switch.

Another noteworthy feature that we find useful during the experiment is the recording/playback function. This feature allows the user to record the arm's trajectory by manually move the arms. Compared to predefined arm trajectory from coding, this feature enables the user to move the arm freely,



Figure 2.1: Baxter robot

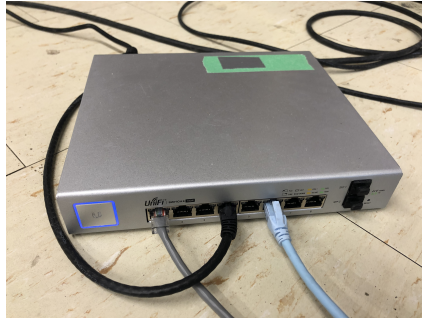


Figure 2.2: The switch used for communication between the Baxter robot and external device

and then replicate the motions recorded, which drastically increase the testing efficiency. We use this feature to record all the datasets used in the external vision-based joint detection and predictive display projects.



Figure 2.3: Monocular camera and IR camera mounted on the cuff

## 2.2 Computer vision preliminaries

This section describes the computer vision concepts and algorithms used in the projects, which include fundamental concepts and the calibration process of the camera.

### 2.2.1 Monocular camera model

A range of monocular cameras was used throughout the projects, each with different characteristics (e.g. field-of-view, resolution, focal length). Modelling these characteristics is essential in almost all computer vision applications. One way to numerically describe these characteristics is through a camera model.

We start by assuming the monocular camera can be defined as a thin lens model, in which all rays entering the lens parallel to the optical axis will intersect at a single focal point. The distance between the focal point and the optical center of the lens is called the focal length.

As shown in Figure 2.4,  $C$  is the optical center of the lens,  $F$  is the focal point, and there is a point  $p$  located at a distance  $Z$  from the thin lens. Point  $p$  is projected onto the image plane at  $p'$ , located at a distance  $z$  from the lens.

By similar triangles the following equation is obtained:

$$\frac{1}{f} = \frac{1}{Z} + \frac{1}{z} \quad (2.1)$$

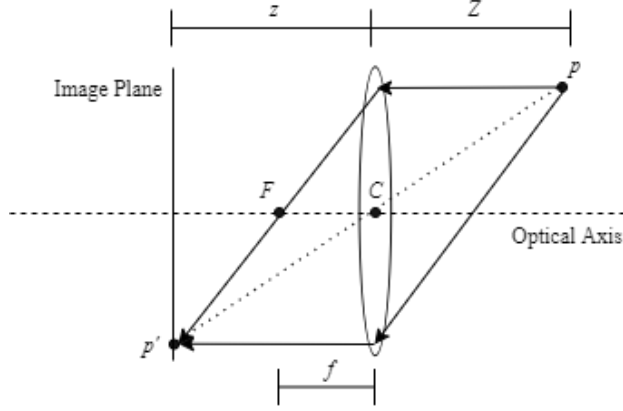


Figure 2.4: Camera thin lens model

### Pinhole camera model

The thin lens model can be further simplified to a Pinhole camera model, in which the aperture size tends to zero. In this model, all rays must pass through the optical center  $C$  to get projected onto the image plane. In this simplified model, the distance between the image plane  $I$  and the optical center reduces to the focal length or  $z = f$ , as illustrated in Figure 2.5.

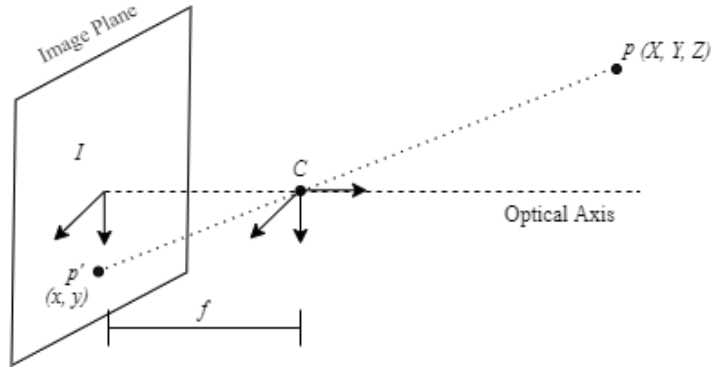


Figure 2.5: Pinhole camera model

Now we define the optical center  $C$  to be the origin of the camera lens fixed reference frame, and we denote the coordinates of the point in space  $p$  with respect to  $C$  as  $(X, Y, Z)$ . Additionally, the point projected onto the image plane  $p'$  has coordinates  $(x, y)$  with respect to the image frame, a 2D space. By similarity of triangles, we can obtain the following equations:



$$x = -f \frac{X}{Z} \quad y = -f \frac{Y}{Z} \quad (2.2)$$

The negative signs in Equation (2.2) mean the image projected onto the image plane is symmetric about the origin of the optical center (upside-down and mirror-flipped). This is usually compensated by inverting the image plane, which is equivalent to the model illustrated in Figure 2.6.

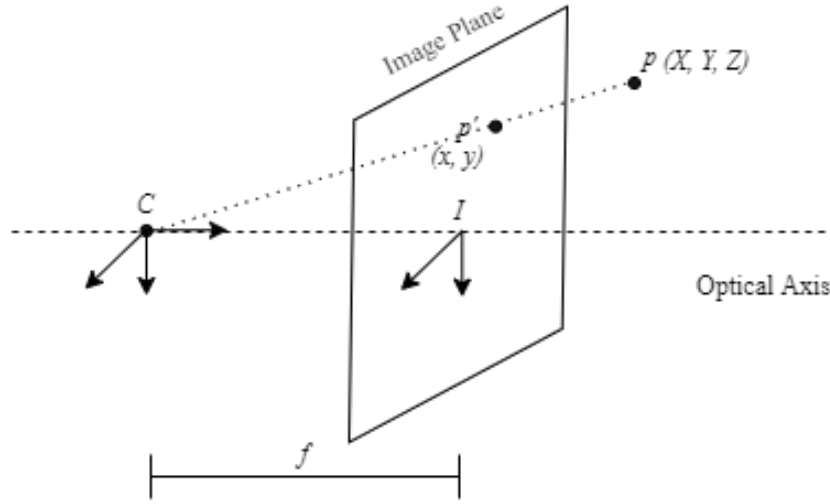


Figure 2.6: Inverted Pinhole camera model

Following this inversion, Equation (2.2) can be rewritten as:

$$x = f \frac{X}{Z} \quad y = f \frac{Y}{Z} \quad (2.3)$$

### Projection matrix

Equation (2.3) can be rewritten in homogeneous coordinates form:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ 1 \end{bmatrix} = \frac{1}{Z} \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P'} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.4)$$

We can decompose the  $3 \times 4$  matrix  $P'$  in Equation (2.4) as the product of  $K_f$  and  $\Pi_0$ :

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K_f} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \quad (2.5)$$

where  $\Pi_0$  is the extrinsic matrix (assuming  $(X, Y, Z)$  is in camera coordinates), and the  $3 \times 3$  matrix  $K_f$  contains the intrinsic parameters of the camera. Note that  $f$ ,  $(X, Y, Z)$  and  $(x, y)$  appearing in the above two equations are in metric units ( $[m]$ ). However, digital cameras represent the image in pixel units ( $[px]$ ). Hence image scaling factors  $[s_x, s_y]$ , in units of  $[px/m]$ , are introduced:

$$\begin{aligned} x_s &= s_x x \\ y_s &= s_y y \end{aligned} \quad (2.6)$$

Following scaling, the coordinates of the point in the image plane are:

$$\begin{aligned} x' &= x_s + c_x \\ y' &= y_s + c_y \end{aligned} \quad (2.7)$$

where  $(c_x, c_y)$  is the coordinate of the *principal point* (intersection of optical axis with image plane) with respect to the image frame's origin. Now we can get the pixel coordinates corresponding to 3D points projected onto the image frame:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \frac{1}{Z} \underbrace{\begin{bmatrix} s_x f & 0 & c_x \\ 0 & s_y f & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.8)$$

where  $K$  is the intrinsic matrix, determined by the optical parameters of the camera. Hence,  $K$  is a constant matrix. We can now define a projection matrix  $P$  which projects any point  $(X, Y, Z)$  in coordinates of the camera-fixed frame to the point  $(x, y)$  is coordinates of the image plane:

$$P = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.9)$$

where  $f_x = s_x f$  and  $f_y = s_y f$ . Note that this expression is valid only when the coordinates of the point  $p$  are expressed in the camera-fixed frame. Now consider a world-fixed frame  $O$  and assume the coordinates  $(X, Y, Z)$  of point

$p$  are expressed in the world-fixed frame. In this case, the projection matrix has the more general form

$$P = K \Pi = \overbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}^{\text{Intrinsic Matrix}} \overbrace{\left( \underbrace{\begin{pmatrix} I & | & \mathbf{t} \end{pmatrix}_{3 \times 4}}_{\text{3D Translation}} \underbrace{\begin{pmatrix} R & | & 0 \\ 0 & | & 1 \end{pmatrix}_{4 \times 4}}_{\text{3D Rotation}} \right)}^{\text{Extrinsic Matrix}} \quad (2.10)$$

where  $(R, t) \in SO(3) \times \mathbb{R}^3$  is the pose of the world-fixed frame w.r.t. the camera-fixed frame.

### 2.2.2 Camera calibration

In Section 2.2.1, we discussed the intrinsic and extrinsic matrices in a pinhole camera model and defined the projection matrix. In some computer vision applications, wide-angle camera lenses are used, which introduce the effect of barrel distortions to images. In order to compensate for this effect, we need to determine the distortion coefficients. Remark the camera model and projection matrix discussed in Section 2.2.1 do not take into account distortion effects. Camera calibration is a process that identifies a camera's intrinsic matrix as well as these distortion parameters. We will use the camera calibration module from OpenCV which determines:

1. Distortion coefficients
2. The  $3 \times 3$  intrinsic matrix of the camera

The image distortion effect is modelled by five parameters  $(k_1, k_2, k_3, p_1, p_2)$ , the first three modelling radial distortion, the second two modelling tangential distortion. The distorted image point coordinates  $(x_d, y_d)$  in the image plane  $I$  are mapped to the corresponding undistorted image point coordinates  $(x, y)$  as

$$\begin{aligned} r^2 &= x_d^2 + y_d^2 \\ x &= x_d (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_d y_d + p_2 (r^2 + 2x_d^2) \\ y &= y_d (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_2 x_d y_d + p_1 (r^2 + 2y_d^2) \end{aligned} \quad (2.11)$$

where both  $(x_d, y_d)$  and  $(x, y)$  are expressed in metric units ([m]).

A typical process to calibrate the camera is to use a printed chessboard with known dimensions (Fig. 2.7). Users specify the dimensions of the chessboard



Figure 2.7: The chessboard for camera calibration

in terms of the number of rows and columns and the physical length of each square. The calibration algorithm identifies corners within the chessboard (as shown in Figure 2.8) and uses the given information about the chessboard to calculate the distortion coefficients and the intrinsic parameters of the camera.



Figure 2.8: Corners extracted from the chessboard for camera calibration

As a numerical example, the distortion coefficients and the camera intrinsic parameters of a Raspberry Camera V2 (in  $640 \times 480$  resolution mode) were

found to be

$$k_1 = 0.176911, k_2 = -0.297734, k_3 = -0.004644, p_1 = 0.002276, p_2 = 0.000000$$

$$K = \begin{bmatrix} 511.4165 & 0 & 325.6115 \\ 0 & 511.9490 & 241.2863 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.12)$$

## 2.3 Imaging sensors

### 2.3.1 RGB camera

RGB cameras used in our projects are referred to as monocular RGB cameras or the RGB sensor from a depth camera (e.g. RGB-D camera). Stereo cameras (e.g. ZED camera) may also have RGB sensors, but they provide depth information from a dual-camera setup. All three projects have the uses of RGB cameras, and they only record RGB channels of the scene. The most commonly used RGB camera in our projects is Logitech C920. Figure 2.9 shows the appearance and mounting mechanism of the C920 camera. The webcam has an auto-focus feature, which may be preferable for average consumers, but this feature brings uncertainty to the experiment, as the camera intrinsic parameters are changed when the focus changes. As covered in Section 2.2.1, the camera model is essential in almost all computer vision applications. Therefore, the auto-focus feature is turned off in all the projects which use this webcam.



Figure 2.9: Logitech C920 webcam

Another RGB sensor which is solely used in the self-driving project is a Raspberry Pi camera module v2 (see Figure 2.10). The camera has a ribbon

cable connection which enables faster data transmission speed than the USB port on the Raspberry Pi board V4.

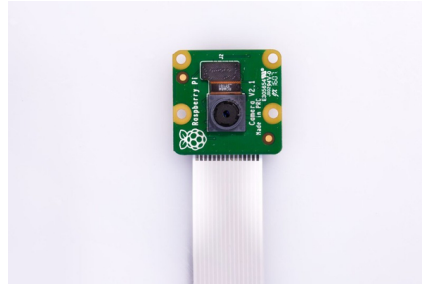


Figure 2.10: Raspberry Pi Camera Module V2

### 2.3.2 RGB-D camera

RGB-D cameras provide both RGB image and 2D depth images. Depth information is very useful in 3D reconstruction, scene segmentation, SLAM, etc. There are two RGB-D models in our lab: Intel RealSense D435i and D415. The main RGB-D sensor used in our project is D415. Compared to D435i, D415 has a smaller Field-of-view (FOV), with a roller shutter, which provides higher depth accuracy.

### 2.3.3 Vicon tracking system

As an image-based motion tracking system, the Vicon tracking system is widely used in video game design, film-making, etc. The system, Vicon Vero (see Figure 2.11), is used in the external vision-based joint detection project, and the object poses provided by the system are considered ground-truth.

Vicon uses infrared cameras to capture infrared rays from reflective surfaces [76]. Despite various noises from the ambient infrared rays reflected from all kinds of surfaces, the tracking system is optimized for picking up the rays specifically from the tracking markers (shown in Figure 2.12), which are coated with retroreflective surfaces.

The system has multiple Vero capture cameras, which observe the capturing volume from different perspectives, and use point triangulation to calculate

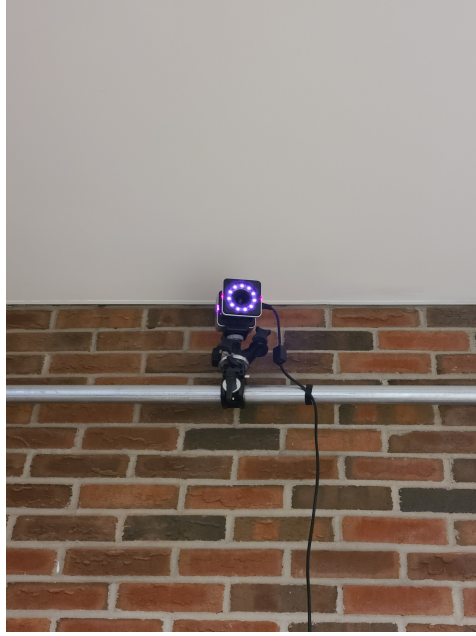


Figure 2.11: Victron Vero capture camera

the 3D coordinates of markers in the predefined coordinate. By grouping different markers, the user can define and track individual objects. In addition, the geometric centre of each object can be customized. This feature allows the alignment of the geometric centre of the defined objects to the mathematical centre of the object represented in URDF files.



Figure 2.12: Victron Vero capture camera

## 2.4 On-board processor

Raspberry Pi V4 and Arduino Mini are used in the self-driving RC project. Due to battery limitation, the on-board processors have less processing power

than desktop PC. Therefore, Arduino Mini is in charge of command executions, whereas Raspberry Pi serves as a data exchange centre, in which it parses all data coming from Arduino Mini, and send the commands to the Arduino Mini.

## **2.5 Off-board processing unit**

With more processing power, the off-board processing unit used in the projects is a desktop PC. The desktop PC is equipped with 6-core Intel Core i7-8700K and a graphics processing unit (GPU), which enables parallel computation in deep learning applications and other matrices operations.



# Chapter 3

## Learned Self-Driving Vehicle

### 3.1 Introduction

Thanks to the extensive work being carried out in computer vision and machine learning, as well as the progress in computer hardware, self-driving vehicles are becoming more realistic and practical. Transportation plays a significant role in modern society, and the self-driving system could drastically improve transportation efficiency, and avoid many traffic injuries or even fatalities. Although many systems have been developed already a few decades ago to assist human drivers (e.g. vehicle adaptive cruise control system [51]), fully autonomous vehicles are still very rare on the road due to inadequate capabilities of the current systems and the legal gap for autonomous vehicles. Inevitably, fully autonomous vehicles will become more common in the near future, and reshape the transport industry forever.

As a very sophisticated system, a fully autonomous driving system has many components, including SLAM (simultaneous localization and mapping), motion planning, traffic sign detection, obstacle avoidance, etc. In this chapter, we aim to design and build a simplified self-driving platform (a RC car with 3D printed elements), implement some of the core functionality of a self-driving vehicle system, and finally test the resulting design in real-world scenarios. We also utilize Carla [20], a sophisticated simulator for autonomous driving research to test our method under more realistic scenarios.

Another noteworthy observation is that deep neural network have demonstrated the ability to control a non-linear dynamic system without a priori

knowledge of its model. For example, in order to control a wheeled ground vehicle (e.g. our RC car) to follow the lane, a nonlinear closed-loop control law would need to be designed and tuned. As an alternative, a CNN (convolutional neural network) that learns the connection between input images from the onboard camera and outputs the resulting controls could entirely replace the conventional control system design, as will be proven by our experiment.

In this chapter we focus on the hardware and software setup of a DIY remote control (RC) car which learns to autonomously operate itself for both lane keeping and obeying traffic stop signs.

## **3.2 Related works**

### **3.2.1 Vehicle automation level standard**

As more and more automotive manufacturers are introducing driving automation systems into their products, and with many of their commercials being misleading to the general public, the Society of Automotive Engineers (SAE) has set up a standard [73] to describe the level of automation of these vehicles, ranging from no-automation (level 0) to fully-automation (level 5) for a total of six levels. Although many vehicles are already equipped with multiple sensors which are capable of providing driving automation, SAE standard J3016 focuses on the engagement of the autonomous features under driving conditions. For example, Elon Musk, the CEO of Tesla, has claimed that his company's cars achieve the same functionality and accuracy as LiDAR using only camera sensors and vision technologies [22]. This allows achieving the same level of autonomy as competing vehicles from e.g. Waymo, despite the latter using much more expensive LiDAR sensors.

### **3.2.2 Image-based driving automation**

#### **Lane detection and curvature estimation**

One of the essential features of an autonomous driving system is controlling steering and throttle inputs according to the road conditions. To achieve this, a classic approach is to decompose the problem into smaller intermediate steps.

For example, the first step can be lane detection. Because lane markings are painted using high-contrast colours, they can be easily masked out using color filters. In addition, since lane marks or curbs typically form continuous lines, one can further optimize the filtered image using the Hough line transform [21].

Once the center lane or the roadside curbs are filtered out from the images, one approach to calculate steering angle outputs is to estimate the curvature of the detected lane. For instance, [30] uses images captured from RGB cameras, and then tries to fit a simple parabolic model to the detected curb. To overcome the issue where lanes are not perfect low-order polynomial functions, much research has been conducted to represent the lanes using more sophisticated models. [77] proposes B-snake, a higher-order fitting function which also accounts for camera intrinsic parameters. This method uses B-splines to fit the lane boundaries, which can take any shape. Additionally, [77] utilizes CHEVP and Minimum Mean Square Error (MMSE) to find respectively the initial position and control points for the B-snake. This method was shown to achieve good robustness under different road conditions.

### **Vehicle control based on lane curvature**

Many lane-following features on modern cars control the steering inputs based on the lane curvature [10], [41]. [41] uses onboard cameras to estimate road information, such as lateral offset and lane curvature, using a parabolic equation model. Then the method uses the dynamic model of the vehicle, along with a pre-defined lane change completion time, to estimate the lateral position and yaw angle of the vehicle on the road. The dynamic model inputs the lateral position on the road, vehicle speed, and yaw angle to calculate a steering angle command. The vehicle steering system executes this command to complete the lane-keeping feature. This systems performs well provided road conditions are clear because it relies on a good lane representation model. When road conditions are bad (e.g. snow-covered or muddy roads, where lane markings are not visible), the system is unable to provide reliable steering control commands.

## Deep learning-based vehicle control

While earlier methods required a lane model to describe road curvature and position of the vehicle, the rise of deep neural networks has inspired new research directions in self-driving vehicles. Many deep learning-based self-driving methods (e.g. end-to-end learning [8] and PilotNet [9]) are reported to provide robust performance even in poor visual conditions. [8] proposes end-to-end learning, which trains a convolutional neural network which inputs raw images directly from the front-facing cameras and then outputs resulting steering commands. Unlike other methods which consist of many intermediate steps (e.g. lane detection followed by vehicle control), [8] argues that end-to-end learning methods will eventually achieve better results because of their internal self-optimization process. This claim is based on the fact that human decisions are made by decomposition of the problem, which may provide the most optimal solution. Also, [8] points out that a smaller network can still provide good performance, which translates to faster processing times. The training data is collected from a camera mounted on the car, as well as the steering angle at each frame. To overcome the issue where the network has a bias to go straight, [8] increases the proportion of frames that represent the road curvatures. Overall, [8] and [9] demonstrate the potential and robustness of end-to-end learning-based self-driving designs.

### 3.2.3 Traffic sign recognition

Traffic signs (e.g. stop signs, yield signs, speed limit signs) play a crucial role in driving because they impose rules and requirements which vehicles need to follow. Therefore, traffic sign recognition is one of the essential components of a self-driving car system. In general, there are two ways of solving traffic sign recognition problems: 1) classic computer vision (e.g. color thresholds, shape detection) 2) Neural networks [19]. The first approach is able to identify the shape and the color of the traffic signs (e.g. round and triangle), but these methods usually fail to understand the context of the signs, because traffic signs may have identical shapes or colors. Conversely, using conven-

tional computer vision techniques results in very fast processing times, which is crucial for self-driving projects. Therefore, this method is good for traffic sign detection but not ideal for classification. [19] lists two means to detect traffic signs:

1. Color thresholding: Generating a mask with only pixel intensity in a color channel within a certain range.
2. Corner detection and extraction: Using optimal corner detector [61] to locate and identify the shape of the traffic signs.

With the vast progress in deep neural networks (DNNs) and improvements in hardware, many researchers have studied the traffic sign detection and classification problem using neural networks (e.g. Multi-column deep neural networks [14], or generic object classifiers such as YOLO [62]). Neural networks usually require a large amount of training data to perform accurately. Therefore there are some big collections of labelled traffic sign data (e.g. German traffic sign detection benchmark, or GTSDb [31]). These learning-based methods have shown to overcome many limitations of non-learning based methods (e.g. image background interference, traffic sign context interpretation). Notably, some of the DNN-based solutions have achieved better recognition performance than humans [14].

### **3.2.4 Decision tree**

Driving a vehicle on the road is a complex task. The top-level goal can be summarized as travelling from point A to point B, but computers need to break this abstract task into multiple sub-tasks (e.g. lane keeping, lane changing, traffic signs detection, obstacle avoidance). Although each individual task may be accomplished by a simple closed-loop control system, the overall system needs a sophisticated design to interface the different modules and make the final decision on vehicle inputs. One way to construct this system is to use a decision tree approach. Li et al. [44] propose POMDP, an explicit decision tree structure for the self-driving system. [44] uses a discrete-time Markov chain

to model the vehicle, with each node in the decision tree corresponding to an action. The driving objectives are represented by reward functions. Therefore, the problem is to find the optimal series of actions (or policy) that achieves the highest cumulative reward.

### 3.3 DIY remote control (RC) car under lab settings

In this section, we describe the hardware and software setup of our developed RC car. Due to time and cost constraints, this experiment is mainly a proof-of-concept and is tested in a simplified environment within the lab.

#### 3.3.1 Dynamic model of the differentially-steered vehicle

In contrast to most real-world passenger vehicles that have steerable wheels, the DIY vehicle built in this project is steered by different speed of the wheels on the each side of the vehicle. As shown in Figure 3.1, the schematic kinematic model of the vehicle, the front wheel of the vehicle is a non-driving wheel and it is solely for stability. The derivations and the kinematic model of the differentially-steered vehicles are based on [17].

We denote the vehicle frame as  $\{B\}$  and the world frame as  $\{O\}$ , the instantaneous center of rotation as  $ICR$ , the width between the wheels is  $W$ , and the rotation angle of frame  $\{B\}$  relative to frame  $\{O\}$  is  $\theta$ . Assuming there is no sideways slip, the velocity of the vehicle (in frame  $\{B\}$ ) is:

$$B_v = (\nu, 0) \quad (3.1)$$

We let  $R_L$  and  $R_R$  denote the radius of curvature for left wheel and right wheel respectively, as shown in Figure 3.1. The angular velocity of  $\{B\}$  is obtained:

$$\dot{\theta} = \frac{v_L}{R_L} = \frac{v_R}{R_R} \quad (3.2)$$

Note that  $R_R = R_L + W$ , so we can rewrite Equation (3.2) as:

$$\dot{\theta} = \frac{v_R - v_L}{W} \quad (3.3)$$

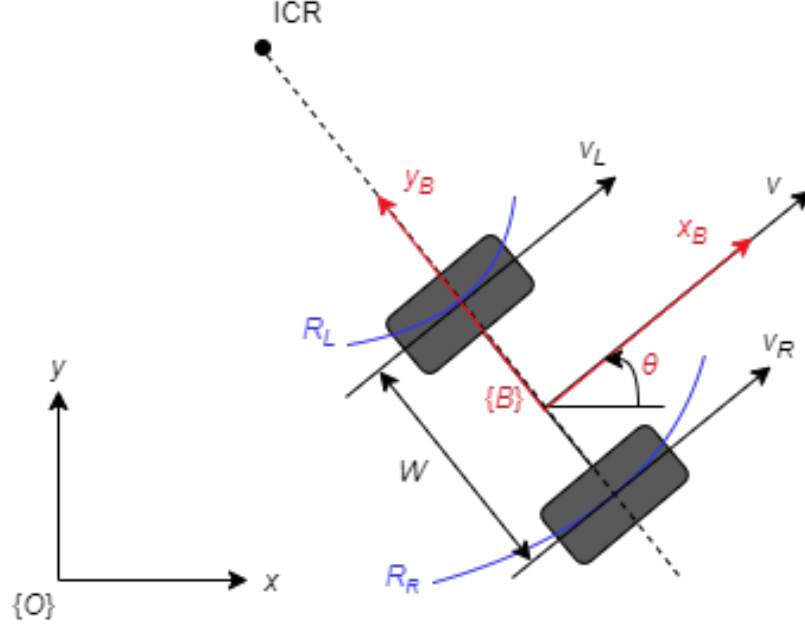


Figure 3.1: The schematic diagram of the kinematic model of the RC car

Let  $v = \frac{v_R + v_L}{2}$  and  $v_\Delta = v_R - v_L$  denote the average and differential velocities of the vehicle respectively, we can obtain the following equations of motion:

$$\begin{aligned} \dot{x} &= v \cos \theta \\ \dot{y} &= v \sin \theta \\ \dot{\theta} &= \frac{v_\Delta}{W} \end{aligned} \tag{3.4}$$

With this dynamic model (3.4), we can theoretically build a closed-loop controller. However, while the above model uses continuous states and inputs, the actual system is driven by discrete-valued (discontinuous) inputs, meaning classical nonlinear feedback design methods do not directly apply. This is a strong motivation for implementing a DNN-based end-to-end learning design to control the vehicle.

### 3.3.2 Hardware setup

The system consists of three major components:

1. Mechanical parts and motors: We designed the chassis of the RC car in SolidWorks, outputted the assembly as an STL file, and then 3D printed

the parts. We measured the dimensions of the logic boards and sensors to assure the designed assembly can hold all the parts. The vehicle is a three-wheel system. We used two direct-current (DC) motors to actuate the RC car. The front wheel is a non-driven wheel used to support the chassis. The complete system is shown in Figure 3.2.

2. Logic boards or on-board processors: There are two logic boards installed on the vehicle: an Arduino Mini and a Raspberry Pi v4. The Arduino Mini controls the two DC motors, and it receives its commands from the Raspberry Pi. The Raspberry Pi receives images from the Raspberry Camera as well as ultrasonic distance-measuring sensors, and communicates with a remote PC. Due to the computational limitations of the Raspberry Pi, and the self-driving features relying on neural networks (which require a GPU for real-time operation), all decisions are computed and sent out from the remote PC. In future revisions, we can get rid of Arduino Mini since the Raspberry Pi board has GPIO ports which enable direct control of the DC motors.
3. Sensors: We installed three different types of sensors on the self-driving RC vehicle: an RGB camera (Raspberry Pi Cam), an ultrasonic distance sensor (HC-SR04), and a scanning Lidar (RPLidar). The RGB camera is the primary sensor and provides RGB images of the front of the vehicle. This information is used to detect the road, calculate road curvature, recognize traffic signs, and identify the target. The ultrasonic sensor is used for obstacle avoidance. The sensor has one ultrasonic sound wave emitter and one receiver. It measures the distance between the reflected surface (e.g. obstacles) and the receiver. The final sensor is the Lidar, which measures distances in a complete circle around the vehicle to return a scan. The resulting laser scans provide 2D mapping information, which is useful for navigation and path planning.



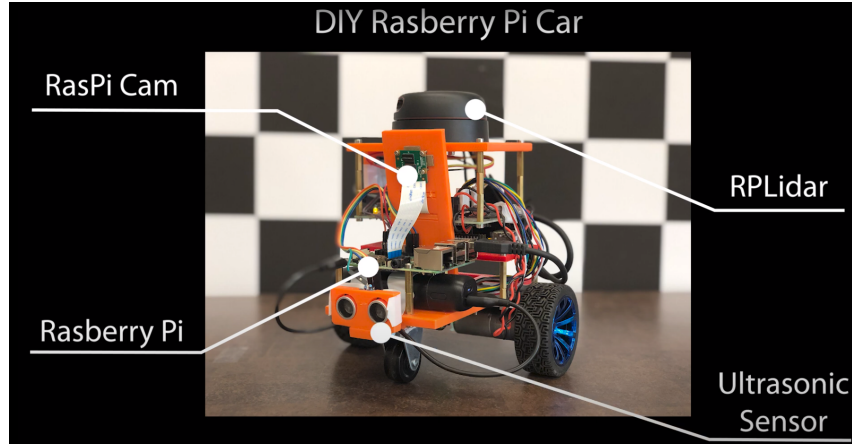


Figure 3.2: System overview of the self-driving RC car with notations

### 3.3.3 Software setup

In this section, we discuss the methodology and configuration of the software packages in the self-driving RC car.

#### System overview

We implement three core features for the self-driving RC vehicle:

1. Road detection and lane following: Controlling the steering of the vehicle such that it stays within the center of the lane
2. Traffic signs and signals recognition: Detecting traffic signs and signals along the road, and following their prescribed rules (e.g. stop at the stop signs and wait for three seconds)
3. Obstacle avoidance: Using ultrasonic sensors to detect and avoid hitting obstacles

As shown in Figure 3.3, due to the inadequate computational resources in the Raspberry Pi and Arduino, we use a powerful offboard PC to run the heavy-duty neural network and decision tree pipeline and make the decision for the RC car. The Raspberry Pi board is in charge of collecting sensor data from the camera and ultrasonic sensor and transmitting this data to the remote PC via a WiFi connection. The Arduino board serves as the command execution center and drives the two DC motors according to commands received.

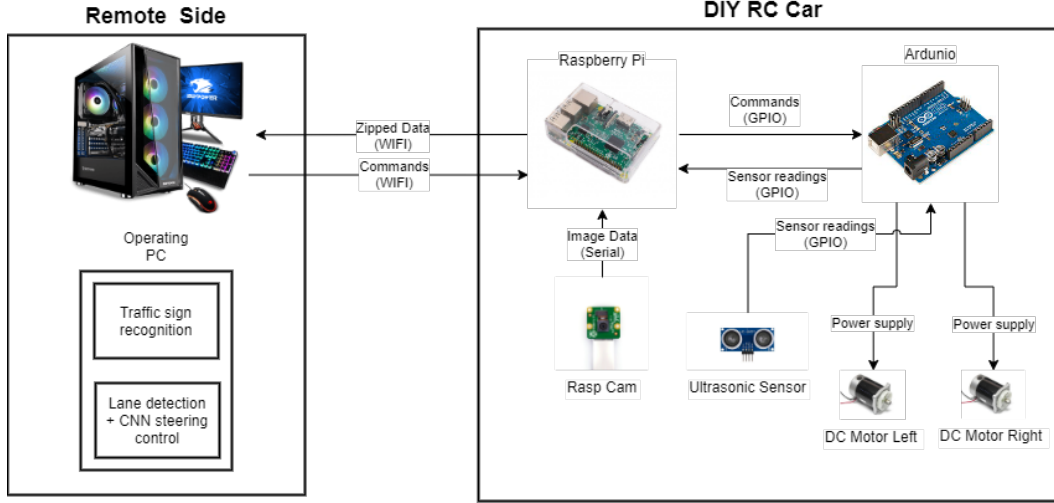


Figure 3.3: An overview of the system of the self-driving RC car

### Road detection

To simulate a road, we use yellow tape to represent lane markings (as shown in Figure 3.4). The road forms a closed loop so the vehicle can drive continuously without human intervention. As discussed in Sec. 3.2.2, there are two main ways to detect the road curvature and follow the lane. We tested both approaches: lane model estimation and end-to-end learning.

The first step is to use color thresholds to filter the lane markings from the noisy background. The lane markings are yellow to match the actual color on the road. Because the color is very distinct from its surroundings, color thresholding is the natural solution to filter these lane markings. The default color channels are RGB channels, representing red (R), green (G), and blue (B) information. However, this color representation is too sensitive, and a small change in color may have a large impact on one or more channel. This means RGB images make it difficult to find the correct color ranges for the lane markings. On the other hand, HSV (Hue, saturation, and value) representation is a better alternative to color representation, because unlike RGB channels, the single hue channel is most sensitive to color type. S and V channels control the saturation and brightness of the color. Hence, we converted the camera image from RGB to HSV space (see Figure 3.5), and performed thresholding



Figure 3.4: An image of the lane markings in the lab captured from the Raspberry Camera mounted in front of the vehicle

based on the values of the three channels.

$$\begin{cases} \text{if } H_{low} \leq th_H \leq H_{High} \\ \text{if } S_{low} \leq th_S \leq S_{High} \\ \text{if } V_{low} \leq th_V \leq V_{High} \end{cases} \Rightarrow \begin{cases} th_H = 1 & \text{else } 0 \\ th_S = 1 & \text{else } 0 \\ th_V = 1 & \text{else } 0 \end{cases} \quad (3.5)$$

As listed in Equation (3.5), we find the upper and lower bounds (Equation (3.6)) for each channel, and then threshold the image to form a binary mask as shown in Figure 3.6.

$$\begin{cases} H_{low} = 20 & \text{and } H_{high} = 35 \\ S_{low} = 50 & \text{and } S_{high} = 255 \\ V_{low} = 100 & \text{and } V_{high} = 255 \end{cases} \quad (3.6)$$

### Lane curvature estimation

Once the lane markings are identified, the next step is to estimate the lane curvature and control the vehicle based on the lane model. The Raspberry PI camera is calibrated offline to obtain its intrinsic parameters. OpenCV provides a built-in function (`cv2.undistort`) to undistort the image given the camera intrinsic matrix. In order to estimate the curvature of the lane, we

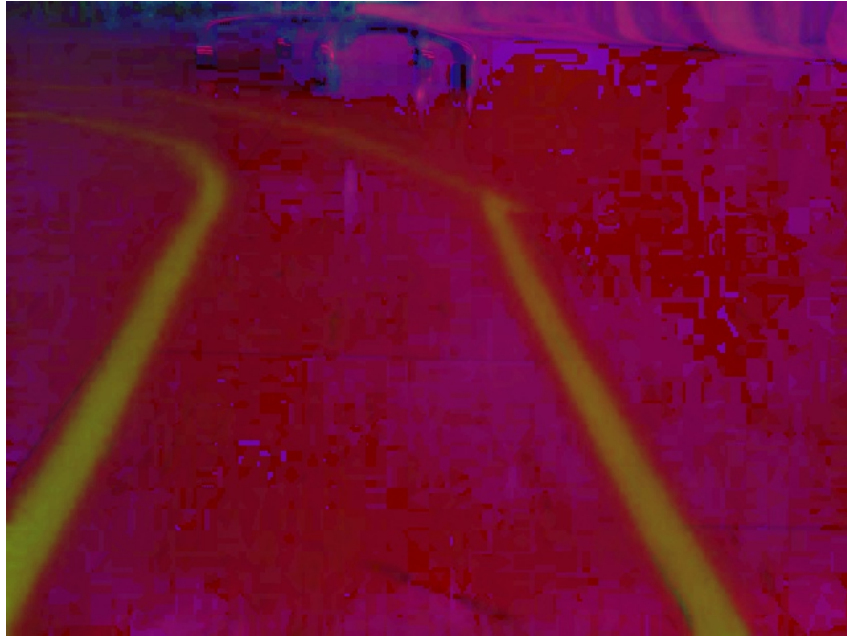


Figure 3.5: The image of the lane markings after the conversion from RGB space to HSV space



Figure 3.6: The binary image of lane masking after the HSV thresholds have been applied

also need to apply perspective transform to the undistorted image to get a birds-eye view of the lane. The perspective transformation process uses four image correspondences to calculate the homography matrix which is applied to the original image. We use the junction lines between tiles on the floor as a reference line to find the destination points. A sample output image is shown in Figure 3.7.



Figure 3.7: The output image of lane markings after the perspective transformation

We use a parabolic model  $x = ay^2 + by + c$  to fit the curvature of the lane from the unwarped binary image, where  $x$  and  $y$  are the horizontal and vertical coordinates of the pixels respectively.

The first step is to locate the pixel locations of the lane within the image. Searching the entire image would be tedious and inefficient. Therefore, we use sliding windows to find the lane pixels(Figure 3.8). Once we locate the locations of the pixels of the lane markings, we can use a polynomial fitting function to find the coefficients  $a$ ,  $b$  and  $c$  of the parabolic equation. Since each non-zero pixel has a unique coordinate  $(x, y)$ , polynomial fitting is applied to these pixel coordinates. However, having a parabolic equation to describe the lane is not our end goal for vehicle control. Rather it is an intermediate step

to get the following core values for the steering: 1) Radius of curvature and 2) Lateral offset of the vehicle relative to the lane center.

$$R_{\text{curve}} = \frac{\left[1 + \left(\frac{dx}{dy}\right)^2\right]^{3/2}}{\left|\frac{d^2x}{dy^2}\right|} \quad (3.7)$$

$$\frac{dx}{dy} = 2ay + b \quad (3.8)$$

$$\frac{d^2x}{dy^2} = 2a \quad (3.9)$$

Equation (3.7) shows the formula for the radius of curvature. The first and second-order derivatives of the parabolic model are given in Equation (3.8) and (3.9), respectively. Therefore, the radius of curvature is given by Equation (3.11). The lateral offset of the vehicle relies on the world metric distance to pixel ratio  $r_{\text{meter per pixel}}$ . Assuming the camera is mounted at the center of the vehicle, the lateral offset can be calculated as:

$$x_{\text{offset}} = (\text{Image Width}/2) * r_{\text{meter per pixel}} \quad (3.10)$$

$$R_{\text{curve}} = \frac{(1 + (2ay + b)^2)^{3/2}}{|2a|} \quad (3.11)$$

Once the radius of the curvature and the lateral offset of the vehicle relative to the center of road have been determined, the next step is to control the vehicle based on these two pieces of information.

### **End-to-end learning approach to vehicle control**

Another approach we tried on the self-driving RC car is the end-to-end learning method for steering and speed control. The most natural way to solve this problem is to design a neural network that inputs an image and outputs the steering angle and the throttle input to control the vehicle. However, our DIY RC car does not have pivoting wheels for turning like conventional cars. Therefore, we design a novel system to control the vehicle based on the image perceived.



Figure 3.8: Sliding window search of the lane markings. Green blocks are each individual search window

We first implement a simple keyboard input algorithm to control the vehicle. There are four possible states of the system, and their corresponding keyboard mappings are listed below:

1. W: Initiate *Forward state*, where both motors (i.e. left and right) have the same direction and amplitude of speed.
2. S: Initiate *Stop state*, where all commands are zero and the vehicle is stationary.
3. A: Initiate *Left turn state*, in which the left motor has around 70% of the right motor's speed. In this case, the vehicle slowly turns to the left, while still maintaining forward driving.
4. D: Initiate *Right turn state*, in which the right motor has around 70% of the left motor's speed. In this case, the vehicle slowly turns to the right, while still maintaining forward driving.

Unlike a steering angle prediction network, which is a single-output regression network, our network needs to produce only three outputs: the probabil-

ities for each state. We thus build a lightweight convolutional neural network (CNN) for this purpose.

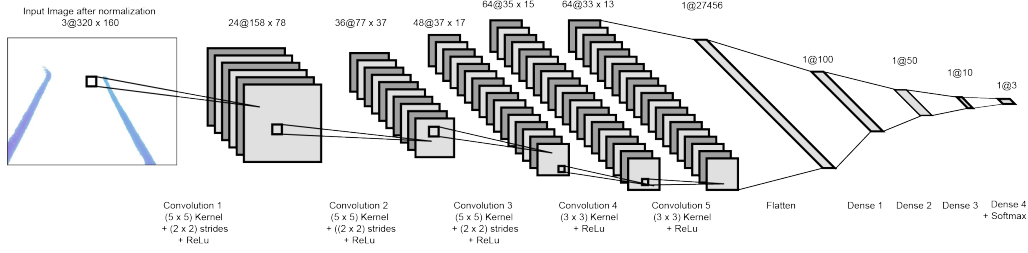


Figure 3.9: The structure of the CNN used in self-driving RC project

As shown in Figure 3.9, our network has 5 convolutional layers. The first three convolutional layers have a  $(5 \times 5)$  kernel size and a  $(2 \times 2)$  stride. This way the network will have fewer parameters to learn while still retaining most of the information from the images. For the dense layers, the first three have no activation function and the last output layer has a *softmax* activation function. This is important because dense layer #3 outputs a vector with 10 real values (which can be negative, zero or positive) and the *softmax* function (Equation (3.12)) converts this vector to a new vector  $\vec{K}$  whose elements sum to 1. Therefore, after this step, each component within the vector  $\vec{K}$  can be interpreted as probabilities. The output dimension is  $(1 \times 3)$ , meaning for each input image, the network outputs a vector  $\vec{K}$  with three elements, each element corresponding to the probability of one command. Although there are four states (as discussed above), we ignore the *Stop state* such that the network handles only the steering for the vehicle. For each perceived RGB frame fed into the CNN, the control action is chosen as the element within  $\vec{K}$  which has the largest probability.

$$f_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (3.12)$$

To train this CNN, we first manually drive the RC car along the track using the keyboard. Simultaneously, we record the RGB image perceived together with the keyboard command given by the operator at the instant the image is



recorded. The keyboard command data is one-hot encoded to match the data type of the output from the CNN. The order of the output command vector is ["W", "A", "D"]. For example, for a pair  $P_a$  with image  $I_a$  and corresponding keyboard input command "A", the training data is

$$P_a : (I_a, [0, 1, 0]) \quad (3.13)$$

Likewise, if some image  $I_n$  is fed into the CNN and the output is  $[0.1, 0.9, 0]$ , this means the probability to execute "W" is 0.1, and the probability of "A" is 0.9. Since we choose the command with the highest probability, in this case we send the command "A" to the Arduino.

In addition, since the track forms a closed loop and at each data collection session we drive the vehicle in one direction, this means that for each training set, the vehicle is driving either clockwise or counter-clockwise. This observation implies the CNN will be biased to make predictions of "A" (turning left, for counter-clockwise training data) or "D" (turning right, for clockwise training data). To overcome this issue, we applied data augmentation to our training set, in which we horizontally flipped the image and corresponding keyboard command (e.g. "A" to "D" or vice versa).

### **Traffic sign detection**

Another important feature in self-driving RC cars is traffic sign detection and recognition. Since there are more than 500 types of road signs, the traffic sign detection pipeline needs to have the ability to detect multiple objects simultaneously. YOLO (You only look once) [62] is one such object classifier which can detect and classify multiple objects at the same time. As listed in Equation (3.14), YOLO has five terms in the loss function. The first two terms correspond to localization loss, the third and fourth terms correspond to confidence loss, and the last term is for classification loss. With this sophisticated loss function, YOLO achieves the best score in many benchmarks compared to other state-of-the-art object classifiers [62]. However, the original YOLO can only detect 49 types of objects, while there are over 500 different types of road signs. To overcome this limitation, Redmon et al. proposed YOLO9000

[63], which can detect over 9000 objects.

$$\begin{aligned}
& \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& \quad + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{K}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{K}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{3.14}$$

To test traffic sign detection performance, we color printed a stop sign and used glue to stick it to a cardboard backing (see Figure 3.10). Although there are many traffic signs, we chose the stop sign since the DIY RC car can respond to the sign and make a full stop. On the other hand, other traffic signs (e.g. yield sign) would need interaction with other vehicles hence the scenario would be much more complex. Therefore, the stop sign is a good choice for a proof-of-concept purpose.

YOLO needs both the location and the class of the objects in the image for training, and more training sets will result in better performance in theory. Hence, we augmented our training sets by randomly placed stop sign patches in various images we collected from the Raspberry Pi camera mounted on the RC car. Each resulting image then has the track and one stop sign in sight.

### Traffic lights detection

Besides road signs, another important piece of road information are traffic lights. One approach is to use YOLO to detect and classify these traffic lights. The training and implementation process remain identical as discussed previously. Another approach which is simple yet robust is to use color thresholdings, contour detection and shape fitting. This approach is based on the fact that traffic lights are circular and have very distinguishable colours. The pipeline for this approach is listed below:

1. Get two binary masks ( $I_G$  and  $I_R$ ): red color thresholding and green color thresholding.



Figure 3.10: The stop sign detection test scene

2. Finds sets of contours  $\{C_G\}$  and  $\{C_R\}$  (using `cv2.findContours`) in both binary masks  $I_G$  and  $I_R$ .
3. Find the center and radius of the circular contour (as shown in Algorithm 1) using a Hough circle transform.

---

**Algorithm 1** Traffic light detection

---

```

1: for  $C$  in  $\{C\}$  do
2:    $a = \text{Area}(C)$ 
3:   if  $(a < th_{lower}) || (a > th_{upper})$  then
4:     Continue
5:   else
6:     if  $C \in \{Circles\}$  then ▷ Check if the contour is a circle
7:       return  $c, r$  ▷ Return the center and radius of the circle
8:     end if
9:   end if
10: end for

```

---

Once this pipeline is implemented, traffic lights can be identified and detected in real time (as illustrated in Figure 3.11).

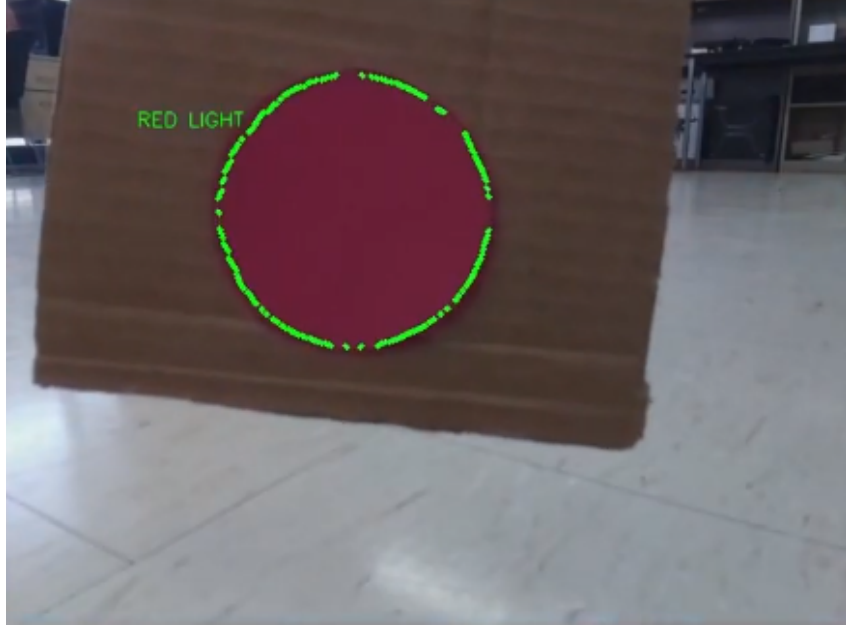


Figure 3.11: The traffic light detector

### Distance estimation from the bounding box and circular contour

Another problem is sign distance estimation based on the bounding box generated by YOLO. YOLO is designed to detect objects at different scales, yet we do not want the vehicle to stop too early when the stop sign is still at a distance. Hence, we need to estimate the distance between the vehicle and the stop sign based on the pixel width and/or height of the bounding box.

Assuming the actual width of the stop sign is  $W$  (which can be measured by a ruler), the pixel width of the bounding box is  $w'$ , and the depth (the distance between the image plane and the actual stop sign plane) is  $Z_w$ , the following expression can be found:

$$Z_w = f_x \frac{W}{w'} \quad (3.15)$$

A similar expression is established for height:

$$Z_h = f_y \frac{H}{h'} \quad (3.16)$$

Both  $Z_h$  and  $Z_w$  represent the distance to the sign. If the width and height resolutions of the image are similar (i.e. the aspect ratio is close to 1), then

we can use the geometric mean  $\bar{x} = \sqrt{x_1 x_2}$  to get a more accurate estimation:

$$\bar{Z} = \sqrt{Z_h Z_w} \quad (3.17)$$

However, if the image has an aspect ratio of 4 : 3 or 16 : 9,  $Z_h$  is more favorable because  $Z_w$  will be too sensitive.

Similarly, we need to estimate the depth to the traffic lights. In this case, the pixel center  $c'$  and the radius  $r'$  are known. Denoting the actual radius of the red light as  $R$ , the following expression can be found for the circular contour:

$$\bar{Z}_r = \sqrt{f_x f_y} \frac{R}{r'} \quad (3.18)$$

### Logical tree control

We design a logical tree that combines each module introduced in above sections into a self-driving system. Since each module has different outputs, a logical tree stitches the outputs from different modules to make a high-level decision and directly control the vehicle. As shown in Figure 3.12, the final outputs of the logical tree are states. Besides the four states (discussed in Section 3.3.3), we add a new “stop sign” state, in which the vehicle stops for three seconds and then reinstates the “forward” state. Therefore, the logical tree can be interpreted as the switch between different driving states.

## 3.4 Experimental results

To illustrate the performance of the proposed self-driving system and the individual sub-modules, we set up three experiments:

1. A non-circular, open-ended track, with traffic lights/signs and obstacles, to test the overall performance of the system and perform qualitative evaluations
2. An enclosed track to test the lane-keeping performance
3. Stop sign detection and distance estimations

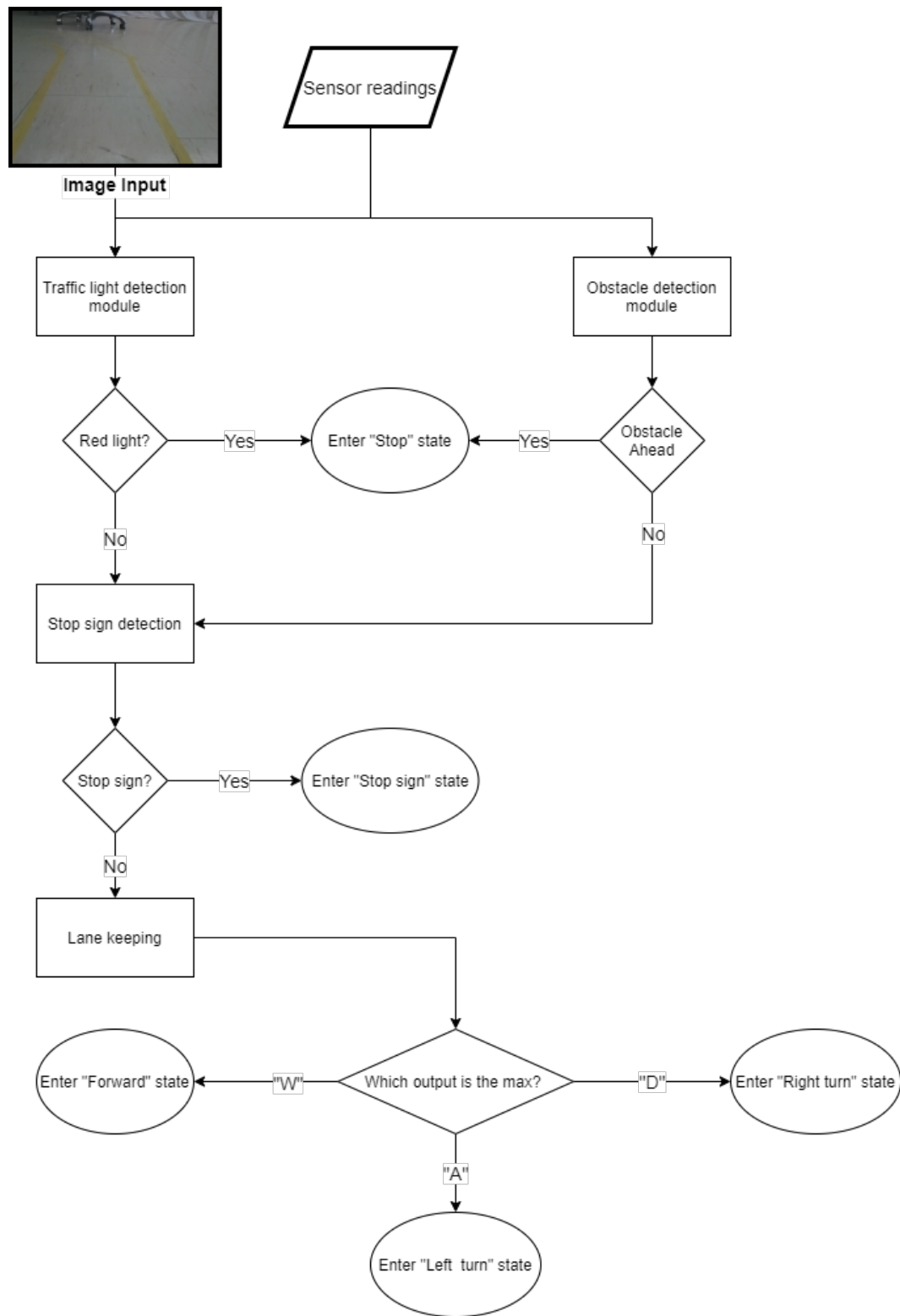


Figure 3.12: The flowchart of the logical tree for the self-driving DIY RC car

The detailed setup procedure for the self-driving simulation can be found in Appendix A.

### 3.4.1 Qualitative evaluation

We employ the logical tree (Section 3.3.3) to process information and make the final decision for the car. To demonstrate the result, we record the screen with all the modules running and added footage of the DIY RC car during the experiment (a screenshot of the video is shown in Figure 3.13). The video was edited so each individual module was demonstrated and is available on YouTube ([https://youtu.be/jP\\_zEGXf\\_p0](https://youtu.be/jP_zEGXf_p0)).

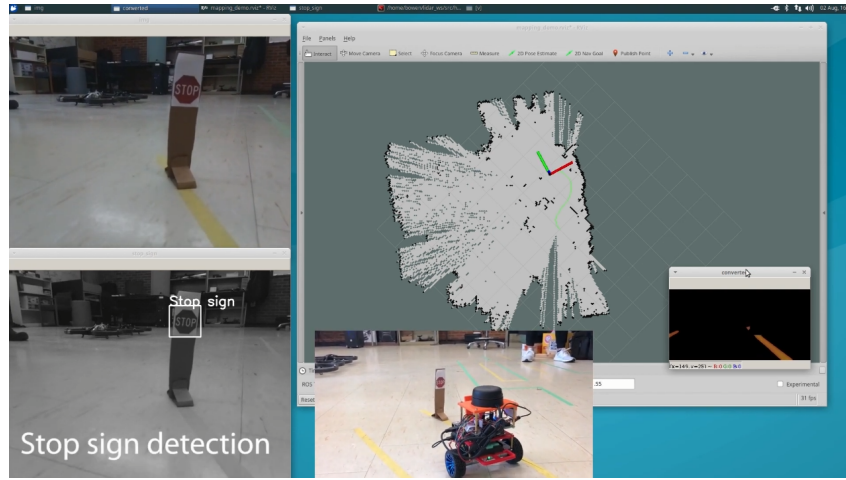


Figure 3.13: A screenshot of the demonstration video for self-driving RC car's performance

The vehicle successfully met the desired objectives:

1. Driving along the track (lane marked by yellow tape) without its wheels brushing the lane
2. Stopping the car when any obstacle is detected within a certain range, and reinstating the car when the obstacle is removed
3. Obeying traffic lights (advancing on a green light and stopping on a red light)
4. Fully stopping at the stop sign and resuming motion after 3 seconds

### 3.4.2 Quantitative evaluation

#### End-to-end learning for lane keeping

We set up an enclosed track (as seen in Figure 3.14) to test the performance of the end-to-end learning approach to lane-keeping functionality. The closed loop ensures data can be collected continuously. To measure the spatial locations of the car, we used 5 Vicon tracker balls to track the coordinates of the car during the experiment. At the same time, we also recorded all the keyboard actions (i.e. "W", "A", "D", as defined in Section 3.3.3) sent from the algorithm or by a human user.

Table 3.1: End-to-end learning datasets

| Dataset Number | Operator          | Direction               |
|----------------|-------------------|-------------------------|
| #1             | Human Operator #1 | Counter-clockwise (CCW) |
| #2             | Human Operator #2 | CCW                     |
| #3             | End-to-end model  | CCW                     |
| #4             | End-to-end model  | Clockwise (CW)          |



Figure 3.14: An overview of the track used for testing lane-keep functionality

The image dataset used for end-to-end model learning was collected from an entirely different track to avoid potential overfitting problem, but both



the training and testing datasets used identical yellow tape. Four datasets were collected (as listed in Table 3.1), each containing two types of data: 1) Trajectory of the car represented by measurements from the Vicon tracking system and 2) The driving state of the car. These two types of data are then paired and synchronized based on timestamps recorded.

Since there are over 6000 discrete locations recorded in each dataset, we subsampled the data (10% of the original data, evenly distributed). Figure 3.15 demonstrates the subsampled trajectories of the car in different datasets. Each lap is plotted in a different color, but the trajectories for lap #1 and #2 are barely visible in Dataset #3 and Dataset #4, which implies that compared to human drivers, the end-to-end learning approach achieves higher levels of consistency in both CW and CCW directions.

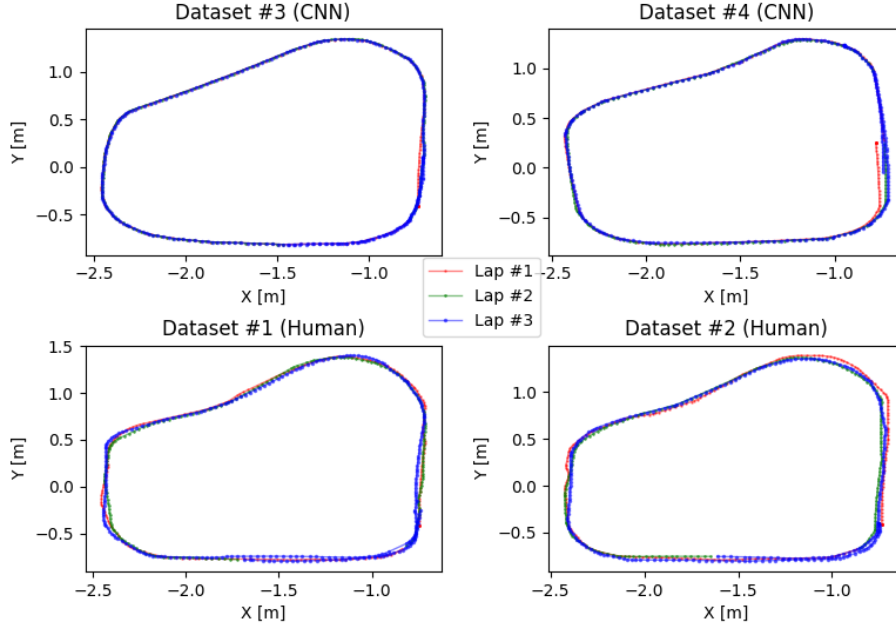


Figure 3.15: Subsampled trajectory of the car in different datasets

Another important data we collected during the experiment is keyboard actions (or driving states). As seen in Figure 3.16, the trajectory for each dataset is color-coded based on instantaneous driving state. In theory, the car only needs to go forward or steer to the right in a CW track, and go forward or

steer to the left in a CCW track. This assumption is validated in the plot for Dataset #3 and #4 in Figure 3.16. However, due to human errors (or human's reaction time), human operators tend to overcorrect the orientation of the car, causing inconsistencies in the trajectory. Overall, although both the algorithm

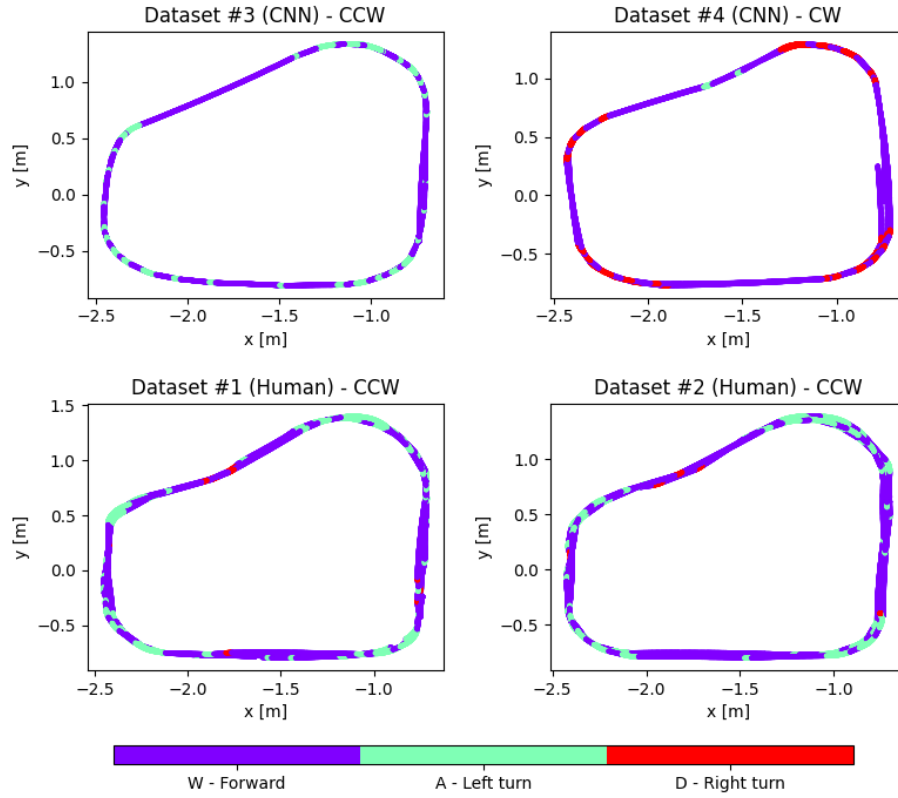


Figure 3.16: Driving states along the trajectory of the car in different datasets

and human drivers complete the track without leaving the lane markings, the end-to-end learning lane-keep approach outperforms human drivers in the following measures:

1. Highly consistent and repeatable trajectory
2. Smoother control of the car, especially along straight lanes

### Stop sign detection and distance measurements

As described in Section 3.3.3, we use the YOLO object detector and its resulting bounding box to detect traffic signs and estimate the distance from the car to the detected signs. We collected 394 images of a stop sign from the on-board camera from the RC car, and used the YOLO object detector to extract the bounding box locations. In each image, the stop sign has different orientation and/or distance to the camera. Among the 394 images, YOLO object detector failed to detect the stop sign on 7 images, thus achieving a 98.223% detection rate.

We use the method described in Section 3.3.3 to estimate the depth of a stop sign. Although in Section 3.3.3, we found  $Z_h$  is preferred to estimate the distance when the aspect ratio of the image is not 1 : 1, the heights of the bounding boxes are not consistent (as shown in Figure 3.17), especially when the stop sign is at a distance to the camera (images were captured in low resolution for real-time performance). Hence, the heights of the bounding boxes are not considered for depth estimation. In order to calculate the distance to the stop sign  $Z_w$  using Equation (3.15), the true width of the stop sign  $W$  and the focal length  $f_x$  are required. The value of the focal length is determined using camera calibration (see Section 2.2.2) and the actual width of the stop sign is measured by a ruler.

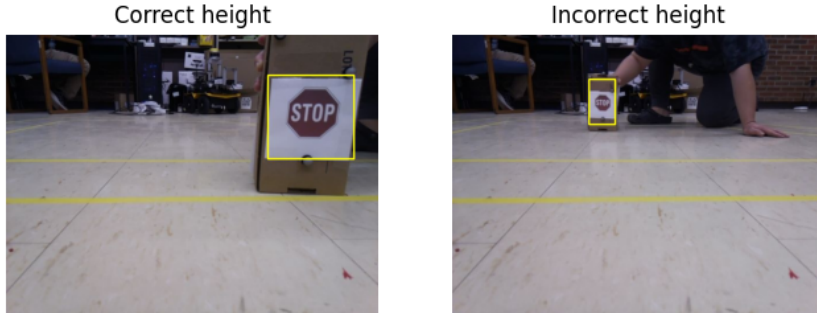


Figure 3.17: Inconsistent bounding box heights

To investigate how the estimation model is sensitive to the pixel errors, we

define the sensitivity in estimation [m/px]:

$$\begin{aligned}
\text{sensitivity} &= \left| \frac{Z_w(w'+1) - Z_w(w')}{1 \text{ pixel}} \right| \\
&= |f_x(= 511.42) * W(= 0.136) \left( \frac{1}{w'+1} - \frac{1}{w'} \right)| \quad (3.19) \\
&= 69.55 * \frac{1}{w'(w'+1)} [m/px]
\end{aligned}$$

which is inversely proportional to the width of the bounding box. In other words, when the bounding box is wider, the distance estimation function is less sensitive to the bounding box pixel errors (as illustrated in Figure 3.18). When the pixel width of the bounding box is larger than 7.80, each pixel error will result in 1-meter error in distance estimations. Hence, when the bounding box is too small or the object is too far (depending on the desired accuracy, 7.8 pixels for 1 meter accuracy), the distance estimated is less precise and should be discarded.

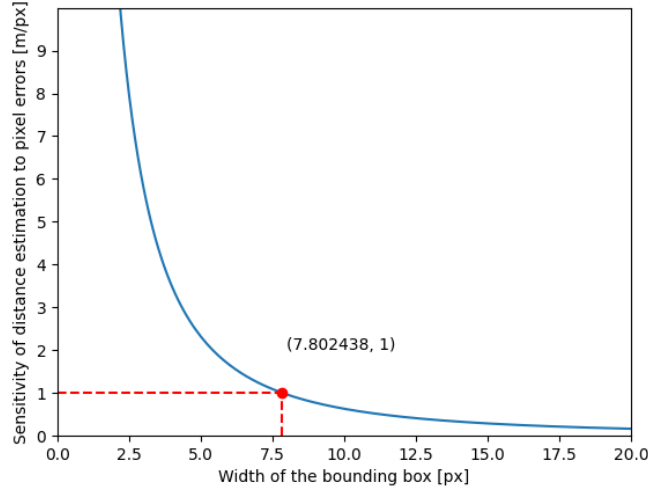


Figure 3.18: The sensitivity of distance estimation to pixel errors

To validate the performance of the distance estimation in the experiment, we used Vicon tracking system to obtain the relative pose of the stop sign to the camera as the ground truth data, and recorded the estimated distance from the bounding box extracted in each image. The stop sign was placed in front of the camera, at distances ranging from 0.6712 meters to 1.7803 meters. We then divided all the data into 5 bins, and calculated their average errors

and standard deviations based on the ground truth data. Results are listed in Table 3.2.

Table 3.2: Average errors and standard deviations in distance estimations

| Distance range [m]   | No. of samples | Avg. errors [m] | Std. deviation [m] |
|----------------------|----------------|-----------------|--------------------|
| 0.3940 $\sim$ 0.6712 | 102            | 0.04143         | 0.03038            |
| 0.6712 $\sim$ 0.9485 | 78             | 0.06933         | 0.06548            |
| 0.9485 $\sim$ 1.226  | 87             | 0.05026         | 0.04008            |
| 1.226 $\sim$ 1.503   | 88             | 0.05550         | 0.04625            |
| 1.503 $\sim$ 1.780   | 33             | 0.1031          | 0.06749            |

As shown in Table 3.2, the estimation function delivers consistent results (around 0.05 meters in error) when the object is not too far away. This conclusion is validated in Figure 3.19, the probability density function (PDF) values versus distance errors, where distance estimation errors have a mean value close to zero and exhibit a small variance from it. This result proves the

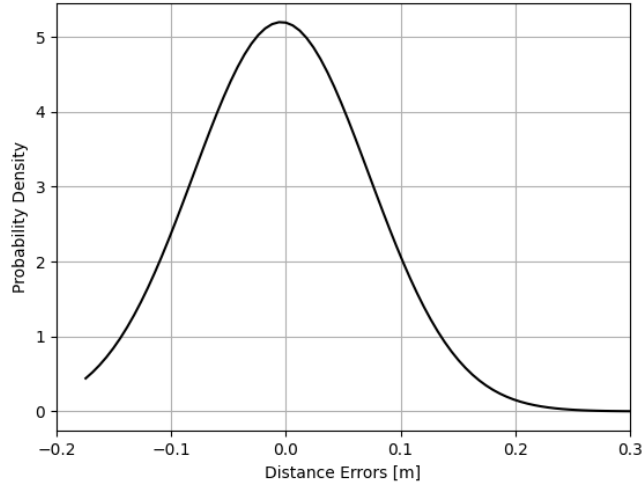


Figure 3.19: The probability density function (PDF) of the errors in distance estimations

feasibility of the proposed traffic sign detection/distance estimation system.

### 3.5 Conclusion

In this chapter we developed a self-driving system on a 3D-printed DIY car, which includes:

- End-to-end learning lane-keeping function
- Traffic light/sign detection
- Bounding box-based traffic sign distance estimation
- Obstacle avoidance using the ultrasonic sensor

A logical tree handles all the outputs from individual modules and makes the final decision on the car's behaviour. Upon testing, the proposed system completes the desired track, while complying with all the traffic lights/signs. Additionally, the system outperforms human operators in lane keeping experiments, delivering a smooth and consistent trajectory. With a 98.233% detection rate, the system detects the stop sign in almost all cases, and the distance estimations have an average error of approximately 0.05 meters in the distance range of 0.3940 to 1.503 meters.

More importantly, the process of developing the self-driving system and the integration of different modules revealed the beauty of computer vision to the writer, and familiarized him with techniques of programming and computer vision. Therefore, the self-driving RC car project is the foundation of this thesis, and it led to the work in the upcoming two chapters.

# Chapter 4

## Predictive Display

### 4.1 Introduction

Tele-operation tasks often face high latency in the communication link. While a lot of progress has been made in optimizing communication over traditional networks [1] or new technologies such as 5G [66], time delays remain a limiting factor in applications such as operating unmanned rovers on Mars [56] and underwater manipulator systems [71]. Predictive display (PD) [7] [35] was introduced to address this issue by rendering a visualization of the scene model to the human operator based on their commanded inputs. This provides direct and instant visual feedback for the operator controlling the remote robot in the face of communication delays.

Existing PD methods use offline [16], [85] or online-based 3D reconstruction [32], [36], [60] to render the scene. Offline methods use known image priors of the remote environment to build their model (e.g. [35] uses structure-from-motion to build a texture model), and do not update it during operation. Therefore, this approach is limited to static environments. Online-based methods update their model using data transmitted from the remote site to the local operating centre. This approach is more flexible and can work in dynamic environments.

Our aim is to develop an online PD system which is more robust to low-textured scenes, while providing photo-realistic images corresponding to the operator’s inputs. To do this, we integrate a generative model with an on-line 3D reconstruction predictive display pipeline based on RGB-D images

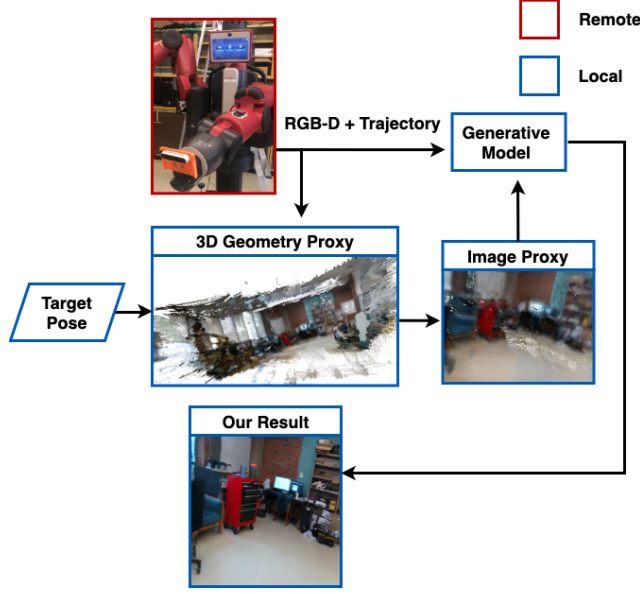


Figure 4.1: An overview of the core blocks in our system. The remote robot moves through an environment while collecting RGB-D images. This data is sent to the operator’s workstation to build a coarse 3D map, whose corresponding 2D images are generated by a camera projection. These images are then refined by a generative model which outputs photo-realistic images of the remote environment to the operator.

captured by the remote robot.

The key contributions of our proposed method are:

- A novel approach to predictive display which uses a coarse 3D map with a generative model to output photo-realistic images of the scene.
- The ability to fill in missing features and upgrade texture details of images rendered from the 3D map.
- Continuous updating of the 3D map of the remote environment to deal with dynamic effects such as object relocation and new objects entering the scene.
- Real-time operation enabling immediate visual feedback to the human operator in the face of communication delays.



## 4.2 Related works

### 4.2.1 Predictive display

Various approaches have been proposed to construct an image-based predictive display system in both offline and online variants. Cobzas and Jägersand [16], [35] proposed an offline predictive display method, which generates a geometric and dynamic texture model using structure-from-motion (SfM). The method integrates the geometric model obtained from SfM with an appearance-based dynamic texture. The remote robot extracts the current pose using a registration-based SSD tracker. The calculated pose is then transmitted back to the operator, and is composed with the operator’s commanded motion to render a visualization. However, this method is reliant on the geometric model provided by SfM, which may fail to achieve feature extraction and matching in low-textured scenes, leading to an incorrect or incomplete geometry model.

Hu et al. [32] proposed an online predictive display based on PTAM [38] and incremental free space volume carving (CARV) [49]. The remote robot estimates its pose and extracts sparse key points using PTAM, then incrementally reconstructs the scene mesh using CARV. Similarly to the issue in [16], PTAM may fail to detect feature points in low-textured areas (for instance the sky), resulting in missing portions of the geometry or background. Hu et al. proposed to address this issue by back-projecting the captured image at the mean depth of the current scene to the proxy image as the synthetic background. However, the results may not be satisfactory, since real images at desired depths are not always available, and the distorted background combined with the projected CARV model is not photo-realistic.

### 4.2.2 3D reconstruction

Schönberger et al. proposed COLMAP [68], [69], a SfM and Multi-View Stereo (MVS) toolchain. COLMAP first extracts image features and then matches them to obtain camera poses. SfM is then used to construct a sparse scene model, followed by MVS to generate a dense model with more details.

Another approach to 3D reconstruction is based on visual SLAM, a real-time method which incrementally upgrades the scene geometric model. For example, ORB-SLAM [54] extracts and tracks ORB features to calculate the camera odometry, and produces sparse point-cloud as the scene model.

Many 3D reconstruction works are based on the Signed Distance Function (SDF). Curless et al. [18] use a cumulative weighted Truncated Signed Distance Function (TSDF) to represent the 3D map and iteratively update the volume with new depth images. Newcombe et al. [55] proposed Kinect Fusion, which stitches successive RGB-D images to create a preliminary map, and continuously updates the global scene model from the estimated camera poses. Kinect Fusion provides smooth surfaces, but its performance is limited when the scene is out of the depth sensor’s working range (i.e. too near or too far). This behaviour often leads to missing and noisy geometries in the reconstructed scenes, especially in an open space. Newer works include Intrinsic3D [50], which utilises Shape-from-Shading (SfS) and spatially-varying spherical harmonics (SVSH) to upgrade the SDF volumes, achieving higher levels of detail.

### 4.2.3 Deep learning based images synthesis

Generative models have shown great potential for image manipulations. While most of the applications focus on 2D image generation and refinement, only a few works have applied generative models to 3D geometric model. Yu and Wang [86] proposed the 3D-Scene-GAN method, which inputs a 3D model and renders synthetic images from it. The generative model iteratively updates the 3D model until the discriminator can no longer distinguish between synthetic images rendered from the (updated) model and the corresponding real images. Although this method works well for fixing defects such as holes and distortions in the input 3D model, since it does not preserve the background (for instance the sky) in the scene model, the rendered images are not photo-realistic. Thus, to build a predictive display, instead of updating the 3D model itself, we use a generative model to refine the images projected from the coarse 3D model.

Other learning-based approaches do not rely on convolutional networks.

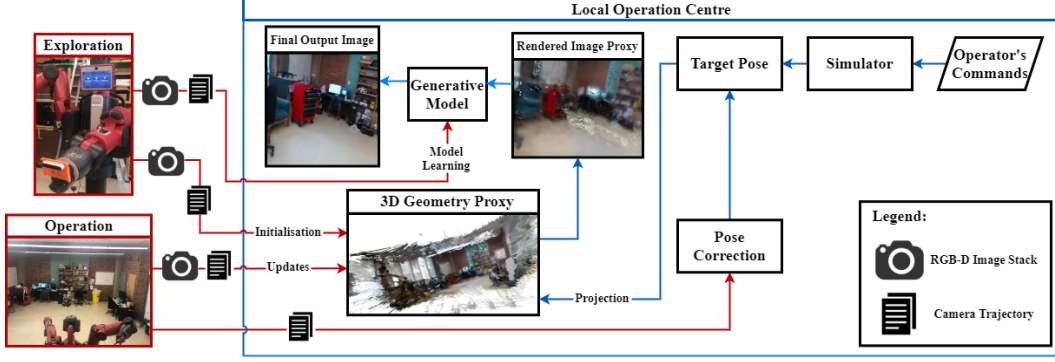


Figure 4.2: Flowchart of the proposed system. After the self-exploration stage is completed, a set of RGB-D images is transmitted to the operator’s workstation and used for reconstructing the 3D scene model. This stage is executed only once, and the remaining steps are run iteratively. A 2D image is projected from the 3D model based on the commanded robot pose. This image is refined by the generative model, and the result is displayed on the operator’s interface. Newly acquired RGB-D images are transmitted and used to dynamically update the 3D geometry model and estimated pose of the remote robot.

NeRF [52], for example, uses the camera’s pose and field of view (FOV) as the network’s inputs, and outputs the volume density and radiance at these locations. A synthetic image is then generated using this information. NeRF can produce photo-realistic images, but it was found to perform poorly in reconstructing large scenes, which are quite common in PD scenarios. Also, the inference time usually takes a few seconds, which makes it unsuitable for real-time predictive display.

## 4.3 Methodology

### 4.3.1 Overview of the system

The proposed system consists of a remote robot and the local operator’s workstation. The remote robot first explores the environment and transmits a set of key frames  $\{I_{rgb}\}$  (RGB images),  $\{I_d\}$  (depth images) and corresponding camera poses  $\{T_{key}\}$  to the operation centre. A coarse 3D geometric model  $\mathcal{M}_0$  is reconstructed using  $\{I_{rgb}\}$ ,  $\{I_d\}$  and  $\{T_{key}\}$ . We render an image set  $\{I_r\}$  at camera keyframe poses  $\{T_{key}\}$  and train the generative model  $G$ . Once

the generative model is trained, given a desired pose  $T$  commanded by the operator, a coarse 2D image  $I_r$  is rendered, and the generative model  $G$  is used to refine the image. The refined image  $I'_r$  is displayed on the operator's interface. The operator thus gets an instantaneous visualization of the scene at the commanded pose and can continue to send inputs to the remote robot. To deal with changes in the environment, each new RGB-D image transmitted from the robot is used to update the coarse 3D scene model  $\mathcal{M}_0$ .

### 4.3.2 Preprocessing

**Data collection:** The robot first explores its working environment, and transmits RGB images  $\{I_{rgb}\}$  and corresponding Depth images  $\{I_d\}$  and camera poses  $\{T_{key}\}$  relative to the initial pose. Camera poses can be estimated from two sources:

1. Computed from a kinematics model of the robot driven by sensors such as joint encoders (arm manipulators), encoders (wheeled robots) or visual odometry plus IMU data (aerial vehicles).
2. Obtained from a SLAM system onboard the robot.

**Coarse geometry reconstruction:** Any 3D reconstruction method can be employed with our method, provided it is capable of incrementally updating the model when new RGB-D images arrive, and can run in real-time. In our case, we use TSDF volumetric integration [18] due to its ability to fill in missing data and fast performance relative to newer methods such as Intrinsic3D [50]. TSDF consists of two steps: volumetric integration followed by hole filling. The first step iteratively integrates triangulated depth images onto the existing mesh, using weighted signed distance functions on sample points:

$$D_{i+1}(x) = \frac{W_i(x)D_i(x) + w_{i+1}(x)d_{i+1}(x)}{W_i(x) + w_{i+1}(x)} \quad (4.1)$$

$$W_{i+1}(x) = W_i(x) + w_{i+1}(x) \quad (4.2)$$

where  $D_i(x)$  and  $D_{i+1}(x)$  are summed signed distance functions,  $W_i(x)$  and  $W_{i+1}(x)$  are summed weights in the  $i^{\text{th}}$  and  $(i+1)^{\text{th}}$  depth image, respectively,

$d_{i+1}(x)$  are the signed distance functions in the  $i+1^{\text{th}}$  depth image, and  $w_{i+1}(x)$  are weights in the  $i+1^{\text{th}}$  depth image. The resulting mesh typically contains many holes. Thus the second step extracts the isosurfaces close to missing data regions in the mesh, and holes are connected using the process described in [18]. The reconstructed coarse 3D model is denoted as  $\mathcal{M}_0$ .

### 4.3.3 Generative model learning

Camera projection is used for both training data preparation and the teleoperation interface. Once the coarse geometric model  $\mathcal{M}_0$  is built, we apply OpenGL’s perspective projection [70] at recorded key frame camera poses  $\{T_{key}\}$ , which projects the points within the viewing frustum of the camera to a resolution matching the RGB images from the remote robot  $\{I_{rgb}\}$ . Another tuning parameter is the field of view (FOV), which affects the projected images. To replicate the real camera used to capture  $\{I_{rgb}\}$ , the FOV is set to match the robot camera’s view frustum, including right, left, top, bottom limits  $(r, l, t, b)$  and user-specified  $(n, f)$  which determine the closest and furthest range of visible space. The perspective projection matrix  $T_{\text{perspective}}$  is built with these six parameters. Multiplying this matrix by the homogeneous coordinates  $(x, y, z, w = 1)$  of points in the 3D model produces the clipped coordinates  $(x', y', z', w')$  of the point. Dividing the transformed coordinates by  $w'$  produces Normalized Device Coordinates (NDC) with values between -1.0 and 1.0. Now the projected coordinates can be cast onto the image plane with a desired image resolution by means of linear interpolation.

**Training data augmentation:** The generative model used in our system is CycleGAN [88], which introduces both conventional adversarial losses [24]  $\mathcal{L}_{GAN}$  and cycle consistency losses  $\mathcal{L}_{cyc}$  to improve the forward and backward consistency. Another advantage of CycleGAN is that it does not require dataset alignments (i.e. the number of images within the two image domains  $X$  and  $Y$  can be different and images can be unpaired). Therefore, in addition to the coarse images  $\{I_{proj}\}$  projected from  $\mathcal{M}_0$  at key frame camera poses  $\{T_{key}\}$ , we also apply random transforms to  $\{T_{key}\}$  and generate additional images  $\{I_{aug}\}$  to augment the training image set. The resulting set of images

is denoted as  $\{I_{proj+aug}\}$ .

Once two training image sets  $X$  and  $Y$  are set up, CycleGAN optimizes two mapping functions  $G : X \rightarrow Y$  and  $F : Y \rightarrow X$  and two discriminators  $D_X$  and  $D_Y$ . We thus train the generative model to map from generated images  $X = \{I_{proj+aug}\}$  to real images  $Y = \{I_{rgb}\}$ . The overall objective is to solve

$$(G^*, F^*) = \arg \min_{G, F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y) \quad (4.3)$$

After the generative model is trained, the generator function  $G^*$  is able to output a refined image  $I'$  given any image  $I_r$  rendered from the coarse model  $\mathcal{M}_0$ :

$$G^* : I_r \rightarrow I' \quad (4.4)$$

#### 4.3.4 Online model updating

Since there can be changes in the environment during the course of remote operation (e.g. existing objects moved or new objects added), it is also important to transmit newly acquired RGB-D images to the local operator's station whenever possible in order to update the 3D model online. When new RGB-D images are received, they are integrated into the existing 3D model by leveraging the weight values in Equations (4.1) and (4.2). Newer observations have more weight, so incoming RGB-D images update the existing 3D model.

#### 4.3.5 Robot pose correction

In the PD interface, the operator drives the remote robot through physical inputs. These commands are sent in parallel to the remote robot and a local simulator. This simulator, for instance Gazebo [39] or iGibson [83], simulates the robot dynamics and calculates its pose  $T_{sim}$ .

Meanwhile, the remote robot executes the sent inputs and moves to  $T_{real}$ . However, if the remote robot encounters disturbances such as new obstacles or unmodeled dynamics, the commanded motions are not fully executed. In this case, errors in the simulated pose of the remote robot will accumulate. To address this issue, we let the remote robot periodically report its current pose

$T_{real}$  with an associated timestamp. Therefore, whenever this pose is received, it is propagated through the simulator using a history of inputs to correct  $T_{sim}$ .

## 4.4 Experimental results

### 4.4.1 Experiment setup

**Remote robot:** We use an Intel RealSense D415 as the RGB-D sensor and attach it to the end-effector mounting plate of the well-known Baxter robot from Rethink Robotics (Fig. 4.3).

**Local operator’s workstation:** The local operator’s workstation is equipped with an Intel i7-8700K CPU and an NVidia GTX 1080 Ti GPU. We use this setup to evaluate the real-time capabilities of the system.

**Dataset capture:** To evaluate the performance of our method, we captured 8 datasets and tested them with TSDF and NeRF. Camera poses are calculated using the robot’s forward kinematics for each dataset. Four datasets focus on small objects at distances between 0.6 meters and 2.1 meters (Near #1-4), while the other four datasets are captured at distances between 3 meters and 6.5 meters to represent a larger scene (Far #1-4). The dataset details are listed in Table 4.1.

Table 4.1: Dataset details. Two generative models were trained for “Near” and “Far” sets respectively. 2916 and 1854 images were used to train the “Far” and “Near” generative models, respectively. Each dataset was captured independently and thus has a different camera trajectory.

| Dataset Notation | Description                  |
|------------------|------------------------------|
| Far #1           | Trajectory used for training |
| Far #2           | Static scene                 |
| Far #3           | Relocated objects            |
| Far #4           | Novel(unseen) objects        |
| Near #1          | Trajectory used for training |
| Near #2          | Static scene                 |
| Near #3          | Relocated objects            |
| Near #4          | 360-degree trajectory        |



Figure 4.3: Experimental setup. The end-effector mounting plate holds a 3D-printed camera holder. An Intel RealSense D415 is mounted on the camera holder. The pose of the camera relative to the world frame is calculated using forward kinematics.

#### 4.4.2 Qualitative evaluation

**Texture upgrades:** To evaluate the quality and realism of processed images, both qualitative and quantitative evaluations were used. For comparison, we provide visual results produced by the TSDF volumetric integration method, the backbone of many online 3D reconstruction methods, as well as COLMAP, an offline method. As seen in Figure 4.4, the improvement obtained by using the generative model in our system is significant, especially in cases where the scene is static. TSDF reconstructs scenes poorly, mainly due to the hardware limitations of the RGB-D camera (low resolution and noisy readings), resulting in missing sections in the rendered images. Our method fills in these missing sections with photo-realistic textures, with only small misalignment created by errors in the generative model and 3D reconstruction.

When the scene changes, or new objects enter the scene, the performance of our method drops. This behaviour is expected since the generative model lacks knowledge about new textures and geometries. Although the level of detail decreases, the new objects do not vanish from the output images. Dataset Far









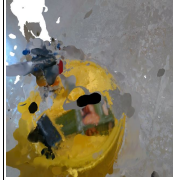
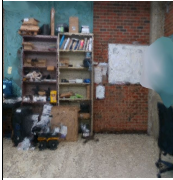

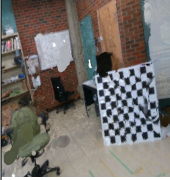


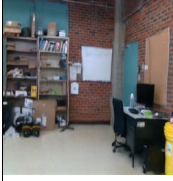
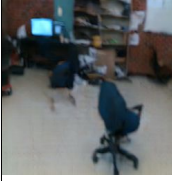






| Dataset No.  | Near #2   | Near #3   | Far #2  | Far #3   | Far #4  |
|--------------|---|---|---|--|---|
| TSDF         |    |    |    |    |    |
| COLMAP       |    |    |    |    |    |
| Ours         |   |   |   |   |   |
| Ground Truth |  |  |  |  |  |

Figure 4.4: Comparison of results from 3D reconstructed proxies processed by different methods: TSDF, COLMAP and our system, together with ground-truth images from the dataset.

#4 (shown in Figure 4.4) is a good example to illustrate the behaviour. Both the chessboard and the green chair are new objects, but they are preserved by the generative model. In general, our method yields best results when the scene is static, providing high-quality and photo-realistic images, but when new objects are present in the scene, the image displayed to the operator still renders the new object, albeit with worse visual quality than previously-seen objects.

**Baseline methods comparison:** As an offline method, COLMAP fails to update the 3D model from new RGB-D images. By default, COLMAP uses feature extraction and feature matching techniques to calculate camera poses. For fairness of comparison, we forced it to use our camera poses estimated from the robot dynamics model, same as in our method. The bundle adjustment and other optimizations used by COLMAP remained unaffected. COLMAP has limitations in reconstructing surfaces in texture-less areas, and some geometries have texturing errors. COLMAP is more stable in large scene reconstruction (as seen in the Far datasets in Figure 4.4) compared to close-range object reconstruction. In texture-rich area (e.g. the chessboard in Far #4), COLMAP provides better texture details than our method. Overall, our method provides more consistent and complete scene images than COLMAP.

We also compare our method with NeRF. As NeRF works best with inward-facing scenes, we use Near #4 to compare the performance of our method versus NeRF. Since NeRF does not export its mesh, we manually select two images which have similar camera perspectives, as shown in Figure 4.5. Although NeRF is seen to provide good-quality images, it is not suitable for PD applications for the following reasons:

1. Inference time is too long. The original NeRF implemented on TensorFlow architecture takes 2 to 3 seconds to process one image query on a GTX 1080Ti GPU, compared to an inference time of around 0.2 seconds in our method.
2. NeRF is not designed and optimized for large scenes, which are common in PD applications. On our Far datasets, training failed to complete.

3. NeRF does not work in dynamic environments. Unlike the generative model used in our method, NeRF does not update its model during runtime.

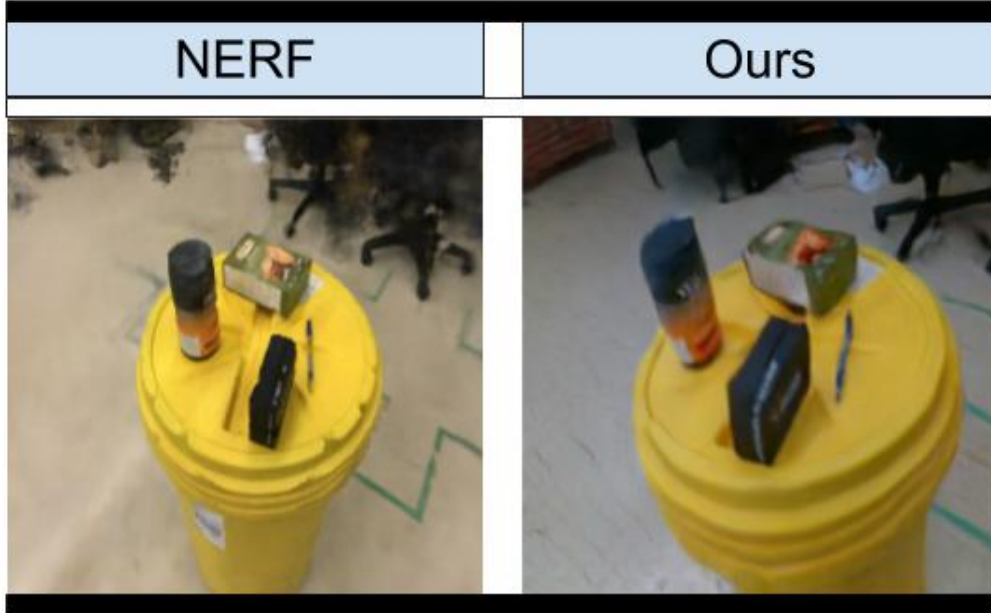


Figure 4.5: Qualitative comparison between NeRF and our method. NeRF has slightly more textural details about the objects at the center, but corners around the image are noisy and lack details (e.g. the bottom of the chairs). Also, some objects have geometry reconstruction errors (the edge of cap of the barrel, the black package etc.)

#### 4.4.3 Quantitative evaluation

Several quantitative image similarity and quality evaluation metrics were used to compare between ground truth and synthetic images: 1. Mean-squared error (MSE); 2. Peak signal-to-noise ratio (PSNR); 3. Structural Similarity Index (SSIM) [78]; 4. Dubbed blind/referenceless image spatial quality evaluator (BRISQUE) [53]; 5. Learned Perceptual Image Patch Similarity (LPIPS) [87]. BRISQUE is unique in that it does not rely on a ground truth image.

The MSE metric computes the mean squared pixel-wise differences between synthetic images and ground truth images, without considering structural in-

Table 4.2: Mean values of evaluation scores. We take average values of all datasets (excluding two training sets Far #1 and Near #1).  $\uparrow$  means higher values represent better performance, and  $\downarrow$  indicates smaller values represent better performance.

| Scores       | MSE $\downarrow$ | PSNR $\uparrow$ | SSIM $\uparrow$ | BRISQUE $\downarrow$ | LPIPS $\downarrow$ |
|--------------|------------------|-----------------|-----------------|----------------------|--------------------|
| TSDF         | 1486.3           | 17.253          | 0.4852          | 28.008               | 0.5951             |
| COLMAP       | 4608.0           | 12.425          | 0.4290          | 31.279               | 0.6633             |
| Ours         | <b>1405.4</b>    | <b>17.631</b>   | <b>0.5168</b>   | <b>25.127</b>        | <b>0.3486</b>      |
| Ground Truth | N/A              | N/A             | N/A             | 18.111               | N/A                |

formation. An image pair with a smaller MSE value has better pixel-wise similarity. The smallest mean MSE values are achieved by our system. The PSNR metric measures the ratio of pixel intensities between images, meaning a larger mean value represents better similarity. SSIM is a metric considering local features including structure, luminance, and contrast in images. BRISQUE, as a non-referenced method, uses locally normalized luminance coefficients to evaluate the naturalness of the image. LPIPS uses a deep neural network to compare perceptual similarity, and is reported to better align with a human’s choices when it comes to similarity as compared to PSNR and SSIM [87].

**Overall performance:** To evaluate the overall performance of our proposed method versus TSDF and COLMAP, we evaluate all images in the datasets (excluding Far #1 and Near #1 which are used for training) against 5 metrics, and take the average of all the scores, as listed in Table 4.2. Our proposed method achieves the best scores in every metric. The improvements in image quality metrics are significant, especially in LPIPS where the score is almost doubled, which indicates our proposed method delivers more natural and realistic images than the TSDF method.

**Performance in different scenarios:** As demonstrated in Figure 4.4, the image quality decreases when objects are relocated or previously unseen objects are present in the scene. In Figure 4.6 we illustrate the LPIPS evaluation scores on different datasets. Our method outperforms TSDF in all scenarios in terms of similarity scores; however, image quality is still negatively affected by both relocated and new objects in the scene.

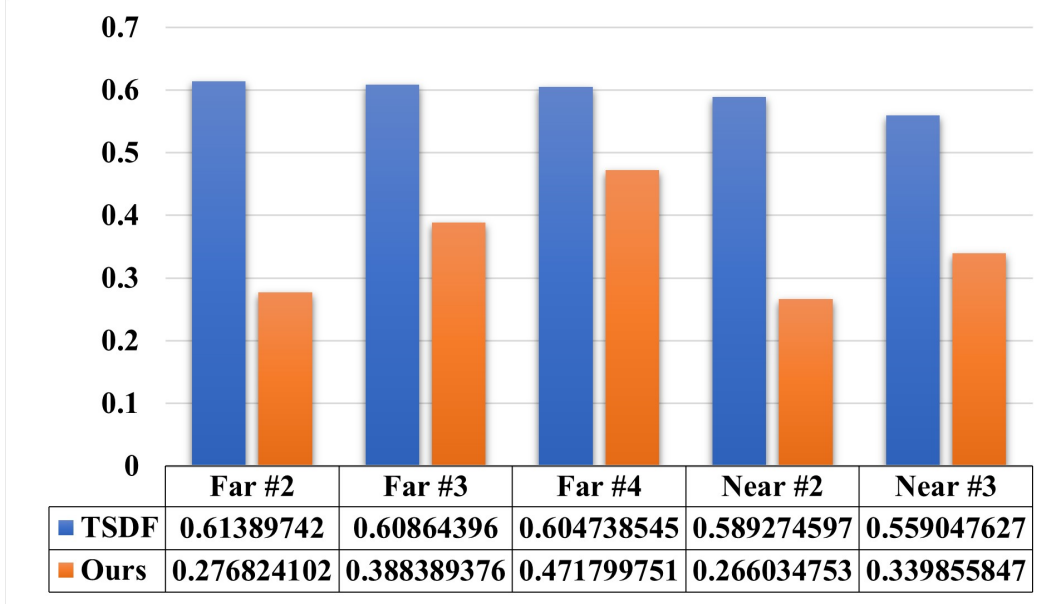


Figure 4.6: LPIPS scores on different datasets. Lower values mean higher perceptual similarity to the ground-truth images.

## 4.5 Conclusion

We presented a novel predictive display pipeline for teleoperation over high latency communication links, which uses a locally stored 3D map of the environment to generate 2D images from the predicted pose of the remote robot, thus providing delay-free visual feedback to the human operator. Our system reconstructs a coarse 3D scene model and uses it as prior knowledge to train a generative model to generate photo-realistic synthetic images for the operator. Experimental results demonstrate that the proposed method has the ability to fill in missing geometries and is able to reconstruct and refine relocated and novel objects, which are common for tele-operation in dynamic environments. Our method outperforms other approaches to online reconstruction in both qualitative and quantitative testing.

# Chapter 5

## External Vision-based Joint Detection System

### 5.1 Introduction

As autonomous robotics arm manipulators are becoming more popular and widely used, such as assembly operations in automotive manufacturing [57], food preparation [13] and logistics management (e.g. Amazon picking challenge [29]). One of the key components in precise robotics tasks like these is motion planning, which includes forward and inverse kinematics. However, forward and inverse kinematics rely on robot geometry and joint position measurements. Although many customer-grade robot arms, such as Barret WAM Arm or Kinova JACO, can provide precise joint measurements from onboard transducers, inexpensive robot arms or human-operated hydraulic system fail to measure the joint states due to:

1. Poor encoders, which lacks precise transducer readings for joint states
2. No joint feedback, which is fairly common in human-operated manipulators (e.g. knuckle-boom cranes)

In order to address these issues, one direct and obvious solution is to add precise encoders (or any joint feedback sensors) to these systems. However, this solution requires throughout customization, which results in high cost, low versatility and maintainability. An alternative solution is to use computer vision to detect the joint angles and end-effector poses, as demonstrated in

[42], [43], [89]. This solution is more appealing thanks to high versatility and easier deployment.

As discussed in Section 2.3, there are many types of imaging sensors, including conventional RGB cameras, RGB-D cameras, infrared cameras (e.g. Vicon motion tracking system). Among all the imaging sensors, RGB cameras is the least costly one, yet providing the most versatility, which makes RGB cameras the most viable choice. Therefore, in this chapter, we focus on joint detection using a monocular RGB camera. 3D coordinates of the joints can be obtained, thereafter, via triangulation. As our problem is technically related to human skeleton tracking, we seek inspiration and adaptation from human skeleton tracking publications. Extensive works on robot manipulator (e.g. [58]) and/or human skeleton tracking (e.g. [2]) have been done using RGB-D cameras. Recently, thanks to the aid of deep neural networks, these problems now can be solved using only monocular RGB cameras (e.g. [4], [75]). One of the key components of deep neural network methods is a large set of data used for training. In this matter, human skeleton tracking is easier because human joint datasets are publicly accessible (e.g. Human3.6M [34]). However, this is hard to apply to robot manipulator joint tracking, because each robot has drastically different appearances. Therefore, in order to train a joint detector, thousands of images are required to train the model, which is time-consuming and each robot model requires a new dataset.

Apart from manually capture a large set of images and label the joint locations individually, a robot simulator (e.g. Gazebo [39] and iGibson [83]) can generate an unlimited number of synthetic images of the robot manipulator along with the corresponding joint labels given any camera perspectives, illumination conditions and background images. However, since the images generated by the simulator remain distinguishable from the real-world scenes, the learning-based joint detector performs badly using these synthetic images. Luckily, thanks to the development of generative neural network (GAN) [25], which enables image domain transfer, can refine the synthetic images to improve the realism of the images.

We design a pipeline which detects the multiple joint positions on the

robot manipulator simultaneously in a real-world environment with monocular RGB cameras. We set up the experiment on the Baxter robot (see Section 2.1, and compare it with the high-precision Vicon motion tracking system (see Section 2.3.3) and on-board joint measurements. This work has three contributions:

- Implementing a combination of image segmentation and domain transfer methods which was found to provide the best-performing processing pipeline
- Extensive validation of the experimental performance in a variety of experimental settings including background types and illumination conditions, and assessing the resulting performance qualitatively against the ground truth
- Demonstrating that our system can perform as well or better than the recent state-of-the-art work [43] which includes a trained network for the Baxter robot

## 5.2 Related works

### 5.2.1 Robot joint angle measurements

Conventional methods use encoders to obtain the joint measurements (e.g. rotary encoders for rotating joints and linear encoders for linear actuators). Robot joint poses or end-effector poses can be determined by forward and inverse kinematics. For those robots which do not have proper joint feedback sensors, forward/inverse kinematics fail to work. DART (Dense Articulated Real-Time Tracking) [67] uses known 3D CAD model and depth images to estimate the pose for each link or joint on the robot arm. The utilizes the optimized signed distance function (SDF) to align the CAD model with the perceived depth images until the difference is minimized. The system is very computationally efficient and is capable of processing in real-time. However, as it relies on depth sensors, which have a very limited working range (normally up to 10 meters), DART fails to produce robust measurements when the object



exceeds the working range of the depth sensors. Another direction, however, utilizes domain randomization (DR) to extract joints/feature points from RGB images. Zuo et al. proposes CRAVES [89], which extracts 17 feature points from a low-cost robot arm and uses a regression network to calculate the joint angles of the robot arm based on the extracted feature points that appeared in the images. DREAM [43], on the other hand, focuses on 10 rotary joints on the Baxter robot and tries to extract these joints from 2D images. DR plays a significant role in both works, as it improves the stability and the robustness of the detector under different environments. When illumination conditions or camera pose changes, the texture/perspective/color of the object change as well, but DR tries to create as many variations as possible during the training, so the detection network has more generality.

### **5.2.2 Marker-based robot joint tracking**

Apart from the Vicon motion tracking system as discussed in Section 2.3.3, which utilizes infrared sensors to track markers and only viable in an indoor environment, there are other marker-based tracking methods which work well with only RGB imaging sensors. For instance, fiducial markers [23], which are encoded markers with special patterns, can be used to mark individual joints. These markers are encoded to contain number, and provide pose information. He et al. [26] use paper-printed markers to estimate the state of the object (including velocities and poses). However, fiducial markers have many limitations and hence become less robust due to:

1. Failed to keep tracking the marker when the joints move to certain angles where cameras are unable to capture the markers
2. Bad illumination conditions may interfere with the tracking algorithm, resulting in tracking lost

### **5.2.3 Articulated human body pose estimation**

Human body pose estimation is very similar to robot joint pose estimation from a technical perspective. Extensive researches have been studied on hu-

man pose estimation based on the human joint annotations datasets, which are either manually labelled [34], or generated by Vicon motion tracking system. As human bodies have fewer variations than robots, and thanks to the widely-available datasets, there are many methods targeting at human posed estimations are published. Yang et al. propose a method [84] which utilizes a neural network to locate each segment of the robot and mark them in bounding boxes. Later works use keypoint detection (e.g. [11], [12]), providing higher resolution than the bounding boxes. These methods output a stack of heatmaps, where each heatmap represents the probability distribution of each keypoint. Therefore, the peak value in each 2D heatmap corresponds to a keypoint.

## 5.3 Methodology

### 5.3.1 System overview

The development of the proposed pipeline is a product of many trials and errors. At first, we built the pipeline which directly outputs the numerical values of the joint angles from perceived 2D images, but the results were not robust. Also, this approach brings uncertainties to the system, because the input of the system is the entire 2D image. Therefore, the output joint angles drastically fluctuate while the object only moves slightly. This type of neural network is hard to converge. Thus we focused on keypoint detectors instead, including:

1. A CNN directly takes original RGB images as inputs and outputs multiple heatmaps for joints
2. Utilizing a generative model to convert synthetic images generated by the simulator to train the detection network

However, both approaches did not produce satisfying results. Hence, we proposed this final pipeline which demonstrates the best results.

As shown in Figure 5.1, the system takes each RGB image  $I_1$  from the camera, and processes it with the following steps:

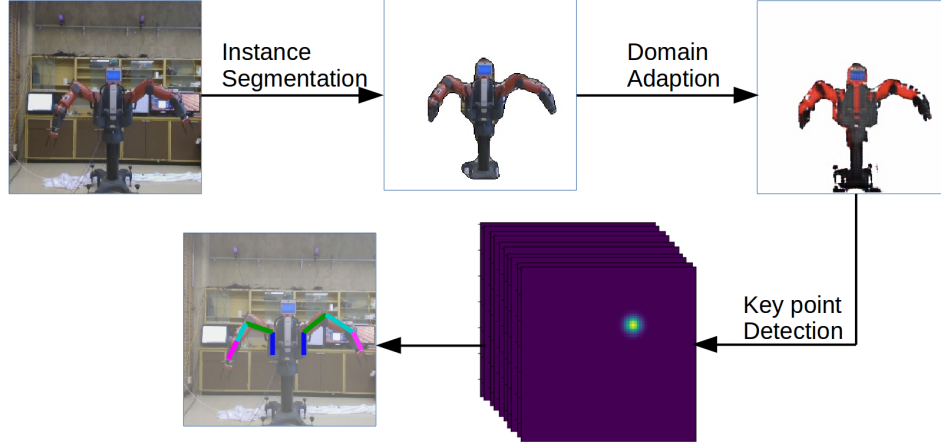


Figure 5.1: Workflow overview of the proposed system

1. Instance segmentation: Masking the robot from the background, which produces an image  $I_2$  with a clean background
2. Domain adaptation: Using a generative neural network (GAN) to transfer real images to simulation-style of images  $I_3$
3. Joint detection: Feeding the simulation-style images  $I_3$  into a joint detector network, which outputs the 2D pixel coordinates of each joint of the Baxter robot arms with an associated label for each joint
4. (Optional) Triangulation: With multiple cameras deployed, and each joint being observed from multiple perspectives (equal or greater than 2), triangulation upgrades the 2D pixel locations of each joint to 3D spatial location

The following sections describe details for each procedure and how the neural networks are trained.

### 5.3.2 Instance segmentation

Instance segmentation is the first step to apply to the RGB image. When the image arrives, instance segmentation extracts the Baxter robot from the background. This step, upon testing, drastically improves the stability and robustness of the detection neural network.

Many works have been established for instance segmentation (e.g. Instant-Cut [37], DIN [5], SGN [46] and Mask R-CNN [27]). Mask R-CNN is our choice for instance segmentation as it provides the most accurate results among the four [27]. Mask R-CNN utilizes Faster R-CNN [64] to classify the objects of interest and crop them with bounding boxes. Then it uses FCN [48] to perform pixel-wise segmentation masking on the focused region. A masked object with clean background ensures domain adaptation to work properly.

### 5.3.3 Domain adaptation

CycleGAN [88] is used to perform domain adaptation, which transfers real images to simulation-style images. Because we train our joint detector based on simulation-style images, domain adaptation modifies the images so the joint detector works. Although there are many GAN available, two core strengths of CycleGAN make it suitable for our pipeline:

- Unparalleled datasets for training, which does not require equal sizes of image domains
- Introducing both conventional adversarial losses  $\mathcal{L}_{GAN}$  and cycle consistency losses  $\mathcal{L}_{cyc}$  to improve the forward and backward consistency

Information about CycleGAN has been covered in Section 4.3.3.

### 5.3.4 Joint detection

Once the image is transferred to simulation-style  $I_3$ , the joint detector takes  $I_3$  as input and detects 10 joints on the Baxter robot arms (as shown in Figure 5.1). Inspired by human skeleton tracking (see Section 5.2.3), we choose ResNet [28] for joint detection. The original ResNet has fully connected layers, but we want heatmaps output to plot the probability distribution. Hence, we use ResNet-50 and keep all its convolutional, pooling and activation layers, but replace the fully connected output layers with 3 transposed convolutional layers ( $3 \times 3$ ) up-scaling to 64 by 64 heatmaps. Ideally, the output heatmaps should have at least 256 by 256 pixels, which matches the input RGB image

size. Higher resolution means higher detection accuracy. However, as the real-time performance of the system is one of the core requirements, and increased output size significantly prolongs the training and processing time, we use 64 by 64 heatmaps as the output size for the system. In addition, upon testing, 64 by 64 output heatmaps demonstrate satisfactory results (as shown in Section 5.4).

### 5.3.5 Training dataset generation

In this section we introduce how we train the instance segmentation (Mask R-CNN), domain adaptation (CycleGAN) and the joint detector. An overview of the training flow is shown in Figure 5.2.

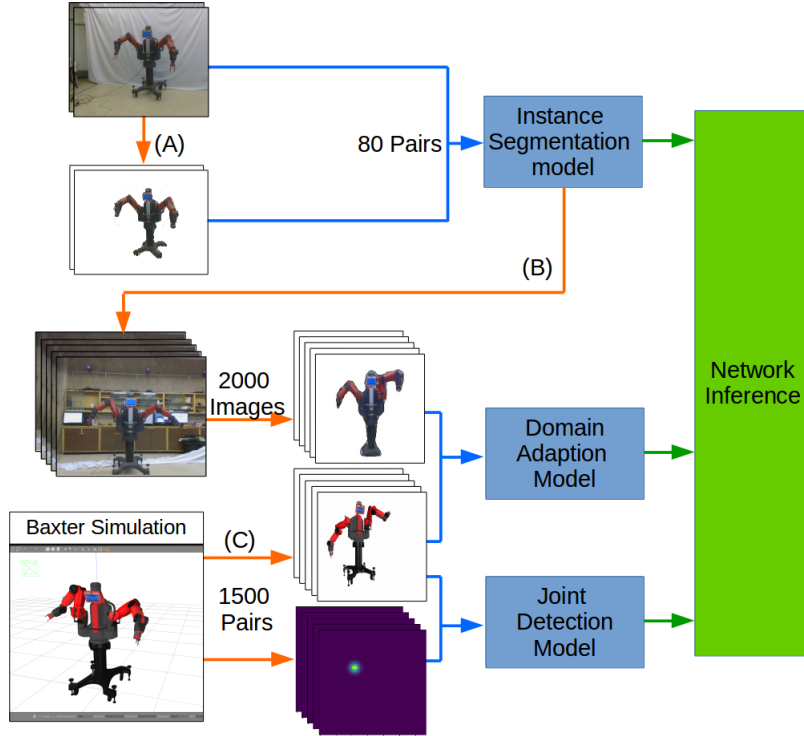


Figure 5.2: Preparation steps for training data

#### Instance segmentation

Mask R-CNN takes RGB images and description files, which contain different object ids and their corresponding polygon masks (represented by vertices of the polygon). Mask R-CNN is capable of multiple objects tracking at the same

time, but in our case, we only have one object (i.e. Baxter robot) to segment. In terms training set labelling, one can label the polygon outline of the target object either automatically or manually.

BasNet [59] is a boundary-aware salient object detection tool, which does not pay attention to any specific objects, but uses saliency maps to predict the salient objects in the images. This method works well with a clean background (e.g. white curtain), but cutouts become less accurate in complex backgrounds (e.g. labs). However, we can use image augmentation to swap the clean background (where BasNet performs well) with complex backgrounds. BasNet produces a binary mask map, in which the object of interest has a pixel intensity of 1 whereas the rest of the images have 0-pixel intensities. We can then use OpenCV to find the contours within the binary image and extract the vertices of the salient object. On the other hand, manual labelling is more accurate and stable (especially at certain camera perspectives, where only the side of the object is revealed), but it is a tedious job.

Therefore, for the training set, we use the combination of both automatic and manual labels. Instead of training a brand new Mask R-CNN model from scratch, we use transfer learning from a pre-trained model. Inside the training set, 80 images are manually labelled, and 60 images are automatically labelled, with randomly augmented backgrounds. However, the current Mask R-CNN still has limitations in producing a sub-pixel level of fine segmentation, resulting in ambiguity around the grippers. Recent works (e.g. Detectron2 [82]) have developed finer masking, which can be employed in our system in future.

## **Domain adaptation**

As discussed earlier, CycleGAN does not require paired image training sets. This feature is favorable for our system, because one can generate an unlimited number of synthetic images from the simulator with different camera poses and robot arm configurations effortlessly, but it is relatively hard to obtain real images with corresponding arm configurations. Hence, our training sets consist of two image domains:

1. Image set A  $\{I_r\}$  (Real images): We captured 2000 images of the Baxter robot using the RealSense D415 RGB-D camera, with only the RGB sensor enabled. The Baxter robot’s base was fixed, while the arms were pre-programmed to execute random manipulations. The camera was held moving in front of the Baxter robot, and yawing up to 70 degrees in either direction. The images were then processed by instance segmentation using Mask R-CNN model we trained previously. This step output images with a clean background.
2. Image set B  $\{I_s\}$  (Synthetic images): We used Gazebo to generate 1500 synthetic images, in which the Baxter robot moves under a random motion sequence. The virtual camera was set to have the same intrinsic parameters as the camera used to capture  $\{I_r\}$ . While the Baxter robot was moving in the simulator, the virtual camera was moving as well (but keep the Baxter robot at the centre of the frame). Besides the lighting source appeared in the virtual scene by default, we added another virtual spotlight to illuminate the background. The background of the virtual scene was set to white to match the set A  $\{I_r\}$ .

At this point, we have two image datasets set up ( $\{I_r\}$  and  $\{I_s\}$ ). The image size for both sets was set to 256 by 256.

### **Joint detection**

Baxter robot has 7 joints on each arm, but we only use 5 joints on each arm due to low visibility for the two joints located at the wrist and the elbow. That translates to 10 heatmaps as output for each execution (5 heatmaps per arm). For training of the joint detector, the goal is to minimize the L2 loss function, which measures the difference between the heatmap and the ground-truth joint locations. This requires the projection from ground-truth 3D locations to 2D image space. Thanks to the domain adaptation we employ, the training step is completed relatively effortlessly with a large set of simulation images. Therefore, we generated 8000 images of Baxter from the Gazebo simulator, each has different arm configurations and camera poses. In order to generate images

that are closer to the simulation-style images obtained from domain adaptation, we import the camera model (Intel RealSense D415) and the URDF files of the Baxter robot. Gazebo simulator has a physics engine to simulate real-world physics. Hence, given any camera pose  $P_{cam}$  and a unique robot arm configuration, a 2D pixel coordinates of each joint  $P_j$  is calculated. These 2D coordinates of the joint locations are then converted to a Gaussian distribution heatmap (64 by 64 pixels). In addition, to improve the robustness of the detector, we apply image augmentation to the training images, including random brightness adjustment, random color saturation, cropping and various backgrounds from COCO datasets [45].

Therefore, we collected 10000 synthetic images of the Baxter robot from the simulator with random backgrounds as discussed above, with each attached to a set of labels of each joint. This dataset was used for joint detector training.

## 5.4 Experimental results

### 5.4.1 Experimental datasets

Ten datasets were collected with three different variables (camera poses, arm actions and image background as listed in Table 5.1). The camera was hand-held and moved in a circular arc in front of the Baxter robot.

There are two types of camera poses: dynamic and static. For dynamic camera poses, camera poses span over 120 degrees while keeping the Baxter robot at the centre of the frame. Due to the relatively low resolution of the images (256 by 256), we intentionally position the camera around 2.5 to 3.5 meters away from the Baxter robot so that the Baxter robot took major space in each frame. On the other hand, in static camera poses, the camera was fixed heading towards the Baxter robot.

Additionally, for arm actions, pick-and-place motion is a pre-programmed sequence in which the Baxter robot uses its gripper to pick black boxes and place them on a tray. Waving actions are random arm motion sequences, but the Baxter robot tries to move all joints to their maximum reachable volumes.

With respect to background selection, the natural background represents



Table 5.1: Testing datasets

| Dataset  | # of images | Features |                |               |
|----------|-------------|----------|----------------|---------------|
|          |             | Camera   | Arm action     | Background    |
| $T_1$    | 10967       | Dynamic  | Waving         | White, Static |
| $T_2$    | 2312        | Static   | Pick-and-Place | White, Static |
| $T_3$    | 6856        | Static   | Waving         | White, Static |
| $T_4$    | 9810        | Dynamic  | Waving         | White, Static |
| $T_5$    | 2503        | Static   | Pick-and-Place | White, Static |
| $T_6$    | 16355       | Dynamic  | Waving         | Dynamic       |
| $T_7$    | 3330        | Dynamic  | Pick-and-Place | Dynamic       |
| $T_8$    | 3190        | Static   | Pick-and-Place | Static        |
| $T_9$    | 7145        | Static   | Waving         | Static        |
| $T_{10}$ | 13233       | Dynamic  | Waving         | Static        |
| $T_{11}$ | 11938       | Static   | Waving         | Static        |
| $T_{12}$ | 8544        | Static   | Waving         | Dynamic       |
| $T_{13}$ | 4483        | Static   | Waving         | Dynamic       |
| $T_{14}$ | 5199        | Static   | Waving         | Dynamic       |
| $T_{15}$ | 4202        | Static   | Waving         | Dynamic       |
| $T_{16}$ | 2104        | Static   | Pick-and-Place | Static        |
| $T_{17}$ | 2180        | Static   | Pick-and-Place | Dynamic       |
| $T_{18}$ | 2287        | Static   | Pick-and-Place | Dynamic       |

the normal lab scene with all random objects appearing in the background. The white background, in contrast, is the scene where we hung a white curtain behind the Baxter robot, so that no other object distractions. All datasets were captured from the same imaging sensor (RGB sensor from the Intel RealSense D415 camera), with the output resolution of  $640 \times 480$  and a framerate of 30 fps.

#### 5.4.2 Groundtruth data generation

Baxter robot is packaged with precise joint feedback sensors, which can produce reliable poses of each joint relative to the robot center using forward/inverse kinematics. Vicon motion capture system is then used to construct robot-to-world poses. Along with the relative poses from each joint to the robot centre, we then obtain the absolute poses of each joint. Additionally, the camera

Table 5.2: PCK@0.2 scores

| PCK@0.2            | Joints |          |        |        |        | Overall |
|--------------------|--------|----------|--------|--------|--------|---------|
| Methods            | Base   | Shoulder | Elbow  | Wrist  | Hand   | mAP     |
| White Background   |        |          |        |        |        |         |
| DREAM              | 0.9994 | 0.9826   | 0.8892 | 0.9763 | 0.9574 | 0.9610  |
| Ours               | 0.9816 | 0.9824   | 0.9919 | 0.9283 | 0.8924 | 0.9554  |
| Natural Background |        |          |        |        |        |         |
| DREAM              | 0.9986 | 0.9936   | 0.9927 | 0.8387 | 0.7900 | 0.9181  |
| Ours               | 0.9967 | 0.9958   | 0.9953 | 0.9559 | 0.8989 | 0.9685  |

poses are tracked by the same Vicon system. Hence, we can calculate the ground-truth coordinates of each joint projected to the image.

### 5.4.3 Quantitative evaluation metrics

We use 2D PCK@0.2 [3] and root mean square error (RMS) to evaluate the overall performance of our system. PCK@0.2 translates to the percentage of correct joint detection within a dataset where the distance between the prediction and the groundtruth is smaller than  $0.2 \times \text{torso diameter}$  [3]. All the bounding boxes of Baxter in the testing set are generated from the instance segmentation step. Besides PCK@0.2, we also evaluated other PCK scores against different tolerances ranging from 1 to 30 pixels (as shown in Figure 5.3 and Figure 5.4.3). RMS measures the 2D euclidean pixel distance between the ground truths and joint coordinates produced by our system. Also, as the Baxter robot has two symmetric arms, per-joint evaluation scores presented in the following tables and figures are the average values of the left and right joints.

### 5.4.4 Baseline methods comparison

Deep Robot-to-camera Extrinsic for Articulated Manipulators (DREAM) [43] is a robot joint detection pipeline published recently. DREAM also uses a simulator to generate training images for the detector. However, there are two significant distinctions between our proposed method and DREAM:

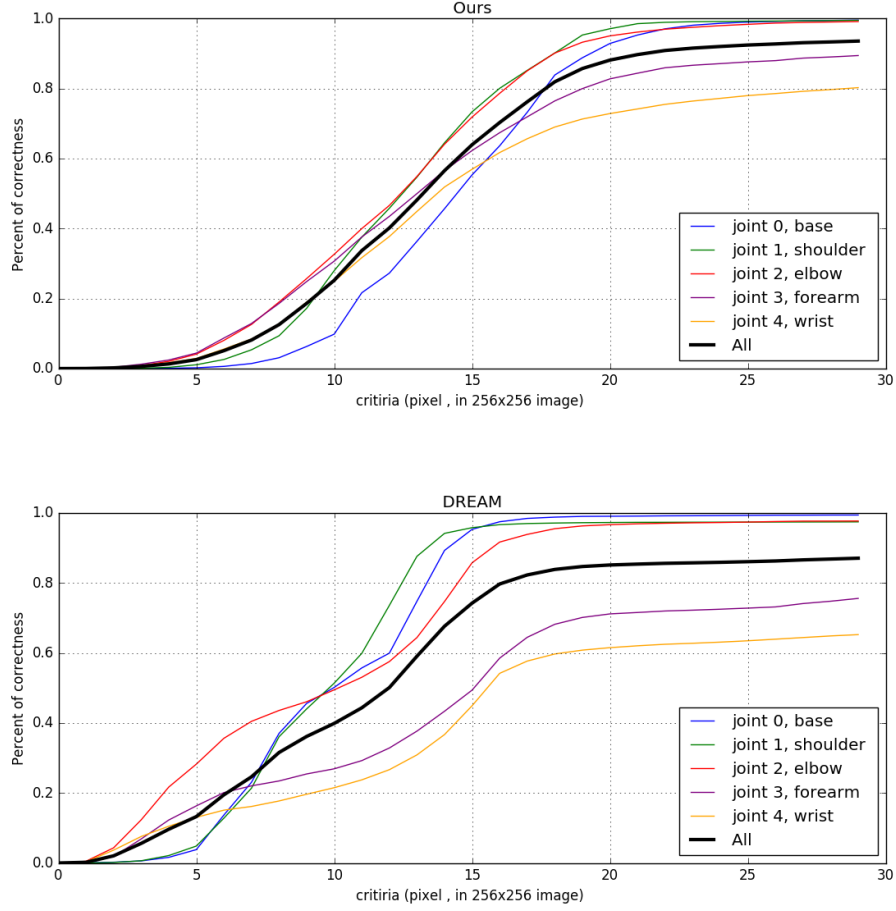


Figure 5.3: PCK scores for robot joint detection over natural lab background: our method versus DREAM [43]

1. DREAM applies random color, texture and illuminations to the robot in the simulator to generate augmented synthetic images for joint detector training. DREAM relies on this step to improve the generality of the joint detector. Therefore, during the execution, DREAM does not modify the input images from the camera. However, in our proposed system, we have an additional process of instance segmentation applied to the image feeds.
2. DREAM only returns joint detections with a sufficiently high confidence level and omits others. Our proposed method, however, always returns the joint detections with the highest probability (within the predicted

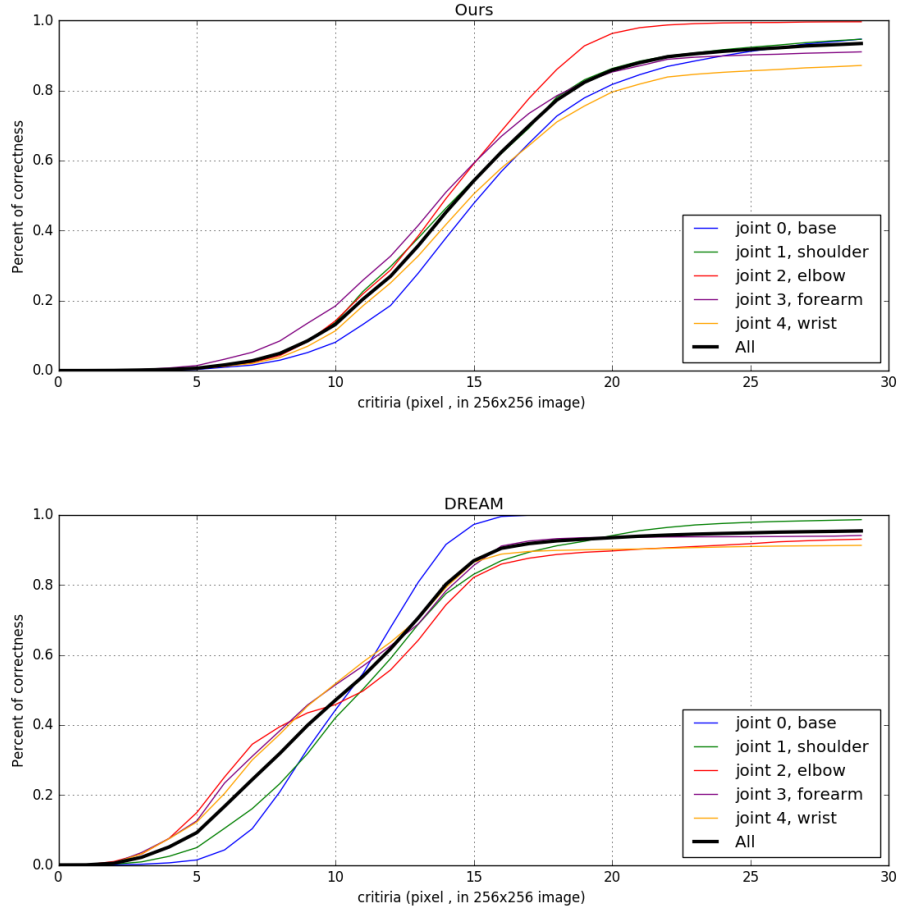


Figure 5.4: PCK result of white background datasets

heatmap) regardless of their absolute values. The confident detection rate returned by DREAM is listed in Table 5.3. Also, for a fair comparison, we only report the evaluation scores of our method on the dataset where DREAM has high confidence (and hence return joint detections).

One of the motivations for using DREAM as the baseline model, besides it is one of the recently published methods which is claimed to have great detection performance [43], it has a pre-trained detection model for the Baxter robot. Therefore, we used this pre-trained model for all the testing and comparisons.

Figure 5.3 and 5.4.3 demonstrate the PCK scores against different pixel

Table 5.3: DREAM Detection Rate

| Percentage | Joints |          |       |       |       | Overall |
|------------|--------|----------|-------|-------|-------|---------|
| Dataset    | Base   | Shoulder | Elbow | Wrist | Hand  | Mean    |
| White      | 93.49  | 87.09    | 66.94 | 67.61 | 72.16 | 78.95   |
| Natural    | 74.03  | 69.37    | 64.64 | 54.30 | 53.06 | 64.37   |

Table 5.4: RMS scores for joint detection in two background settings

| RMS                | Joints |          |       |       |       | Overall |
|--------------------|--------|----------|-------|-------|-------|---------|
| Methods            | Base   | Shoulder | Elbow | Wrist | Hand  | Mean    |
| White Background   |        |          |       |       |       |         |
| DREAM              | 9.62   | 11.44    | 16.13 | 13.68 | 17.38 | 13.63   |
| Ours               | 13.94  | 14.14    | 13.00 | 16.61 | 20.78 | 15.69   |
| Natural Background |        |          |       |       |       |         |
| DREAM              | 9.67   | 10.44    | 11.04 | 26.66 | 34.02 | 19.37   |
| Ours               | 13.26  | 12.42    | 13.45 | 15.22 | 22.02 | 15.28   |

errors in natural backgrounds and white backgrounds respectively. Detection performance is reported on all 5 joints, and the black line within the two figures represents the average performance over all of the joints. Both methods (ours and DREAM) perform relatively worse in detecting the wrist, which is the distal joint. In our case, this behaviour is caused by instance segmentation imperfection, where the distal joint is often chopped off by the algorithm mistakenly. In addition, compared to our method, DREAM provides very inconsistent detection results in different background settings, as demonstrated in Table 5.2 and 5.4. DREAM slightly outperforms our method in white background, but the detection performance drops significantly in natural background. This is due to the instance segmentation and domain adaptation steps in our pipeline, which drastically reduce the interference from the background. This dependence on instance segmentation and domain adaptation, on the other hand, also causes failure cases, as illustrated in Figure 5.5.

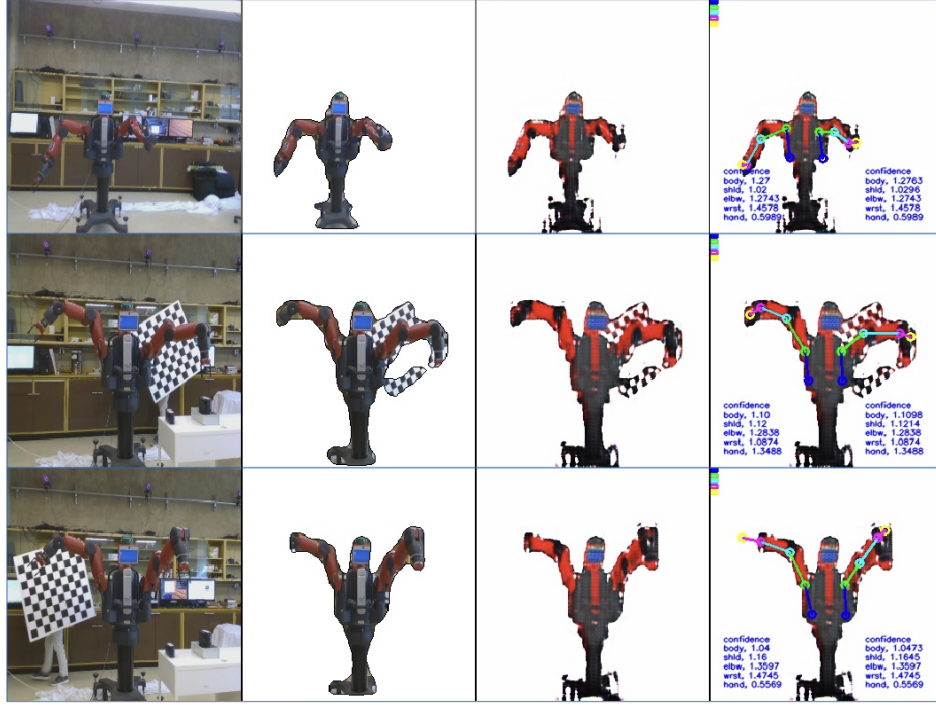


Figure 5.5: Examples of some common failure cases. Each row is one case. In the top one, the left-arm wrist portion is cutoff in segmentation resulting in the incorrect detection for the left wrist joint (the joint labelled as yellow circle marker), and the right wrist is in an occluded pose where CycleGAN failed to reconstruct the proper texture. In the second row, a person was holding a checkerboard moving in the back, and segmentation produced a mask consisting of some noise from the checkerboard, and the left wrist is cutoff again. In the third row, masking managed to filter out the checkerboard but still affected by it. Overall, the wrist and elbow joints detections are affected most.

## 5.5 Conclusion

In this chapter we presented a novel pipeline for robot arm joint detection using only RGB images. Our proposed method is capable of estimating the joint poses without joint feedback (e.g. encoders). The system consists of instance segmentation, domain adaptation and joint detection based on modified ResNet. Thanks to the publicly accessible pre-trained instance segmentation model and domain adaptation, the system shows good performance with minimal manually labelled datasets and substantial synthetic images generated by the robot simulator. Extensive quantitative evaluations show our proposed

method outperforms other state-of-the-art methods (DREAM) in terms of detection consistency and accuracy in different backgrounds.

# Chapter 6

## Conclusion and Future Work

### 6.1 Conclusion

This thesis has demonstrated the development and implementation of three engineering projects employing computer vision and machine learning. All three proposed systems demonstrated good results in hardware testing, and revealed the ability of computer vision and machine learning to yield designs which would not be possible without them.

In the learned self-driving vehicle project, we built a low-cost hardware platform and developed software for the vehicle enabling the following functionalities based on end-to-end learning:

- Lane-keeping control
- Traffic light/sign detection
- Bounding box-based traffic sign distance estimation
- Obstacle avoidance using an ultrasonic sensor

The lane-keeping control, which employs a trained multi-layer convolutional neural network (CNN), outperforms human drivers in terms of consistency and smoothness when driving on a closed-loop track. Traffic sign detection, which is based on the YOLO V4 classifier, achieves a 98.233% detection rate and estimates the distance of the sign within  $\pm 0.05$  meters.

In the second project, we successfully demonstrated a predictive display pipeline for robotic teleoperation tasks over high latency communication links,



which generates 2D images from a local 3D scene model given the predicted pose of the remote robot, providing instant visual feedback for the human operator. To address any scene changes to the remote site, the pipeline updates the local 3D map according to the new images received. The system uses the 3D scene model and the images captured during the pre-processing stage as the prior knowledge to train a generative model to produce photo-realistic synthetic images for the human operator. Our proposed pipeline outperforms other state-of-the-art methods in terms of quality and completeness of the images in both qualitative and quantitative experiments.

The third project presented a pipeline for vision-based estimation of robot manipulators’ joint positions. In contrast to high-end robot arms which employ high-resolution onboard encoders to measure joint angles, our method employs only RGB images from a monocular camera. Our pipeline implements instance segmentation, domain adaptation and joint detector. The system removes the need for manual labelling of training data by employing a generative model plus a simulation environment to produce a large amount of labelled training data. Comparing our method against other state-of-the-art joint detection methods in both qualitative and quantitative tests demonstrates that our design performs as well or better than existing work.

## 6.2 Limitations of work

While all three projects described in this thesis showed good performance, limitations of the current work were identified in all three cases:

- Learned self-driving vehicle project:
  1. The end-to-end learning-based driving control outputs only discrete commands, which works well for the simple hardware car we built for this project, but would not be feasible for controlling a steered-wheel vehicle or real-world passenger car.
  2. The traffic sign distance estimation module relies solely on 2D bounding box information. This approach is valid when the car

approaches traffic signs head-on. However, this assumption may be violated in real life if the car approaches the signs at an angle.

3. The experimental arena is an idealized environment which lacks the presence of static obstacles and dynamic actors such as pedestrians or other vehicles.

- Predictive display project:

1. While the proposed system has the ability to handle previously unobserved scenes, performance is significantly reduced compared to the previously-seen case.
2. The 3D scene reconstruction, which is based on TSDF volume integration, fails to update dynamic elements in the scene such as moving people.
3. The pre-processing stage (data collection and generative model training) takes a lot of time, which is not suitable for rapid exploration of unknown environments.

- Vision-based joint detection system:

1. The system requires a CAD model of the robot arm (or other manipulator) in order to generate synthetic images for training. Without such a model, manual data labeling is required.
2. Since the system is dependent on both instance segmentation and domain adaptation, the pipeline may fail to estimate joint states at frames when the background has a similar appearance to the robot arm.

## 6.3 Future work

In order to address the limitations listed in Section 6.2, the following future work should be performed:

1. Implement an end-to-end learning controller using a steered-wheel vehicle, and test the resulting design in a realistic car simulator (e.g. Carla).

2. Develop a more sophisticated self-driving perception system which fuses radar and LiDAR sensors with the camera feed.
3. Modify the generative model to better support novel (previously unobserved) objects in the scene.
4. Integrate scene semantic understanding and other geometric representation methods with our pipeline to achieve better visual rendering performance within dynamically changing environments.
5. Improve the output resolution of the heatmaps used in the joint detector to improve the accuracy of joint states.
6. Implement a layer of outlier removal based on optical flow or statistical prediction methods to increase robustness and accuracy of the joint state estimator.

# References

- [1] M. L. Aarizou and N.-E. Berrached, “ROS-based telerobotic application for transmitting high-bandwidth kinematic data over a limited network,” *International Journal of Control, Automation and Systems*, vol. 17, no. 2, pp. 445–453, 2018. DOI: 10.1007/s12555-018-0047-4.
- [2] D. S. Alexiadis, P. Kelly, P. Daras, N. E. O’Connor, T. Boubekeur, and M. B. Moussa, “Evaluating a dancer’s performance using kinect-based skeleton tracking,” in *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 659–662.
- [3] M. Andriluka, L. Pishchulin, P. V. Gehler, and B. Schiele, “2d human pose estimation: New benchmark and state of the art analysis,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, IEEE Computer Society, 2014, pp. 3686–3693. DOI: 10.1109/CVPR.2014.471. [Online]. Available: <https://doi.org/10.1109/CVPR.2014.471>.
- [4] M. Andriluka, S. Roth, and B. Schiele, “Monocular 3d pose estimation and tracking by detection,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 2010, pp. 623–630.
- [5] A. Arnab and P. H. S. Torr, “Pixelwise instance segmentation with a dynamically instantiated network,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, IEEE Computer Society, 2017, pp. 879–888. DOI: 10.1109/CVPR.2017.100. [Online]. Available: <https://doi.org/10.1109/CVPR.2017.100>.
- [6] D. H. Ballard and C. M. Brown, *Computer vision*. Prentice-Hall, 1982.
- [7] A. K. Bejczy, W. S. Kim, and S. C. Venema, “The phantom robot: Predictive displays for teleoperation with time delay,” in *Proceedings of the 1990 IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, USA, May 13-18, 1990*, IEEE, 1990, pp. 546–551. DOI: 10.1109/ROBOT.1990.126037. [Online]. Available: <https://doi.org/10.1109/ROBOT.1990.126037>.
- [8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.

- [9] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, “Explaining how a deep neural network trained with end-to-end learning steers a car,” *arXiv preprint arXiv:1704.07911*, 2017.
- [10] R. H. Borcherts, J. L. Jurzak, S.-P. Liou, and T.-L. A. Yeh, *System and method for automatically steering a vehicle within a lane in a road*, US Patent 5,245,422, Sep. 1993.
- [11] Z. Cao, G. H. Martinez, T. Simon, S.-E. Wei, and Y. A. Sheikh, “Open-pose: Realtime multi-person 2d pose estimation using part affinity fields,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019. DOI: 10.1109/tpami.2019.2929257.
- [12] Y. Chen, C. Shen, X.-S. Wei, L. Liu, and J. Yang, “Adversarial posenet: A structure-aware convolutional network for human pose estimation,” *CoRR*, vol. abs/1705.00389, 2017. arXiv: 1705.00389. [Online]. Available: <http://arxiv.org/abs/1705.00389>.
- [13] P. Y. Chua, T. Ilchner, and D. G. Caldwell, “Robotic manipulation of food products—a review,” *Industrial Robot: An International Journal*, 2003.
- [14] D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural networks*, vol. 32, pp. 333–338, 2012.
- [15] A. Coates, P. Abbeel, and A. Y. Ng, “Apprenticeship learning for helicopter control,” *Communications of the ACM*, vol. 52, no. 7, pp. 97–105, 2009.
- [16] D. Cobzas and M. Jägersand, “Tracking and predictive display for a remote operated robot using uncalibrated video,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*, IEEE, 2005, pp. 1847–1852. DOI: 10.1109/ROBOT.2005.1570382. [Online]. Available: <https://doi.org/10.1109/ROBOT.2005.1570382>.
- [17] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised, Extended And Updated Edition*. Springer International Publishing, 2017.
- [18] B. Curless and M. Levoy, “A volumetric method for building complex models from range images,” *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH 96*, 1996. DOI: 10.1145/237170.237269.
- [19] A. de la Escalera, L. E. Moreno, M. A. Salichs, and J. M. Armingol, “Road traffic sign detection and classification,” *IEEE Transactions on Industrial Electronics*, vol. 44, no. 6, pp. 848–859, 1997. DOI: 10.1109/41.649946.

- [20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
- [21] R. O. Duda and P. E. Hart, “Use of the hough transformation to detect lines and curves in pictures,” *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.
- [22] *Elon musk on cameras vs lidar for self driving and autonomous cars*, Apr. 2019. [Online]. Available: <https://www.youtube.com/watch?v=HM23sjhtk4Q>.
- [23] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, 2014. DOI: 10.1016/j.patcog.2014.01.005. [Online]. Available: <https://doi.org/10.1016/j.patcog.2014.01.005>.
- [24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.
- [25] —, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [26] G. He, S. Zhong, and J. Guo, “A lightweight and scalable visual-inertial motion capture system using fiducial markers,” *Autonomous Robots*, vol. 43, no. 7, pp. 1895–1915, 2019.
- [27] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, “Mask R-CNN,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2020. DOI: 10.1109/TPAMI.2018.2844175. [Online]. Available: <https://doi.org/10.1109/TPAMI.2018.2844175>.
- [28] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: <http://arxiv.org/abs/1512.03385>.
- [29] C. Hernandez, M. Bharatheesha, W. Ko, H. Gaiser, J. Tan, K. van Deurzen, M. de Vries, B. Van Mil, J. van Egmond, R. Burger, *et al.*, “Team delft’s robot winner of the amazon picking challenge 2016,” in *Robot World Cup*, Springer, 2016, pp. 613–624.
- [30] T. H. Hong, C. Rasmussen, T. Chang, and M. Shneier, “Road detection and tracking for autonomous mobile robots,” in *Unmanned Ground Vehicle Technology IV*, International Society for Optics and Photonics, vol. 4715, 2002, pp. 311–319.

- [31] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, “Detection of traffic signs in real-world images: The german traffic sign detection benchmark,” in *The 2013 international joint conference on neural networks (IJCNN)*, IEEE, 2013, pp. 1–8.
- [32] H. Hu, C. P. Quintero, H. Sun, and M. Jägersand, “On-line reconstruction based predictive display in unknown environment,” in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, IEEE, 2015, pp. 4446–4451. DOI: 10.1109/ICRA.2015.7139814. [Online]. Available: <https://doi.org/10.1109/ICRA.2015.7139814>.
- [33] X. Huang and S. J. Belongie, “Arbitrary style transfer in real-time with adaptive instance normalization,” in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, IEEE Computer Society, 2017, pp. 1510–1519. DOI: 10.1109/ICCV.2017.167. [Online]. Available: <https://doi.org/10.1109/ICCV.2017.167>.
- [34] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, “Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1325–1339, 2013.
- [35] M. Jägersand, “Image based predictive display for tele-manipulation,” in *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, IEEE Robotics and Automation Society, 1999, pp. 550–556. DOI: 10.1109/ROBOT.1999.770034. [Online]. Available: <https://doi.org/10.1109/ROBOT.1999.770034>.
- [36] J. Jin, L. Petrich, S. He, M. Dehghan, and M. Jägersand, “Long range teleoperation for fine manipulation tasks under time-delay network conditions,” *CoRR*, vol. abs/1903.09189, 2019. arXiv: 1903.09189. [Online]. Available: <http://arxiv.org/abs/1903.09189>.
- [37] A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother, “Instancecut: From edges to instances with multicut,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, IEEE Computer Society, 2017, pp. 7322–7331. DOI: 10.1109/CVPR.2017.774. [Online]. Available: <https://doi.org/10.1109/CVPR.2017.774>.
- [38] G. Klein and D. W. Murray, “Parallel tracking and mapping for small AR workspaces,” in *Sixth IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR 2007, 13-16 November 2007, Nara, Japan*, IEEE Computer Society, 2007, pp. 225–234. DOI: 10.1109/ISMAR.2007.4538852. [Online]. Available: <https://doi.org/10.1109/ISMAR.2007.4538852>.

- [39] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154.
- [40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [41] J.-w. Lee, *Model based predictive control for automated lane centering/changing control systems*, US Patent 8,190,330, May 2012.
- [42] T. E. Lee, J. Tremblay, T. To, J. Cheng, T. Mosier, O. Kroemer, D. Fox, and S. Birchfield, "Camera-to-robot pose estimation from a single image," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 9426–9432. DOI: 10.1109/ICRA40945.2020.9196596.
- [43] T. E. Lee, J. Tremblay, T. To, J. Cheng, T. Mosier, O. Kroemer, D. Fox, and S. Birchfield, "Camera-to-robot pose estimation from a single image," in *International Conference on Robotics and Automation (ICRA)*, 2020. [Online]. Available: <https://arxiv.org/abs/1911.09231>.
- [44] N. Li, H. Chen, I. Kolmanovsky, and A. Girard, "An explicit decision tree approach for automated driving," in *Dynamic Systems and Control Conference*, American Society of Mechanical Engineers, vol. 58271, 2017, V001T45A003.
- [45] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., ser. Lecture Notes in Computer Science, vol. 8693, Springer, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1\_48. [Online]. Available: [https://doi.org/10.1007/978-3-319-10602-1\\_48](https://doi.org/10.1007/978-3-319-10602-1_48).
- [46] S. Liu, J. Jia, S. Fidler, and R. Urtasun, "SGN: sequential grouping networks for instance segmentation," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, IEEE Computer Society, 2017, pp. 3516–3524. DOI: 10.1109/ICCV.2017.378. [Online]. Available: <https://doi.org/10.1109/ICCV.2017.378>.
- [47] Z. Liu, K. Hu, and K.-w. Chung, "Nonlinear analysis of a closed-loop tractor-semitrailer vehicle system with time delay," *Mechanical Systems and Signal Processing*, vol. 76, pp. 696–711, 2016.



- [48] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, IEEE Computer Society, 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965. [Online]. Available: <https://doi.org/10.1109/CVPR.2015.7298965>.
- [49] D. Lovi, “Incremental free-space carving for real-time 3D reconstruction,” M.S. thesis, University of Alberta, 2011.
- [50] R. Maier, K. Kim, D. Cremers, J. Kautz, and M. Nießner, “Intrinsic3D: High-quality 3D reconstruction by joint appearance and geometry optimization with spatially-varying lighting,” in *International Conference on Computer Vision (ICCV)*, 2017.
- [51] I. Masaki, *Adaptive motor vehicle cruise control*, US Patent 4,987,357, Jan. 1991.
- [52] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “Nerf: Representing scenes as neural radiance fields for view synthesis,” *CoRR*, vol. abs/2003.08934, 2020. arXiv: 2003.08934. [Online]. Available: <https://arxiv.org/abs/2003.08934>.
- [53] A. Mittal, A. K. Moorthy, and A. C. Bovik, “No-reference image quality assessment in the spatial domain,” *IEEE Transactions on Image Processing*, vol. 21, no. 12, pp. 4695–4708, 2012.
- [54] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. DOI: 10.1109/RO.2017.2705103.
- [55] R. A. Newcombe, A. Fitzgibbon, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and et al., “KinectFusion: Real-time dense surface mapping and tracking,” *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011. DOI: 10.1109/ismar.2011.6092378.
- [56] S. Oleson, G. Landis, M. McGuire, and G. Schmidt, “HERRO mission to mars using telerobotic surface exploration from orbit,” *Journal of the British Interplanetary Society*, vol. 64, pp. 304–313, Sep. 2011.
- [57] C. Park and K. Park, “Design and kinematics analysis of dual arm robot manipulator for precision assembly,” in *2008 6th IEEE International Conference on Industrial Informatics*, IEEE, 2008, pp. 430–435.
- [58] K. Pauwels, L. Rubio, V. Ivan, S. Vijayakumar, and E. Ros, “Real-time rgb-d-based object and manipulator pose estimation,” in *Proc. Workshop on RGB-D Advanced Reasoning with Depth Cameras, Robotics: Science and Systems (R: SS2014)*, Citeseer, 2014.

- [59] X. Qin, Z. V. Zhang, C. Huang, C. Gao, M. Dehghan, and M. Jägersand, “Basnet: Boundary-aware salient object detection,” in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, Computer Vision Foundation / IEEE, 2019, pp. 7479–7489. DOI: 10.1109/CVPR.2019.00766. [Online]. Available: [http://openaccess.thecvf.com/content%5C\\_CVPR%5C\\_2019/html/Qin%5C\\_BASNet%5C\\_Boundary-Aware%5C\\_Salient%5C\\_Object%5C\\_Detection%5C\\_CVPR%5C\\_2019%5C\\_paper.html](http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Qin%5C_BASNet%5C_Boundary-Aware%5C_Salient%5C_Object%5C_Detection%5C_CVPR%5C_2019%5C_paper.html).
- [60] A. Rachmielowski, N. Birkbeck, M. Jägersand, and D. Cobzas, “Realtime visualization of monocular data for 3d reconstruction,” in *Fifth Canadian Conference on Computer and Robot Vision, CRV 2008, Windsor, Ontario, Canada, May 28-30, 2008*, IEEE Computer Society, 2008, pp. 196–202. DOI: 10.1109/CRV.2008.48. [Online]. Available: <https://doi.org/10.1109/CRV.2008.48>.
- [61] K. Rangarajan, M. Shah, and D. Van Brackle, “Optimal corner detector,” *Computer Vision, Graphics, and Image Processing*, vol. 48, no. 2, pp. 230–245, 1989.
- [62] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: <http://arxiv.org/abs/1506.02640>.
- [63] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.
- [64] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 91–99. [Online]. Available: <http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks>.
- [65] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [66] J. Sachs, L. A. A. Andersson, J. Araújo, C. Curescu, J. Lundsjö, G. Rune, E. Steinbach, and G. Wikström, “Adaptive 5G low-latency communication for tactile internet services,” *Proceedings of the IEEE*, vol. 107, no. 2, pp. 325–349, 2019.

- [67] T. Schmidt, R. A. Newcombe, and D. Fox, “DART: dense articulated real-time tracking with consumer depth cameras,” *Auton. Robots*, vol. 39, no. 3, pp. 239–258, 2015. DOI: 10.1007/s10514-015-9462-z. [Online]. Available: <https://doi.org/10.1007/s10514-015-9462-z>.
- [68] J. L. Schönberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [69] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, “Pixelwise view selection for unstructured multi-view stereo,” in *European Conference on Computer Vision (ECCV)*, 2016.
- [70] D. Shreiner, *OpenGL reference manual: the official reference document to OpenGL, version 1.4: OpenGL architecture review board*. Addison Wesley, 2004.
- [71] S. Soylu, F. Firmani, B. J. Buckham, and R. P. Podhorodeski, “Comprehensive underwater vehicle-manipulator system teleoperation,” in *OCEANS 2010 MTS/IEEE SEATTLE*, IEEE, 2010, pp. 1–8.
- [72] R. Szeliski, *Computer vision: algorithms and applications*. Springer, 2011, pp. 10–16.
- [73] *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*. DOI: 10.4271/j3016\_201806. [Online]. Available: [https://doi.org/10.4271/j3016\\_201806](https://doi.org/10.4271/j3016_201806).
- [74] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, “Stanley: The robot that won the darpa grand challenge,” *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.
- [75] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, IEEE Computer Society, 2014, pp. 1653–1660. DOI: 10.1109/CVPR.2014.214. [Online]. Available: <https://doi.org/10.1109/CVPR.2014.214>.
- [76] *Vero: Compact super wide camera by vicon*, Jun. 2020. [Online]. Available: <http://www.vicon.com/hardware/cameras/vero/>.
- [77] Y. Wang, E. K. Teoh, and D. Shen, “Lane detection and tracking using b-snake,” *Image and Vision computing*, vol. 22, no. 4, pp. 269–280, 2004.
- [78] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/tip.2003.819861.

- [79] B. Widrow and M. A. Lehr, “30 years of adaptive neural networks: Perceptron, madaline, and backpropagation,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.
- [80] C. R. Winter and B. Widrow, “Madaline rule ii: A training algorithm for neural networks,” in *Second Annual International Conference on Neural Networks*, 1988, pp. 1–401.
- [81] J. Wu, Q. Wang, X. Wei, and H. Tang, “Studies on improving vehicle handling and lane keeping performance of closed-loop driver-vehicle system with integrated chassis control,” *Mathematics and Computers in Simulation*, vol. 80, no. 12, pp. 2297–2308, 2010.
- [82] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, <https://github.com/facebookresearch/detectron2>, 2019.
- [83] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese, “Interactive gibbon benchmark: A benchmark for interactive navigation in cluttered environments,” *IEEE Robotics Autom. Lett.*, vol. 5, no. 2, pp. 713–720, 2020. DOI: 10.1109/LRA.2020.2965078. [Online]. Available: <https://doi.org/10.1109/LRA.2020.2965078>.
- [84] Y. Yang and D. Ramanan, “Articulated human detection with flexible mixtures of parts,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2878–2890, 2013. DOI: 10.1109/TPAMI.2012.261. [Online]. Available: <https://doi.org/10.1109/TPAMI.2012.261>.
- [85] K. Yerev, D. Cobzas, and M. Jägersand, “Predictive display models for tele-manipulation from uncalibrated camera-capture of scene geometry and appearance,” in *Proceedings of the 2003 IEEE International Conference on Robotics and Automation, ICRA 2003, September 14-19, 2003, Taipei, Taiwan*, IEEE, 2003, pp. 2812–2817. DOI: 10.1109/ROBOT.2003.1242018. [Online]. Available: <https://doi.org/10.1109/ROBOT.2003.1242018>.
- [86] C. Yu and Y. Wang, “3d-scene-gan: Three-dimensional scene reconstruction with generative adversarial networks,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, OpenReview.net, 2018. [Online]. Available: <https://openreview.net/forum?id=SkNEsmJwf>.
- [87] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *CVPR*, 2018.
- [88] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

- [89] Y. Zuo, W. Qiu, L. Xie, F. Zhong, Y. Wang, and A. L. Yuille, “CRAVES: controlling robotic arm with a vision-based economic system,” *CoRR*, vol. abs/1812.00725, 2018. arXiv: 1812.00725. [Online]. Available: <http://arxiv.org/abs/1812.00725>.

# Appendix A

## Carla simulator setup

### A.1 Self-driving car simulator

Although we can evaluate the individual performance of each software component listed in Section 3.3.3, it is relatively difficult to evaluate the self-driving system as a whole. For instance, it is hard to quantitatively test the performance of the lane keeping control, as there is no ground-truth information regarding the distance of the vehicle to the lane markings. We could manually count the number of lane crossing in a certain session, but this would be tedious and inaccurate to do.

A self-driving car simulator (e.g. Carla [20]), on the other hand, provides a number of useful features to evaluate the performance of a self-driving control system. In this section, we describe the concept and the testing of our design within the Carla environment.

#### A.1.1 An overview of Carla simulator

Carla is an open-source platform designed for self-driving research which runs on the Unreal Engine 4 (UE4) [20]. There are two core elements of the Carla simulator:

- **World:** The core of the simulator. This contains the map, weather, lighting and other simulation settings. The world is a library which contains all the assets, while the Unreal Engine runs the simulation. Figure A.1 is an example of a Carla world. Users can view the world

from the screen but cannot directly interact with it or any actors (e.g. vehicles, pedestrians, sensors).

- Client: The Carla client connects to the world and retrieves information from it. The Client is the gateway for users to interact with the world and its actors. For example, once we set up the Carla world, we need to initialize a client instance to spawn and control all actors. Figure A.2 provides a screenshot of the manual control interface running on a client instance. In this mode, the user can control the throttle and steering angle of the spawned vehicle and thus interact with the world (e.g. crossing lanes, hitting a wall).



Figure A.1: A bird's eye view of the Carla world

In addition, Carla supports fully customized assets, which includes buildings, vehicles, roads, traffic signs, etc. This feature allows researchers from different parts of the world to employ this simulator, since road conditions (e.g. buildings, road signs, lane markings) may vary greatly from country to country.

### A.1.2 Self-driving features to be tested in Carla

As discussed before, a self-driving system is a sophisticated design which consists of multiple modules. It was not practical to build a full self-driving system



Figure A.2: An example of Carla client manual control interface

and test it in Carla. Hence, our goal is to test features developed for the DIY self-driving RC car which could not be quantitatively evaluated in lab settings.

In addition to sensors which can be mounted on vehicles (e.g RGB cameras, LIDAR, radar), Carla also has a collision detector and a lane invasion detector. These detectors enable quantitative evaluation of road detection and lane keeping as well as the obstacle avoidance function.

We will thus employ two functions of the Carla simulator: 1) Lane detection and keeping and 2) Obstacle avoidance.

### A.1.3 Simulated sensors setup

#### RGB camera

In Section 3.3.3 we discuss two approaches for lane keeping:

1. Road marking detection from perceived RGB images, followed by fitting the lane curvature to a polynomial function. We then control the vehicle based on the lateral offset and the dynamics model of the vehicle
2. End-to-end learning using a CNN which inputs RGB images and directly outputs the corresponding driving command.

Since the DIY RC car does not steered wheels, the first approach is difficult or impossible to use due to a lack of precise steering control available.



However, most automotive manufactures use the first approach (e.g. [10], [41]) for their lane departure warning systems. Therefore, in the Carla simulator environment, we test the model-based approach to lane keeping. An RGB camera is attached to the center of the vehicle and is facing forward. The FOV of the camera is set to 90 degrees to reduce distortion effects (as shown in Figure A.3).



Figure A.3: An image captured by the front-facing camera mounted on the vehicle in Carla simulator

### **Radar sensor**

In the DIY RC car, we use an ultrasonic sensor to measure distance to objects in front of the vehicle. However, Carla does not include ultrasonic sensors, but rather radar. A radar sensor emits Frequency Modulated Continuous Wave (FMCW), which is more reliable than an ultrasonic sound wave. Radar measurements provided by Carla are an array of points containing polar coordinates, distance and velocity as demonstrated in Figure A.4.



Figure A.4: Radar measured points drawn in front of the vehicle

### Events detector

The key functionality of the Carla simulator is the ability to detect collision and lane invasion. These features are achieved by two detectors:

1. Collision detector: We attach this detector to our vehicle (parent), and it registers an event every time the car collides against anything in the world. Hence we can log these events for evaluation.
2. Lane invasion detector: the world is created by OpenDRIVE files which contain road information such as lane markings. This detector determines if any lane marking is invaded based on the space between the marking and each wheel.

These two detectors enable us to collect information for quantitative evaluations.