Vision-Based Methods for Joint State Estimation of Robotic Manipulators

by

Mingjie Han

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

Master of Science

Department of Mechanical Engineering

University of Alberta

Abstract

This thesis applied a combination of machine learning and computer vision to an engineering research project, using a two-armed Baxter robot hardware platform. The challenge was estimating the robot arm's joint angles from monocular camera images. After evaluating several methods from traditional computer vision, we settled on the method of convolutional neural networks, which provided better accuracy and outlier rejection performance. A simulation environment toolchain was developed to generate automatically labelled training images for the neural network in order to eliminate the tedious manual labelling usually required for these methods. This brought the challenge of the domain gap between simulation and real-world images, which was solved using a generative adversarial network for transferring image textures. A hardware evaluation was performed for both joint keypoint detection and joint angle estimation performance, whose ground-truth values were accurately captured in the laboratory environment.

Preface

Chapter 4 of this thesis was submitted to IEEE Robotics and Automation Letters as "Image-Based Joint State Estimation Pipeline for Sensorless Manipulators" by Mingjie Han, Bowen Xie, Martin Barczyk and Alireza Bayat. Bowen was responsible for training the generative adversarial network used in the work. I was responsible for the remainder of the technical work, including keypoint detection network training, joint state estimation development, running experiments, and documenting results. Dr. Martin Barczyk and Dr. Alireza Bayat provided valuable advising and supervision throughout the research, and assisted in editing the paper.

Acknowledgements

I would like to thank Dr. Martin Barczyk and Dr. Alireza Bayat for all the guidance, enlightenment and supervision throughout the course of my graduate studies; Bowen Xie for the collaboration on the research projects; and Jun Jin for advice.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Humans performing simple and repetitive tasks manually use their eyes for sensing the environment and their brain for understanding it. In order to automate such tasks, computer-based alternatives to eyes and brains are needed. Cameras are an excellent and affordable alternative to human eyes, and camera types such as depth, infra-red (IR) and spectral offer capabilities beyond those of the human eye. Alternatives to the human brain are computer algorithms, which include traditional computer vision as well as the emerging learning-based artificial neural network algorithms.

Traditional computer vision algorithms have shown excellent performance in certain tasks, such as edge detection, but fail to handle more complicated cases such as detecting pedestrians [18]. For this reason, learning-based methods have become a popular tool to solve more complex problems involving computer vision. The most common learning-based approach for processing image data is the convolutional neural network (CNN), which was first proposed in the 1980s [20], [38].

The complexity of CNNs was limited by available computing power, but this changed when GPUs (Graphics Processing Unit) became widely available. The massive parallel computation ability of GPUs significantly boosted research into convolutional neural networks and their applications. At the same time, software frameworks for implementing and training neural networks have become more sophisticated and optimized. At the time of writing,

TensorFlow [45] and PyTorch [53] are the most popular platforms for machine learning, each providing an easy-to-use Application Programming Interface (API) for implementing and training neural networks.

Some applications of convolutional neural networks involve only monocular camera images, for instance human face recognition [37], [52], [70] and traffic monitoring [6], [13], [31]. But in other applications such as robotic manipulators and autonomous vehicles, the fields of kinematics and dynamics, closed-loop control and mechatronic designs must be integrated with CNN-based image processing.

Robotic manipulators have been deployed in different fields including academic research, recreation, and industrial applications. Most of these manipulators are equipped with motion planning and feedback systems. The resulting systems provide excellent dexterity, but fundamentally rely on joint state feedback from sensors embedded in the manipulator, typically encoders. Joint state feedback is a standard feature on industrial robot arms such as the Franka Emika Panda, Kuka Fortec, and UR10e. But this is not the case for equipment specifically designed for human control (e.g. cranes and excavators) or low cost manipulators (e.g. toys and do-it-yourself (DIY) manipulators).

Our research was motivated by studying the problem of adapting a human-operated knuckle-boom loading crane to autonomous operation. The existing system was not equipped with joint state feedback sensors. This is what led to our interest in developing a method to estimate the joint states of an articulated manipulator based on computer vision. As stated above, both conventional and CNN-based computer vision algorithms can be used for this purpose. We developed and tested several methods from both categories in our research.

1.1.1 Work Performed

The work performed and described in this thesis was the development of a software pipeline inputting a two-dimensional (2D) Red-Green-Blue (RGB) image of a robot arm and outputting its estimated joint angles. While this pipeline is built from three neural networks, the manual labour of labeling training data

is nearly eliminated by using a simulation environment to generate data, and the resulting domain gap is solved by utilizing a novel generative adversarial network to process the simulated images. This work was submitted as [24].

1.2 Thesis Outline

Chapter 1 explained the background and motivation of the research performed.

Chapter 2 explains the hardware and software used throughout this project. The hardware includes a Baxter robot, an Intel RealSense RGB-D (Red-Green-Blue and Depth) camera, and a Vicon motion capture system. For each of these systems, the hardware, functionality and software drivers are discussed. Analytical tools including forward kinematics, camera projection and camera calibration are discussed. A literature survey of software algorithms related to this work is provided.

Chapter 3 documents different approaches evaluated for sensorless robot arm joint state estimation, including color filtering, depth image processing, and convolutional neural networks. In each method, background information is provided followed by test results and observations. None of these methods ended up being used for the final system design due to a variety of factors, which are explained in detail.

Chapter 4 covers our chosen method for joint state estimation. This approach was found to provide the best performance in joint detection and joint angle estimation, while also minimizing the work associated with training data labeling. Related works are discussed, the proposed method is covered in detail, followed by result analysis and conclusion.

Chapter 5 concludes the thesis. Results are summarized and limitations of the present work are listed. Future work to address these and improve performance are then provided.

Chapter 2

Hardware and Software

This chapter discusses the hardware used for the research, including cameras, robots, computers, and lab setup. Everything is connected together with ROS (Robot Operating System) [55], an open-source software platform running on Linux which provides connectivity between hardware and software modules. ROS and other software will be discussed in more detail in the next chapter. Following this, a literature review of various software tools and methods for tasks carried out in this thesis will be provided.

2.1 Baxter robot

The Baxter is an industrial robot developed by Rethink Robotics Inc. It has two arms and one screen serving as the "face" for the user interface. Each arm has 7 revolute joints and one removable gripper as the end-effector. There are two types of grippers offered by Rethink Robotics: an electrical gripper powered from onboard the Baxter, and a pneumatic gripper which requires an external air supply. We use electrical grippers in our setup. The arms are outfitted with encoders providing joint angle and joint angular velocity feedback to the robot's onboard computer, which runs Gentoo Linux and ROS.

In addition to grippers, there are also other sensors on each hand link. An Inertial Measurement Unit (IMU) measures the 3-axis linear acceleration and angular velocity of the hand. An infrared (IR) range sensor measures distance to obstacles in front of the hand. Cameras on each hand provide a video stream at 640×400 resolution with a 30 Hz frame rate.

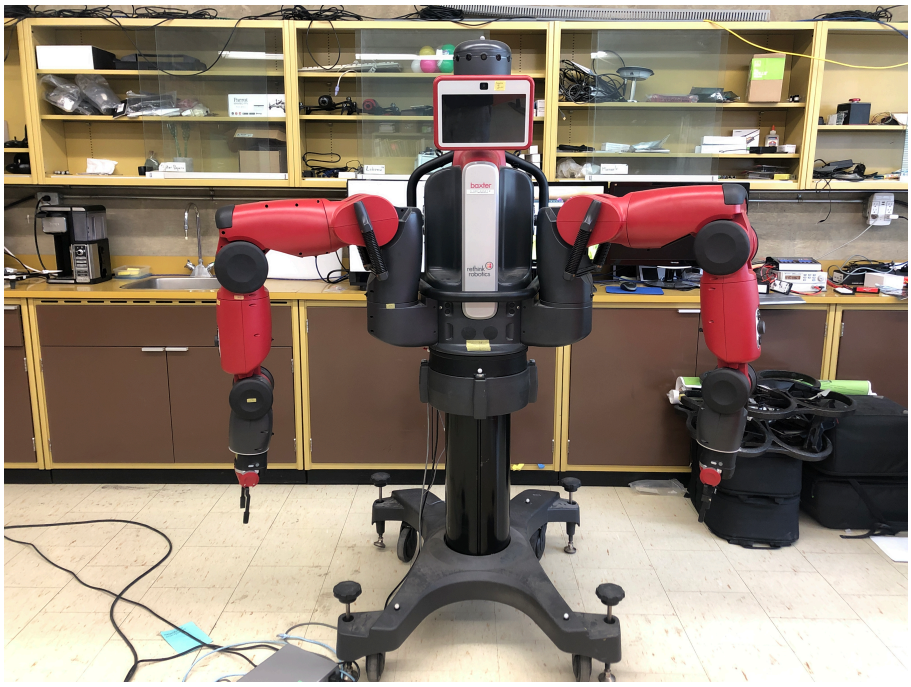

Figure 2.1: Baxter robot in neutral arm position with natural complex background in lab environment.

2.1.1 ROS Integration

The computer inside the Baxter connects over wired Ethernet to a router which has a laptop connected to it, and this laptop is running as one of the slave devices while Baxter computer is the master device. The robot state information is sent out from the onboard computer, and commands for manipulating the robot are sent out from the laptop. ROS provides services for logging data and sending commands over the network connection.

The joint angle and velocity data are sent on the topic `joint_state_publisher` using `JointStates` messages. The 3D location (x, y, z) and 3D orientation as quaternion (x, y, z, w) of each joint are published on the topic `tf` (transform). The robot also publishes its hand camera video feed and IMU measurements, although this data is not used for our purposes.

2.1.2 Forward Kinematics

Forward kinematics computes the end-effector pose relative to the base link of the robot. We employ the Product of Exponentials (PoE) formulation of

forward kinematics [48]. This can be used to compute the pose of either the end-effector or any intermediate link on the robot, since the end-effector pose is accumulated from the proximal to the distal link. To compute the 3D coordinates $(X_{n+1}, Y_{n+1}, Z_{n+1})$ of the $(n+1)^{\text{th}}$ joint, we specialize the PoE formula as

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \\ Z_{n+1} \\ 1 \end{bmatrix} = e^{X(\hat{\xi}_1)\theta_1} \cdots e^{X(\hat{\xi}_n)\theta_n} T^0_{b(n+1)} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.1}$$

where $\hat{\xi}_n$ is the twist axis of the n^{th} joint and $T^0_{b(n+1)} \in SE(3)$ is the pose of the link carrying the $(n+1)^{\text{th}}$ joint at the zero configuration $\theta_1 = \cdots = \theta_n = 0$ with respect to the base link frame. The twist axes and zero pose can be obtained from the Unified Robot Description Format (URDF) file describing the geometry of the robot. Note for $n = 0$, $[X_1, Y_1, Z_1, 1]^T = T^0_{b1}[0, 0, 0, 1]^T$ is a constant, because the position of the first joint is fixed relative to the base link frame.

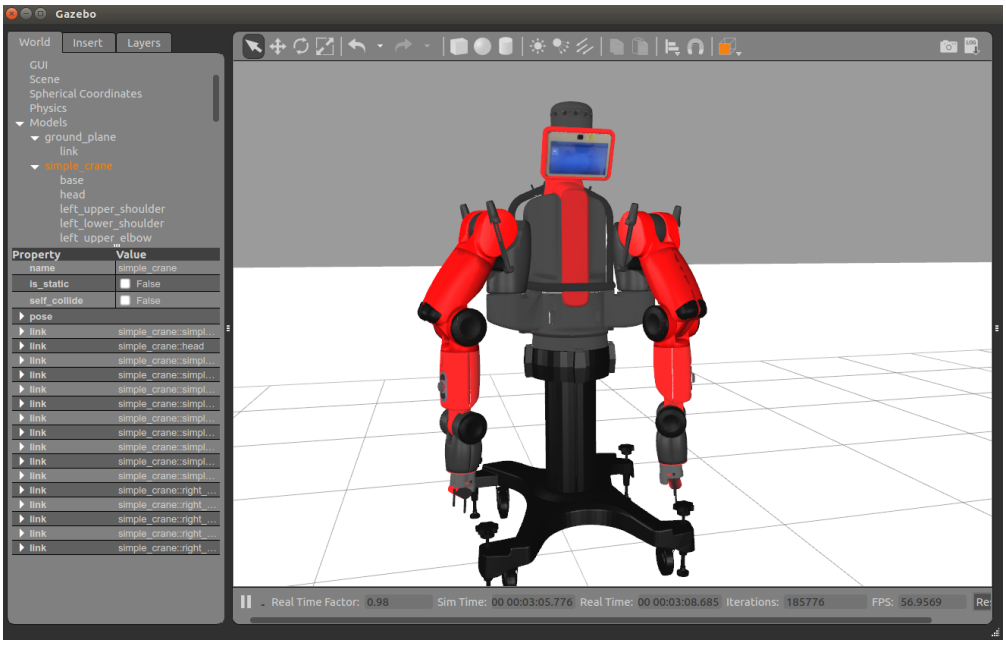

Figure 2.2: Simulated Baxter robot in Gazebo simulator, with white ground texture and grey sky texture. The arm poses are controlled through ROS commands.

2.1.3 Simulation Model

The URDF and 3D mesh files (in dae format) of the Baxter robot are used to render the Baxter robot in a ROS simulation environment such as Gazebo [33], iGibson [73] or AirSim [61] (based on Unreal engine). We chose Gazebo as the simulation environment due to its straightforward integration with the ROS environment.

Simulation is important in our project because it renders realistic images of the Baxter and provides ground truth values for joint angles, 3D positions, as well as intrinsic and extrinsic matrices of the virtual camera(s) used to capture 2D images. The simulation environment was used at two stages in the project. First, a joint detection convolutional neural network was trained using datasets generated with this tool. The simulation environment also played a key role in the toolchains proposed in Chapter 4.

2.2 RGB-D Camera

Figure 2.3: RealSense D415 Depth camera consisting of an RGB camera, an infrared projector and two infrared cameras. Slots on the top of aluminum shell allow airflow for cooling.

The Intel RealSense D415 RGB-D camera was the primary camera used throughout this thesis. It captures both monocular 2D images (RGB) and coloured 3D point clouds (RGB-D), each at a resolution of 1280×720, with a 30 Hz frame rate. The depth sensing technology is active stereoscopic, with

an operational range of 0.16 to 10 meters. RGB-D camera is used in one of the methods in Chapter 3, while all the other methods discussed in Chapter 3 and 4 use RGB cameras.

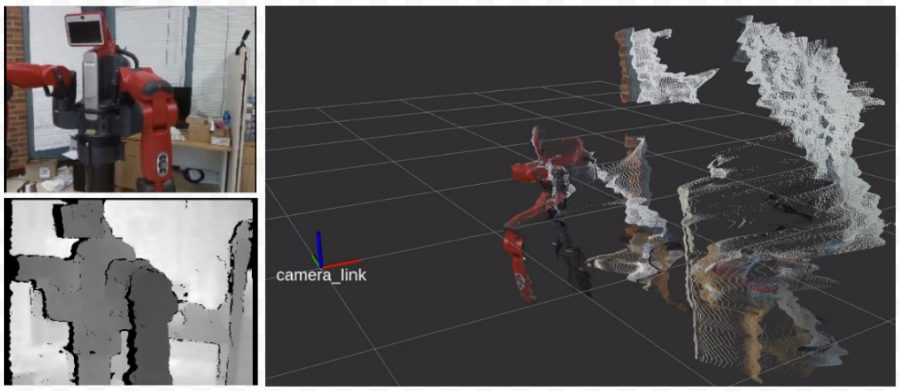

Figure 2.4: RealSense D415 camera RGB-D image visualization. Top left image is RGB image, bottom left image is depth image, and the image on the right is a combination of RGB color and depth on all pixels. The 3D location of each pixel is relative to the camera lens coordinate, denoted as "camera_link" in the image.

2.2.1 Camera Projection Model

A digital camera employs an imaging sensor mounted behind a transparent lens. The lens is usually arced in shape, and light rays from objects in front of the lens travel through the lens while being refracted, then hit the imaging sensor plane. This is illustrated in Figure 2.5.

The model represented above is realistic but overly complex for most computer vision tasks. For this reason, the simplified pinhole model shown in Figure 2.6 is more commonly used. The figure on the right is the real-world object, while the smaller upside down figure on the left is the projection onto the imaging plane. The real object is located at a distance of Z meters in front of the camera lens, and has a height of Y meters. Its captured image has a height of y in the imaging plane, and the distance between the image plane and the lens is f, known as the focal length of the camera. Note only the Y-Z plane is shown to make the figure clear and easy to understand, but the same effects occur in the X-Z (out of page) plane.

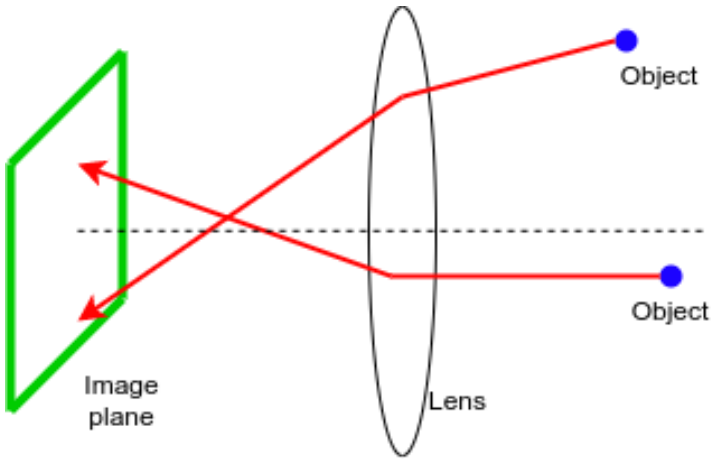

Figure 2.5: Example of lens projection with two sample objects (blue dots). The lens represents the physical arc shaped glass lens in the camera. The red lines illustrate the light rays of each object. They are refracted by the lens and land on the image plane (imaging sensor chip).

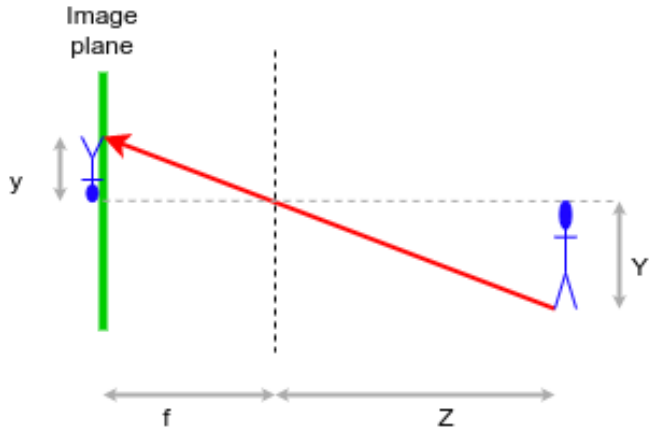

Figure 2.6: Illustration of pinhole projection model. The blue person on the right is the object in front of camera, and the blue person upside down on the left is the imaginary object being captured on the image plane. The lens in this model is simplified as a pinhole (dotted vertical line) and the ratio between Y and Z is same as the ratio between f and y.

An equivalent representation of the model in Figure 2.6 is shown in Figure 2.7. Here the image plane is placed at a distance f *in front* of the lens, such that the image is rightside-up. The geometry relation for this equivalent pinhole model is

$$\frac{f}{Z} = \frac{y}{Y} = \frac{x}{X} \tag{2.2}$$

The X, Y, Z terms in the above equation are the point coordinates of the real

object with respect to a camera lens-fixed frame. This frame is located at the center of the camera lens, with the z-axis pointing outwards from the lens, the x-axis pointing to the right and the y-axis pointing down.

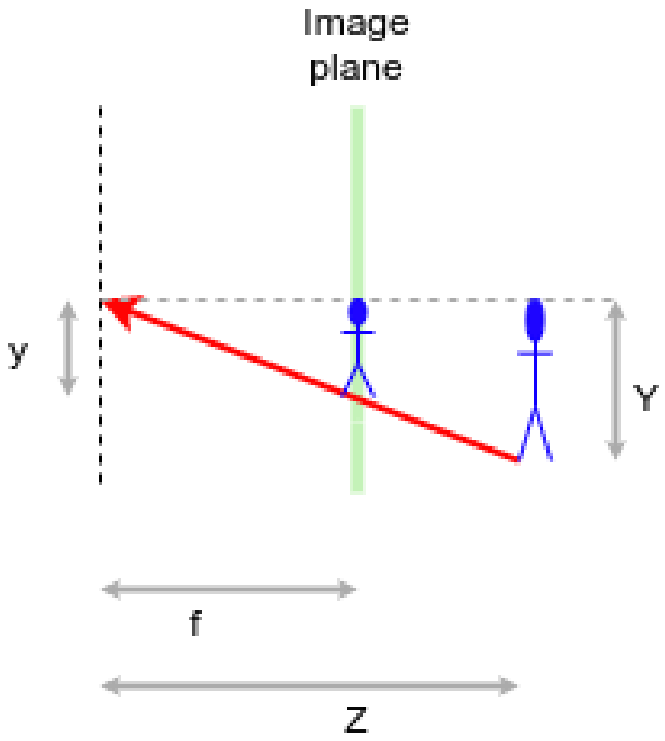

Figure 2.7: Illustration of pinhole projection model (version 2). This model is similar to the model presented in Figure 2.6, and the difference in this one is that both the real object and imaginary object are placed on the same side of lens for explaining the geometry relation more intuitively.

The complete mathematical model for the pinhole camera which projects 3D points (X, Y, Z) to 2D image plane pixels (u, v) is

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

where f_x and f_y are the focal length f multiplied by scaling factors s_x and s_y in units of pixels/m. If pixels are exactly square then $s_x = s_y$ and $f_x = f_y$, but they may not be identical due to manufacturing imperfections. (c_x, c_y) represent the coordinates of the optical center of the lens in image frame

coordinates. They are usually close, but not exactly equal to, half the image frame resolution. The matrix Π_0 contains the extrinsic rotation matrix of the world reference frame relative to the camera-fixed frame. If the camera-fixed frame is used as the world frame, this matrix is as given in (2.3), but in general the world frame may be different (e.g. camera-in-hand applications).

2.2.2 Camera Calibration

The intrinsic camera matrix K models the optics of the camera, and is usually obtained by a camera calibration. In addition to the intrinsic matrix K, the calibration process also estimates the distortion coefficients $[k_1, k_2, k_3, p_1, p_2]$. The coefficients $[k_1, k_2, k_3]$ model radial distortion created by the curvature of the lens, resulting in straight lines in the real world being curved inwards or outwards from the optical center in the captured image. The coefficients $[p_1, p_2]$ model tangential distortion created by the lens not being perfectly aligned with the imaging plane, causing straight lines to be curved laterally outwards from the optical center. The relation between the locations of point in distorted image (x_{dist}, y_{dist}) and undistorted image (x, y) is expressed as follow equation [8]:

$$\begin{aligned} r^2 &= x_d^2 + y_d^2 \\ x_{\text{dist}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{\text{dist}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned} \tag{2.4}$$

We use a 10 by 10 square chess board (Figure 2.8), with a cell size of 10 cm for calibrating the camera. The full calibration toolchain is available within OpenCV. A total of 50 images of the board is used to calculate the distortion coefficients. These images include various poses of the chessboard, obtained by rotating the chessboard around its three axes and placing it at various distances from the camera. The sample image in Figure 2.9 shows the corners detected by OpenCV.

The 2D RGB imaging module in Realsense D415 camera can output resolution of 1920×1080. At this resolution, D415 camera was found to have the

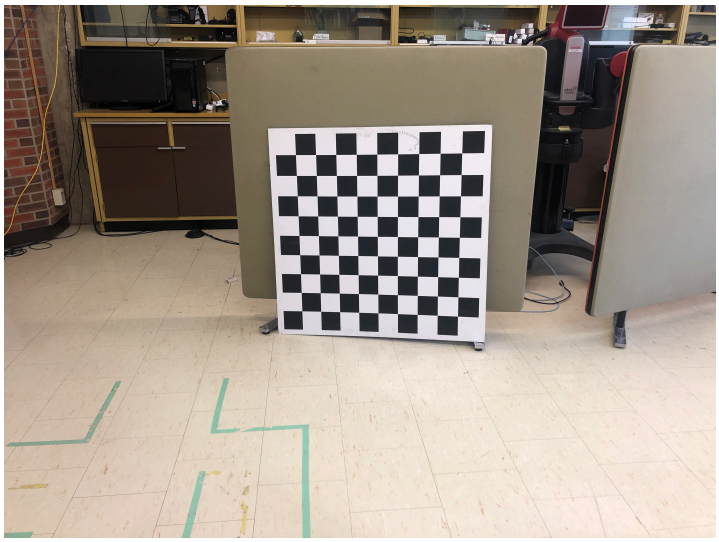

Figure 2.8: Chess board used for camera calibration. It is printed on a rigid flat board, and each cell is 10 cm by 10 cm square.

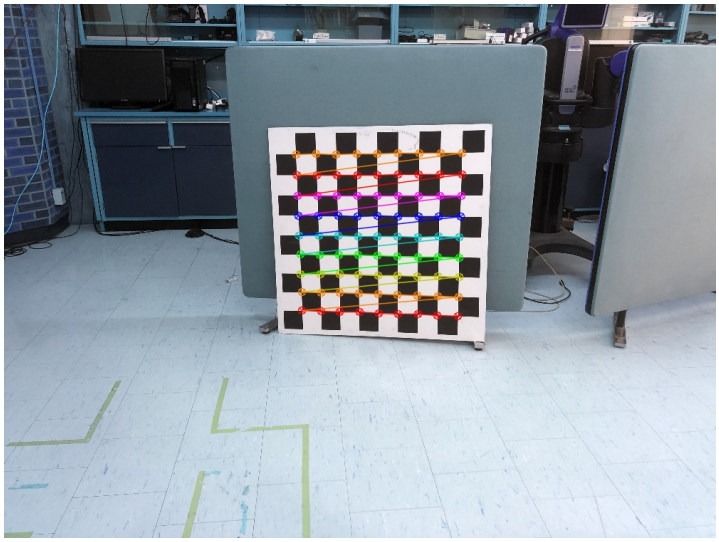

Figure 2.9: Chess board with detected corners labelled by OpenCV. The overall brightness of the original image in Figure 2.8 is reduced for better visualization of the color labels.

intrinsic camera matrix

$$K = \begin{bmatrix} 1408.861118 & 0.000000 & 967.112103 \\ 0.000000 & 1405.073465 & 549.061123 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix} \tag{2.5}$$

and the distortion coefficients

$$\begin{bmatrix} k_1 & k_2 & k_3 & p_1 & p_2 \end{bmatrix} = \begin{bmatrix} 0.090983 & 0.154693 & 0.005158 & 0.007692 & 0.000000 \end{bmatrix} \tag{2.6}$$

2.2.3 ROS Integration

Intel provides a software development kit (SDK) for its RealSense cameras for a variety of software platforms, including ROS. The SDK publishes point clouds and monocular images at 30 Hz, and we set the image resolution at 640×480 to provide smooth video and reduce bandwidth requirements. While the camera supports higher resolutions, these are not required by the neural networks as explained in the upcoming chapters.

2.3 Vicon Motion Capture System

A Vicon optical motion capture system is installed in the lab, consisting of 10 Vero 2.2 cameras plus a stand-alone computer running Vicon's Tracker software. Using strobed infra-red (IR) light, the system tracks the 3D position of reflective markers in the capture volume. In order to track the pose of a rigid body, several reflective markers are fixed to the surface of an object in an asymmetric pattern. The set of markers is used within the software to define a rigid-body, whose pose is then calculated with respect to a user-defined world frame within the lab. The cameras and markers are shown in Figure 2.10. A screenshot of the running Tracker software is shown in Figure 2.11.

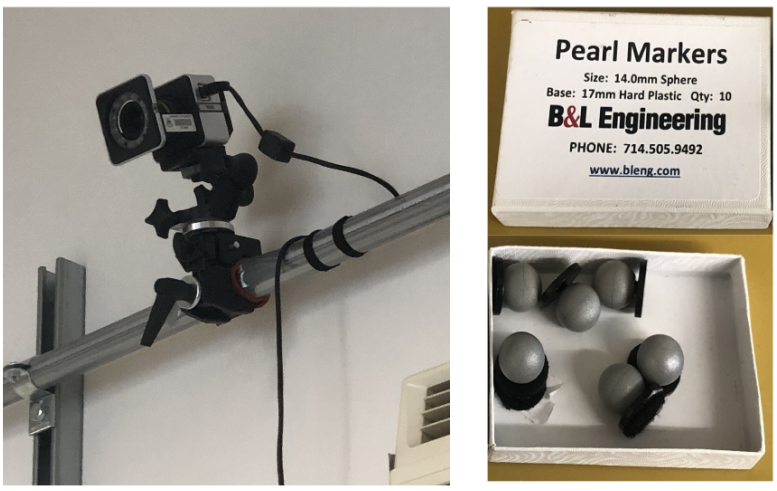

Figure 2.10: Vicon Vera cameras (left) and Vicon markers (right). Vicon cameras are mounted on the metal hanger installed near the ceiling of the lab. The Vicon marker balls are attached to the black plastic plate.

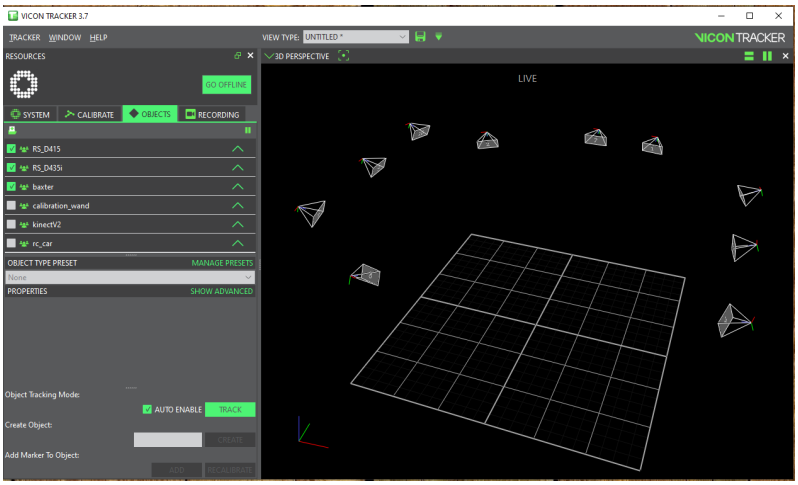

Figure 2.11: Vicon tracker software screenshot. The list on the left are the objects being tracked. The right part is the 3D view of the capture volume. The square grids represent the floor, and each pyramid represents one Vicon camera.

2.3.1 ROS Integration

The `vicon_bridge` package is used to stream the output of Vicon's Tracker software into ROS. The result is published on the `tf` topic at a rate of 100 Hz.

In order to measure the pose of the Baxter's base link frame, six reflective markers are attached to various spots on the robot as shown in Figure 2.12. The marker pattern is placed asymmetrically on the Baxter in order to achieve more robust rigid-body detection. The location and orientation of the Baxter's body-fixed frame assigned by Tracker does not match with the `base_link` frame defined within Baxter's URDF file. Thus the former is manually adjusted in software to match with the latter.

The D415 camera and stand are also outfitted with reflective markers as shown in Figure 2.13. Both D415 and D435 cameras are shown in the figure because we tested both cameras in our experiments. As discussed above, the camera model assumes the reference frame is placed at the center of the lens, thus the body-fixed frame of the rigid body defined within Tracker is manually configured to achieve this.

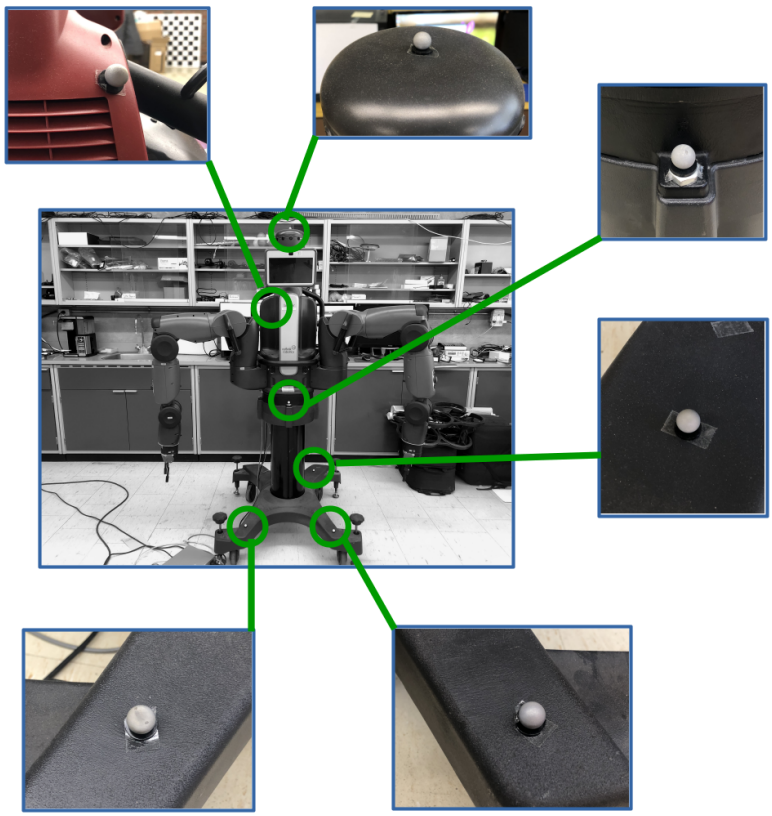

Figure 2.12: Baxter robot with Vicon markers attached around the body. In total six markers are used for tracking the pose of the Baxter body, and each marker is shown in the sub-images around the complete Baxter image.

2.4 Literature review

This Section outlines the background references related to the various software tools which will be used for subsequent work in this thesis. A more focused and specialized literature review will be provided in Chapter 4, in the context of our chosen method for tracking the joint states of the lab robot.

2.4.1 Computer Vision and Machine Learning

Computer vision has several applications, including 3D reconstruction and image processing. All of these can be performed using conventional techniques, but learning-based convolutional neural networks have shown superior performance in the latter two areas in recent research works.

3D reconstruction can be achieved by either RGB-D cameras or (con-

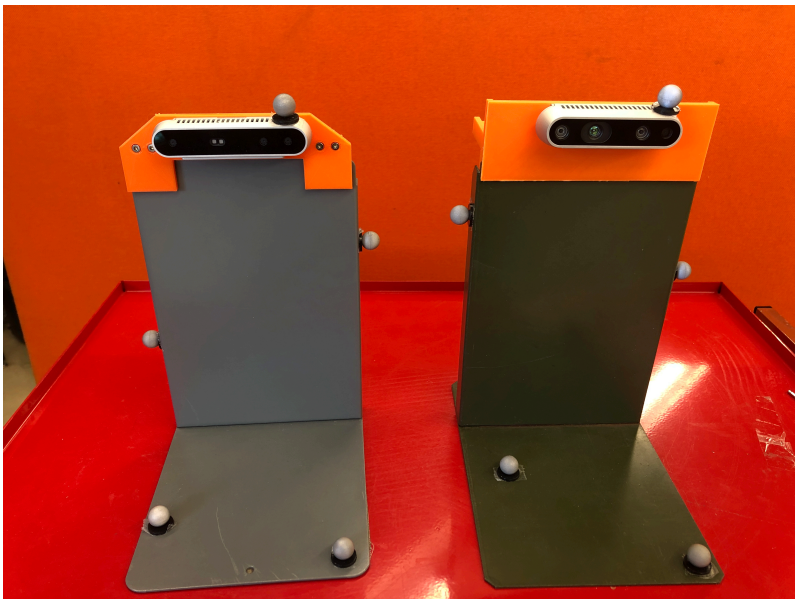

Figure 2.13: RealSense D415 (left) and D435 (right) cameras and holders with Vicon markers. Each camera has five Vicon markers attached on the metal base frame and camera shell.

ventional) monocular cameras. RGB-D cameras capture 2D color images alongside a depth image describing the depth of each pixel, while monocular cameras capture only 2D images. Using RGB-D cameras, TSDF (Truncated Signed Distance Function) [17] is a method for 3D reconstruction of objects, while RTAB-Map (Real-Time Appearance-Based Mapping) [35], [36] also reconstructs 3D scenes but is optimized for SLAM (Simultaneous Localization And Mapping) applications. Structure From Motion (SfM) is a method to reconstruct 3D models from monocular camera 2D images [60], [67], [71]. Orb-Slam [46], [47] compute the 3D pose of a camera using its monocular images. Orb-Slam runs in real time by tracking features points rather than complete textures, while SfM usually runs offline due to its estimation of depth images used to build a complete 3D mesh of the scene. In our project, we evaluated several 3D reconstruction methods including both RGB-D and monocular camera-based, and chose TSDF because of its fast computation and reconstruction quality.

Object detection is an important application of computer vision, for instance detecting all instances of humans in an image and outputting their

coordinates in terms of a rectangular bounding box. YOLO [56] is an object detection convolutional neural network which is capable of detecting multiple types of objects in high-resolution images in real-time. The fast performance is facilitated by using a GPU, whose hardware keeps achieving higher computational power every year. A sub-domain of object detection is instance segmentation using regional convolutional networks [26], [57]. Instead of rectangular bounding box, instance segmentation outputs the actual boundary of the object(s) of interest. This is used to remove image backgrounds, for instance.

While certain image processing tasks can be performed using conventional computer vision methods (e.g. color correction and contrast modification), but many other tasks can only be achieved by convolutional neural networks (e.g. texture style modification). A Generative Adversarial Network (GAN) [22] is a type of neural network which can manipulate content or change texture style in images [62], [77]. In our projects, CycleGan [77] is used for transferring robot images between simulation style and real-world style for robot joint detection and state estimation.

An active area of research is the application of machine learning techniques to engineering problems which were not previously solvable using conventional computer vision methods. For example, conventional object detection needs to rely on shape, color and size, while neural network-based methods can distinguish between objects and classify previously unseen items, for instance. Machine learning also enables synthesizing closed-loop control designs which are too complex for conventional methods. An early example of this is [14], which implemented a model helicopter controller trained using a human pilot's inputs. The resulting design was able to autonomously perform 3D acrobatics and auto-rotations, which had not been achieved previously using conventional control design.

2.4.2 Robot Joint State Estimation

Conventional sensors to measure joint states of robot arms are encoders, either rotary or linear [58]. Hydraulic cylinders, typically used to actuate heavy

machinery such as construction equipment, can be equipped with position feedback sensors such as LVDT (Linear Variable Differential Transformers) or other technologies [30]. Two vision-based approaches to measuring robot arm joint states are 2D RGB image-based methods, based on fiducial markers or trained Convolutional Neural Networks (CNNs), and 3D sensor-based methods, based on RGB-D cameras or LiDAR. In the former, fiducial markers [21] provide good performance in nominal conditions due to their easy-to-spot nature, but may fail in the case of motion blur, unfocused cameras, or occluded or dirty markers. CNNs are a promising approach to this problem, and research works have shown them capable of delivering excellent performance and robustness to effects such as motion blur and dynamic environments. RGB-D cameras are capable of directly measuring depth and thus avoid scale ambiguity, but typically have a limited sensing range and may not work reliably in outdoor scenes. LiDAR works in both indoor and outdoor environments, but good-quality units are much more expensive than the previous sensing technologies. In our work, after evaluating both fiducial markers and RGB-D cameras, we settled on the CNN approach.

2.4.3 Human Skeleton Tracking from 2D images

A number of recent research works have focused on human skeleton tracking from 2D images, which all require a large set of training images annotated with joint locations, either by manual labeling [3] or obtained from specialized equipment [69]. Manual labelling is typically the best approach since it can be applied to a wide range of human subjects and can be done on both indoor and outdoor images. The CNNs developed for human pose estimation have several variations. Early work focused on detecting a bounding box for individual limbs of the human body [75]. Skeleton joint detection was then found to provide better performance, for instance Toshev [65] and Carreira [10] employed an iterative regression method to detect joints. Meanwhile, one-pass joint detectors suitable for real-time skeleton detection were demonstrated in [9], [11], [74]. These methods produce a set of heat maps, one for each joint of interest, indicating the statistical likelihood of the joint's location. Mask R-CNN [26],

a neural network designed for instance segmentation in images, can also be adapted to perform human skeleton tracking.

2.4.4 Robot Arm Pose Estimation

The robot being observed is an articulated body consisting of multiple links and joints rather than single body objects, the problems get more difficulty. But similar to single body object pose estimation, the pose of articulated objects can be treated as a collection of the poses of each single link or joint.

For relatively simple open kinematic chains, namely less than 3 or 4 links, the configuration of the arm can be obtained by estimating the pose of the end-effector and then using inverse kinematics to calculate the joint states. However, this approach will not work for fully- or over-actuated robot manipulators, such as the Baxter's twin 7 DoF arms, where IK has multiple solutions. In this case individual joint angle detection is necessary. One notable example of this approach is DART (Dense Articulated Real-Time Tracking) [59], which employ RGB-D images and a 3D CAD (Computer-aided design) geometry of the robot to track link poses in real time. This method is based on optimization using a signed distance function to minimize the error between model and RGB-D point cloud. The limitation of this method is the reliance on a depth camera, which as discussed before provides a limited depth range.

Two recent works addressing the domain gap problem for robotic arms using domain adaptation are CRAVES (Controlling Robotic Arm with a Vision-based Economic System) [78] and DREAM (Deep Robot-to-camera Extrinsics for Articulated Manipulators) [39]. In CRAVES, 17 distinguishable feature points on the surface of a low-cost robot arm manipulator are selected as keypoints to train a 2-stack hourglass network [50], and an optimization is used for regression of the joint angles to match the detected keypoints. In DREAM, the individual joints of the Baxter robot are used as keypoints, and the detection model is trained using a CNN with VGG-19 [63] as encoder followed by a customized decoder. In these methods, the key point detection networks are trained on simulation images, and tested on real images.

Chapter 3

Preliminary Approaches to Manipulator Joint State Estimation

This chapter documents our early approaches to vision-based joint angle estimation of an encoderless multi-link manipulator. One approach is to estimate the poses of individual links, then computing their relative angles using the known robot geometry.

For each method, we discuss the background knowledge, software and hardware setup, and results assessed with either quantitative and qualitative metrics. As will be shown, none of the methods in this chapter provided acceptable performance. The next chapter focuses on a method which was found to work and provide good performance for estimating the joint angles.

As mentioned in Chapter 1, the original motivation for this research was the automation of an outdoor loading crane. Since we did not have ready access to this system nor a source of ground truth, we performed all our experiments on the Baxter robot in our research lab. The outdoor crane has 4 joints, while the Baxter robot has 7 joints on each of its two arms. By moving only 4 of these joints, the Baxter robot arm was made to approximate the structure of the crane. Throughout this work, we used both a simulation environment and the physical robot hardware to test the different approaches.

3.1 AR Markers

3.1.1 Background

The first approach we tried was using Augmented Reality (AR) markers, shown in Figure 3.1. AR markers are widely used in both research environments and industrial applications, thanks to their robust performance and affordability. In order to estimate the 3D pose of an object relative to a camera, we only need to print an AR marker with a known size and pattern, affix the marker to the object, and use the video feed from a regular monocular camera to calculate the pose.

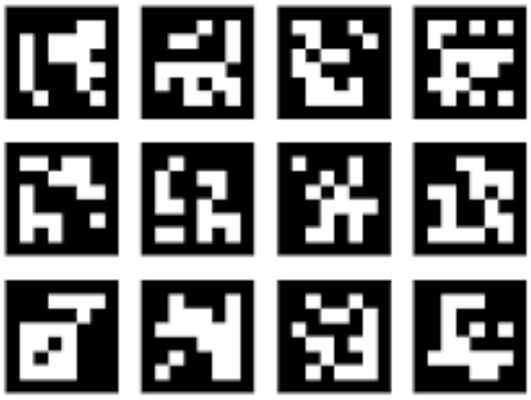

Figure 3.1: Examples of Aruco AR markers. These are 6 by 6 markers, meaning each marker has 6 rows and 6 columns cells. Other sizes are also available.

AR markers provide reliable pose estimates when they are in complete and clear view of the camera. They are robust to small levels of occlusion (e.g. one corner being covered) or lack of camera focus, but will not work for larger levels.

3.1.2 Algorithm

AR marker pose estimation is executed in three steps. First, the RGB image is transformed into a grayscale image and then thresholded to yield a binary (black and white) image as shown in Figure 3.2. The image portion containing

the marker is extracted then warped into a square by a perspective correction. After this, marker cells are applied to the detected marker region, which produces an n by n array containing 0 or 1 in each cell. Using this array, the marker is identified from a dictionary of all predefined marker types. The pose of the marker is estimated using the perspective correction along with the known dimensions of the printed marker.

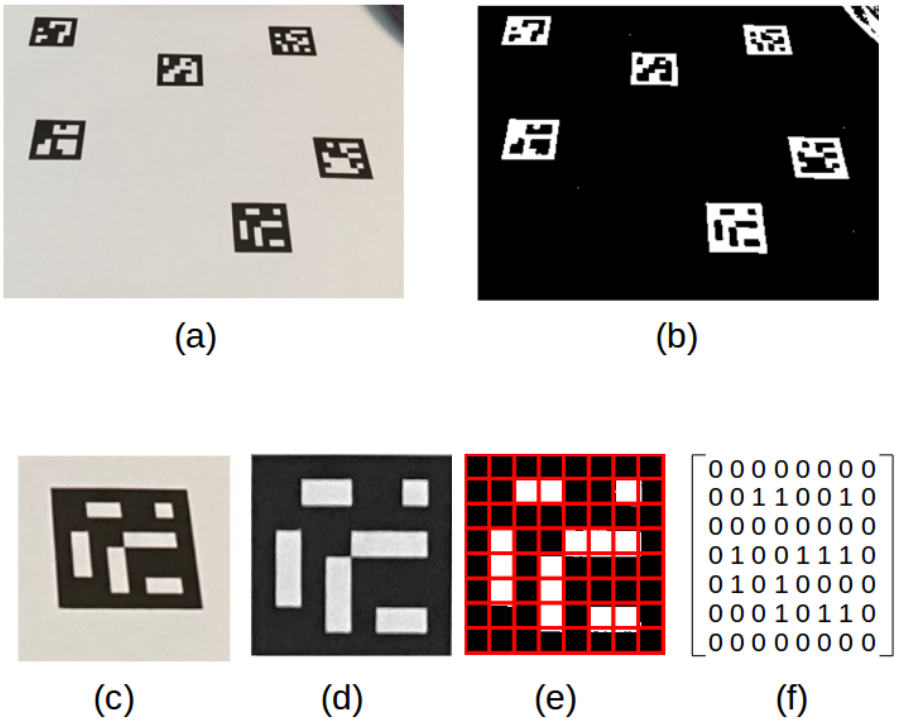

Figure 3.2: Visualization of AR marker detection and identification steps. (a) is the raw image captured by camera containing multiple markers. (b) is the binarized image for contour extraction which will be further used for marker candidate extraction. (c) is one of the extracted markers from the raw image. (d) is the marker after the perspective effect is removed. (e) is segmenting the marker pattern into a grid with a given size, and (f) is the binary array converted from the grid in (e) and it is used for marker index identifying.

3.1.3 Implementation

In order to estimate the joint angles on the robot, we attached separate AR markers to each of its links as well as the base frame. By computing the 3D pose of each link, we can calculate the joint angles between them using the

joint locations on each of the links, which are known.

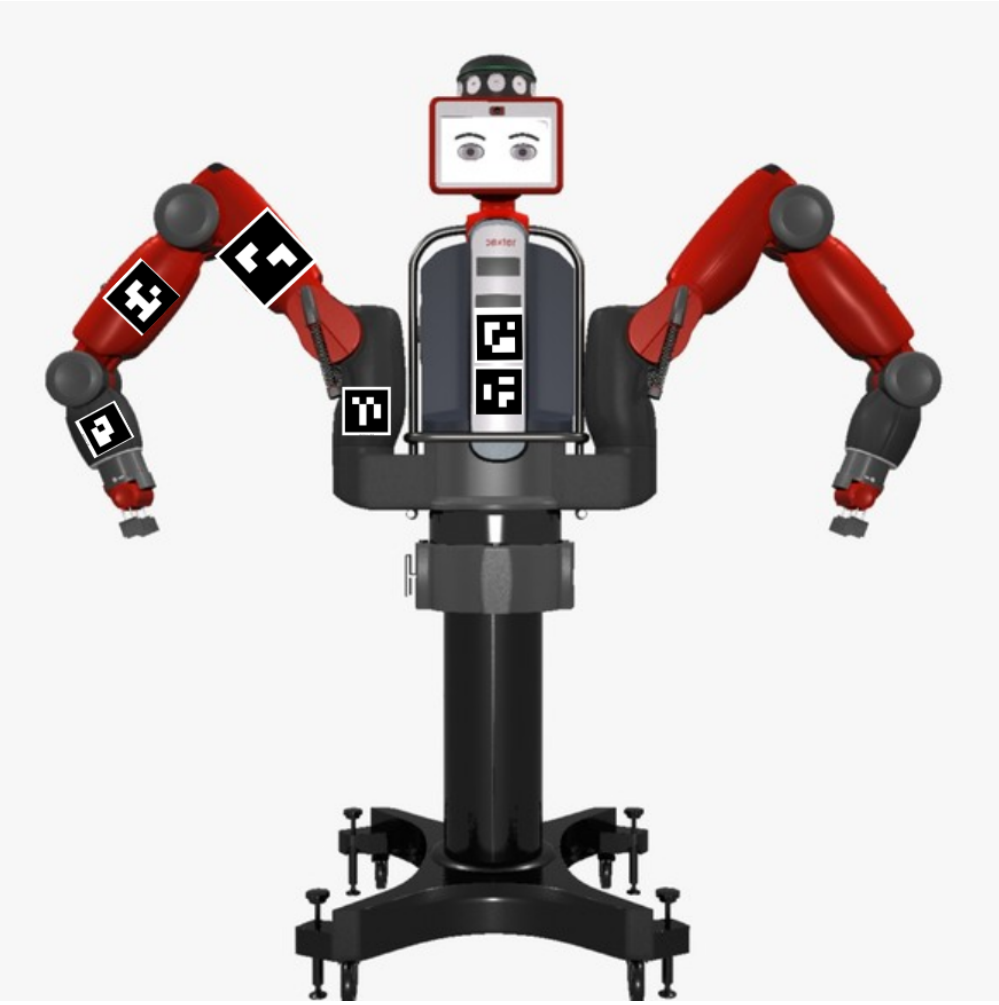

Figure 3.3: Illustration of AR marker placement on Baxter arm. The markers are attached on four links on the robot arm, and also the stationary robot body. In this figure, only left side arm is marked for visualizing the difference between marked and unmarked robot arm, and the actual implementation will applied for both arms if needed.

3.1.4 Test and Outcomes

Hardware testing was performed in our lab, using ROS' `ar_track_alvar` package and a Logitech C920 camera. We printed markers on paper and mounted them on rigid foam boards. For the AR tag detection algorithm, accuracy and speed are optimized when the marker view is large, meaning that using physically larger markers leads to more better estimation performance. At the

same time the markers should not be larger than the dimensions of each link of the robot arm of the Baxter. Thus we printed out markers with 28.9 cm square dimensions for testing. For our first test, the marker was held perpendicular to the camera's lens axis and moved away, starting from 1 meter and ending at 9 meters. Detection was nearly at frame rate over the first 4 meters, but the detection rate started lagging for larger distances. Meanwhile, pose estimation was accurate within the 1–4 m range, but pose errors and detection drop-outs started to be seen at larger distances.

3.1.5 Discussion

While the theoretical idea of AR Marker detection for joint state estimation is valid, things are much more complex in practice. The first issue is occlusions due to the 360 degree rotation mobility of the base joint, meaning all sides of each crane link need separate markers to maintain link detection. The second issue is the clarity of markers in captured images. The sharpness of the markers rendered in the images affects the accuracy of the estimated pose. In order to obtain clear images, the camera needs to focus on the AR markers, but these can be located at various distance from the lens, easily resulting in defocused images. This issue is aggravated when by the camera has a large distance to the robot, which can be expected in field testing conditions. Lastly, environmental issues such as the markers getting fouled by mud or snow or physically damaged by the elements, would also lead to poor detection results. For these reasons, we decided not to pursue the AR marker idea further.

3.2 Color Strips

3.2.1 Background

As explained in the previous Section, AR markers do not perform well at longer distances. We next considered the idea of marking the links with color strips, acting as fiducial markers. Theoretically, 8 unique feature points in a flat pattern are sufficient to estimate 3D pose [44], and if a smaller number is available, pose can be estimated down to a small set of possibilities, which can

be further refined by real-world constraints. Meanwhile color-based markers are much less sensitive to distance and camera focus than AR markers for detection purposes. In our case the markers were rectangular stripes with different colors

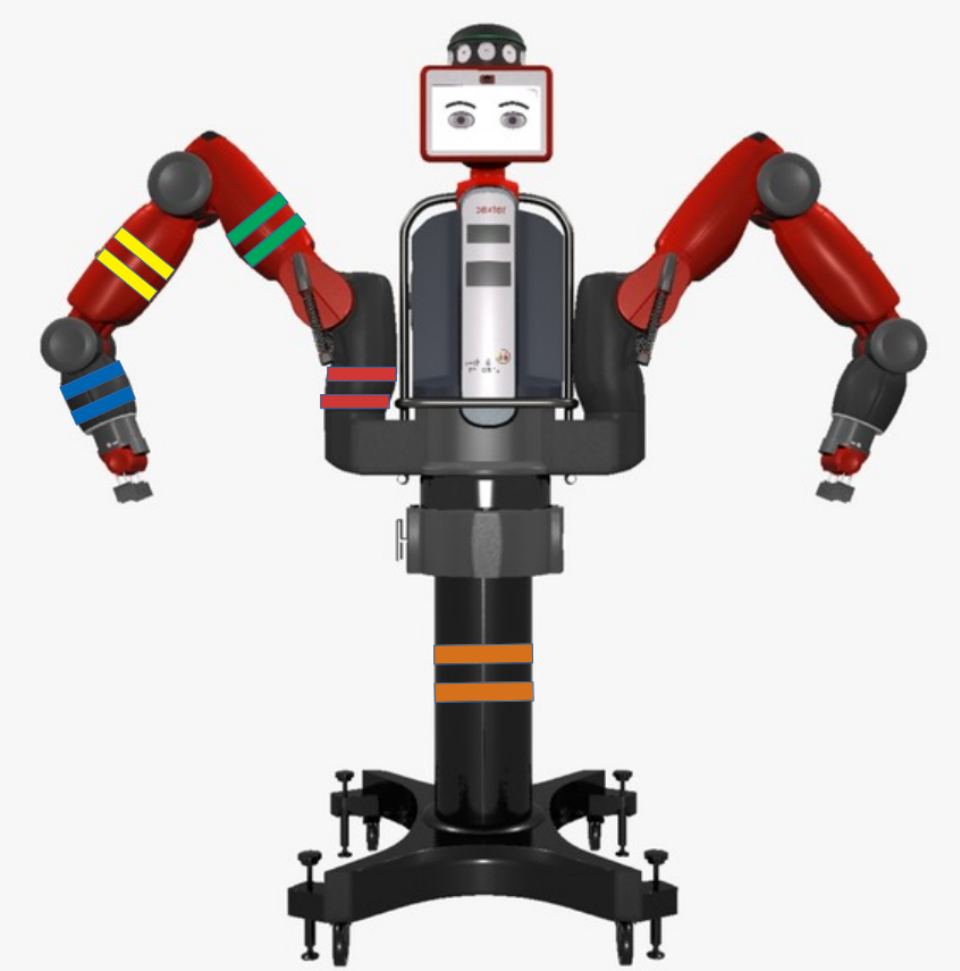

Figure 3.4: Illustration of color strip markers on Baxter arm. The color strips are attached on four links on the robot arm, and also the stationary robot body. In this figure, only left side arm is marked for visualizing the difference between marked and unmarked robot arm, and the actual implementation will applied for both arms if needed.

3.2.2 Test and Outcomes

Two colored strips are wrapped around each link of the manipulator, and each link has a pair of strips with a particular color. When an image of the manipulator is captured by the camera, the image is first transferred from

RGB to HSV (Hue, saturation, brightness) color space, then color filters are used to extract the regions corresponding to each color.

We started testing using a single green rectangle as shown in Figure 3.5. The original image was filtered by a fixed-range HSV color space filter. The resulting image is shown in the middle of Figure 3.5. In this example, the background is clean and of a different color than the object, but this will not always be the case in a real-world environment, where random elements of the scene may also pass through the thresholding filter. The last step of the detection pipeline is corner extraction using the Harris method [25]. In our testing, this corner extraction performed well as shown in Figure 3.5, but let to the problem of the corner detection numbering being random. This could be resolved by using a tracking algorithm for the individual corners. Note corner numbers are shown in the right side of Figure 3.5.

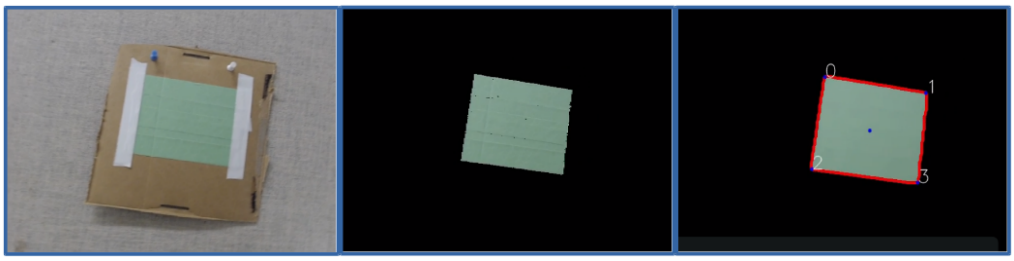

Figure 3.5: Color rectangle detection experiment. Left image is the original image containing the green marker. Middle image is the marker being filtered out. Right image has four edges labelled as red lines, and four corners labelled by numbers.

The second test involved color filtering thin strips attached to the Baxter robot's arm as shown in Figure 3.6 (left side). The middle of this Figure shows the filtered image, obtained from a fixed-range thresholding of the HSV color space. The green strips on the Baxter are correctly extracted, but random background color elements are also filtered out as seen in the right-bottom corner of this image. Meanwhile, the extracted color strips have lost some of their shape near the top-left corner, due to light shadowing in that area. The performance of the filtering was even worse in more complex environments. Thus this method is very sensitive to the trade-off between shape extraction

and noise rejection.

Figure 3.6: Color strip detection on Baxter arm. Left image is the original image containing the Baxter robot arm with green strips. Right image is the green strips being filtered.

As demonstrated above, the color strip method has several weaknesses which will cause it to perform poorly in a real-world operating environment:

1. Each link must have a unique color to distinguish it from the others, and the colors must be sufficiently distinguishable. For a four-link manipulator we can readily find such colors, but this may not be the case for more complicated manipulators.

2. There will always be regions of matching colors in a natural scene background. This will make it more difficult to identify the the true regions of interest, and may result in erroneous corner detections.

3. Just like the AR marker method, the visibility of the color strips is critical for performance. If the strips are fouled, occluded by another object or subject to glare, detection will fail and pose will not be estimated.

4. Within the setup discussed above, eight points (four corners of each color strip) are available for pose estimation, the minimum number required to estimate pose. However if the strip filtering produces a distorted image with less than 4 corners, pose estimation will not return a unique pose.

3.3 3D Model Fitting onto 2D Images

3.3.1 Background

The pose of an object can be estimated if its image has distinguishable vertices (corners) and edges (lines). Lines and corners can be extracted from images through standard computer vision techniques, e.g. Canny edge detection on grayscale images, and if a correspondence can be found between features in the image and features on a pre-defined 3D model of the object, then the pose of the real object can be estimated in real time. This method was proposed in [15].

3.3.2 Software

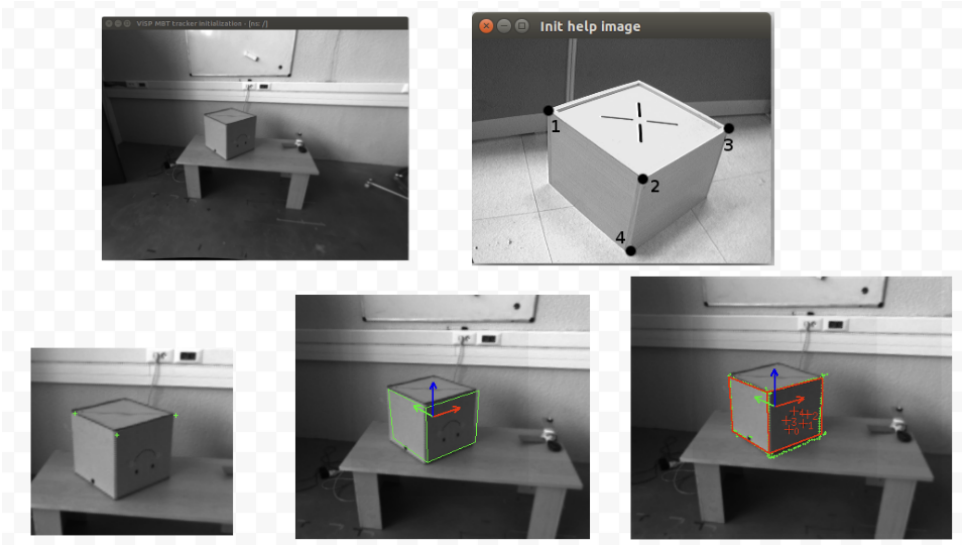

Figure 3.7: Object pose tracking stages. Images in order from top left to bottom right are: camera video stream, feature helper image, feature selection on the first image, initial pose estimation, corrected pose estimation over time.

The stages involved in applying this method to a single rigid-body object are demonstrated in Figure 3.7. The top-left image is the scene seen by the camera, and the top-right image is a helper image of the object to be tracked. This image contains four clearly distinguishable vertices, which are indexed from 1 to 4. To identify the location of these features on the actual object, the user needs to manually click on the locations of these features in the helper

image, as shown in the bottom-left image with the green markers showing the clicked locations. Once these 4 feature points are selected in the helper image, the algorithm fits the detected lines and corners to the camera image to the 3D model to estimate the pose of the object. However, due to the limited resolution of the image and errors in manual feature selection, there are noticeable offsets between the object's identified features, whose edges are shown as green lines in the bottom-center image, and the actual image. The algorithm will further refine the fitting of the 3D model to minimize the error between the projected version and actual image. The result is shown in the bottom-right image as red lines, demonstrating a good fit.

Since this method requires 3D models of the object to be tracked, we focused on the Baxter platform for which geometry information is available as STL (STereoLithography) files, as shown in Figure 3.8 for the arm link. Just as in the previous method, joint angles can be computed from the pose of the two links connected by the joint. Since each robot arm contains seven links, estimating their pose along with the geometry of the manipulator would enable estimating the values of the 7 joint angles.

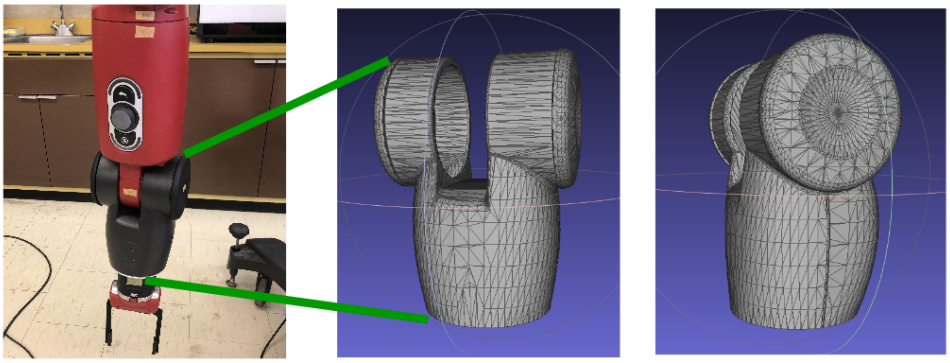

Figure 3.8: 3D geometry model of Baxter arm link. Left image is a picture of the link of the robot arm being track tracked. Two images on the right are the 3D CAD model of this robot arm link (front view and right view).

3.3.3 Test and Outcomes

Testing was performed in our lab, employing ROS' `vision_visp` package and a Logitech C910 webcam. The initial test involved a simple plastic box. The di-

mensions were measured manually and a CAD (Computer-aided design) model of the box was created. Using the procedure explained in the previous Section, real-time object tracking was achieved. The performance was found to be very good, likely due to the box's easily distinguishable edges and corners.

Unfortunately this method did not work for the links on the robot arm. These objects are different in nature from the box in the previous test. The first difference is the lack of distinguishable edges and corners on the smoothly contoured links. The second difference is the severe occlusions created by the presence of other links on the arm. However the primary reason for the the failure of tracking by this method is that the algorithm relies on matching edges in the CAD model with visible edges in the camera images. While in the case of the box, there are 8 vertices and 12 edges to be matched which are clearly visible in the camera images, the Baxter arm link model has 7,379 vertices and 13,754 edges. This detailed model allows a detailed rendering of the Baxter arm link, but almost none of these CAD vertices and edges are clearly visible on the real arm link. As a result the algorithm is unable to extract enough features from images to match the features of the CAD model, even when these are accurately initialized by the user using a helper image.

Another problem we noticed for this package is the necessary computing power. Even when tracking the simple box, the algorithm used nearly full CPU (Central Processing Unit) power on an Intel Core i7-8750H based system. This means tracking complex and multiple objects such as arm links is not likely to work in real-time, although this needs further testing.

3.4 Iterative Closest Point-based registration

3.4.1 Background

While all the other pose estimation methods discussed so far rely on RGB images from a monocular camera, the next method relies on capturing point clouds with an RGB-D camera. When the object is in the field of view of the camera, the captured point cloud contains a portion of the surface of the object, as shown in Figure 3.9. The top-left image is the RGB image captured

by the camera, which provides color information in the 2D plane. The bottom-left image is the depth image captured simultaneously with the RGB image, whose pixel values are 16-bit integers proportional to the physical distance from the image plane to the object. This image is rendered with darker pixels for closer range and brighter pixels for further range. The RGB-D camera used to capture this image was an Intel RealSense D415, which has an upper range limit of about 10 meters. The 3D mesh on the right of Figure 3.9 is the point cloud created by combining RGB and depth information for each pixel, visualized in RVIZ. The 3D location of each voxel (pixel in 3D space) relative to the camera is calculated from the depth image and the camera projection model, and each voxel is colorized using the corresponding pixel in the RGB image.

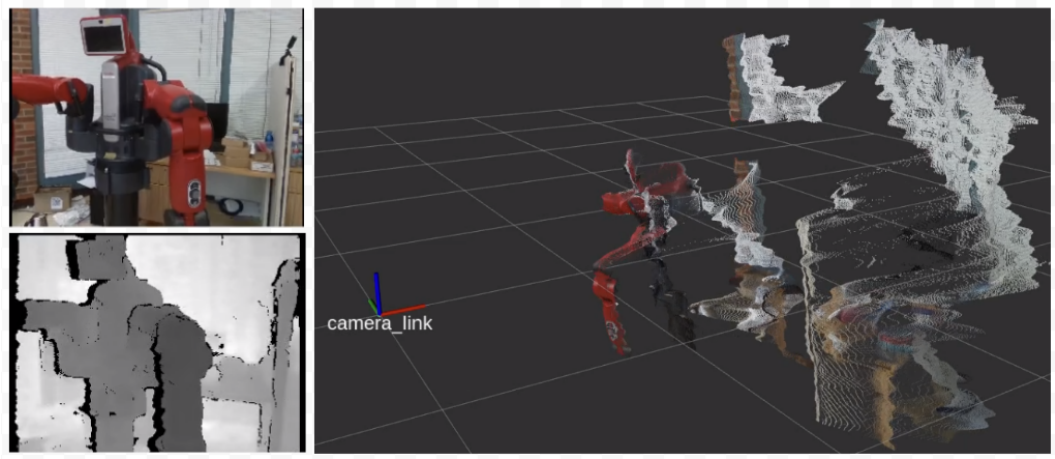

Figure 3.9: RGB-D camera output visualization. Top left image is RGB image, bottom left image is depth image, and the image on right is a combination of RGB color and depth on all pixels. The 3D location of each pixel is relative to the camera lens coordinate, denoted as "camera_link" in the image.

3.4.2 Test and Outcomes

Assuming we have a 3D CAD model of the object of interest, for instance a robot arm link, the pose of this link can be estimated by fitting the 3D model to the captured point cloud. One algorithm for doing this task is ICP (Iterative Closest Point) [7]. We used the implementation provided in Open3D [76], an open-source software package for 3D point cloud manipulation. ICP takes two

point clouds as inputs, one captured by the RGB-D camera, the other from the 3D model (known as the target). The algorithm iteratively adjusts the pose of the target point cloud until the distances between voxels of the captured and target point cloud are minimized. To test the capability of this algorithm, we performed experimentation in two steps. We first tested the method for using a chair in the lab, representing a single rigid body. Each link in the articulated robot arm represents a rigid body which needs to be tracked individually, with the added challenge of occlusions created by other links affecting the tracking accuracy of the system. Thus the single chair experiment works as an idealized experiment to test the best-case performance of the system. Figure 3.10 shows an example of ICP being applied to point clouds of a chair. The blue portion is the target point cloud, while the yellow is the captured point cloud. The algorithm successfully finds the pose required to align the blue point cloud to the yellow point cloud. The result gives the pose of the actual 3D object relative to the RGB-D camera-fixed frame.

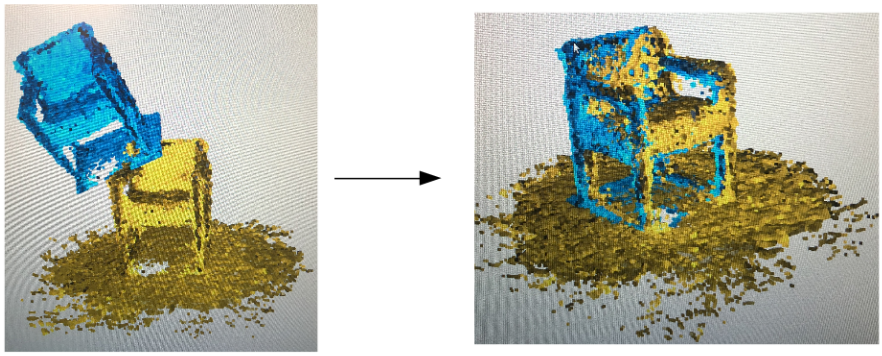

Figure 3.10: ICP experiment on a chair. This experiment was performed for this chair because the chair is a single rigid body, and the result in this experiment is the baseline reference of the ICP approach to multi-link tracking. The yellow model is fixed, and the blue one is being moved. In the first image, the two models are placed at a random pose, serving as the initial guess. The second image shows the result of the pose estimation. Two models are aligned.

The second part of testing involves the articulated robot arm. While ICP performed well in the first trial, it will not reach the same performance for our task of estimating the poses of individual robot arm links. While in the

previous example, the "captured" point cloud (yellow) is actually stitched from different views of the chair from all directions. The run-time RGB-D images will contain only a partial surface of the object, as seen in Figure 3.9. Applying the ICP algorithm in this case will not produce satisfying results in our experiments due to this and the occlusion issues mentioned above.

3.5 DART: Dense Articulated Real-Time Tracking

3.5.1 Background

Since the conventional ICP algorithm did not produce satisfactory results, we tried the related Dense Articulated Real-Time Tracking (DART) method developed by [59]. This method uses SDF (signed distance function) for aligning a 3D model with an RGB-D image of the object.

3.5.2 Discussion

The authors of DART have made the software toolchain for running DART with Baxter publicly available. While testing with our Baxter model yielded good estimation results, there are limitations to this approach preventing it from being used for an outdoor crane. First, the depth range of our RealSense RGB-D camera is 10 m, which is too close to capture a full view of the outdoor crane. Second, depth sensing in the camera is performed by projecting and measuring an IR image ahead of the camera, but this signal is lost in bright sunlight which can be expected in outdoor conditions. One possible alternative is a stereo camera, which works by correlating images from two parallel monocular cameras to estimate depth and generate point clouds. However based on our testing, commercial stereo cameras such as StereoLabs' ZED are unable to generate sufficiently detailed point clouds at longer (> 10 m) distances as well.

Based on our informal experiments, methods relying on depth cameras were dropped from consideration, since they could not function reliably in an outdoor setting, which is the longer-term goal of the work presented in this

thesis.

3.6 Learning numerical joint angles from RGB images

3.6.1 Background

The approaches discussed in the previous sections are based on traditional computer vision methods, which work reliably in certain scenarios but may fail when the image background is too busy or the tracked object is too complex. Inspired by the many real-world successes based on convolutional neural networks (CNNs) in recent years, we decided to try a CNN-based approach to joint angle estimation.

Since our goal is to estimate the joint angles on a robot arm, the input of our network are RGB images, and the output is a set of four joint angles. The image is first processed by convolutional layers whose function is to extract features from the arm, followed by subsequent layers used to calculate a 1D array of joint angles. This idea is illustrated in Figure 3.11.

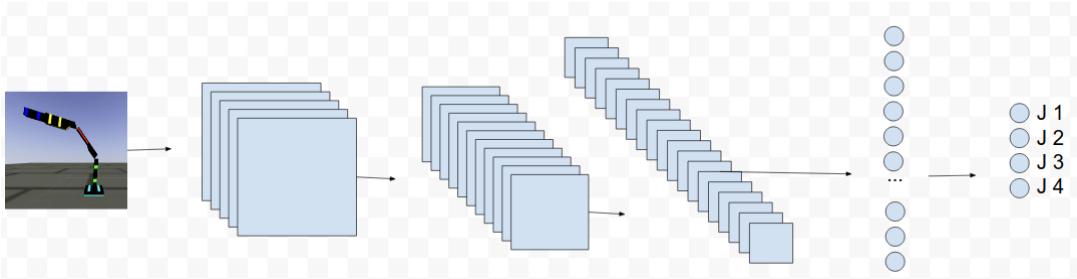

Figure 3.11: CNN architecture for direct joint angle prediction. The image of the crane on the left is the input. The square groups in the middle are the convolutional layers. The sizes of squares show the relative sizes of processed arrays, and numbers of squares show the relative amount of kernels in each stage. The two columns of circles on the right are the fully connected layers, and the four circles labelled with J1 - J4 are the outputs. The sizes and numbers of elements in this Figure are only approximate values to show the shape of the neural network because the figure created with the actual parameters was impractical to visualize.

In this approach, a simulated crane is used instead of the Baxter robot because this neural network solution is end-to-end (from image directly to joint

angles) while the previous approaches consist of sequential steps (e.g. extracting features from image, computing individual link poses, and then computing the joint angles), and the crane has 360 degree base-joint rotation range while the Baxter arm base-joint has only a 200 degree rotation range. The sequential steps require analytical forward/backward kinematics model to compute the joint angles, which brings the benefit that the solutions developed for the Baxter robot could work on the crane as well simply by changing the kinematics model. But since the behavior of this end-to-end network solution is less analytic, the network architecture developed for the Baxter arm would not be representative of the crane. Thus the crane is exclusively considered in this approach. Note that the subsequent methods use sequential steps again, so the Baxter robot is used in these approaches.

In order to train such a network for a real crane, we would need a library of images of the real crane together with the corresponding joint angles. However, since the field crane does not provide joint angle feedback, this approach cannot be directly applied to the real crane. As a result, we built a simulated crane in the Gazebo simulation environment, as shown in Figure 3.12. This simulated crane consists of a stationary base plus 4 moving links. Each link has a pair of strips with a unique color attached to it. It would have been ideal if a 3D CAD model of the crane were provided by the manufacturer, but we were unable to obtain this. Therefore we created the simplified model shown in Figure 3.12 in Blender, using the actual dimensions of the field crane listed in its user manual [1] to increase realism. The geometry of the crane was encoded in a URDF and a configuration file. The simulated crane has four joints, three revolute and one prismatic; note the real crane is designed with a 3-section telescoping section, but this can be modeled as a single prismatic joint. The intrinsic parameters of the simulated monocular camera were set to match those of the Logitech C920 webcam, while the extrinsic parameters (pose) were set to just fit the simulated crane into the image frame throughout its range of motions. Virtual lighting was also added for additional realism.

We used a simple network inspired by VGG [63] to evaluate this method. The network has 5 convolutional sets in sequence, each consisting of two con-

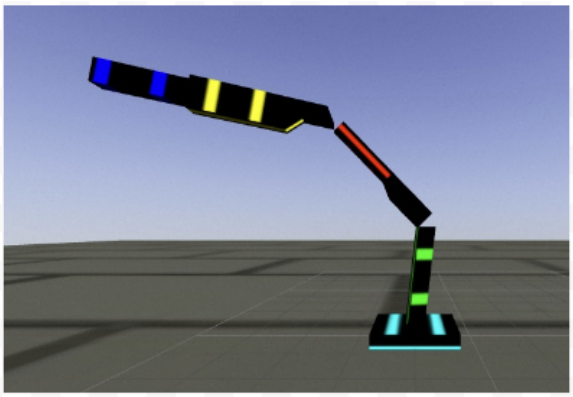

Figure 3.12: Simulated 4 DoF crane in Gazebo. It consists of 5 links, and each link is outfitted with uniquely colored strips. The base link is marked by cyan strips and it is stationary. The next three links are covered by green, red and yellow strips. The green-colored link rotates around the vertical axis while the other two links rotate around horizontal axes. The link with blue strips has a linear rather than a rotational motion. All four joints are controlled by the ROS joint controller.

volutional layers and one max-pooling layer. Then it has two fully connected layers followed by the output layer. The input to this CNN was 256×256 RGB images. While the virtual camera returns 640×480 images, these were downsized to 256×256 to allow training on our consumer-grade GPU. The input images were then processed by 4 convolutional layers with a kernel size of 4 by 4, followed by fully connected layers, outputting for values corresponding to the joint states.

To train this network, training data was generated inside the Gazebo simulation environment. The training set consisted of 3000 images of the simulated crane in various poses, together with the corresponding joint states. Another 2000 pairs of images and joint state data were generated for testing. The virtual camera was kept in a fixed pose throughout. Training was performed using the PyTorch deep learning environment using the Adam optimizer and L2 loss function.

3.6.2 Test and Outcomes

The test results were not as good as expected, but still demonstrated the potential of a CNN-based approach. Following multiple rounds of training

while tuning the hyper-parameters of the network, the trained CNN was only capable of correctly estimating a portion of the 4 joint states. For instance, using the validation dataset, the network would produce two joint estimates within 5 degrees of the ground truth values, but the two remaining estimates would exhibit large errors.

3.6.3 Discussion

Even if the results from this method would work well in simulation, it could not be applied to the real-world crane since we cannot obtain the joint angles in that system, which are required for training the network. However, this approach still proved valuable since it demonstrated the interest of CNN-based methods relative to conventional computer vision methods for joint angle estimation. This motivated the author to pursue convolutional neural networks, which play a key role in the method discussed in the next section, and are the basis of the eventually chosen approach documented in Chapter 4.

3.7 Baxter keypoint detection in Simulation

3.7.1 Background

Inspired by the CNN approach discussed in the previous section as well as reading other research, we found that our project is related to human pose estimation, the latter relying on estimating the positions of joints on a human skeleton without access to direct sensing.

There are several excellent recent works on human body joint detection from monocular RGB images based on CNNs [9], [10], [74]. We thus tried this class of methods in our simulation environment. We chose to use Baxter robot for this portion since the simulation results could be validated on a real Baxter in our lab, unlike the outdoor crane where a detailed 3D model was not available. Even if such a model was available, validation would be difficult since the ground truth of joint states of the full-sized crane is not available to evaluate the accuracy of the state estimates.

As before, the simulation is performed with the Gazebo simulation environment [33], using the Baxter model provided by Rethink Robotics. One fixed monocular camera is placed in front of the Baxter, and several lighting sources are added to the scene in order to enhance the color contrast. During data collection, both arms of the Baxter are commanded to move in random motions throughout its reachable workspace, and the virtual camera acquires images at a rate of 1 Hz. The ground truth joint state data is saved in `json` files corresponding to the captured pictures. The background in the simulation environment is set to be pure white, such that post-collection background augmentation can applied using simple color filters. The background of each image was filled by single images randomly selected from the COCO image dataset [41]. The resulting augmented images along with the logged joint states were used as training data for the network. The output of the network are heatmaps for each detected joint, reflecting the probability distribution of the joint's location within the pixel image.

There are several types of convolutional neural network architectures available, for instance VGG-19 [63], hourglass [50] and ResNet [27]. We chose ResNet because of its ability to handle larger numbers of layers than the other two networks, and its superior accuracy in classification problems [27]. There are several versions of ResNet, including ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152, indicates the number of convolutional layers used by the network. We chose ResNet-50 as a balance between speed and accuracy.

Since ResNet was designed for classification problems, its output layers are fully connected, which is not compatible with the desired joint heatmaps. Thus the fully connected output layers were replaced by transposed convolutional layers which generate larger 2D output arrays from an input of smaller arrays.

The output data are heatmaps built based on the joint location in pixel coordinates. For each keypoint, a 2D Gaussian distribution with certain σ is generated.

3.7.2 Test and Outcomes

After training the network with simulated images, we first applied the network back to the training dataset to evaluate performance. The result was a PCK@0.2 score of 0.99, indicating that 99% of joint detections had errors (Euclidean distance between the detected and ground truth location of joints) less than 20% of the robot's bounding box dimensions. Figure 3.13 shows three sample images, with the detected joints draw as circles with various colors.

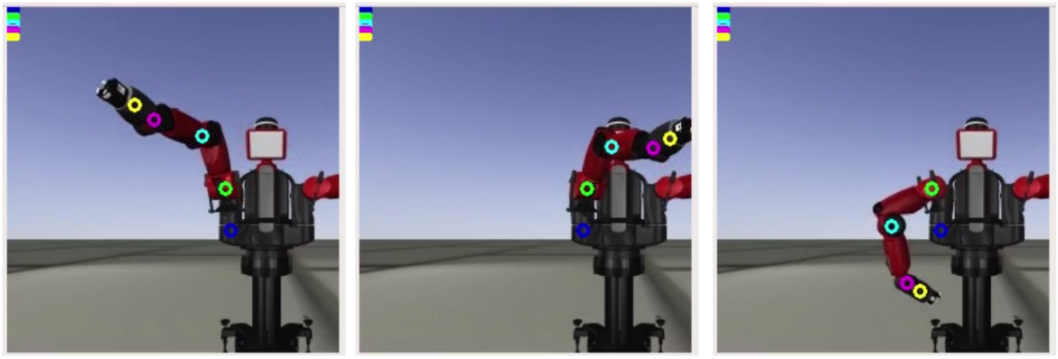

Figure 3.13: Examples of joint keypoint detection for simulated Baxter images. The circles labelled on the robot arm are drawn at locations identified by the joint detection network. In these three examples, the results accurately identify the locations of joints on the robot arm.

Interestingly, applying the trained network to real-world images of the Baxter provided very inaccurate results. This brought up the requirement for domain adaptation, a method of bridging the gap between simulated and real images. In our case, this gap is caused by the texture difference between the simulation images and real-world images. There are several methods to bridge this gap [16], including Domain Randomization and Generative Adversarial Networks. We chose to implement the latter method, as discussed in the next Chapter.

Chapter 4

Image-based joint state estimation pipeline for sensorless manipulators

The contents of this Chapter were submitted as [24]. The method presented was found to provide the best real-world joint state estimation performance as compared to the methods covered in the previous chapter.

4.1 Abstract

Motion planning is a solved problem for robot arms with joint state feedback, but remains an area of research for sensorless manipulators such as toy robot arms and heavy equipment such as excavators and cranes. A promising approach to this problem is deep learning, which employs a pre-trained convolutional neural network to identify manipulator links and estimate joint states from a monocular camera video feed. Whereas manual labeling of training image sets is tedious and non-transferable, a simulation environment can automatically generate labeled training image sets of any size. The issue is the gap between simulated and real-world images. This chapter solves this problem by implementing a Generative Adversarial Network. The complete joint state estimation pipeline is implemented and tested in hardware experiments to validate our proposed approach.

4.2 Introduction

Thanks to the continuing growth in sensing and computing power, robot arm manipulators are now being deployed in high-precision tasks such as picking items off shelves [29], assembly operations in manufacturing [51], [34], and food preparation [12], [19], [49]. A necessary part of these precise operations is motion planning, which relies on forward and inverse kinematics [48], which in turn require knowledge of the robot's geometry and joint states (angles for revolute joints or displacements for prismatic joints). While commercial-quality robot arms such as the Barrett WAM arm, the Franka Emika Panda or the Kinova JACO can measure joint states nearly perfectly, the same cannot be said in the following cases: (i) using inexpensive robot arms whose joint encoders are subject to poor resolution and backlash issues and (ii) adapting human-operated manipulators without joint feedback, such as knuckle-boom cranes, for autonomous operations. In either of these cases, using computer vision to estimate joint angles and/or end-effector pose is an appealing solution, as shown in the recent works [78], [39], [43], [28], [40].

In this chapter, we focus on joint state estimation in robot arms using images from a monocular camera, a sensor which is much cheaper and provides a longer working range than depth-sensing technologies such as stereo vision, RGB-D or LiDAR (Light Detection and Ranging). Our approach has close ties to human skeleton tracking, which was popularized around 2010 by the mass-produced Kinect v1 RGB-D camera which used this feature for video games [2], and which today is offered as an SDK for Intel's RealSense line of RGB-D cameras [64]. Human skeleton tracking from monocular images was seen in [65], [9], [11], [23] and is founded on convolutional neural network (CNN)-based machine learning. One key element for all learning-based methods is training data which consists of a large set of images annotated with joint labels. While a number of human joint datasets are publicly available [4] [41] [68], this is not the case for robot manipulators, since each has a unique geometry. This means that in order to train a joint detector, thousands of images of a specific manipulator would need to be taken and manually labeled,

an extremely tedious and time-consuming process.

One alternative to manual joint labelling is to use a CAD model of the robot inside a simulation environment (e.g. [33], [54], [73]), which can generate unlimited numbers of images of the robot and its associated joint labels in different poses, viewing angles and environmental settings such as backdrop and lighting. Since the simulated images remain distinguishable from the real robot, we need to employ a process of domain adaptation [16] in order to apply the network trained on simulated images to real-world monocular camera images.

In this chapter, we present a pipeline to detect joint positions and estimate joint states of a robot manipulator using images from a monocular camera. We employ the two-armed Baxter robot from Rethink Robotics to experimentally validate our approach, using the unit's high-accuracy joint angle measurements as a ground truth for our joint state estimates. The specific contributions of our work are:

- Implementing a combination of image segmentation and domain transfer methods which were experimentally found to provide the best-performing processing pipeline.
- Demonstrating that our system can perform joint detection as well or better than recent state-of-the-art work [39] which provides a ready-to-use joint detection system for the Baxter robot.
- Testing of the experimental joint state estimation performance in a variety of test settings including different backdrops and robot arm motions, and assessing the resulting performance quantitatively against the ground truth.

4.3 Related Work

Conventional sensors to measure joint states of robot arms are encoders, either rotary or linear. Hydraulic cylinders, typically used to actuate heavy machinery such as construction equipment, can be equipped with position sensors

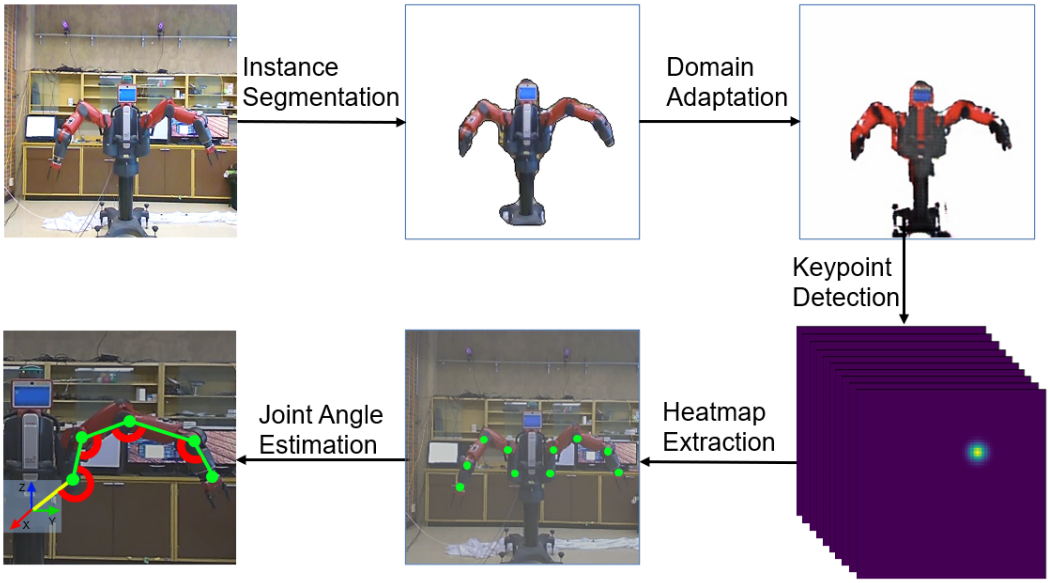

Figure 4.1: Overview of our system pipeline. The camera image is processed by robot instance segmentation, removing the background. The resulting image undergoes domain adaptation to change the coloring and texture of the robot to a simulation style. The keypoint detection network processes the synthetic images and outputs heat maps for robot joint locations. The detected joints are used with the robot's geometry to estimate joint angles.

such as LVDT (Linear Variable Differential Transformers) or other technologies [30]. Vision-based approaches to measuring robot arm joint states can be divided into 2D image-based methods, based on fiducial markers or trained Convolutional Neural Networks (CNNs), and 3D sensor-based methods, based on RGB-D cameras or LiDAR. In the first category, fiducial markers [21] provide good performance in nominal conditions due to their easy-to-spot nature, but may fail in the case of motion blur, unfocused cameras, or occluded or dirty markers. CNNs are a promising approach to this problem, and research works have shown them capable of delivering excellent performance and robustness to real-world effects such as motion blur and dynamic environments. In the second category, RGB-D cameras are capable of directly measuring depth and thus avoid scale ambiguity, but typically have a limited sensing range and may not work reliably in outdoor scenes. LiDAR works in both indoor and outdoor environments, but good-quality units are much more expensive than the other sensing technologies. In our studies, after first evaluating fiducial markers and

RGB-D cameras, we settled on the CNN approach.

4.3.1 Human Skeleton Tracking from 2D images

A number of recent research works have focused on human skeleton tracking from 2D images, which all require a large set of training images annotated with joint locations, obtained from either manual labeling [4] or specialized equipment [69]. Manual labelling is typically the preferred approach since it can be applied to a variety of human subjects in both indoor and outdoor images. The CNNs developed for human pose estimation have several variations. Early work focused on detecting a bounding box for individual limbs of the human body [75]. Skeleton joint detection was then found to provide better performance, for instance [65] and [10] employed an iterative regression method to detect joints. Meanwhile, one-pass joint detectors suitable for real-time skeleton detection were demonstrated in [9], [11], [74]. These methods produce a set of heat maps, one for each joint of interest, indicating the statistical likelihood of the joint's location.

4.3.2 Robot Arm Pose Estimation

For relatively simple open kinematic chains, namely with 4 or less links, the arm joint states can be obtained by estimating the pose of the end-effector and then using inverse kinematics (IK). However, this approach will not work for fully- or over-actuated manipulators, such as the Baxter's twin 7 DoF arms, since in this case IK has multiple solutions. In this case individual joint angle detection is necessary. One notable example of this approach is DART (Dense Articulated Real-Time Tracking) [59], which employs RGB-D images and a 3D CAD model of the robot to track poses of individual links in real time. DART is based on optimization using a signed distance function to minimize the error between model and RGB-D point cloud. The limitation of this method is the requirement for a depth camera, which as discussed earlier has disadvantages in cost and range over a monocular camera.

Two recent works addressing the domain gap problem for robotic arms using domain adaptation are CRAVES (Controlling Robotic Arm with a Vision-

based Economic System) [78] and DREAM (Deep Robot-to-camera Extrinsics for Articulated Manipulators) [39]. In CRAVES, 17 distinguishable feature points on the surface of a low-cost robot arm manipulator are selected as keypoints to train a 2-stack hourglass network [50], and a least squares fitting is used to find joint angles which best match the detected keypoints. In DREAM, the individual joints of the Baxter robot are used as keypoints, and the detection model is trained using a CNN with VGG-19 [63] as an encoder followed by a customized decoder. In both these papers, the keypoint detection networks are trained on simulation images, then applied to real images.

4.4 Method

4.4.1 System Overview

We tried several approaches, such as affixing coloured strips to robot links and detecting their angles through colour thresholding, or using a pre-trained CNN [63] with transfer learning based on robot images and corresponding encoder measurements to detect joint states directly, but the results were disappointing. We then moved to keypoint detection. Several variations of the keypoint detection process were also tried. After much trial and error, the approach described below was found to provide the best results.

In our chosen methodology, each monocular camera image I_1 is processed in three stages, as shown in Figure 4.1. Instance segmentation is performed to separate the robot from its background, yielding image I_2 of the robot over a white background. Next a generative neural network is used to perform domain adaptation of the physical image I_2 into a simulation-style image I_3. A joint detection CNN inputs I_3 and outputs the pixel coordinates of the 10 joints of the two Baxter robot arms. The joint locations are combined with a geometry model of the robot to estimate joint angles. The following sections provide details about each step, focusing on two aspects: the choice of method and the training data preparation steps.

4.4.2 Instance Segmentation

The first processing step applied to the image is instance segmentation, which extracts an object (here the Baxter robot) from the image. This is required for the next step, the generative neural network, which requires a segmented image of the robot. There are many image segmentation methods available, e.g. InstanceCut [32], DIN [5], SGN [42], and Mask R-CNN [26]. We chose to use Mask R-CNN as our segmentation method because it was found to have higher accuracy compared to other methods.

To train the Mask R-CNN model, individual image frames of the Baxter robot at their original resolution (1920×1080 pixels) are placed in X, and a polygon outline of the Baxter robot in the image is placed in Y. The mask labelling can be done either automatically or manually. For automatic labelling, a uniform background is needed, and the robot's outline can be extracted using standard colour filters. Manual labelling can yield higher-precision outlines, but the required labour made it impractical for our study.

Thanks to Mask R-CNN's pre-trained network and transfer learning, we found that using only 80 images of the Baxter robot captured in our lab was sufficient to adequately train the masking network. The network weights from Detectron2 [72] were pre-trained on a large number of common objects, so training the network for the Baxter took only 30 minutes. Each image in our mask training dataset includes the majority of the robot body, taken at different viewing angles, distances from the robot, and configurations of the robot arms.

Since Mask R-CNN employs polygon outlines, masked images may fail to capture fine extremity details such as robot grippers or caster wheels. There are new variants of Mask R-CNN which employ finer masks [72] and which could be implemented in future work.

4.4.3 Domain Adaptation

The purpose of domain adaptation is to transfer real images of the Baxter into simulation-style images, which can be used for joint detection in the next step

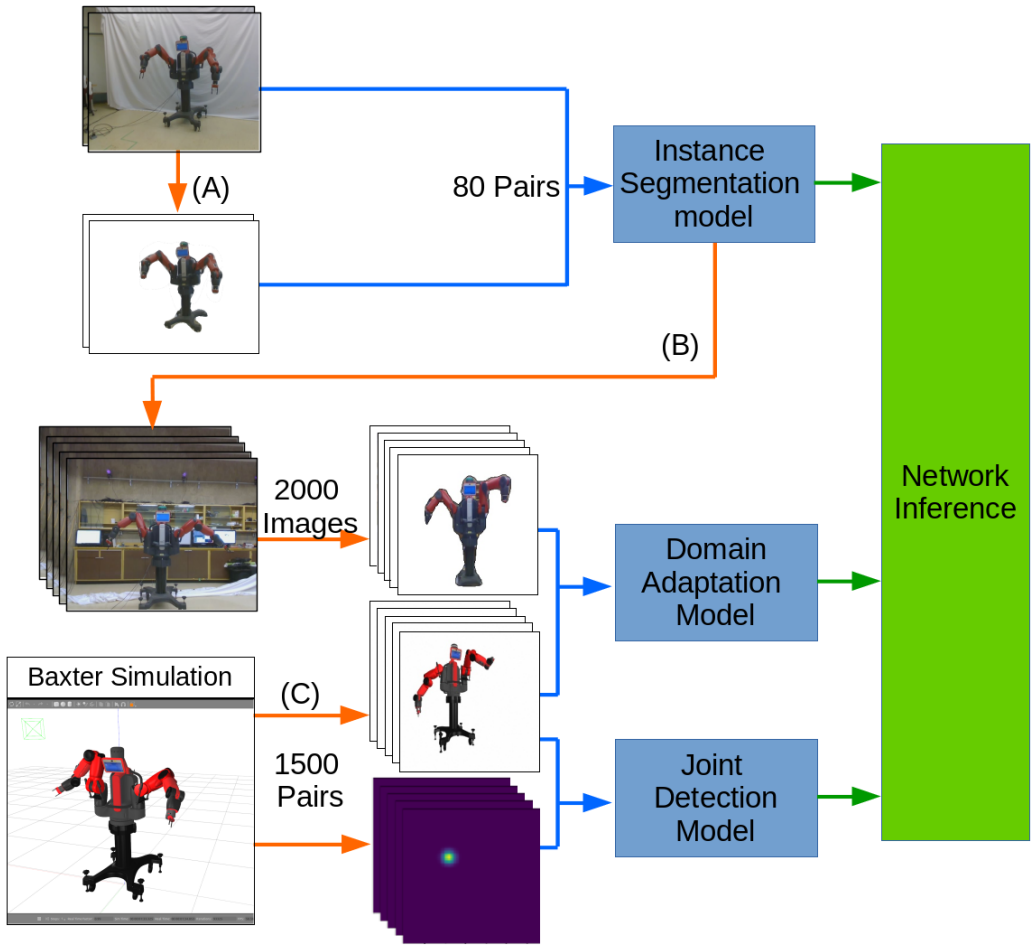

Figure 4.2: Diagram of training process. Step (A): outlines are automatically generated for real Baxter images and used along with corresponding images to train the instance segmentation model. Step (B): the segmentation model is applied to real Baxter images, resulting in images with a blank background. Step (C): the simulator generates simulation images using joint angles acquired in step (B). The step (B) and (C) images are used to train the domain adaptation model. The results are used with joint locations computed in step (C) to train the joint detection model. Once all models are trained, they are ready to be used for inference in our processing pipeline.

of the pipeline. We chose to use the generative adversarial network CycleGAN [77] for this purpose because of its excellent performance relative to other methods, and the fact that it does not require paired training images, which as explained below is an important feature for our pipeline.

We used Gazebo [33] and ROS (Robot Operating System) [55] along with CAD model files of the Baxter robot provided by its manufacturer to generate simulation images. CycleGAN consists of two generators $\{G_{XY}, G_{YX}\}$ and two discriminators $\{D_X, D_Y\}$. We assign the segmented real images as the input set $\{X\}$ and the simulation-style images as the output set $\{Y\}$. Generator G_{XY} is trained to transfer images $\{X\}$ into images $\{Y\}$, G_{YX} does the reverse, and the two discriminators D_X, D_Y are used to determine the quality of the resulting images. While all four parts are involved in the training process, only the real-to-simulated image generator model G_{XY} is retained for use in our proposed pipeline.

Data for the CycleGAN domain adaptation method consists of two image sets: real camera images $\{I_r\}$ and simulation images $\{I_s\}$. As mentioned above, CycleGAN has the noteworthy feature that the two sets of images do not need to be paired, in other words the real images and simulation images do not need to have identical arm configurations nor camera-to-robot poses. Due to this, our CycleGAN training process does not require measuring camera poses during data collection nor matching the robot arm movements between experiment and simulation, which greatly simplifies data collection. This feature is essential if applying our pipeline to a manipulator without joint state feedback, such as those mentioned in Section 4.2.

In our training, we captured 2000 images of the real Baxter robot in our lab, employing a RealSense D415 camera (using only the monocular camera images) moving in front of the Baxter and yawing up to 70 degrees in either direction, while having the Baxter executing a series of random arm motion sequences. The resulting images were processed with the Mask R-CNN model trained at the previous step, resulting in 2000 images consisting of the masked-out robot over a white background. Due to the nature of Mask R-CNN as well as limited number of training data, the resulting masked images of the robot

were not always clean, in particular distal joints of the arms were cropped off in a portion of the images. Meanwhile, 1500 images were captured in the Gazebo simulation environment, employing a random motion for the virtual camera and having the simulated Baxter perform random arm motions. In order to boost color contrast of the images, additional virtual spotlights were added to the scene. The simulation background was set to white to match the segmented images output from Mask R-CNN. The intrinsic parameters of the virtual camera were set to match those of the D415 2D camera.

4.4.4 Joint Detection

The joint detection process is performed on the set $\{I_3\}$ of post-image segmentation and domain adaptation images. The Baxter robot has 7 joints on each arm, but 2 of them located on the elbow and wrist links are nearly unobservable from a video, so we chose to focus on 5 joints on each arm of the Baxter, as illustrated in Figure 4.1. Inspired by current research into human joint detection and skeleton tracking [9], [74], we chose the ResNet [27] architecture for robot joint detection. Since the original ResNet has fully connected output layers, we used ResNet-50 with all its convolution, pooling and activation layers preserved, but replaced the output layer by transposed convolutional layers (3×3) for upscaling resolution. The network inputs a monocular image with a resolution of 640 by 480 pixels, and outputs 10 heat maps (one for each joint) each with the same resolution as the input, ideally containing a bivariate Gaussian distribution rendering the estimated position and associated uncertainty for the joint.

In order to train the joint detection module, we captured 8000 simulation images of the Baxter at different arm configurations and camera poses. The arm configurations spanned the full range of motion of each joint, while the camera poses covered a frustum in front of the robot with an angle of 140 degrees and depths between 3 and 4 meters. The 3D coordinates and 2D projections of each joint were calculated using the robot's CAD model and camera intrinsics, respectively. The combination of 2D simulated images and corresponding joint locations was used to train the ResNet joint detection network.

To improve the quality of the trained network, we applied image augmentation processes including random color saturation and brightness adjustment, cropping, and background augmentation using images from the COCO dataset [41].

4.4.5 Joint State Estimation

The link geometries of the Baxter robot are contained inside a Unified Robot Description Format (URDF) file. Using this information together with joint states, we employ forward kinematics to calculate the 3D position of each joint relative to a frame attached to the base link of the robot. Then, using the $SE(3)$ pose of this frame relative to the camera, we can project the estimated joint positions into the image frame, where they are compared against the detected joints.

The relationship between pixel locations (u, v) in the image frame and 3D coordinates (x, y, z) relative to the camera lens-fixed frame is

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.1}$$

where K is the camera intrinsic matrix, whose entries can be obtained from a calibration process, and Π_0 is a projection matrix. The 3D coordinates (x, y, z) in the camera-fixed frame are obtained from the coordinates (X, Y, Z) in the robot's base link frame as

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T_{cb} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.2}$$

where $T_{cb} \in SE(3)$ is the pose of the base link relative to the camera, which can be measured directly as explained in Section 4.5.1. Finally, the 3D coordinates $(X_{n+1}, Y_{n+1}, Z_{n+1})$ of the $(n+1)^{\text{th}}$ joint are given by forward kinematics, specifically the Product of Exponentials formulation [48]

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \\ Z_{n+1} \\ 1 \end{bmatrix} = e^{X(\hat{\xi}_1)\theta_1} \cdots e^{X(\hat{\xi}_n)\theta_n} T^0_{b(n+1)} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.3}$$

where each $\hat{\xi}_n$, the twist axis of the n^{th} joint, and $T^0_{b(n+1)} \in SE(3)$, the pose of the reference frame with origin at the $(n+1)^{\text{th}}$ joint and fixed to link $(n+1)$, can be obtained from the URDF file. Note for $n = 0$, $[X_1, Y_1, Z_1, 1]^T = T^0_{b1}[0, 0, 0, 1]^T$ because the position of the first joint is fixed relative to the base link frame.

Combining (4.1), (4.2) and (4.3) yields the (nonlinear) function

$$(u_{n+1}, v_{n+1}) = f_n(\theta_1, \cdots, \theta_n) \tag{4.4}$$

which relates the 2D image coordinates of the $(n+1)^{\text{th}}$ joint to the set of joint states $(\theta_1, \cdots, \theta_n)$ affecting it. In our case, each arm has five detected joint locations $(u_n^{\text{det}}, v_n^{\text{det}})$, $1 \leq n \leq 5$, giving the set of four residuals

$$\begin{bmatrix} (u_2^{\text{det}}, v_2^{\text{det}})^T - f_1(\theta_1) \\ \vdots \\ (u_5^{\text{det}}, v_5^{\text{det}})^T - f_4(\theta_1, \cdots, \theta_4) \end{bmatrix} \tag{4.5}$$

which can be minimized using a standard nonlinear least-squares method. We employ a trust-region-reflective algorithm, with bounding values for the θ vector set to the physical joint limits of the Baxter robot.

4.5 Experimental Results

4.5.1 Datasets

We collected seven datasets using the Baxter robot, varying the two conditions listed in Table 4.1: arm actions and image background. The camera was fixed to face the robot head-on throughout data collection, at a height of 1.5 meters and depth of 3 meters. This camera placement was chosen to ensure the robot had a reasonable size within the image frame, and that its arms did not leave the frame during motions. The two types of arm actions were waving and pick-and-place. In waving, both robot arms moved in random motions throughout their reachable workspace, while in pick-and-place operations one arm was programmed to sequentially pick up then drop three black blocks into a designated container. The backdrop of the robot was our research lab, as seen in Figure 4.1, in a static and a dynamic variant. In the former the

backdrop was fixed, while in the latter a person walked behind the robot while holding up a large checkerboard. All images were captured with a resolution of 640×480 and a framerate of 30 fps.

Throughout the image capture process, the joint angle encoder measurements of the robot arms were logged, and the camera-to-robot pose was measured using a Vicon Vero motion capture system [69] installed in the lab. Combined with the forward kinematics of the robot and the intrinsic parameters of the camera, the ground truth coordinates of the individual joints were calculated at each frame. The camera-to-robot pose was also used in the joint state estimation calculations described in Section 4.4.5, with the measured joint states used as the corresponding ground truth.

Table 4.1: Experimental datasets

Dataset	# of images	Arm actions	Background
T_1	5583	Waving	Dynamic
T_2	5789	Pick-and-Place	Dynamic
T_3	4364	Pick-and-Place	Static
T_4	4012	Waving	Static
T_5	3987	Waving	Static
T_6	7684	Waving	Dynamic
T_7	5667	Waving	Dynamic

4.5.2 Joint Detection Evaluation

Before testing joint angle estimation, we first quantified the performance of joint detection in 2D images.

Methodology comparison

A recently proposed methodology for robot joint detection is DREAM (Deep Robot-to-camera Extrinsics for Articulated Manipulators) [39]. Their method is capable of detecting robot keypoints such as joints in real images subject to varying environmental lighting and surface texturing. The authors of DREAM released a joint detection network trained on a Baxter robot, allowing us to

immediately use their system for comparison purposes.

Joint Detection Evaluation Metrics

For each image, the 2D euclidean pixel distances between the detected and ground truth coordinates of the 10 joints were calculated. The metrics chosen to quantify the performance of the system up to 2D keypoint detection are Percentage of Correct Keypoints (PCK) and Mean Average Error (MAE). The PCK@0.2 metric [66] provides the percentage of joints in a dataset whose euclidean distance in pixels is within 20% of the robot's bounding box size (the maximum of length and width). In addition to PCK@0.2, we will also provide plots of PCK scores against a distance varying from 1 to 30 pixels. The MAE metric is the mean of the euclidean distances in pixels between a detected joint and its ground truth location across a data set.

Since the Baxter robot has symmetric arms, the per-joint evaluation scores were averaged between corresponding joints on the left and right arm to reduce the amount of data being reported. For the joint detection evaluation, the results for each joint were also averaged out across the seven datasets. The results will thus reflect the typical performance of the two methods across a variety of tasks and conditions. Performance will be broken out by dataset in Section 4.5.3, when evaluating the joint estimation performance.

Simulation Testing

The performance of our joint detection method was first tested entirely in simulation. We collected 10,000 simulated images of the Baxter robot, and augmented their backgrounds with random images from the COCO datasets. A joint detection network was trained using these synthetic images, using joint coordinate labels provided by the simulation environment. The trained network was tested on 8,000 synthetic images, this time using the simulator-provided joint locations as the ground truth. The resulting PCK@0.2 score for the test was 99.91%, giving us confidence about our implementation.

Testing on Real Datasets

Next, we applied the two joint detection methods to the seven experimental datasets described in 4.5.1, taking the average of the performances across all seven datasets. Figure 4.3 illustrates the average joint detection performance of both methods for each joint in terms of PCK versus detection pixel distance. As expected, all PCK scores climb as the threshold pixel distance is increased. We see both methods provide roughly similar results for the Base and Shoulder joints, with DREAM having an advantage in the Elbow joint but our method having an advantage in the more distal Wrist and Hand joints. These results are confirmed in Table 4.2, which lists the average PCK@0.2 score for each joint. Our method thus exhibits a slight edge over DREAM in terms of joint detection performance.

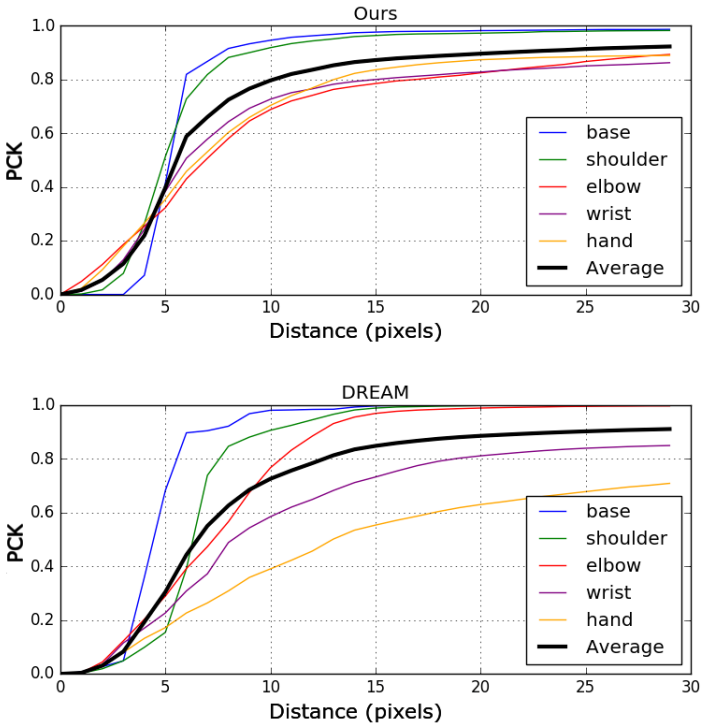

Figure 4.3: PCK versus pixel distance. Top figure is the plot of result from our method. Bottom figure is from DREAM method. Black curves in the plots are the averages of the curves of all joints.

The accuracy of joint detection is quantified in Table 4.3, showing the

Table 4.2: PCK@0.2 scores

	Joints					Overall
Method	Base	Shoulder	Elbow	Wrist	Hand	Mean
DREAM	0.99	0.99	**0.99**	0.94	0.86	0.95
Ours	0.99	0.99	0.98	**0.96**	**0.92**	**0.97**

MAE errors between detected and ground truth locations of each joint, averaged out over the seven datasets, for each method. Here our method exhibits better performance than DREAM across all joints. Remark both joint detection methods exhibit significantly larger MAE errors at the more distal wrist and hand joints. In DREAM, this may be due to these joints being smaller, making them more difficult to distinguish from the background. In our proposed method, the segmentation module occasionally crops the wrist joint of the robot, leading to outliers in the data which increase the MAE. This last effect is discussed in Section 4.5.4.

Table 4.3: Joint Detection MAE Errors

	Joints					Overall
Method	Base	Shoulder	Elbow	Wrist	Hand	Mean
DREAM	11.82	11.74	12.98	15.64	20.43	14.52
Ours	**10.01**	**10.62**	**8.09**	**13.17**	**18.08**	**11.99**

4.5.3 Joint Angle Estimation

We now combine our joint detection method with the joint state estimation method described in Section 4.4.5. We employ the seven datasets described in Section 4.5.1, breaking out by dataset. The performance metric chosen for this part was the mean of the joint angle errors $(\theta_k^{\text{true}} - \theta_k)$, $1 \leq k \leq 4$, corresponding to the base, shoulder, elbow and wrist joints. To reduce the amount of data shown, corresponding joints on the left and right arms were combined in the "Waving" datasets in Table 4.1. In the "Pick-and-Place" datasets, the joint data from the single active arm was reported. The results

are shown in Table 4.4. Note that since angle errors may be either positive or negative, the same is true of their average value.

Table 4.4: Mean of joint angle estimation errors

Dataset	Base θ_1	Shoulder θ_2	Elbow θ_3	Wrist θ_4
T_1	11.13	4.98	5.11	-15.60
T_2	4.62	9.69	-13.92	18.55
T_3	4.55	7.72	-8.09	-6.75
T_4	16.66	5.76	-6.01	-7.05
T_5	9.56	7.37	-4.29	-1.26
T_6	13.45	3.25	5.37	-20.11
T_7	11.72	3.38	2.43	-9.83
Mean	10.24	6.02	-2.77	-6.01

The results in Table 4.4 are mixed. The Base has relatively poor estimation results. A likely reason is the nature of the motions: in both T_2 and T_3, the pick-and-place operations, the robot executes a series of slow arm sweeps about the Base joint, and we see the errors are considerably lower than the other datasets. The opposite is seen in the Waving datasets (T_1 and T_4–T_7), where the Shoulder and Elbow joints carry out larger motions while the Base joint does not move as much, which is reflected by their associated estimation errors. The Wrist joint estimates are uniformly poor, but this is due to the calculation of θ_4 relying exclusively on the detected position of the hand joint (c.f. Section 4.4.5), which as seen in Section 4.5.2 is the most error-prone. Despite all these factors, the proposed method shows promise, and the overall estimation errors of roughly 5 degrees are sufficiently good to perform rough motion planning. We suspect that system performance could be greatly improved by using higher-resolution input images, since the current input data consisting of 640×480 images of the robot at a distance of 3 m from the camera naturally leads to large uncertainties within the joint state estimates.

Figure 4.4: Examples of some common failure cases during testing. It includes the occlusion failure, background segmentation failure, and robot arm segmentation failure.

4.5.4 Failure Cases

Examples of common failure cases are shown in Figure 4.4. Each row is one case. In the first one, the left arm's wrist is cropped during segmentation, resulting in an incorrect detection for the left wrist joint. The right wrist is in an occluded pose, and CycleGAN fails to reconstruct the proper texture. In the second case, a person holding a checkerboard was moving in the background, leading to the segmentation module producing a mask containing a portion of the board; the right wrist was again cropped. In the third case, masking filtered out the checkerboard, but the accuracy of the mask region was affected.

4.6 Conclusion

In this chapter, we presented a pipeline which employs monocular images of a Baxter two-armed robot and performs joint detection for the purpose of estimating the arm configuration without the use of joint encoders. The pipeline is built on a combination of instance segmentation, domain adaptation, and joint detection modules based on CNNs. Our proposed approach requires very minimal manual labeling of the training data, and relies on a generative adversarial network (GAN) to perform domain adaptation. We performed extensive quantitative evaluation of our method and showed that it performs as well or better than other current state-of-the-art joint detection algorithms. The interest of our approach is the ability to use the resulting joint state information to enable motion control with either low-cost robot arms which are equipped with low-quality encoders and/or are subject to significant levels of backlash, or manipulators without joint state measurements such as construction equipment.

Future work will include applying our method to cases where precise CAD files of the manipulator are not readily available, for instance construction equipment. This is possible since GANs are capable of transferring images between domains while altering their style. Other improvements include investigating performance improvements from training on high-resolution images, adding outlier detection and removal to reduce false positive detections, and implementing model-based filtering for the detected joint locations to provide smoother data.

Chapter 5

Conclusion

5.1 Summary of Thesis

This thesis documented the research work performed using the Baxter robot manipulator testbed along with computer vision and convolutional neural network techniques. The work focused on solving the challenge of vision-based joint state estimation.

We developed a pipeline for estimating the joint angles of a robotic manipulator using monocular camera images as the only input. While we employed the Baxter robot for hardware testing, since the longer-term goal is to implement this method on a large outdoor loading crane, the pipeline needs to be able to operate in outdoor conditions and at significant camera depths. Before settling on the chosen method, we tried several different strategies for joint state estimation, including feature extraction from 2D images using conventional computer vision filters, feature matching with 3D models, and depth image registration. These preliminary trails identified deficiencies such as poor performance for complex backgrounds and/or objects, while the methods relying on depth images were subject to range and outdoor operation restrictions imposed by the sensing hardware. In the ultimately chosen method, three neural networks are used to process the monocular camera images and estimate the location of joints in the image frame, which is then used in a module employing the forward kinematics of the manipulator, camera projection model and optimization algorithm to estimate the joint angles. The main challenge involved with this method was solving the domain gap between simulated

and real-world images. The joint detection subsystem produced accuracy and stability which were as good or better than other published methods for joint detection from monocular images. The joint angle estimation calculations produced good results for a subset of the joints; obtaining good performance for all the joints has been left for future work.

5.2 Limitations

The proposed solution shows promising result, but there remain limitations as listed below:

- The current instance segmentation model is based on Mask-RCNN [26], which is capable of generating object masks in real-time, but the resulting mask tends to crop out fine details associated with small components on the robot's end-effector gripper.

- The Generative Adversarial Network used in our pipelines is CycleGAN [77], which did not successfully train with higher-resolution images. While the current image resolution is capable of rendering textures and features on the robot, high-detail textures are dropped due to the necessity of resizing the image.

- The joint detection of distal joints is not not as stable as for the proximal ones. This problem leads to inaccurate joint angle estimates at the optimization calculations.

- In the joint state estimation, the number of equations is exactly equal to the number of unknowns. This enables computing the values of the joint states, but there is no redundant information which could increase the robustness of estimation to outliers and provide estimates with better accuracy.

5.3 Future work

Certain parts of the approaches proposed in this thesis could be improved in the future by either implementing newer algorithms or using a different setup:

- A more advanced instance segmentation network could be implemented to yield better mask generation capable of capturing finer details.
- Picking up extra keypoints on each robot arm link in order to increase robustness and accuracy of the joint state estimation calculations
- Since CycleGAN can transfer texture between images, implementing the proposed pipeline with a simplified CAD model of the manipulator is a promising research avenue. A specific benefit would be the elimination of the need for a detailed 3D model of the manipulator.

References

[1] H. I. AB. [Online]. Available: https://www.arwtruck.com/media/truck-mounted/hiab/xs-166/brochures/HIAB-XS-166-CLX-Basic-Data.pdf.

[2] D. S. Alexiadis, P. Kelly, P. Daras, N. E. O'Connor, T. Boubekeur, and M. Ben Moussa, "Evaluating a dancer's performance using Kinect-based skeleton tracking," in *Proceedings of the 19th ACM international conference on Multimedia*, Scottsdale, AZ, Nov. 2011, pp. 659–662.

[3] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[4] ——, "2D human pose estimation: New benchmark and state of the art analysis," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, Jun. 2014, pp. 3686–3693.

[5] A. Arnab and P. H. S. Torr, "Pixelwise instance segmentation with a dynamically instantiated network," in *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, Jul. 2017, pp. 879–888.

[6] C. M. Bautista, C. A. Dy, M. I. Mañalac, R. A. Orbe, and M. Cordel, "Convolutional neural network for vehicle detection in low resolution traffic videos," in *2016 IEEE Region 10 Symposium (TENSYMP)*, IEEE, 2016, pp. 277–281.

[7] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992. DOI: 10.1109/34.121791.

[8] D. C. Brown, "Decentering distortion of lenses," *Photogrammetric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[9] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: Realtime multi-person 2D pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 172–186, Jan. 2021.

[10] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik, "Human pose estimation with iterative error feedback," in *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, Jun. 2016, pp. 4733–4742.

[11] Y. Chen, C. Shen, X.-S. Wei, L. Liu, and J. Yang, "Adversarial PoseNet: A structure-aware convolutional network for human pose estimation," in *Proceedings of 2017 IEEE International Conference on Computer Vision*, Venice, Italy, Oct. 2017, pp. 1221–1230.

[12] P. Y. Chua, T. Ilschner, and D. G. Caldwell, "Robotic manipulation of food products – a review," *Industrial Robot: An International Journal*, vol. 30, no. 4, pp. 345–354, Aug. 2003.

[13] J. Chung and K. Sohn, "Image-based learning to measure traffic density using a deep convolutional neural network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 5, pp. 1670–1675, 2017.

[14] 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.

[15] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette, "Real-time markerless tracking for augmented reality: The virtual visual servoing framework," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 615–628, 2006. DOI: 10.1109/TVCG.2006.78.

[16] G. Csurka, "Domain adaptation for visual applications: A comprehensive survey," in *Domain Adaptation in Computer Vision Applications*, ser. Advances in Computer Vision and Pattern Recognition, G. Csurka, Ed., Cham, Switzerland: Springer, 2017, pp. 1–35.

[17] 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.

[18] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 304–311. DOI: 10.1109/CVPR.2009.5206631.

[19] Y. Fernando, A. Mathath, and M. A. Murshid, "Improving productivity: A review of robotic applications in food industry," *International Journal of Robotics Applications and Technologies*, vol. 4, no. 1, pp. 43–62, 2016.

[20] K. Fukushima, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. DOI: doi.org/10.1007/bf00344251.

[21] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marın-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, Jun. 2014.

[22] 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*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., vol. 27, 2014, pp. 2672–2680.

[23] R. A. Güler, N. Neverova, and I. Kokkinos, "DensePose: Dense human pose estimation in the wild," in *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, Jun. 2018, pp. 7297–7306.

[24] M. Han, B. Xie, M. Barczyk, and A. Bayat, "Image-based joint state estimation pipeline for sensorless manipulators," *IEEE Robotics and Automation Letters*, 2021, Submitted.

[25] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the Alvey Vision Conference, AVC 1988, Manchester, UK, September, 1988*, C. J. Taylor, Ed., Alvey Vision Club, 1988, pp. 1–6. DOI: 10.5244/C.2.23. [Online]. Available: https://doi.org/10.5244/C.2.23.

[26] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 386–397, Feb. 2020.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, Jun. 2016, pp. 770–778.

[28] C. Heindl, S. Zambal, T. Pönitz, A. Pichler, and J. Scharinger, *3d robot pose estimation from 2D images*, arXiv:1902.04987, Feb. 2019.

[29] C. Hernandez *et al.*, "Team delft's robot winner of the amazon picking challenge 2016," in *RoboCup 2016: Robot World Cup XX*, ser. Lecture Notes in Artificial Intelligence, S. Behnke, R. Sheh, S. Sariel, and D. D. Lee, Eds., vol. 9776, Cham, Switzerland: Springer, 2017, pp. 613–624.

[30] C. C. P. Inc, *Cpi hydraulic cylinder position sensor*, https://www.cpi-nj.com/hydraulic-cylinder-position-sensors.

[31] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, "Traffic monitoring and accident detection at intersections," *IEEE transactions on Intelligent transportation systems*, vol. 1, no. 2, pp. 108–118, 2000.

[32] A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother, "InstanceCut: From edges to instances with MultiCut," in *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, Jul. 2017, pp. 7322–7331.

[33] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep. 2004, pp. 2149–2154.

[34] M. Kyrarini, M. A. Haseeb, D. Ristić-Durrant, and A. Gräser, "Robot learning of industrial assembly task via human demonstrations," *Autonomous Robots*, vol. 43, no. 1, pp. 239–257, Jan. 2019.

[35] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[36] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[37] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997. DOI: 10.1109/72.554195.

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[39] 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 *Proceedings of 2020 International Conference on Robotics and Automation (ICRA)*, Paris, France, May 2020, pp. 9426–9432.

[40] Z. Li, K. Okada, and M. Inaba, "Searching a suitable keypoint detection network for robotic assembly," in *Proceedings of the 2020 IEEE/SICE International Symposium on System Integration (SII)*, Honolulu, HI, Jan. 2020, pp. 377–383.

[41] T.-Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Computer Vision - ECCV 2014*, ser. Lecture Notes in Computer Science, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693, Cham, Switzerland: Springer, 2014, pp. 740–755.

[42] S. Liu, J. Jia, S. Fidler, and R. Urtasun, "SGN: Sequential grouping networks for instance segmentation," in *Proceedings of 2017 IEEE International Conference on Computer Vision*, Venice, Italy, Oct. 2017, pp. 3516–3524.

[43] J. Lu, F. Richter, and M. C. Yip, *Robust keypoint detection and pose estimation of robot manipulators with self-occlusions via sim-to-real transfer*, arXiv:2010.08054, Oct. 2020.

[44] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision*, ser. Interdisciplinary Applied Mathematics. New York, NY: Springer-Verlag, 2004, vol. 26.

[45] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[46] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[47] 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.

[48] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[49] G. A. Nayik, K. Muzaffar, and A. Gull, "Robotics and food technology: A mini review," *Journal of Nutrition & Food Sciences*, vol. 5, no. 4, 2015.

[50] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *Computer Vision - ECCV 2016*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9912, Cham, Switzerland: Springer, 2016, pp. 483–499.

[51] C. Park and K. Park, "Design and kinematics analysis of dual arm robot manipulator for precision assembly," in *Proceedings of the IEEE International Conference on Industrial Informatics*, Daejeon, Korea, Jul. 2008, pp. 430–435.

[52] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," 2015.

[53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[54] W. Qiu *et al.*, "UnrealCV: Virtual worlds for computer vision," in *Proceedings of the 25th ACM international conference on Multimedia*, Mountain View, CA, Oct. 2017, pp. 1221–1224.

[55] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, May 2009.

[56] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[57] 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`.

[58] RLS, *Rls robotic encoders products*, `https://www.rls.si/eng/products`.

[59] T. Schmidt, R. Newcombe, and D. Fox, "DART: dense articulated real-time tracking with consumer depth cameras," *Autonomous Robots*, vol. 39, no. 3, pp. 239–258, Oct. 2015.

[60] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[61] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. eprint: `arXiv:1705.05065`. [Online]. Available: `https://arxiv.org/abs/1705.05065`.

[62] T. R. Shaham, T. Dekel, and T. Michaeli, "Singan: Learning a generative model from a single natural image," *CoRR*, vol. abs/1905.01164, 2019. arXiv: `1905.01164`. [Online]. Available: `http://arxiv.org/abs/1905.01164`.

[63] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations*, San Diego, CA, May 2015.

[64] *Skeleton tracking SDK getting started guide*, `https://www.cubemos.com/skeleton-tracking-sdk`, cubemos.

[65] A. Toshev and C. Szegedy, "DeepPose: Human pose estimation via deep neural networks," in *Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, Jun. 2014, pp. 1653–1660.

[66] J. Tremblay, T. To, A. Molchanov, S. Tyree, J. Kautz, and S. Birchfield, "Synthetically trained neural networks for learning human-readable plans from real-world demonstrations," in *Proceedings of the 2018 International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 5659–5666.

[67] S. Ullman, "The interpretation of structure from motion," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 203, no. 1153, pp. 405–426, 1979.

[68] *VGG human pose estimation datasets*, `https://www.robots.ox.ac.uk/~vgg/data/pose/`.

[69] Vicon-Motion-Systems-Ltd-UK, *Vicon motion systems*, `https://www.https://www.vicon.com/`, 2019.

[70] M. Wang and W. Deng, "Deep face recognition: A survey," *Neurocomputing*, vol. 429, pp. 215–244, 2021. DOI: `10.1016/j.neucom.2020.10.081`. [Online]. Available: `https://doi.org/10.1016/j.neucom.2020.10.081`.

[71] M. J. Westoby, J. Brasington, N. F. Glasser, M. J. Hambrey, and J. M. Reynolds, "'structure-from-motion' photogrammetry: A low-cost, effective tool for geoscience applications," *Geomorphology*, vol. 179, pp. 300–314, Dec. 2012.

[72] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, `https://github.com/facebookresearch/detectron2`, 2019.

[73] F. Xia *et al.*, "Interactive Gibson Benchmark: A benchmark for interactive navigation in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 713–720, Apr. 2020.

[74] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Computer Vision - ECCV 2018*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11210, Cham, Switzerland: Springer, 2018, pp. 472–487.

[75] Y. Yang and D. Ramanan, "Articulated human detection with flexible mixtures of parts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2878–2890, Dec. 2012.

[76] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[77] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of 2017 IEEE International Conference on Computer Vision*, Venice, Italy, Oct. 2017, pp. 2242–2251.

[78] Y. Zuo, W. Qiu, L. Xie, F. Zhong, Y. Wang, and A. L. Yuille, "CRAVES: Controlling robotic arm with a vision-based economic system," in *Proceedings of 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, Jun. 2019, pp. 4209–4218.