CHALMERS
UNIVERSITY OF TECHNOLOGY

Cobot + Vision to remove plastic straps from pallets

An evaluation of the automation of de-strapping of pallets using a Universal Robot and a Photoneo 3D-scanner

Master's thesis in Systems, Control and Mechatronics

Roger Lokku
Alexander Gustafsson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Cobot + Vision to remove plastic straps from pallets

An evaluation of the automation of de-strapping of pallets using a Universal Robot and a Photoneo 3D-scanner

Roger Lokku
Alexander Gustafsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of ELECTRICAL ENGINEERING
Division of Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2026

Cobot + Vision to remove plastic straps from pallets
Automation of de-strapping of pallets using a Universal Robot and a Photoneo 3D-scanner
Roger Lokku, Alexander Gustafsson

Supervisor: Emmanuel Dean, Electrical Engineering
Supervisor: Mikael Granbom, Volvo Group
Examiner: Emmanuel Dean, Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A UR10e robot in action - holding a strap based on its detection, identification and localization using a Photoneo MotionCam-3D.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Cobot + Vision to remove plastic straps from pallets
Automation of de-strapping of pallets using a Universal Robot and a Photoneo 3D-scanner
Roger Lokku, Alexander Gustafsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis aims at assisting Volvo Trucks to automate the de-strapping of the incoming pallets on a conveyor line using a vision-based robotic solution. Volvo provides a UR10e cobot, a Photoneo 3D-scanner and Aurora Design Assistant (ADA) to carry out this thesis. In return, Volvo requires the answer to the question whether this robot, camera, and software configuration is satisfactory to get the job done. The solution, however, needs to be divided into two main parts - building the system architecture along with integration of the given equipments and development of methods for de-strapping of the pallets.

This thesis proposes the system architecture along with integration of the given equipments and the three methods that have been developed for de-strapping of the pallets. Each method is a combination of the force mode functionality of the robot and the image processing functionality via ADA after image acquisition via the camera. A python script is used to act as the middle man between the robot's controller and ADA, facilitating data exchange and logging of data.

The methods will be judged on the ability of the camera to identify and localize the strap, the ability of the UR controller to form a valid robot path to grab the straps and the cycle time. The accuracy and dependability of the robot, camera and software is judged by doing 30 cycles of grabbing the straps at various pallet offset and/or rotated from/at a fixed point. This will give quantitative data. Thus, answering Volvo's question of "Can a UR10e cobot, a Photoneo 3D-scanner and ADA be used to de-strap their pallets?".

The calibration error of the estimated homogeneous transformation matrix was found to be 3.5 mm. The fastest method was Method 3 with an average cycle time of 39.24 s. The camera, when deployed on the production line, was able to correctly determine the presence of straps with a success rate of 76.79% and the success rate of the colour identification of the straps was 75.00%. The camera was found to be able to solve the strap identification and colour identification but having a 3D camera is deemed excessive, and the image quality could be the limitation of the robustness of strap identification and colour identification.

Keywords: Cobot, Camera, Python, Image acquisition & processing, System architecture & integration.

Acknowledgements

This thesis was made possible by the guidance of Emmanuel Dean from Chalmers together with Amanda Kristensson and Ian Rathgeb from Volvo.
A huge thanks to *Universal Robot*, *OEM AUTOMATIC*, and *SCHUNK* for providing the equipment and hands-on training.
A big thanks to Mikael Granbom for the overall coordination between all the companies, the first introduction to Volvo, and the overall supervision at Volvo.
A final thanks to *Volvo* and all its employees for the opportunity of conducting this thesis and for all the help along the way.
We also want to extend our heartfelt gratitude to our family and friends for their mental support during this thesis.

Alexander Gustafsson and Roger Lokku, Gothenburg, May 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ADA	Aurora Design Assistant
C	DGUV-certified unit
Co-act	Collaborative actuator
Cobot	Collaborative Robot
CSV	Comma-separated values
EGP	Electric Small Parts Gripper
ID	Identification
IDE	Integrated Development Environment
IP	Internet Protocol
ISO	International Organization for Standardization
LAN	Local Area Network
MP	megapixel
PC	Personal Computer
PoE	Power over Ethernet
Py	Python
RGBHL	Red, green, blue, hue and luminance
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
TCP	Tool Center Point
UR	Universal Robot
URCap	Universal Robot Capability
URID	Universal Robots/with feed-through (electr. tool interface)
XML	Extensible Markup Language

Contents

List of Figures

List of Tables

1
Introduction

In the wake of the pandemic and the uncertainty of the world powers, the EU aims to move more manufacturing back to the EU [1] [2]. In order for manufacturing plants to be competitive on a global scale while being located in the EU, more automation will be needed [22]. In recent years, the importance of having production locally in the EU has been made apparent [2]. Advancements in automation have continuously been made, but now it is under more pressure for development as manufacturing plants in Europe have to be competitive on a global scale. With high salaries in the EU, the reward for automation is greater than in countries with lower salary levels. With growing competition, it is of utmost importance to optimize production, making it cost effective. A possible way of optimization is automation, i.e., transferring the manpower to higher priority tasks by automating tasks that do not require manual labor.

1.1 Background

De-strapping of pallets is one such task that could be automated. On a daily basis, Volvo Trucks receives numerous pallets that need to be de-strapped. Each pallet consists of containers that are placed side by side and stacked on top of each other, thus forming layers. A pallet can have up to four layers of containers. There are two types of pallet configuration, L- and K-type. Each pallet has a top lid, bound by straps. The straps vary in terms of colour and each colour corresponds to a material. A pallet on the conveyor is shown in fig. 1.1. On the conveyor line, three sensors can be seen behind the pallets. These sensors detect the pallet on the conveyor and stop the conveyor.

Figure 1.1: A L-type pallet with two layers of containers on the conveyor.

1.2 Purpose

Currently, the straps are cut and sorted manually (fig. 1.2). So are the top lids. Volvo desires to automate this process by using a Cobot (Universal Robots UR10e) and a camera (Photoneo PhoXi 3D Scanner gen3). The idea is to use the camera to identify and/or localize the straps and relay this information to the robot for the de-strapping process.

Figure 1.2: Manually cutting straps.

1.3 Scope

Due to safety and security reasons, the robot will not have any cutting tools attached to its end-effector. The straps will be cut manually. The thesis will be limited to - identifying the straps, moving the robot in position to grab, pull and sort them.

1.4 Limitations

Since the removal and sorting of the lids is not part of the main goal of this thesis, it can be worked on if time permits.

1.5 Hypothesis

It has been observed that there is a considerable gap between the straps and the containers (fig. 1.3). In this thesis, we propose to make use of this gap since it is advantageous to program the robot to move to this position and hold the strap. Thus, minimizing any form of damage to the containers or top lids or to the robot itself.

Figure 1.3: Gap between straps and containers.

1.6 Research questions

- Is the Aurora Design Assistant a suitable image processing software for automating the de-strapping pallets?
- Is the UR10e a suitable robot for automating the de-strapping pallets?
- Is the PhotoNeo 3D MotionCam a suitable camera for automating the de-strapping pallets?
- Is the combination of these hardware and software suitable for automating the de-strapping pallets?

2
Literature Review

In the research-based project [3], the authors present a solution to cut plastic coverings from pallets. They set out to solve the automation of removing plastic from boxes on pallets by researching how a program could find a path for the robot to cut the plastic. They did not evaluate for plastic straps but only plastic wrappings. Neither did they do anything about how to handle the plastic after it has been cut. For the different box configuration on the pallets they did not evaluate if the boxes would have a lid. The subject of cutting straps, how to deal with them after being cut and doing this when the pallets with boxes arrive with lids on them is where this thesis will compliment their work. However, the useful information for this thesis is about how to plan a path to cut plastic straps close to or touching boxes as that is how the plastic straps on Volvo's pallets are oriented. Their work on using a force sensor to detect the strap and when the plastic is being cut is also useful in solving the issue of de-strapping.

Muliter et al [6] was able to detect straps on boxes by converting them to grayscale and putting a sharpening filter on them. They had only white and light yellow straps placed mostly on a tan box as a background. They displayed one example that it could identify white straps on a blue-green box with a diagonal yellow and white pattern. However, the straps they identified were horizontal, vertical, or very close to that. They lack strap identification that are diagonally placed on boxes of the same colour as the straps. They used 2D-images and OpenCV to do the image processing and did not evaluate using 3D-images or industrial image processing software like Aurora Design Assistance (ADA).

Another tactic is to be sensor-less when cutting the straps. Daher Naseem and Shammas Elie [7] did that by building a fixture that is lowered down on top of the pallets that got into position with guiding rails. But this was only for one specific lid with specific strap position. They did not build a system that could handle variation in lid designs or even account for variation in the strap placement on the boxes. For Volvo, the system must be able to handle four different types of lids and any strap placement that goes over the lid and on any side of the pallets. Their report did not comment on the sorting of the straps, but they did cut and handle the straps by feeding them into a feeder which is relevant for the de-strapping of Volvo's pallets.

For research on strap-recognition and identification, the article *Intelligent recognition and handling of carton packing straps using 3D laser vision and deep learning* [5] describes the use of deep learning for strap recognition. Specifically using the open-

source computer vision model architecture for detection, classification, segmentation, and more, YOLOv8 with the Sea_Attention mechanism trained on 300 images plus the same images with added noise, bringing the total to 1,050 images. Their pallets go into a fixture and are analyzed under a very controlled light environment and not set in an open industrial setting. They did use straps of various colours such as white, green, black, and transparent but not blue. All of the strap identification was done on tan cardboard boxes, and therefore, not finding straps of the same colour as the background or with letters in the background.

3

Theory

This chapter highlights the key theoretical concepts necessary for understanding this thesis.

3.1 Structured light

In this report, the 3D scanner operates on the principle of sequential structured light. This works with a structured light projector and a camera at a set distance and calibration. The projector projects a set of lines that are known to the camera and thus, by analyzing the distortion of the lines, it can calculate the 3D texture of the object [4].

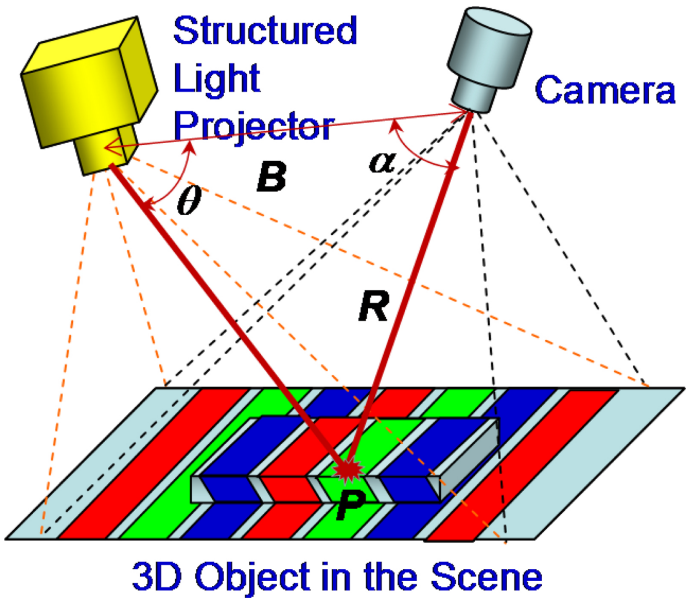

Figure 3.1: Illustration of structured-light 3D imaging [4].

3.2 Homogeneous transformation

Homogeneous transformation is a transformation matrix that transforms coordinates from one reference frame to another. This will be needed to transform the strap coordinates from the camera frame to the robot base frame (see Sec. 5.4). To get the coordinates in the other reference frame, the coordinates are multiplied with

the homogeneous transformation matrix. The homogeneous transformation matrix is made up of a rotational matrix and an offset or translational vector.

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.1}$$

The **R**-matrix will have one row and column for each dimension of the coordinates that will rotate the coordinate system to the new orientation. The **t**-vector adds the offset to the rotated coordinates. The full **H**-matrix in a 3D coordinate system is a four by four matrix.

3.3 Least squares

The desired results are put into the vector $\mathbf{b}^{1\times N}$ and the matrix $\mathbf{A}^{n\times N}$ is made of the known inputs. N is the number of reference points and n is the number of variables used per reference point. The least squares method is used to determine the vector $\mathbf{x}^{n\times 1}$ to minimize the equation [8]:

$$\mathbf{Q} = \sum_{j=1}^{N} (\mathbf{A}_j x - \mathbf{b}_j)^2 = \sum_{j=1}^{N} ((\mathbf{A}_{0j}\mathbf{x}_0 + \mathbf{A}_{1j}\mathbf{x}_1 + \mathbf{A}_{2j}\mathbf{x}_2 + ... + \mathbf{A}_{nj}\mathbf{x}_n) - \mathbf{b}_j)^2 \to 0 \tag{3.2}$$

3.4 Robot Base and TCP coordinate frames

The robot base frame coordinate is shown in Fig. 5.9. The base frame has the z axis pointing out of the page, the x axis pointing towards the left, and the y axis pointing away from the base. The origin of the base frame is at the center of the robot's base.

The TCP coordinate frame is shown in Fig. 3.2. The z axis points out of the page, the x axis points to the left, and the y axis points down. The TCP is centered at the tip of the end-effector. The 3D coordinates and orientation of the TCP need to be specified in PolyScope in millimeters and in radians respectively. Table 3.1 lists the configuration of the TCP pose of the 3D-printed end-effector (see Sec.4.1.4) specified in PolyScope (see Sec.4.2.1).

Pose	**Values**
x	54.88 mm
y	-97.88 mm
z	248.15 mm
Rx	1.5534 rad
Ry	0.2739 rad
Rz	0.2739 rad

Table 3.1: Configuration of the TCP pose.

Figure 3.2: TCP coordinate frame

3.5 Camera coordinate frame

The camera coordinate frame can be seen in Fig. 3.3. The 3D coordinates obtained from the camera are w.r.t it's own coordinate frame, where, the z axis points away from the camera, the x axis points towards the right of the camera and the y axis is pointing into the page. The origin of the coordinate frame is at the center of the camera.

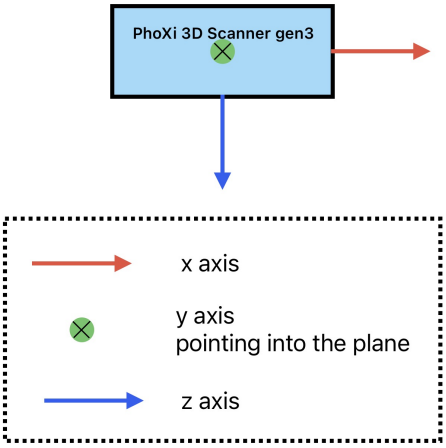

Figure 3.3: Camera coordinate frame

4
System Setup

This chapter introduces the hardware and software components (see Table 4.2 for a brief description), communication protocols, programming languages, system integration and architecture and networking.

4.1 Hardware

To test our development practically, the following hardware components were provided by Volvo:

4.1.1 Cobot - UR10e

A medium duty industrial collaborative robot [19] by UR with the capability to reach 1300 mm, delivering a payload of 12.5 kg [18]. It is a 6-axis cobot, with all six joints as rotating joints. The repeatability of the pose according to ISO 9283 is ±0.05 mm [18]. Table 4.1 provides the name, working range, and maximum speed of each joint of the cobot.

Joint name	**Working range**	**Maximum speed**
Base	±360°	$\pm 120°/s$
Shoulder	±360°	$\pm 120°/s$
Elbow	±360°	$\pm 180°/s$
Wrist 1	±360°	$\pm 180°/s$
Wrist 2	±360°	$\pm 180°/s$
Wrist 3	±360°	$\pm 180°/s$

Table 4.1: Joints information of UR10e [18].

The cobot is equipped with force sensing, tool flange/torque sensor having a precision of ±5.0 N and ±0.2 Nm and an accuracy of ±5.5 N and ±0.5 Nm [18].

4.1.2 SCHUNK Co-act EGP-C 40-N-N-URID

An electric 2-finger parallel gripper certified for collaborative operations with actuation via 24 V and digital I/O [13] by SCHUNK. The gripper has a collision protective cover that helps minimize the risk of injury during use in collaborative operations [13]. We specifically use the URID type of the gripper since it is compatible with UR cobots and can be actuated by connecting it to the tool interface of the cobot

[13]. The stroke of the gripper jaw is 8 mm, the minimum gripping force is 35 N and the maximum gripping force is 140 N [13]. The recommended workpiece weight is 0.7 kg[13].

4.1.3 Camera - Photoneo PhoXi 3D-scanner gen3 & MotionCam-3D Color

The 3D Scanner and MotionCam use structured light (see Sec.3.1) to produce a 3D point cloud and 2D image. The 3D-scanner has a five MP sensor while the MotionCam has a two MP sensor. The cameras come in different sizes for different scanning volumes and the sizes used during this thesis are the large 3D scanner and the large MotionCam. The scanning of the 3D-scanner and MotionCam can be seen in fig. 4.1 and fig. 4.2 respectively. The accuracy of the scans differs depending on the size of the scanner, for the large 3D-scanner it's 0.200 mm [11] and for the large MotionCam it's < 1.25 mm [10].

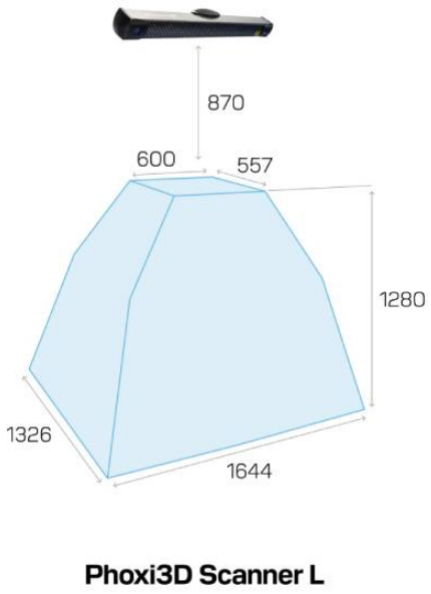

Figure 4.1: Scanning volume of the large Photoneo PhoXi 3D scanner in mm [11].

4.1.4 3D printed end-effector tool

Attached to the SCHUNK Co-act (see Sec.4.1.2 is a 3D-printed gripper see Fig.4.3). The gripper turns the jaws 90 degrees from the SCHUNK Co-act to avoid the robot getting its joints aligned and therefore reaching a singularity pose. The band-aid was applied so that the scanners could see the gripper when calibrating the homogeneous transformation matrix (see Sec.5.4). The triangulation of the three joints in the gripper keeps it stable both, when opened and closed to ensure precise movement. The gripper has a maximum jaw-opening of 15 mm.

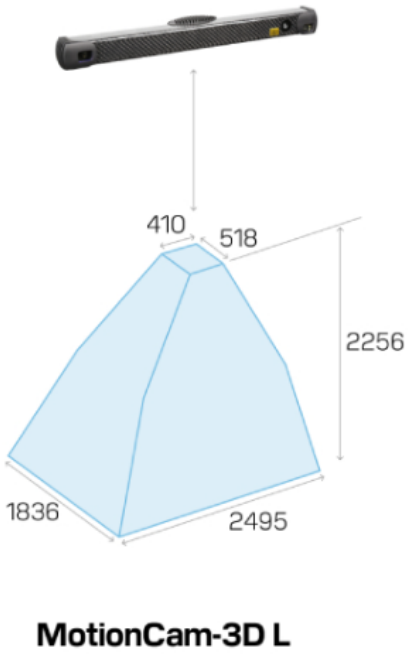

Figure 4.2: Scanning volume of the large Photoneo MotionCam-3D in mm [10].

Figure 4.3: 3D-printed end-effector tool (see Sec. 4.1.4), mounted on the SCHUNK Co-act gripper (see Sec. 4.1.2).

4.2 Software

For hardware to work in the intended way, software plays a crucial role. For development and testing purposes, the following software components were used:

4.2.1 URSim & PolyScope

URSim is a simulation software made for Linux operating system [14]. The Oracle Virtual Box is required for running URSim on another operating system [14]. URSim is used for offline programming and simulation of the robot program [14]. However, there are some limitations to simulate the force mode [14].

PolyScope is a software interface [15] same as that on a physical teach pendant. It allows a user to program, teach, control the robot and much more [15].

4.2.2 URCaps

It is a software extension for Universal Robots, allowing developers to customize and extend functionality of UR robots by wrapping the scripting language into user-friendly programming screens [16].

Schunk has developed a URCap for their gripper which needs to be registered in PolyScope. This URCap adds the functionality of opening and closing of the gripper. The following functions enable us to actuate the gripper once the URCap has been registered successfully.

- *Open Co-act EGP-C* - Opens the gripper.
- *Close Co-act EGP-C* - Closes the gripper.

4.2.3 Aurora Design Assistant (ADA)

The software ADA is a machine vision application [23] and is used to process the captured scans in order to extract the number of straps, position and colour. The software is based on flowcharts making it easy to program but some lines of code are still necessary for this thesis.

Functions offered by the ADA software that are used are:

- Image filter - an image filter called "vertical" is applied in order to ease the edge location step.
- Edge location - It evaluates how quickly and how much the image changes across a specified span of pixels. It will mark something as an edge if the change is quick and large enough.
- Align plane - Using four areas it rotates the camera coordinate frame to specify the z direction as being perpendicular to the average surface of the four areas.
- Depth filtering - When using the 3D-coordinates it can filter out points to display only points between selected z values (see Fig. 5.3).
- Position - For every point it can return the 3D-coordinate of said point.
- RGB and luminance value - For every point the Red, Green, Blue and Luminance value can be returned and also the average value over a selected area.
- Inverse Transformation - For 3D-coordinates obtained after the function Align plane has been used, a transformation back into the camera coordinate frame is required.

4.2.4 Oracle VirtualBox

A cross-platform virtualization application that enables the execution of multiple operating systems within multiple virtual machines (VMs) on an existing computer [9]. Almost always, computers are never used to their full potential, resulting in unused hardware resources [9]. Virtualization plays an important role and can significantly reduce hardware and electricity costs [9].

The purpose of using Oracle VirtualBox was to install URSim (see Sec. 4.2.1), a disk image based on Linux operating system. Since the host PC has windows, URSim had to be installed on a virtual machine in VirtualBox. Thus, simulation-based development was possible since URSim hosts PolyScope.

4.3 Communication Protocols

Communication protocols enable hardware and software to transmit data to each other. Physically integrating hardware and software is one thing, but enabling them to communicate is another. The following communication protocols are used in this project:

4.3.1 TCP Sockets

A socket is a programming interface developed by the University of California at Berkeley to make network programming easy [24]. TCP sockets use the TCP protocol, a type of transport protocol that is reliable and commonly used [24].

A TCP server based on the TCP socket was created using Python (see Sec.4.4.1 and 5.3). The ADA (see Sec.4.2.3, 5.2.1 and 5.2.2) connects to this server, thus enabling bi-directional data transfer. Data over TCP sockets are sent and received in the form of bytes.

4.3.2 XML-RPC

A Remote Procedure Call (RPC) method that uses HTTP as the transport layer along with XML to encode function calls and their parameters. In this project, Python (see Sec.4.4.1) acts as an XML-RPC server, making available functions (see Sec.5.3) that the UR controller can invoke. The UR controller is an XML-RPC client.

4.4 Programming Languages

Once hardware and software are able to communicate, it becomes necessary to encode and decode data. This is important for human readability and actions to be triggered as and when required. The following programming languages are introduced:

4.4.1 Python

An interpreted, interactive, object-oriented [12], high-level programming language developed by Guido van Rossum [21]. Python is popularly used for web-development (server-side), software development, mathematics [21], and much more.

Python is used as a middle-man in this project i.e, data exchange and mathematics computation. It hosts two servers, one using TCP sockets and the other using the XML-RPC protocol. It is also used to log the data we receive, compute the homogeneous transformation matrix of the camera with respect to the robot, transform the coordinates of the strap into the robot frame and keep sequential flow of logic.

4.4.2 URScript

A custom programming language, developed by Universal Robots, that controls the robot arm [20]. Any interactions with PolyScope are converted into URScript commands and sent to the robot for execution [20].

In PolyScope i.e., in URSim and the teach pendant, the script functionality is used to code the URScript commands. URScript commands are used to connect to the XML-RPC server and to invoke functions using the XML-RPC protocol.

Hardware	**Software**	**Purpose**
Cobot - UR10e Robot	• Simulation: URSim(PolyScope). • Physical Implementation: PolyScope on teach pendant.	Automate the removal of straps from pallets: • To validate in simulation that the robot reaches the strap coordinates successfully. • To only hold the strap and sort it during physical implementation.
SCHUNK Co-act EGP-C 40-N-N-URID	URCap	A gripper to hold the strap. A Plugin that adds functionality for actuation of gripper.
Camera - PhoXi 3D Scanner gen3	Aurora Design Assistant 10.7 (ADA)	Image capturing and processing for object identification i.e., • Strap identification. • Colour and pose estimation of the strap.
Ethernet switch	-	A physical interface to enable connection of multiple devices using RJ45 to RJ45 cable.

PoE (Power over Ethernet) Injector	-	A physical interface module • To power the camera. • Transmit data from camera to PC and vice versa.
M12-X to RJ45 cable	-	To physically interface the PoE Injector and camera.
RJ45 to RJ45 cable	-	To physically interface - • The PoE Injector with the Ethernet switch. • The UR controller with the Ethernet switch. • The Ethernet switch to PC/Laptop.
-	Python	To setup a TCP Server using sockets for the camera and an XML-RPC server for UR.
-	Oracle VirtualBox	To host the URSim. Enables us to simulate the robot.

Table 4.2: List of hardware and software components and their purpose.

4.5 System Integration and Architecture

The approach in this project was to first simulate the robot in URSim in Oracle VirtualBox along with the ADA loaded with pre-taken photos of the pallets and with Python hosting the servers. This helped gain a concrete understanding of the behavior of the entire system. This then allowed us to implement the same but practically, with real-time photographs and a UR10e cobot, thus validating our development. We present two almost identical system architectures, one for simulation purposes and the other for physical demonstration.

4.5.1 Simulation

Fig. 4.4, shows the system integration and architecture for simulation purposes. The PhoXi 3D scanner gen3 is connected to a PoE injector using a M12-X to RJ45 cable. Through this cable, the PoE injector powers the camera. The cable also supports data transmission. An RJ45 to RJ45 cable is used to connect the PoE injector to the PC. In the PC, ADA executes the image processing program, the Python script hosts the two servers such that data can be transmitted between them easily. Also, the URSim (PolyScope), a disk image, is hosted by the Oracle VirtualBox on the PC.

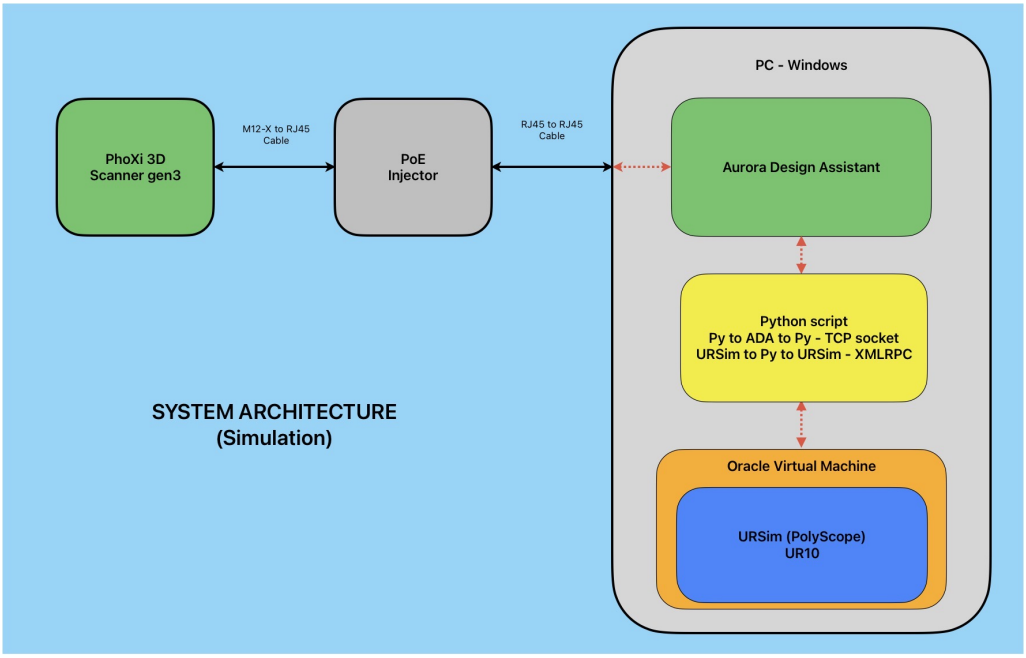

Figure 4.4: System Architecture - Simulation.

4.5.2 Physical Implementation

Fig. 4.5, shows the system integration and architecture for physical implementation purposes. The camera and the PoE injector are interfaced in a similar manner as described before. The PoE injector is connected to an Ethernet switch via an RJ45 to RJ45 cable. The UR controller has three connections, one with the UR10e cobot, another with the teach pendant (PolyScope) and to an Ethernet switch using an RJ45 to RJ45 cable. The PC is then connected to the Ethernet switch with an RJ45 to RJ45 cable. The ADA for image processing and the Python script hosting the two servers are executed.

4.5.3 Software Integration

Fig. 4.6, shows a block diagram of software integration. Developed programs for the respective methods (mentioned in sec. 5.1) are loaded into the teach pendant via a USB stick or into URSim. At a time, only one of these programs can be executed. The **main()** function invokes functions of two sub-programs in a sequential manner. The two sub-programs that we have developed are:

1. UR to Py & vice versa - XML-RPC: is used for server initiation and registering functions and function calls that the UR controller or URSim can invoke. Thus, communicating with physical or virtual robot.
2. ADA to Py & vice versa - TCP: used for initiation of server based on TCP socket to send a "Take image flag" to trigger the camera for image acquisition when requested by the UR controller or URSim and to obtain information of the straps after image processing. Thus, communicating with the camera via ADA IDE.

In ADA IDE, we can load the image acquisition and processing programs for the

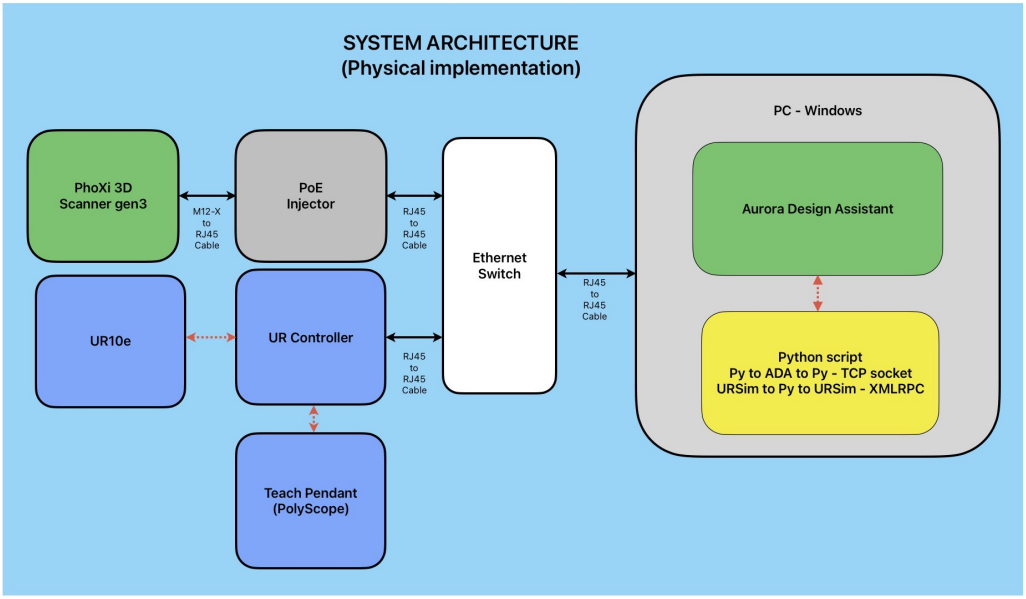

Figure 4.5: System Architecture - Physical Implementation.

respective methods. At a time only one program is executed.

4.6 Networking

Networking plays a key role in automation. Without network, the devices will not be able to connect with each other as they would not know exactly which device to talk to. A LAN network without the internet was set up. Static IP addresses were assigned to the PC and the robot. Oracle VirtualBox auto assigns IP addresses to the computer and the virtual machine when in the network settings either as "Host-only Adapter" or "Bridged Adapter" options are selected. The camera is auto-recognized by ADA by the device name or ID. Table 4.3 shows the IP addresses that were set.

Device	**IP address**	**Set by**
Cobot - UR10e Robot	192.168.100.100	Set manually using the teach pendant.
PC or Laptop (Physical Ethernet port)	192.168.100.1	Set manually.
PC or Laptop (Virtual Ethernet)	192.168.56.1	Set by Oracle VirtualBox.
URSim	192.168.56.101	Set by Oracle VirtualBox.

Table 4.3: List of devices and their IP addresses.

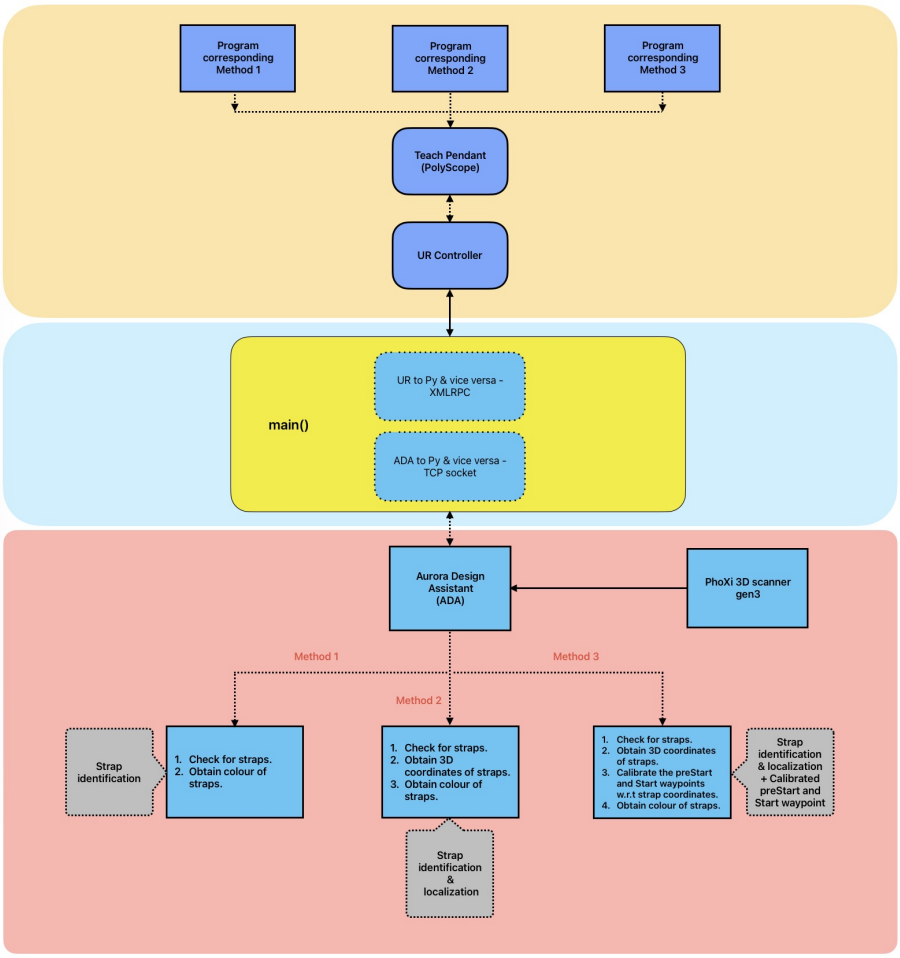

Figure 4.6: Block Diagram of software integration.

5
Methods

In this chapter, the topics discussed are de-strapping methods using vision-based robotics, image processing using ADA, the main Python script, estimation of the homogeneous transformation matrix, robot program, strap detection and verification, simulation in URSim, installation of safety features and the practical setup.

5.1 Methods for de-strapping via vision-based robotics

Volvo wants to evaluate the camera on the basis of whether a simple camera or an advanced camera such as the Photoneo PhoXi scanner gen3, would be useful for the application scenario. The methods presented are sequential development of the presented system to better evaluate the technology. The three methods are presented below and the results are presented in Sec. 7.

1. Method 1: Strap identification & manually taught waypoints.
2. Method 2: Strap identification & localization with detection of the box via robot in force mode.
3. Method 3: Strap identification & localization with the *preStart* and *Start* waypoints being calibrated w.r.t the strap coordinates.

A very short and concise description of the three methods is shown in fig. 5.1.

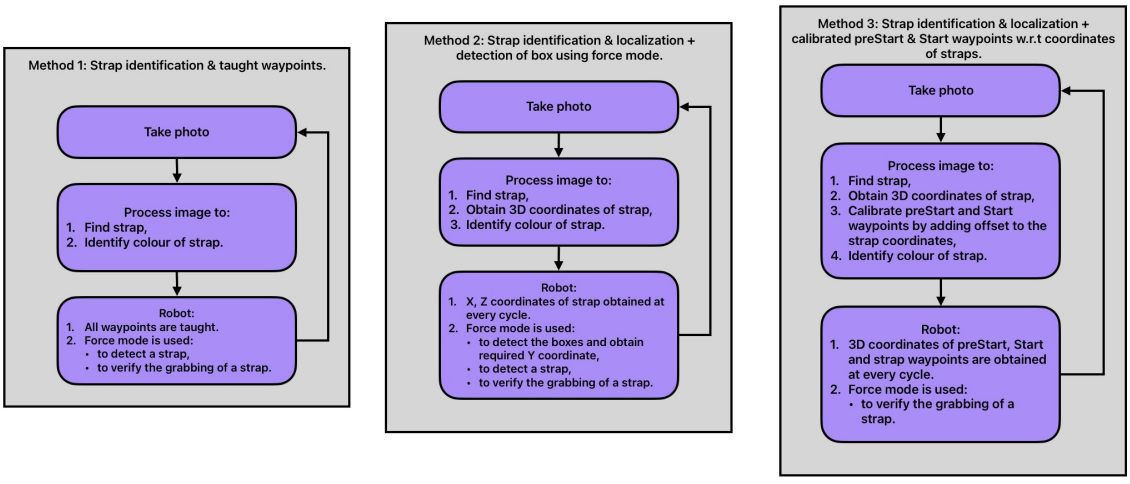

Figure 5.1: Comparison of the three proposed de-strapping methods, briefly highlighting the image processing and robot programming steps.

5.1.1 Method 1: Strap identification & manually taught waypoints

In this method, a 2D image is obtained from the camera and processed in ADA IDE. The image processing flowchart in ADA IDE (described in Sec.5.2.1) is designed to identify the straps and the colour of the straps. The information related to the straps is then sent to the TCP server as a list of strings (see Sec.5.3). Once the UR controller or URSim requests for this data, it is sent in the form of a list of integers. The waypoints for the robot are taught manually using the teach pendant or freedrive feature. Thus, the robot moves to the corresponding waypoints only (see. Sec.5.5 and Sec.5.5.1).

Once the robot reaches the *Detect* waypoint, it enters into force mode. In this mode, the robot is instructed to apply a force in a particular direction. There are two force applications (see Sec.5.6) developed namely:

- Detect Strap.
- Verify Strap Grab.

5.1.2 Method 2: Strap identification & localization with detection of the box via robot in force mode

A 3D image is captured using the camera, and the point cloud is processed in ADA IDE (see Sec.5.2.2) to identify the strap and obtain the 3D coordinates of the strap in the camera frame. Once the strap has been localized in the 3D image, the 2D coordinates are used to localize the strap in the 2D image for colour recognition. The data is transmitted to the TCP server in the form of a list of strings.

The robot is manually taught a *preStart* waypoint. Once the robot reaches this waypoint, using the force mode, the robot is made to move in the y direction (refer to fig. 5.9 for coordinate directions) until it detects a box. Once a box is detected, the robot retreats by 12 mm. The retreat is done so that the y coordinate can be used since the depth functionality of the camera will not be used (even though the 3D coordinates of the straps are available).The UR controller or URSim then requests for the strap data and its waypoints (see Sec.5.5 and Sec.5.5.1).

When the UR controller or URSim makes a request for the strap coordinates, the current TCP pose is sent to the Python script (see Sec.5.3). Thus, from the current TCP pose we obtain the y coordinate and the orientation of the end-effector so that the next waypoint can be constructed. The data and the waypoint of the straps (x, z coordinates of the strap after homogeneous transformation from camera to robot frame, along with the y coordinate and the orientation of the end-effector from the current TCP pose) are sent. Thus, two less manually taught waypoints and more dependency on the force-torque sensor of the robot.

Once the robot reaches the strap waypoint, it enters into force mode (see Sec.5.6) and carries out the:

- Detect Straps and,
- Verify Strap Grab

functionalities.

5.1.3 Method 3: Strap identification & localization with the *preStart* and *Start* waypoint being calibrated w.r.t the strap coordinates

Again, a 3D image is captured using the camera, and the point cloud is processed in ADA IDE (see Sec.5.2.2) to identify the strap and obtain the 3D coordinates of the strap in the camera frame. Once the strap has been localized in the 3D image, offsets are added to obtain calibrated *preStart* and *Start* waypoints. The 2D coordinates are then used to localize the strap in the 2D image for colour recognition. The data is transmitted to the TCP server in the form of a list of strings.

The robot is solely dependent on the camera & image processing for the waypoints now i.e., the *preStart*, *Start* and *Strap* waypoints. In the Python program , for each type of waypoint, a function has been created (see Sec.5.3). The UR controller or URSim then requests for the strap data and the waypoints.

When the UR controller or URSim makes a request for the waypoints (see Sec.5.5 and Sec.5.5.2), depending on the type of waypoints, the following takes place:

- Request for *preStart* or *Start* waypoint: the UR controller or URSim must send the current TCP pose and a number that specifies which *preStart* or *Start* waypoint it needs. Only in the case of the *preStart* waypoint, the orientation of the end-effector is hard-coded such that the gripper is oriented in the right manner. But from the *Start* waypoint onwards, the orientation is obtained via the tcp pose.
- Request for *Strap* waypoints: the UR controller or URSim only needs to send the current TCP pose.

All waypoints are transformed from the camera coordinate frame to the robot base frame. When the robot reaches the strap waypoint, to make sure it has grabbed the strap, it enters the force mode and performs the "Verify Strap Grab" function (see Sec.5.6). Thus, this method is designed to ensure flexibility and save time i.e., cycle time.

5.2 Image processing using ADA

The image processing is done by a software called *Aurora Design Assistant* (ADA). The team received training on the software by professionals and have the access to the software manual. Also, the professionals provided support as and when contacted.

5.2.1 Method 1

Describing the events of Method 1 displayed in figure 5.2:

- **Start:** The program starts
- **Connect to Python TCP server:** The program tries to connect to the TCP server.
- **Check connection:** The connection is checked before continuing.
- **Send "Method choice":** The selected method is communicated to the server so it knows what information will be sent.
- **Read "Take Image flag":** Here is where the loop starts by the UR program prompting the TCP server to tell the ADA software to take an image.
- **Check Take Image flag:** It verifies that the software was able to take an image.
- **Add filtering to the image:** A filter is added to make vertical edges more distinct to make the straps easier to locate.
- **Find straps:** Straps are found by locating a measurement between two edges that are within certain distance from one and another. It does this once for the nearest box and once for the box furthest away and stores the number of straps and their location. Depending on the number of straps and their location the program selects a path that does the same thing with some small details and thresholds difference.
- **Read RGBHL values with respective colour thresholds:** In relation to where the strap(s) are it analyses the Red, Green, Blue, Hue and Luminance (RGBHL) values of a predefined area over the strap(s).
- **Compare RGBHL values with respective colour thresholds:** By comparing all the values it sorts the straps as White, Blue, Green or Black.
- **Store the info and colour of straps:** It stores the number of straps, what side of the pallet they are and what colour they are.
- **Send data to Py:** The data stored in the previous step is now transmitted to the Python program as an array.

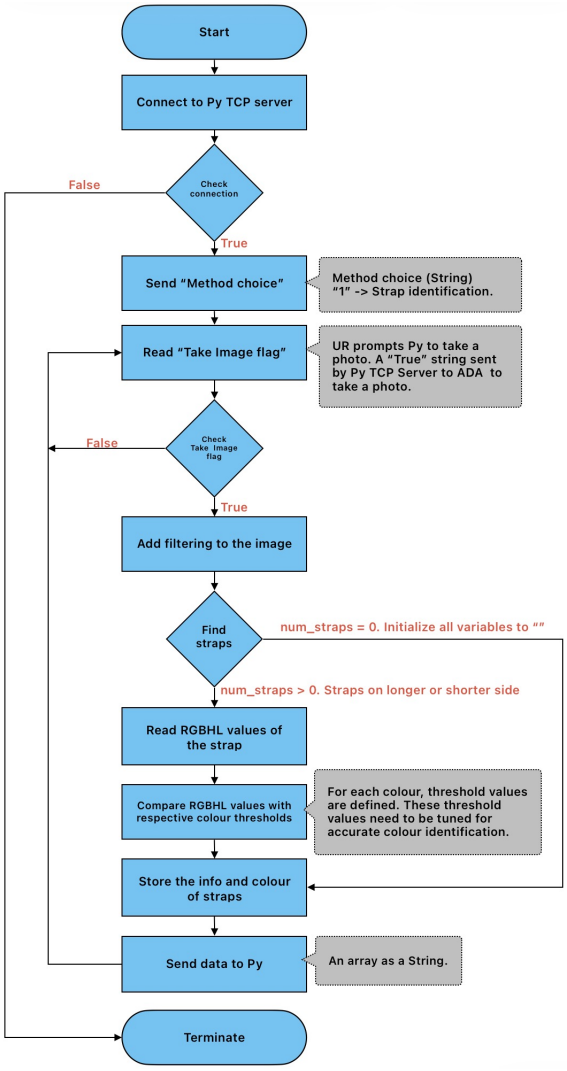

Figure 5.2: Flowchart of image processing for Method 1 in ADA.

5.2.2 Method 2 and 3

Describing the events of Method 2 and 3 displayed in figure 5.4 and 5.5:

- **Start:** The program starts
- **Connect to Python TCP server:** The program tries to connect to the TCP server.
- **Check connection:** The connection is checked before continuing.
- **Send "Method choice":** The selected method is communicated to the server so it knows what information will be sent.
- **Read "Take Image flag":** Here is where the loop starts by the UR program prompting the TCP server to tell the ADA software to take an image.
- **Check Take Image flag:** It verifies that the software was able to take an image.
- **Add filtering to the image:** A filter is added to make vertical edges more distinct to make the straps easier to locate.

- **Find straps:** Straps are found by locating a measurement between two edges that are within certain distance from one and another. It does this once for the nearest box and once for the box furthest away and stores the number of straps and their location. Depending on the number of straps and their location the program select a path that does the same thing with some small details and threshold differences.
- **Align the image plane to the boxes:** In relation to where the strap(s) are four areas surrounding the strap are taken and a new plane is made with z direction being perpendicular to the new plane.
- **Filter out the boxes to get precise straps:** Due to the z direction being perpendicular to the boxes it can filter out the boxes and only leave the straps (see Fig.5.3).
- **Obtain 3D coordinates of straps in aligned plane:** Now without a background the software can get a robust and precise 3D coordinate of the straps.
- **(Method 3) Calibrate *preStart* and *Start* waypoints:** An offset is added to the straps coordinates to give a *preStart* and *Start* waypoint to give the robot a valid approach.
- **Transform the 3D coordinates to camera frame:** The coordinates of the strap and in Method 3's case the preStart and Start waypoint, are transformed back from the aligned plane frame of reference to the cameras coordinate frame of reference.
- **Read RGBHL values with respective colour thresholds:** In relation to where the strap/s are it analyse the Red, Green, Blue, Hue and Luminance (RGBHL) values of a predefined area over the strap/s.
- **Compare RGBHL values with respective colour thresholds:** By comparing all the values it sorts the straps as White, Blue, Green or Black.
- **Store the info and colour of straps:** It stores the number of straps, what side of the pallet they are, their 3D-coordinates, in Method 3 also stores the 3D-coordinates of the *preStart* and *Start* waypoint and what colour they are.
- **Send data to Py:** The data stored in the previous step is now transmitted to the Python program as an array.

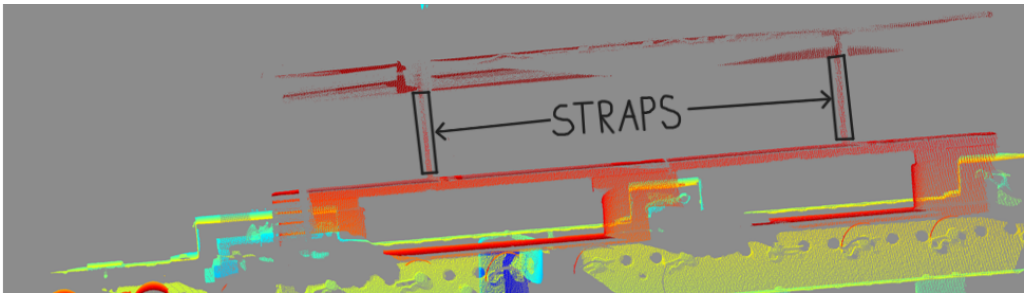

Figure 5.3: 3D point cloud of pallet with straps after depth filtering.

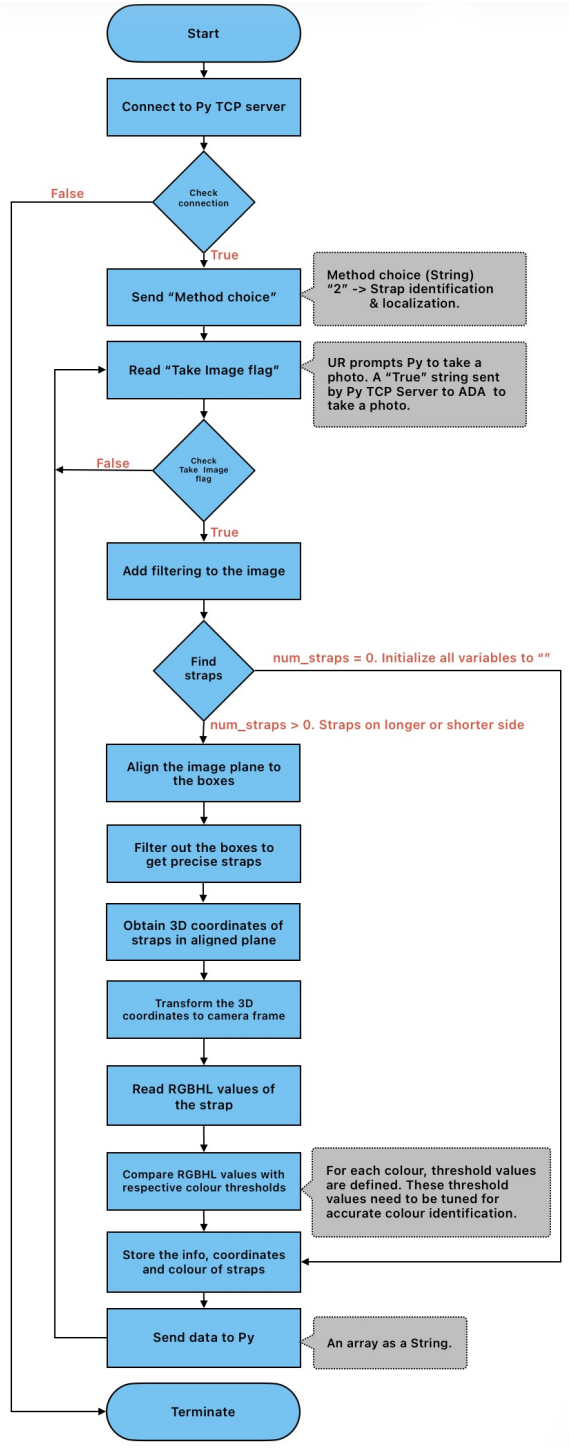

Figure 5.4: Flowchart of image processing for Method 2 in ADA.

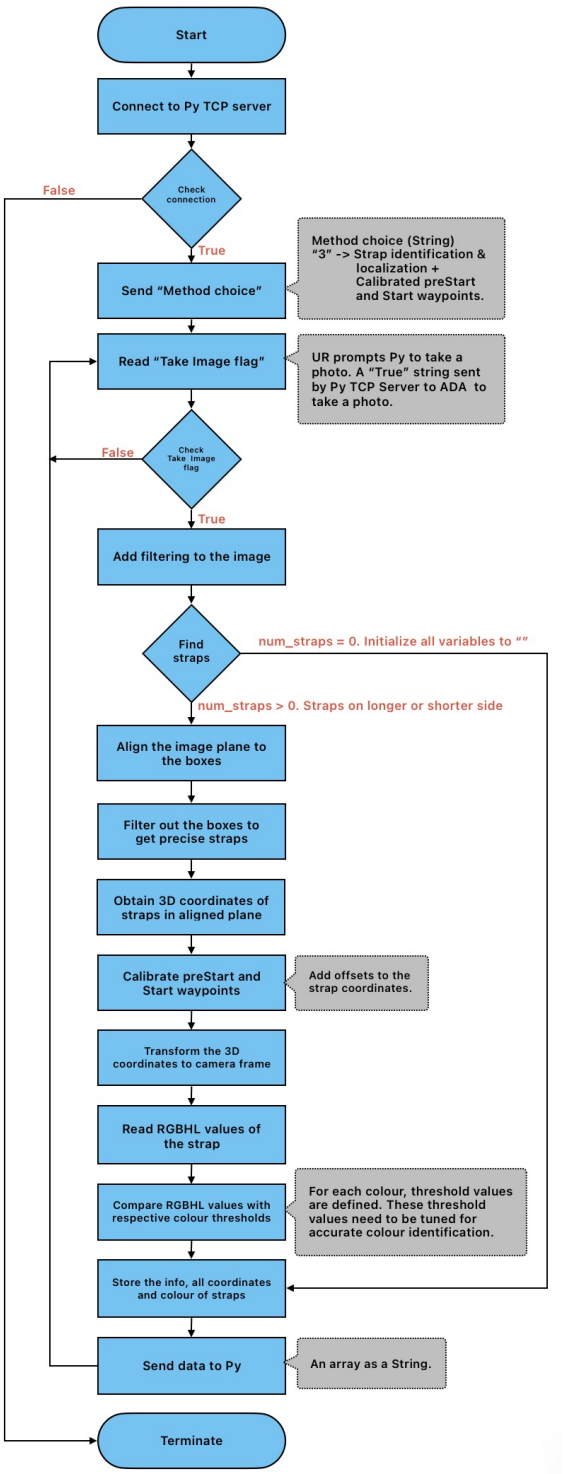

Figure 5.5: Flowchart of image processing for Method 3 in ADA.

5.3 Flowchart of main Python script

The **main()** in Python is responsible for being a necessary link between the ADA IDE and the UR controller. The **main()** invokes functions that set up the logging functionality, initiate servers, decode data, load files, and registers the function call statements for the XML-RPC protocol. Figure 5.6 shows the flowchart of the Python program. Upon executing the script,

1. A *try and except* block is used. The try and except block is a very useful way of coding. It helps in catching all exceptions, especially the unexpected ones, for debugging purposes. The try block attempts to execute the tasks within its body, but as soon as an error is encountered, the except block catches the error.
2. If any error is encountered, the *comms_flag* is checked first. If a connection between the Python script, camera and UR exists, the flag will result in being true and the program needs to exit by first shutting down the servers and closing the TCP server. If an error occurs before any connection was established, the *comms_flag* would be false and the program would exit directly.
3. The *setup_logs()* function creates a log file in which all important logs or events can be written. For example, the server is initiated, a client is connected, or an error has occurred. Log files are useful for easily decoding or resolving issues that can occur in the longer run.
4. The *initiate_XMLRPCServer()* function starts the XML-RPC server hosted by Python so that the UR controller can establish a communication with the Python script. Once the server is created, the *xmlrpc_server_flag* is set to **True**.
5. The *initiate_comms()* function is used to start the TCP server, hosted by Python. It waits for the ADA to connect to the TCP server. Once the server is up and the connection with ADA is successful, the boolean flags - *cam_conn_flag* and the *comms_flag* are set to **True**.
6. If the connection is not successful, mostly because ADA is unable to connect, a timeout will occur and it will throw an error.
7. If for some reason, the boolean flags *xmlrpc_server_flag* and *cam_conn_flag* are not assigned a **True** value, an exception is raised.
8. Once, *xmlrpc_server_flag* and *cam_conn_flag* are checked to be **True**, the *determine_method* function is called. The ADA sends the method number i.e., 1 for method 1 or 2 for method 2 or 3 for method 3, based on the program the user has loaded in ADA IDE and is in execution. To the *determine_method(camera_conn)* function, the connection object *camera_conn* needs to be passed as an argument. It then reads the TCP buffer and decodes the method number. Also, it initializes a boolean variable called *methodError_flag*, based on if the user has opted for the right method.
9. Based on the choice of method, the homogeneous transformation matrix is needed. If the method choice is 2 or 3, the matrix which is stored as a .csv file is loaded. This is taken care by the function *load_HT(method_choice)* to which the *method_choice* variable is passed as a parameter.
10. An if statement is used to check for the negation of *methodError_flag* to ensure

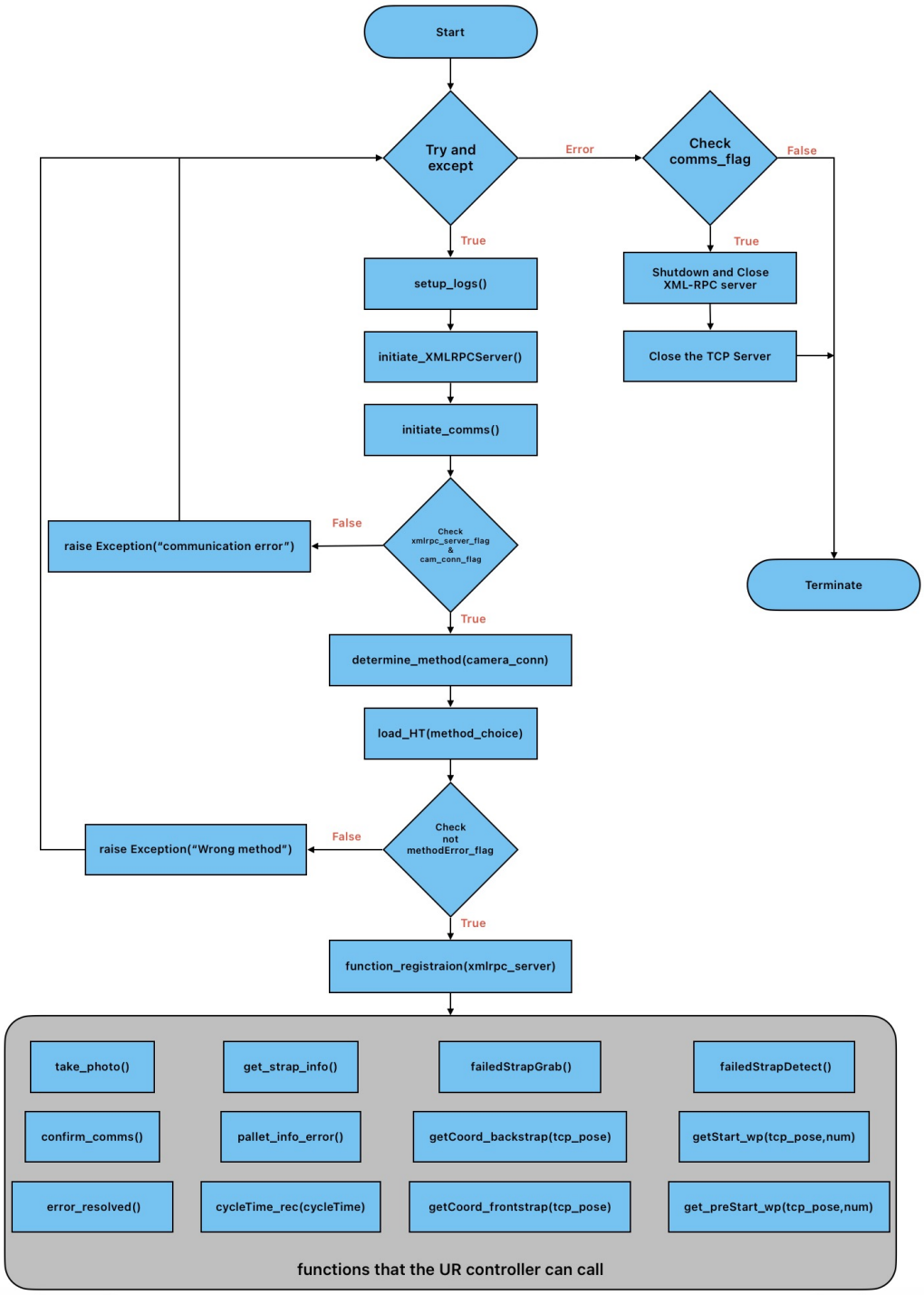

Figure 5.6: Flowchart of main Python script.

that the user has not made a mistake in choosing the right method. If the *methodError_flag* is false, its negation is true.

11. On the contrary, if the *methodError_flag* is true, its negation is false, an exception is raised and the program is terminated by first shutting down the servers.
12. Once all necessary flags have been scanned, the method *function _registraion*

(xmlrpc_server) is called. This function makes available all the functions in the gray box as shown in fig. 5.6 so that it can be called by the UR controller or URSim using the XML-RPC protocol.

The functions that are made available for the UR controller or URSim for calling using the XML-RPC protocol are:

- *confirm_comms()*: This function is called at the beginning of every cycle, to ensure that communication is established. It returns a **True** flag. If there is a communication problem between Python and UR controller, the function will not return a value thus leading to a communication failure.
- *take_photo()*: In every cycle, the UR controller calls this function such that Python then has to pass **True** as a string to ADA IDE in order to take a photo of the pallet.
- *get_strap_info()*: Once the UR controller or URSim invokes this function, Python needs to read the TCP buffer and decode the data sent by ADA IDE to it. This data, after decoding is then sent to UR controller or URSim.
- *pallet_info_error()*: This function is invoked when a distinction needs to be made between two possible options. One, there are no straps on the pallet and two, there is an error in the data received from ADA IDE. The function uses an existing variable - *info_error_flag*, a boolean value is assigned based on the length of the string received from ADA. The value assigned to *info_error_flag* is returned to the UR controller or URSim. The *info_error_flag* can either be **True** if the data is not of the right length i.e., corrupted or can be **False** indicating the data is correct and that there are no straps on the pallet.
- *failedStrapDetect()*: This function is invoked if the robot in force mode fails to detect a strap.
- *failedStrapGrab()*: This function is called if the robot in force mode fails to verify that it has grabbed a strap.
- *get_preStart_wp(tcp_pose, num)*: Only when using the third method, the *preStart* waypoint is obtained by calling this function. The UR controller or URSim needs to send the TCP pose and the number (num) while calling the function. The parameter num takes either 1 - *preStart* waypoint for the front strap or 2 - *preStart* waypoint for the back strap. Thus, the function returns the respective *preStart* waypoint.
- *getStart_wp(tcp_pose, num)*: Only when using the third method, the *Start* waypoint is obtained by calling this function. The UR controller or URSim needs to send the TCP pose and the number (num) while calling the function. The parameter num takes either 1 - *Start* waypoint for the front strap or 2 - *Start* waypoint for the back strap. Thus, the function the respective *Start* waypoint.
- *getCoord_frontstrap(tcp_pose)*: This function is invoked if either the program corresponding to the second method or the third method is executed. The TCP pose is passed to the function call statement. An important distinction between method 2 and method 3 while invoking this function is, method 2 only uses of 2D coordinates of the strap after homogeneous transformation (x and z) of the 3D coordinates. The y coordinate along with TCP orientation is taken from the *tcp_pose* (see Sec.5.1.2). Method 3 however, makes complete

use of the 3D coordinates obtained after homogeneous transformation, only the orientation of the end-effector is taken from the *tcp_pose* (see Sec.5.1.3). Thus, the function then returns the *frontstrap* waypoint.

- *getCoord_backstrap(tcp_pose)*: This function is invoked if either the program corresponding to the second method or the third method is executed. The TCP pose is passed to the function call statement. An important distinction between Method 2 and Method 3 while invoking this function is, Method 2 only uses of 2D coordinates of the strap after homogeneous transformation (x and z) of the 3D coordinates. The y coordinate along with TCP orientation is taken from the *tcp_pose* (see Sec.5.1.2). Method 3 however, makes complete use of the 3D coordinates obtained after homogeneous transformation, only the orientation of the end-effector is taken from the *tcp_pose* (see Sec.5.1.3). Thus, the function then returns the *backstrap* waypoint.
- *error_resolved()*: If the data being transmitted to the robot is corrupt because data obtained from ADA was corrupt, a pop-up on the teach-pendant of the robot needs to be acknowledged by the supervisor stating the issue has been resolved. Once this pop-up is acknowledged, this function is called and the operation resumes.
- *cycleTime_rec(cycleTime)*: Before a new cycle starts, the cycle time of the current cycle is sent to Python for logging purposes.

5.4 Estimation of homogeneous transformation matrix

To calculate the homogeneous transformation matrix (3.2), the NumPy library's linear algebra function *lstsq* will be used to obtain the most optimal solution for $\mathbf{Ax} = \mathbf{b}$ (see Sec.3.3). Where $\mathbf{A}$ is a matrix with the points in the camera's frame of reference and $\mathbf{b}$ is a vector with the points in the robot's frame of reference. The function returns the vector $\mathbf{x}$ that will make the 3 by 4 homogeneous transformation matrix. The estimated homogeneous transformation matrix is saved as a .csv file.

$$\mathbf{A} = \begin{bmatrix} p_{1x}^C & p_{1y}^C & p_{1z}^C & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{1x}^C & p_{1y}^C & p_{1z}^C & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{1x}^C & p_{1y}^C & p_{1z}^C & 1 \\ p_{2x}^C & p_{2y}^C & p_{2z}^C & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{2x}^C & p_{2y}^C & p_{2z}^C & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{2x}^C & p_{2y}^C & p_{2z}^C & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{5.1}$$

$$\mathbf{b} = \begin{bmatrix} p_{1x}^R \\ p_{1y}^R \\ p_{1z}^R \\ p_{2x}^R \\ p_{2y}^R \\ p_{2z}^R \\ \vdots \end{bmatrix} \tag{5.2}$$

$$\mathbf{x} = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{14} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{24} \\ h_{31} \\ h_{32} \\ h_{33} \\ h_{34} \end{bmatrix} \tag{5.3}$$

5.5 Flowchart of Robot program

It is essential to have a sequential cyclic execution in this project since each function needs to be called one after the other, no parallelism is required. However, a thread is used to constantly acquire the force reading using the force-torque sensor of the cobot. The cobot has been programmed using the PolyScope interface on the teach pendant. The cobot via its controller is responsible for initiating the process of taking a photo, requesting for data and moving to the necessary positions to detect or hold the straps.

The robot program is divided into three parts, one that is executed only once and at the beginning of every new execution of the program (blue boxes shown in yellow background), the second part is an endless iterative loop (blue boxes shown in green background) that is called the main robot program and the third is the thread (blue boxes shown in red background) that runs in parallel with the main robot program. Fig.5.7 and 5.8 highlights the flow of control for Method 1, 2 and 3 respectively.

Before explaining the difference between the programs for Method 1, 2 and 3, the common parts of the programs are explained here. On execution of the program:

1. The blues boxes in the yellow background are executed first. This happens whenever the program is started from scratch.
2. Then, the flow splits in two, executing a thread and the main robot program simultaneously. In the thread, the force readings are obtained from the force-torque sensor of the robot and a *sync()* function is used to avoid real-time errors.
3. The robot is instructed to move to the **Home** position. The **Home** position is a safe position for the robot to ensure that no damage occurs to the cobot, the operators, or other equipment around it, when not in use.
4. The gripper is actuated to open so that the jaws can be used to detect or hold and grab the straps (see Sec.5.6).
5. The UR controller then attempts to connect to the XML-RPC server hosted by the Python program (see Sec.5.3).

6. The cycle time is reset to zero at the start of every cycle.
7. Since we are in the iterative part of the program, the robot must be moved to **Home** before it can request for a photo to be taken. Thus, the camera will have a clear visual of the pallet. Not only this but also for each cycle, the robot should start from a safe position.
8. The controller then invokes a function *confirm_comms()* (see Sec.5.3) that is available via XML-RPC protocol. The function returns a boolean flag associated with the variable *comms_flag* in the main Python script.
9. Once the robot is at **Home** and communication is healthy, the cycle time is initiated.
10. The *take_photo()* (see Sec.5.3) is invoked to enable Python to send a **True** string to ADA IDE for the sole purpose of triggering the camera to take a photo.
11. A wait function, delaying or pausing the execution by 0.01 seconds is necessary to ensure a good sync between calls and data transfer.
12. To obtain the information of the straps, the controller needs to invoke *get_strap_info()* (see Sec.5.3). The strap information arranged in a list of integers and passed to the controller.
13. Based on the where the strap(s) are present on a *pallet_side*, the program's flow splits into three cases. A switch case is used to decide the action to be performed based on the integer corresponding to *pallet_side*. The following cases are:
    - Case 1 - Longer side of pallet, (refer Fig.5.9 to understand which is the longer side of the pallet)
    - Case 2 - Shorter side of pallet, (refer Fig.5.9 to understand which is the shorter side of the pallet)
    - Default Case - No straps / Error.
14. If the default case is chosen, it is either there are no straps or an error has occurred. To differentiate between the two, a function *pallet_info_error()* (see Sec.5.3) is called and it returns a boolean value.
    - If the boolean value returned by the function is **True**, it indicates an error has occurred and a pop-up on the teach pendant saying, "Error, a supervisory action is needed," appears. Once the supervisor resolves/acknowledges the error, the pop-up needs to be toggled and the *error_resolved()* (see Sec.5.3) is called thus, resuming operations.
    - If the boolean value returned by the function is **False**, it indicates there are no straps present on the pallet. A pop-up saying "No straps" appears on the teach-pendant. Once the pop-up is acknowledged, the operation resumes.
15. If straps are on the shorter side of the pallet, case 2 is executed. A pop-up stating "straps on the shorter side appears". Once acknowledged, the operation resumes.
16. If straps are present on the longer side, irrespective of which strap it is i.e., front or back (see Fig.5.9), the common functions *Operator_cut_strap*, *Pull-Out_strap*, Feed_Strap and *cycleTime_rec(cycle_time)* are executed.
17. Due to safety concerns, a gripper or end-effector tool with a sharp object

was not allowed to be used for this thesis. The function *Operator_cut_strap* initially had a pop-up to tell the operator to cut the strap but to replace it and estimate a more realistic cycle time, the controller opens the gripper, moves the robot away from the strap and closes the gripper once again.

18. The function *PullOut_strap* is called to show how the robot would pull strap off from the pallet. It has a taught waypoint called *PullOut* that the robot needs to move to.
19. After the robot has reached the *PullOut* waypoint, the *Feed_Strap* function is called. The robot feeds the strap to an imaginary feeder. A pop-up appears asking the operator if the strap has been fed to the feeder, once acknowledged, the robot move to a *Away_feed* waypoint.
20. Once all the necessary functions have been performed, the cycle time is sent to Python by invoking the call statement - *cycleTime_rec(cycle_time)* (see Sec.5.3 to read about the cycle time function in Python).

5.5.1 Robot Program for Method 1 and 2

This section will delve deeper into case 1 of the switch case - longer side of the pallet. Although the flowchart of the robot program for Method 1 and 2 (shown in Fig.5.7) is the same, there are some differences in the functions created in the robot program for Method 1 and 2.

1. Depending on the number of straps located on the longer side of the pallet and if it is the front strap or the back strap or both (see Fig.5.9 more intuition for side of pallet and strap position), the flow of the program will execute the necessary blocks.
2. The two functions *Longside_FrontStrap* and *Longside_BackStrap*, as the name suggests, have waypoints for robot to move towards the respective straps. But there is a subtle difference in these functions for Method 1 and 2.
   - Since Method 1 deals with waypoints that are taught to the robot, both *Longside_FrontStrap* and *Longside_BackStrap* have the taught waypoints only. (see Sec.5.1.1 to read about Method 1.)
   - In the case of Method 2, both the functions have one less taught waypoint each and it is replaced with one functionality. Also call statements to respective XML-RPC functions - *getCoord_frontstrap(tcp_pose)* and *getCoord_backstrap(tcp_pose)* (see Sec.5.3 to read more about the two functions) are placed in the function. The *Start* waypoint is replaced with robot operating in the force mode to detect the boxes (see Sec.5.1.2 to read about Method 2). To obtain the waypoints of the straps, the UR controller or URSim invokes *getCoord_frontstrap(tcp_pose)* or *getCoord_backstrap(tcp_pose)* depending on which function is being executed in relation to the location of the strap.
3. A URScript function - *zero_ftsensor()* is used to zero the TCP force/torque measurement by subtracting the current measurement from the subsequent ones [17]. Doing so has helped in using the force sensing functionality of the robot.
4. The *Detect_Strap* (see Sec.5.6 for a more detailed explanation) function is

used to detect a strap by applying a force.

5. A check statement is used to check if a strap is detected via a boolean variable *strap_detected*, if it is **True**, the robot proceeds to verify the strap is held by its end-effector.
6. The *Verify_StrapGrab* (see Sec.5.6 for a more detailed explanation) function is used to pull the strap by applying a force.
7. Another check statement is used to check is the robot is pulling the strap via a boolean variable *strap_grabbed*, if it is **True** the UR controller or URSim proceeds to inform the operator to cut the strap.

5.5.2 Robot Program for Method 3

This section will also dive more deep into case 1 of the switch case - longer side of the pallet but w.r.t Method 3 and its flowchart of the robot program (shown in Fig. 5.8).

1. Depending on the number of straps located on the longer side of the pallet and if it is the front strap or back strap (see Fig.5.9 more intuition for side of pallet and strap position), the flow of the program splits.
2. The two functions *Longside_FrontStrap* and *Longside_BackStrap*, as the name suggests, have waypoints for robot to move towards the respective straps. But all the waypoints are now obtained from ADA via Python thus eliminating taught waypoints w.r.t the straps.
3. The functions invoked via the XML-RPC protocol, in the *Longside_FrontStrap* and/or *Longside_BackStrap* functions are:
   - *get_preStart_wp(tcp_pose, num)* to get the *preStart* waypoint (read more about the function in Sec.5.3).
   - *get_Start_wp(tcp_pose, num)* to get the *Start* waypoint (read more about the function in Sec.5.3).
   - *getCoord_frontstrap(tcp_pose)* to get the *frontstrap* waypoint (read more about the function in Sec.5.3). This function is only invoked in *Longside_FrontStrap*.
   - *getCoord_backstrap(tcp_pose)* to get the *backstrap* waypoint (read more about the function in Sec.5.3). This function is only invoked in *Longside_BackStrap*.
4. A URScript function - *zero_ftsensor()* is used to zero the TCP force/torque measurement by substracting the current measurement from the subsequent ones [17]. Doing so has helped in using the force sensing functionality of the robot.
5. In Method 3, the robot enters the force mode only during the *Verify_StrapGrab* function which is used to pull the strap by applying a force. (see Sec.5.6) The *Detect_Strap* has been eliminated in the robot program for Method 3 so as to prove the accuracy and trustworthiness of the strap waypoints obtained via ADA and the camera. (see Sec.7.2.3 for the results that validate this assumption)
6. A check statement is used to check is the robot is pulling the strap via a boolean variable *strap_grabbed*, if it is **True** the UR controller or URSim proceeds to

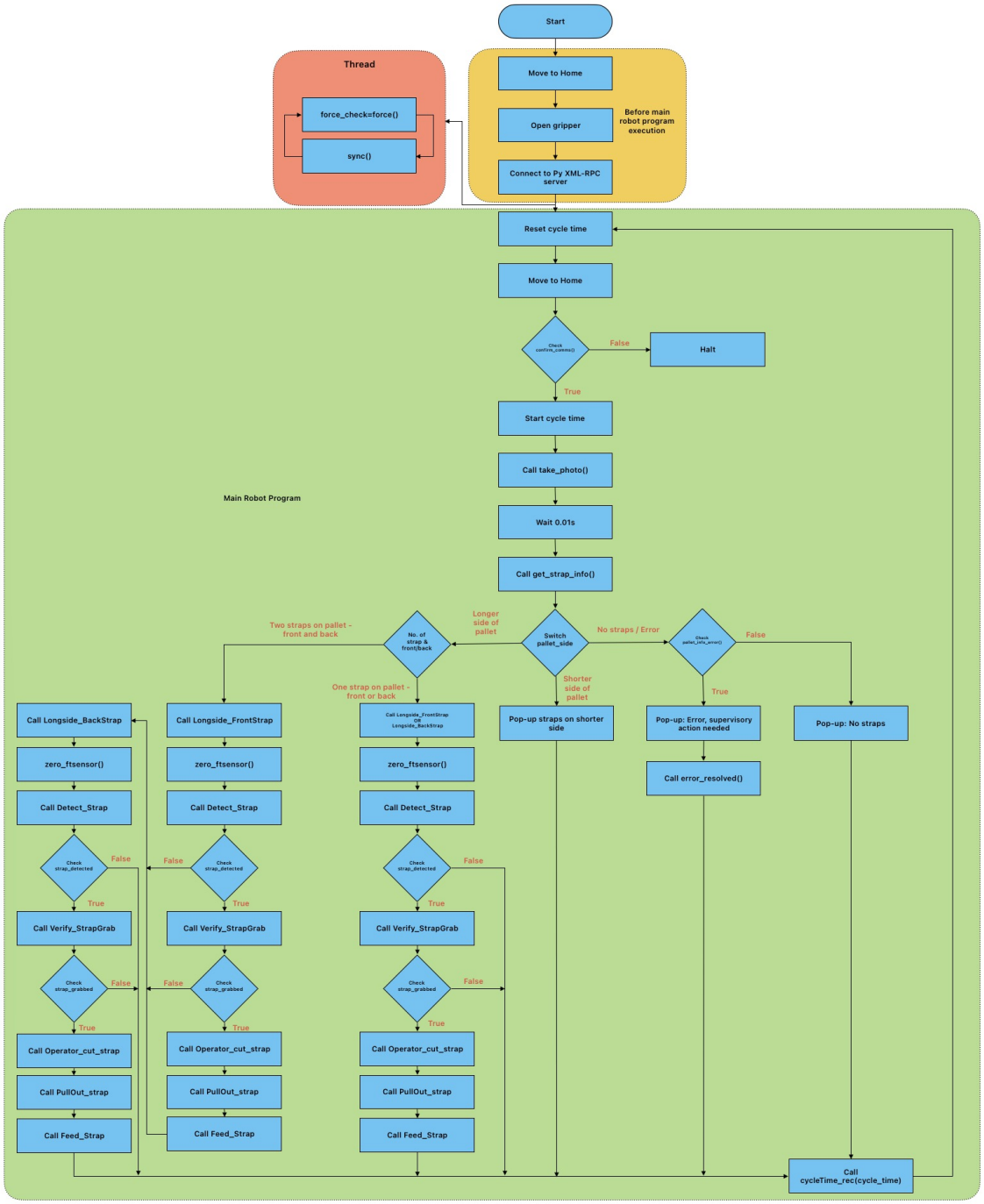

Figure 5.7: Flowchart of Robot Program for Method 1 & 2.

inform the operator to cut the strap.

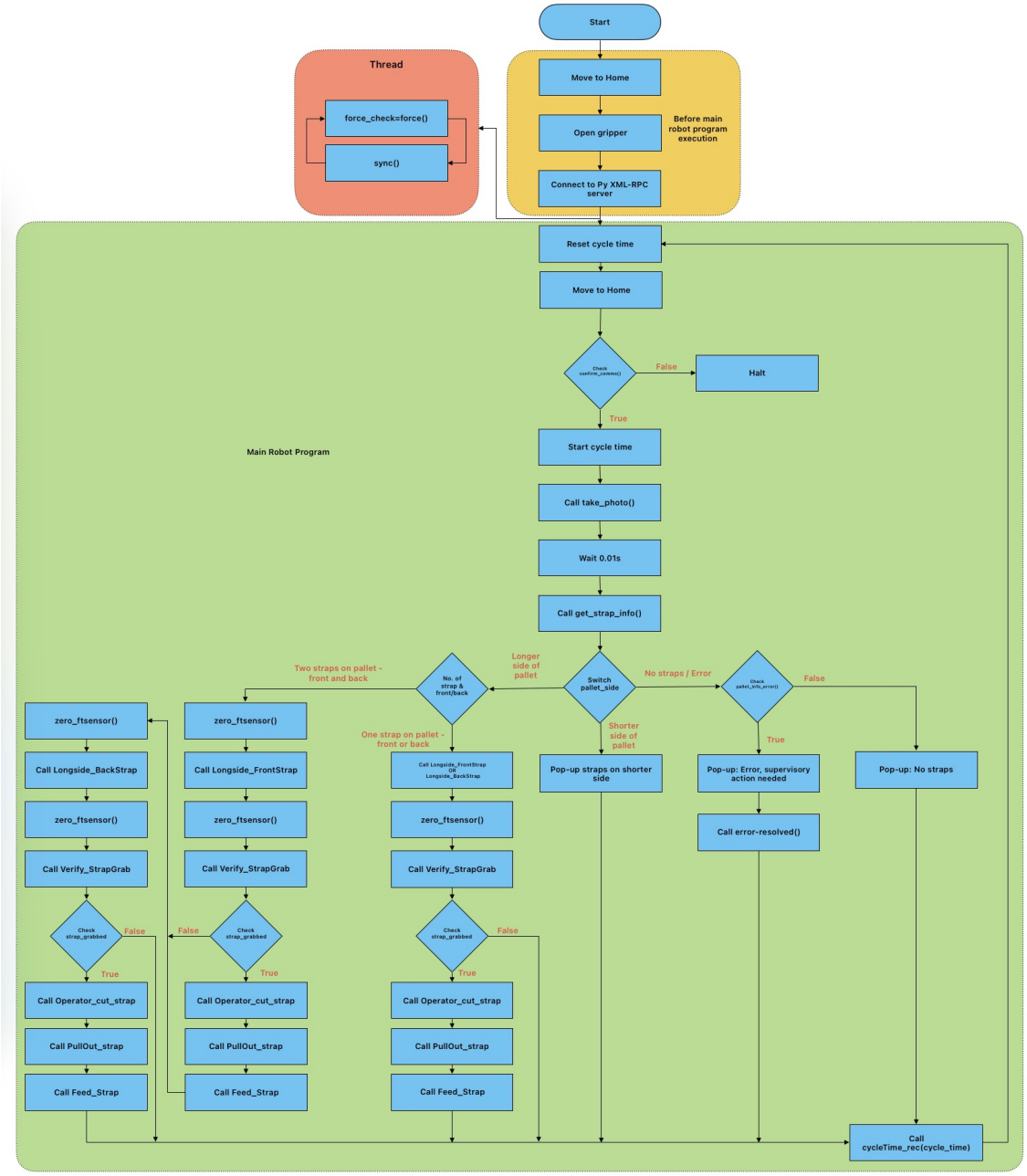

Figure 5.8: Flowchart of Robot Program for Method 3.

5.6 Strap detection and verification

The camera is one form of confirmation that there are straps on the pallet. We aim to create a redundant system. As mentioned earlier, the UR10e cobot is equipped with force sensing, tool flange/torque sensor. This can be used to detect straps and verify if the cobot has successfully grabbed the strap. The following functions are seen in the flowchart of the robot programs - Fig.5.7 and Fig.5.8:

1. Detect_Strap: In this function, using the force mode, the robot applies a force with its end-effector in the positive x direction (see Fig.5.9 for coordinate

directions) until it feels a force being exerted by the strap in the opposite direction. Once the force being exerted by the strap exceeds a certain threshold and it takes place within the timeout period, a boolean flag *strap_detected* is set to **True** and the gripper is actuated to close thus, holding the strap. In case the robot fails to detect a strap or a timeout occurs while searching for the strap, the *strap_detected* flag is set to **False**. For safety reasons, the gripper is actuated to open and the robot is moved to the **Home** position if the *strap_detected* flag is set to **False**.

2. Verify_StrapGrab: This function is another way of verifying that a strap is being held by the gripper is that the cobot moves in the negative y direction to replicate the action of pulling. The strap being tightly bound around the pallet will exert a force in the opposite direction i.e., resisting the pull. The force readings are checked, and once it crosses a threshold within a timeout period, the boolean flag *strap_grabbed* is set to **True**. If the robot fails to grab a strap, it will continue the pulling action until a timeout occurs. For safety reasons, the gripper is actuated to open and the robot is moved to the **Home** position if the *strap_grabbed* flag is set to **False**.

5.7 Cobot simulation in URSim

Once URSim is launched using Oracle VirtualBox, open PolyScope for UR10 cobot. A GUI window replicating the one on the teach pendant of UR10e cobot will appear. The robot program (.urp or .urscript files) for a specific method is then loaded into PolyScope.

Next, power on the robot in the PolyScope to be able to view the robot in the graphics tab in the program tab and use it for simulation purposes. Then, execute the loaded program and the robot is simulated according to the commands in the program.

5.8 Installation of safety features

Lastly, it is of utmost importance to create a safe environment, both for cobots and humans. UR allows us to define tool position, boundaries (planes), tool direction, and many more safety features. We have set some safety features to ensure a safe collaborative workspace between a human and a cobot. The installed safety features are as follows:

5.8.1 Tool position

There is always a risk of damaging the end-effector tool of the cobot due to collision. In order to minimize this risk, UR developed a safety feature called "Tool position". This safety feature allows one to create a sphere around the end-effector of the cobot. The sphere can have a maximum radius of 300 mm. The position of this sphere can

also be set by entering the desired offset in x, y, and z in mm. These offsets are measured with respect to the TCP frame (see Fig.3.2).

5.8.2 Safety boundaries

To restrict the cobot within a defined workspace, safety boundaries can be created. UR allows us to create these safety boundaries using two features:

- Point feature: A point feature is a reference point taught by the user to tell the robot the location of this point with respect to itself. Using this reference point, we can create a plane feature.
- Plane feature: Using the point feature, a plane is generated along the x-y plane of the point. This plane acts as a virtual boundary that can be configured to restrict the tool or the elbow from crossing over. The level of restrictions can be set as:
  - Disabled - plane is not active.
  - Normal - the robot cannot pass the plane with its TCP only in normal mode.
  - Reduced - the robot cannot pass the plane with its TCP only in reduced mode.
  - Both - the cobot cannot pass the plane with its TCP.
  - Trigger Reduced Mode - the cobot can pass through the plane switching from one mode to another.

  The following planes were created and configured as normal i.e., the robot cannot cross these planes (see Fig.5.9):
  1. Left Safety Plane.
  2. Plane behind the Robot i.e., Back Safety Plane.
  3. Safety Plane between pallet and feed-through machine.
  4. Safety Plane to protect the camera.

5.9 Practical Setup

Fig. 5.9 shows the top-view of the practical setup of the thesis. The robot base frame is at the center of the robot's base with the x axis pointing in the left direction, the y axis in the downward direction and the z axis pointing out of the plane of the page. Position of the pallet, camera, feeder (imaginary) and the safety planes are set with respect to the robot's base frame. The positions of the pallet and camera were recorded using the TCP pose w.r.t robot's base frame. This enables to accurately re-setup the scenario in another location.

5.9.1 Orientation of camera

The orientation of the camera is decided by placing it in a manner such that the camera captures the pallet and the straps clearly. Once a suitable angle was found, using the TCP pose readings of the robot, the orientation of the camera w.r.t the robot was calculated. The following coordinates were used to calculate the angle:

- Point 1 $(x_1, y_1) : (-788.58, -249.13)$.

- Point 2 (x_2, y_2) : $(-1360.09, -42.05)$.

To calculate the angle of camera about the z axis i.e., x-y plane (α_{cam}):

$$\alpha_{cam} = \arctan \frac{y_2 - y_1}{x_2 - x_1} \tag{5.4}$$

$$\alpha_{cam} \approx 19.92^{\circ} \tag{5.5}$$

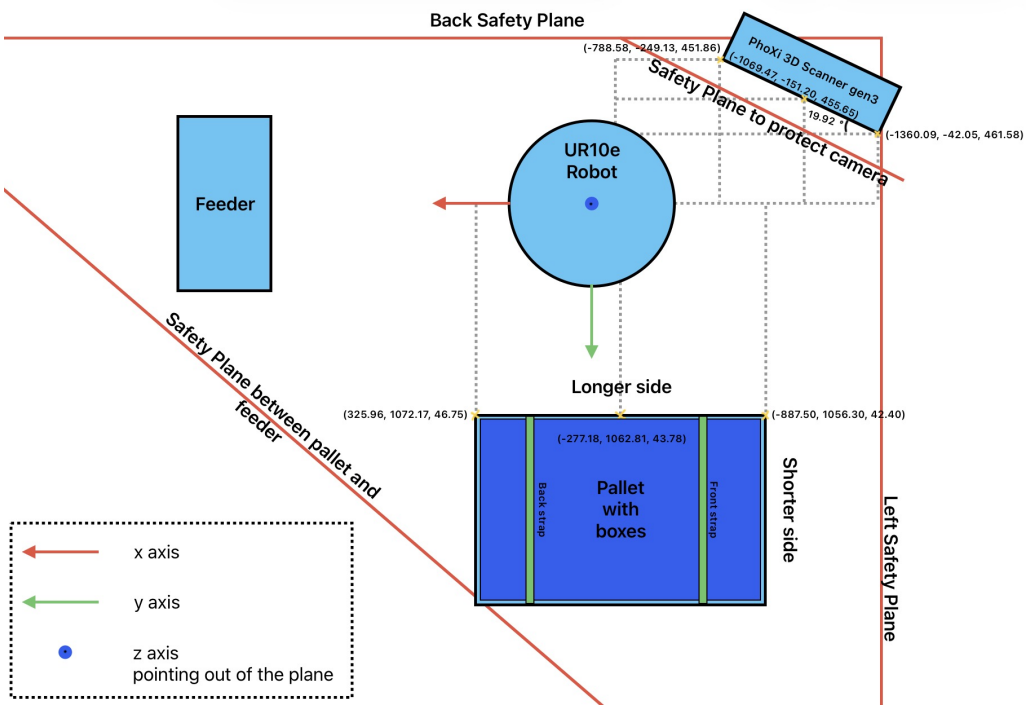

Figure 5.9: Top-view layout of the practical setup, showing the robot and placement of camera, pallet, feeder area and safety planes w.r.t robot's base frame. 3D coordinates of the camera and pallet are shown. The orientation of the camera has also been highlighted.

6

Experimental design

Here, the test scenarios to obtain quantitative results are described.

6.1 Testing the accuracy of the estimated homogeneous transformation matrix

To estimate the homogeneous transformation matrix, the camera was used to detect the robot's gripper. Since the camera does not detect objects in black well enough due to poor reflection of the emitted laser, a band-aid was applied to one of the jaws of the gripper. The coordinates of the gripper in the camera frame are then recorded and using the teach pendant of the robot, the respective TCP coordinates were recorded. Nine points were used to estimate the homogeneous transformation. Thus, after the homogeneous transformation matrix was estimated using the least squares approach (explained in Sec.5.4), six new points were recorded in the camera frame and the robot frame (respective TCP pose). The points in the camera frame were multiplied with the estimated transformation matrix to obtain the estimated TCP coordinates. Refer to the Sec.7.1 for results.

6.2 Testing the methods developed for de-strapping

To test the practicality of the methods proposed in this thesis and obtain quantitative results, the following test case scenarios were created. Note that, 30 cycles were tested for each pallet placement mentioned in the test scenarios.

6.2.1 Method 1

As described in Sec.5.1.1, method 1 can be assumed to work well only when the pallet is at the origin or is offset in a translational manner in the x direction. However, due to the manual teaching of the waypoints, the pallet can neither have any translational offset in the y direction nor can it have any yaw offset. Fig.6.1 shows the testing scenario for Method 1 where, corner 1 (C1) of the pallet is marked as the origin and markers are placed at every 50 mm from corner 2 (C2) of the pallet, indicating the pallet will be offset by 50 mm for each test case. The assumptions were proven after all the test cases were carried out and the results pertaining to the success of the robot trajectory (Sec.7.2.1), the strap identification (Sec.7.3.1), strap colour

identification (Sec.7.4.1) and cycle time (Sec.7.5.1) are presented in their respective sections.

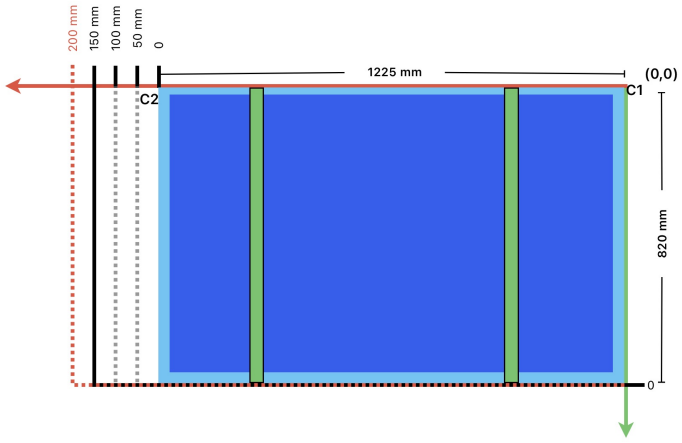

Figure 6.1: Translational offset possibilities in x (red solid line).

6.2.2 Method 2

Method 2 is described in Sec.5.1.2. Method 2 was designed to account for translational offset not only in the x direction but also in the y direction. However, this assumption was only possible after the test cases were modified. Originally, when the corner 1 (C1) of the pallet is at the origin, it means that the pallet is placed exactly as that shown in the practical setup Fig.5.9. But, the pallet could not be offset in the y direction since the robot arm is completely extended if an offset in y exists. Only an offset of 50 mm in the x direction could be achieved as shown in Fig.6.2.

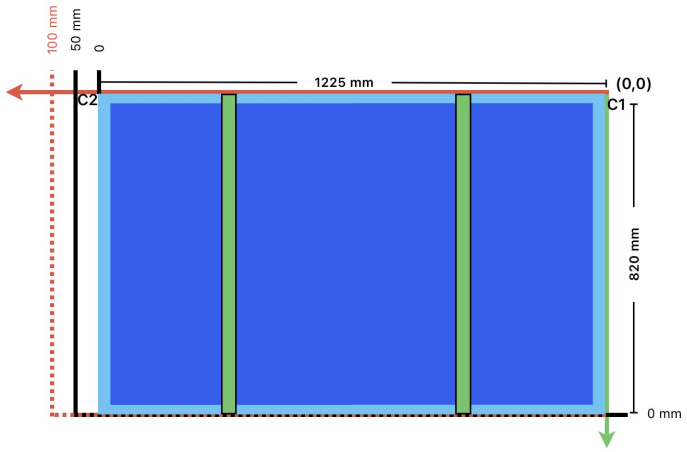

Figure 6.2: Translational offset possibilities in x (red solid line).

To prove the earlier assumption, the pallet was offset by 100 mm in the x direction and thus the origin is shifted by 100 mm and referred to as the new origin (shown in Fig 6.3). The *preStart* waypoints were re-taught as per the new origin. On doing

so, the assumption of being able to offset the pallet in the x and y direction was validated and the results pertaining to the success of the robot trajectory (Sec.7.2.2), the strap identification (Sec.7.3.2), strap colour identification (Sec.7.4.2) and cycle time (Sec.7.5.2) are presented in their respective sections. However, the offset as a result of a yaw rotation was not favorable.

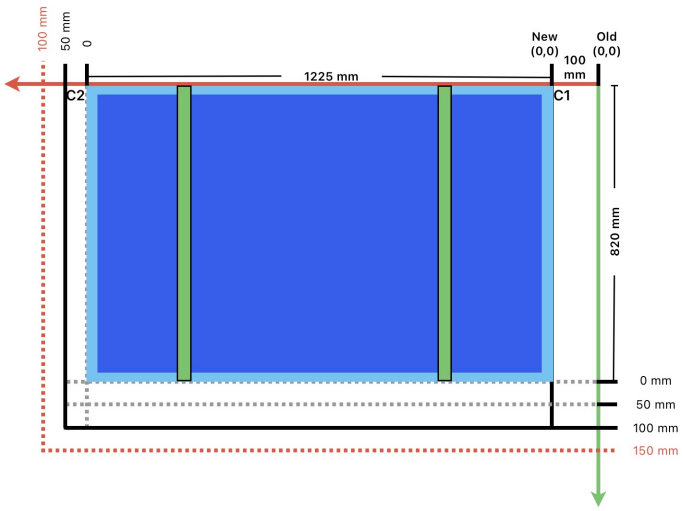

Figure 6.3: Translational offset possibilities in x (red solid line) and y (green solid line) after pallet is moved to a new origin.

6.2.3 Method 3

Method 3 (presented in depth in Sec.5.1.3) was developed to be the most robust solution to handle de-strapping, allowing for translational (in x and y) and rotational (in yaw) offsets. Five test scenarios were considered to prove the robustness of Method 3.

- The first scenario shows (Fig.6.4) the possibilities of translational offset in x and y direction. In this scenario, the pallet was offset by 50 mm, first only in the x direction, then in the y direction and lastly diagonally. Thus leading to a rectangular space enclosed by solid black lines as shown in the Fig.6.4.
- The second test scenario was to rotate the pallet around corner 1 (C1) of the pallet but no translational offset i.e., the pallet is at the origin (as shown in Fig.6.5). The pallet was rotated by about 7.03°.
- The third test scenario was to rotate the pallet around corner 2 (C2) of the pallet but no translational offset i.e., the pallet is at the origin (as shown in Fig.6.6). The pallet was rotated by about 7.03°.
- The fourth test scenario was to rotate the pallet around corner 1 (C1) of the pallet but after a translational offset of 200 mm in the x direction (as shown in Fig.6.7). The pallet was rotated by about 7.03°.
- The last test scenario was to rotate the pallet around corner 2 (C2) of the pallet but after a translational offset of 200 mm in the x direction (as shown in Fig.6.8). The pallet was rotated by about 7.03°.

Thus, after conducting these tests, the assumptions were validated and are presented in the robot trajectory (Sec.7.2.3), the strap identification (Sec.7.3.2), strap colour identification (Sec.7.4.2) and cycle time (Sec.7.5.3) respectively.

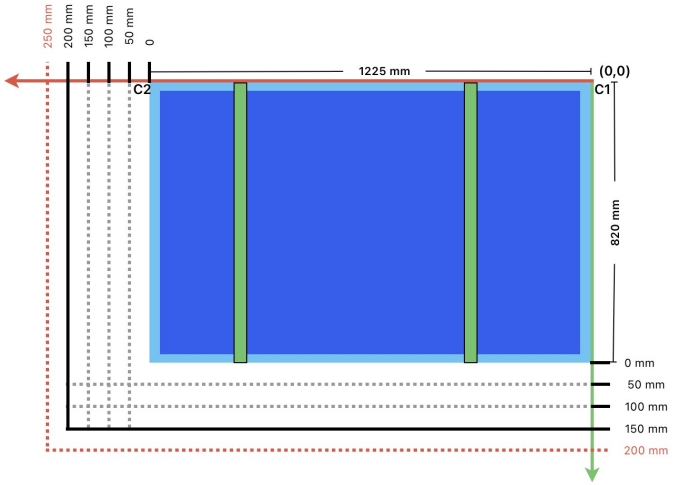

Figure 6.4: Test grid for Method 3 with translational offset in x (red solid line) and y (green solid line).

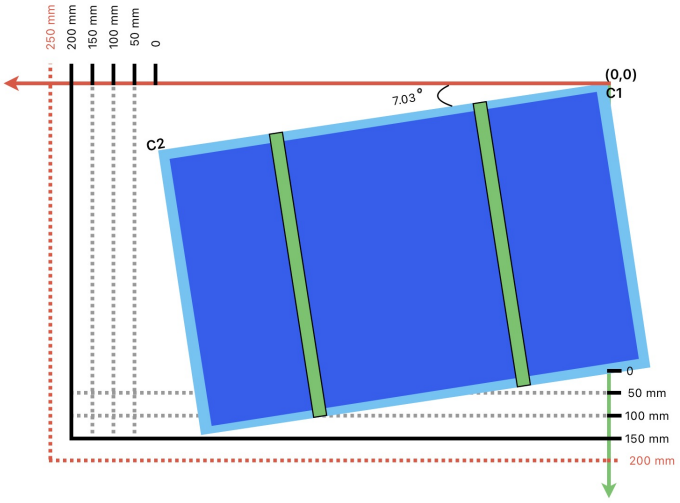

Figure 6.5: Pallet at origin and rotated around C1.

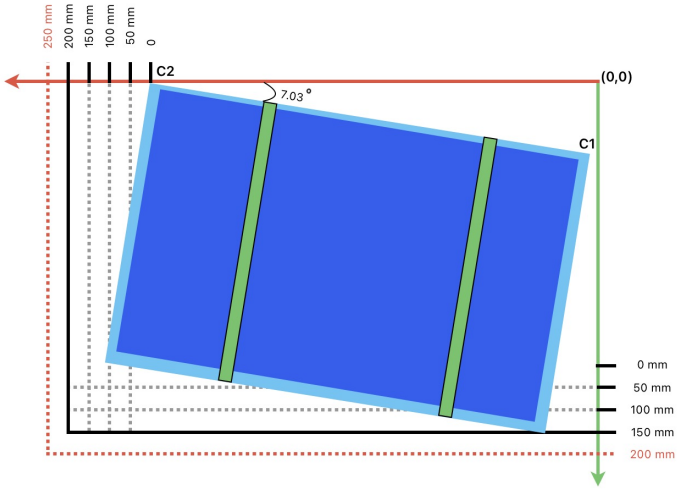

Figure 6.6: Pallet at origin and rotated around C2.

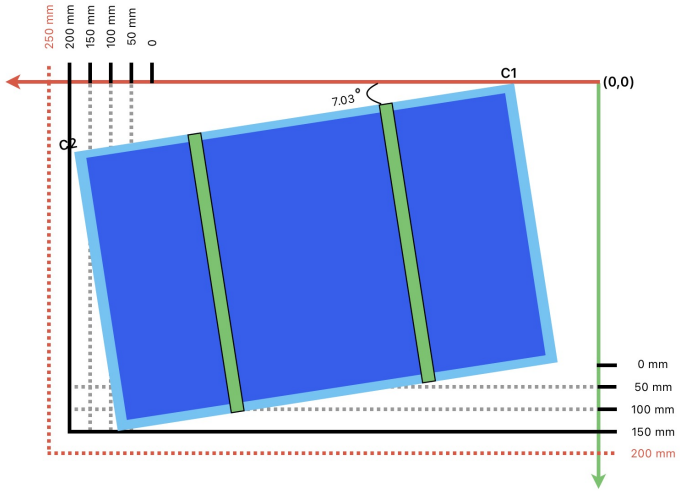

Figure 6.7: Pallet offset by 200 mm in x (red solid line) and rotated around C1.

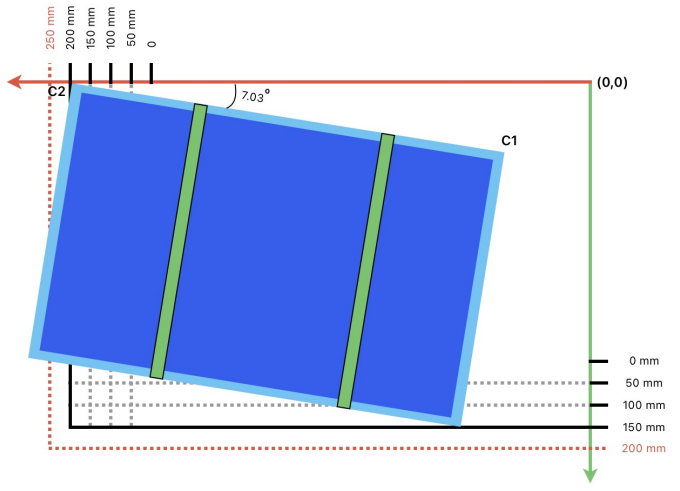

Figure 6.8: Pallet offset by 200 mm in x (red solid line) and rotated around C2.

7

Results

In this chapter, the results obtained during the various trials have been presented.

7.1 Accuracy of the estimated homogeneous transformation matrix

The test scenario to obtain the result is described in Sec.6.1. The nine reference points that were taken for the estimation of the homogeneous transformation matrix are shown in Table 7.1. The six test points along with the estimated TCP points obtained after transformation are shown in Table 7.2.

Reference points						
Coordinates → Reference point ↓	Camera x [mm]	Camera y [mm]	Camera z [mm]	Robot x [mm]	Robot y [mm]	Robot z [mm]
1.	-317.05	-0.55	1107.01	-744.95	912.29	363.90
2.	-370.75	5.87	910.02	-860.35	745.05	372.39
3.	201.01	-26.52	1386.89	-161.20	1000.40	354.95
4.	-234.18	-79.06	1339.07	-583.51	1105.63	424.43
5.	153.66	-44.52	1329.15	-225.48	963.07	378.92
6.	-33.65	-181.01	1377.91	-379.69	1079.85	518.45
7.	-353.88	-41.50	1275.30	-719.19	1084.11	396.00
8.	204.86	-32.29	1393.09	-156.2	1005.40	359.95
9.	-373.57	18.28	914.51	-865.35	750.05	359.80

Table 7.1: Reference points taken for the estimation of the homogeneous transformation matrix calculation.

Test points								
Camera			Measured tcp			Estimated tcp		
x	y	z	x	y	z	x	y	z
-441.71	-34.64	1134.98	-853.41	982.19	396.25	-849.94	982.39	399.96
-110.12	-40.43	1317.82	-478.94	1042.49	384.91	-476.75	1041.83	383.25
309	-107.87	1440	-40.89	1018.85	430.58	-37.29	1017.20	430.37
-87.49	-209.91	1318.60	-447.43	1043.39	549.68	-443.82	1042.24	553.64
-120.65	-67.16	1529.70	-415.05	1246.24	395.56	-414.31	1245.73	395.90
-166.85	-226.05	1284.64	-533.65	1039.65	569.94	-528.69	1037.99	574.65

Table 7.2: Test points taken for the verification of the estimated homogeneous transformation matrix.

Fig.7.1 shows the trend of errors in the x, y and z axis along with the absolute error. It is observed that after using six out of nine reference points for the estimation of the homogeneous transformation matrix, and testing the six test points, the absolute error does not deviate much. Thus, the error settles around 3.5 mm if more than six reference points are used to estimate the transformation matrix and tested for the same six test points.

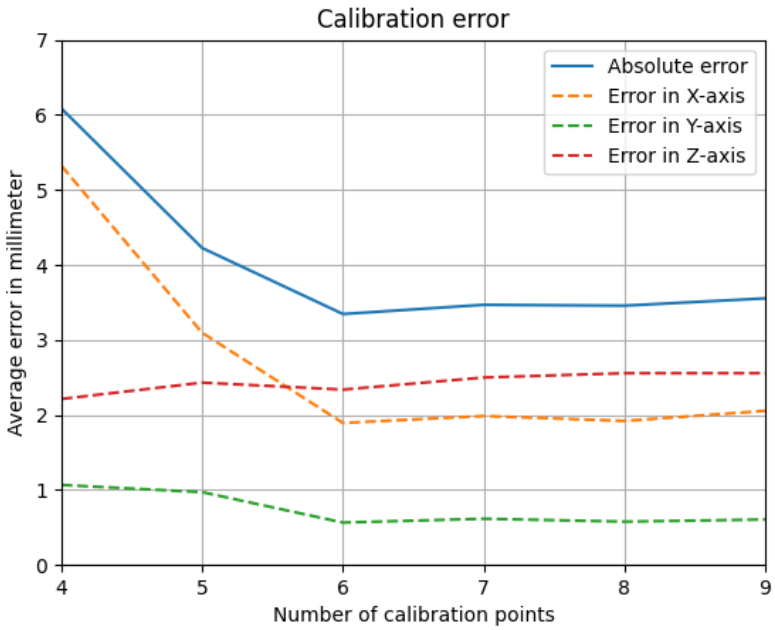

Figure 7.1: Calibration error as a function of the number of reference points used to estimate the homogeneous transformation matrix.

7.2 Robot path to grab straps

The Methods developed (see Sec.5.1) play an important role in modifying the path for the robot to traverse. In this section the success rate is computed based on the

capability of the UR controller to find a valid path for the robot, to move the robot along it and to grab the straps. A successful robot path is one where it can grab an identified strap.

7.2.1 Method 1

As Method 1 relies on manually taught waypoints, the robot is able to handle offsets only in the x direction (see the test scenario described in Sec.6.2.1). However, any changes in the y direction larger than half of the grippers opening will cause it to miss the straps.

Due to the predefined waypoints the Method is not robust enough to handle a lot of variation in angle. It can handle a maximum angle of

$$\alpha =^{+}_{-} sin^{-1}\left(\frac{\text{Gripper's opening in mm}}{910}\right) \tag{7.1}$$

, assuming that the pallet is rotated around its center. If the rotation is around one corner the maximum angle it can handle is

$$\alpha =^{+}_{-} sin^{-1}\left(\frac{\text{Gripper's opening in mm}}{2 \times 1055}\right) \tag{7.2}$$

For the gripper used in these trials the maximum angle for the pallet if it is being rotated around its center would be $\alpha =^{+}_{-} 0.944°$ and for rotated around one corner $\alpha =^{+}_{-} 0.407°$.
Table 7.3 shows the success rate of the robot when the pallet is at the origin and when the pallet has been offset in the x and y directions.

Translational offset						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	100.0%	100.0%		
[0.05m]	0.00%	0.00%				

Table 7.3: Success rate of robot path using Method 1 after introducing translation offsets.

7.2.2 Method 2

Method 2 had two test scenarios, see Sec.6.2.2. Table 7.4 shows the success rate of the robot in both the scenarios for translational offsets. Since Method 2 relies on detecting the box (see Sec.5.1.2), it allows more variation in the y direction given that the robot can reach the box.
Method 2 allows for some variation in rotation of the pallet. It can handle a maximum angle of

$$\alpha =^{+}_{-} sin^{-1}\left(\frac{\text{Grippers opening in mm}}{382.5}\right) \tag{7.3}$$

irrespective of the pallet being rotated around its center or a corner. For the gripper used in these trials the maximum angle for the pallet is $\alpha = \pm 2.247°$. However, it was able to handle the angle of 3.52° (as shown in Table 7.5) because the straps were located in the middle of their respective possible placement while the calculations represent the strap at any possible placement.
NOTE: * in the table indicate tests done with a new origin set at (0.1,0) to test the robustness in the y direction and not being limited by the robots reach.

Translational offsets						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	100.0%*			
[0.05m]			100.0%*	100.0%*		
[0.10m]			100.0%*			

Table 7.4: Success rate of robot path using Method 2 after introducing translation offsets.

Rotational offsets		
Position \Rotation	[+3.52°]	[+7.03°]
[C1(5,0)]	100.0%	

Table 7.5: Success rate of robot path using Method 2 after introducing rotational offsets.

7.2.3 Method 3

Regardless of the position or rotation of the pallet, Method 3 (see Sec.5.1.3) was the most flexible Method where the waypoints were obtained using the image processed in ADA. The UR controller was able to find a valid path for the robot given that the robot could reach the point and the image processing was able to correctly identify the strap and strap position. The test scenario for Method 3 is described in Sec.6.2.3. Table 7.6 and 7.7 show the success rate when the pallet is offset in a translational manner and a rotational manner respectively.

Translational offsets						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
[0.05m]	100.0%	100.0%				
[0.10m]	100.0%		100.0%			
[0.15m]	90.00%			100.0%	100.0%	
[0.20m]	0.00%				100.0%	
[0.25m]						100.0%

Table 7.6: Success rate of robot path using Method 3 after introducing translational offsets.

Where it fails is when the accuracy of the points are worse due to being to far away from the camera.

Rotational offset		
Position \Rotation	[7.03°]	[3.52°]
[C1(0,0)]		100.0%
[C1(20,0)]	0.00%	0.00%
[C2(0,0)]	100%	
[C2(20,0)]	100%	

Table 7.7: Success rate of robot path using Method 3 after introducing rotational offsets.

7.3 Strap identification

The success rate of strap identification is determined by testing for the ability of the camera and image-processing in ADA, to correctly find a strap and obtain the coordinates (only for Method 2 and 3).

7.3.1 Method 1

The strap identification fails quickly as the pallet is moved from its original placement as seen in Table 7.9 but it was overall accurate as can be seen in Table 7.8.

Camera strap → Strap present ↓	T	F
P	230/240	10

Table 7.8: Confusion matrix for strap identification on all Method 1 tests

Strap identification using Method 1 - translational offsets						
y \x	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	100.0%	80.33%		

Table 7.9: Strap identification using Method 1 - translational offsets.

7.3.2 Method 2

The strap identification of Method 2 got worse as the pallet was moved in the x direction as the predefined search boxes defined in the image processing flowchart in ADA does not move with the box. Table 7.11 and 7.12 highlight the success rate. **NOTE:** * in the table indicate tests done with a new origin set at (0.1,0) to test the robustness in the y direction and not being limited by the robots reach. Overall it has a good ability to correctly identify the straps as seen in Table 7.10.

Camera strap → Strap present ↓	T	F
P	374/420	46

Table 7.10: Confusion matrix for strap identification on all Method 2 tests

Strap identification using Method 2 - translation offsets						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	93.33%	100.0%	76.66%*			
[0.05m]			100.0%*	83.33%*		
[0.10m]			100.0%*			

Table 7.11: Strap identification using Method 2 - translation offsets.

Strap identification using Method 2 - rotational offsets		
Position \Rotation	[3.52°]	[7.03°]
[C1(5,0)]	70.00%	

Table 7.12: Strap identification using Method 2 - rotational offsets.

7.3.3 Method 3

The strap identification using Method 3 got worse as the pallet was moved in the x direction as it uses the same strap identification process as Method 2. The success rate is shown in Table 7.14 and 7.15. Method 3 has a very good strap identification on the test overall that can be seen in Table 7.13.

Camera strap → Strap present ↓	T	F
P	1012/1080	58

Table 7.13: Confusion matrix for strap identification on all Method 3 tests

Strap identification of Method 3 - translational offsets						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	80.00%	96.67%	93.33%	60.00%
[0.05m]	100.0%	100.0%				
[0.10m]	100.0%		100.0%			
[0.15m]	93.33%			100.0%	100.0%	
[0.20m]					100.0%	
[0.25m]						100.0%

Table 7.14: Strap identification of Method 3 - translational offsets.

Strap identification using Method 3 - rotational offsets		
Position \Rotation	[3.52°]	[7.03°]
[C1(0,0)]		73.33%
[C2(0,0)]	100.0%	
[C2(20,0)]	90.00%	

Table 7.15: Strap identification using Method 3 - rotational offsets.

7.4 Strap colour identification

In this section the success of the strap colour identification is shown.

7.4.1 Method 1

The colour identification fails quickly as the pallet is moved away from the original position as observed in Table 7.17 and the overall success can be seen in Table 7.16.

Camera strap colour → Strap colour ↓	Blue	Green	White	Black
Green	2	188/234	42	2

Table 7.16: Confusion matrix for strap colour identification on all Method 1 tests

Strap colour identification using Method 1 - translational offset						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	88.33%	48.33%	80.36%		

Table 7.17: Strap colour identification using Method 1 - translational offset.

7.4.2 Method 2

Method 2 can do the colour identification robustly but as the pallet is moved far enough the accuracy declines as shown in Table 7.19 and 7.20. **NOTE:** * in the table indicate tests done with a new origin set at (0.1,0) to test the robustness in the y direction and not being limited by the robots reach. Overall it had a very good colour identification as seen in Table 7.18.

Camera strap colour → Strap colour ↓	Blue	Green	White	Black
Green	4	303/312	5	0

Table 7.18: Confusion matrix for strap colour identification on all Method 2 tests

Strap colour identification using Method 2 - translational offset						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	89.13%*			
[0.05m]			100.0%*	100.0%*		
[0.10m]			96.66%*			

Table 7.19: Strap colour identification using Method 2 - translational offset.

Strap colour identification using Method 2 - rotational offset		
Position \Rotation	[7.03°]	[3.52°]
[C1(5,0)]		95.24%

Table 7.20: Strap colour identification using Method 2 - rotational offset.

7.4.3 Method 3

Method 3 can perform the colour identification robustly but as the pallet is moved far enough the accuracy is declining. Table 7.22 and 7.23 shows the results. In the Table 7.21 Method 3 has a success rate of 91.08%.

Camera strap colour → Strap colour ↓	Blue	Green	White	Black
Green	50	929/1020	0	41

Table 7.21: Confusion matrix for strap colour identification on all Method 3 tests

Strap colour of Method 3 results						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	100.0%	100.0%	100.0%	100.0%	58.33%	95.00%
[0.05m]	50.0%	100.0%				
[0.10m]	100.0%		100.0%			
[0.15m]	100.0%			93.33%	93.33%	
[0.20m]					100.0%	
[0.25m]						100.0%

Table 7.22: Strap colour identification of Method 3.

Angled results of Method 3		
Position \Rotation	[7.03°]	[3.52°]
[C1(0,0)]	54.17%	
[C2(0,0)]	100%	
[C2(20,0)]	88.33%	

Table 7.23: Strap colour identification of Method 3.

7.5 Average cycle time

Since almost 30 cycles per possible combination for each Method was carried out, the average cycle time for each combination is computed.

7.5.1 Method 1

The average cycle time for the tested pallet placements is shown in table 7.24. Thus, the mean cycle time considering all test scenarios for Method 1 is 47.34 s.

Avg. cycle time of Method 1 [s]						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	46.25	47.02	47.64	48.45		

Table 7.24: Average cycle time of Method 1 - translational offset.

7.5.2 Method 2

The mean cycle time for all possible pallet placements before the recalibration for Method 2 is 50.45 s whereas, after recalibration, it is 51.41 s. Table 7.25 and 7.26 show the average cycle times for the tested pallet positions.
NOTE: * in the table indicate tests done with a new origin set at (0.1,0) to test the robustness in the y direction and not being limited by the robots reach

Avg. Cycle time of Method 2 [s]						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	50.36	50.76	49.43*			
[0.05m]			51.39*	51.54*		
[0.10m]			53.28*			

Table 7.25: Average cycle time of Method 2 - translational offset.

Avg. Cycle Time of Method 2		
Position \Rotation	[7.03°]	[3.52°]
[C1(5,0)]		50.24

Table 7.26: Average cycle time of Method 2 - rotational offset.

7.5.3 Method 3

The mean cycle time for all possible pallet placements for Method 3 is 39.24 s. Table 7.27 and 7.28 show the average cycle time for the tested pallet placement.

Avg. cycle time of Method 3 [s]						
$y \backslash x$	[0.00m]	[0.05m]	[0.10m]	[0.15m]	[0.20m]	[0.25m]
[0.00m]	39.03	38.59	37.96	37.87	37.65	37.42
[0.05m]	40.64	39.11				
[0.10m]	41.16		39.32			
[0.15m]	41.17			39.57	39.16	
[0.20m]					39.94	
[0.25m]						40.33

Table 7.27: Average cycle time of Method 3 - translational offset.

Avg. cycle time of Method 3 [s]		
Position \Rotation	[7.03°]	[3.52°]
[C1(0,0)]	39.45	
[C2(0,0)]	39.88	
[C2(20,0)]	38.13	

Table 7.28: Average cycle time of Method 3 - rotational offset.

7.6 Strap identification and strap colour testing on incoming pallet line

Using the strap identification and strap colour identification of Method 2 and 3 on the incoming pallet line, images of 26 pallets were taken using the MotionCam. There was nothing done to the improve the lighting conditions. The results obtained are:

Camera strap colour → Strap colour ↓	Blue	Green	White	Black
Blue	0/0	0	0	0
Green	0	16/18	0	2
White	0	0	2/2	0
Black	0	4	0	0/4

Table 7.29: Confusion matrix for strap colour identification on incoming pallet line test

$$\text{Strap colour identification success} = \\ = \frac{\text{Correct colour}}{\text{Total Number of straps}} = \frac{18}{24} = 75.00\% \tag{7.4}$$

The image processing was not able to sort out black straps. Also, two pallets were wrapped in plastic and hence the colour identification failed. Only considering

pallets with white and green straps, the success rate of colour identification was computed to be:

$$\frac{18}{20} = 90\% \tag{7.5}$$

Camera strap → Strap present ↓	T	F
P	28/32	4
N	15/20	5

Table 7.30: Confusion matrix for strap identification on incoming pallet line test

$$\begin{aligned} &\text{Strap identification success} = \\ &= \frac{\text{True Negative} + \text{True Positive}}{\text{Total Number of Negatives} + \text{Total Number of Positives}} = \\ &= \frac{15 + 28}{20 + 32} = 76.79\% \end{aligned} \tag{7.6}$$

The success rate for strap identification for true negative case was 75.00% and for the true positive case was 87.50%.

8

Discussion and future work

8.1 Accuracy of the estimated homogeneous transformation matrix

The waypoints obtained via the camera, when multiplied by the estimated homogeneous transformation matrix, had an error of 3.5 mm. This level of accuracy was obtained after 6 points of calibration were taken (refer Fig.7.1). This calibration and accuracy of the camera and robot can be used to de-strap pallets.
During the quantitative test for Method 3 (see Sec.7) it never failed because of error in the communicated points from the camera to the robot. The end-effector tools 15 mm opening (see Sec.4.1.4) can handle an accuracy of 7.5 mm and it can therefore be concluded that the deviation during the 450 cycles that tested the Method 3 never got above 7.5 mm.
The accuracy may further be improved by having the end-effector fit with a smaller and highly reflective dot that can easily be identified in the pictures and knowing the exact position of said dot.
The camera has an accuracy of 1.25 mm [10] and the UR10e robot has an accuracy of 0.05 mm [18] giving this setup a theoretical maximum accuracy of 1.3 mm if the homogeneous transformation is perfect. But that could only be done by precisely measuring the coordinates manually for the camera and robot thus not relying on the accuracy of the camera and robot for calibration points. This would make the calibration more tedious but give an theoretical improvement of 65.86% from the calibration done in this report. Knowing this leads to the conclusion that improvements can be done.

8.2 Method comparison

Comparing how the different methods preformed on different tasks.

8.2.1 Robot path

Method 1 cannot handle more variation in the y direction than half of the grippers opening while its tolerance in the x direction is dependent on the timeout period to feel for a strap. In the robot path tests for Method 1 (see Sec.7.2.1) it could handle 0.15 m of offset in the x direction, 0.05 m of offset in the y direction and 0.407 degrees of pallet rotation.

Method 2 can handle variation in the y direction as long as the robot can reach it but cannot handle more than 0.0725 m of variation in the x direction since it needs to touch the box to set the y coordinate (see Sec.7.2.2). It can handle about 5.52 times more rotation than Method 1 being able to handle 2.247 degrees of rotation (see Sec.7.2.2).
Method 3 can handle the variations tested assuming the robot can reach and the image processing is able to find the straps (see Sec.7.2.3).

8.2.2 Strap identification

Method 1 does not have the same accuracy (see Sec.7.3.1) as Method 2 (see Sec.7.3.2) and 3 (see Sec.7.3.3) due to Method 2 and 3 filters on depth after the initial strap identification (see Sec.5.1.2 & Sec.5.1.3) allowing a more precise position of the strap to be obtained.
The test on strap identification on the incoming pallet line where multiple strap positions and colours were tested, the accuracy dropped significantly (see Sec.7.6). The success rate of identifying the straps was 87.50% and only 75.00% when there was no strap present. The main issues was some large letters on the boxes sides interfered with the strap detection - they had an edge threshold higher than the lowest edge strength of the straps.

8.2.3 Strap colour identification

As concluded in section 8.2.2 Method 1 does not obtain a precise strap position and therefore cannot obtain exact coordinates of where to analyze the colour in the 2D image. This results in Method 1 having a worse colour identification (see Sec.7.4.1) than Method 2 (see Sec.7.4.2) and 3 (see Sec.7.4.3). Method 3 had worse overall success rate than Method 2 but only looking at the shared pallet orientations Method 3 had a higher success rate.
The test on strap colour identification on the incoming pallet line where multiple strap positions and colours were tested, the image processing had an issue with identifying black coloured straps. Removing the eight black straps the accuracy is improved up to 90.00% (see Eq.7.5). The colours are sorted by RGB and luminance values and have an issue of the darkest blue and green straps are darker than the most reflective black straps.

8.2.4 Cycle time

Comparing the cycle times at the positions (0, 0) and (0.05, 0), that all the methods completed from Table 7.24, 7.25 and 7.27. It shows that Method 1 takes 120% of the time it takes Method 3 and Method 2 takes 130% of the time it takes Method 3.
The fastest method is Method 3 because it doesn't feel for the strap. It only verifies that it has grabbed it making it possible for the robot to move faster and shorter distances. Method 2 takes 8.33% more time than Method 1. This is because Method

2 needs to feel for the box to set a y coordinate and needs to feel for the strap in the x direction while Method 1 only feels for the strap in the x direction making it faster than Method 2.

8.3 Industrial feasibility

The combination of Method 1 and 2 is the most feasible solution for Volvo at the moment. But for implementation, the system would be need to be more robust.

8.4 Safety

The UR10e is a collaborative robot but the safety of the entire system would depend on how much Volvo would integrate the system with people.

8.5 Cost and benefit

The fastest method is Method 3 however regarding the price difference between 3D-scanners and regular 2D cameras the argument can be made that combining Method 1 and 2 by using the robot program of Method 2 with the 2D-image processing of Method 1 can give a robust enough system for a possible cheaper price at the cost of 30% more cycle time (see Sec.8.2.4).

8.6 Camera suitability

The effect that various camera qualities can have on strap identification has not been tested. But it can be argued that a camera with more than 2 MP would be able to give clearer images for ADA to process and therefore make it more accurate to not mistake large letters for straps. A higher quality camera could also improve the accuracy of colour identification since the distinction between red, green, blue and black straps would be more clear. Though with more time and better lit workstation the image processing can be improved but studying the images it is doubtful that it would be enough to get reliable results

8.7 Limitations

Initial development was carried out using the Photoneo 3D scanner gen3 but due to an unforeseeable circumstance, it was replaced with the Photoneo MotionCam at the end when the test where about to be carried out. This cost two weeks of testing and optimization taking it down from three and a half weeks to only one and a half week of testing and optimization.

8.8 Future work

More research needs to be done in order to implement it on the incoming pallet line and find the most optimal solution for Volvo.

8.8.1 Methods

Test on a combination of Method 1 and 2 with Method 1's 2D image and Method 2's robot program to find the strap would need to be tested. All the methods would need to be tested on more cycles and pallet variation to get more data on their respective robustness.

8.8.2 UR10e Robot and software

Tests using a larger robot would be needed to handle all of the strap orientations and be able to remove the top lids.

8.8.3 ADA

More calibration of the strap colour identification. Currently the colour sorting has only been tuned on 20 pallets and this is not enough to have a robust colour sorting. Even if ADA has been found to be satisfactory testing open-source option like OpenCV could save on cost if that is also found to be satisfactory and is therefore recommended as further work.

8.8.4 Camera

The expectation on future work on image quality is that the 2 MP 3D MotionCam does not have enough image quality to reliably identify the colour of the present straps.
Evaluation of using a purely 2D camera would also need testing to verify if the 3D images are unnecessary.

9
Conclusion

A UR and Photoneo 3D MotionCam with ADA can be used to de-strap pallets.

9.1 Methods

All the methods can solve robot paths to de-strap pallets but they can handle different amounts of variations in pallet positions. All the methods strap identification can be used to identify the straps and their colour. All the methods had cycle times below one minute allowing them to de-strap at least 480 pallets during an eight hour time frame which is more than the 300-400 pallets Volvo receive in a day.

9.2 UR10e Robot and software

The UR10e doesn't have enough reach to be able to reach both the short and long side of the standard pallets. However the UR software is enough and the accuracy of the UR10e is good enough, so a larger UR robot with longer reach should work but that needs to be verified. The force censoring capabilities of the UR10e robot would still be needed on the larger robot.

9.3 ADA

The ADA software can do the image processing and has been found to be satisfactory. Due to the results it was able to obtain with a 2 MP camera and the sheer number of functions that there wasn't enough time to test.

9.4 Camera

The Photoneo 3D MotionCam can solve the strap identification and colour identification, but not very robustly.

Bibliography

[1] European Union. *EU Industrial Policy*. Accessed: Feb. 23, 2026. 2024. URL: https://www.consilium.europa.eu/en/policies/eu-industrial-policy/.

[2] European Union. "Post COVID-19 Value Chains: Options for Reshoring Production Back to Europe in a Globalised Economy". In: *Policy Department for External Relations* (2021).

[3] C. Gabellieri et al. "Autonomous Unwrapping of General Pallets: A Novel Robot for Logistics Exploiting Contact-Based Planning". In: *IEEE Transactions on Automation Science and Engineering* 20.2 (2023), pp. 1194–1211. DOI: 10.1109/TASE.2022.3182362.

[4] Jason Geng. "Structured-Light 3D Surface Imaging: A Tutorial". In: *Advances in Optics and Photonics* 3.2 (2011), pp. 128–160. DOI: 10.1364/AOP.3.000128.

[5] Zhiyuan Liang et al. "Intelligent Recognition and Handling of Carton Packing Straps Using 3D Laser Vision and Deep Learning". In: *Proc. 2nd Int. Conf. Industrial Automation and Robotics (IAR)*. 2025, pp. 85–90. ISBN: 979-8-4007-1600-3. DOI: 10.1145/3778886.3778899.

[6] Márk Muliter et al. "Designing a Camera-Based Box Strap Detector for Robotic Depalletizing". In: *Proc. Intelligent Robotics FAIR (IntRob)*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 117–123. DOI: 10.1145/3759355.3759630.

[7] Daher Naseem and Shammas Elie. "Sensorless Localization of a Minimally-Actuated Robotic System for Automated Pallet De-strapping". MA thesis. American University of Beirut, 2021.

[8] NIST. *4.4.3.1 Least Squares*. Accessed: May 19, 2026. 2021. URL: https://www.itl.nist.gov/div898/handbook/pmd/section4/pmd431.htm.

[9] Oracle Corporation. *Introduction to Oracle VM VirtualBox*. Accessed: Apr. 14, 2026. n.d. URL: https://www.virtualbox.org/manual/topics/Introduction.html.

[10] Photoneo. *MotionCam-3D Color*. Accessed: May 4, 2026. n.d. URL: https://www.photoneo.com/motioncam-3d/.

[11] Photoneo. *PhoXi 3D Scanner*. Accessed: May 4, 2026. n.d. URL: https://www.photoneo.com/phoxi-3d-scanner/.

[12] Python Software Foundation. *What Is Python?* Accessed: Apr. 14, 2026. 2025. URL: https://docs.python.org/3/faq/general.html#what-is-python.

[13] SCHUNK GmbH & Co. KG. *Co-act EGP-C 40 Catalog Section*. Accessed: Apr. 14, 2026. n.d. URL: https://d16vz4puxlsxm1.cloudfront.net/asset/076200133045-Prod/document_rb43s6iuit23v9sch869jk325h/Co-act%20EGP-C%2040%20Catalog%20section.pdf.

[14] Universal Robots. *Offline Simulator CB-Series Non-Linux URSim 3.15.8*. Accessed: Apr. 14, 2026. n.d. URL: https://www.universal-robots.com/download/software-cb-series/simulator-non-linux/offline-simulator-cb-series-non-linux-ursim-3158/.

[15] Universal Robots. *PolyScope 5*. Accessed: Apr. 14, 2026. n.d. URL: https://www.universal-robots.com/products/polyscope-5/.

[16] Universal Robots. *PolyScope 5 URCap*. Accessed: Apr. 14, 2026. n.d. URL: https://www.universal-robots.com/developer/ur-cap/polyscope-5-urcap/.

[17] Universal Robots. *Script Manual e-Series and UR-Series SW 5*. Accessed: Apr. 14, 2026. n.d. URL: https://s3-eu-west-1.amazonaws.com/ur-support-site/219817/scriptmanualG5_.pdf.

[18] Universal Robots. *UR10e e-Series Collaborative Robot Datasheet*. Accessed: Apr. 17, 2026. n.d. URL: https://www.universal-robots.com/media/1807466/ur10e_e-series_datasheets_web.pdf.

[19] Universal Robots. *UR12e Collaborative Robot*. Accessed: Apr. 17, 2026. n.d. URL: https://www.universal-robots.com/products/ur12e/.

[20] Universal Robots. *URScript Programming Language*. Accessed: Apr. 14, 2026. n.d. URL: https://www.universal-robots.com/developer/urscript/.

[21] W3Schools. *Python Introduction*. Accessed: Apr. 14, 2026. n.d. URL: https://www.w3schools.com/python/python_intro.asp.

[22] Deborah Winkler and Adnan Seric. "COVID-19 Could Spur Automation and Reverse Globalisation – To Some Extent". In: *CEPR VoxEU* (2020).

[23] Zebra. *Aurora Design Assistant*. Accessed: May 4, 2026. n.d. URL: https://www.zebra.com/gb/en/software/machine-vision-and-fixed-industrial-scanning-software/aurora-design-assistant.html.

[24] Cheng Zhang. *TCP Sockets*. Accessed: Apr. 13, 2026. n.d. URL: https://users.cs.fiu.edu/~czhang/teaching/cen4500/project/TCP_Sockets.htm.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY