KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Embedded Systems

First cycle, 30 credits

Implementation of Bolt Detection and Visual-Inertial Localization Algorithm for Tightening Tool on SoC FPGA

MUHAMMAD IHSAN AL HAFIZ

Stockholm, Sweden, 2023

Implementation of Bolt Detection and Visual-Inertial Localization Algorithm for Tightening Tool on SoC FPGA

MUHAMMAD IHSAN AL HAFIZ

Master's Programme, Embedded Systems, 120 credits
Date: September 13, 2023

Supervisors: Dimitrios Stathis, Sofia Olsson
Examiner: Ahmed Hemani
School of Electrical Engineering and Computer Science
Host company: Atlas Copco
Swedish title: Implementering av bultdetektering och visuell tröghetslokaliseringsalgoritm för åtdragningsverktyg på SoC FPGA

Abstract

With the emergence of Industry 4.0, there is a pronounced emphasis on the necessity for enhanced flexibility in assembly processes. In the domain of bolt-tightening, this transition is evident. Tools are now required to navigate a variety of bolts and unpredictable tightening methodologies. Each bolt, possessing distinct tightening parameters, necessitates a specific sequence to prevent issues like bolt cross-talk or unbalanced force.

This thesis introduces an approach that integrates advanced computing techniques with machine learning to address these challenges in the tightening areas. The primary objective is to offer edge computation for bolt detection and tightening tools' precise localization. It is realized by leveraging visual-inertial data, all encapsulated within a System-on-Chip (SoC) Field Programmable Gate Array (FPGA).

The chosen approach combines visual information and motion detection, enabling tools to quickly and precisely do the localization of the tool. All the computing is done inside the SoC FPGA. The key element for identifying different bolts is the YOLOv3-Tiny-3L model, run using the Deep-learning Processor Unit (DPU) that is implemented in the FPGA. In parallel, the thesis employs the Error-State Extended Kalman Filter (ESEKF) algorithm to fuse the visual and motion data effectively. The ESEKF is accelerated via a full implementation in Register Transfer Level (RTL) in the FPGA fabric.

We examined the empirical outcomes and found that the visual-inertial localization exhibited a Root Mean Square Error (RMSE) position of 39.69 mm and a standard deviation of 9.9 mm. The precision in orientation determination yields a mean error of 4.8 degrees, offset by a standard deviation of 5.39 degrees. Notably, the entire computational process, from the initial bolt detection to its final localization, is executed in 113.1 milliseconds.

This thesis articulates the feasibility of executing bolt detection and visual-inertial localization using edge computing within the SoC FPGA framework. The computation trajectory is significantly streamlined by harnessing the adaptability of programmable logic within the FPGA. This evolution signifies a step towards realizing a more adaptable and error-resistant bolt-tightening procedure in industrial areas.

Keywords

Bolt detection, Visual-Inertial localization, System-on-Chip (SoC), Field-Programmable Gate Array (FPGA), Machine learning, Perspective-n-Points,

Error-State Extended Kalman Filter (ESEKF), High-Level Synthesis (HLS), YOLO, Tightening tool

Sammanfattning

Med framväxten av Industry 4.0, finns det en uttalad betoning på nödvändigheten av ökad flexibilitet i monteringsprocesser. Inom området bultåtdragning är denna övergång tydlig. Verktyg krävs nu för att navigera i en mängd olika bultar och oförutsägbara åtdragningsmetoder. Varje bult, som har distinkta åtdragningsparametrar, kräver en specifik sekvens för att förhindra problem som bultöverhörning eller obalanserad kraft.

Detta examensarbete introducerar ett tillvägagångssätt som integrerar avancerade datortekniker med maskininlärning för att hantera dessa utmaningar i skärpningsområdena. Det primära målet är att erbjuda kantberäkning för bultdetektering och åtdragningsverktygs exakta lokalisering. Det realiseras genom att utnyttja visuella tröghetsdata, allt inkapslat i en System-on-Chip (SoC) Field Programmable Gate Array (FPGA).

Det valda tillvägagångssättet kombinerar visuell information och rörelsedetektering, vilket gör det möjligt för verktyg att snabbt och exakt lokalisera verktyget. All beräkning sker inuti SoC FPGA. Nyckelelementet för att identifiera olika bultar är YOLOv3-Tiny-3L-modellen, som körs med hjälp av Deep-learning Processor Unit (DPU) som är implementerad i FPGA. Parallellt använder avhandlingen algoritmen Error-State Extended Kalman Filter (ESEKF) för att effektivt sammansmälta visuella data och rörelsedata. ESEKF accelereras via en fullständig implementering i Register Transfer Level (RTL) i FPGA-strukturen.

Vi undersökte de empiriska resultaten och fann att den visuella tröghetslokaliseringen uppvisade en Root Mean Square Error (RMSE) position på 39,69 mm och en standardavvikelse på 9,9 mm. Precisionen i orienteringsbestämningen ger ett medelfel på 4,8 grader, kompenserat av en standardavvikelse på 5,39 grader. Noterbart är att hela beräkningsprocessen, från den första bultdetekteringen till dess slutliga lokalisering, exekveras på 113,1 millisekunder.

Denna avhandling artikulerar möjligheten att utföra bultdetektering och visuell tröghetslokalisering med hjälp av kantberäkning inom SoC FPGA-ramverket. Beräkningsbanan är avsevärt effektiviserad genom att utnyttja anpassningsförmågan hos programmerbar logik inom FPGA. Denna utveckling innebär ett steg mot att förverkliga en mer anpassningsbar och felbeständig skruvdragningsprocedur i industriområden.

Nyckelord

Bultdetektering, visuell-tröghetslokalisering, System-on-Chip (SoC), Field-Programmable Gate Array (FPGA), Machine Learning, Perspective-n-Points, Error-State Extended Kalman Filter (ESEKF), High-Level Synthesis (HLS), YOLO, åtdragningsverktyg

Acknowledgments

I would like to thank Sofia Olsson and Dimitrios Stathis as my supervisors for guiding me in my thesis project. I am also sincerely grateful to my examiner, Prof. Ahmed Hemani, for his feedback, guidance, and insights for my thesis project. I thank Atlas Copco for providing the resources and environment conducive to my research. I also want to say thank you to the many researchers and scholars that I cited in this report. Their original work has been the base for my research. Last but not least, I am forever grateful for my family and friends, who always support me.

Stockholm, September 2023
Muhammad Ihsan Al Hafiz

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

AP	Average Precision
AXI	Advanced eXtensible Interface
CLB	Configurable Logic Block
CNN	Convolution Neural Networks
Cobot	Collaborative Robot
CPU	Central Processing Unit
DPM	Deformable Parts Model
DPU	Deep-learning Processor Unit
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
ESEKF	Error-State Extended Kalman Filter
FPGA	Field-Programmable Gate Array
HLS	High-Level Synthesis
IMU	Inertial Measurement Unit
IoU	Intersection over Union
IP	Intellectual Property
KTH	KTH Royal Institute of Technology
LUT	Look-Up-Table
MAC	Multiply-Accumulate
mAP	mean Average Precision
PC	Personal Computer
PL	Programmable Logic
PnP	Perspective-n-Point
PS	Processing System
R-CNN	Regions with Convolutional Neural Networks

RAM	Random Access Memory
RMSE	Root Mean Square Error
RTDE	Real-Time Data Exchange
RTL	Register Transfer Level
SoC	System-on-Chip
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
USB	Universal Serial Bus
YOLO	You Only Look Once

Chapter 1

Introduction

In this thesis, we examine the implementation of bolt detection and visual-inertial localization on a tightening tool using System-on-Chip (SoC) Field-Programmable Gate Array (FPGA).

1.1 Background

Commonly, the assembly line for manufacturing has a specific design for one type of product. In the evolving landscape of Industry 4.0, a significant shift towards enhanced flexibility in assembly processes is observed. The flexible assembly line means that the assembly line can be used for several types of products and be used as modular. Manufacturer such as Scania wants to shift to a more flexible assembly line [1]. This trend for greater flexibility, though beneficial in many respects, has inadvertently led to an increase in the complexity of these procedures. It can introduce several types of human error due to the complexity of the procedure that is flexible. A typical source of such errors is related to the improper use of tightening tools within manufacturing environments. The importance of the bolt tightening sequence lies in its influence on achieving a balanced force load, which directly impacts the overall assembly's structural integrity, longevity, and performance [2], [3], [4]. Misplacement of these tools or the incorrect sequencing of bolts frequently contributes to issues known as 'bolt cross-talk'. This phenomenon is characterized by an uneven load distribution, which is far from optimal within an assembly setup. In addition, the wrong tightening parameters, such as torque value for several types of bolts, can become another type of human error in the flexible assembly line.

In terms of a bolt cross-talk problem, numerous corrective strategies have

been proposed to mitigate the incidence of bolt cross-talk, among which vision and tracking systems hold considerable promise. Previous research conducted by Soni in 2020 [5] provides encouraging evidence of the effectiveness of these strategies. By employing a combination of camera technology and machine learning algorithms, Soni [5] could alert operators to the incorrect placement of tools with an impressive probability of over 85%.

In an attempt to build upon this promising foundation, this study aspires to implement bolt detection with machine learning in conjunction with a localization algorithm. This system is particularly devised for tightening tools that demand low-power and compact devices, such as SoC FPGAs. This strategy considers visual and inertial data, as well as the bolt reference position, to provide a comprehensive input for the localization system. With an emphasis on high accuracy and processing speed, the objective is to develop a system capable of real-time processing. It is anticipated that the successful implementation of this research will pave the way for developing future tightening tools that are impervious to bolt cross-talk, facilitated by an accurate localization system.

Atlas Copco, a global leader in providing tightening tools within the Industrial Technique business sector, is expected to benefit significantly from the outcomes of this research. Conducted within the Total Work Station division of Atlas Copco, this research aims to greatly enhance the accuracy and reliability of tightening processes within the industrial context. Ultimately, the results from this study could contribute to the broader objectives of improving operational efficiency, reducing error rates, and ensuring the longevity of assembly systems in various industrial settings.

1.2 Problems

1.2.1 Original problem and definition

- How accurate is the machine learning that can differentiate the bolt type?
- How accurate is the implementation of tightening tool localization based on bolt detection and visual-inertial localization?
- How fast the implementation of the visual-inertial localization algorithm on SoC FPGA can be achieved for real-time application?

1.2.2 Scientific and engineering issues

- Bolt Detection: The accuracy of machine learning in differentiating between the types of bolts used in the assembly process needs to be evaluated. The system must be trained and tested on various bolts to ensure its robustness and effectiveness.
- Tightening Tool Localization: The accuracy of implementing the tightening tool localization algorithm based on bolt detection and visual-inertial data needs to be assessed. This requires the development and integration of a reliable visual and inertial sensor system to provide accurate localization information for the tightening tool.
- Real-time Processing: Implementing the visual-inertial localization algorithm on SoC FPGA needs to be optimized for real-time processing. This requires designing and developing an efficient hardware architecture and software algorithms to provide accurate and timely localization information for the tightening tool.

1.3 Purposes

The purposes of this thesis are the following

- To provide the implementation of the visual-inertial localization algorithm implemented on SoC FPGA.
- To demonstrate the feasibility and effectiveness of implementing the visual-inertial localization algorithm on SoC FPGA for the tightening tool.
- To provide insights into the potential benefits and limitations of using SoC FPGA to implement visual-inertial localization algorithms.

The benefit of the project in the technical things and intelligent proprietary will be acquired by Atlas Copco as the host company. The result of this research will open the possibility of enhancing the future tightening process for the Atlas Copco industry with no human error in the tightening process. On the other hand, in the scientific contribution, the project will bring benefit to KTH Royal Institute of Technology (KTH) and the scientific community because of the contribution to the knowledge of computer vision, hardware implementation, and object localization with a published thesis

report. Moreover, the research supports the 2030 agenda for sustainable development from the United Nations in sustainability goals 9: Industries, Innovation, and Infrastructure.

1.4 Goals

The goal of this project is to provide the hardware implementation of tightening tool localization based on bolt detection and visual-inertial localization algorithm. This has been divided into the following three sub-goals:

1. Subgoal 1: Bolt detection algorithm that is implemented in FPGA with YOLOv3-Tiny-3L has mean Average Precision (mAP) value minimum 33 based on default YOLOv3-Tiny model [6].

2. Subgoal 2: The visual-inertial localization algorithm can provide the average accuracy less than 100 mm for the translation.

3. Subgoal 3: The execution time for Bolt detection and localization process is less than 150 ms.

A related previous publication defined subgoal 1, whereas subgoals 2 and 3 were defined by discussing with host industry requirements. The deliverable for this project is an implemented SoC FPGA for tightening tool localization using a camera and Inertial Measurement Unit (IMU) sensor.

1.5 Research Methodology

This study employs an empirical research method [7], aiming to present a feasible implementation of bolt detection alongside a visual-inertial localization algorithm for tightening tools on SoC FPGA. This approach is predicated on combining machine learning and localization algorithms to detect bolts and track the positioning of the tightening tool. The input data for the localization system will be visual and inertial data in conjunction with the bolt reference position.

The research employs a quantitative strategy to evaluate the performance of the machine learning algorithm and the implementation of the visual-inertial localization methodology. Quantitative research leverages mathematical models and statistics to analyze and interpret data in response to research

questions. In the context of this study, the bolt recognition and visual-inertial localization algorithm on SoC FPGA for real-time applications are investigated for accuracy and execution speed. Quantitative data collection and analysis are essential to resolve these issues. While qualitative research, which utilizes observations, interviews, and other subjective methods, offers profound insights into human experiences and perceptions, it may not be the optimal approach for this research. The focus of this study is to measure and quantify the performance of the bolt detection and localization algorithm in real-time scenarios.

The SoC FPGA implementation will be tested with real data from the laboratory. The FPGA implementation will also inspect the utilization and power estimation of the logic block. The empirical evidence obtained from the experimental results will be utilized to form a definitive conclusion for this research.

This research operates under the assumption that the incidence of errors in assembly processes can be reduced through the application of advanced technology, specifically computer vision and sensor IMU. Furthermore, it is posited that the implementation of a bolt detection and visual-inertial localization algorithm for tightening tools can contribute to the development of an error-free, flexible tightening process in the future.

1.6 Delimitations

The following are the delimitations of this project

- The project will only cover localization and exclude the mapping.
- The testing and evaluation will be done in the laboratory of the tightening process.
- The system has the first location reference, and the location of every bolt target for tightening has been known.
- The ground truth reference is obtained from the collaborative robot platform.
- The bolt detection consists of two types of bolt
- The data evaluation in the result section is done with a dataset that has been taken before.

1.7 Structure of the thesis

Chapter 2 presents relevant background information about object detection, localization algorithms, and other topics related to this study. Chapter 3 breaks down the methodology and the approach to solving the stated in sub-chapter 1.2. The next chapter, Chapter 4, describes how the implementation was done for bolt detection and localization algorithm. In Chapter 5, the most important results are shown. In Chapter 6, the results will be discussed further. In the last chapter, Chapter 7, the conclusions of the results are stated, and discussions about the limitations of the project and possible future work in this area are made.

Chapter 2

Background

This chapter provides basic background information about object detection and localization algorithms, including visual and inertial data. Additionally, this chapter describes the algorithm for implementing machine learning on the FPGA. The chapter also describes related work in bolt detection and visual-inertial localization.

2.1 Object Detection

Object detection is one of the cases that is commonly addressed by machine learning. the aim is to detect all of the objects and recognize the classes, for example, cars, cats, trains, and so on, in an image. Object detection consists of object recognition and image classification. Object recognition is utilized to recognize the possible object without knowing the label of the object. Image classification is used to classify the object within the defined class. Therefore, the output of object detection consists of object class labels, confidence, locations, and the size data of the object.

2.1.1 You Only Look Once (YOLO)

In 2016, Redmon et al. [8] presented a new approach for object detection that is called You Only Look Once (YOLO). The basic idea is to use single-step detection for detecting the object instead of two-step detection. Previously, the common algorithm for object detection used a two-step process that consisted of object recognition and an image classifier, such as the Fast Region-based Convolutional Network method (Fast R-CNN) [9]. When extrapolating from natural images to other domains like artwork, YOLO performs better than

other detection techniques like Deformable Parts Model (DPM) and Regions with Convolutional Neural Networks (R-CNN) [8].

The YOLO object detection model is known for its simplicity, as it utilizes a single convolutional neural network to predict multiple bounding boxes and their corresponding class probabilities. YOLO also trains on full images and optimizes detection performance directly, making it a more efficient and effective approach compared to traditional methods of object detection.

YOLO has been developed and upgraded with several release versions. From the first release in 2016 from Redmon et al. [8], which is YOLOv1, until the last update in 2023, which is YOLOv8, published by Ultralytics [10]. Due to the limitation of support of YOLO implementation for Xilinx FPGA, in this research, we will use YOLOv3-Tiny-3L for bolt detection implementation.

2.1.2 YOLOv3

In 2018, Redmon and Farhadi [11] presented some updates to the YOLO algorithm by introducing YOLOv3, which has some design changes to make it better. Even though it is bigger than the previous version, it is still faster than the comparable model. YOLOv3 achieves 57.5 AP_{50} in 51 ms, which is faster than RetinaNet, which achieves 57.5 AP_{50} in 198 ms when both models were run on a Titan X.

The YOLOv3 approach views object detection as a form of regression problem. This approach employs a single, feed-forward convolutional neural network to directly calculate class probabilities and bounding box adjustments from entire images. It discards the need for generating region proposals and feature re-sampling by integrating all stages into a singular network, thereby realizing a truly end-to-end detection mechanism. The YOLOv3 strategy partitions the input image into a grid made up of SxS cells. If an object's centre point falls within one of these grid cells, that specific cell becomes responsible for the object's detection. Each cell is designed to predict the bounding box location data and determine the 'objectness' scores associated with these bounding boxes [12].

With the successful implementation of YOLO as an object detection algorithm, there is a need for a version with lightweight and more speed. YOLOv3-tiny was introduced [13] to improve the speed and detection accuracy. YOLOv3-Tiny achieved more than 200 FPS on the Pascal VOC-2007 dataset. YOLOv3-tiny is a streamlined network comprised of 13 convolution layers, 6 max-pooling layers, 1 upsampling layer, 1 cascading layer, and 2 output points. The initial network output measures 13×13×255

and is generated by subjecting the input layer to a sequence of 6 alternating convolution and pooling layers, followed by an additional 4 convolution layers. The second network output measures 26×26×255. This is achieved by merging the fifth convolution layer's output with the sixth pooling layer's output, which is then passed through two more convolution layers [13]. Furthermore, to improve the accuracy of YOLOv3-tiny, in 2018, Gong et al. [14] introduced the improved version of YOLOv3-tiny. They added more networks and made the output from 2 layers into 3 layers. After evaluating their dataset, they achieved 6.3% better accuracy with a detection speed of 31.8 FPS [14].

2.2 FPGA design platform

2.2.1 PYNQ Platform

PYNQ is an adaptive computing platform that uses Python language and libraries to program underlying an integrated processor in the FPGA. PYNQ is developed by AMD for their Xilinx FPGA that has integrated processor Zynq, Zynq UltraScale+, Zynq RFSoC, Alveo accelerator boards and AWS-F1. The aim is developers can exploit the benefits of integrating microprocessors and programmable logic in one chip to expand more possibilities of complex and exciting electronic systems applications [15].

PYNQ utilizes Python programming with jupyter notebook, which is a browser-based interactive computing environment. PYNQ is implemented under the Linux operating system. The programmable logic for the FPGA is implemented with the concept of overlays. Overlays are the group of files that consist of bitstream files, design blocks, and device trees from digital logic design. overlays are loaded in a similar way that Python libraries are loaded in Python programming.

2.2.2 Vitis AI

Xilinx® Vitis AI is an integrated development environment for accelerating AI inference on Xilinx platforms. This toolchain provides the user with optimized Intellectual Property (IP), tools, libraries, and models, as well as resources such as example designs and tutorials to assist in the development procedure. It is created with high efficiency and user-friendliness in mind, releasing the maximum potential of AI acceleration on Xilinx SoC [16].

Vitis AI consists of the following essential elements [16]:

- Deep-learning Processor Units (DPUs) - Configurable computational engines optimized for neural convolution networks. IP circuits that are both efficient and scalable and can be tailored to satisfy the requirements of various applications and devices.
- Model Zoo - An exhaustive collection of pre-trained and pre-optimized models that are prepared for deployment on Xilinx devices.
- Model Inspector - a tool and methodology for validating model architecture support for developers.
- Optimizer - An optional, commercially licensed tool that allows users to refine a model by as much as 90 per cent.
- Quantizer - A potent quantizer that supports the quantization, calibration, and precise refining of models.
- Compiler - Compiles the quantized model for execution on the DPU accelerator being targeted.
- Runtime (VART) - An embedded application inference runtime.
- Profiler - Analyzes the effectiveness and utilization of AI inference implementations on the DPU.
- Library - Provides high-level C++ APIs for embedded and data center AI applications.

In this research, we will utilize the Vitis AI to quantize the YOLOv3-Tiny-3L model from the floating point unit into int8-bit data. The quantization includes a calibration process with training data. After the quantization, we compile the quantized model into an xmodel file for input into DPU core.

2.2.3 Deep-learning Processing Unit (DPU)

The Xilinx® Deep-Learning Processor Unit (DPU) is a Central Processing Unit (CPU) or programmable engine that is specifically designed for convolutional neural network calculation and processing. The unit incorporates a register configure module, data controller module, and convolution computing module. It uses a specific instruction set for doing the calculation that is specially designed for doing efficient calculations for multiple Convolution Neural Networks (CNN) architecture [17].

The DPU is used as an IP-core that is integrated as a block in programmable logic design. The DPU only supports the development environment from Xilinx, such as Vivado and Vitis software. it communicates directly with Random Access Memory (RAM) for storing or fetching data and instructions. The data machine learning model and instruction are supplied from the xmodel file from the compiled file Vitis AI. The DPU IP core has configurable hardware architecture, including B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096. Those hardware architectures indicate the number of Multiply-Accumulate (MAC) operation that is used inside the DPU [17]. In this research, the DPU will be used to implement bolt detection with the YOLOv3-Tiny-3L model.

2.3 Perspective-n-Points

The term Perspective-n-Point (PnP) was devised by Fischler and Bolles [18] for the problem of determining the pose of a calibrated camera from n-points correspondences between three-dimensional reference points and their two-dimensional projections [19]. Iterative or noniterative strategies are used to solve the PnP problem. Noniterative approaches are effective, but one of their drawbacks is instability when there is noise, especially when the number of points is less than or equal to 5. Since 1841, there have been numerous closed-form solutions to the 3-point problem, the smallest subset of PnP [18], [19].

Good camera pose estimation is achieved by knowing the correspondences between 2D points in the image and 3D points in the object. The transformation matrix ${}^{c}\mathbf{T}_{w}$ is used to define the rotation $\mathbf{R}$ and translation $\mathbf{t}$ between object position and camera position [20]. However, it has an assumption that 3D position points in the object have accurate values. fig. 2.1 illustrates the PnP problem with the camera and object related to world coordinate points.

$$ {}^{c}\mathbf{T}_{w} = \begin{pmatrix} {}^{c}\mathbf{R}_{w} & {}^{c}\mathbf{t}_{w} \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix}, \tag{2.1} $$

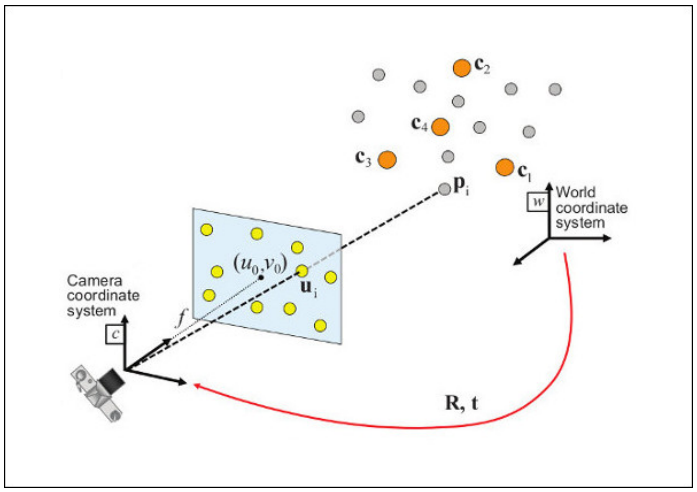


Figure 2.1: Perspective-n-Points illustration [21]

The complete relation between perspective projection in the image and the actual point position of the object is shown by eq. (2.2)

$$\bar{\mathbf{x}} = \mathbf{K}\ \mathbf{\Pi}\ {}^{c}\mathbf{T}_{w}\ {}^{w}\mathbf{X}, \tag{2.2}$$

where $\bar{\mathbf{x}} = (u, v, 1)$ is perspective projection coordinates that are expressed in pixel of the point in the image; ${}^{w}\mathbf{X} = ({}^{w}X, {}^{w}Y, {}^{w}Z, 1)$ is 3D points location with similar correspond to 2D points; $\mathbf{K}$ is the intrinsic parameters matrix of the camera and is defined by [20]:

$$\mathbf{K} = \begin{pmatrix} p_x & 0 & u_0 \\ 0 & p_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.3}$$

The coordinates $(u_0, v_0, 1)$ represent the principal point, which is where the optical axis intersects the image plane. The parameters p_x and p_y are influenced by both the focal length of the lens, denoted as f, and the pixel size. More specifically, p_x is derived by dividing the lens's focal length by the pixel's width (l_x), represented by the formula $p_x = f/l_x$. Similarly, p_y is the ratio of the focal length to the height of a pixel (l_y), defined as $p_y = f/l_y$. in the case of a perspective projection model, The projection matrix $\mathbf{\Pi}$ is given by [20]:

$$\mathbf{\Pi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

2.4 Collaborative Robot

The Collaborative Robot (Cobot) is the robot platform that is produced by Universal Robot. The data from the Cobot can be used as reference pose data because it contains position and orientation with a fixed reference in its mounting base. The Cobot has feature data. A feature represents a six-dimensional pose of an object that contains position and orientation relative to the robot base. The robot base is located in the base mounting of the robot that is shown in Figure 2.2. The robot base can be defined as a base feature whose position and orientation start from zero [22].

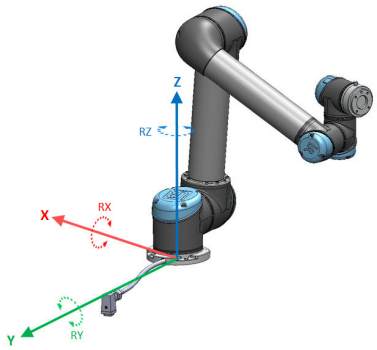


Figure 2.2: Cobot based position reference [22]

The Cobot has a tool center point, which is the point that has a pose relative to the base point. The tool center point has a tool feature which is a six-dimensional pose (position and orientation) from the mounted tool in the Cobot. The tool center point can be adjusted to follow the mounting tool in the Cobot. It is represented by Figure 2.3

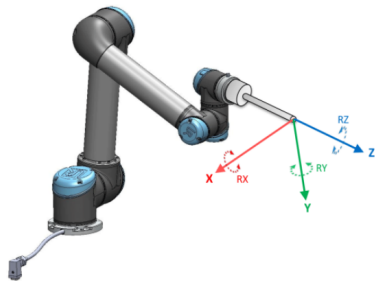


Figure 2.3: Device position related to Cobot [22]

2.5 Error State Extended Kalman Filter

The Error-State Extended Kalman Filter (ESEKF) is a variant of the Extended Kalman Filter (EKF) that estimates the error in states rather than the states themselves [23]. This approach, which uses linear error state dynamics, allows for optimal updates in error states and their covariance without the need for repeated tuning related to noise covariances [23]. The ESEKF demonstrates superior performance in terms of accuracy when integrating data from a multi-sensor system, as compared to estimations derived from a single-sensor state [24]. This highlights its effectiveness in enhancing the precision of sensor data fusion. Given these advantages, this study employs the ESEKF to optimally fuse data from visual pose estimation results, inertial sensor data, and reference from the bolt reference position, aiming to achieve improved data fusion outcomes.

The fundamental idea behind the ESEKF involves splitting the state vector into two components: the nominal state, referred to as x, and the error state, denoted as δx [25]. In the system that consists of IMU data and updated position, the state consists of position, velocity and orientation. Nominal state ($\mathbf{x}_k$) that is updated in every sampling rate k is represented by

$$\mathbf{x}_k = [\, \mathbf{p}_k, \mathbf{v}_k, \mathbf{q}_k \,]^T \in R^{10} \tag{2.4}$$

The IMU measurement ($\mathbf{u}_k$) that contains specific force ($\mathbf{f}_k$) and angular rate ($\boldsymbol{\omega}_k$) measurement is defined by:

$$\mathbf{u}_k = [\, \mathbf{f}_k, \boldsymbol{\omega}_k \,]^T \in R^6 \tag{2.5}$$

The motion model with k sampling that consist of position update ($\mathbf{p}_k$), velocity update ($\mathbf{v}_k$), and orientation update ($\mathbf{q}_k$) is represented by:

$$\mathbf{p}_k = \mathbf{p}_{k-1} + \Delta t\ \mathbf{v}_{k-1} + \frac{\Delta t^2}{2}(\mathbf{C}_{ns}\mathbf{f}_{k-1} + g) \tag{2.6}$$

$$\mathbf{v}_k = \mathbf{v}_{k-1} + \Delta t(\mathbf{C}_{ns}\mathbf{f}_{k-1} + g) \tag{2.7}$$

$$\mathbf{q}_k = \mathbf{q}_{k-1} \otimes \boldsymbol{q}\ (\boldsymbol{\omega}_{k-1}\ \Delta t) = \Omega(\boldsymbol{q}\ (\boldsymbol{\omega}_{k-1}\ \Delta t))\ \mathbf{q}_{k-1} \tag{2.8}$$

where Δt is the delta time between sampling, $\mathbf{C}_{ns}$ is the rotation matrix on previous sampling, and g is gravity. The nominal state moves forward in time. However, this process does not consider unpredictable elements like

noise and disturbances. The ESEKF algorithm, on the other hand, accounts for these variations by tracking them in an 'error state vector'. The next phase involves looking at these errors in a simplified, linear way to understand how they change over time. The following is the error state ($\delta \boldsymbol{x}_k$)

$$\delta \boldsymbol{x}_k = [\, \delta \boldsymbol{p}_k,\ \delta \boldsymbol{v}_k,\ \delta \boldsymbol{\Phi}_k \,]^T \in R^9 \tag{2.9}$$

When it moves in time, it becomes error dynamics that is defined by

$$\delta \boldsymbol{x}_k = \boldsymbol{F}_{k-1}\ \delta\ \boldsymbol{x}_{k-1} + \boldsymbol{L}_{k-1}\ \boldsymbol{n}_{k-1} \tag{2.10}$$

where $\boldsymbol{n}_k$ is measurement noise.

$$\boldsymbol{n}_k \sim N(\boldsymbol{0},\ \boldsymbol{Q}_k) \tag{2.11}$$

Motion model jacobian ($\boldsymbol{F}_{k-1}$), Motion model noise jacobian ($\boldsymbol{L}_{k-1}$), and IMU noise covariance ($\boldsymbol{Q}_k$) are defined by

$$\boldsymbol{F}_{k-1} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} \cdot \Delta t & 0 \\ 0 & \boldsymbol{I} & -[\boldsymbol{C}_{ns}\ \boldsymbol{f}_{k-1}]_X\ \Delta t \\ 0 & 0 & \boldsymbol{I} \end{bmatrix} \tag{2.12}$$

$$\boldsymbol{L}_{k-1} = \begin{bmatrix} 0 & 0 \\ \boldsymbol{I} & 0 \\ 0 & \boldsymbol{I} \end{bmatrix} \tag{2.13}$$

$$\boldsymbol{Q}_k = \Delta t^2 \begin{bmatrix} \boldsymbol{I} \cdot \sigma_{acc}^2 & 0 \\ 0 & \boldsymbol{I} \cdot \sigma_{gyro}^2 \end{bmatrix} \tag{2.14}$$

where $\boldsymbol{I}$ is identity matrix 3 by 3. The Jacobian matrices of the motion model are used to propagate the state uncertainty forward in time. The state covariance matrix symbolizes the uncertainty in the state. As time progresses, this uncertainty escalates until it is restrained by incoming measurements. The predicted state covariance ($\check{\boldsymbol{P}}_k$) is represented by:

$$\check{\boldsymbol{P}}_k = \boldsymbol{F}_{k-1}\ \boldsymbol{P}_{k-1}\ \boldsymbol{F}_{k-1}^T + \boldsymbol{L}_{k-1}\ \boldsymbol{Q}_{k-1}\ \boldsymbol{L}_{k-1}^T \tag{2.15}$$

The measurement update is utilized to update the state of the system. In the case of measurement, only update one of the variables from the three-state variable, which is position, the measurement model Jacobian ($\boldsymbol{H}_k$) and sensor noise covariance ($\boldsymbol{R}$) are represented as

$$\boldsymbol{H}_k = [\, \boldsymbol{I}\ \ 0\ \ 0 \,] \tag{2.16}$$

$$\boldsymbol{R} = \boldsymbol{I} \cdot \sigma^2_{sensor} \tag{2.17}$$

The Kalman gain is computed by state covariance matrix and inverse state covariance matrix after it is combined with noise covariance and measurement model Jacobian. Kalman gain ($\boldsymbol{K}_k$) is represented by:

$$\boldsymbol{K}_k = \check{\boldsymbol{P}}_k \, \boldsymbol{H}_k^T \, (\boldsymbol{H}_k \, \check{\boldsymbol{P}}_k \, \boldsymbol{H}_k^T + \boldsymbol{R})^{-1} \tag{2.18}$$

Kalman gain is used to compute the error state. the error state uses the measurement update position and compares it with the current nominal state position. The error state ($\delta \boldsymbol{x}_k$) is represented by:

$$\delta \boldsymbol{x}_k = \boldsymbol{K}_k (\boldsymbol{y}_k - \check{\boldsymbol{p}}_k) \tag{2.19}$$

The result of error states is utilized to correct the state variable (position, velocity and orientation). Kalman gain is also utilized to produce corrected state covariance. the following are corrected state variables and state covariance.

$$\hat{\boldsymbol{p}}_k = \check{\boldsymbol{p}}_k + \delta \boldsymbol{p}_k \tag{2.20}$$

$$\hat{\boldsymbol{v}}_k = \check{\boldsymbol{v}}_k + \delta \boldsymbol{v}_k \tag{2.21}$$

$$\hat{\boldsymbol{q}}_k = \boldsymbol{q}(\delta \, \boldsymbol{\Phi}_k) \otimes \check{\boldsymbol{q}}_k \tag{2.22}$$

$$\hat{\boldsymbol{P}}_k = (\boldsymbol{I} - \boldsymbol{K}_k \, \boldsymbol{H}_k) \, \check{\boldsymbol{P}}_k \tag{2.23}$$

The corrected state variables and state covariance are used as the saved variables in time sampling k.

2.6 Related work area

2.6.1 Visual and inertial sensor fusion for mobile X-ray detector tracking

This paper delves into the domain of object position and orientation estimation, focusing particularly on the field of medical imaging. The authors highlight the importance of accurate 3D tracking of medical devices in improving

the quality of medical images, speeding up the imaging workflow, and paving the way for autonomous imaging and robotic operations. They further emphasize that in mobile X-ray imaging, accurate pose tracking of the X-ray detector is crucial for a 2D X-ray imaging system and innovative 3D tomosynthesis systems. The authors scrutinize existing high-accuracy 3D pose tracking techniques, including motion capturing techniques (e.g., Vicon and Optitrack systems) and electromagnetic tracking techniques (e.g., trakSTAR system). However, they point out the prohibitive cost of these solutions for many applications. They also explore computer vision methods, specifically marker-based methods and visual odometry (VO) techniques. Despite their affordability and wide application in augmented reality, robotics, and more, these methods are sensitive to occlusion, light condition changes, and image acquisition errors. To fortify the robustness of vision-based pose tracking, the authors propose a novel approach that combines visual-inertial odometry (VIO) with an IMU sensor. They critique existing VIO algorithms that use the Kalman filter (KF) or extended Kalman filter (EKF) framework, or loosely-coupled or weighted average approaches, for their dependency on motion dynamic model or weight parameters and intrusive installation. The authors present an alternative solution— a novel visual-inertial sensor fusion framework demonstrated via a real-time implementation of a tightly coupled sensor fusion algorithm, the inertial perspective-n-point (IPNP) algorithm. This prototype system integrates visual and inertial sensors, but unlike VIO, it only requires an IMU sensor attached to the object, making it less intrusive and reducing payload. This approach also significantly reduces the number of visual key points required for tracking, compared to the classical perspective-n-point (PnP) algorithm and the five-point visual odometry algorithm. It only needs to detect two key points to track all six degrees of freedom of a planar object, such as a mobile X-ray detector. This results in a 50% reduction in the required number of key points compared with the vision-based perspective-n-point algorithm [26].

2.6.2 Using Perspective-n-Point Algorithms for a Local Positioning System Based on LEDs and a QADA Receiver

This paper presents an innovative approach to indoor local positioning systems (LPS) utilizing LED lighting and a quadrant photodiode angular diversity aperture (QADA) receiver. The LED lighting is transmitted from a set of beacons, and the image position of these transmitters is used to determine the

receiver's 3D pose. This approach essentially frames the receiver's location problem as a Perspective-n-Point (PnP) issue, a classical problem in computer vision and photogrammetry. The researchers designed and validated an infrared positioning system based on LED lighting and a QADA receiver, which they modelled as a perspective camera. The system involved three independent coordinate systems and utilized a synchronized approach with the receiver and transmitters operating simultaneously. The system employed a Code-Division Multiple Access (CDMA) technique to differentiate signals from different beacons. The received signals were then used to calculate the image points for each beacon. The researchers examined non-iterative and iterative PnP methods for the PnP problem. Non-iterative methods, such as Efficient Perspective-n-Point Camera Pose Estimation (EPnP), Robust Non-Iterative Solution of Perspective-n-Point (RPnP), and Infinitesimal Plane-Based Pose Estimation (IPPE), were used. Iterative methods were also examined, known to be more accurate but computationally intensive and requiring near-optimum initialization. The study employed three well-known PnP methods (EPnP, RPnP, and IPPE) to ascertain the pose of the receiver from the image points of each transmitter. Each method provided unique advantages and limitations in solving the PnP problem. EPnP was used for the weighted sum of the eigenvectors of a 12 × 12 matrix, RPnP expressed the PnP problem as a least squares polynomial function, and IPPE exclusively solved the Planar-PnP problem through a homographic transformation between the image coordinates and the 3D plane containing the transmitters. The results of the study showed that the three PnP methods provided accurate localization of the receiver. The system achieved 3D centimeter accuracy inside intelligent environments, similar to previous work using triangulation techniques but with the added advantage of determining the complete pose of the receiver. The system was also scalable to large spaces of 2 × 2 m x, y, and z, respectively. The average absolute errors and standard deviations for the estimation of the receiver's position at three specific points for roll angles of the receiver of gamma=0°, 120°, 210°, and 300° obtained using the IPPE algorithm are 4.33 cm, 3.51 cm, and 28.90 cm; and 1.84 cm, 1.17 cm, and 19.80 cm, respectively, in the coordinates x, y, and z [27].

2.6.3 3-D Positioning System Based QR Code and Monocular Vision

This paper dives into the world of 3D navigation and positioning, particularly aimed at unmanned vehicles and robots. The authors stress the significance of

accurate 3D position and pose determination for these autonomous entities to effectively execute their designed tasks. They further highlight that in both indoor and outdoor settings, precise tracking of the vehicles' or robots' position and orientation is crucial for smooth navigation and operations. The authors scrutinize existing 2D QR code positioning techniques, such as those utilized by the Amazon warehouse robot KIVA and other indoor Automatic Guided Vehicles (AGVs), and highlight their limitations, particularly their inability to extend to 3D environments. They also address the limitations of alternative 3D positioning techniques, which require multiple QR codes in different positions or stereo cameras, thus reducing efficiency and increasing complexity. To overcome these challenges, the authors propose an innovative system that integrates a monocular visual system with QR codes to determine both the autonomous entities' 3D position and orientation. This novel approach uses two main modules - an image processing module and a pose calculation module. The system employs an efficient perspective-n-point (EPnP) camera pose estimation algorithm in the pose calculation module, enabling it to determine the 3D position and the Euler angles of the camera in a physical coordinate system. Unlike previous methods, this approach only needs a single monocular camera and can work with QR codes placed anywhere visible to the camera, significantly reducing environmental restrictions. This design also reduces the complexity and increases the system's efficiency, providing a more reliable solution for 3D navigation and positioning. Experimental testing of this system showed promising results with an error of Euler angles within 3 degrees and an error of translations within 60mm, demonstrating its potential for precise 3D navigation and positioning [28].

2.6.4 Error State Extended Kalman Filter Localization for Underground Mining Environments

This paper delves into the issue of real-time localization of mobile robotic platforms in environments that deny access to Global Navigation Satellite Systems (GNSS). The authors examined the complexities surrounding the establishment of precise positioning in such areas, especially in underground mining contexts, that demand continual updating of layout plans and monitoring of roof subsidence. They underscored the potential of integrating novel technologies such as computer vision, predictive model control, machine learning, and neural networks with robotic equipment for advancing the efficiency of mining operations. To address the localization challenge in

GNSS-denied settings, the authors developed a system fusing data from an IMU, a magnetometer, and encoders. They chose the Error-state Extended Kalman Filter (ES EKF) for its ability to generate symmetrical error gauss distribution for the measurement model, which they posited could lead to superior performance. The ES EKF system works in two stages: vector state propagation based on accelerometer and gyroscope data and correction via measurements from additional sensors. Highlighting the merits of ES EKF, the authors illustrated its ability to isolate IMU measurements from additional sensor data, with its subsequent addition in the correction phase. They found this feature advantageous, as it allows the system to be more resilient to auxiliary sensor failures, an aspect which distinguishes the ES EKF from the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). Validating their approach, they simulated the Robot Operating System (ROS) and Gazebo environment using a mathematical model they created. They generated trajectories for the ES EKF, EKF, and UKF algorithms and calculated absolute position errors for all trajectories. The authors demonstrated that the ES EKF system achieved comparable position errors to those of the EKF and UKF, implying its efficacy in managing localization in complex environments. The paper concluded by acknowledging that all three algorithms could benefit from increased localization accuracy [25].

Chapter 3

Method

3.1 Research Process

Figure 3.1 shows the steps conducted in order to carry out this research.

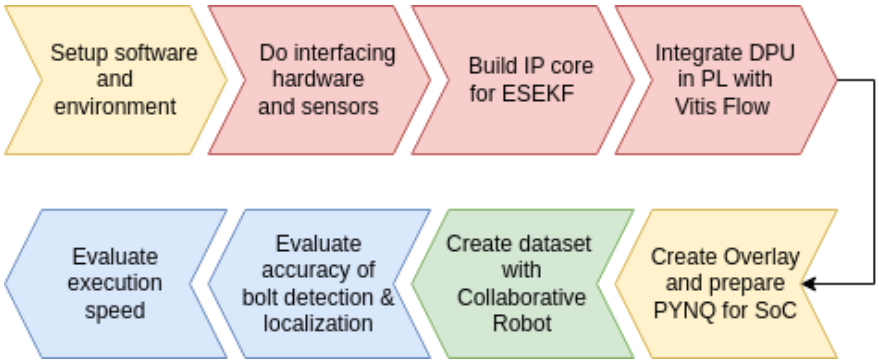


Figure 3.1: Research Process

The research process depicted in Figure 3.1 employs varied colours to denote distinct types of processes. Yellow signifies software-based processes; red denotes hardware-based ones; green highlights data acquisition, and blue indicates data processing and analysis. The following is a detailed explanation of the research processes:

Step 1 The software and research environment encapsulate hardware implementation on the FPGA, High-Level Synthesis (HLS) process design, and machine learning deployment. Hardware implementation necessitates software for IP core building, integration of the IP block into programmable logic and overlay creation. The HLS process

employs Matlab Simulink to implement the algorithm and translate it into Register Transfer Level (RTL) language. Machine learning needs platforms to train the bolt detection data and establish a model for FPGA implementation.

Step 2 This research employs two sensors: a camera and an IMU. The camera interfaces via the Universal Serial Bus (USB) protocol, managed by an ARM processor in the Zynq platform. The IMU interfaces through the Serial Peripheral Interface (SPI) protocol, capturing 3-axis accelerometer and gyroscope data. The programmable logic side of FPGA manages the IMU interface and uses Advanced eXtensible Interface (AXI)-bus to communicate with the ARM processor.

Step 3 The ESEKF algorithm, implemented on the Programmable Logic (PL) side, should follow the IP core format. The IP core's creation entails building a block diagram in Simulink and validating it. The block diagram is then converted into RTL language and IP core through the HLS process, which employs specific settings to adjust the RTL implementation for optimal hardware utilization and runtime.

Step 4 Machine learning implementation in FPGA uses the DPU IP core to expedite convolutional neural network computations. The DPU IP core's integration relies on Vitis flow, a software-oriented FPGA development approach, as it automatically adjusts the DPU connection. The first step is to ready all IP blocks, excluding the DPU IP core, followed by running the Vitis Flow process to generate a bitstream.

Step 5 Once hardware implementation is completed, an overlay is created based on the bitstream and the DPU IP core's hardware architecture. The overlay is vital for the PYNQ platform, allowing interfacing between the Processing System (PS) side and PL side of ZYNQ using Python programming. The PYNQ platform operates under the embedded Linux operating system, running on a ZYNQ ARM processor.

Step 6 Dataset creation amalgamates data from collaborative robots and sensors. The collaborative robot, holding the camera and IMU, simulates movement above the bolt placement setup. This setup comprises a bolt placement sequence of two different types in random configurations. The dataset is built by acquiring synchronized timestamp data from the camera, IMU, and collaborative robot reference during the simulation.

Step 7 Bolt detection accuracy is evaluated by analyzing the sequence of image results from the dataset, confirming the correct detection of two distinct bolt types. Localization assessment compares the 6-DoF pose from camera pose estimation with collaborative robot data.

Step 8 The execution time is evaluated for every function, including sensor data acquisition, bolt detection with machine learning, camera pose estimation calculation, prediction and update processes in ESEKF. Finally, the total execution time is integrated and evaluated against the set target.

3.2 Data Collection

The data collection is done in the workshop laboratory of the total workstation at Atlas Copco. the aim is to create the dataset for testing and evaluating the algorithm. The dataset includes timestamps, position references, orientation references, 3-axis accelerometers, 3-axis gyroscopes, and images. The position references and orientation references are taken from collaborative robots through Transmission Control Protocol (TCP) communication. The dataset is acquired with the FPGA platform by integrating all of the sensors and the collaborative robot data. All of the data is synchronized in timestamps from the SoC FPGA.

3.2.1 Sampling

The data acquisition for this study employs a variety of connections: an SPI connection for the accelerometer, a USB connection for the camera, and an Ethernet TCP connection for pose references from the Cobot. Data collection was conducted with reference to the UNIX timestamp of the SoC FPGA operating system. The sampling rates for the position references, orientation references, 3-axis accelerometers, and 3-axis gyroscopes range from approximately 30Hz to 35Hz. On the other hand, the camera's sampling data range is slightly lower, standing between 10Hz and 12Hz. Data acquisition for the camera operates on an independent thread process, which stores data in a buffer denoted by a flag status. Data acquisition initiates with data collection from the IMU. If data is available, the pose data from the Cobot is also acquired, ideally at a similar timestamp. Subsequently, the process verifies the data from the camera's thread process. If the flag status indicates data availability, the image is captured and saved with a timestamp similar to

when the IMU data was collected. However, in instances where camera data is unavailable, the process continues with no image being saved. It is important to note that the speed and efficiency of data acquisition are influenced by the process of image saving and the latency of intermediary processes, including SPI, USB, and Ethernet connections.

3.2.2 Sample Size

This research utilizes mathematical calculations to derive results, rendering the number of samples inconsequential to the outcome's integrity. Instead, the sample size mirrors the movement scenarios of the tool represented by the localization calculation process outcomes. As such, the sample size was decided upon in collaboration with the industry supervisor to ensure it adequately represents the system's operation. The dataset used to evaluate the algorithm encompasses a total of 8520 data points. This dataset comprises synchronized timestamps, reference data, IMU data, and camera data. The collected data corresponds to a total testing duration of 286 seconds, as determined by the timestamps.

3.2.3 Target Population

The focus of this study is on bolts, selected as the target population. Given the lack of prior studies addressing this topic, the research utilised a simplified version, examining only a minimal selection of bolts for object detection purposes. Consequently, this study incorporates two types of bolts as the target population. The target populations are

- Black bolt, with reference name ”bolt1”
- Silver bolt, with reference name ”bolt2”

3.3 Experimental design

The experimental design illustration is shown by Figure 3.2

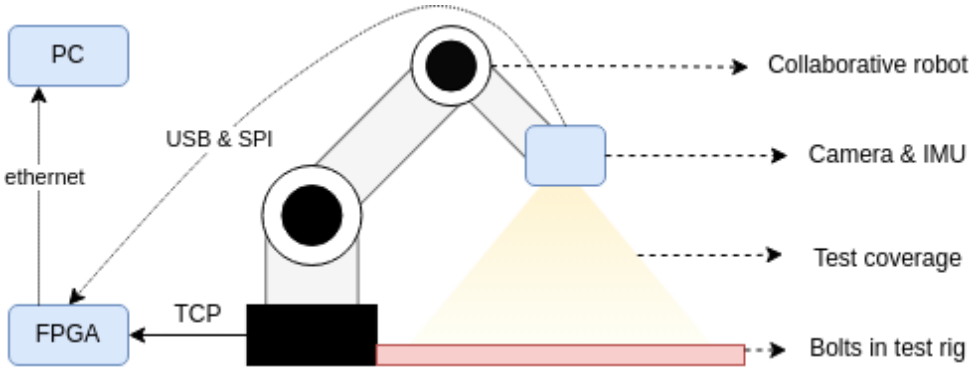


Figure 3.2: Experiment setup

In this study, a collaborative robot is employed to yield precise position and orientation references during experimental trials and tests. Its role is to emulate the tightening tool's movements, mimicking real-world application scenarios. When required, this robot dispatches reference data to the FPGA through TCP communication. The FPGA plays a crucial role in acquiring and consolidating data from different sources. Specifically, it retrieves data from the camera using USB communication and from the IMU through SPI communication.

The camera is strategically positioned to cover only the test rig area, which comprises bolts. As a result, all images captured from this viewpoint exclusively contain the bolt object, which is central to this research. This design ensures the accuracy and relevance of the visual data gathered. To allow full accessibility and functionality, the test rig is located adjacent to the collaborative robot. This positioning ensures the robot's ability to reach all necessary areas during the testing process.

Connectivity between the FPGA and a Personal Computer (PC) is established through an Ethernet connection. The FPGA, operating on an embedded Linux operating system and Python Productivity for Zynq (PYNQ) platform, can be accessed by the PC through Jupyter Notebook. This accessibility is achieved using a specific Internet Protocol (IP) address and a given port. Consequently, the PC can control the processes running on the FPGA and has the capability to upload and modify the code from the PS side of ZYNQ, providing flexibility and adaptability to the experimental setup.

3.3.1 Test environment

The testing environment for this study is exclusively designed to incorporate two types of bolts strategically placed within the test rig. The test rig, devised to simplify and effectively emulate the tightening environment, plays a pivotal role in our setup. This rig features holes spaced 50mm apart, both horizontally and vertically, thereby maintaining a consistent structure for our experiments. Two variants of bolts, black and silver, are incorporated into the testing rig in a random sequence to reflect diverse and unpredictable real-world scenarios. A subset of the black bolts is outfitted with a silver ring, positioned randomly to further enhance the variability of our testing setup.

In order to minimize visual noise and potential distractions during image capture, we have implemented a simple, effective camouflage strategy. Any portions of the test rig that are not occupied by bolts are obscured with white paper. This ensures that our images are focused solely on the bolts, optimizing the accuracy and efficiency of our object detection processes. The detail of the test environment is shown by Figure 3.3

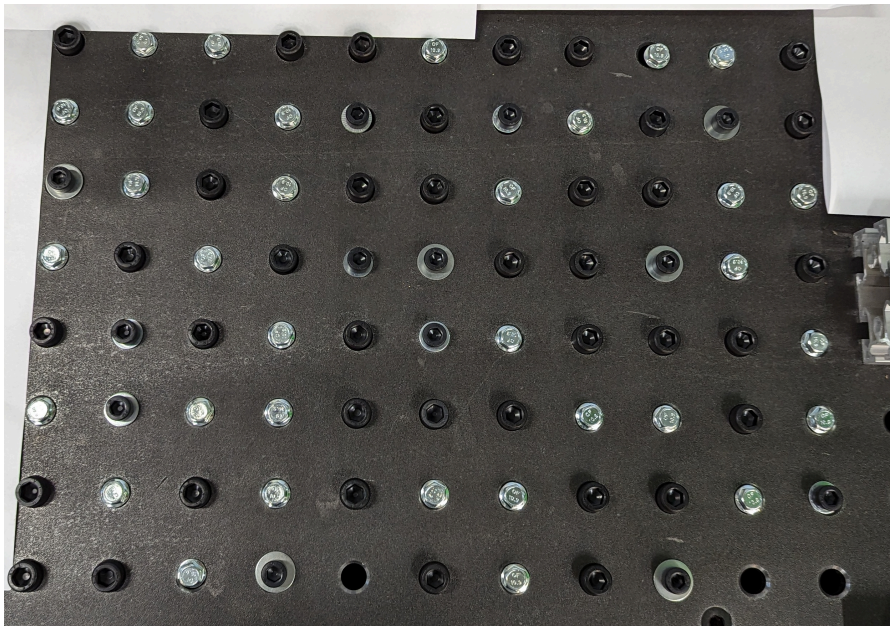


Figure 3.3: Environment of bolt position for testing

3.3.2 Hardware and Software

The following is a list of hardware used in the test

- Kria KV260 Vision AI Starter kit as the SoC FPGA that is used for the implementation and evaluation.
- Camera USB with the type Logitech Brio 4K. The camera is used to take sequential image data in real-time testing.
- IMU sensor: PMODNav based on LSM9DS1. The IMU sensor is used to acquire accelerometer and gyroscope data.
- Collaborative Robot UR5e from Universal Robots. The collaborative robot is utilized to acquire the reference location for the testing and is used to hold the camera and IMU sensor during the test.

The following is a list of software used in the test

- Ubuntu Linux 22.04 for Kria KV260.
- PYNQ Platform.
- Vitis AI v2.5

3.4 Assessing reliability and validity of the data collected

Table 3.1 shows all of the data collected, unit, and source of the data.

Table 3.1: The data collected

The data collected	Unit	Source of data
Position references	mm	collaborative robot UR5e
Orientation references	rad	collaborative robot UR5e
3-axis accelerometers	m/s^2	PmodNav (LSM9DS1)
3-axis gyroscope	rad/s	PmodNav (LSM9DS1)
RGB images	size (800x600)	USB camera Logitech BRIO 4K

3.4.1 Validity of method

In this study, we used various methods to ensure the validity of our data acquisition from multiple sources: the collaborative robot, the IMU sensor, and the camera. The collaborative robot data was collected using TCP

communication and the officially provided Real-Time Data Exchange (RTDE) Python client library by the robot's manufacturer, Universal Robots [29]. To ensure the validity of this acquisition method, we implemented a handshake communication procedure prior to data transfer. This process confirmed a successful connection between the devices, thereby validating the data's integrity. For the IMU sensor, we utilized the SPI protocol to communicate with the FPGA (Field Programmable Gate Array). Since the sensor's output data has already been digitized and factory calibrated [30], its accuracy is assured. We further validated the acquisition method by reading the IMU sensor's WHOAMI register and cross-referencing it with the value specified in the datasheet. This check validated the sensor's identity and confirmed successful communication, supporting the method's validity. Lastly, the camera data was collected via a USB protocol communicating with the FPGA using the OpenCV Python library. We validated the camera's communication status by checking the USB port before the data acquisition. This preliminary check ensured a smooth and error-free data transfer from the camera. In summary, each data acquisition method was meticulously validated, ensuring the robustness and reliability of the overall data acquisition process.

3.4.2 Reliability of method

The data acquisition process utilized in this study consists of several methods, each designed for specific tasks and all confirmed to be highly reliable. Firstly, for the collaborative robot, we employed the RTDE library in Python [29], coupled with a TCP connection. This approach is not arbitrary but comes directly recommended by the robot's manufacturer. Numerous tests have been conducted to validate this method, where the acquired data was consistently found to correspond with the robot's official display readings. This regular alignment confirms the reliability of this data acquisition method. For the IMU sensor data, the SPI communication protocol was applied, conforming to the guidelines outlined in the official datasheet. To maintain the data's consistency and accuracy, we instituted a routine check of the WHOAMI register every time the sensor was accessed. This step guarantees that the sensor functions correctly and the data acquired is reliable. While an alternative protocol, the I2C, is available to access IMU data, it was not used due to its inherent limitations. The I2C operates at a slower clock speed and does not support full-duplex communication, making it less efficient than the chosen SPI method. Finally, in regard to camera data acquisition, we adopted the widely used and accepted OpenCV Python library. This library is known for its reliability and

is a common choice in the field of camera data collection. In summary, each data acquisition method used in this study - for the collaborative robot, the IMU sensor, and the camera - has been selected based on its proven reliability and efficiency, thereby ensuring the credibility of the results obtained.

3.4.3 Data validity

The validity of the data acquired in this study was confirmed through a careful series of checks and comparisons, ensuring that the data was accurate and representative of actual conditions. Starting with the collaborative robot, the collected position and orientation data were validated through direct comparison with actual data from the official HMI (Human-Machine Interface) monitor. This procedure entailed gathering data using the Python library and then cross-referencing it with the data displayed on the HMI monitor. This comparison assured the accuracy of the collected data, thus establishing its validity. With regard to the IMU (Inertial Measurement Unit), the data output was evaluated prior to the data acquisition process. For the accelerometer, a gravity reference check was conducted. This process ensures that the accelerometer readings align with the gravitational force, which is a known constant. Similarly, for the gyroscope, a movement reference check was performed, assessing whether the output data accurately reflected physical movements. These reference checks aimed to confirm that the digital output provided sensible and rational data, thereby verifying its validity. Lastly, the camera data's validity was confirmed through real-time output video analysis from a Jupyter Notebook. This validation process included checking the output images against specific parameters, such as colour, frame size, and image results. By ensuring the output images met these criteria, we confirmed the images accurately represented their subjects, thus validating the camera data. In summary, each piece of data gathered in this study, from the collaborative robot, the IMU sensor, and the camera, underwent rigorous validation procedures. This comprehensive approach ensured the data was both accurate and reliable, reinforcing the overall validity of the research.

3.4.4 Reliability of data

The reliability of the data shown in Table 3.1 was established using a careful and systematic approach. Each data source was initially tested more than three times using the same positions and setup. This repetition ensured that the data produced was consistent and comparable across each trial, which is crucial

before integrating it into the full data acquisition process. Furthermore, during each data acquisition, a safeguard was put in place to compare the collected data against a sample dataset. This step was crucial in spotting any potential errors in real-time and ensuring the accuracy of the data. By following this procedure, we could promptly identify and correct any inconsistencies, reinforcing the dependability of the data. In summary, the comprehensive validation approach, including the pre-integration tests and ongoing checks during data acquisition, helped guarantee the data's reliability. This rigorous method ensured that the data was both accurate and consistent, providing a solid foundation for the entire data acquisition process.

3.5 Planned Data Analysis

The output data planned from this research is shown in Table 3.2. The data analysis will be conducted for two different types of processing that will be used.

Table 3.2: The data output from processing

Input data	**Type of processing**	**Output data**
3-axis accelerometer, 3-axis gyroscope, images, Bolt reference data	localization process	6-DoF pose estimation (Position and Orientation)
Images	Bolt detection process	Type of bolt
3-axis accelerometer, 3-axis gyroscope, images, Bolt reference data	timing measurement	execution time

3.5.1 Data Analysis Technique

In this study, we employed various data analysis techniques to evaluate different aspects of the research.

The localization process involved using the Root Mean Square Error (RMSE) technique for assessing position and translation errors. The RMSE technique is a standard method for evaluating position and rotation data in visual-inertial pose estimation, as utilized in Liu et al.'s research [31]. We applied this technique to the 6-DoF output data, comparing it against the ground truth. For this study, the ground truth was defined as the position

and orientation of the collaborative robot. The RMSE value thus indicated the integrated error across the three axes.

The bolt detection performance was assessed using the Mean Average Precision (mAP) analysis technique. This method is widely used for evaluating object detection algorithms, as demonstrated in the YOLOv3 paper [11]. The mAP score provides an average precision across all object classes. This analysis used test data, which was drawn from a dataset consisting of 50 randomly selected poses with assigned labels.

Finally, execution time was evaluated for each process, including bolt detection and the update and prediction processes of the ESEKF. The execution times were aggregated to determine the required time for all processes to output bolt detection and localization results. The evaluation of execution time was benchmarked against the research targets set at the study's outset.

Chapter 4

Algorithm Implementation

4.1 Overview System

The entire system's functionality, including inputs, outputs, and processing blocks, is illustrated in the system block diagram (Figure 4.1).

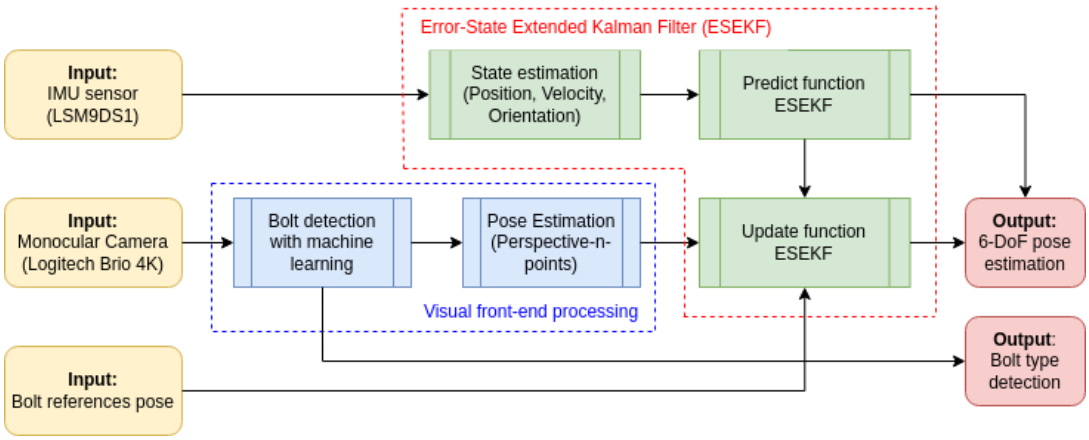


Figure 4.1: System Block Diagram

The system overview of bolt detection and visual-inertial localization for the tightening tool consists of three input elements, two output elements, and two major processing blocks. The three input elements are an IMU sensor, a monocular camera, and a bolt reference pose. These inputs provide the raw data, which is then processed to yield meaningful outcomes. The two output elements, 6-DoF pose estimation and bolt-type detection, provide insightful information that is crucial for the functioning of the system. The two major

processing blocks are the visual front-end processing and the ESEKF. The visual front-end processing incorporates a machine-learning algorithm that enables bolt detection and pose estimation using perspective-n-points. This process allows the system to identify and locate bolts with high precision. The ESEKF processing integrates predict and update functions, as well as state estimation from the IMU sensor. These processes together generate estimations of position, velocity, and orientation, all of which are vital to the system's operations.

Given that this system is designed for a tool-tightening application, the bolt reference pose is essential and is obtained during the actual tool-tightening process. The position and orientation of the tightening tool at the moment of tightening define this reference pose. In our study, we incorporated this bolt reference pose into the ESEKF process as an updated state. It serves as a real-world reference point to ensure the accuracy of our simulations. During testing, we made the assumption that this bolt reference pose would be updated at arbitrary intervals. Specific to this study, we found this particularly useful when dealing with image sequences that suffer from long delays in data acquisition, helping to maintain system reliability even under less-than-ideal conditions.

The system under study is integrated into a SoC FPGA, specifically the Kria KV260 board from Xilinx. This FPGA board is composed of a Processing System (PS) ARM processor and Programmable Logic (PL) FPGA fabric. The ARM processor runs on an embedded Linux system, which provides the advantage of easily managing network communications and other interfaces thanks to its Linux compatibility. Furthermore, it operates under the PYNQ platform that binds the Python environment, allowing for coordinated operation between the PS and PL. Python brings to the table a host of benefits, primarily in the form of an extensive library base that includes OpenCV and numpy, among others. Moreover, the PYNQ platform is also compatible with the DPU IP core, making it a favourable choice for machine learning implementation in this research. The DPU IP core is instrumental in operationalizing the machine learning model in a manner that is both efficient and fast. The efficiency and speed are attributed to the fact that the operations are conducted in the PL FPGA fabric. For the actual FPGA implementation, the DPU IP core utilizes Vitis AI to compile the machine learning model.

4.2 Bolt Detection using Machine Learning

4.2.1 Building the YOLOv3-Tiny-3L Model

In this research, we simplified bolt detection only with two types of bolts. Figure 4.2 shows the bolt image for bolt detection. The black bolt is identified as ”bolt1”, and the silver bolt is identified as ”bolt2”.

Figure 4.2: Type of bolt

For bolt detection, this study employed the YOLOv3-Tiny-3L model implemented in Darknet [32]. The selection of the YOLOv3 tiny model was driven by its known compatibility with object detection tasks in embedded systems. Additionally, the DPU IP core and Vitis AI have demonstrated effective compatibility with YOLOv3, as evidenced by existing examples showcasing model compilation.

In the context of this research, the training phase involved using 200 distinct images, which were labelled and extracted separately from the dataset intended for testing. The model was trained over 8000 iterations to optimize the accuracy of object detection. The end product of this training phase is a Darknet-trained model, accompanied by a configuration file and weights file. The architecture of YOLOv3-Tiny-3L is shown in Table 4.1

Table 4.1: YOLOv3-Tiny_3L Architecture Configuration

Layer	**Layer Type (Activation Function)**	**Filters**	**Size/ Stride**	**Input**	**Output**	**Biases/ Weights**
0	Convolutional (Leaky)	16	3x3/1	416x416x3	416x416x16	16 / 3x16x3x3
1	Maxpool		2x2/2	416x416x16	208x208x16	
2	Convolutional (Leaky)	32	3x3/1	208x208x16	208x208x32	32 / 16x32x3x3
3	Maxpool		2x2/2	208x208x32	104x104x32	
4	Convolutional (Leaky)	64	3x3/1	104x104x32	104x104x64	64 / 32x64x3x3
5	Maxpool		2x2/2	104x104x64	52x52x64	
6	Convolutional (Leaky)	128	3x3/1	52x52x64	52x52x128	128 / 64x128x3x3
7	Maxpool		2x2/2	52x52x128	26x26x128	
8	Convolutional (Leaky)	256	3x3/1	26x26x128	26x26x256	256 / 128x256x3x3
9	Maxpool		2x2/2	26x26x256	13x13x256	
10	Convolutional (Leaky)	512	3x3/1	13x13x256	13x13x512	512 / 256x512x3x3
11	Maxpool		2x2/1	13x13x512	13x13x512	
12	Convolutional (Leaky)	1024	3x3/1	13x13x512	13x13x1024	1024 / 512x1024x3x3
13	Convolutional (Leaky)	256	1x1/1	13x13x1024	13x13x256	256 / 1024x256x1x1
14	Convolutional (Leaky)	512	3x3/1	13x13x256	13x13x512	512 / 256x512x3x3
15	Convolutional (Linear)	27	1x1/1	13x13x512	13x13x27	27 / 512x27x1x1
16	YOLO					
17	Route					
18	Convolutional (Leaky)	128	1x1/1	13x13x256	13x13x128	128 / 256x128x1x1
19	Upsample		2x2/1	13x13x128	26x26x128	
20	Route					

Continued on next page

Table 4.1 – *Continued from previous page*

Layer	Layer Type (Activation Function)	Filters	Size/ Stride	Input	Output	Biases/ Weights
21	Convolutional (Leaky)	256	3x3/1	26x26x384	26x26x256	256 / 384x256x3x3
22	Convolutional (Linear)	27	1x1/1	26x26x256	26x26x27	27 / 256x27x1x1
23	YOLO					
24	Route					
25	Convolutional (Leaky)	128	1x1/1	26x26x256	26x26x128	128 / 256x128x1x1
26	Upsample		2x2/1	26x26x128	52x52x128	
27	Route					
28	Convolutional (Leaky)	128	3x3/1	52x52x128	52x52x128	128 / 256x128x3x3
29	Convolutional (Linear)	27	1x1/1	52x52x128	52x52x27	27 / 128x27x1x1
30	YOLO					

The YOLOv3-Tiny-3L uses a 3-layer output YOLO to achieve more accurate object detection. The architecture is inspired by [14] and [32]. The basic YOLOv3-Tiny has a 2-layer output with less hidden layer [11], whereas the 3-layer version adds the hidden layer to generate the third yolo output layer. It adds more trained variables and computation time than the default YOLOv3-tiny but is still smaller than YOLOv3.

4.2.2 Vitis AI Quantize and Compile Model

This research used the Vitis AI version 2.5 to follow the compatible compiler for DPU in the PYNQ platform. The YOLOv3-Tiny-3L machine-learning model from the darknet platform consists of configuration (.cfg) and weight parameters (.weights) files. Since the Vitis AI does not support the input-trained model from the darknet, the trained model needs to be converted into the supported file. One of the supported model files is TensorFlow. Figure 4.3 shows the steps for processing the darknet-trained model into a compiled model that is ready to use for integrating with the DPU IP core. the blue

colour represents the process of conversion outside the Vitis AI, and the red colour represents the process inside the Vitis AI.

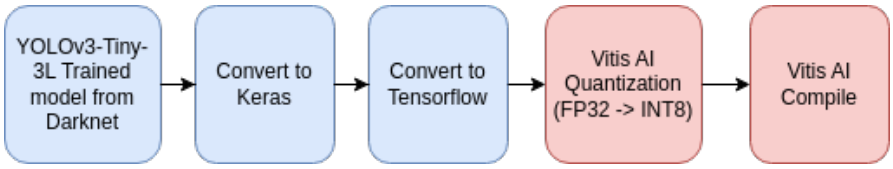


Figure 4.3: Process of quantization and compilation

The conversion model from darknet into TensorFlow consists of two steps conversion with intermediate conversion into Keras model. The conversion used the public repository code from David8862 [33]. The model for input into the Vitis AI is the TensorFlow model with an extension protocol buffer (.pb).

The quantization process in Vitis AI converts the trained model from a floating point 32-bit to an integer 8-bit. Normally, the quantization process will reduce the accuracy of the trained model. To maintain the accuracy of the trained model, during the quantization process, the trained model will be calibrated with train data images. After the quantization process, the accuracy of the prediction will often have a loss of less than 1% [34]. The quantization process will produce a file format similar to TensorFlow, which is a protocol buffer (.pb) with a quantized value.

The compilation process in Vitis AI will convert the quantized model into the xmodel file. It uses the additional input DPU architecture file with extension json. the DPU architecture is adjusted with the used architecture in Table 4.2. Inside the compilation process, there are multiple optimizations, such as batch normalization operations.

4.2.3 Build DPU IP core

The architecture of the DPU IP core can be adjusted based on the requirement. Table 4.2 presents the detailed architecture of the DPU for this research.

Table 4.2: DPU Architecture Configuration

Parameters	**Value**
Number of core	1
Architecture size	B3136
UltraRAM	Enable
DRAM	Disable
RAM Usage	Low
Channel Augmentation	Enable
ALU parallel	Default (1)
CONV RELU Type Configuration	RELU, LEAKYRELU, RELU6
ALU RELU Type Configuration	RELU, RELU6
DSP48 Usage Configuration	High
Power Configuration	Low Power Disable
DEVICE Configuration	MPSOC

The DPU architecture configuration is prepared for integration into the overall programmable logic design. The configuration parameters are chosen based on iteration to adjust the utilization fit with the amount of available programmable logic resources. The architecture size indicates the number of MAC units per DPU clock cycle. The machine learning parameters, such as weights, bias, and intermediate feature maps, are kept on-chip by enabling UltraRAM and disabling Dynamic Random Access Memory (DRAM) to reduce power consumption. RAM usage is tried to keep low to reduce the utilization of block RAM, which impacts reducing the flexibility in handling the intermediate data inside the core. The channel augmentation is enabled to improve the efficiency of the core when the number of input channels is lower than the available channel, but it will cost an additional Look-Up-Tables (LUTs).

The building process of the programmable logic in this research used Vitis flow. There are two types of flow that are usually used for building programmable logic. First is Vivado flow, and second is Vitis Flow. Vivado flow uses fixed-platform design, which the design has been completed in the Vivado Design Suite software. All of the IP blocks have been integrated, and the synthesis and implementation have been done. The output from the fixed-platform design is a Xilinx Shell Archive (.xsa) or bitstream file. The file is ready to use for the next development process using Vitis IDE for C/C++ programming or the PYNQ platform for Python programming.

On the other hand, the Vitis flow uses the extensible platform. this platform

allows the addition of programmable components such as IP core to produce a complete embedded system design. The design flow utilizes the hardware container (.xsa) file that is produced by Vivado Design Suite. The extensible $.xsa$ file provides the base hardware design for the $V++$ linker to extend by updating a dynamic region of the design.

Overall the integration of the DPU IP core is shown in Figure 4.4

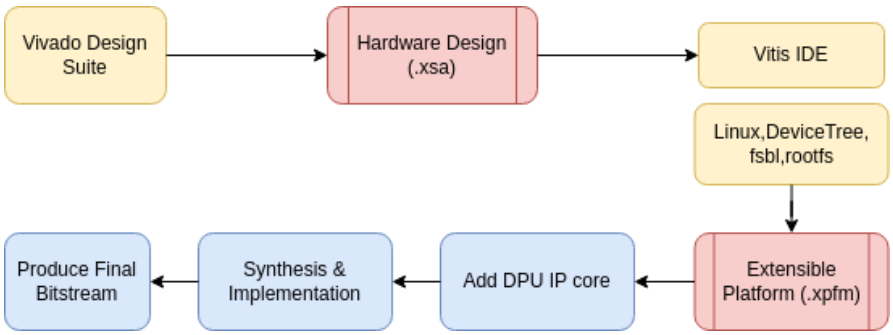


Figure 4.4: Process of DPU IP core integration

The Vivado Design Suite consolidates all of the IP cores for the design except the DPU IP core, such as the SPI, ESEKF, clock wizards, and so on. The hardware design (.xsa) is imported with a pre-synthesize configuration. The hardware design is loaded into Vitis IDE, and set up the configuration is set up as Linux without generating a boot. After the building platform in Vitis IDE, the extensible platform (.xpfm) file will be generated. The file will be used for Vitis flow that adds the DPU IP core, does synthesis and implementation and also produces the final bitstream. The Vitis flow is represented as a blue colour block in Figure 4.4.

4.3 Visual Localization

4.3.1 Assignment 3D Location for The Bolts

The pairs of a 3D location and a 2D point in the image become important inputs for visual localization to determine pose estimation with the perspective-n-points algorithm. The 2D points can easily be acquired from the image, whereas the 3D location for the bolts becomes another challenge to be acquired. For a system that has a base reference position, for this research is a Cobot base reference location; the 3D location of the bolts uses the Cobot

base reference, so the pose estimation output has a similar base reference. For implementing the assignment 3D location algorithm for the bolts, the following are several assumptions that are made for this research

- The location for all the bolts in the first time of operation has been known.
- The camera data (image) are supplied continuously (more than 5 FPS), or the 3D position of the bolts in the next image should be reassigned.
- The dimension or the shape of the bolt placement area has been known before. This research used a flat test rig with holes for bolt placement that has a distance of 50 mm on the x-axis and y-axis.

The assignment 3D location for the bolts follows the rules below

- The 3D location of the bolt uses the previous sequence of the image to determine the current image
- the assignment 3D location from detected bolts in the new image, the 2D points in the new image will be compared to 2D points in the previous image. If the Euclidean distance between 2D points is below a certain threshold, the 3D location of the bolt that correlated to the certain 2D point will be assigned to the nearest 2D point in the new image.
- unassigned bolt points in the new image will be determined with the knowledge of test rig placement dimension. the new 3D location will be estimated based on the 2D point location related to the nearest 2D points. Because the test rig platform has a fixed distance of 50 mm, the estimation will add or subtract 50 mm on the axis x or y, which is decided based on the new 2D points in the image.

4.3.2 Camera Pose Estimation

The camera pose estimation within our research will be implemented using the PnP algorithm. This particular algorithm was chosen due to its relevance and compatibility with the key point's bolt positions. The positions of the bolts give a unique blend of 3D space data and 2D image data. Basically, the PnP algorithm is meant to figure out the location and angle of a camera in a 3D space, given sets of matching 3D and 2D points. So, it works out where the camera is and how it is tilted in the real world by using the 3D coordinates of certain points and their matching 2D images. It makes the algorithm

particularly handy when we need to work out the camera's pose using the data we have gathered about the bolts. In the context of pose estimation algorithms, the PnP is commonly adopted for systems that leverage key points equipped with a 3D location reference and a corresponding set of 2D points in an image [26], [27], [28]. This makes it a suitable choice for our study, considering our key point data consists of the bolt's 3D location and its 2D correlation within the image.

We have sourced the PnP function from the OpenCV python library, a reliable resource for image processing and computer vision applications [21]. It provides a robust implementation of the PnP algorithm, aiding in simplifying the process of camera pose estimation. To visualize this process, refer to Figure 4.5, which elucidates the application of the perspective-n-points calculation in the context of our study. It provides a step-by-step representation of how we obtain the camera pose estimation through the PnP algorithm, illustrating the effective transformation of raw 3D and 2D data points into actionable camera pose information.

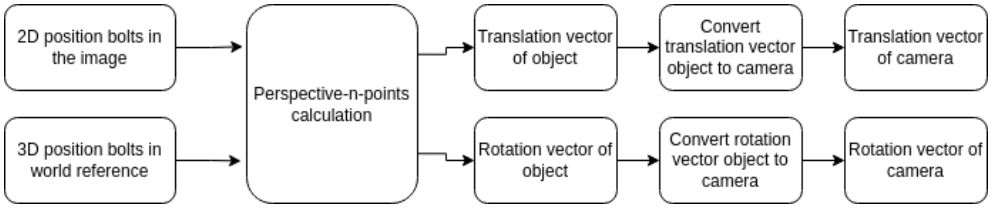


Figure 4.5: Perspective-n-Points calculation to get camera pose estimation

The input of perspective-n-points calculation is 2D position bolts in the image and 3D position bolts in the world reference. The 2D position will be obtained after the bolt detection process. The middle position of the bolt detected will become the 2D input position for this calculation. After the bolts have been detected and got the 2D position, the assignment location for the bolt will provide the 3D location of each bolt. The location of the bolt uses a reference from the collaborative robot.

To run the calculation, the input points should be six or more. The calculation will result in the translation and rotation vector of the object. The output should be converted into a camera perspective. To convert the perspective object to the camera, additional calculation is needed. the translation and rotation vector of the object needs to be multiplied by the negative transposed rotation matrix from the object.

4.4 Error State Extended Kalman Filter

The ESEKF algorithm that is explained in Section 2.5 is implemented in FPGA logic fabric with IP core package for most of the mathematical expression. The ESEKF implementation is inspired by the Python implementation from Sanchez [35]. The ESEKF IP core became one of the major IP cores in the implementation regarding utilization usage. The design process of the ESEKF IP core used iteration to evaluate the utilization and adjust the implementation architecture to reduce the utilization. The final design block of the ESEKF IP core is shown in Figure 4.6.

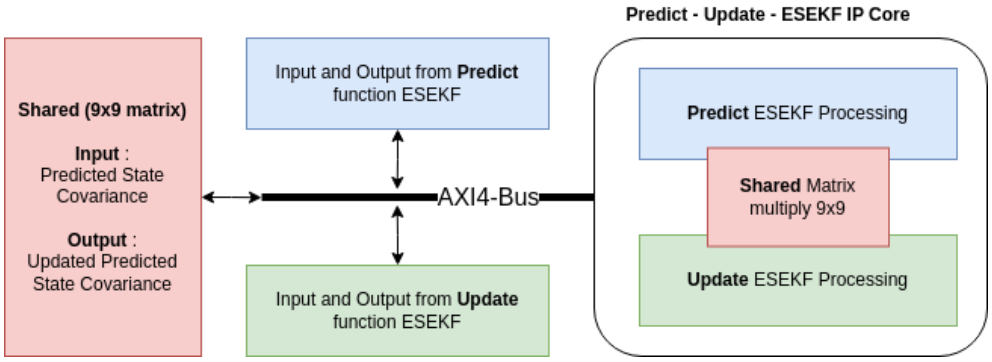


Figure 4.6: ESEKF IP core design

The predict and update function has a bunch of input and output data. The IP core is designed to accommodate communication by incorporating the AXI4 bus. AXI4 is a high-performance memory-mapped data and address interface that is easily integrated within the Xilinx development environment. It has the capability to burst access to memory-mapped devices. It is suitable for high-bandwidth and low-latency IP core design and capable of providing high-frequency operation [36]. It is important for the ESEKF IP core because the IP core will handle 9x9 matrix transfer for input and output, and also the other input and output variables.

The design of the ESEKF IP core consists of shared resources on the input-output interface and the computation processing. The input-output for both predict and update functions needs to accommodate predicted state covariance with the size of a 9x9 matrix. Moreover, the computation for both functions requires matrix multiplication for a 9x9 matrix. Therefore, the identical functionalities from both functions are shared because the operation of the

IP core for both functions is not in parallel. The control signal is provided to handle the switching between the functions.

The IP core development utilizes the HLS process. It uses the Simulink Matlab HDL coder. The functionalities system is built with Simulink block designs that support the HDL coder process. Most of the processes are the math and logic manipulation functions. In the block design, the targeted architecture of the RTL design is defined, such as the serial or parallel process, the pipeline, the usage of Digital Signal Processing (DSP) and so on. The functionalities of the system are simulated and verified with Simulink. After all of the operation processes have been verified, the conversion of the Simulink block into RTL is started. The conversion process can be set to produce the IP core package directly by defining the targeted device of FPGA. The generated IP core is then checked with Vivado Design Suite software to observe the utilization consumption. If the utilization is above the target, the iteration process is done by defining architecture in the Simulink block to achieve the targeted resource utilization. In this research, the utilization of the logic component is a main concern to optimize because of the limited logic resource of the Kria KV260 FPGA. Whereas the timing is not the problem, most of the architecture logic in this research utilized serial operation instead of parallel operation.

The variable type for the ESEKF IP core mostly uses fixed point 16-bit with fraction 14-bit. The reason is to reduce and simplify the logic design and save the logic resources. However, the design still uses other variable types, such as fixed point 32-bit, for some parts of the operation to keep the precision of the operation. But if the variable can be represented with a fixed point 16-bit without significant loss, the variable will be pushed to use as minimal resources.

The ESEKF IP core has 2 control variables and 1 status output variable. The two control variables are the control switch functionality and the start operation signal. the control switch functionality controls the IP-core for the predict or update function. The start operation signal needs to be injected with signal one to indicate that the operation inside the core should be started. the one status output variable is the status that indicates all of the operations in the core have been done.

4.4.1 Predict Function ESEKF

All of the mathematical operations in the predict function ESEKF are mapped into RTL logic except the conversion from gyroscope data times delta time into

the quaternion format. The following is the input and output variable from the predict function with its variable type.

Table 4.3: Input and output variables for predict function ESEKF

Variables	Direction	Type	Size
Accelerometer data	Input	Fixed 32-bit (frac 26-bit)	3
Delta time	Input	Fixed 32-bit (frac 26-bit)	1
Variance accelerometer	Input	Fixed 16-bit (frac 14-bit)	1
Variance gyroscope	Input	Fixed 16-bit (frac 14-bit)	1
Previous quaternion	Input	Fixed 32-bit (frac 26-bit)	4
Current quaternion	Input	Fixed 32-bit (frac 26-bit)	4
Previous state covariance	Input	Fixed 16-bit (frac 14-bit)	81
Predicted position	Output	Fixed 32-bit (frac 21-bit)	3
Predicted velocity	Output	Fixed 32-bit (frac 21-bit)	3
Predicted orientation	Output	Fixed 32-bit (frac 26-bit)	4
Predicted state covariance	Output	Fixed 16-bit (frac 14-bit)	81

4.4.2 Update Function ESEKF

The update function requires an inverse matrix operation for calculating the Kalman gain. Since there is no direct support of block operation in Matlab HDL coder for inverse matrix, the inverse matrix operation is conducted in the processor with Python numpy library. Another exception is the calculation of error state compensation for orientation, which is computed in Python code. The other mathematical expressions are mapped in RTL logic. The following is the input and output variable from the update function with its variable type.

Table 4.4: Input and output variables for update function ESEKF

Variables	Direction	Type	Size
Predicted position	Input	Fixed 32-bit (frac 21-bit)	3
Predicted velocity	Input	Fixed 32-bit (frac 21-bit)	3
Updated position	Input	Fixed 32-bit (frac 21-bit)	3
Inverse matrix	Input	Fixed 32-bit (frac 16-bit)	9
Predicted state covariance	Input	Fixed 16-bit (frac 14-bit)	81
Corrected position	Output	Fixed 32-bit (frac 21-bit)	3
Corrected velocity	Output	Fixed 32-bit (frac 21-bit)	3
error state orientation	Output	Fixed 32-bit (frac 26-bit)	3
Corrected state covariance	Output	Fixed 16-bit (frac 14-bit)	81

4.5 System Integration

The system integration integrates all of the IP cores that are used in this research, such as the ESEKF IP core, SPI IP core, and DPU IP core. The integration used the Vitis flow, which is explained in Section 4.2.3 and with Figure 4.4. The Vitis flow integration used the DPU PYNQ script [17] for automating the process. Vitis flow was chosen because it automatically integrates the DPU IP core into the platform. Then, it will automatically start the synthesis, implementation, and generation of a bitstream and the generation of overlay files. The result of integration is the full schematic of the SoC FPGA architecture, the FPGA implementation properties, and the overlay files for the PYNQ platform.

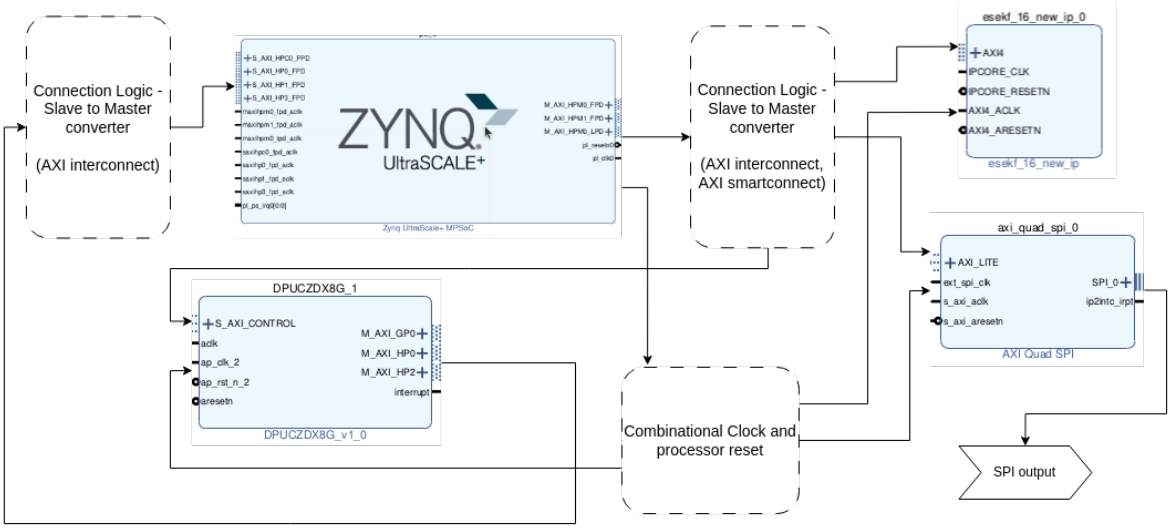


Figure 4.7: Simplify FPGA schematic

Figure 4.7 shows the simplification of the SoC schematic. The routing arrow in Figure 4.7 is just a high-level abstraction, whereas the full schematic can be accessed in Appendix A.1. All of the IP cores are connected to the ZYNQ ultrascale+ processor via AXI bus connection with AXI interconnect and AXI smartconnect. The IP cores use several clock configurations, such as ESEKF and SPI uses 100 Mhz, and DPU uses 300 Mhz and 600 Mhz clock supply. The clock sources are created by the clock wizard IP core with a single supply clock output from the ZYNQ processor.

Chapter 5

Results

5.1 Major results

5.1.1 Bolt Detection Result

The graph below illustrates the training loss of bolt detection on YOLOv3-Tiny-3L.

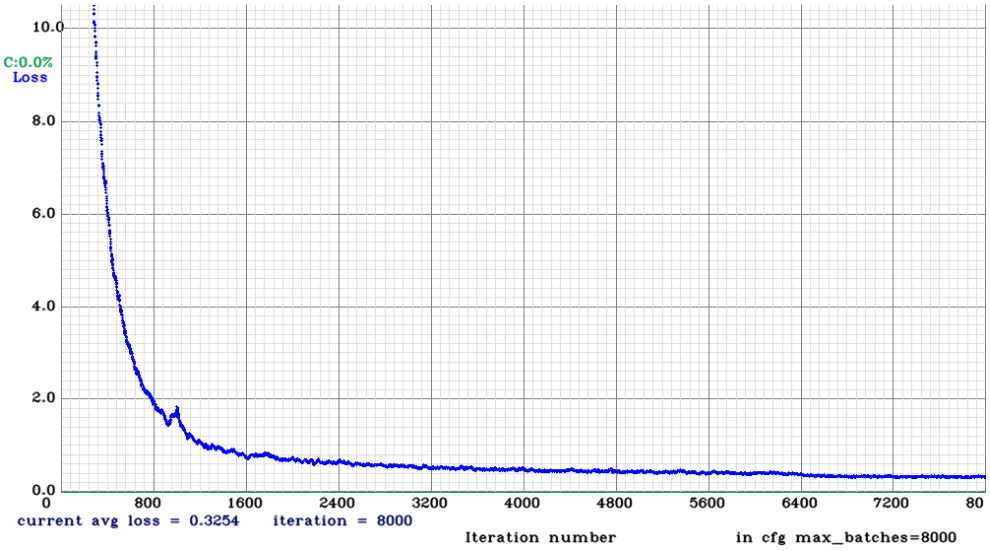
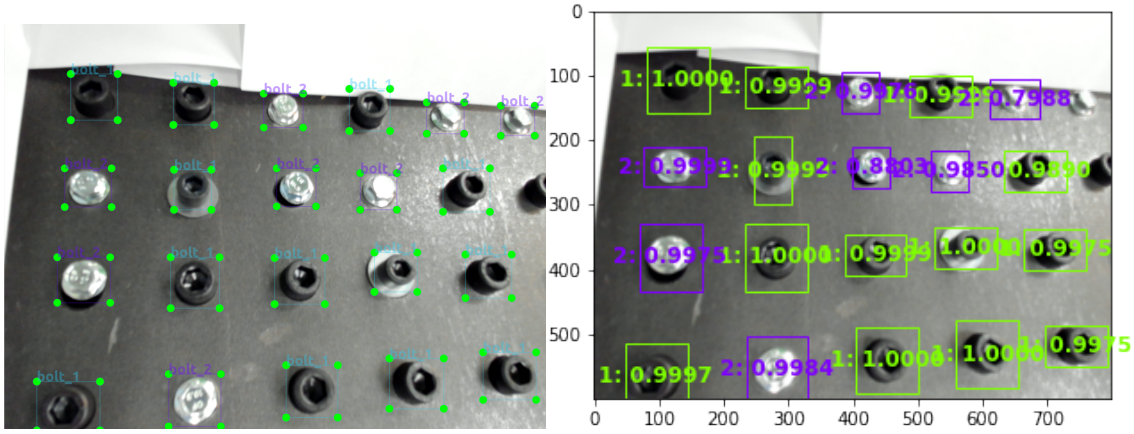


Figure 5.1: Training loss of bolt detection model on YOLOv3-Tiny-3L

Figure 5.1 shows that after 8000 iterations, the average loss of the model is 0.3254. After the training, the trained model was converted into an xmodel file, which is a compatible file for the DPU IP core. The conversion between

the trained darknet model into the xmodel file is described in Section 4.2.2. The xmodel file was used to test actual bolt detection in the FPGA. The pictures below present a comparison of bolt detection between the test dataset and the result of bolt detection.



(a) (b)

Figure 5.2: (a) Labeled dataset. (b) Bolt detection YOLOv3-Tiny-3L.

Figure 5.2a shows the manually labelled dataset for testing purposes on the trained model, and Figure 5.2b shows the bolt detection result in The FPGA. Most of the bolts have been detected correctly. However, the bounding boxes from Figure 5.2b do not fit fully as a manually labelled dataset, and there is a missing bolt from detection. Figure 5.2b is the sample result, Whereas the population of bolt detection are represented in Table 5.1.

Table 5.1: Accuracy of Bolt detection

Precision	**Classes**	**YOLOv3-Tiny-3L**
AP (%)	Bolt 1 (Black)	29.43
AP (%)	Bolt 2 (Silver)	36.71
mAP (%)		33.07

Table 5.1 provides the result of Average Precision (AP) from two classes and the mean Average Precision (mAP). The results in Table 5.1 were acquired with the Intersection over Union (IoU) threshold of 0.5. Overall, these results indicate that bolt detection with YOLOv3-Tiny-3L has successfully detected the bolt, whereas the detection accuracy is categorized as low.

5.1.2 Localization Result

The localization result consists of a six-dimensional degree of freedom pose estimation, which is a 3-axis position and a 3-axis orientation. All of the data for pose estimation has reference from the base location of the Cobot that is described in Section 2.4. Figure 5.3 compares the 3D plot of ground truth data and visual-inertial position estimation.

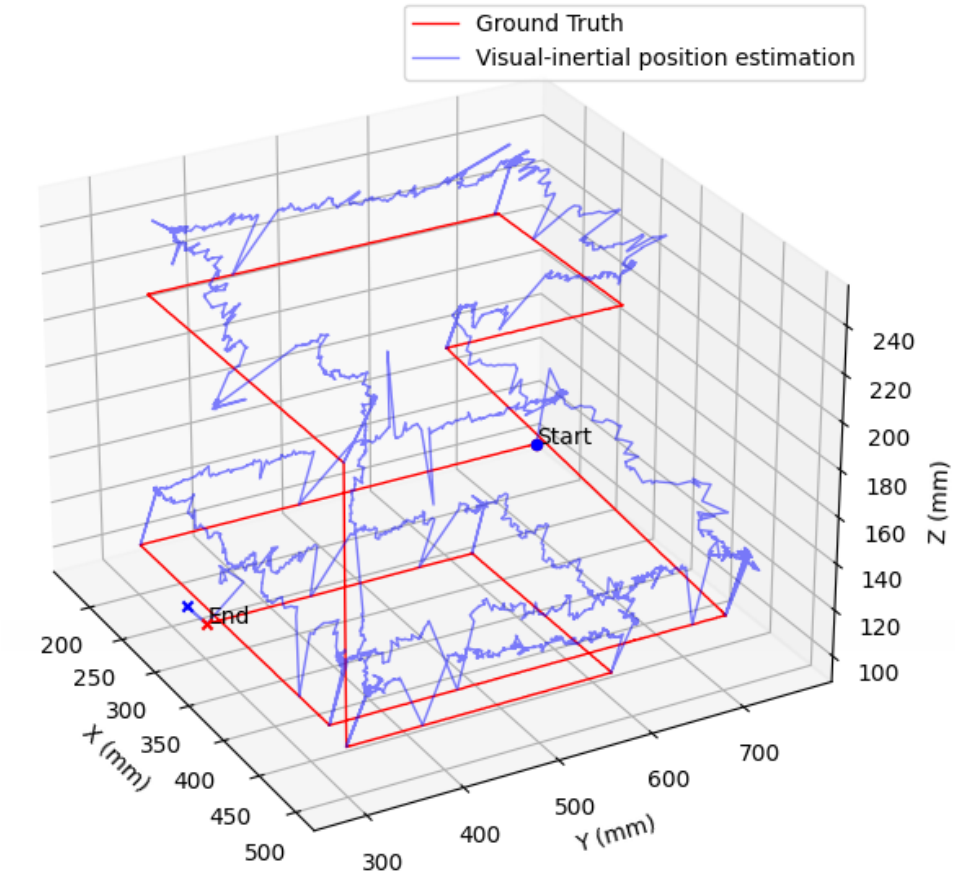


Figure 5.3: 3D tracking Visual-inertial position estimation

As shown in Figure 5.3, the visual-inertial position estimation overall follows the ground truth data movement. The test movement object starts and ends in the marked texts ”start” and ”end” in Figure 5.3. The comparison of position estimation from each axis is shown in the figures below.

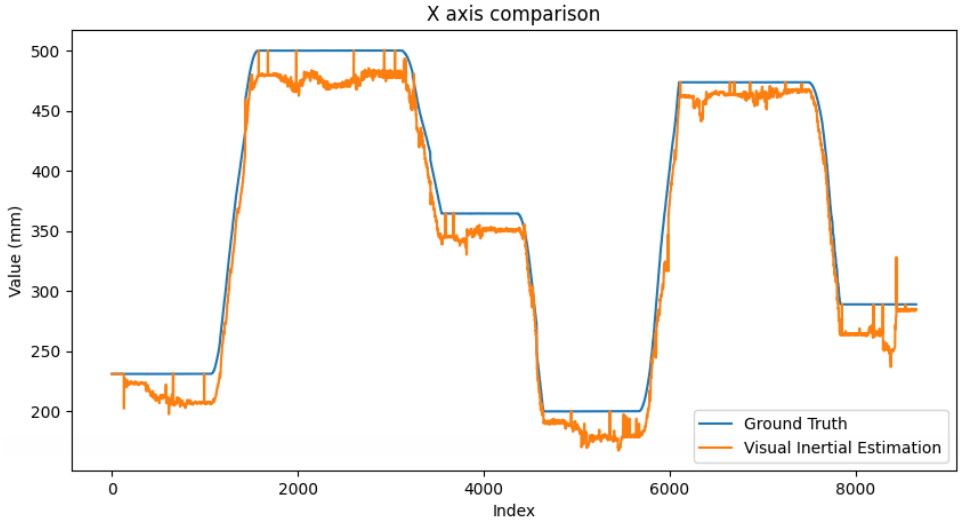


Figure 5.4: X axis comparison for Visual-inertial position estimation

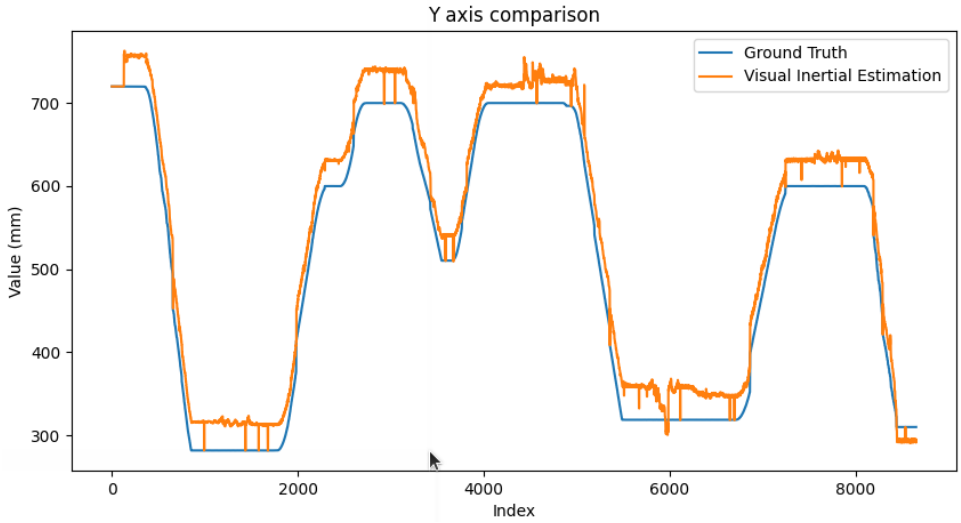


Figure 5.5: Y axis comparison for Visual-inertial position estimation

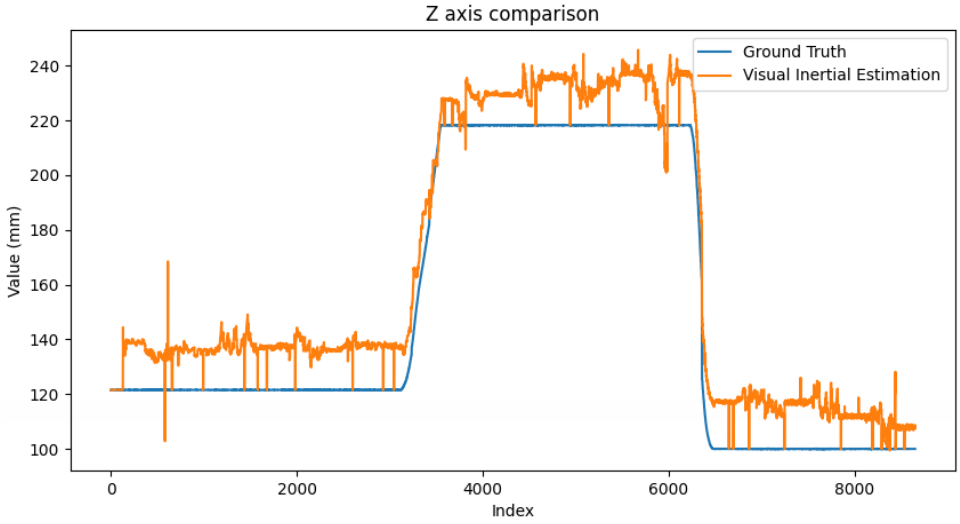


Figure 5.6: Z axis comparison for Visual-inertial position estimation

Figure 5.4, Figure 5.5 and Figure 5.6 show the difference between the position estimation in each axis with ground truth for all of the index dataset. To see the distribution of RMSE from the position, the histogram of RMSE position data is shown in Figure 5.7.

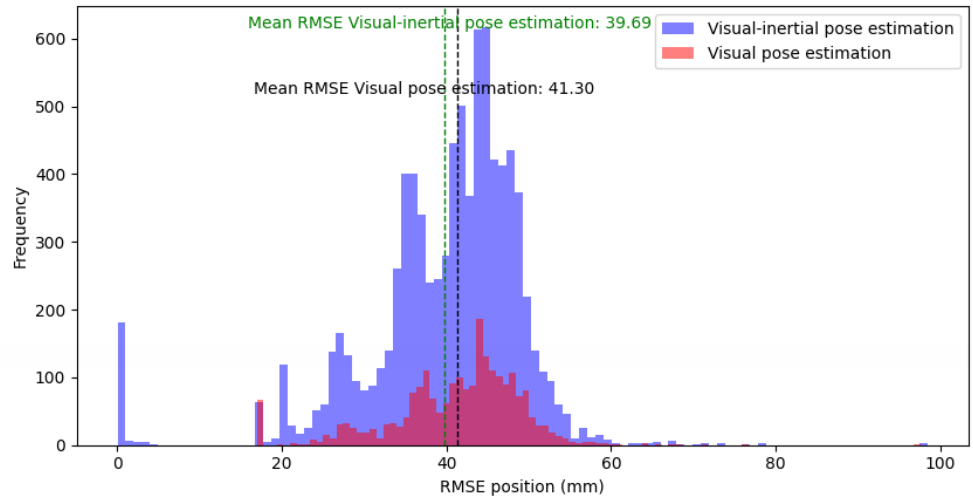


Figure 5.7: RMSE position histogram

Figure 5.7 provides the RMSE position histogram from visual-inertial pose estimation and visual pose estimation. The visual pose estimation is

obtained from the calculation of perspective-n-points in the image dataset, whereas the visual-inertial pose estimation is obtained from the data fusion between the visual pose estimation and the IMU dataset with the ESEKF algorithm. From the graph, it can be seen that the visual and visual-inertial have closely distributed data, where the mean value of RMSE position in visual-inertial is slightly lower than the mean value of RMSE position in visual pose estimation. Furthermore, the error orientations from both visual and visual-inertial pose estimation are shown in the figures below.

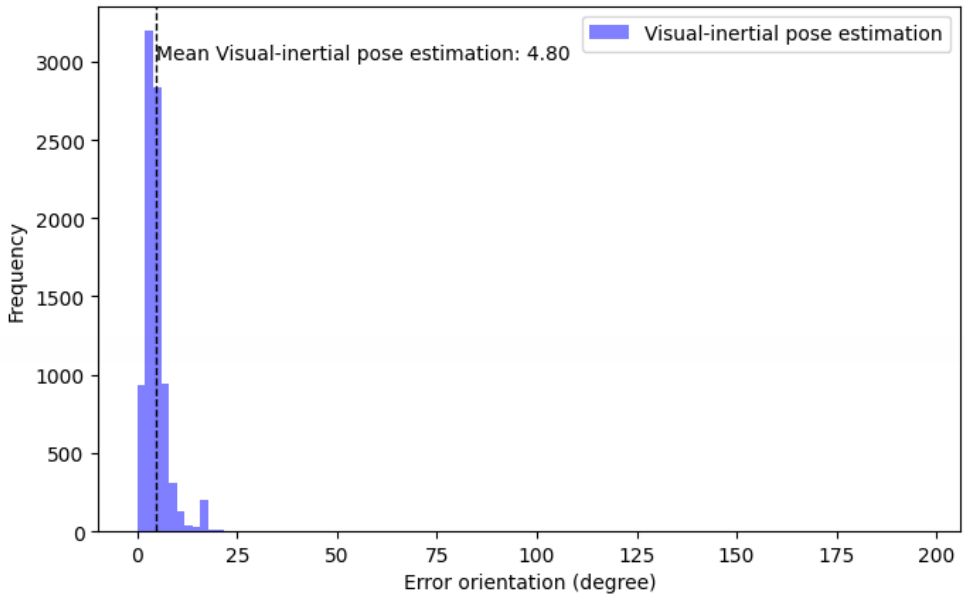


Figure 5.8: Error orientation histogram for Visual-Inertial estimation

Figure 5.8 presents the histogram of error orientation from visual-inertial pose estimation. The histogram shows mostly empty in the middle, with most of the data in the left part under the 25 degree because there are several outliers data near the value of 200. Because of the outliers, the distribution of the histogram becomes unrepresentative.

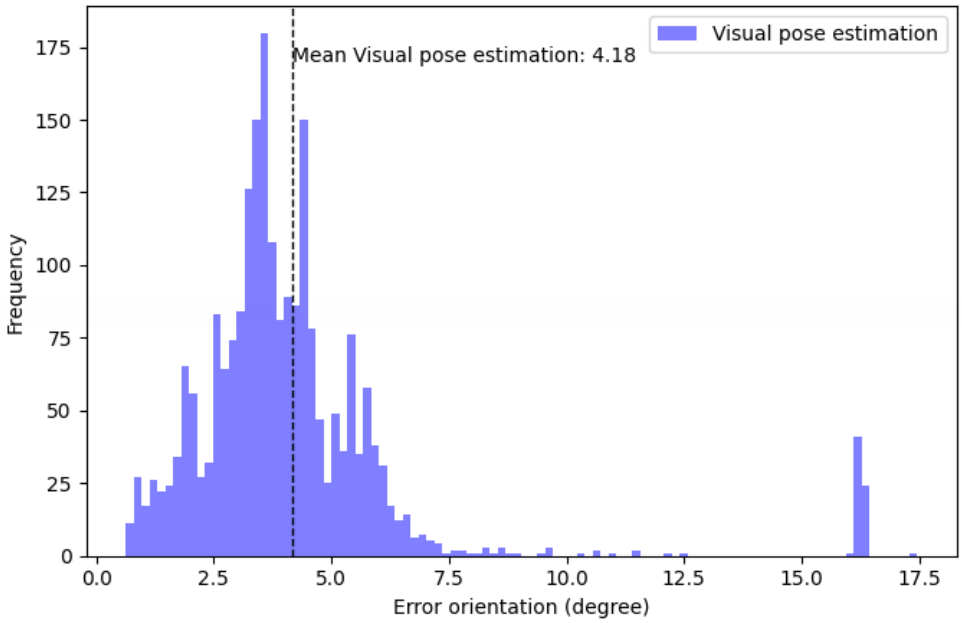


Figure 5.9: Error orientation histogram for Visual estimation

Figure 5.9 presents the histogram of error orientation from visual pose estimation. The histogram from Figure 5.9 is more representative than Figure 5.8 because the outliers are still below the value of 17 degrees. However, the mean value of error orientation in both configurations has a close value, whereas the visual pose estimation has a slightly better error with 4.18 degrees.

The overall localization parameters value distribution is shown in Table 5.2.

Table 5.2: Localization parameter from Visual-Inertial pose estimation

Parameters	**Mean**	**Standard Deviation**
RMSE Position (mm)	39.69	9.90
Absolute Error X-axis (mm)	18.30	8.71
Absolute Error Y-axis (mm)	30.98	8.61
Absolute Error Z-axis (mm)	14.39	4.66
Error Orientation (Degree)	4.80	5.39

Table 5.2 presents the results of localization parameters derived from a visual-inertial pose estimation system with mean and standard deviation from

position and orientation error. For the RMSE position, the data is distributed closely from the mean value that is indicated by the standard deviation value has a value significantly lower than the mean value. Whereas the error orientation has more distributed data with a standard deviation higher than the mean value.

5.1.3 Time Execution

Since the system is implemented in SoC FPGA, that means some of the processes are executed in FPGA logic fabric and DSP slices, and the others are executed in the processor. The information on time execution can provide insight into efficiency, bottlenecks, complexity and benchmarking from every process that is executed in the system. Table 5.3 provides summary information of time execution from major processes in the system.

Table 5.3: Time execution summary

Function	**Process**	**Time Execution**
DPU for YOLOv3-Tiny-3L (Without inference)	FPGA Logic & DSP Slices	25 ms
Full inference YOLOv3-Tiny-3L with **DPU**	FPGA Logic & DSP Slices & ARM Processor	85 ms
Full inference YOLOv3-Tiny-3L with **ZYNQ-ARM processor**	ARM Processor	1430 ms
ESEKF IP-core internal process (without AXI4 communication)	FPGA Logic & DSP Slices	176 μS
Predict function ESEKF (with AXI4 communication)	FPGA Logic & DSP Slices & ARM Processor	8.8 ms
Update function ESEKF (with AXI4 communication)	FPGA Logic & DSP Slices & ARM Processor	10 ms
Pre-pose estimation process (3D assignment location, etc)	ARM Processor	7 ms
Pose estimation process	ARM Processor	2.3 ms

As can be seen from Table 5.3, the execution of the YOLOv3-Tiny-3L model in the FPGA has a significant time reduction compared to processor

execution. The process inside the DPU IP core or only in FPGA logic fabric and DSP slices has an even lower time execution. A similar pattern also can be seen in the ESEKF IP core, where the execution inside the FPGA logic has significantly lower time execution than the full process with the processor that includes the AXI4 communication.

5.2 Minor result

5.2.1 FPGA Implementation Result

The FPGA implementation refers to the implementation of architecture that consists of the ZYNQ processor, the bus connection, and the other IP cores. The FPGA implementation is described in Section 4.5. The FPGA implementation results in several important information such as utilization, timing, and power estimation.

The table below shows the proportion of the utilization in the final implementation of logic fabric in the FPGA.

Table 5.4: FPGA Utilization

Utilization	DPU	ESEKF	Others	Total	Percentage
CLB LUTs (117120)	45103	25753	4035	74891	64%
CLB Registers (234240)	80293	59495	4687	144475	62%
CARRY8 (14640)	1018	568	2	1588	11%
F7 Muxes (58560)	2233	1648	3	3884	7%
Block RAM (144)	78	9.5	0	87.5	61%
UltraRAM (64)	40	0	0	40	63%
DSPs (1248)	566	329	0	895	72%

What is interesting about the data in Table 5.4 is that most of the utilization in the FPGA is utilized by the DPU and ESEKF IP core. The other IP cores, such as SPI, clock generation, bus interconnect, and so on, use a small portion of logic utilization. The Configurable Logic Block (CLB) LUT and CLB registers consume 64% and 62%, respectively, and internal RAM that consists of block RAM and UltraRAM consume 61% and 63%, respectively. The DSPs block is utilized more with 72% of utilization.

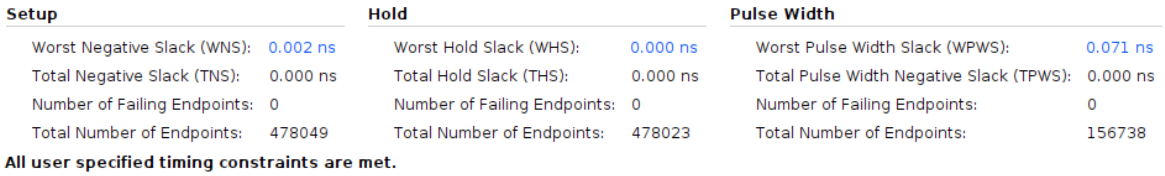


Figure 5.10: Timing summary

Figure 5.10 shows the timing summary after the implementation process. After the logic block placement and routing, there is no timing violation in setup and hold timing. All of the logic can be operated within the default timing constraints based on defined clocks in full schematic Appendix A.1.

The power analysis uses the default assumption operation of logic without a defined activity file for the logic blocks. The result of power analysis is shown in Figure 5.11.

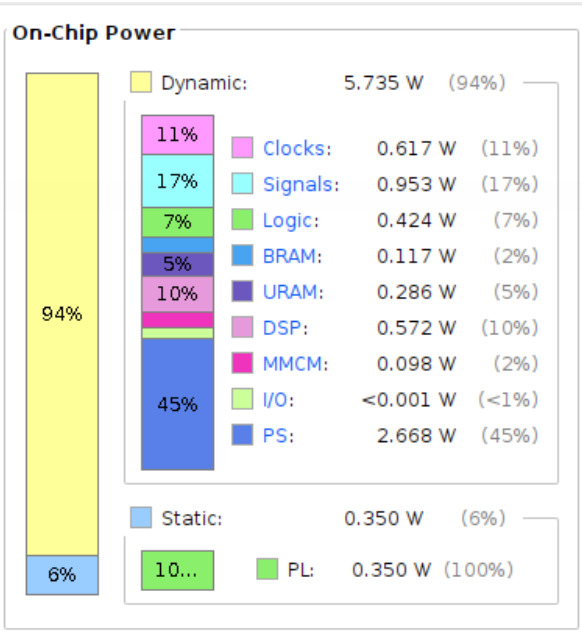


Figure 5.11: Power estimation report

As shown in Figure 5.11 about the power estimation, the dynamic consumes 5.735 W or 94% of on-chip power, and static consumes 0.350 W or 6% of on-chip power.

Furthermore, the trained model of YOLOv3-Tiny-3L implementation in the DPU IP core requires adjustment on the workload. The updated workload is also determined by the DPU architecture. The workload of the YOLOv3-Tiny-3L model before and after the implementation in the DPU IP core is shown in Table 5.5.

Table 5.5: YOLOv3-Tiny_3L workload on DPU IP core

Layer	**Layer Type (Activation Function)**	**Filters**	**Size/ Stride**	**Input / Output**	**Workload (MAC ops)**	**Workload on DPU arch (MAC ops, Memory access)**
0	Convolutional (Leaky)	16	3x3/1	416x416x3 / 416x416x16	152289280	411873280
1	Maxpool		2x2/2	416x416x16 / 208x208x16	2768896	4845568
2	Convolutional (Leaky)	32	3x3/1	208x208x16 / 208x208x32	400105472	917629440
3	Maxpool		2x2/2	208x208x32 / 104x104x32	1384448	1817088
4	Convolutional (Leaky)	64	3x3/1	104x104x32 / 104x104x64	399413248	573139840
5	Maxpool		2x2/2	104x104x64 / 52x52x64	692224	757120
6	Convolutional (Leaky)	128	3x3/1	52x52x64 / 52x52x128	399067136	514084480
7	Maxpool		2x2/2	52x52x128 / 26x26x128	346112	407680
8	Convolutional (Leaky)	256	3x3/1	26x26x128 / 26x26x256	398894080	557927552
9	Maxpool		2x2/2	26x26x256 / 13x13x256	173056	221312
10	Convolutional (Leaky)	512	3x3/1	13x13x256 / 13x13x512	398807552	515986016
11	Maxpool		2x2/1	13x13x512 / 13x13x512	346112	430976

Continued on next page

Table 5.5 – *Continued from previous page*

Layer	**Layer Type (Activation Function)**	**Filters**	**Size/ Stride**	**Input / Output**	**Workload (MAC ops)**	**Workload on DPU arch (MAC ops, Memory access)**
12	Convolutional (Leaky)	1024	3x3/1	13x13x512 / 13x13x1024	1595057152	2009425600
13	Convolutional (Leaky)	256	1x1/1	13x13x1024 / 13x13x256	88647936	114694944
14	Convolutional (Leaky)	512	3x3/1	13x13x256 / 13x13x512	398807552	515986016
15	Convolutional (Linear)	27	1x1/1	13x13x512 / 13x13x27	4677075	6039488
16	YOLO					
17	Route					
18	Convolutional (Leaky)	128	1x1/1	13x13x256 / 13x13x128	11097216	15520960
19	Upsample		2x2/1	13x13x128 / 26x26x128		
20	Route					
21	Convolutional (Leaky)	256	3x3/1	26x26x384 / 26x26x256	1196336128	1561798784
22	Convolutional (Linear)	27	1x1/1	26x26x256 / 26x26x27	9363276	12416768
23	YOLO					
24	Route					
25	Convolutional (Leaky)	128	1x1/1	26x26x256 / 26x26x128	44388864	62083840
26	Upsample		2x2/1	26x26x128 / 52x52x128		
27	Route					
28	Convolutional (Leaky)	128	3x3/1	52x52x128 / 52x52x128	1595230208	1952379520
29	Convolutional (Linear)	27	1x1/1	52x52x128 / 52x52x27	18763056	22911616
30	YOLO					

Continued on next page

Table 5.5 – *Continued from previous page*

Layer	**Layer Type (Activation Function)**	**Filters**	**Size/ Stride**	**Input / Output**	**Workload (MAC ops)**	**Workload on DPU arch (MAC ops, Memory access)**
Total					**7116656079**	**9772377888**

As it can be seen in Table 5.5, the workload on DPU architecture has considerably increased compared to the initial workload on the model. The workload on the DPU architecture increases in every model layer.

Chapter 6

Discussion

6.1 Bolt Detection

This research utilized a relatively small dataset of 200 images for training the YOLOv3-tiny-3L model. While the training process managed to achieve a low loss of 0.3254 at 8000 iterations, it is essential to discuss the implications of the size of the dataset on the model's performance. With deep learning models, especially for complex tasks such as object detection, a large dataset is usually desirable to capture a broad range of variability and prevent overfitting, which occurs when the model learns the training data too well, including noise or outliers, compromising its ability to generalize to new data. Although the achieved loss value suggests that the model fits the training data effectively, the potential for overfitting is significant, given the limited size of the dataset. Hence, despite the promising training results, we must proceed with caution. It is paramount to validate the model on a separate set, consisting of previously unseen data to evaluate the model's ability to generalize.

When evaluating our YOLOv3-tiny model on the test dataset of 50 images, the mAP stood at 33.07% with an IoU threshold of 0.5. This threshold is a common benchmark for object detection tasks, and it means that a prediction is considered correct if the overlap between the predicted and actual bounding box is 50% or more. The value of mAP is categorized as normal for the tiny yolo model because, in the original model YOLOv3-tiny with COCO dataset, the mAP value is 33.1 [11] [6]. The mAP value from the bolt detection successfully achieved the first subgoal for this study in Section 1.4, which is minimum achieve mAP 33%.

Regarding each class performance, the model achieved an average precision (AP) of 29.43% for the first class and 36.71% for the second class.

This shows that the model is currently more efficient at identifying objects from the second class than the first one.

Further examination of the test results also revealed that the predicted bounding boxes by the model were not perfectly aligned with the ground truth from the labelled dataset, as shown in Figure 5.2b. This could result from several factors, such as the small size of the training dataset, inadequate variety in the training samples, or the model's insufficient capacity to learn complex patterns. This could also be why the mAP value is lower than the normal YOLOv3 model [6]. However, for this research aim, the bounding box is not a problem as long as the bolt still can be detected because the important output from bolt detection in this research is the middle point of bolts.

Moreover, it was observed that the model failed to detect some instances of the bolt object in the test images. For instance, in Figure 5.2b shows one bolt in the right upper corner has missed from detection. This suggests that the model might be struggling with identifying certain objects, possibly due to a lack of representative samples in the training data or due to complexities and similarities in features shared with the background or other objects.

Furthermore, the confidence level of the detected bolts in Figure 5.2b is categorized as very high, which is near one. It is because the training data is limited, which is 200 images and the training data was obtained from the same environment setup as testing data. For further development, the training should incorporate more data in various environment setups to produce more representative testing results.

To address these issues, future iterations of this study might consider expanding the training dataset, particularly with more diverse samples of the underrepresented or misclassified objects. Using data augmentation techniques can also enrich the dataset and allow the model to learn from a wider range of scenarios. Fine-tuning the model parameters and adjusting the model's architecture could further improve both object detection and bounding box precision. Also, experimenting with different IoU thresholds could potentially lead to more accurate performance metrics, reflecting the model's performance more precisely.

6.2 Localization

The result of this study shows that the visual-inertial pose estimation with perspective-n-points, bolt detection, and Kalman filter can estimate close results from ground truth data. The Kalman filter fuses the visual pose estimation, IMU data and bolt references data to get the result of visual-

inertial pose estimation. The position results from pose estimations, which are represented as the blue line in Figure 5.3, can follow the track movement of the ground truth position. Mostly, the position estimation results are seen to have a bias from the ground truth, but some of the parts of the position are seen to move towards the ground truth. Due to the limitation of communication handling with Cobot as base reference position provider, the red line in Figure 5.3 as ground truth is observed has several gaps on the movement. However, since the data acquisition was conducted in synchronisation for all of the data, for instance, IMU data, images data, and ground truth data, then after the gap communication in a Cobot, the other data are still valid, the gaps in the ground truth affect the 3D location assignment of the bolts because of loss tracking. In this research, the problem was encountered by reassigning the 3D location of the bolt manually. It becomes the limitation of this study that needs to be solved in further study. In addition, the comparison of ground truth and visual-inertial estimation in the X, Y and Z axis shows that the position estimation can follow the movement in ground truth well. These results are consistent with the 3D tracking result that was discussed earlier.

One interesting finding is that the RMSE of positions is normally distributed for both visual-inertial pose estimation and visual pose estimation. The mean value of visual-inertial pose estimation has a slightly better value than visual pose estimation, with values of 39.69 mm and 41.30 mm, respectively. The reason is that the data fusion with IMU data and bolt references position makes the estimation move towards the ground truth. In the visual inertial pose estimation, some of the errors have values near zero; that is because the bolt references update to the Kalman filter. The inertial data still seem insignificant to help the estimation because the data rate of the IMU in this study is still low, which is around 30 Hz. Further study is expected can fix it and improve it to get better estimation results. Moreover, the sensor variance value for visual pose estimation was set quite low, which is 0.001, which means the confidence of the visual pose estimation is high. Whereas the variance value for bolt reference update is 0, which means the bolt reference position has confidence as ground truth. Because of variance from visual pose estimation is low, the distribution of visual-inertial pose estimation is close to the visual pose estimation.

For the orientation result, orientation errors for visual-inertial estimation and visual pose estimation show the normal distribution. However, one unexpected finding was there are outliers in error orientation from visual-inertial pose estimation that make the histogram in Figure 5.8 not seen as well distributed. It is indicated because the Kalman filter uses 16 bits of a fixed

point in the state covariance matrix, and there was saturation in the calculation. The orientation result from visual pose estimation was not used directly in the Kalman filter update because the reason was to keep the calculation small in the Kalman gain, but the orientation result was used undirectly in the update process. Furthermore, the mean value between error orientation in visual-inertial and visual pose estimation is close. However, the visual-inertial estimation has a slightly higher error orientation because of the outliers data.

One of the key results of this study is the relatively low mean value of the RMSE for the position, which stands at 39.69mm. This result aligns with our second subgoal detailed in Section 1.4. Furthermore, the standard deviation for RMSE position being lower than its mean value suggests that the RMSE position values are tightly clustered, reflecting consistent performance in the model's localization capability. Variations in performance are observed upon dividing the absolute inaccuracy in the positional estimates along the X, Y, and Z axis. The mean absolute error and standard deviation on the X-axis are 18.30mm and 8.71mm, respectively. A larger mean absolute error of 30.98mm and an 8.61mm standard deviation are displayed on the Y-axis. According to the lowest mean absolute error of 14.39mm and the smallest standard deviation of 4.66mm among them, the model appears to perform best when estimating positions along the Z-axis. Additionally, our analysis of the model's performance in estimating orientation reveals an average error of 4.80 degrees. However, the standard deviation of 5.39 degrees suggests a significant spread in the orientation error rates, indicating a possible area for enhancement. Overall, the Visual-Inertial pose estimation method is reasonably robust in estimating both position and orientation. Nevertheless, certain aspects, notably the Y-axis localization and orientation estimation, show room for improvement. Future iterations of our work could benefit from fine-tuning the model parameters or incorporating advanced algorithms to bolster performance in these areas.

Moreover, the visual localization in this study necessitates at least six key points for the perspective-n-point algorithm, represented by the bolt objects. This requirement emerges as a constraint of the system. If fewer than six bolts are detected, the visual localization fails to provide any localization estimates, compelling the system to rely solely on the inertial sensor's localization. However, the potential for future improvements exists in refining the fusion algorithm between the visual and inertial sensors, which could lessen the requisite number of key points for localization estimation. Notably, Zhao et al. [26] proposed an alternative fusion approach for visual and inertial sensors, which requires fewer key points. They introduced a novel algorithm named

Inertial Perspective-n-Point (IPNP), which demands only two key points. Such an advancement would be highly advantageous in scenarios like bolt tightening, where the possibility of detecting fewer than six bolts increases as the tightening tool progresses with the task. Consequently, further exploration and adaptation of this algorithm could pave the way for a more flexible and reliable localization process in future iterations of this work. This avenue of future research could potentially transform the system's limitations into opportunities for innovation and efficiency in bolt detection and localization.

6.3 Time Execution

Table 5.3 provides a detailed overview of the time execution for various processes involved in our object detection and localization task. Starting with the FPGA logic, the DPU for YOLOv3-Tiny-3L took 25ms to complete, not including the time for inference. This gives an indication of the base computational time required by the FPGA for processing the object detection model without actual inference.

When considering the full inference time, two different systems were analyzed: FPGA Logic & Processor and the ZYNQ-ARM Processor. Full inference using the DPU on FPGA Logic & Processor took significantly less time, 85ms, than the ZYNQ-ARM Processor alone, which required a substantial 1430ms. This shows a clear advantage in using the DPU with FPGA for inference regarding execution speed.

Looking at the ESEKF IP-core internal process, without including AXI4 communication, the execution time is quite small at 176 microseconds. However, when considering the AXI4 communication, the execution time increases to 8.8ms for the Predict function and 10ms for the Update function. Moreover, the time spent on the processor for pre-pose estimation and pose estimation processes is relatively minimal, at 7ms and 2.3ms, respectively. The total time for bolt detection and localization is 113.1 ms, which integrates the full inference bolt detection with DPU, predict and update function ESEKF, pre-pose and pose estimation process. The total time of bolt detection and localization achieves the third subgoal of this study in Section 1.4, where the total time of 113.1 ms is less than the target of 150 ms.

Overall, this analysis of time execution emphasizes the efficiency of using FPGA logic fabric and DSP slices in conjunction with an ARM processor for inference tasks and also highlights the importance of considering communication time when dealing with functions that involve external communication, like the ESEKF Predict and Update functions. Further

optimization of these processes could lead to more efficient execution times, leading to real-time or near-real-time performance, which is often crucial in object detection and localization tasks.

6.4 FPGA Implementation

Table 5.4 provides a summary of FPGA resource utilization for the DPU, ESEKF, and other components. The utilization is presented in absolute values and the corresponding percentage of total available resources. The DPU and ESEKF together accounted for a substantial majority of the utilized resources across all categories. For example, in terms of CLB LUTs and Registers, the DPU and ESEKF used 64% and 62% of the total available resources, respectively. The high utilization indicates an efficient use of the FPGA's resources by these functions, but it also suggests that there is limited room for adding more complex functionalities without exceeding the FPGA's resource capacity. Among other resources, the DSPs were the most utilized, with 72% of the total available DSPs used. This high usage is driven primarily by the DPU and ESEKF, underlining the computation-intensive nature of these components. The relatively low usage of the CARRY8 and F7 Muxes, at 11% and 7%, respectively, suggests these resources are not significantly required by the current implementation. From the perspective of utilization percentage, it seems that there is a possibility to enhance the DPU architecture or increase the complexity of ESEKF computations. Numerous iterations have been conducted to improve the DPU architecture and augment its overall complexity. Nevertheless, subsequent to these modifications, issues have arisen in the routing logic. The present utilization outcomes represent the optimal and feasible outcomes for execution. In general, this table highlights the resource efficiency of the current FPGA implementation but also signals potential limitations for further expansion or complexity increase due to the already substantial resource usage.

The implementation strategy that was used in the implementation stage in Vivado design suite software was Congestion_SpreadLogic_low. This implementation strategy was used to avoid the timing violation. As can be seen in Figure 5.10, there is no timing violation in hold or setup. When the implementation strategy used the default, which is Performance_Explorer, there was a timing violation in the setup. Congestion_SpreadLogic_low strategy prioritizes reducing congestion by spreading the logic out in the available FPGA area. It tries to balance logic density and distribution to avoid high-congestion areas, which can cause timing violations because signals take

longer to travel through congested areas. By spreading out the logic and reducing congestion, the timing of the design could be better controlled, and timing violations could be avoided.

The power estimation in Figure 5.11 did not incorporate the activity file or Switching Activity Interchange Format (SAIF) file. Hence, this could be seen as a preliminary or approximate power estimation. The inherent power calculation from Vivado incorporates the power drained when the device is at rest, referred to as static power consumption, and the power consumed when the device is in active mode, known as dynamic power consumption. However, it lacks detailed insight into the design's switching activity, which could cause disparities from the true power usage. Therefore, while this default power estimation is a useful starting point, it might not accurately represent the design's actual power usage during real-world operation. Since, in this study, the power is not the main concern, the rough power estimation can be used as future bench-marking when the system is improved.

The YOLOv3-Tiny-3L workload on the DPU IP core, as displayed in Table 5.5, provides significant insights into the performance of the DPU core during the execution of different layers. It can be seen that the convolutional layers bear a significant portion of the total workload, which reaches approximately 7.1 billion operations. However, when executed on a DPU architecture, the workload rises to nearly 9.8 billion operations. The observed discrepancy between the workload and workload on the DPU architecture is due to two main factors. Firstly, the workload on the DPU is influenced by layer-specific operations, which include convolutions, pooling, and activation functions. These operations are determined by various parameters such as layer type, filter size, stride, and the dimensions of the input and output. Secondly, the DPU architecture adds its own complexities. It considers not just the basic computational operations of the network layers but also several additional aspects integral to its architecture. This includes overheads related to loading weights and biases into memory, transferring data between the CPU and DPU, managing memory hierarchies, data type conversions, buffer handling, instruction scheduling, and latency. In addition, the DPU's parallel processing model may require synchronization between different processing elements, contributing to the total workload. This suggests that when deploying a deep neural network like YOLOv3-Tiny-3L on a DPU, attention must be given to both network-specific and architecture-specific parameters. Ensuring optimal execution requires careful consideration of the workload distribution across layers, the DPU architecture, and potential overheads. It underscores the importance of hardware-software co-design and co-optimization when

deploying deep learning models on specialized hardware like DPU.

Chapter 7

Conclusions and Future work

In the subsection conclusions, we examine our objectives and assess the outcomes. Some of the study's limitations will be mentioned in subsection limitations. In the section titled "Future Work", we outline what we consider are the study's reasonable next steps as well as any remaining issues.

7.1 Conclusions

The primary objective of this study was to implement a visual-inertial localization mechanism incorporating bolt detection with machine learning, the perspective-n-points algorithm, and data fusion using the Kalman filter. This research has offered fresh perspectives on the performance and efficient utilization of Field Programmable Gate Arrays (FPGAs) when employing the YOLOv3-Tiny-3L and ESEKF algorithms.

The bolt detection implementation using YOLOv3-Tiny-3L yielded an mAP value of 33.07 for two classes of detection, successfully fulfilling the first subgoal of this study. Our findings indicate that implementing the YOLOv3-Tiny-3L model in FPGA using a DPU IP core can drastically reduce execution time compared to processor-based implementation, from 1430 ms to a notably quicker 85 ms.

The visual-inertial localization utilizing the ESEKF algorithm yielded promising results in pose estimation. The average RMSE for the position was 39.69 mm with a standard deviation of 9.9 mm, and the average orientation error was 4.8 degrees with a standard deviation of 5.39 degrees. Regarding localization position, the average value of 39.69 mm surpasses the second subgoal of this study, delivering an accuracy of the position below the targeted 100 mm. Additionally, the total time required for bolt detection through to

complete localization is a modest 113.1 ms, which is well below the target set for the third subgoal.

The insight of this study is the FPGA logic implementation can drive the faster and more efficient execution of object detection and the ESEKF algorithm. Whereas, faster execution require complex logic implementation, which drives to use of more utilization resource in FPGA. The balancing between the performance and hardware resources is important when defining the target of the study.

7.2 Limitations

This study encountered a limitation pertaining to the generation of a synchronized dataset. The design of our IMU sensor, camera, and Cobot communication systems was intended to produce synchronous data accompanied by a timestamp. However, any latency or overload from any single device could bottleneck the data from other devices. Consequently, this study's dataset only incorporates 10Hz image data and 30Hz IMU data.

Another limiting factor in this study is the available hardware resources in the FPGA. Due to this constraint, the DPU architecture could not employ its highest supported variant. Additionally, the hardware resource limitations hindered the full implementation of the ESEKF algorithm within the FPGA logic. Some computational aspects within the ESEKF algorithm, including converting the Euler angle into quaternion and the inverse matrix process, are still executed outside the FPGA due to these limitations.

Therefore, while this research provides significant insights into the performance and utilization of FPGA with YOLOv3-Tiny-3L and ESEKF algorithms, it also underscores the importance of addressing potential bottlenecks in data synchronization and resource constraints within FPGA implementations. These findings can guide future efforts to further optimize the performance and resource utilization of these algorithms on FPGA.

7.3 Future work

Future work in the domain of bolt detection could require the assembly of a more comprehensive dataset for model training and testing. If feasible, the inclusion of additional types of bolts could yield more representative and precise object detection outcomes. Additionally, investigating different YOLO models and versions for FPGA implementation could yield intriguing

comparative results with the existing implementation in this study.

In the context of localization, future studies could strive to utilize higher sampling rates for the camera and IMU, potentially enhancing the accuracy of estimations. Furthermore, exploring a wider range of camera and IMU sensors could offer intriguing avenues for further investigation.

Regarding the implementation of the ESEKF algorithm, future work could expand the calculation model to update not only the position variables but also the velocity and orientation variables. The state covariance matrix could also be broadened to encompass additional parameters, thereby enhancing the precision of the calculation model. However, such expansions would necessitate a more powerful FPGA device with greater hardware resources. These prospective directions demonstrate the exciting potential for ongoing innovation and advancement in this field.

7.3.1 What has been left undone?

The present study did not address the integration and optimization of the PYNQ platform with Python code to implement all algorithms in real time. We primarily relied on a dataset for algorithm testing. Consequently, future research that focuses on integrating these algorithms and conducting tests with real-time data could constitute a significant advancement in this area of study. This approach could potentially yield more dynamic and practical results, thereby offering a comprehensive perspective on the performance of the algorithms in a real-world scenario.

7.4 Reflections

This study utilized images as visual data that possibly have ethical concerns. For handling the ethical question, we did not include humans in the frame of images in the testing process. The images only contain the tightening objects and will be taken only with the permission of the company host for this project, which is Atlas Copco. The image data from the dataset is stored safely in the company computer with secure and limited access. The actual implementation tools utilized a camera without a saving procedure for the image that has been taken.

As part of the 2030 Agenda for Sustainable Development, the United Nations adopted 17 sustainability goals to end poverty, protect the planet, and ensure peace and prosperity. This research will contribute to Goal 9: Industries, Innovation, and Infrastructure. The results of this research will

open the possibility of enhancing the future tightening process for the Atlas Copco industry with no human error in the tightening process.

References

[1] R. Danielsson, "Analysis of Simulation tool for Future Flexible Assembly lines," Master's thesis, Luleå University of Technology, Sweden, 2022. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-91545 [Page 1.]

[2] S. Kumakura and K. Saito, "Tightening Sequence for Bolted Flange Joint Assembly," in *Pressure Vessels and Piping Conference*. American Society of Mechanical Engineers Digital Collection, Aug. 2008. doi: 10.1115/PVP2003-1867 pp. 9–16. [Online]. Available: https://dx.doi.org/10.1115/PVP2003-1867 [Page 1.]

[3] I. Coria, M. Abasolo, J. Aguirrebeitia, and I. Heras, "Study of bolt load scatter due to tightening sequence," *International Journal of Pressure Vessels and Piping*, vol. 182, p. 104054, May 2020. doi: 10.1016/j.ijpvp.2020.104054. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308016120300326 [Page 1.]

[4] M. Abid, "The effect of bolt tightening methods and sequence on the performance of gasketed bolted flange joint assembly," *Structural engineering & mechanics*, vol. 46, pp. 843–852, Jun. 2013. doi: 10.12989/sem.2013.46.6.843 [Page 1.]

[5] V. M. S. i. m. e. Soni, "A machine learning optical system to ensure that human assembly technicians use the specified bolt tightening sequence in assembly line manufacturing," Thesis, University of Texas at Austin, USA, May 2020, accepted: 2021-08-12T18:37:56Z. [Online]. Available: https://repositories.lib.utexas.edu/handle/2152/86986 [Page 2.]

[6] J. Redmon and A. Farhadi, "YOLO: Real-Time Object Detection." [Online]. Available: https://pjreddie.com/darknet/yolo/ [Pages 4, 63, and 64.]

[7] Peter Bock, *Getting It Right : R&D Methods for Science & Engineering*, 1st ed. San Diego: Academic Press, 2001. ISBN 978-0-12-108852-1. [Online]. Available: https://www.elsevier.com/books/getting-it-right/bock/978-0-12-108852-1 [Page 4.]

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," May 2016, arXiv:1506.02640 [cs]. [Online]. Available: http://arxiv.org/abs/1506.02640 [Pages 7 and 8.]

[9] R. Girshick, "Fast R-CNN," Sep. 2015, arXiv:1504.08083 [cs]. [Online]. Available: http://arxiv.org/abs/1504.08083 [Page 7.]

[10] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," Jan. 2023, original-date: 2022-09-11T16:39:45Z. [Online]. Available: https://github.com/ultralytics/ultralytics [Page 8.]

[11] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," Apr. 2018, arXiv:1804.02767 [cs]. [Online]. Available: http://arxiv.org/abs/1804.02767 [Pages 8, 31, 37, and 63.]

[12] L. Zhao and S. Li, "Object Detection Algorithm Based on Improved YOLOv3," *Electronics*, vol. 9, no. 3, p. 537, Mar. 2020. doi: 10.3390/electronics9030537 Number: 3 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2079-9292/9/3/537 [Page 8.]

[13] R. Mehta and C. Ozturk, "Object detection at 200 Frames Per Second," May 2018, arXiv:1805.06361 [cs]. [Online]. Available: http://arxiv.org/abs/1805.06361 [Pages 8 and 9.]

[14] H. Gong, H. Li, K. Xu, and Y. Zhang, "Object Detection Based on Improved YOLOv3-tiny," in *2019 Chinese Automation Congress (CAC)*, Nov. 2019. doi: 10.1109/CAC48633.2019.8996750 pp. 3240–3245, iSSN: 2688-0938. [Pages 9 and 37.]

[15] Xilinx, "PYNQ," Jun. 2023, original-date: 2016-01-20T01:16:27Z. [Online]. Available: https://github.com/Xilinx/PYNQ [Page 9.]

[16] Xilinx, "Vitis AI — Vitis™ AI 3.0 documentation." [Online]. Available: https://xilinx.github.io/Vitis-AI/ [Page 9.]

[17] Xilinx, "DPU on PYNQ," May 2023, original-date: 2020-04-29T23:43:53Z. [Online]. Available: https://github.com/Xilinx/DPU-PYNQ [Pages 10, 11, and 46.]

[18] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. doi: 10.1145/358669.358692. [Online]. Available: https://dl.acm.org/doi/10.1145/358669.358692 [Page 11.]

[19] S. Li, C. Xu, and M. Xie, "A Robust O(n) Solution to the Perspective-n-Point Problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1444–1450, Jul. 2012. doi: 10.1109/TPAMI.2012.41 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. [Page 11.]

[20] E. Marchand, H. Uchiyama, and F. Spindler, "Pose Estimation for Augmented Reality: A Hands-On Survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 12, p. 2633, 2016. doi: 10.1109/TVCG.2015.2513408. [Online]. Available: https://inria.hal.science/hal-01246370 [Pages 11 and 12.]

[21] "OpenCV: Perspective-n-Point (PnP) pose computation." [Online]. Available: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html [Pages xi, 12, and 42.]

[22] Universal Robots, "Universal Robots e-Series User Manual," 2022. [Online]. Available: https://www.universal-robots.com/download/ [Pages xi and 13.]

[23] V. Madyastha, V. Ravindra, S. Mallikarjunan, and A. Goyal, "Extended Kalman Filter vs. Error State Kalman Filter for Aircraft Attitude Estimation," in *AIAA Guidance, Navigation, and Control Conference*. Portland, Oregon: American Institute of Aeronautics and Astronautics, 2012. doi: https://doi.org/10.2514/6.2011-6615. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2011-6615 [Page 14.]

[24] L. Marković, M. Kovač, R. Milijas, M. Car, and S. Bogdan, "Error State Extended Kalman Filter Multi-Sensor Fusion for Unmanned Aerial Vehicle Localization in GPS and Magnetometer Denied Indoor Environments," in *2022 International Conference on Unmanned Aircraft*

Systems (ICUAS), Jun. 2022. doi: 10.1109/ICUAS54217.2022.9836124 pp. 184–190, iSSN: 2575-7296. [Page 14.]

[25] I. Brigadnov, A. Lutonin, and K. Bogdanova, "Error State Extended Kalman Filter Localization for Underground Mining Environments," *Symmetry*, vol. 15, no. 2, p. 344, Feb. 2023. doi: 10.3390/sym15020344 Number: 2 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2073-8994/15/2/344 [Pages 14 and 20.]

[26] Y. Zhao, E. Tkaczyk, and F. Pan, "Visual and inertial sensor fusion for mobile X-ray detector tracking: demo abstract," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020. doi: 10.1145/3384419.3430435. ISBN 978-1-4503-7590-0 pp. 643–644. [Online]. Available: https://dl.acm.org/doi/10.1145/3384419.3430435 [Pages 17, 42, and 66.]

[27] E. Aparicio-Esteve, J. Ureña, . Hernández, D. Pizarro, and D. Moltó, "Using Perspective-n-Point Algorithms for a Local Positioning System Based on LEDs and a QADA Receiver," *Sensors*, vol. 21, no. 19, p. 6537, Jan. 2021. doi: 10.3390/s21196537 Number: 19 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/21/19/6537 [Pages 18 and 42.]

[28] G. Pan, A. H. Liang, J. Liu, M. Liu, and E. X. Wang, "3-D Positioning System Based QR Code and Monocular Vision," in *2020 5th International Conference on Robotics and Automation Engineering (ICRAE)*, Nov. 2020. doi: 10.1109/ICRAE50850.2020.9310908 pp. 54–58. [Pages 19 and 42.]

[29] U. robots, "RTDE client python library." [Online]. Available: https://github.com/UniversalRobots/RTDE_Python_Client_Library [Page 28.]

[30] STMicroelectronics, "LSM9DS1 - 9-axis iNEMO inertial module (IMU): 3D magnetometer, 3D accelerometer, 3D gyroscope with I2C and SPI - STMicroelectronics," Mar. 2015. [Online]. Available: https://www.st.com/en/mems-and-sensors/lsm9ds1.html [Page 28.]

[31] J. Liu, W. Gao, and Z. Hu, "Visual-Inertial Odometry Tightly Coupled with Wheel Encoder Adopting Robust Initialization and

Online Extrinsic Calibration," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019. doi: 10.1109/IROS40897.2019.8967607 pp. 5391–5397, iSSN: 2153-0866. [Page 30.]

[32] Alexey, "Yolo v4, v3 and v2 for Windows and Linux," May 2023, original-date: 2016-12-02T11:14:00Z. [Online]. Available: https://github.com/AlexeyAB/darknet [Pages 35 and 37.]

[33] David8862, "keras-YOLOv3-model-set." [Online]. Available: https://github.com/david8862/keras-YOLOv3-model-set [Page 38.]

[34] Xilinx, "Developing a Model — Vitis™ AI 3.0 documentation." [Online]. Available: https://xilinx.github.io/Vitis-AI/docs/workflow-model-development.html [Page 38.]

[35] A. Sanchez, "Vehicle State Estimation on a Roadway," Jun. 2023, original-date: 2020-12-17T04:26:53Z. [Online]. Available: https://github.com/jasleon/Vehicle-State-Estimation [Page 43.]

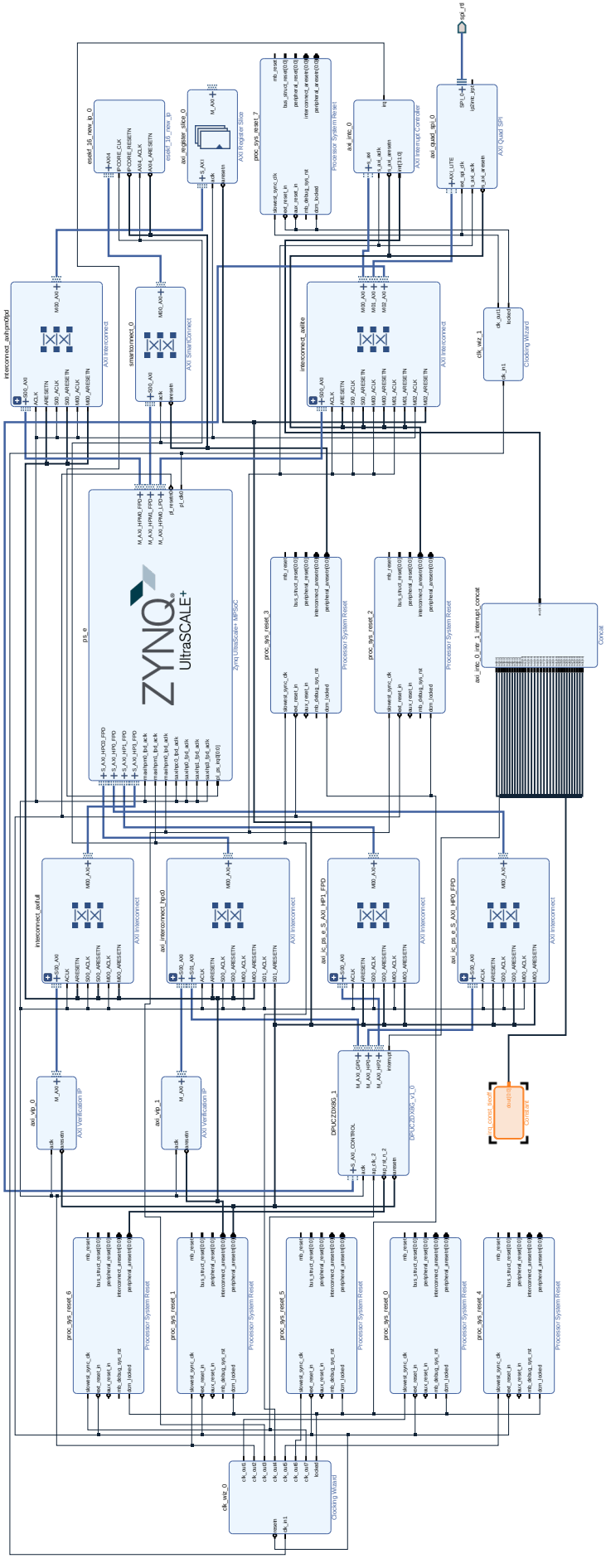
[36] ARM, "AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite," 2013. [Online]. Available: https://www.arm.com/architecture/system-architectures/amba/amba-specifications [Page 43.]

Appendix A

FPGA Schematic

A.1 Full FPGA Schematic

The following is a complete schematic of the final FPGA integration results.


ZYNQ
UltraSCALE+
ps_e
Zynq UltraScale+ MPSoC
axi_interconnect_hp0
interconnect_axifull
interconnect_ahmphbo
interconnect_aelfie
smartconnect_0
axi_quad_spi_0
axi_intc_0
axi_intc_0_irq_1_interrupt_concat
Concat
proc_sys_reset_0
proc_sys_reset_1
proc_sys_reset_2
proc_sys_reset_3
proc_sys_reset_4
proc_sys_reset_5
proc_sys_reset_6
Processor System Reset
AXI Interconnect
AXI SmartConnect
AXI Quad SPI
AXI Interrupt Controller
Clocking Wizard
AXI Verification IP
AXI Register Slice
Processor System Reset
DPUCZDX8G_1
DPUCZDX8G_v3_0
axi_vip_0
axi_vip_1
clk_wiz_0
clk_wiz_1
spi_rtl

TRITA-EECS-EX- 2023:746