Tracking and Pose Estimation of a Robotic System using Computer Vision

Karl Spiteri

Supervisor: Dr Ing. Jeremy Scerri

June 2023

A dissertation submitted to the Institute of Engineering and Transport in partial fulfilment of the requirements for the degree Bachelor of Engineering (Honours) in Control and Electronics Engineering.

Authorship Statement

This dissertation is based on the results of research carried out by myself, is my own composition, and has not been previously presented for any other certified or uncertified qualification.

The research was carried out under the supervision of Dr Ing. Jeremy Scerri.

6th June 2023

Copyright Statement

6th June 2023

____

Acknowledgements

Thank You to my supervisor, friends, and family who supported me throughout my thesis. Their support, encouragement, and faith in me were significant. Their support helped me overcome obstacles and hurdles I encountered.

Abstract

This thesis presents a comprehensive approach to tracking and pose estimation of a robotic arm using computer vision techniques. The primary goal of the research is to develop a real-time computer vision system capable of tracking objects, estimating joint angles, and providing a live skeleton view of the robotic arm.

The development process involved several key steps. Firstly, a tracking system was constructed to enable robust object tracking. This involved implementing algorithms for feature detection, matching, and tracking to accurately follow the movement of the target object. Additionally, a marker generator was built to create unique markers tailored to the robotic arm. These markers were designed to provide distinct visual features for reliable detection and pose estimation.

To ensure precise marker placement on the robotic arm, 3D placers were designed using CAD techniques. These placers were custom designed to fit the specific dimensions and geometry of the robotic arm, facilitating accurate marker positioning. The markers were strategically placed on the arm to enable precise joint angle estimation and tracking.

Marker data acquisition was a crucial step in the process. The computer vision system read the marker data and extracted the necessary matrices for subsequent calculations. These matrices contained essential information about the position, orientation, and motion of the markers, enabling further analysis.

Although time constraints limited the completion of the final calculations and the construction of the skeleton view, the groundwork for these tasks was laid. The calculations would involve processing the marker data to estimate the joint angles of the robotic arm accurately. This information would contribute to real-time pose estimation and tracking of the arm. Additionally, a live skeleton view of the arm would be generated, visually depicting the arm's structure and movements in real-time.

The proposed computer vision system has the potential to enhance the capabilities of robotic arms by providing real-time object tracking, joint angle measurement, and visual representation. Future work should focus on finalizing the calculations and implementing the skeleton view to realize the complete vision of the project.

Contents

List of Figures

List of Tables

List of Equations

No table of Equations entries found.

Glossary of Terms

Apologies for the confusion. Here is the updated glossary of terms with the abbreviation letters and a brief explanation:

AI: Artificial Intelligence - Refers to the development of computer systems capable of performing tasks that typically require human intelligence.

ALEXNET: AlexNet Convolutional Neural Network - A specific architecture of convolutional neural network (CNN) that achieved breakthrough results in image classification tasks.

API: Application Programming Interface - A set of rules and protocols that allows different software applications to communicate and interact with each other.

AR: Augmented Reality - An interactive experience that combines virtual elements with the real world to enhance the user's perception and interaction with their surroundings.

CAD: Computer-Aided Design - The use of computer software to create and modify designs, typically for engineering, architectural, or manufacturing purposes.

CNN: Convolutional Neural Networks - A class of deep neural networks designed to process structured grid-like data, such as images or video, using convolutional layers for feature extraction.

COCO: Common Objects in Context - A widely used large-scale object detection, segmentation, and captioning dataset that provides a benchmark for evaluating computer vision models.

COCOAPI: COCO API (Common Objects in Context) - A software library that provides tools and utilities for working with the COCO dataset and evaluating object detection and segmentation algorithms.

CSV: Comma-Separated Values - A simple file format used to store tabular data, where each value is separated by a comma. It is commonly used for data interchange between different software applications.

DH: Denavit-Hartenberg - A convention used in robotics to represent the geometric transformations and kinematics of robotic manipulator systems.

LXML: Lightweight XML - A library for processing and manipulating XML (Extensible Markup Language) data in a Python programming environment.

MAP: Mean Average Precision - A metric commonly used to evaluate the accuracy of object detection and segmentation models by measuring the precision and recall of the predicted results.

NFT: Natural Feature Tracking - A marker less pose estimation technique that tracks and analyses distinct features in the environment or on an object to estimate its position or pose.

NMS: Non-Maximum Suppression - A technique used in object detection algorithms to remove duplicate or overlapping bounding box predictions, retaining only the most confident and accurate detections.

PATH: Portable Accessibility to Health Records - A term referring to a system or technology that enables the secure and portable access and sharing of personal health records among healthcare providers and patients.

PBTEXT: Protocol Buffers Text - A human-readable format for defining and serializing structured data using the Protocol Buffers framework.

PROTOBUF: Protocol Buffers - A language-agnostic data serialization format developed by Google, used for efficient and platform-independent data interchange between different software systems.

RCNN: Region-based Convolutional Neural Networks - A family of object detection algorithms that generate region proposals and use convolutional neural networks to classify and refine the proposed regions.

RESNET: Residual Network - A deep neural network architecture known for its residual connections, which allow for training very deep networks by addressing the vanishing gradient problem.

ROI: Region of Interest - A specific area or region within an image or video frame that is selected for further analysis, processing, or extraction of features.

SLIM: TensorFlow Slim - A lightweight high-level API for defining, training, and evaluating complex machine learning models using the TensorFlow framework.

SSDS: Single Shot Detectors - A type of object detection algorithm that predicts bounding boxes and class labels in a single pass through the network, making it efficient for real-time applications.

SVMS: Support Vector Machines - A popular machine learning algorithm used for both classification and regression tasks, known for its ability to handle complex decision boundaries and high-dimensional data.

TFRECORD: TensorFlow Record - A file format used for storing large amounts of data in

TensorFlow, typically used for efficient input pipelines in machine learning tasks.

VGG: Visual Geometry Group - A research group that has developed various deep learning architectures, including VGGNet, known for its simplicity and effectiveness in image recognition tasks.

VGGNET: VGG Network - A deep convolutional neural network architecture known for its uniform structure with multiple layers and achieved state-of-the-art performance on various visual recognition tasks.

VR: Virtual Reality - An immersive experience that simulates a computer-generated environment, allowing users to interact and explore a virtual world.

YOLO: You Only Look Once - An object detection algorithm that processes the entire image in a single pass, making it efficient for real-time applications and known for its fast inference speed.

List of Abbreviations

AI	Artificial Intelligence
ALEXNET	AlexNet Convolutional Neural Network
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
CNN	Convolutional Neural Networks
COCO	Common Objects in Context
COCOAPI	COCO API (Common Objects in Context)
CSV	Comma-Separated Values
DH	Denavit-Hartenberg
LXML	Lightweight XML
MAP	Mean Average Precision
NFT	Natural Feature Tracking
NMS	Non-Maximum Suppression
PATH	Portable Accessibility to Health Records
PBTEXT	Protocol Buffers Text
PROTOBUF	Protocol Buffers
RCNN	Region-based Convolutional Neural Networks
RESNET	Residual Network
ROI	Region of Interest
SLIM	TensorFlow Slim
SSDS	Single Shot Detectors
SVMS	Support Vector Machines
TFRECORD	TensorFlow Record
VGG	Visual Geometry Group
VGGNET	VGG Network
VR	Virtual Reality
YOLO	You Only Look Once

1. Introduction

Robotic systems can carry out complex tasks thanks to recent advances in robotics. Real-time robotic arm tracking and position estimation are essential for optimal environmental interaction.

This thesis aims to develop a computer vision system capable of real-time object tracking and joint angle estimation for a robotic arm. Additionally, the system endeavours to provide a live skeleton view of the arm, offering a visual representation of its structure and movements. By bridging the principles of computer vision and robotics, this research seeks to enhance the capabilities of robotic arms and elevate their interaction with the environment.

The development of such a system entails various challenges. Real-time object tracking necessitates the implementation of robust algorithms capable of handling occlusions, variations in lighting conditions, and diverse object appearances. Accurate joint angle estimation for the robotic arm requires the fusion of visual information with kinematic models. Furthermore, generating a live skeleton view necessitates the seamless integration of real-time pose estimation and visualization techniques.

To tackle these challenges, this thesis adopts a systematic approach. A tracking system is constructed, leveraging state-of-the-art computer vision algorithms for feature detection, matching, and tracking. Custom-designed markers, tailored specifically for the robotic arm, are created to facilitate precise tracking, and pose estimation. Utilizing CAD techniques, 3D placers are employed to strategically position the markers on the arm, ensuring accuracy and reliability.

Marker data acquisition plays a pivotal role in the operation of the system. The computer vision system reads the marker data and extracts essential matrices that contain information regarding the markers' position, orientation, and motion. These matrices serve as input for subsequent calculations, enabling the estimation of joint angles and facilitating real-time pose estimation and tracking of the robotic arm.

While this thesis outlines the design and implementation of the tracking system, marker generator, and marker placement, certain components remain incomplete due to time constraints. The final calculations for joint angle estimation and the construction of the skeleton view are envisioned as future work.

The potential applications of this research are vast and far-reaching. A robust computer vision system for tracking and pose estimation of robotic arms can significantly enhance performance across various domains, including industrial automation, healthcare, and exploration. By accurately perceiving and interpreting their surroundings, robotic arms can operate with greater autonomy and efficiency, opening new possibilities for human-robot collaboration and interaction.

In conclusion, this thesis presents a novel approach to tracking and pose estimation of a robotic arm using computer vision techniques. The system aims to track objects, estimate joint angles, and provide a live skeleton view of the arm in real-time. Although certain aspects are still unfinished, the groundwork has been created for future field advancements. Computer vision and robotics can improve robotic arms and expand their uses.

2. Literature Review

2.1 Introduction

This chapter examines the implementation of custom object detection techniques for accurately tracking the robotic arm within its environment. Additionally, it investigates the acquisition of precise pose data through ArUco markers, which is then utilized to calculate joint angles using kinematics. The review emphasizes the unique challenges and considerations specific to tracking and pose estimation in the context of a robotic arm system.

2.2 Tracking Techniques in Computer Vision

2.2.1 Traditional Tracking Approaches

Tracking is a fundamental task in various fields, and traditionally, it has relied on different methods to estimate the position and motion of objects over time. These methods often involve the use of filter techniques, which aim to refine and update the estimated states based on available measurements. Filter techniques encompass a range of algorithms that apply mathematical operations to estimate the object's trajectory and improve tracking accuracy. These algorithms utilize a variety of principles, including statistical models, sensor measurements, and motion dynamics. By incorporating these filter techniques, tracking systems can make informed predictions about an object's future location, facilitating numerous applications across industries.

2.2.1.1 Kalman Filters

In the realm of state estimation and tracking, the Kalman filter reigns as a fundamental algorithm, offering a powerful solution for accurate estimation in the presence of noisy measurements. Its elegance lies in its prediction and update steps, which form the essence of this time-tested filter.

The prediction step in the Kalman filter sets the stage for forecasting the system's state based on its previous estimate and a mathematical model of its dynamics. By leveraging the system model, this step projects the state estimate forward in time, granting us a glimpse into the system's future behaviour. Additionally, the prediction step estimates the uncertainty associated with the projected state, providing a measure of confidence in the prediction. This forward-looking aspect of the filter equips us with a valuable tool for anticipating the state of the system, which is particularly useful in tracking applications.

The update step, the second pillar of the Kalman filter, capitalizes on new measurements to refine the state estimate. Through a meticulous process, the filter calculates the difference between the measured value and the predicted measurement based on the projected state, known as the measurement residual. This discrepancy acts as a compass, steering the estimation process towards aligning the predicted measurement with the actual measurement. The Kalman gain, a crucial factor in this step, orchestrates the balance between the prediction and measurement information, assigning weights that optimize the update of the state estimate. By judiciously combining the predicted state, measurement residual, and Kalman gain, the filter reaches an updated state estimate that represents a synthesis of the available information.

The benefits of the Kalman filter are far-reaching and profound. Its ability to leverage a system model and incorporate noisy measurements makes it robust against uncertainties and noise. By optimally fusing the prediction and measurement information, the Kalman filter provides an optimal estimate, maximizing accuracy while considering the confidence levels associated with the information sources. This filter has found extensive applications in diverse domains, including computer vision, robotics, and aerospace, due to its capability to yield accurate and efficient state estimation results.

However, it is important to acknowledge the limitations of the Kalman filter. One significant drawback arises from its reliance on linear and Gaussian assumptions. If the system dynamics or measurement characteristics deviate substantially from linearity or Gaussianity, the filter's performance may degrade. Moreover, the Kalman filter requires an accurate system model, which may be challenging to obtain in certain scenarios. Inaccurate modelling can lead to estimation errors and poor performance. Additionally, the filter assumes knowledge of the measurement noise covariance matrix, which might be difficult to determine precisely in practice.

In summary, the Kalman filter's prediction and update steps constitute the heart of this renowned algorithm. By leveraging system dynamics and measurements, the filter provides a robust and optimal state estimation framework.

While the filter offers significant benefits in terms of accuracy and efficiency, its reliance on linear and Gaussian assumptions, accurate modelling, and knowledge of noise characteristics imposes limitations that should be considered. A balanced view appreciates both the strengths and weaknesses of the Kalman filter, allowing for informed utilization and exploration of alternative approaches in cases where its assumptions are not met.

2.2.1.2 Particle Filters

Particle filters, also known as sequential Monte Carlo methods, are a powerful class of algorithms used for state estimation and tracking in dynamic systems. They can manage non-linear and non-Gaussian systems under unpredictable conditions, making them popular in robotics, computer vision, and tracking systems.

Initializing the particle filter method generates a set of particles representing system states. Prior knowledge or initial distribution samples these particles. This stage estimates the system's status.

Particle filters are based on a repeatable process. It involves prediction, measurement update, and resampling.

A dynamic model of the system's evolution propagates particles ahead in time during prediction. This phase adds uncertainty to particle estimates to account for system volatility. Particles capture system state transitions during prediction.

The measurement update refines particle estimates using observed data. Measurement models calculate particle likelihoods. Particle weights are changed based on how well they match measurements. Real-time information improves particle estimates in this update step.

Resampling keeps particles diverse and prevents degeneracy. Higher-weight particles are more likely to be selected several times in this step, while lower-weight particles may be eliminated. Resampling concentrates particles around high-probability locations, enhancing estimation accuracy. Particle degeneracy and impoverishment are addressed using various resampling procedures to keep the posterior distribution representation diversified and informative.

Particle filters are advantageous. First, they can handle non-linear and non-Gaussian systems. They capture the system's genuine state by approximating complicated, multi-modal distributions. Particle filters can adapt to changing dynamics and incorporate real-time measurements, making them suited for real-time tracking.

Particle filters have downsides. Particle degeneracy, where a few particles dominate the distribution, reduces diversity and accuracy. Particle poverty can also reduce particle spread and limit posterior distribution representation. Selecting resampling algorithms and using adaptive methods can mitigate these concerns.

Particle filters can estimate and track state in dynamic systems. They approximate the system's posterior distribution by initializing, predicting, measuring, and resampling. To estimate accurately, they must handle particle degeneracy and poverty.

2.2.1.3 Mean Shift

Mean Shift tracking is a fantastic approach for video object tracking in computer vision. This non-parametric method can monitor objects without knowing their appearance or motion model.

Initialization starts Mean Shift tracking. In the first frame of the movie, a region of interest is carefully selected around the target object. This window defines the search space for later tracking.

Mean Shift tracking computes a Colour or Feature Histogram within the window. This histogram efficiently captures the target object's detailed colour or feature distribution. With this histogram, Mean Shift begins the intriguing Window Shifting.

Shifting the Window unfolds in each video frame. The algorithm calculates the mean of colour or feature values in the current window and nudges the window toward the highest density. It methodically repeats this procedure to approach convergence.

Mean Shift tracking is all about convergence. The program continuously shifts the window to the domain of similar colour or feature values. This dance continues until the window's position stabilizes, usually with a small change.

Object Tracking is Mean Shift's crowning achievement. The program successfully calculates the object's location and size with the window steady. Examining the final window position helps pinpoint the object's location.

Mean Shift tracking's robustness to typical tracking techniques makes it appealing. It handles lighting, scale, and position well. Mean Shift excels when the target object's appearance can change.

However, Mean Shift tracking has limitations. Similar-looking objects or occlusions might cause tracking errors. In such cases, Mean Shift needs Kalman filtering or multiple hypothesis tracking to improve performance.

Due to its durability and versatility, Mean Shift tracking dominates computer vision. Shifting the Window, Convergence, and Object Tracking are built on Initialization, Colour or Feature Histogram, and Weighted Histogram. As with any algorithm, knowing its limitations lets you weigh its pros and cons.

2.2.2 Deep Learning-based Tracking Techniques

In contrast to traditional tracking methods, recent developments in deep learning have cleared the way for cutting-edge tracking approaches. Deep learning-based tracking makes use of neural networks' ability to extract significant characteristics and patterns from visual data, allowing for more reliable and precise monitoring. These methods frequently include object detection and tracking algorithms that recognize and track important things in real time using CNNs or other architectures. These techniques may deal with complex circumstances, occlusions, and variations in scale and posture by making use of deep learning's capabilities. These cutting-edge object tracking methods based on deep learning present encouraging improvements with the potential to transform several applications and industries.

2.2.2.1 Single Shot Detectors

Deep Learning SSDs revolutionize object detection by employing deep neural networks to detect and locate objects within images swiftly and accurately. This cutting-edge algorithm combines high precision with real-time processing, rendering it invaluable for a diverse range of computer vision applications like object recognition, pedestrian detection, and autonomous driving.

SSDs operate through a comprehensive series of steps that facilitate their exceptional performance. The first step entails constructing the architecture, which encompasses a base network, often a pre-trained CNN such as VGG or ResNet. This base network is complemented by additional convolutional layers that progressively decrease the image's spatial dimensions while concurrently increasing the number of channels.

The subsequent step involves feature extraction, where the base network extracts high-level feature maps from the input image. These feature maps capture intricate details at different levels of abstraction, enabling the detection of objects at various scales. Crucially, SSDs incorporate multiple feature maps, each responsible for detecting objects of specific sizes. Lower-resolution feature maps are adept at detecting larger objects, while higher-resolution feature maps excel at detecting smaller objects.

To propose potential object locations, SSDs employ anchor boxes (or default boxes) in their third step. These anchor boxes are pre-defined bounding boxes of varying sizes and aspect ratios, positioned at multiple scales and locations within the feature maps. By using anchor boxes, SSDs effectively address objects of diverse shapes and sizes.

Step five encompasses the predictions made by SSDs for each anchor box. Specifically, they predict two essential aspects: the class probabilities (e.g., person, car, etc.) and the offsets necessary to refine the anchor boxes and accurately fit the objects' locations. By leveraging this predictive capability, SSDs establish a robust framework for object detection.

The subsequent step in the SSD pipeline is the calculation of loss, a pivotal aspect of training the model. The loss function measures the discrepancy between the predicted and ground truth boxes. It comprises two components: the localization loss, which gauges the accuracy of predicted box coordinates, and the classification loss, which evaluates the precision of predicted class probabilities. This loss calculation allows the SSD model to continually refine its predictions and improve its performance during training.

Finally, SSDs employ NMS in step seven to eliminate redundant and overlapping detections. By retaining only, the most confident detection for each object while suppressing others with significant overlap, NMS ensures accurate and concise results.

SSDs offer numerous benefits in object detection. Their integration of multi-scale feature maps and anchor boxes enables efficient detection of objects at varying scales and aspect ratios. Additionally, their real-time processing capabilities make them ideal for time-sensitive applications, such as video analysis and robotics. Furthermore, SSDs achieve a commendable balance between accuracy and speed, delivering remarkable performance.

However, it's crucial to consider SSDs' drawbacks. The detection of extremely small or significantly occluded things can be difficult for them even though they are excellent at detecting objects of various sizes. Furthermore, the reliance on anchor boxes assumes prior knowledge of the object sizes and aspect ratios, which may not always be accurate. Moreover, the computational complexity of SSDs can be higher than other object detection algorithms, necessitating substantial computational resources.

In summary, Deep Learning SSDs offer a state-of-the-art approach to object detection. By integrating multi-scale feature maps, anchor boxes, precise predictions, loss calculation, and non-maximum suppression, SSDs achieve remarkable accuracy and real-time processing. However, it is important to consider their limitations, such as difficulties in detecting small or occluded objects and the computational resources required. Nonetheless, SSDs have emerged as a prominent and versatile tool in the field of computer vision, facilitating a wide range of applications with their balanced performance.

2.2.2.2 You Only Look Once (YOLO)

YOLO is a ground-breaking deep learning technique that revolutionized object detection.

YOLO detects and localizes images in six steps. First, divide the input image into a grid of cells that anticipate items within their bounds. Grid-based processing simplifies complex scene processing.

YOLO relies on anchor boxes. Anchor boxes are predefined before training. Anchor boxes guide bounding box predictions. Anchor boxes allow YOLO to detect objects of different sizes more accurately.

A strong CNN underpins YOLO. The CNN analyses the incoming image and creates a high-level representation of key object traits. YOLO can recognize objects and detailed patterns in difficult visual situations using this feature extraction approach.

YOLO predicts grid cells after CNN extracts characteristics. Predictions include bounding boxes, class probabilities, and presence probabilities. Bounding box predictions comprise the object's centre (x, y), width (w), and height (h) for precise localization. Class probabilities evaluate the chance of different item classes in the bounding box, while presence probabilities estimate each grid cell's object presence. YOLO detects many objects simultaneously using these predictions.

YOLO uses NMS to reduce duplicate bounding box predictions. This post-processing stage eliminates duplicate detections and picks the most accurate bounding boxes based on presence probabilities. NMS keeps just the best predictions, making YOLO's object detection output more robust.

YOLO outputs objects with bounding boxes and class labels. This quick technique allows YOLO to detect objects in real time, making it suitable for time-critical applications like video analysis, surveillance systems, and autonomous driving.

YOLO has many benefits. Real-time image processing allows object detection. Its single-pass approach to all detection jobs maximizes efficiency. Anchor boxes and the CNN backbone allow YOLO to handle objects of various sizes and complexity, improving its accuracy and versatility.

YOLO has certain drawbacks. YOLO may have trouble identifying tiny things due to its rigid grid and anchor boxes. Real-time performance sacrifices accuracy compared to slower, more sophisticated methods. Despite these drawbacks, YOLO is a prominent object detection approach that pushes real-time computer vision applications.

2.2.2.3 Region-based Convolutional Neural Networks (R-CNN)

R-CNN revolutionized the field of object detection and localization by combining the power of deep learning with a region proposal mechanism. The R-CNN approach encompasses four essential steps to achieve accurate object detection.

Firstly, R-CNN begins by generating potential object regions known as region proposals within an image. These regions act as candidate bounding boxes that may contain objects of interest. Techniques like Selective Search or EdgeBoxes employ low-level image features to efficiently generate these proposals.

The next crucial step involves extracting informative feature vectors using a CNN for each region proposal. By processing the proposed region as a sub-image, the pre-trained CNN, such as AlexNet or VGGNet, extracts a fixed-length feature vector that captures the region's content. This representation serves as an input for subsequent stages.

Object classification forms the heart of R-CNN. The extracted feature vectors are fed into class specific linear SVMs. These SVMs are trained to categorize the region proposals into different object classes or background. By training multiple SVMs (equal to the number of object classes plus one for the background), R-CNN achieves accurate classification.

In addition to object classification, R-CNN also focuses on refining the initially proposed bounding boxes for better alignment with actual object boundaries. This is achieved through a separate regression model, typically a linear regression, which takes the CNN features of the region proposal as input. The regression model predicts corrections needed to refine the bounding box coordinates, thus improving localization accuracy.

R-CNN offers several benefits. By utilizing CNNs, it takes advantage of their exceptional ability to learn hierarchical features from images, enhancing the model's ability to recognize objects accurately. Additionally, the inclusion of region proposals allows R-CNN to focus computation only on relevant regions, making it more efficient than exhaustive search-based methods.

However, R-CNN has its drawbacks. The training and inference processes of R-CNN are computationally expensive due to the need to extract CNN features individually for each region proposal. This leads to slower execution and hinders real-time applications. The multi-stage architecture of R-CNN also introduces complexity and makes it challenging to optimize end-to-end.

In summary, R-CNN combines the power of CNNs with region proposal mechanisms for object detection. It involves region proposal generation, CNN feature extraction, object classification using SVMs, and bounding box refinement through regression. R-CNN benefits from accurate object recognition and the ability to focus on relevant regions, but it suffers from computational complexity and slower execution, limiting its real-time applicability.

2.3 Custom Object Detection using TensorFlow.

2.3.1 TensorFlow for Object Detection

TensorFlow for Object Detection is an extensively used framework for building and deploying object detection models. It offers a comprehensive suite of tools, libraries, and resources that simplify the entire object detection workflow, consisting of eight key steps.

Firstly, TensorFlow Object Detection API serves as the foundation for this framework. It provides a cohesive ecosystem, incorporating pre-trained models and facilitating the training, evaluation, and deployment of custom models. This API empowers developers to tackle a wide range of object detection tasks effectively.

The pre-trained models available in TensorFlow Object Detection API are invaluable starting points. Trained on vast datasets such as COCO or Open Images, these models demonstrate superior performance and serve as a strong foundation for building custom solutions. The Model Zoo within the API provides a diverse array of models, catering to different requirements, including lightweight models for mobile devices and high-accuracy models for demanding applications.

One of the critical steps in object detection is dataset preparation. TensorFlow Object Detection API supports various annotation formats, allowing developers to label their training data with bounding boxes around objects of interest. The API further facilitates data pre-processing and conversion, streamlining the data preparation process.

Training an object detection model is an iterative process wherein the model learns to recognize objects by optimizing its internal parameters. TensorFlow Object Detection API simplifies this step by providing scripts and configurations for efficient training on both CPU and GPU devices. During training, the model adjusts its weights and biases to minimize detection errors and improve accuracy.

Evaluation is a vital step in assessing the trained model's performance. TensorFlow Object Detection API offers evaluation scripts that compute important metrics such as precision, recall, and mAP. These metrics help measure the model's accuracy and generalization capabilities, enabling developers to fine-tune their models accordingly.

Finally, TensorFlow Object Detection API facilitates inference and deployment of the trained models. It provides tools for performing inference using the models on new, unseen data. Integration with popular frameworks like TensorFlow

Serving or TensorFlow Lite enables seamless deployment in a variety of production environments, including web applications and mobile devices.

While TensorFlow for Object Detection offers numerous benefits, it is important to consider its limitations as well. The API's vast ecosystem and pre-trained models significantly reduce the development time and effort required. However, the training process can be computationally intensive, particularly when using complex architectures or large datasets. Adequate computational resources are necessary for efficient training and inference.

2.3.2 Custom Object Detection Workflow

TensorFlow custom object detection in computer vision is powerful. This literature-style summary highlights data gathering, model selection, training, and model optimization in the workflow. It balances the pros and cons of this method.

A good object detection model starts with data collection. Objects are recognized in a diversified group of photos. Annotating things with bounding boxes or masks help train them. Stages build on this annotated dataset.

TensorFlow has many pre-trained object identification models for model selection. EfficientDet, SSD, and Faster R-CNN models have been trained on large datasets like COCO. Transfer learning jumpstarts unique object detection projects.

The annotated dataset is used to train the model. A loss function and optimizer optimize the model's weights to detect and localize objects. Training takes time and computational power. Depending on dataset size and model complexity, training may take hours or days. Patience is essential. Data augmentation increases training efficacy by diversifying training examples.

Evaluate the model's performance after training. To evaluate the model's generalization and object detection abilities, a separate dataset is used. Precision, recall, and mAP quantify model efficacy.

Model optimization then improves detection. Adjusting hyperparameters, datasets, or architecture may improve model performance. Until results are satisfactory, the model's accuracy and efficiency are improved iteratively.

TensorFlow custom object detection has many benefits. First, using pre-trained models speeds up development. TensorFlow's broad ecosystem gives academics and developers many tools and resources. Fine-tuning models for specific areas increase their adaptability and usefulness.

This workflow has limitations. Annotating and accessing varied datasets takes time and resources. Training deep learning models requires a lot of computer power, making it inaccessible to low-resource enterprises. Model optimization takes skill and experimentation to balance performance and efficiency.

Finally, the TensorFlow custom object identification pipeline includes data collection, model selection, training, and optimization. Pre-trained models and TensorFlow's extensive ecosystem help, but data gathering, processing needs, and model refining are difficult. Custom object detection model deployment requires balancing these elements.

2.3.3 Transfer Learning and Fine-Tuning

Transfer learning and fine-tuning are powerful TensorFlow custom object detection methods.

Transfer learning applies large-scale dataset-trained model expertise to new tasks. Custom object recognition relies on pre-trained models like EfficientDet, SSD, or Faster R-CNN. These models can extract meaningful object representations from COCO-trained generic features.

Transfer learning helps overcome bespoke object detection dataset restrictions. The network can use pre-trained models to discover patterns and accurately recognise items across object categories. This significantly decreases training data, saving time and resources.

Transfer learning requires fine-tuning the pre-trained model to the object detection task. Fine-tuning trains the pre-trained model on the custom dataset to refine its features and adapt to the target domain. Fine-tuning updates model parameters without losing pre-training knowledge.

Transfer and fine-tuning have several benefits. They eliminate the requirement for expensive, labelled bespoke datasets. Developers can start custom object identification projects using pre-trained models and obtain excellent results with minimal data.

Transfer learning and fine-tuning enable cross-domain knowledge transfer. Pre-trained models using varied datasets can extract features from a wide range of object types. Fine-tuning on a bespoke dataset helps the model detect target domain objects more accurately.

Transfer learning and fine-tuning accelerate prototyping and deployment. Researchers and developers may swiftly adapt TensorFlow's enormous library of pre-trained models to object identification tasks. This accelerates development and iterations, encouraging creativity and experimentation.

Transfer learning and fine-tuning have constraints. If the domain or object classes varies greatly, pre-trained models may not capture all the nuances and intricacies of target object detection. The models may have biases or restrictions that require substantial fine-tuning or architectural changes to overcome.

Fine-tuning also involves hyperparameters including learning rates, regularization methods, and optimization algorithms. Overfitting occurs when the model performs well on training data but fails to generalize to new data due to improper fine-tuning.

Transfer learning and fine-tuning improve TensorFlow custom object detection. Developers can get excellent results with less tagged data and quicker development cycles by adapting pre-trained models to specific applications. However, domain mismatch and the requirement for fine-tuning to optimize performance must be considered. Transfer learning and fine-tuning can be used to develop robust custom object detection models by balancing pros and cons.

2.4 Pose Estimation in Robotics

2.4.1 Marker-Based Pose Estimation Techniques

Computer vision applications use pose estimation to estimate 3D object and human body positions and orientations. Marker-based methods are popular due of their effectiveness and adaptability.

Encoded markers are a typical marker-based method. Cameras may easily identify these indicators, estimating pose. Other marker-based methods are worth mentioning.

Marker-based methods use square markers with encoded patterns and circular regions for detection and pose estimation. These approaches can handle occlusions and lighting fluctuations in real-world augmented reality and object tracking applications. However, complicated settings or fast-moving objects may impair marker detection and tracking.

NFT is another marker-based method. Tracking natural elements like corners or edges in an image series allows for better flexibility and adaptation. Markers can be used for pose initialization and NFT-based pose tracking to supplement marker-based methods. Marker-less NFT is flexible, but limited texture or unclear features may reduce posture estimate accuracy.

The marker-based posture estimate methods have many benefits. These methods are useful for many applications since they are accurate and real-time. They also tolerate lighting, occlusions, and depth alterations. Marker-based techniques are used in robotics, AR/VR, animation, motion capture, and more.

Marker-based methods have limitations. Tracked items or subjects may be limited by explicit markers. Markers may also increase detection and tracking algorithm computation. Marker-based systems may struggle with fast or unpredictable movements, restricted marker visibility, or occlusions.

2.4.2 ArUco Marker System

ArUco Marker System pose estimation is used in computer vision and augmented reality. ArUco markers, specially designed square markers, are recognized and tracked by a camera to reveal the camera's pose (position and orientation) in relation to these markers.

The ArUco Marker System relies on rigorous marker design. ArUco markers are square with alternating black and white squares. Computer vision algorithms easily discover and identify these patterns, demonstrating their inventiveness.

The camera's ArUco markers start pose estimation. Using thresholding, edge detection, and contour analysis, this step locates the marker's corners and determines its exact location in the image.

After detection, the algorithm meticulously identifies each ArUco marker and ID. These IDs help the algorithm identify many markers in the scene by being embedded in the patterns.

Pose estimation begins with marker detection. The system accurately computes the camera's pose by using the markers' 3D geometry, including their size and shape, and the camera's intrinsic parameters, such as calibration.

Coordinate system transformation is important to the ArUco Marker System. The system changes the marker's pose using information like its position in the global picture or its interactions with other markers. This transformative leap translates the camera's pose into real-world coordinates, providing a complete grasp of its position and orientation in the greater surroundings.

The ArUco Marker System has many benefits, yet also has drawbacks. Its simplicity and efficiency make it adaptable to research and industrial applications. Its precision and durability make posture estimation reliable. Like any system, it has drawbacks. In obscured or poorly lighted markers, finding and tracking them is difficult. Visual ambiguity in the marker design may cause mis-identification and posture estimation mistakes.

The ArUco Marker System excels in pose estimation and augmented reality. Its voyage includes carefully constructed markers, detection, and recognition, pose estimate computation, and the transformation into real-world coordinates. The approach is simple, efficient, accurate, and robust, although marker detection and ambiguity are limits. The ArUco Marker System remains useful in computer vision applications by balancing pros and cons.

2.4.3 Calibration of Marker System

Motion capture and tracking in animation, biomechanics, virtual reality, and sports analysis require marker system calibration. This overview covers marker placement, reference measurement, calibration, and validation. Understanding calibration's benefits and drawbacks helps us get trustworthy findings.

Marker placement begins marker system calibration. Marking individuals or objects strategically is essential for capturing and tracking motions. The program specifies each marker's position. Markers should be appropriately placed on objects or anatomical landmarks to accurately capture motions.

Reference measurement creates a stable coordinate system. Points or things are measured using reference points or equipment. These reference measurements ensure accurate and real-world-aligned tracking results. However, reference instrument imperfections and measurement mistakes must be considered.

Calibration procedures vary by marker system. Wand calibration calculates transformation matrices and aligns the coordinate system by moving a known-sized object through the capture volume. Multi-view calibration captures marker positions from many camera views concurrently. The technique calculates marker 3D positions by triangulating these positions. Static posture calibration captures marker positions from different angles while the subject or object remains still. These approaches increase tracking accuracy by aligning and calibrating. Each method may add complexity and take longer to execute.

Optimizing calibration results after the initial calibration is common. This approach minimizes calibration parameter-based marker position inaccuracies. Bundle adjustment and least-squares optimization improve calibration accuracy. Optimization strategies are computationally demanding and may slow calibration.

Marker system accuracy depends on validation and fine-tuning. To verify tracked motions, compare them to ground truth data or conduct validation trials. This validation procedure identifies differences and allows fine-tuning to improve calibration accuracy. Validation and fine-tuning may take time and money, complicating the calibration workflow.

2.5 Kinematics for Robotic Arm Pose Estimation

2.5.1 Introduction to Kinematics

Kinematics, a fascinating branch of physics, studies object motion without considering forces. It analyses and describes object movement using location, velocity, and acceleration. Kinematics explores motion's "how's" rather than "whys."

Kinematics mathematically represents motion. It involves displacement, velocity, acceleration, and the rate at which an object's location changes. Kinematics quantifies and visualizes moving things using equations and graphs.

Studying kinematics has broad scientific applications. Kinematics is essential to astronomy, engineering, and other fields. It helps scientists and engineers forecast and create systems.

Robotics and animation depend on kinematics. Engineers and animators may build believable robot and CGI character movements by knowing kinematics. This application has transformed entertainment, gaming, and automation, improving daily life and technology.

Kinematics has constraints. The main problem is that it solely evaluates object motion without addressing causes. Kinematics tells us things' position, velocity, and acceleration, but not why. Dynamic forces are needed to fully understand motion.

Kinematics also requires idealized conditions. Air resistance, friction, and external influences can dramatically affect object velocity. Pure kinematics ignores this complexity, simplifying real-world occurrences. To make reliable predictions and assessments, kinematic principles must account for these elements.

Kinematics describes and analyses object motion. Its mathematical underpinning helps explain scientific and technological applications. Kinematics alone does not describe motion and may oversimplify real-world events. Researchers, engineers, and fans can use kinematics effectively while evaluating its pros and downsides.

2.5.2 Forward Kinematics

Forward Kinematics is essential to understanding and predicting robot end-effector location and orientation. Starting from the robot's base, this elegant method sequentially applies geometric alterations along the kinematic chain to record robotic limb movement.

Forward Kinematics uses mathematical equations and algorithms to calculate a robot's end-effector's 3D position and orientation from its connection angles and lengths. Forward Kinematics calculates the end-effector's pose relative to the robot's base frame using rotation matrices, translation vectors, and homogeneous transformations. Robotic arm control, motion planning, virtual simulations, and computer-aided design benefit from this data.

Forward Kinematics is easy and intuitive. Beginners and experts can understand the concept. Engineers and academics can examine and observe robot movement without understanding dynamics or control methods. Forward Kinematics helps design and optimize robot trajectories for pick-and-place, assembly line automation, and surgical robotics by precisely anticipating end-effector position and orientation.

Forward Kinematics does have downsides. The simplified kinematic model ignores joint friction, backlash, and dynamic effects. In fast or dynamic situations, this oversimplification may cause end-effector position discrepancies. Forward Kinematics does not reveal the robot's internal forces, torques, or joint restrictions, limiting its use in force control and collision avoidance tasks.

Forward Kinematics only provides one joint angle solution. Forward Kinematics cannot distinguish joint configurations that produce the same end-effector posture. Inverse kinematics, which determines joint angles to position an end-effector, might be affected by this uncertainty.

In conclusion, Forward Kinematics predicts a robot's end-effector position and orientation. It's easy to use for robot motion analysis and trajectory design. When applying Forward Kinematics to increasingly complicated robotics jobs, the assumptions of a simpler kinematic model and lack of internal stresses and torques should be examined.

2.5.3 Inverse Kinematics|

In robotics and computer graphics, inverse kinematics determines the joint configurations of a mechanism or virtual character given its end effector positions and orientations. It estimates the end effector's position from joint configurations, reversing Forward Kinematics. Animation, motion planning, robotics control, and biomechanics analysis use inverse kinematics.

Inverse Kinematics makes controlling complex machinery and character motions easier. The underlying technique can efficiently compute the joint angles needed to pose an end effector, such as a robot's gripper or a character's hand, by describing its position and orientation. Operators or animators can focus on high-level goals rather than manually tweaking joint angles, allowing intuitive control.

Inverse Kinematics has drawbacks, though. Multiple or no solutions for specific combinations is a downside. The inverse problem may have numerous valid joint configurations depending on mechanism complexity and desired position, making it difficult to identify the best solution. Ambiguity can cause unnatural movements or unanticipated accidents, which require limits or optimization to mitigate.

Inverse Kinematics is computationally expensive. Inverse problems need intricate mathematical computations and iterative procedures. Computational overhead can impact system performance in real-time applications like robotics control and interactive animation. Algorithms and approximation methods have been developed, however finding a balance between accuracy and processing efficiency is still a study topic.

In conclusion, Inverse Kinematics helps control and animate complex machinery and virtual figures. It has easier control, natural-looking movements, and many applications. However, multiple solutions and computing cost should be considered, and further methods may be needed to overcome these restrictions.

2.5.4 DH Parameters

Robotics and mechanical engineering employ DH parameters. They allow precise analysis and control of robotic manipulator kinematics. This balanced literature summary will discuss DH parameters' pros and cons.

DH parameters simplify robot representation. DH parameters create a hierarchical framework that captures joint interactions by assigning parameters to each manipulator joint. This hierarchical representation simplifies robot kinematics modelling and analysis, making forward and inverse computations easier.

DH settings also enable robot modularity and reuse. They simplify interchange and integration by standardizing robot linkages. Modularity simplifies developing and building complicated robotic systems by letting engineers focus on individual components without thinking about the whole system.

The robot's Jacobian matrix, which defines joint and end-effector velocities, can be calculated using DH parameters. Trajectory planning, motion control, and obstacle avoidance require this. DH parameters easily compute the Jacobian matrix, allowing accurate robot movement control.

DH parameters have restrictions. Fixed coordinate frames can misrepresent the physical robot. Non-standard robot designs and flexible component robots show this restriction. In such cases, DH parameter representation adjustments may complicate analysis.

DH characteristics are susceptible to manufacturing tolerances and joint clearance. Small joint parameter or link length discrepancies might cause large kinematics problems. This sensitivity may affect the robot's accuracy and require calibration or adjustment.

DH parameters represent and analyse robotic manipulator kinematics. They provide precise control, modularity, and system simplification. However, fixed coordinate frames and manufacturing tolerances are essential considerations. Engineers can use DH parameters in robotics by understanding their pros and cons.

2.5.5 Real-time Pose Estimation

Real-time pose estimation is a computer vision technique that uses visual input to estimate an item or person's position and orientation. Robotics, augmented reality, human-computer interface, and surveillance systems have used this technology.

Real-time pose estimate benefits. First, it uses deep learning for accurate, real-time tracking. Time-sensitive applications benefit from real-time interaction, analysis, and decision-making.

AR applications require real-time pose estimation. It improves AR by aligning virtual items with real-world environments. Users can interact with virtual items that blend into the real environment.

Real-time pose estimation makes human-computer interaction natural and intuitive. It tracks human positions to enable gesture detection for interfaces, gaming systems, and virtual reality environments. This enhances usability.

Robotics and autonomous systems use real-time pose estimation. It aids robot perception and interaction. Robots can utilize pose estimation to find items, navigate complex settings, and manipulate objects precisely.

Real-time pose estimation has drawbacks. First, it demands powerful hardware and efficient implementations for calculation. On resource-constrained devices or in real-time scenarios, processing needs may limit such solutions.

Real-time pose estimate is also affected by lighting, occlusions, and clutter. Pose estimation may lose accuracy and robustness in difficult conditions, lowering system performance.

Deep learning-based pose estimation approaches require training data and are difficult to generalize. These approaches need lots of labelled training data to work. Obtaining and annotating such datasets takes time and money. Additionally, trained models may struggle to adapt to new settings or positions, limiting their application.

Real-time posture estimation algorithms track and analyse human poses, raising privacy and ethics concerns. To protect privacy and prevent misuse, sensitive information must be handled carefully, and technology used properly.

Finally, real-time posture estimation has several applications. It improves tracking, augmented reality, human-computer interface, robotics, and autonomous systems. To properly utilize real-time posture estimation, computing complexity, environmental constraints, training data requirements, and privacy concerns must be addressed. Understanding its pros and cons helps us create and deploy this technology.

3. Research Methodology

3.1 Introduction

This chapter describes the system study and implementation approach. Furthermore, a comprehensive account of the research methodology employed in this thesis is presented, encompassing various techniques such as 3D CAD design for marker holders, development of vision scripts, establishing AI models, and implementation of kinematic principles.

3.2 Research Methodology and Techniques

To investigate an inquiry, the research approach and technique used must be carefully considered. Two main approaches commonly employed in studies are quantitative and qualitative research. Quantitative research involves analysing numerical data using mathematical methods, such as statistics, to provide explanations, while qualitative research focuses on understanding underlying reasons and gaining insights into the context of a problem. For this study, a qualitative approach was chosen, aiming to gather knowledge, and understanding others' opinions and assumptions regarding computer vision.

3.3 CAD Design

This research has been conducted with the aim of designing a marker holder customized for the mentor robotic arm supplied by the school, intending to provide enhanced functionality and accuracy in robotic positioning and movement.

3.3.1 Marker Placement and Design

In the initial design phase, markers were placed on all four joint sides (front, back, left, and right) of the robotic arm. This configuration was selected to provide the most comprehensive positional tracking and enable full orientation visibility to the tracking system from any viewpoint.

The markers were designed to fit perfectly on the robotic arm joints, considering the shape, size, and the arm's mechanical constraints. These markers function as reference points for the CAD design software, aiding in accurate modelling of the robotic arm.

3.3.2 End Effector Marker Placement

The end effector of the robotic arm was also equipped with markers to enhance precision. Five markers were placed at the front, back, left, right, and top of the end effector. This enhanced placement was intended to increase tracking accuracy as the pose of the end effector is crucial for inverse kinematics.

3.4 Vision Scripts

The methodology for this research primarily involves the creation and application of various scripts to generate the markers, perform camera calibration, extract necessary data from ArUco markers, perform kinematic calculations, build object detection models, and manage the camera feed. The following sections detail each script's function.

3.4.1 Generating Markers

The first part of our methodology involves generating ArUco markers. We developed a script that produces these markers with specific sizes, IDs, and dictionaries. This script provides the customization necessary to create unique markers for distinct applications and various environmental conditions, ensuring the robustness and flexibility of our system.

3.4.2 Camera Calibration

Camera calibration is crucial in computer vision applications to remove lens distortion and estimate perspective geometry. Our research utilizes the ChArUco board calibration method for this purpose. A script was written to perform camera calibration using a ChArUco board, offering higher precision and accuracy than traditional methods and being suitable even for low-resolution cameras.

3.4.3 ArUco Marker Data Extraction

Once the camera calibration is complete, we employ another script that captures rotational and translational matrices from the ArUco markers. This script processes the camera feed in real-time, detecting and decoding the markers, and then computes the rotational and translational matrices. These matrices are essential for determining the markers' spatial orientation and location, a critical step in tracking and analysing movements.

3.4.4 Object Detection using TensorFlow

In addition to tracking markers, our system can detect custom objects. A script was written to build a detection model using TensorFlow, a popular machine learning framework. This model allows for real-time object detection, further extending the functionality and applicability of our system.

3.4.5 Camera Feed Management

To improve performance, we developed a script that forwards the client's camera feed to a server-based computer vision system. After extensive testing, we found that using a Raspberry Pi or a hotspot connection yielded insufficient results due to high latency. Therefore, we opted for this server-based approach, which significantly reduced latency and increased the frame rate to approximately 8-10 fps, compared to the previous 0.2 fps.

3.4.6 Integration and Final Output

Finally, a script was developed to integrate the key parts of the other scripts to produce the final output. This script merges all the separate components into a cohesive system, presenting a webcam view tracking the markers in real-time, showing a 3D axis, and displaying a separate window with a skeleton view of the robotic arm's movements. This integrated approach offers a comprehensive visualization of the system's operations, greatly aiding analysis and troubleshooting.

This methodology, built on a suite of custom scripts, provides a robust and versatile computer vision system capable of accurately tracking and controlling a robotic arm while offering real-time object detection.

3.5 TensorFlow Custom Tracking

The research methodology used in this study involves the utilization of the TensorFlow API, dataset preparation, model training, and model inference for the purpose of building a custom object detection system.

3.5.1 TensorFlow API and Environment Setup

The first phase of the study involves setting up the TensorFlow Object Detection API environment. The TensorFlow API is a powerful tool for object detection, offering extensive features and utilities that can be utilized to build an object detection model. To build the environment, Python is first installed, followed by TensorFlow and other necessary dependencies such as Protobuf, pillow, lxml, tf_slim, Jupyter, Matplotlib, Cython, contextlib2, cocoapi. After these installations, the TensorFlow model’s repository is cloned from GitHub, and the Protobuf libraries are compiled. PATH is then set, and Python path environment variable is configured to access the scripts and modules inside the 'models' directory.

3.5.2 Dataset Preparation

The second phase of the study involves preparing the dataset for custom object detection. This process follows a four-step pipeline.

3.5.2.1 Data Collection

Photos are collected from the environment where the objects to be detected are located. It is important that these photos are representative of the diverse scenarios in which the system will operate.

3.5.2.2 Data Annotation

In the data annotation phase, the region in each photo where the object of interest appears is marked. This can be accomplished using various tools such as labelImg, an open-source graphical image annotation tool.

3.5.2.3 Dataset Split

Once the images are annotated, the dataset is split into a training set and a testing set. In this study, we follow a 70:30 split where 70% of the data is used for training and 30% is kept for testing. This random split ensures the generalization of the model on unseen data.

3.5.2.4 File Creation

Following the dataset split, a .csv file for each set is created to store the metadata of the images and the annotations. Moreover, a label_map.pbtxt file is created which holds the mapping between class names and class IDs. Furthermore, TensorFlow record files (.tfrecord) are created which are efficient to use and easy for TensorFlow to read.

3.5.3 Transfer Learning

The third phase involves transfer learning, where a pre-trained model is taken as a starting point. This model has been trained on a large dataset and has gained the ability to extract features from images. This model is then fine-tuned by continuing the training on our custom dataset. The architecture chosen for transfer learning will depend on factors such as computational resources and the desired balance between speed and accuracy.

3.5.4 Training the Model

The custom object detection model is trained in this phase. A configuration file is set up to specify the model architecture, the path for the training and testing data, the path for the label map, the number of classes, and other parameters. Training the model consists of feeding it with the training data, adjusting its weights and biases based on the output, and iterating this process until an optimal set of weights and biases are found.

3.5.5 Model Inference

After the model is trained, it is then put to the test to make predictions on unseen data. In this inference phase, the model's accuracy in detecting objects in new images is assessed. Scripts are written to use the trained model to detect objects in new images, videos, or in real-time scenarios, depending on the requirement. The model's performance can be evaluated using metrics such as precision, recall, and mean average precision.

In summary, this research methodology includes several steps, each of which contributes to the development of a custom object detection system using TensorFlow. By leveraging transfer learning, this process can be significantly expedited while maintaining high levels of accuracy.

3.6 Kinematic Interpretation

In this research, our focus is on kinematic interpretation of a robotic arm using acquired data from ArUco Markers. ArUco markers are a type of augmented reality marker that consists of a square with a binary coding pattern. The markers are designed to be easily detected and provide a reference for the pose of the robotic arm. The kinematic interpretation process involves the use of inverse and forward kinematics along with Denavit-Hartenberg (DH) parameters for digital reconstruction of the robotic arm's skeleton view in real time. The methodology is described in detail below.

3.6.1 Inverse Kinematics

Inverse kinematics is a fundamental step in understanding the working of a robotic system, in which the joint parameters are calculated from the known end effector's position and orientation. We utilize the data acquired from ArUco Markers as the input to our inverse kinematics algorithm.

Let's assume we know the desired position of the end effector represented by a coordinate vector P = (Px,Py,Pz). The goal is to determine the joint angles θ = (θ1,θ2,...θn) that will place the end effector at this desired position. Given the complexity of the inverse kinematics problem, a numerical solution such as the Jacobian method might be used to iteratively compute the joint angles.

3.6.2 Forward Kinematics

Following inverse kinematics, the obtained joint angles are used to perform forward kinematics. This process involves the use of joint angles to calculate the position and orientation of the end effector or any other point of interest on the robotic arm. The forward kinematics equations for a standard robotic arm are typically a set of transformation matrices that translate and rotate the links from the base of the robot to the end effector.

In the forward kinematics process, the joint parameters (θ) are input into a function F(θ), which computes the position of the end effector in the workspace, given as F(θ) = (Fx,Fy,Fz). This mapping from joint space to task space confirms the results obtained from the inverse kinematics process.

3.6.3 Denavit-Hartenberg (DH) Parameters

DH parameters are a systematic method to describe the geometry of a robotic arm. They are four parameters that define the relative distances and angles between adjacent links in the robotic arm. Using these parameters, transformation matrices are derived which enable the computation of the position and orientation of each joint of the robot arm in the workspace.

To create the skeleton view of the robotic arm digitally in real time, a model of the robotic arm is constructed using the DH parameters. The positions and orientations of the joints calculated using forward kinematics are then used to animate the digital model, creating a real-time digital representation of the physical robotic arm.

The entire process from extracting the joint angles using inverse kinematics, then confirming those using forward kinematics and finally building the digital skeleton view using DH parameters provides an accurate and real-time digital representation of a robotic arm. The effectiveness and accuracy of this methodology will be verified through rigorous experiments and validations in the subsequent sections of this research.

3.7 Block Diagram of System

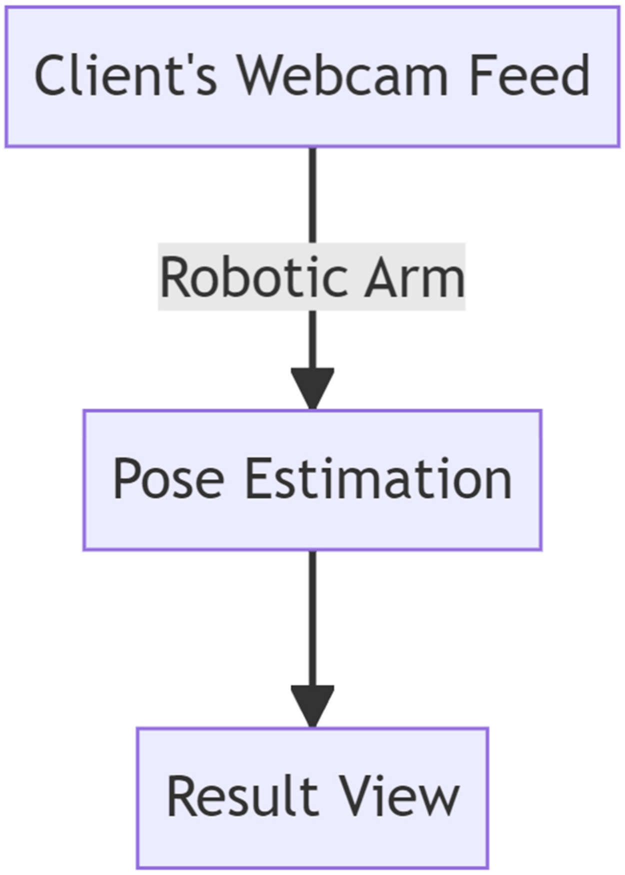


Figure 1: Block Diagram of System

3.8 Flowcharts

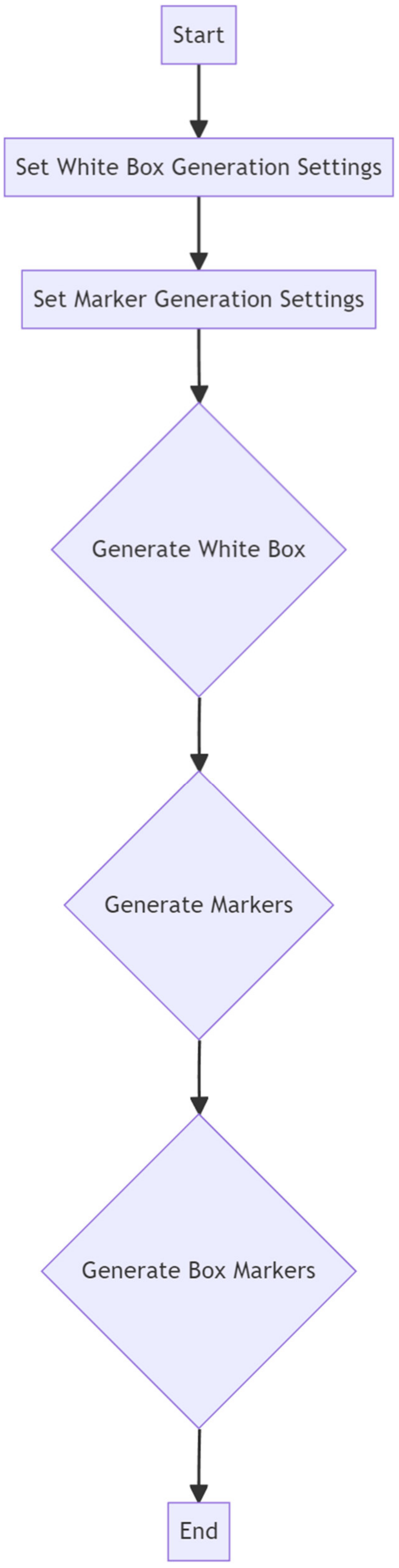

Start
Set White Box Generation Settings
Set Marker Generation Settings
Generate White Box
Generate Markers
Generate Box Markers
End

Figure 2: Marker Generation for Pose Estimation

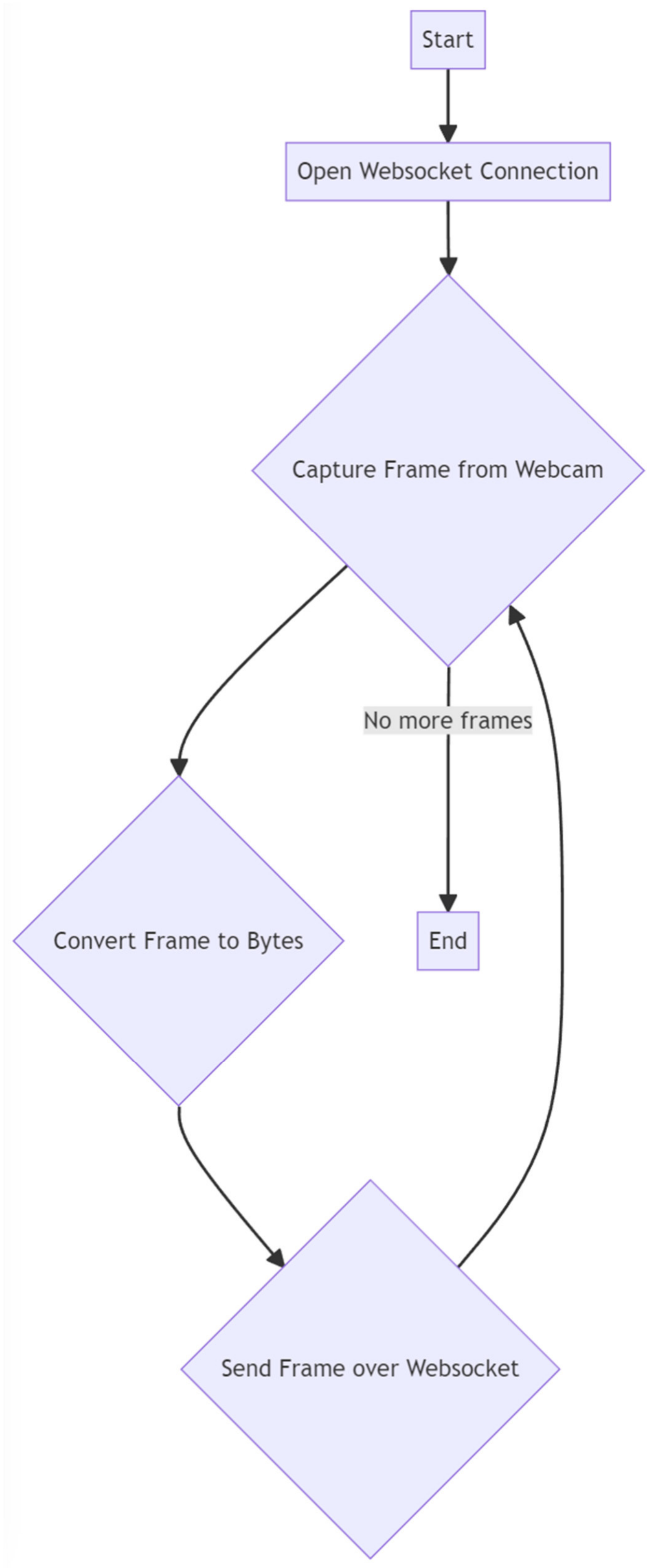
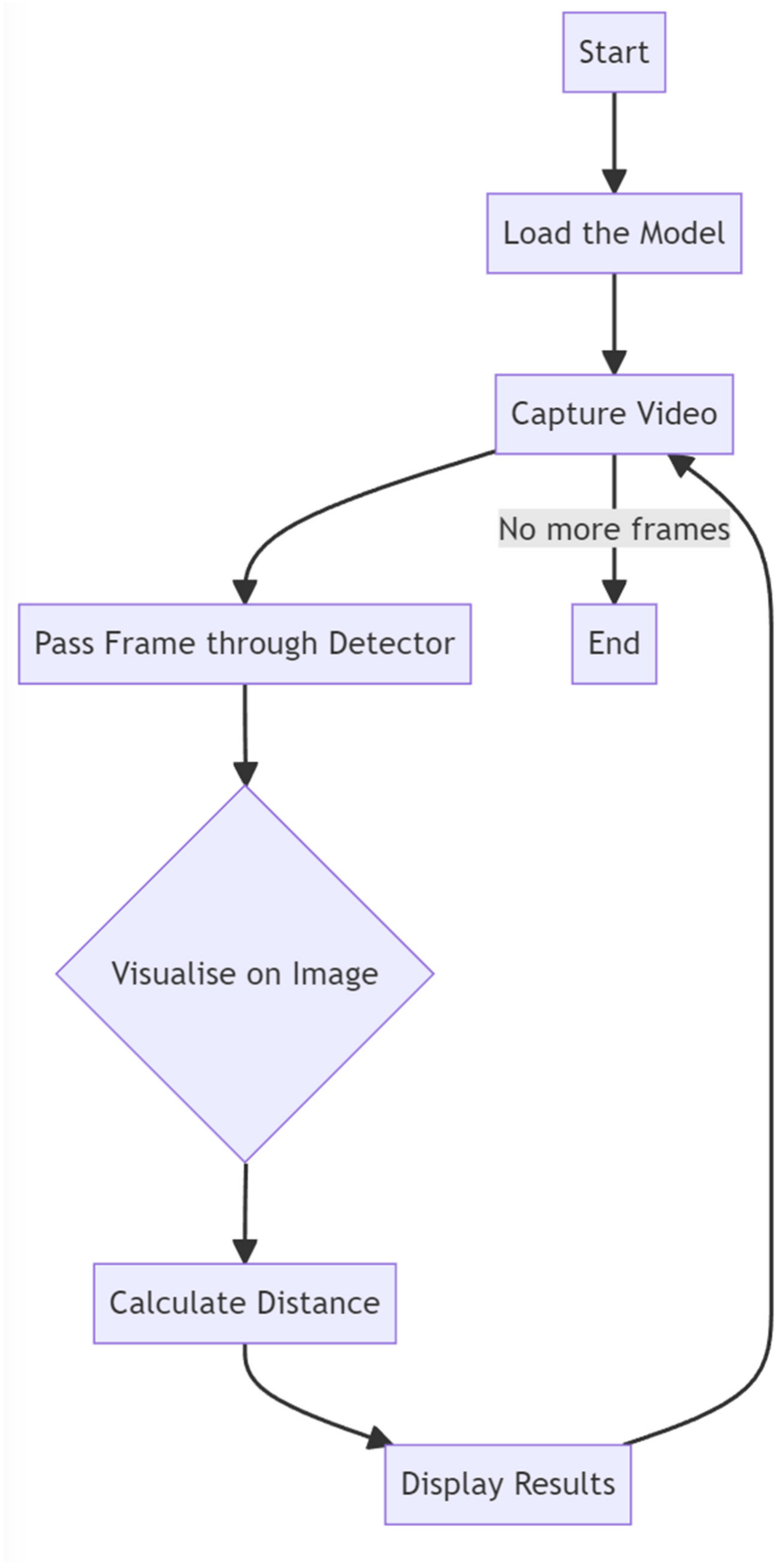

Start
Open Websocket Connection
Capture Frame from Webcam
No more frames
Convert Frame to Bytes
End
Send Frame over Websocket

Figure 3: Receive Webcam Feed Code

Figure 4: Tracking Code



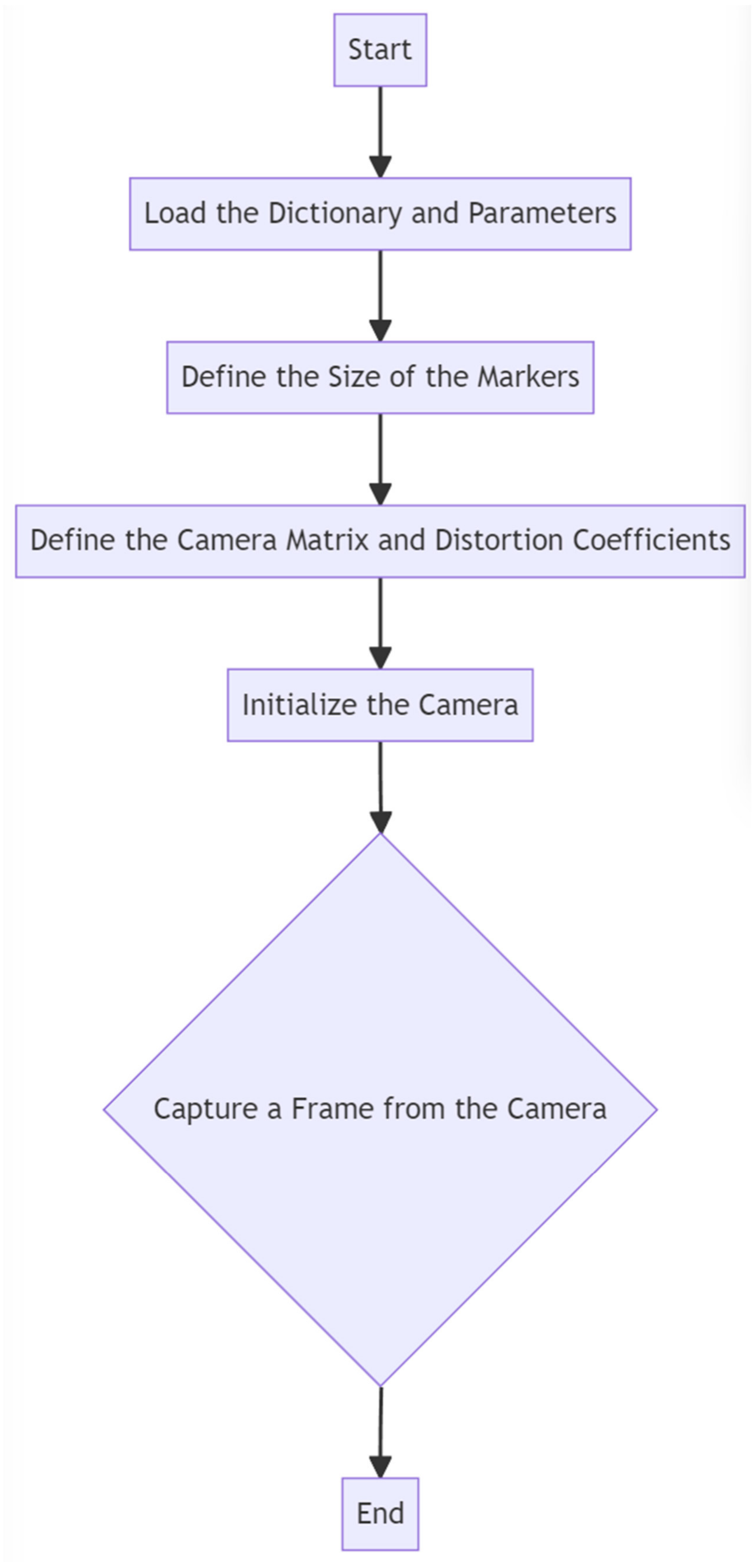

Start
Load the Dictionary and Parameters
Define the Size of the Markers
Define the Camera Matrix and Distortion Coefficients
Initialize the Camera
Capture a Frame from the Camera
End

Figure 5: Pose Data Extraction Code

4. Analysis of Results and Discussion

4.1 Introduction

In this section, I discuss the analysis of results obtained from the tracking and pose estimation system developed for a robotic system using computer vision techniques. I acknowledge the limitations due to time constraints and highlight areas where further detailed testing could be conducted if more time was available.

4.2 Testing and Discussion

4.2.1 Testing of CAD Designs

Before proceeding to full-scale production, I conducted testing on the CAD designs to ensure their usefulness, viability, and accuracy. The testing process involved three key stages: test fitting, iterative design and printing, and final design detailing.

4.2.1.1 Test Fitting

The initial step in the testing process was test fitting, where I printed the necessary dimensions of the CAD design to verify its fit. This stage is crucial for prototyping as it allows for the detection of potential dimensional mismatches that could affect the design's functionality. By utilizing 3D printing, I replicated the design and obtained an accurate representation of the physical object. This process provided insights into how the final product would fit in its intended environment. Test fitting helped identify any discrepancies in the design early on, preventing costly mistakes or revisions in later stages.

4.2.1.2 Iterative Design and Printing

After the test fitting stage, the next step involved refining the design based on the feedback and observations gathered from the initial test. This iterative refinement process included editing the CAD file and printing it again to verify improvements in fit and form. This iterative cycle of design, testing, and feedback was crucial to gradually enhance the precision of the design. It ensured that the final product would seamlessly fit into its intended application.

4.2.1.3 Final Design Detailing

Once the prototype met all the requirements regarding fit and form, the final step was to incorporate detailed features into the design. This stage required meticulous attention to detail, ensuring that all necessary elements were accurately represented in the CAD design before finalizing it for printing. Details such as textures, material finishes, or specific component attachments were added at this stage to enhance the functionality and aesthetic appeal of the design. The finalized design was then printed and tested one last time to verify that all added details were correctly incorporated without compromising the fitting and functionality confirmed in previous stages.

4.2.2 Testing of Vision Scripts

Throughout the experiments, I conducted multiple test cases to validate the functionality and reliability of the vision scripts. The testing primarily focused on three key areas: generating markers suitable for the robot arm, camera calibration using a ChArUco board, and extrapolating the pose of each marker relative to the camera. The following discussion provides insights into the testing approach, results, and the significance of the findings.

4.2.2.1 Generation of Markers

Generating markers of optimal size for the robot arm was crucial to ensure accurate object detection and tracking. The testing process involved tweaking the marker size to determine the optimal range that allows for maximum detection accuracy. The results demonstrated a clear correlation between the size of the markers and their detection rate. Markers that were too small often failed to be accurately detected, while excessively large markers posed issues due to the limited field of view of the camera. After multiple iterations, an optimal marker size was determined that consistently delivered high detection rates while ensuring compatibility with the operational constraints of the robot arm.

4.2.2.2 Camera Calibration with ChArUco Board

Camera calibration plays a significant role in obtaining accurate measurements from 2D images. For the tests, a ChArUco board was utilized as it provides an efficient and robust method for calibration, combining the benefits of ArUco markers and a checkerboard pattern. The size of the markers and the ChArUco board were adjusted accordingly to maintain the relative size ratio between them. Precise calibration was found to be dependent on this ratio. If the markers were disproportionately large compared to the ChArUco board, calibration errors were introduced.

Upon ensuring the appropriate size of the ChArUco board, the camera calibration significantly improved, leading to more accurate pose estimation. The calibrated camera parameters also enhanced the overall accuracy of the tracking algorithm, enabling more precise movements of the robotic arm.

4.2.2.3 Extrapolation of Marker Poses

Accurately extrapolating the pose of each marker relative to the camera was another crucial aspect of the testing process. The scripts had to compute the position and orientation of the markers accurately from the camera's perspective. This extrapolation is fundamental for the robot arm's accurate and efficient manipulation of objects.

Rigorous testing confirmed that the pose extrapolation was mostly accurate, allowing the robot arm to successfully recognize and interact with the markers. However, there were occasional instances where the pose estimation was slightly off, resulting in misalignment of the robot arm with the target marker. Further investigation revealed that these inaccuracies were due to minor calibration errors and suboptimal lighting conditions, which will be addressed in future iterations.

In conclusion, the testing phase played a vital role in refining the vision scripts and ensuring their reliability. The key findings emphasize the importance of optimal marker size, precise camera calibration, and accurate pose extrapolation for effective robotic manipulation. Future work will focus on implementing improvements to address the identified issues and further enhance the vision scripts.

4.2.3 Testing of TensorFlow Tracking

During this study, extensive tests were conducted to evaluate the implementation of the TensorFlow-based tracking system, with particular emphasis on the TensorFlow Lite version. The primary goal was to assess the real-world applicability of the model across various hardware environments.

Initially, high-end Raspberry Pi systems were used for the tests, utilizing the TensorFlow Lite framework. TensorFlow Lite was chosen due to its lighter computational footprint, making it potentially suitable for low-resource environments. Unfortunately, the results were disappointing as the performance was deemed inadequate. This could be attributed to the resource-constrained nature of the Raspberry Pi, despite it being the most powerful variant. It became evident that TensorFlow Lite, although optimized for devices with lower computational power, could not perform satisfactorily within this environment.

Given these limitations, the focus shifted towards an alternative approach to improve performance. The hypothesis was that a Windows workstation, with significantly more processing power, would be able to handle the required computations more efficiently. The Windows workstation was expected to outperform the Raspberry Pi by approximately 50 times, considering its superior hardware specifications. To test this hypothesis, a pipeline was implemented to transmit the client's webcam feed directly to the Windows workstation for processing.

As expected, the results were promising. The Windows workstation demonstrated significantly improved performance compared to the Raspberry Pi, validating the hypothesis. However, this approach also introduced its own set of challenges, primarily related to network constraints.

When using the setup over a hotspot, a certain degree of degradation in the results was observed. This issue was not surprising since hotspot connections often do not provide the stable, high-speed data transfer required for such data-intensive tasks. Nevertheless, even with the limitations imposed by hotspot usage, the results were still better than those obtained with the Raspberry Pi setup.

In summary, this testing phase highlighted the need for powerful computational resources for the successful implementation of TensorFlow-based tracking. While efforts to port this functionality to lower-resource systems using TensorFlow Lite may seem promising, the tests revealed the challenges associated with this approach. On the other hand, utilizing more powerful systems such as a Windows workstation, despite network-related limitations, proved to be a more effective solution. These findings emphasize the importance of appropriate hardware choices when implementing real-time, machine learning-based tracking systems.

Future work should focus on further optimizing the network and data transfer pipelines to reduce the impact of network instability on the overall system performance. This optimization would potentially make the system more robust even in hotspot-based or otherwise constrained network conditions.

4.2.4 Testing of Kinematic Principles

During the study, comprehensive testing of the Kinematic Principles was initially planned, including the testing of inverse kinematics, forward kinematics, and the application of Denavit-Hartenberg (DH) parameters. However, due to unforeseen time constraints, this segment of the experiment could not be completed as originally planned. This section provides an analysis of the consequences of this situation and outlines potential future testing methods.

The testing phase is crucial for confirming the validity of kinematic equations in a practical setting. Through practical application and testing, theoretical equations can be validated, and any discrepancies or limitations can be identified and understood.

Testing the forward kinematics equation, for example, is essential to understand how the robot's position and orientation are related to its joint parameters. Conversely, testing the inverse kinematics equation provides insights into determining the possible joint parameters for a given position and orientation.

Additionally, implementing and testing the Denavit-Hartenberg (DH) parameters would provide an understanding of how each individual joint affects the overall robot configuration. The DH parameters provide a systematic way to define the geometry of a robot, and testing these parameters is crucial for understanding the behaviour of the robot arm in different configurations.

Unfortunately, due to time constraints, the planned tests could not be carried out, which has a significant impact on the reliability and completeness of the research. Without practical validation, there is a risk of hidden complications, unforeseen behaviours, or subtle errors that may only become apparent during practical implementation. It is important to acknowledge this limitation in the current stage of the project.

The critical lesson learned from this experience is that sufficient time should be allocated for rigorous testing phases in future works. It is recommended to conduct preliminary testing early in the research process to identify potential pitfalls, inaccuracies, or points of failure in the proposed methodologies. Follow-up testing after modifications can then confirm the accuracy and robustness of the equations and methodologies.

In conclusion, despite the time constraints and the inability to complete the full suite of planned tests, the intention is to pursue this testing in future work. Testing the forward and inverse kinematics equations, as well as the application of DH parameters, remains a high priority. By adopting this approach, a more reliable and comprehensive exploration of Kinematic Principles can be provided in the future.

4.3 Achieved Results

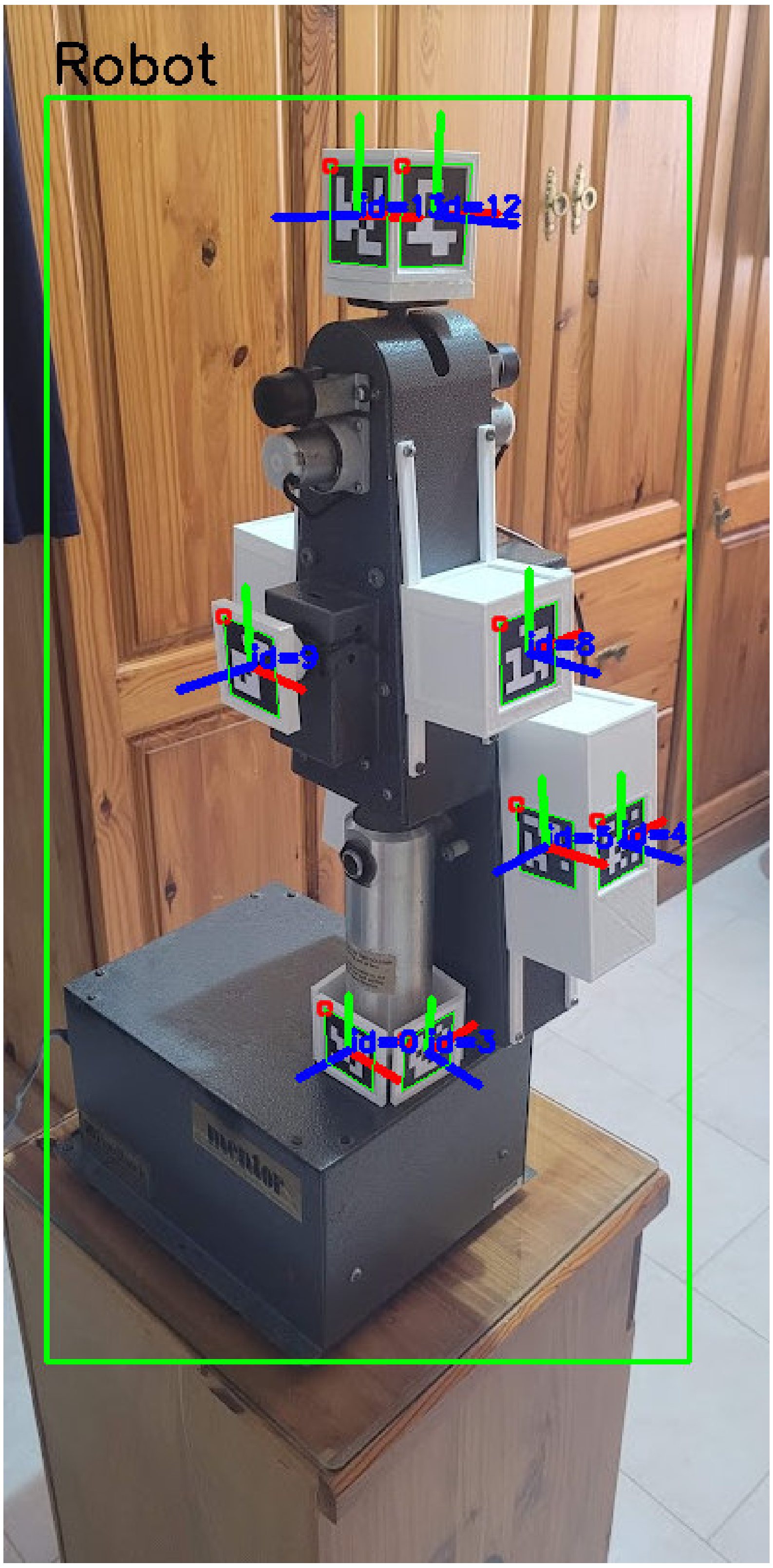


Figure 6: Achieved Results

5. Conclusions and Recommendations

5.1 Achievements and Contributions

This project has provided an important step in the advancement of tracking and pose estimation in robotic systems, with a specific emphasis on computer vision. The extraction and interpretation of pose data from markers alone has demonstrated the project's potential, and in doing so, has created a viable pathway for further investigation and development in the field.

Despite these advancements, the full scope of the project is yet to be entirely achieved. Data extrapolation processes are still a work in progress, requiring further refinement to optimize system performance. Furthermore, the robot's tracking mechanisms, although functional, require additional enhancements to meet the original project objectives fully. Notably, the lack of incorporation of kinematics for determining joint angles indicates an area of the project that requires additional attention.

5.2 Suggestions for Future Work

Given the current stage of the project, it is challenging to offer concrete recommendations for future work. The project's aim to balance precision and modularity in marker design remains an ongoing task, requiring further investigation and development.

In terms of incorporating kinematics to compute joint angles, this is a valuable addition to the system. However, as it is yet to be implemented, future recommendations in this area can only be given once this task is completed and the impact on system performance is understood.

The current focus should be on achieving the existing project objectives, particularly the implementation of kinematics for joint angle determination and the final design for the markers. This groundwork is critical for creating a more robust and precise robotic system capable of accurate pose estimation.

To summarize, while considerable progress has been made in tracking and pose estimation for robotic systems using computer vision, further work is needed to fully realize the project's objectives. As such, concrete recommendations for future work cannot be provided at this stage. Nevertheless, the work conducted thus far has laid a foundation for advancing the capabilities of such systems, which can have broad implications in the field of robotics.

References

Appendix A