electronics

MDPI

Article

Motion Prediction and Object Detection for Image-Based Visual Servoing Systems Using Deep Learning

Zhongwen Hao [1,2], Deli Zhang [1] and Barmak Honarvar Shakibaei Asli [2,*]

[1] College of Mechanical and Electrical Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[2] Centre for Life-Cycle Engineering and Management, Faculty of Engineering and Applied Sciences, Cranfield University, Cranfield, Bedfordshire MK43 0AL, UK
* Correspondence: barmak@cranfield.ac.uk

Abstract: This study primarily investigates advanced object detection and time series prediction methods in image-based visual servoing systems, aiming to capture targets better and predict the motion trajectory of robotic arms in advance, thereby enhancing the system's performance and reliability. The research first implements object detection on the VOC2007 dataset using the Detection Transformer (DETR) and achieves ideal detection scores. The particle swarm optimization algorithm and 3-5-3 polynomial interpolation methods were utilized for trajectory planning, creating a unique dataset through simulation. This dataset contains randomly generated trajectories within the workspace, fully simulating actual working conditions. Significantly, the Bidirectional Long Short-Term Memory (BILSTM) model was improved by substituting its traditional Multilayer Perceptron (MLP) components with Kolmogorov–Arnold Networks (KANs). KANs, inspired by the K-A theorem, improve the network representation ability by placing learnable activation functions on fixed node activation functions. By implementing KANs, the model enhances parameter efficiency and interpretability, thus addressing the typical challenges of MLPs, such as the high parameter count and lack of transparency. The experiments achieved favorable predictive results, indicating that the KAN not only reduces the complexity of the model but also improves learning efficiency and prediction accuracy in dynamic visual servoing environments. Finally, Gazebo software was used in ROS to model and simulate the robotic arm, verify the effectiveness of the algorithm, and achieve visual servoing.

Keywords: visual servoing; detection transformer; particle swarm optimization; bidirectional long short-term memory; Kolmogorov–Arnold network

check for updates

Citation: Hao, Z.; Zhang, D.; Honarvar Shakibaei Asli, B. Motion Prediction and Object Detection for Image-Based Visual Servoing Systems Using Deep Learning. *Electronics* **2024**, *13*, 3487. https://doi.org/10.3390/electronics13173487

Academic Editor: Dah-Jye Lee

Received: 31 July 2024
Revised: 23 August 2024
Accepted: 29 August 2024
Published: 2 September 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Visual servoing [1], a pivotal domain within robotics and automation, integrates visual feedback to control the motion of robotic manipulators or mobile robots. This technique, fundamentally rooted in the principles of control theory and computer vision, utilizes image data captured in real time from cameras to dynamically adjust the trajectory and orientation of a robot relative to its environment or specific objects within it. There are primarily two distinct approaches to visual servoing: Image-Based Visual Servoing (IBVS) and Position-Based Visual Servoing (PBVS) [2]. As shown in Figure 1, IBVS directly uses image feature errors computed between the current and desired camera views to drive the robot actuators [1] while PBVS computes the object's position in a world frame and uses this information to control the robot as shown in Figure 2. Both methods aim to minimize positional errors based on visual information, enhancing the robot's ability to interact with its surroundings with high precision and reliability. More recently, advancements in machine learning, particularly deep learning, have begun to forge a new pathway in visual servoing, where models are trained to predict control actions directly from complex and

diverse visual datasets, thus promising to revolutionize the adaptability and efficiency of visual servo systems.

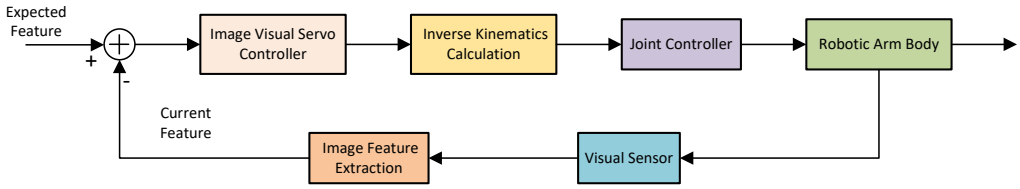


Figure 1. Image–based visual servoing flowchart.

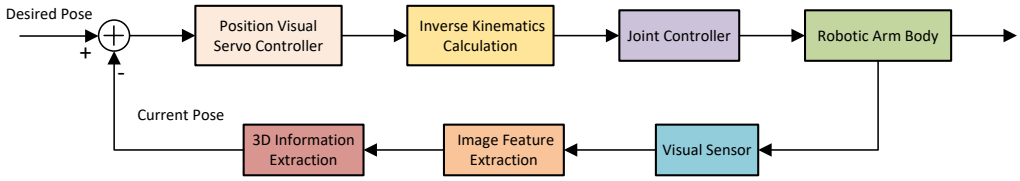


Figure 2. Position–based visual servoing flowchart.

As illustrated in Table 1, various methods of visual servoing differ significantly in their approach, advantages, and disadvantages, each tailored to specific robotic control scenarios.

Table 1. Comparison of visual servoing techniques.

Category	Description	Advantages	Disadvantages
Image-Based Visual Servoing (IBVS)	Uses image features (e.g., points, lines) directly in the control loop to minimize feature error [3].	Reduces calibration errors; direct feedback from image data.	Sensitive to large camera displacements; may lead to inefficient paths.
Position-Based Visual Servoing (PBVS)	Uses 3D models to compute camera pose relative to objects, controlling to minimize pose error [4].	Effective with accurate models; straightforward path planning.	Depends on precise camera calibration and 3D scene knowledge.
Deep-Learning-Based Visual Servoing	Employs neural networks to predict control actions from images or to enhance feature extraction and pose estimation [5].	Handles complex environments; improves robustness and generalization.	Requires significant training data and computational power; lacks transparency.

2. Related Work

2.1. Visual Servoing

Thuilot et al. [6] introduces a novel approach ensuring that the object remains in the camera's field of view throughout the robot's motion by tracking an iteratively computed trajectory. This foundational work underscores the critical importance of maintaining visual contact, which is pivotal for successful PBVS. Martinet et al. [7] build on this foundation by integrating 3D visual features into the closed robot control loop. They introduce a non-linear state feedback method that separately controls camera translation and rotation, enhancing accuracy and stability in visual servoing processes.

As the field matured, researchers explored more refined control strategies and algorithms to tackle the challenges posed by more dynamic environments and complex tasks. Dong and Zhu [8] develop a real-time vision-based pose and motion estimation algorithm using photogrammetry and an extended Kalman filter, demonstrating significant improvements in tracking and capturing non-cooperative targets in space. Their work exemplifies the shift toward leveraging sophisticated algorithms for enhanced control. Park et al. [9] further this development by introducing the concept of a 3D visible set that ensures robust

global stability under field-of-view constraints. This method not only addresses theoretical aspects but also enhances practical application by managing uncertainties effectively.

With the theoretical and control enhancements in place, the focus shifts to applying these technologies in more complex and dynamically challenging environments. Lippiello et al. [10] address PBVS in multiarm robotic cells using a hybrid camera configuration, focusing on the real-time estimation of target poses and effectively managing occlusions caused by robot links and tools. This paper showcases the practical application of PBVS in industrial settings. Parsapour et al. [11] present a sliding mode control for PBVS, utilizing robust estimators that combine an unscented Kalman observer with a Kalman filter. Their work demonstrates the effectiveness of robust control strategies in maintaining stability and accuracy under modeling uncertainties and measurement noise.

As applications grow in complexity, so does the need for more advanced control dynamics and performance enhancements. Ribeiro et al. [12] explore second-order position-based visual servoing, proposing an acceleration-based controller that substantially improves dynamic properties and tracking performance. This study represents a significant leap toward addressing the limitations of traditional velocity-based controllers.

To round off the review, it is essential to consider comparative studies that evaluate the efficacy of various approaches under a unified framework. Deng L. [13] provides a comprehensive comparison between image-based and position-based visual servoing methods, identifying performance differences and suggesting hybrid motion control and planning strategies to enhance both methods' effectiveness, particularly under large motion commands.

Recent research in IBVS demonstrates significant strides in refining the performance, reliability, and adaptability of robotic systems using sophisticated image processing and control methodologies. From high-speed interception to robust maneuvering in cluttered environments, the scope of IBVS has broadened, reflecting its increasing relevance in practical applications.

Yang et al. [14] develop a high-speed interception technique for multicopters using a strap-down camera system that effectively addresses the dynamic challenges of intercepting fast-moving drones. By integrating a delayed Kalman filter, the system compensates for sensor imaging delays relative to attitude changes, enhancing interception accuracy and response times in high-speed scenarios. Similarly, Albekairi et al. [15] propose a novel collision-free navigation method for mobile robots that utilizes a monocular sensor to manage and control the dynamics of trajectories directly in the image plane. Their approach leverages the differential flatness of the system's dynamics, ensuring precise tracking of the trajectory while avoiding collisions and maintaining the target within the camera's field of view.

Occlusion management has also been a focal point, with Zhang et al. [16] presenting a novel approach using probabilistic control barrier certificates to ensure that visual servoing tasks remain occlusion-free, integrating these with model predictive control to handle uncertainties in feature point measurements effectively. Zhu et al. [17] present a fuzzy adaptive model predictive control strategy for a six-degree-of-freedom robot manipulator. This strategy utilizes a successive linearization method to transform the nonlinear IBVS model into a linear time-invariant model at each sampling instant, optimizing the tracking of desired feature points under varying conditions.

The work of Peng et al. [18] on constrained visual servoing with a third-order sliding-mode observer addresses challenges related to time-varying disturbances and system uncertainties, showing the effectiveness of advanced observers in maintaining control precision. Moreover, Ramani et al. [19] explore the application of visual servoing in controlling the manipulator arm of teleoperated ground vehicles. Their approach enhances the position estimation of objects detected using visual features extracted from a monocular camera, demonstrating the practical utility of IBVS in teleoperation settings.

Innovations extend to hardware enhancements, where Tsai et al. [20] innovate with light field cameras in visual servoing that improve feature detection and correspondence

by leveraging light field geometry constraints. This technology enhances visual servoing performance, especially under conditions of field-of-view limitations and occlusions. McFadyen et al. [21] tackle the issue of unknown point feature correspondence in visual servoing by using a finite-time optimal control framework. This approach simplifies the feature identification process during control selection, enhancing the robustness and effectiveness of visual servoing systems for underactuated robots.

Additionally, the incorporation of deep learning in visual servoing has been explored by Harish et al. [22], who incorporate deep learning into visual servoing with their development of a deep flow-guided scene agnostic approach. By predicting optical flow using deep neural networks and integrating it with depth estimates, they significantly enhanced the accuracy and generalizability of IBVS across diverse scenes. Lastly, Machkour et al. [23] provide a comprehensive survey on both classical and deep-learning-based visual servoing systems. Their review highlights the shift toward adaptive and learning-based methods in robotics, emphasizing the broad applicability and improved performance of modern visual servoing systems.

2.2. Object Detection

In recent advancements within the field of object detection using deep learning, several studies have demonstrated significant contributions and innovations. Aref Miri Rekavandi and colleagues [24] explore the application of transformers in small object detection (SOD), highlighting their effectiveness across various contexts, such as aerial, medical, and underwater imaging. Their research reveals that transformer-based models excel beyond traditional CNN-based detectors, offering enhanced detection capabilities and introducing a compilation of underutilized large-scale datasets to set new benchmarks in SOD. Simultaneously, Juan Terven and his team [25] provide an exhaustive review of the evolution of the YOLO (You Only Look Once) architectures, from YOLOv1 to the integration with transformers in YOLOv8 and YOLO-NAS as shown in Figure 3. Their insights into architectural innovations and training methodologies underline the significance of YOLO in real-time applications like robotics and autonomous vehicles, suggesting directions for future improvements in detection systems.

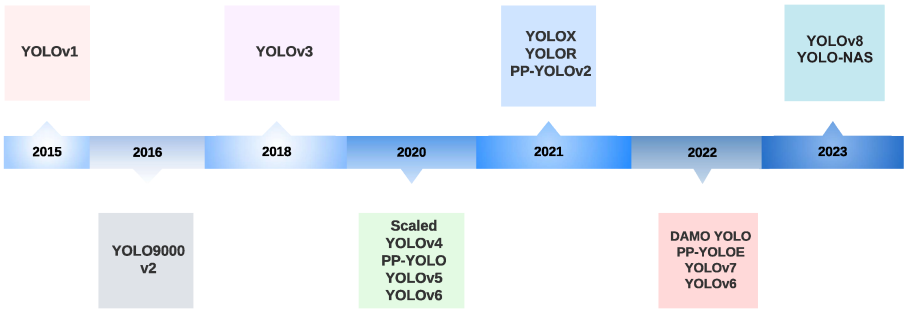


Figure 3. A timeline of YOLO versions—Juan Terven et al. [25].

Further contributions to the field include the DINO model developed by Hao Zhang and associates [26], which improves upon DETR-like models through advanced denoising and query selection techniques, significantly boosting performance on benchmarks like the COCO dataset. In parallel, Wassim El Ahmar and his group [27] introduce a novel RGB–thermal fusion technique using a sigmoid-activated gating mechanism that seamlessly integrates thermal and RGB data, enhancing object detection under diverse environmental conditions. This method not only improves detection performance but also adds minimal overhead to the processing speed, offering valuable tools and datasets for further research in multi-modal vision systems.

Additionally, Dillon Reis and colleagues [28] utilize the latest YOLOv8 architecture for the real-time detection of flying objects, showcasing the power of transfer learning from a generalized model trained on a diverse set of flying objects to a refined model tailored for

complex real-world challenges. Their approach sets new standards in terms of detection accuracy and processing speed. The UniDetector, introduced by Zhenyu Wang and his team [29], addresses universal object detection by harnessing multi-source images and heterogeneous label spaces. This model excels in zero-shot generalization across extensive vocabularies, setting new benchmarks for detecting a vast array of categories with minimal training data.

2.3. Deep Learning in Visual Servoing

Recent innovations in deep-learning-based visual servoing have revolutionized the field of robotic control, offering enhanced capabilities for a variety of complex tasks. The integration of advanced neural networks and sophisticated algorithms is transforming how robots interact with their environments, enabling more precise and adaptable operations.

Katara et al. [30] present a deep model predictive visual servoing framework called DeepMPCVS that optimizes trajectory planning and generalization in unseen environments through deep networks for optical flow predictions. This approach not only refines trajectory accuracy but also speeds up convergence in realistic indoor settings, highlighting significant performance gains over traditional methods. Li et al. [31] propose a model predictive control strategy for robotic manipulators, enhanced by reinforcement learning to handle image-based visual servoing under constraints. The control scheme of the presented Deep-Deterministic-Policy-Gradient-based Model Predictive Control MPC (DDPG-MPC) IBVS algorithm is given in Figure 4. They use a deep-deterministic-policy-gradient-based algorithm to train the control objective function, showing effective and stable control in comparative simulations. Fu et al. [32] address visual servoing challenges for UAVs under field-of-view constraints using deep reinforcement learning. Their work establishes a Markov model to dynamically adjust servo gains, improving UAV control by preventing target loss and enhancing efficiency, verified through simulations with a monocular camera. Lee, Levine, and Abbeel [33] combine learned visual features, predictive dynamics models, and reinforcement learning in their study, focusing on low-data visual servoing for target following. Their approach uses bilinear predictive models and deep features to develop robust visual servoing mechanisms that are highly adaptable to new targets and environmental variations.

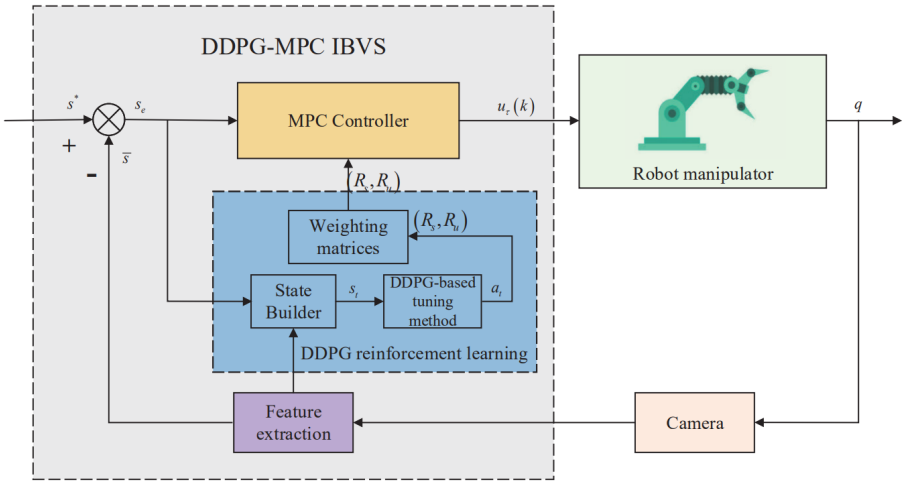


Figure 4. The control scheme of the proposed DDPG–MPC IBVS algorithm—Li et al. [31].

We now move from control systems that enhance basic operations to methods that refine the sensory input through which robots perceive their environments. Adrian et al. [34] introduce the Deep-Feature-Based Visual Servo (DFBVS) method, which utilizes deep learning for automatic feature extraction and matching, improving the generalizability of visual servoing in cluttered scenes. They also employ a render engine to synthesize target images, facilitating more accurate robotic grasping tasks. Liu and Li [35] explore the

application of Convolutional Neural Networks (CNNs) in robotic manipulation, proposing a visual servoing approach that autonomously learns to extract features and estimate the Jacobian matrix. Their two-stream CNN design demonstrates effective visual servo control for robot manipulators, verified by experimental results. He et al. [36] focus on pose prediction for robotic manipulators, employing deep learning to enhance the accuracy of pose estimation based on image similarity. They use a CNN trained with spherical projection data for position-based visual servoing, resulting in robust performance against occlusion disturbances in simulation and real tests with a UR3 manipulator.

To provide a clear comparison and better illustrate the diverse applications of Convolutional Neural Networks (CNNs) in visual servoing, Table 2 summarizes the key studies mentioned, focusing on their innovations and outcomes.

Table 2. Applications of CNNs in visual servoing.

Authors	Focus of Study	Key Innovations	Practical Applications or Outcomes
Liu and Li [35]	Robotic Manipulation	Autonomous feature extraction and Jacobian matrix estimation with two-stream CNN	Effective visual servo control for robot manipulators
He et al. [36]	Pose Prediction for Robotic Manipulators	Pose estimation enhancement using a CNN trained with spherical projection data	Robust performance against occlusion disturbances
Ribeiro et al. [12]	Robotic Grasping in Dynamic Environments	Training a CNN with the Cornell Grasping Dataset for grasp detection	High accuracy in real-time applications with a Kinova Gen3 manipulator
Lazo et al. [37]	Intraluminal Navigation in Soft Robots	Movement management using a CNN-trained approach	Demonstrated performance in anatomical phantoms
Copot et al. [38]	Position-Based Visual Servoing	Comparing deep learning models in visual servoing using a CNN	Effectiveness in precise repositioning tasks, validated in simulations and real environments

The practical applications of these advanced computational techniques are demonstrated in challenging environments. Ribeiro et al. [12] discuss the use of supervised deep learning for robotic grasping tasks in dynamic environments. They train a CNN with the Cornell Grasping Dataset to process visual data for grasp detection, achieving high accuracy in real-time applications, and showcase the controller's precision with a Kinova Gen3 robotic manipulator. Lazo et al. [37] detail a solution for intraluminal navigation using deep-learning-based visual servoing in a soft robot, designed to safely navigate hollow organs. Their CNN-trained approach manages movement in constrained environments, demonstrating the robot's performance in anatomical phantoms. Abdulhafiz et al. [39] present a direct visual servoing approach for continuum robots using deep learning, eliminating the need for intermediate feature extraction steps. Their methodology significantly enhances the end-point positioning accuracy of robots, validated in various lighting and occlusion scenarios. Jin et al. [40] explore policy-based deep reinforcement learning for the visual servoing of mobile robots with visibility constraints. They develop an adaptive law to maintain feature visibility and enhance servo efficiency, confirmed through various comparative experiments. Felton et al. [41] introduce a novel visual servoing method that controls robot motion in a latent space, combining the accuracy of photometric methods with the robustness of pose-based techniques. They shape this space through metric learning, effectively minimizing differences between latent image representations. Asayesh et al. [42] propose a scalable visual servoing strategy using deep reinforcement learning and optimal control. Their hybrid approach enhances scalability and convergence rates, demonstrating remarkable performance across diverse scenes and showcasing its applicability in real-world environments with noise and occlusions. In addition, there are

general improvements to visual servoing. Copot et al. [38] compare deep learning models in position-based visual servoing, applying a CNN to a UR10 cobot for visual control tasks. Their study highlights the effectiveness of different network architectures in achieving precise repositioning tasks, validated in both simulated and real environments.

Building on the previous work in the field, this research focuses on improving the accuracy and efficiency of robotic systems through several key methodologies:

- Utilizing a DETR for object detection and tracking to enhance the accuracy and speed of visual feedback.
- Implementing PSO to optimize the trajectory of robotic manipulators and developing a unique dataset for training and evaluation.
- Integrating BILSTM and KANs to improve the predictive performance of robot movement.
- Conducting simulations in the ROS environment using Gazebo to verify the effectiveness of the proposed algorithms.

The structure of this paper is as follows: Section 1 provides a preliminary introduction to the entire study. Section 2 discusses related works in the fields of visual servoing, object detection, and deep learning, examining each area separately. Section 3 first introduces the two datasets used in this study and then focuses on DETR for object detection, PSO and polynomial trajectory planning, and BILSTM-KAN for trajectory prediction. Section 4 conducts experimental validation of the algorithms discussed in Section 3 using ROS and Gazebo and analyzes the results. The conclusion summarizes this paper, evaluates the strengths and weaknesses, and plans future work.

3. Methodology

3.1. Datasets

This study utilizes two primary datasets to enhance the performance and accuracy of the visual servoing system. The first dataset is the VOC07 dataset [43], widely recognized for its diverse and extensive range of indoor scene images. This dataset, comprising 9963 images are employed to train the DETR model for robust object detection. The VOC07 dataset is meticulously annotated, providing detailed class labels and bounding box coordinates for each object, which are essential for developing and evaluating object detection models. The variety and complexity of the scenes within this dataset make it ideal for testing the DETR model's capability to accurately recognize and localize objects under different conditions.

The second dataset is generated using the Particle Swarm Optimization (PSO) algorithm for trajectory planning. This dataset consists of the end-effector motion coordinates of a robotic arm, sampled during the execution of optimized trajectories. By planning the paths within the robotic arm's operational workspace, the PSO algorithm ensures smooth and feasible motion paths, capturing a wide array of dynamic scenarios. The sampled trajectory data points serve as input for training the BILSTM-KAN network, which is responsible for precise trajectory prediction. This dataset plays a critical role in enabling the model to learn and predict the complex movements of the robotic arm with high accuracy, ensuring reliable and efficient operation in real-world applications.

Dataset for Object Detection

Given that this research's simulation environment is a Gazebo, which includes a setup mimicking an interior setting, the VOC07 dataset is particularly appropriate due to its wide variety of indoor scenes. This dataset, known as the Pascal Visual Object Classes Challenge 2007, comprises 9963 images divided into a training set of 5011 images and a test set of 4952 images. The VOC07 dataset spans 20 diverse object categories, including animals (such as birds, cats, and dogs), vehicles (including bikes, cars, and buses), and household items (like bottles, chairs, and dining tables) as shown in Figure 5.



Figure 5. Dataset sample—from http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html (accessed on 20 July 2024).

The dataset's structure is meticulously organized: for each image, there is a corresponding annotation file in XML format. These files detail the contents of the image, listing the present objects, their classes, and the bounding box coordinates that mark each object's position within the image. This structured format makes VOC07 an invaluable resource for developing and benchmarking computer vision algorithms, enabling standardized access to and interpretation of ground truth data.

Opting for the VOC07 dataset to evaluate the DETR model leverages its complex and varied scenarios, which present challenges such as occlusion, varying object scales, and diverse backgrounds. These conditions are ideal for testing the robustness of the DETR model in recognizing and accurately localizing objects under different circumstances, thus highlighting its effectiveness in real-world applications compared to traditional detection systems.

3.2. DETR Model

DETR (Detection Transformer) is a target detection model based on the transformer architecture that views target detection as a direct set prediction problem [44]. Unlike traditional methods based on anchors and region proposal networks, DETR uses a novel bipartite matching loss function and end-to-end training approach, thereby avoiding complex post-processing steps such as non-maximum suppression. This model processes the entire image with a set of object queries through its transformer encoder–decoder

structure, directly predicting the category and bounding box of all objects in parallel. The transformer architecture adopted by DETR is shown in Figure 6.

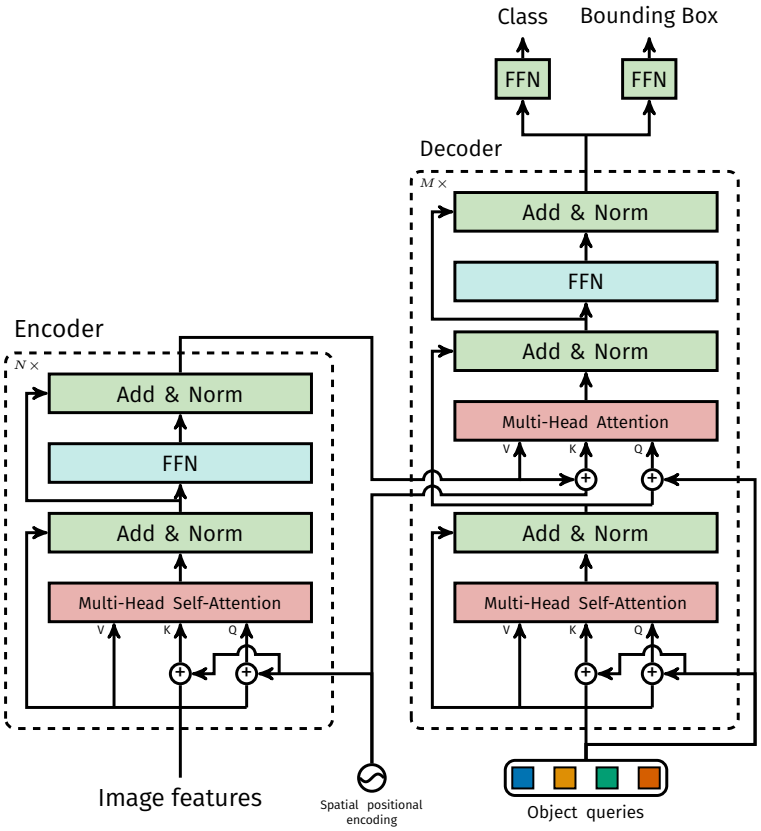


Figure 6. The architecture of DETR's transformer [44].

The main advantage of using DETR for object tracking in complex backgrounds is its global perspective and parallel processing capabilities. In vision servo systems, this feature is particularly useful because the system often needs to quickly and accurately locate and track objects in a variable and complex environment. DETR's capabilities can significantly improve recognition accuracy and tracking stability, making vision servo systems more effective at handling objects that are occluded or moving rapidly, thus achieving higher performance and reliability in automation and robotics applications.

3.2.1. Object Detection Set Prediction Loss

In the DETR model, object detection is formulated as a direct set prediction problem, where a fixed number N of predictions is inferred in a single pass through the decoder. To address the challenge of effectively comparing predicted objects (class, position, size) with ground truth, the model employs an optimal bipartite matching strategy, leveraging a specific loss function designed to enforce unique predictions.

We define the true set of objects as y and the set of N predictions as $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$. Given that N is typically larger than the actual number of objects in an image, we consider y as a set of size N padded with $\emptyset$ (no object). The goal is to find an optimal permutation σ of N elements that minimizes the following total matching cost:

$$\hat{\sigma} = \arg\min_{\sigma \in S_N} \sum_{i=1}^{N} L_{\text{match}}(y_i, \hat{y}_{\sigma(i)}), \tag{1}$$

where L_{match} is a pairwise matching cost between a ground truth object y_i and a prediction $\hat{y}_{\sigma(i)}$.

This optimal assignment is computed efficiently with the Hungarian algorithm as shown in Algorithm 1. The Hungarian algorithm is a typical online association algorithm based on graph theory that is used to find the maximum matching in bipartite graphs and is often employed to determine the maximum matching number and minimum vertex cover. It is an online and rapid matching method used in multi-object tracking to associate targets between consecutive frames. A bipartite graph is divided into two sets of data, where each set can connect directly to the other but not within the same group. In multi-object tracking, the detection box sets from consecutive frames are considered as two sets of data. These sets have a matching relationship, where the detection boxes of the same target in consecutive frames form a pair. After applying non-maximum suppression within the same frame, it is assumed that each detection box represents a distinct target and there are no matching relationships between them.

Algorithm 1: Hungarian algorithm

Input: A bipartite graph $G = (X, Y)$
Input: Condition - The graph G contains a matching from X to Y
Output: Identify a matching M such that $|X| > |Y|$, with M being the maximum matching in G

1 **Step 1:** Initialize M to be empty; Start with any vertex $x \in X$ that is not in M and set $S = \{x\}, T = \emptyset$.
2 **Step 2:** If $N(S) \subseteq T$, adjust G and the unmatched vertices in Y: Add every $y \in N(S) - T$ to T.
3 **Step 3:** If a vertex $y \in Y$ is found, update M, set $S = S \cup \{z\}$, $T = T \cup \{y\}$, and repeat Step 2; otherwise, adjust M by using the alternating path $P(x, y)$, update $M = M \cup E(P) - M \cap E(P)$, and return to Step 1.

The Hungarian loss function is defined as

$$L_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} L_{\text{box}}\left(b_i, \hat{b}_{\hat{\sigma}}(i)\right) \right], \tag{2}$$

The bounding box loss is

$$L_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{\text{IoU}} L_{\text{IoU}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}} \| b_i - \hat{b}_{\sigma(i)} \|_1. \tag{3}$$

This matching cost is computed using both the classification accuracy and geometric similarity of bounding boxes.

Here, L_{box} is a bounding box loss, which typically combines terms such as the IoU (Intersection over Union) and L_1 losses to evaluate the accuracy of the bounding box prediction. The bounding box prediction component is crucial for aligning the predicted boxes with the ground truth. Each ground truth object y_i can be described as a pair (c_i, b_i), where c_i is the class label and b_i is a vector representing the bounding box. The prediction for the corresponding index $\sigma(i)$ includes a class probability $p_{\hat{\sigma}(i)}(c_i)$ and a bounding box prediction $\hat{b}_{\sigma(i)}$. The matching cost L_{match} is then formulated as

$$L_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = -\mathbf{1}_{c_i \neq \varnothing} p_{\hat{\sigma}(i)}(c_i) + \mathbf{1}_{c_i \neq \varnothing} L_{\text{box}}(b_i, \hat{b}_{\sigma(i)}). \tag{4}$$

This strategy of bipartite matching and direct set loss computation significantly streamlines the training process by reducing reliance on heuristic pre-processing or post-processing steps, which are common in traditional object detection frameworks.

3.2.2. DETR Architecture

The DETR model showcases a streamlined architecture, as illustrated in Figure 7, comprising three core components: a CNN backbone for feature extraction, an encoder–decoder transformer mechanism, and a straightforward feed-forward network (FFN) for the final detection outputs.

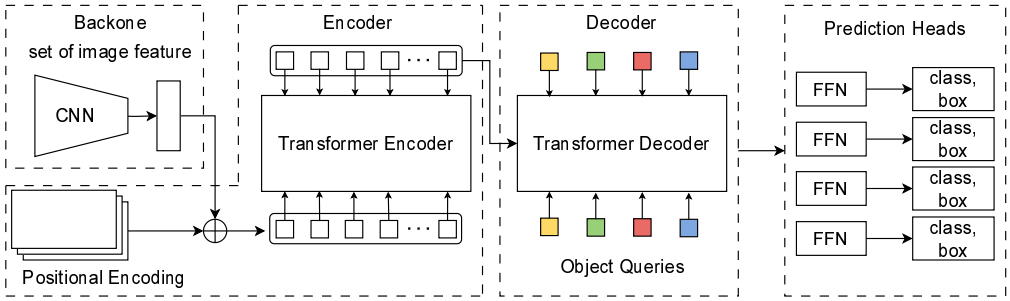


Figure 7. DETR structure diagram.

Backbone: The initial input, an image x_{img} represented as $\mathbb{R}^{3\times H_0\times W_0}$ (indicating three color channels), is processed through a standard CNN backbone. This backbone transforms the input into a lower-resolution feature map f characterized by dimensions $\mathbb{R}^{C\times H\times W}$, where, typically, $C = 2048$ and both H and W are $\frac{H_0}{32}$, $\frac{W_0}{32}$, respectively.

Transformer Encoder: The feature map f undergoes a dimension reduction via a 1×1 convolution from C channels down to d channels, resulting in a new feature map $z_0 \in \mathbb{R}^{d\times H\times W}$. For processing in the encoder, the spatial dimensions of z_0 are merged into a single dimension, creating a feature sequence of $d \times HW$. Each encoder layer incorporates a standard setup consisting of a multi-head self-attention mechanism and an FFN. To accommodate the transformer's requirement for positional data due to its permutation invariance, fixed positional encodings are added to the sequence before each attention layer.

Transformer Decoder: This component adheres to a conventional transformer structure, where N embeddings of size d are processed using both self-attention and encoder–decoder attention mechanisms. Unlike the autoregressive model described by Vaswani et al., this setup decodes N objects simultaneously at each decoder stage. The invariance to permutation in the decoder requires distinct initial embeddings for each object, known as object queries. These are incorporated with learned positional encodings added to each layer's input, facilitating the transformation of N object queries into their respective output embeddings, which are subsequently decoded by an FFN into bounding box coordinates and class labels, producing N predictions.

Prediction Feed-Forward Networks (FFNs): A three-layer perceptron with ReLU activations and a hidden dimension of d, capped with a linear projection for class labeling via a softmax function, computes the final prediction. The architecture is designed to predict a fixed number N of bounding boxes, which typically exceeds the number of actual objects, incorporating an additional special class label $\emptyset$ to denote unoccupied slots.

Auxiliary Decoding Losses: To enhance training efficiency and accuracy, auxiliary losses are utilized in the decoder, particularly to fine-tune the predicted object counts per class. Shared parameters are employed across all prediction FFNs, with each decoder layer followed by a shared layer norm to standardize the inputs to these FFNs from different layers.

3.3. Trajectory Prediction Model

3.3.1. Particle Swarm Optimization and Polynomial Interpolation Trajectory Planning

In our research, we chose the PSO algorithm to generate trajectory datasets for robotic arms. The PSO algorithm effectively solves optimization problems in continuous spaces by simulating group behaviors found in nature, making it suitable for parameter optimization and motion optimization in trajectory planning. Compared to the A* algorithm, which is primarily suited for shortest path searches in static environments, and the RRT, which is designed for path planning in more complex environments, PSO demonstrates higher efficiency and global search capabilities in scenarios that do not require complex obstacle avoidance.

Given the spatial coordinates of the robot's starting point, two path points, and the endpoint in the Cartesian coordinate system, the joint angles at four interpolation points

are solved using inverse kinematics, denoted by θ_{ij}, where $i = 1, 2, \ldots, n$ represents the number of joints and $j = 1, 2, 3, 4$ represents the indices of the four interpolation points.

The authors in [45] proposed the 3-5-3 trajectory planning method. First, the trajectory from the starting point to the first midpoint is planned using a third-order polynomial; second, the trajectory from the first point to the second midpoint is planned using a fifth-order polynomial; third, the trajectory from the second midpoint to the endpoint is planned using a third-order polynomial. During the planning process, the velocities and accelerations at the connection points remain continuous.

The general form of the 3-5-3 spline polynomial for the i-th joint is

$$\begin{cases} l_{j1}(t) & = a_{j13}t^3 + a_{j12}t^2 + a_{j11}t + a_{j10} \\ l_{j2}(t) & = a_{j25}t^5 + a_{j24}t^4 + a_{j23}t^3 + a_{j22}t^2 + a_{j21}t + a_{j20}\,. \\ l_{j3}(t) & = a_{j33}t^3 + a_{j32}t^2 + a_{j31}t + a_{j30} \end{cases} \tag{5}$$

$l_{j1}(t)$, $l_{j2}(t)$, and $l_{j3}(t)$ represent the trajectories of the 3-5-3 spline polynomial. The coefficients can be determined based on the constraint conditions, and the matrix A can be determined based on the constraints and boundary conditions related to time t.

$$A = \begin{bmatrix} B & C & 0 \\ 0 & D & E \\ 0 & 0 & F \\ G & 0 & 0 \\ 0 & H & I \end{bmatrix}, \tag{6}$$

where each sub-matrix is defined as follows:

$$\boldsymbol{B} = \begin{bmatrix} t_{i1}^3 & t_{i1}^2 & t_{i1} & 1 \\ 3t_{i1}^2 & 2t_{i1} & 1 & 0 \\ 6t_{i1} & 2 & 0 & 0 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 \end{bmatrix},$$

$$\boldsymbol{D} = \begin{bmatrix} t_{i2}^5 & t_{i2}^4 & t_{i2}^3 & t_{i2}^2 & t_{i2} & 1 \\ 5t_{i2}^4 & 4t_{i2}^3 & 3t_{i2}^2 & 2t_{i2} & 1 & 0 \\ 20t_{i2}^3 & 12t_{i2}^2 & 6t_{i2} & 2 & 0 & 0 \end{bmatrix},$$

$$\boldsymbol{E} = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -2 & 0 & 0 \end{bmatrix}, \boldsymbol{F} = \begin{bmatrix} t_{i3}^3 & t_{i3}^2 & t_{i3} & 1 \\ 3t_{i3}^2 & 2t_{i3} & 1 & 0 \\ 6t_{i3} & 2 & 0 & 0 \end{bmatrix},$$

$$\boldsymbol{G} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \boldsymbol{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\boldsymbol{I} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The vector θ contains the joint angles at the four interpolation points:

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \theta_{i3} & 0 & 0 & \theta_{i0} & 0 & 0 & \theta_{i2} & \theta_{i2} \end{bmatrix}^{\mathrm{T}}. \tag{7}$$

The coefficients vector a can be determined by solving the following equation:

$$\boldsymbol{a} = \boldsymbol{A}^{-1}\boldsymbol{\theta} = \begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix}^{\mathrm{T}}, \tag{8}$$

Here, A_1, A_2, and A_3 are the vectors of coefficients for the following spline polynomials:

$$A_1 = \begin{bmatrix} a_{i13} & a_{i12} & a_{i11} & a_{i10} \end{bmatrix},$$

$$A_2 = \begin{bmatrix} a_{i25} & a_{i24} & a_{i23} & a_{i22} & a_{i21} & a_{i20} \end{bmatrix},$$

$$A_3 = \begin{bmatrix} a_{i33} & a_{i32} & a_{i31} & a_{i30} \end{bmatrix}.$$

PSO is a type of evolutionary computation technique inspired by the study of bird flocking behavior. The fundamental idea behind PSO is to find the optimal solution through collaboration and information sharing among individuals within the group. PSO's advantages include its simplicity, ease of implementation, and the lack of many parameters to tune. The PSO algorithm simulates birds within a flock using massless particles that possess only two attributes: velocity and position. Velocity represents the speed of movement while position indicates the direction of movement. Each particle independently searches for the optimal solution within the search space, identifying it as the current individual extremum. This individual extremum is then shared with other particles in the swarm. The best individual extremum found is considered the current global optimum for the entire swarm. All particles in the swarm adjust their velocities and positions based on the individual extremum that they have found and the current global optimum shared by the entire swarm.

In PSO, the algorithm is initialized with a group of random particles (random solutions). Then, the optimal solution is found through iteration. In each iteration, particles update themselves by tracking two "extremes" (pbest and gbest). After finding these two optimal values, particles update their velocity and position using the following formula.

$$v_i = v_i + c_1 \times \text{rand}() \times (\text{pbest}_i - x_i) + c_2 \times \text{rand}() \times (\text{gbest}_i - x_i), \tag{9}$$

$$x_i = x_i + v_i, \tag{10}$$

where

- v_i is the velocity of the particle.
- rand() is a uniform random number between 0 and 1.
- x_i is the position of the particle.
- c_1, c_2 are acceleration coefficients, usually $c_1 = c_2 = 2$.
- To prevent the velocity from exploding, the velocity is limited to $V_{\max}$. If the speed exceeds $V_{\max}$, then $\|v_i\| = V_{\max}$.

Based on the two formulas above, the standard form of Particle Swarm Optimization (PSO) is established.

$$v_i = \omega \times v_i + c_1 \times \text{rand}() \times (\text{ pbest }_i - x_i) + c_2 \times \text{rand}() \times (\text{ gbest }_i - x_i), \tag{11}$$

ω is called the inertia weight factor, and its value is non-negative.

- A larger value strengthens global search ability and weakens local search ability.
- A smaller value weakens global search ability and strengthens local search ability.

Dynamic ω can achieve better optimization results than a fixed value. The dynamic ω can change linearly during the PSO search process or it can change dynamically according to some performance measure function of the PSO.

Currently, the most commonly used method is based on the Linearly Decreasing Weight (LDW) approach, which is defined by the following formula:

$$\omega^{(t)} = (\omega_{\text{ini}} - \omega_{\text{end}}) \frac{(G_k - g)}{G_k} + \omega_{\text{end}}, \tag{12}$$

where

- G_k: maximum number of iterations;

- ω_{ini}: initial inertia weight;
- ω_{end}: inertia weight at the maximum number of iterations.

The flow of the PSO algorithm is shown in Figure 8.

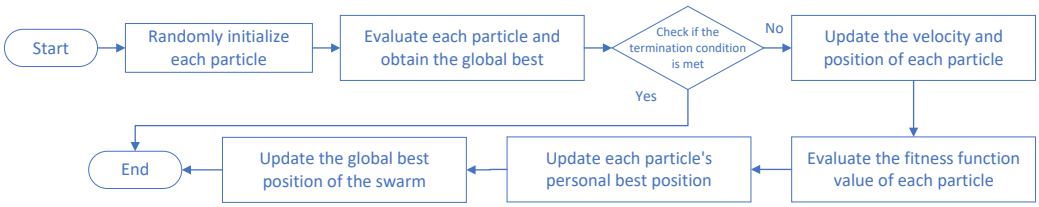


Figure 8. PSO algorithm flow.

3.3.2. Generating Unique Datasets

This section introduces the method used to generate a unique dataset specifically tailored for training machine learning models to predict robotic arm trajectories, thereby reducing target loss during operation. The dataset generation process begins with a detailed modeling of the robotic arm using Denavit–Hartenberg (DH) parameters. The DH convention provides a systematic way to represent each joint and link of the robotic arm, allowing for the computation of transformations between successive coordinates. Based on these parameters, the workspace of the robotic arm—defined as the set of all points that the end-effector can reach—was calculated. Understanding the workspace is crucial because it determines the points between which the arm can move.

To simulate realistic operational scenarios, trajectories were planned within the determined workspace. Two arbitrary points within this space were selected as the start and end points of each trajectory. Employing PSO techniques and polynomial interpolation discussed in the previous chapter, we optimized the trajectory path to ensure smooth and feasible motion of the robotic arm. The use of PSO allowed for efficient exploration of the solution space, finding optimal trajectory paths by simulating a population of candidate solutions. Polynomial interpolation was then applied to these paths to generate smooth curves that minimize jerky movements and ensure continuity of motion.

A total of 1000 trajectories were generated through the aforementioned method. To construct the dataset, the end-effector's position (x, y, z coordinates) was sampled at every fourth point along each trajectory. The resulting data points from these trajectories were concatenated to form a comprehensive dataset, consisting of approximately 280,000 rows. Each row in the dataset represents a distinct position of the robotic arm's end-effector in the 3D space, encapsulating the diverse range of movements that the arm can execute within its operational workspace. A part of the dataset is shown in Table 3 below.

Table 3. Sampled data points for trajectory dataset.

Step	X	Y	Z
1	2.8289	−40.519	−7.604
5	2.8221	−40.52	−7.6043
9	2.7747	−40.532	−7.6062
13	2.6879	−40.554	−7.6097
17	2.3943	−40.625	−7.6213
21	1.9782	−40.723	−7.6377
25	1.3541	−40.86	−7.6617
29	0.47641	−41.036	−7.6946
33	−0.70274	−41.241	−7.7372
37	−2.2342	−41.455	−7.7897
41	−4.1713	−41.642	−7.8852

3.4. BILSTM-KAN Prediction

3.4.1. BILSTM

As we know, RNNs have difficulties in learning long-term dependencies. LSTM-based models are an extension of RNNs that are able to address the vanishing gradient problem in a very clean way. LSTM models essentially extend the RNNs' memory to enable them to keep and learn long-term dependencies of inputs. This memory extension allows them to remember information over a longer period and thus enables them to read, write, and delete information from their memories. The LSTM memory is called a "gated" cell, where the word gate is inspired by the ability to make the decision of preserving or ignoring the memory information. An LSTM model captures important features from inputs and preserves this information over a long time. The decision to delete or preserve the information is made based on the weight values assigned to the information during the training process. Hence, an LSTM model learns what information is worth preserving or removing.

In general, an LSTM model consists of three gates: forget, input, and output gates. The forget gate makes the decision to preserve/remove the existing information, the input gate specifies the extent to which the new information will be added to the memory, and the output gate controls whether the existing value in the cell contributes to the output. Figure 9 is a schematic diagram of the structure of LSTM.

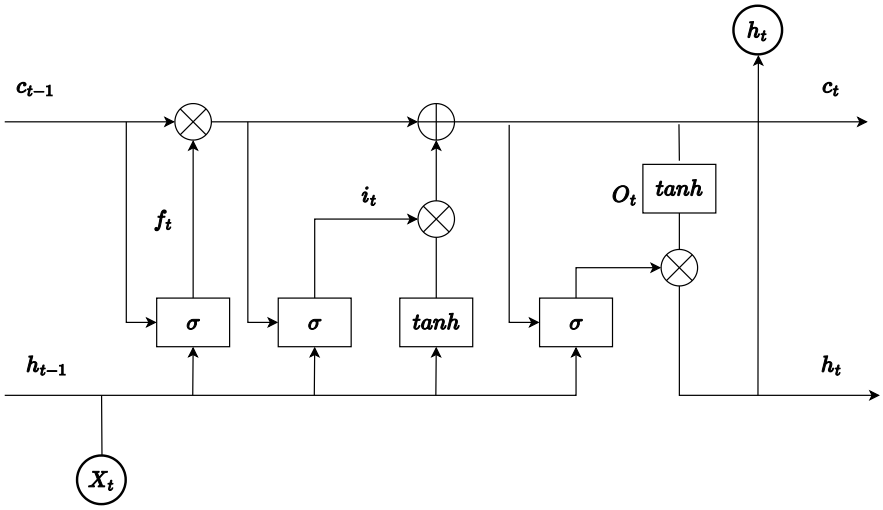


Figure 9. LSTM unit structure diagram.

Forget gate: Decide which information to discard or retain. Information from the previous hidden state and the current input is simultaneously passed into a sigmoid function, outputting a value between 0 and 1, where closer to 0 means that more should be discarded and closer to 1 means that more should be retained.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{13}$$

where σ is the sigmoid function, W_f and b_f are the weight and bias of the forget gate, and f_t is the forget gate output.

Input gate: The input gate is used to update the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i). \tag{14}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \tag{15}$$

where W_i and W_c are the weights of the input gate, b_i and b_c are the biases of the input gate, $\tanh()$ is the activation function, i_t is the input gate output, and $\tilde{C}_t$ is the candidate cell state.

The cell state of the previous layer is element-wise multiplied with the forget vector.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \tag{16}$$

where C_t is the cell state at time t and $\odot$ is the element-wise multiplication.

Output gate: The output gate determines the value of the next hidden state, which contains information from previous inputs.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{17}$$

$$h_t = o_t \odot \tanh(C_t), \tag{18}$$

where h_t is the output at time t and W_o and b_o are the weights and biases of the output gate.

As shown in Figure 10, BILSTM is an extension of the LSTM model where two LSTM layers are applied to the input data. In the first pass, LSTM is applied to the input sequence (i.e., the forward layer). In the second pass, the reverse form of the input sequence is fed into the LSTM model (i.e., the backward layer). Applying LSTM twice can improve the learning of long-term dependencies, thereby enhancing the accuracy of the model.

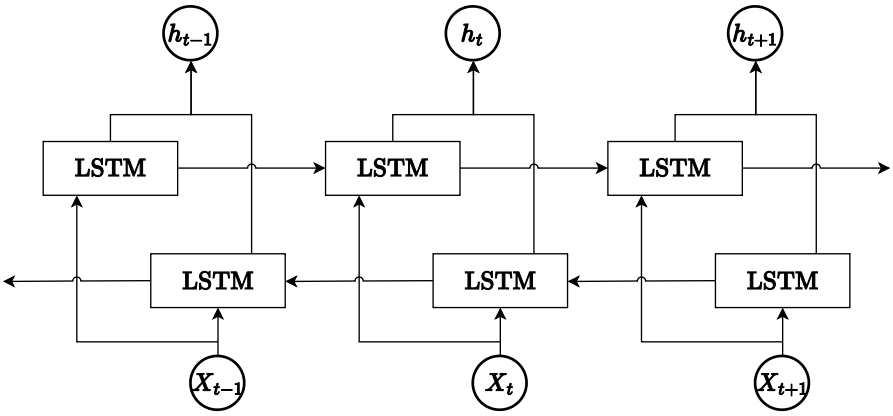


Figure 10. BILSTM structure diagram.

3.4.2. Attention Mechanism: Kolmogorov–Arnold Networks

A Multilayer Perceptron (MLP) is a fundamental theoretical module of deep learning and is currently the default model for approximating nonlinear functions. Its representational ability has been proven by the universal approximation theorem. However, MLPs also have notable drawbacks. For instance, in transformers, the number of parameters in an MLP is enormous, and it usually lacks interpretability.

In addition, KANs and attention mechanisms are fundamentally different. Attention mechanisms are a weighting mechanism used to dynamically select important parts of the input data. They determine which parts of the data are most important for the current task by calculating the correlations (attention weights) between different parts of the input. On the other hand, KANs focus on accurately fitting complex functional relationships through learnable activation functions and serve as an alternative to MLPs.

To enhance representational ability, MIT proposed KANs. KANs are essentially a combination of splines and MLPs, integrating the advantages of both. Specifically, a KAN = an MLP + spline. The comparison between MLPs and KANs is shown in Figure 11.

The universal approximation theorem states that a two-layer neural network with suitable activation functions can approximate any continuous function to any desired degree of accuracy. This theorem forms the foundation of the Multilayer Perceptron (MLP), enabling MLPs to handle and approximate various nonlinear functions.

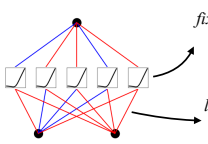
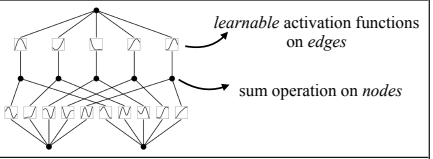
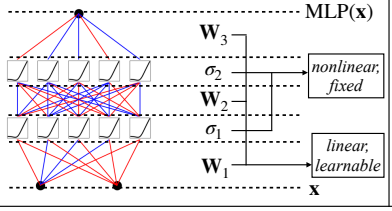
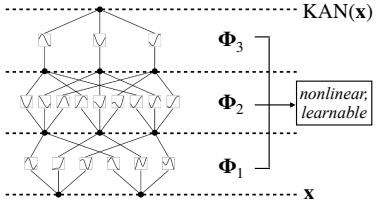
Model	**Multi-Layer Perceptron (MLP)**	**Kolmogorov-Arnold Network (KAN)**
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)		
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)		

Figure 11. Comparison of Multi-Layer Perceptron (MLP) and Kolmogorov–Arnold Network (KAN)—Ziming Liu et al. [46].

In contrast, the K-A theorem, proposed by Kolmogorov and Arnold, suggests that any multivariate continuous function can be represented as a composition and addition of a series of univariate continuous functions. Specifically, the theorem states that a multivariate function can be expressed as

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \varphi_{q,p}(x_p) \right). \tag{19}$$

This theorem elegantly demonstrates that all multivariate continuous functions can be decomposed into a combination of univariate functions. The activation functions of MLPs determine the fixed points, adjusting them through training. However, the main drawback of MLPs lies in the potential need for a large number of parameters to achieve high precision, which also affects interpretability.

KANs, inspired by the K-A theorem, improve the network representation ability by placing learnable activation functions on fixed node activation functions. Each multivariate function is represented as a univariate function (original node function plus sample function), significantly enhancing parameter efficiency and model interpretability. The structure of a KAN is shown in Figure 12.

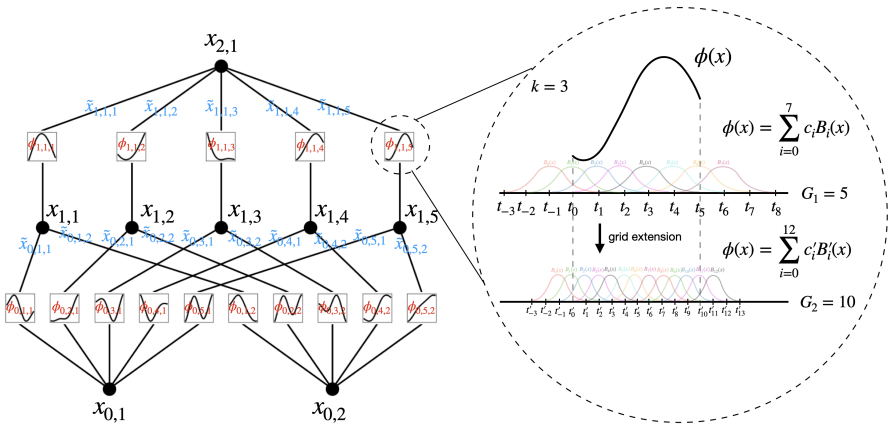


Figure 12. Basic architecture diagram of KAN—Ziming Liu et al. [46].

Basic Composition of KAN:

Neurons: In KANs, neurons perform simple addition operations without nonlinear activation functions.

Edges: Unlike traditional MLPs, the nonlinear activation functions in KANs are located on the edges (weights) rather than on the neurons. This means that each weight parameter is replaced by a learnable univariate function, which is parameterized as a spline curve.

KAN Layer: A KAN layer is actually a one-dimensional function matrix composed of an input of dimension n_{in} and an output of dimension n_{out}:

$$\Phi = \phi_{q,p}, \quad p = 1, 2, \ldots, n_{in}, \quad q = 1, 2, \ldots, n_{out}. \tag{20}$$

The KAN structure based on Kolmogorov–Arnold consists of an internal layer with $n_{in} = n$ and $n_{out} = 2n + 1$ and an external function composed of $n_{in} = 2n + 1$ and $n_{out} = 1$.

In the KAN, the inputs and outputs of each layer are transformed through a series of compositions and additions of univariate functions. We use the following notation to describe the structure of a KAN layer:

- $\mathbf{x}^{(l)}$: input vector of layer l.
- $\mathbf{x}^{(l+1)}$: output vector of layer $l + 1$.
- $\varphi_{i,j,l}$: activation function connecting the i-th node of layer l to the j-th node of layer $l + 1$.
- n_l: number of nodes in layer l.
- n_{l+1}: number of nodes in layer $l + 1$.

A transformation from the input vector $\mathbf{x}^{(l)}$ to the output vector $\mathbf{x}^{(l+1)}$ in a KAN layer can be expressed as

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \quad j = 1, \cdots, n_{l+1}, \tag{21}$$

where $x_{l+1,j}$ is the output of the j-th node of layer $l + 1$, $x_{l,i}$ is the input of the i-th node of layer l, and $\varphi_{l,j,i}$ is the univariate activation function connecting these two nodes. In matrix form, this can be represented as

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\mathbf{\Phi}_l} \mathbf{x}_l. \tag{22}$$

Here, Φ_l represents the function matrix corresponding to the l-th KAN layer. A general KAN is a combination of L layers: Given an input vector $\mathbf{x}_0 \in \mathbb{R}^{n_0}$, the output of the KAN is

$$\mathrm{KAN}(\mathbf{x}) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)\mathbf{x}. \tag{23}$$

$\varphi(x)$ is defined as

$$\varphi(x) = w(b(x) + spline(x)), \tag{24}$$

where $b(x)$ is

$$b(x) = silu(x). \tag{25}$$

The SiLU (Sigmoid Linear Unit) function, also known as the swish function, is a smooth, non-linear activation function that has been found to work well in deep learning models. It is defined as

$$silu(x) = \frac{x}{1 + e^{-x}}. \tag{26}$$

This function can help to mitigate the vanishing gradient problem and provide better performance compared to traditional activation functions such as ReLU.

In the KAN structure designed by the author, spline(x) is parameterized as a linear combination of B-splines:

$$spline(x) = \sum_i c_i B_i(x). \tag{27}$$

A complete KAN consists of multiple KAN layers. Suppose that a KAN has L layers, with input $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and output y; the computation process of the network is as follows:

1. First layer: Input $\mathbf{x}^{(0)} = \mathbf{x}$, output $\mathbf{x}^{(1)}$ is calculated through the activation function matrix Φ_0:

$$x_{1,j} = \sum_{i=1}^{n_0} \phi_{0,j,i}(x_{0,i}). \tag{28}$$

2. Intermediate layer (Layer l): Input $\mathbf{x}^{(l)}$, output $\mathbf{x}^{(l+1)}$ is calculated through the activation function matrix Φ_l:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}). \tag{29}$$

3. Final layer: Input $\mathbf{x}^{(L-1)}$, output y is calculated through the activation function matrix Φ_{L-1}:

$$y = \sum_{i=1}^{n_{L-1}} \phi_{L-1,1,i}(x_{L-1,i}). \tag{30}$$

3.4.3. BILSTM-KAN Model

The BILSTM-KAN model is designed to process sequences of spatial pose trajectory coordinates of a robotic arm's end effector. The model integrates the strengths of BILSTM networks and Kernelized Activation Networks (KANs) to achieve accurate and interpretable results. The structure of the BILSTM-KAN model can be described as follows:

- **Input Layer**: The input layer of the model receives the sequence of spatial pose trajectory coordinates of the robotic arm's end effector. These coordinates represent the position and orientation of the robotic arm over time, which are crucial for understanding and predicting the arm's movements.
- **BILSTM Layer**: The sequence data from the input layer are then fed into a BILSTM layer. The BILSTM layer consists of two LSTM networks: one processes the sequence in the forward direction and the other processes it in the backward direction. This bidirectional processing allows the model to capture dependencies and patterns from both past and future contexts within the sequence, enhancing the learning of temporal dynamics and improving the model's ability to understand the intricate movements of the robotic arm.
- **KAN Layer**: The output from the BILSTM layer, which encapsulates the learned features and temporal dependencies, is subsequently fed into a KAN. The KAN layer replaces the traditional fully connected (dense) layer typically used in neural networks. Instead of simple linear transformations, the KAN layer utilizes a combination of MLP and spline functions to transform the input. This approach leverages the representational power of MLPs while incorporating the flexibility and smoothness of spline functions, resulting in a more expressive and interpretable model.

3.5. Visual Servo Simulation Based on ROS and Gazebo

In this chapter, we will provide a detailed description of the vision-based robotic arm control system that we developed, which integrates cutting-edge image processing and deep learning technologies to achieve precise execution of dynamic tasks. The core of the system is visual servoing technology, which adjusts the movement of the robotic arm

based on visual information captured from the environment to achieve the predetermined operational goals.

As shown in Figure 13, the workflow of the system begins with real-time image analysis using the DETR (Detection Transformer) model. The system acquires real-time images from visual sensors installed in the environment. These images are fed into a DETR model for object detection.

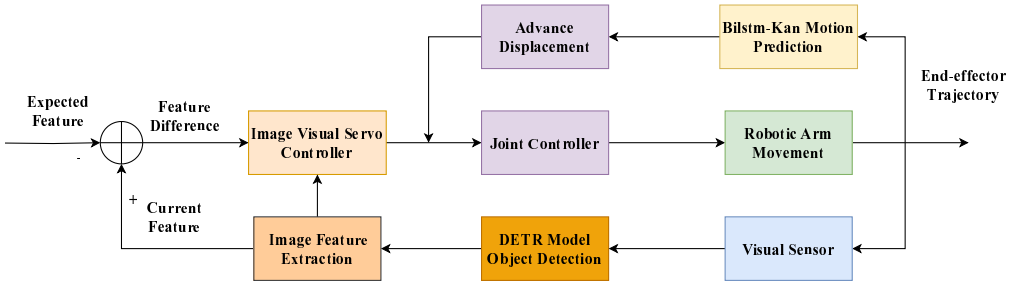


Figure 13. Visual servo simulation flowchart.

The detected object features are compared with the features of the target images preset in the system. The difference between the current detected features and the desired features is calculated through this comparison. These differences serve as the input for the subsequent joint controller.

The controller receives the feature differences and uses these data to calculate the desired angles for the six joints of the robotic arm, utilizing the Jacobian matrix. Based on the results, the controller issues motion commands to guide the robotic arm to adjust its posture to reach the target state.

On top of this process, the control system inputs the historical end-effector trajectory of the robotic arm into a BILSTM-KAN network for motion prediction. The prediction model outputs expected positions, which are converted into the joint angles needed by the joint controller. The joint controller adjusts based on these predicted angles in advance, allowing the robotic arm to make a small, pre-emptive movement to track the target object more effectively and quickly.

4. Experiments and Results

4.1. ROS and Gazebo Simulation

The ROS is a flexible framework used for developing complex robotic software systems. It is not a traditional operating system per se but rather a collection of software frameworks designed for robotic applications. ROS offers a rich set of tools and libraries to help developers to create robotic applications, including hardware abstraction, low-level device control, commonly used functionalities, inter-process communication, and package management. Its design aims to promote code reuse in robotics research and development. The core of ROS is its publisher–subscriber message-passing system, which allows data to flow between different parts of the system, enhancing its scalability and portability across various hardware configurations.

Gazebo is an open-source robotic simulation platform widely used for robot design and testing. It provides a realistic environment where robots can safely test their sensors and behavioral actions without risks. Gazebo excels in its ability to simulate physical environments closely resembling the real world, including lighting, gravity, and other physical phenomena. Moreover, Gazebo supports various physics engines, such as ODE and Bullet, allowing users to select the most suitable engine for their needs. With its seamless integration with ROS, developers can test their code developed under the ROS framework within Gazebo, which is particularly crucial for developing complex robotic behaviors. Modeling within Rviz and Gazebo are shown in Figures 14 and 15. The robotic

arm adopts the six-degree-of-freedom robotic arm model ER20-1780 from Aston Company, with an arm span of 1780 mm. The DH parameters are shown in Table 4 below.

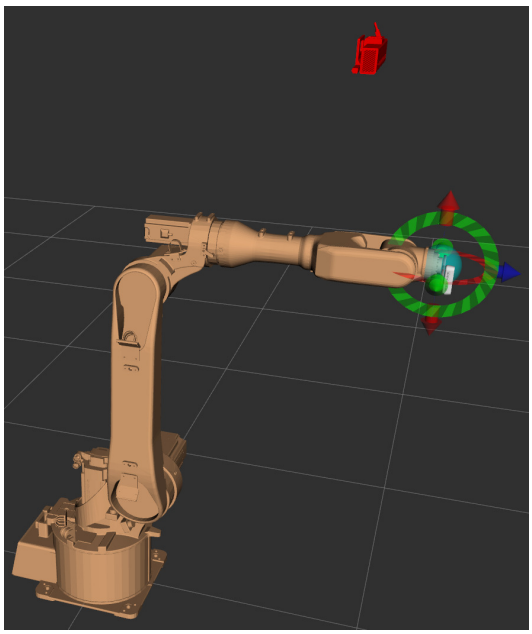


Figure 14. Robotic arm model display in Rviz.

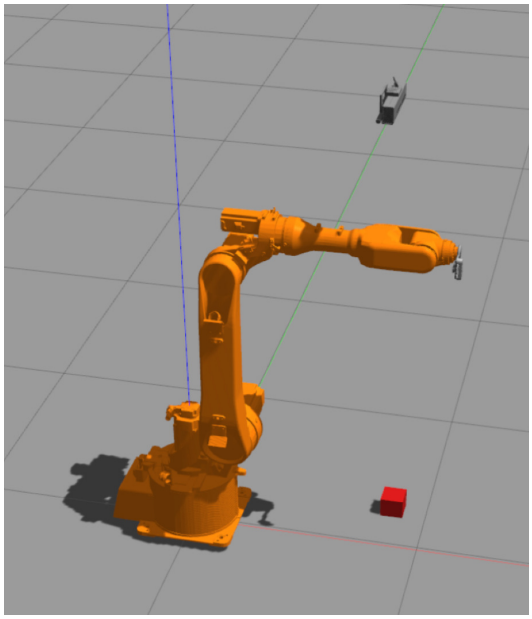


Figure 15. Robotic arm model display in Gazebo.

Table 4. DH parameters for the robotic arm.

Link	d (m)	a (m)	α (Rad)	Offset (Rad)
1	0.5	0.2	$-\frac{\pi}{2}$	0
2	0	0.79	0	$-\frac{\pi}{2}$
3	0	0.14	$-\frac{\pi}{2}$	0
4	0.78	0	$\frac{\pi}{2}$	0
5	0	0	$-\frac{\pi}{2}$	0
6	0.104	0	0	0

In addition, two cameras are added in the space: one fixed at the end of the six-axis robotic arm for visual servo tracking and one fixed to the side, used for system initialization and indicating that the robotic arm has reached the target position. The parameters for these cameras are detailed in Table 5.

Table 5. Two cameras' parameters

Parameter	camera_rgb_sensor	camera_ir_sensor
Link	camera_rgb_link	camera_ir_link
Type	Depth camera	Depth camera
FOV	1.04	1.04
Image Format	B8G8R8	L8
Resolution	640 × 480	640 × 480
Clip Near	0.01	0.01
Clip Far	5	5
Update Rate	30	30
Point Cloud Cutoff	0.1	0.1

Figure 16 shows a display of the camera mounted on the robot arm (eye-in-hand), highlighting that the target object is always centered in the frame. Figure 17 displays the view from another camera.

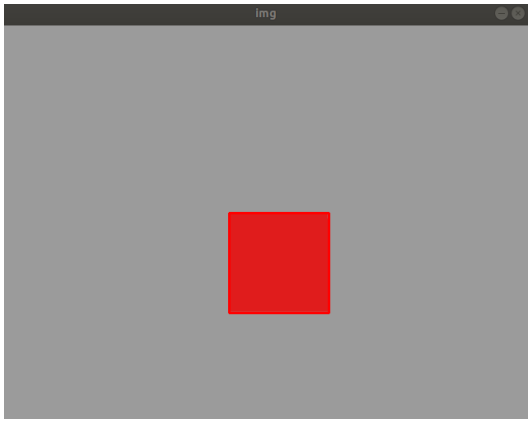


Figure 16. Eye-in-hand camera view.

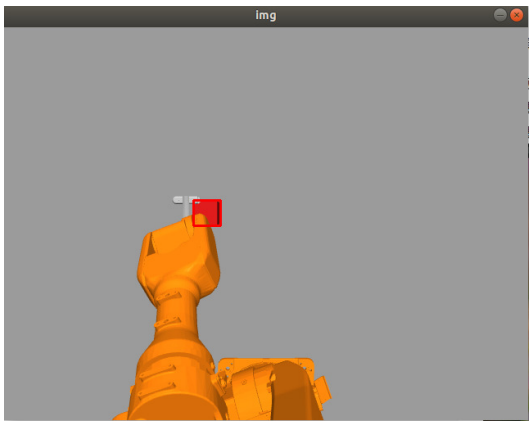


Figure 17. Spatial camera view.

During the servo initialization, the robot arm first contracts to allow the camera to locate the object and avoid obstructions caused by the arm itself, as shown in Figure 18.

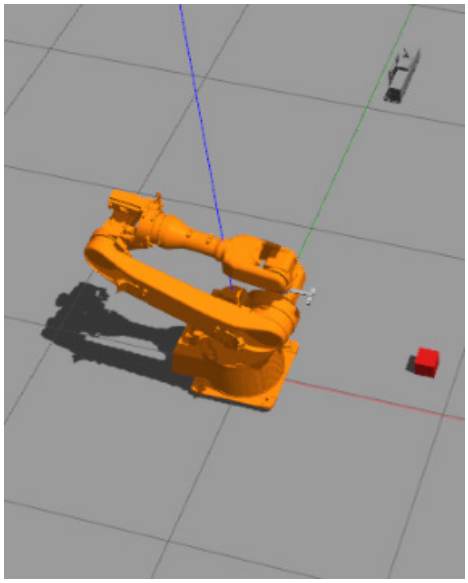


Figure 18. Robot arm contraction.

After initialization, the robot arm remains above the object, waiting to start the servo operation, as shown in Figure 19.

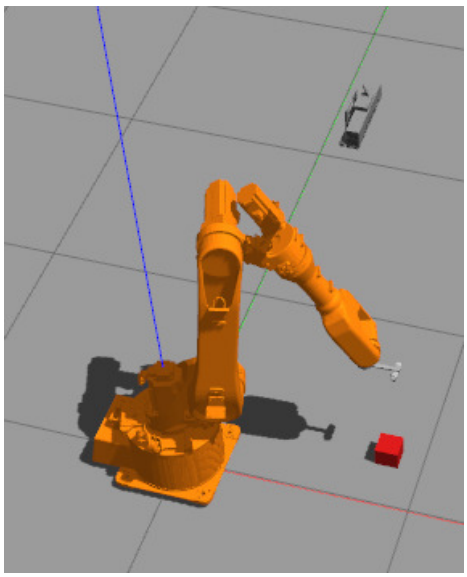
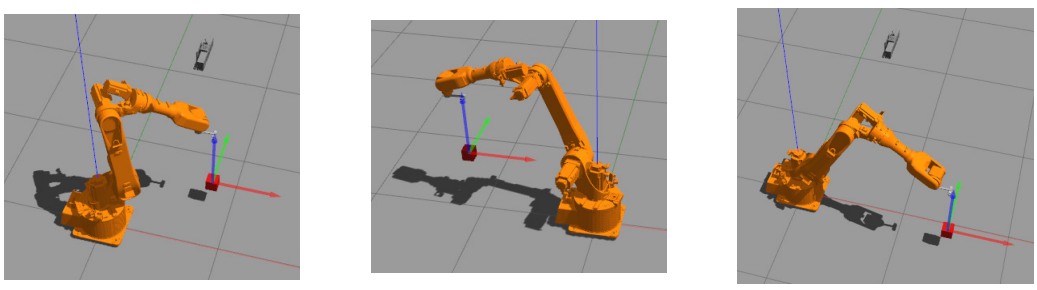


Figure 19. Initialization complete.

Some of the poses during the servo process are shown in Figure 20.



(**a**) (**b**) (**c**)

Figure 20. Examples of different poses during the servo process (**a**–**c**).

4.2. Target Detection

Firstly, calculate AP for each category separately; for each category, sort the detection results by confidence. For each threshold, calculate the precision and recall for that threshold. Accuracy is the number of correctly detected targets divided by the total number of detected targets. The recall rate is the number of correctly detected targets divided by the total number of actual targets. As the recall rate increases, record accuracy and form an accuracy recall rate curve. A curve example is shown in Figure 21.

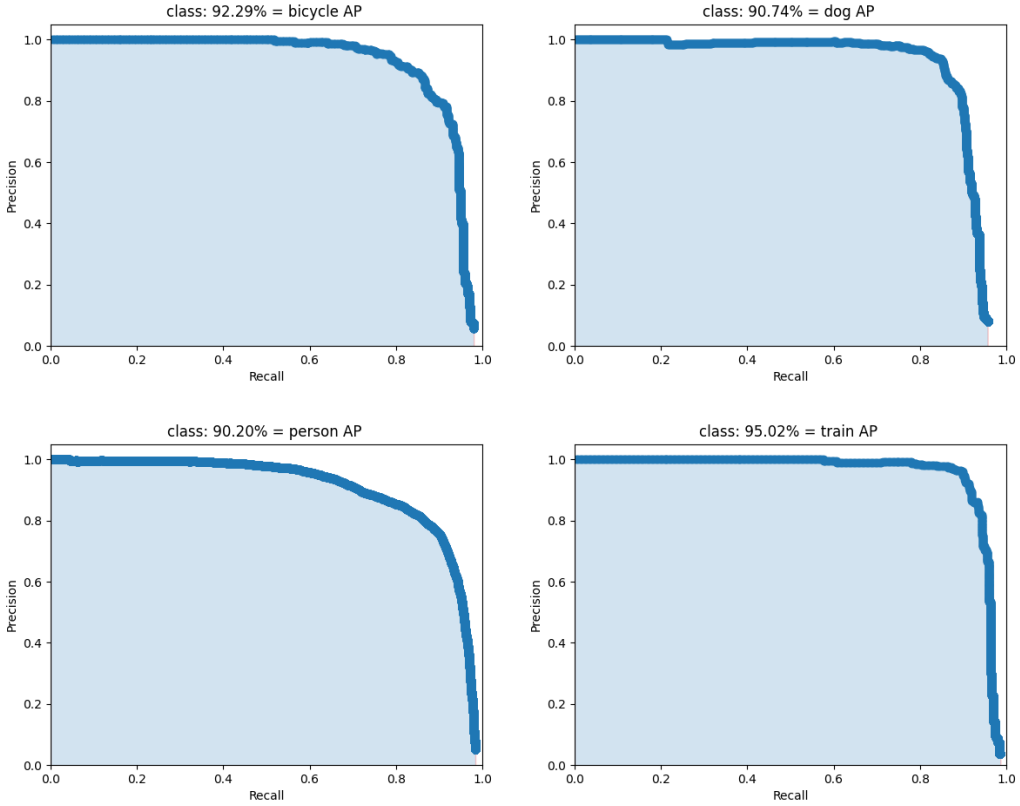


Figure 21. AP scores for bicycle, dog, person, and train.

The mean Average Precision (mAP), also known as the average value of Average Precision (AP), is the main evaluation indicator of object detection algorithms. Object detection models usually use speed and accuracy (mAP) metrics to describe their strengths and weaknesses. The higher the mAP value, the better the detection performance of the object detection model on a given dataset. In this paper, a map is used to evaluate the training effect of the model, with a value of 84.47%.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i, \tag{31}$$

where N is the number of classes and AP_i represents the average precision for the i-th class. The map scores for all types are shown in Figure 22.

Table 6 below summarizes mAP scores for various object detection models on the VOC07 dataset, including a comparison of recent models. Notably, the DETR model achieves an impressive mAP score, showcasing its superior performance relative to other contemporary architectures.

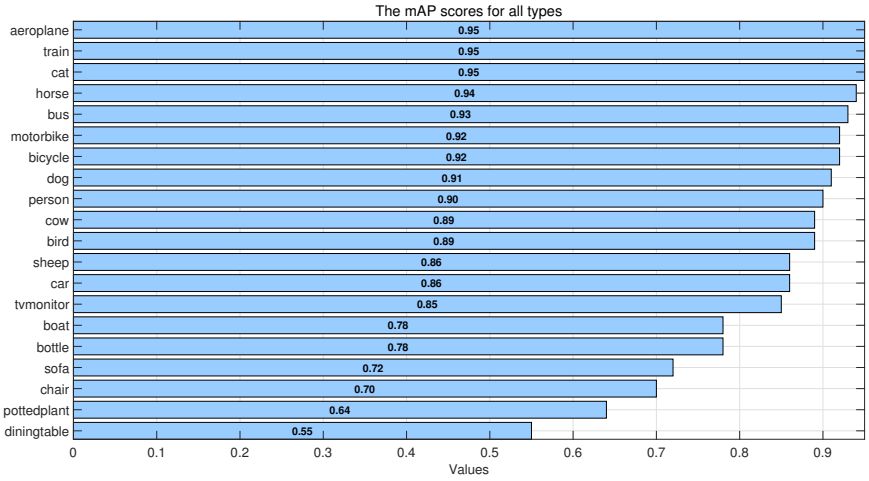


Figure 22. The mAP scores for all types.

Table 6. Training configurations and mAP scores for different models on VOC07 dataset.

Name	Model	Epochs	Mini Batch Size	Learning Rate	Image Size	MAP (%)
Hierarchical Shot Detector [47]	VGG16	250	32	0.004, decreases by a factor of 10 at 150 and 200 epochs	512×512	83.00
Coupling Global Structure with Local Parts [48]	CoupleNet	-	2	10^{-3} for 80k, then 10^{-4} for 30k iterations	resized from 480 to 864	82.70
Early Exit Evolutionary Neural Architecture [49]	EEEA-Net-C2	200	32	initial 0.01, then 10^{-3} for 40k, 10^{-4} for 10k	320×320	81.80
Single Shot MultiBox Detector [50]	SSD512	-	32	0.001 for 40k, then 10k iterations with 10^{-4} and 10^{-5}	512×512	81.60
BlitzNet [51]	BlitzNet512	-	32	initial learning rate is 10^{-4} and decreases twice during training by a factor 10.	512×512	81.50
DETR	DETR	300	8	initial 10^{-4}, cosine decay	800×800	84.47

The F_1 score provides a single indicator of model accuracy by combining precision and recall as shown in Figure 23. Due to its consideration of both accuracy and recall, it is particularly suitable for situations where both false negative and false positive classes are heavily penalized. The range of F1 scores is from 0 to 1, where 1 represents perfect accuracy and recall and 0 represents at least one being zero.

The F_1 score is calculated as the harmonic mean of precision and recall:

$$F_1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{32}$$

- **Precision** is the ratio of correctly predicted positive observations to the total predicted positives. It is given by

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{33}$$

where TP denotes a True Positive and FP denotes a False Positive.

- **Recall** (also known as sensitivity) is the ratio of correctly predicted positive observations to all observations in the actual class. It is calculated as

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{34}$$

where FN denotes a False Negative.

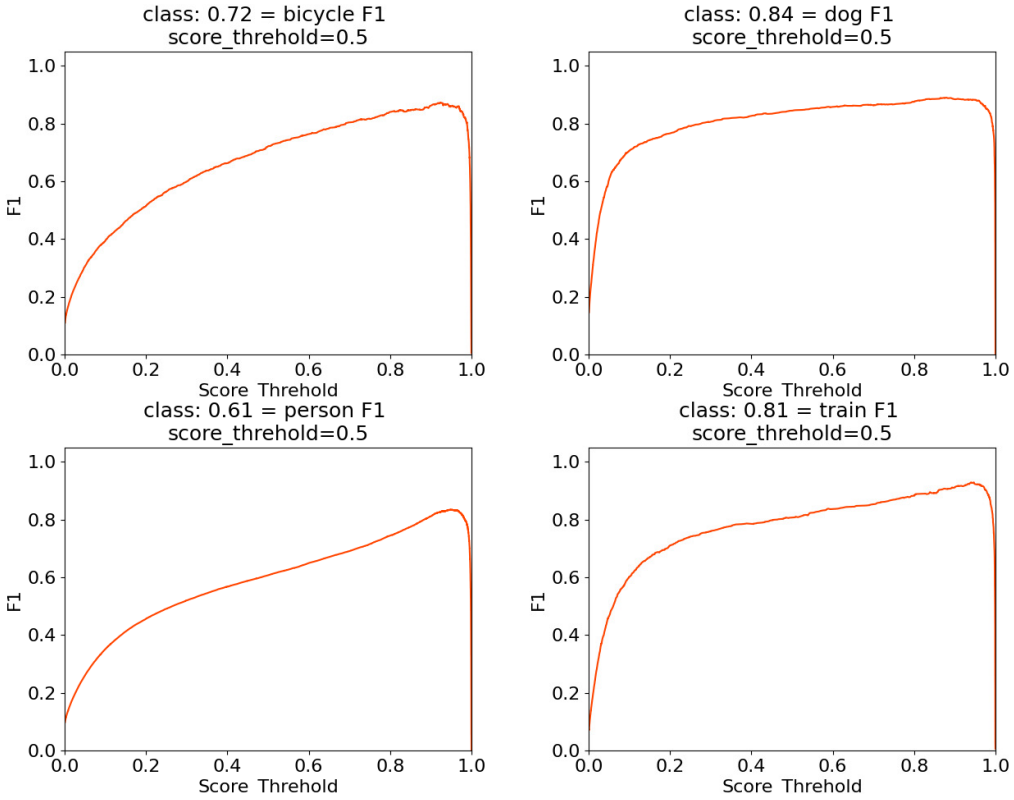


Figure 23. F_1 scores for bicycle, dog, person, and train.

The results of the object detection task performed on these images are as shown below. Each image displays various objects (e.g., people, vehicles, animals, furniture), with bounding boxes and confidence scores indicating the model's certainty for each detection. The model performs well with a variety of objects in different scenes. Confidence scores are usually high, suggesting that the model is very certain about its predictions, as shown in Figures 24 and 25.

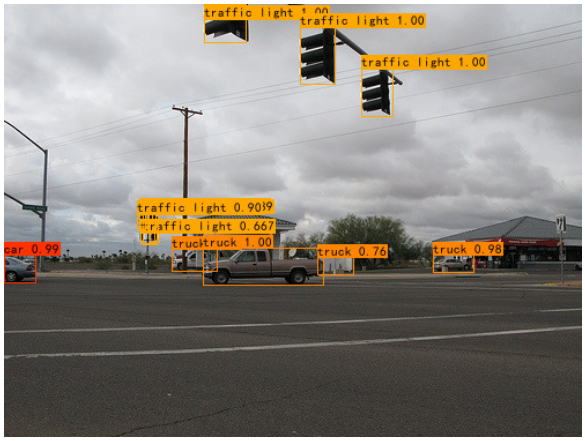


Figure 24. VOC07 dataset: detection of traffic lights, trucks, and car.

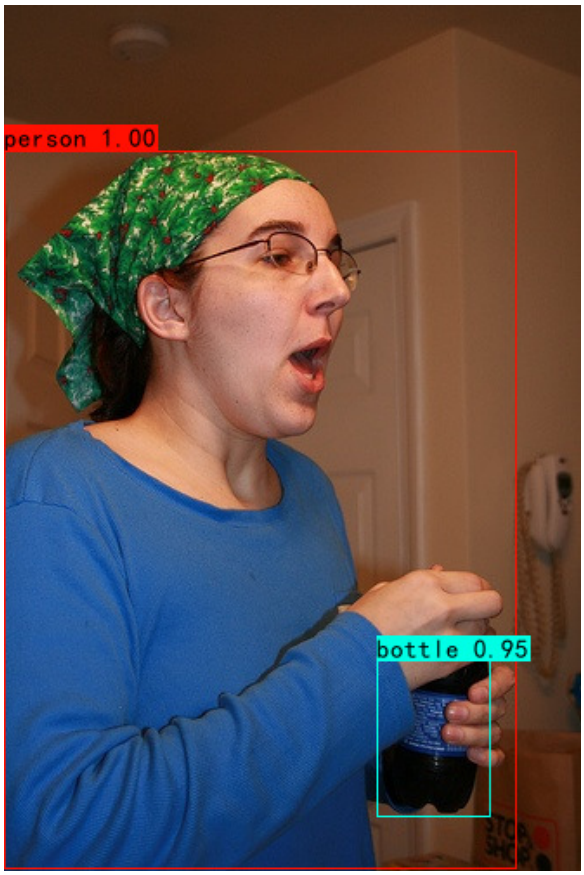


Figure 25. VOC07 dataset: detection of person and bottle.

The model is capable of handling complex scenes with multiple objects. In urban environments with complex traffic or crowds, it is still able to effectively differentiate between various types of objects. For instance, Figure 26 can clearly distinguish between trucks and cars, and despite the dense crowd in Figure 27, it still accurately frames each individual.

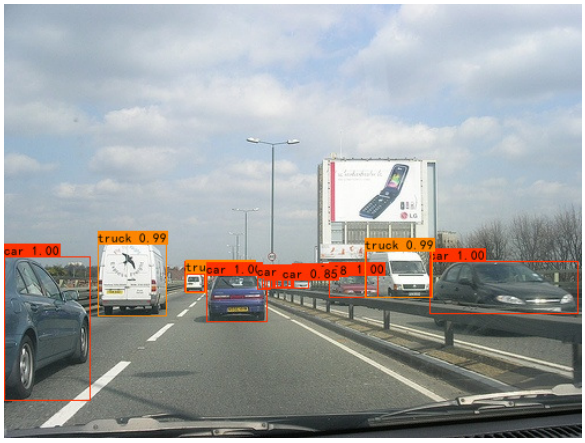


Figure 26. VOC07 dataset: detection of cars and trucks.

The model can recognize a wide range of objects: not only common ones like cars and people but also smaller or less obvious objects like mobile phones, books, or water cups as shown in Figure 28. For example, in Figure 29, even though the books on the shelf are only visible from the side, they are still distinctly recognized.

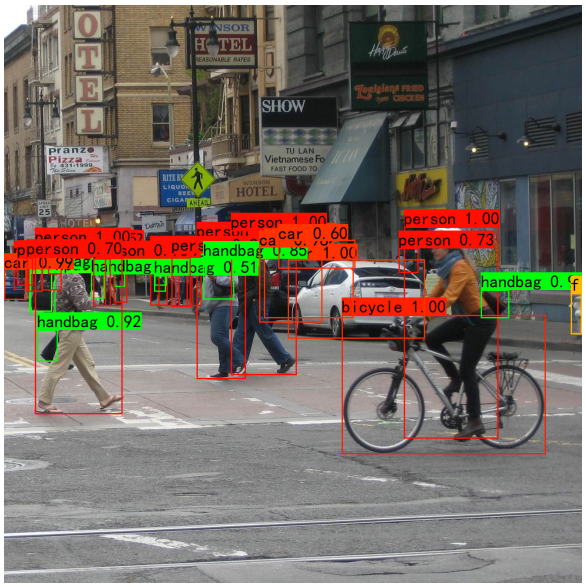


Figure 27. VOC07 dataset: detection of bicycle, persons, and handbags.

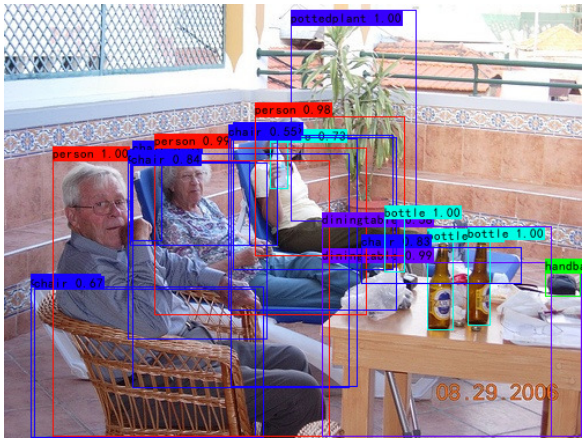


Figure 28. VOC07 dataset: detection of persons, chairs, dining tables, bottles, and handbag.

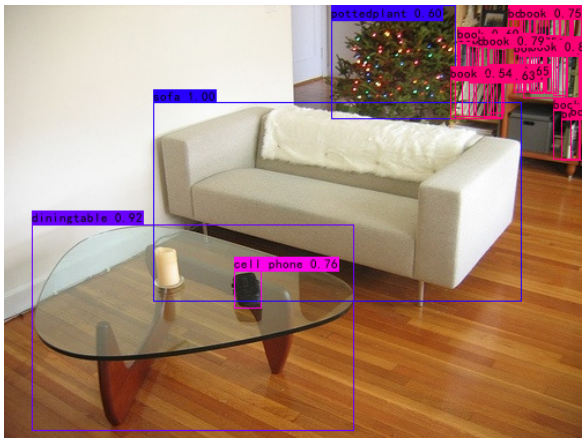


Figure 29. VOC07 dataset: detection of sofa, dining table, potted plant, and cellphone.

Even when targets appear in the image from different perspectives, they can still be successfully recognized and achieve high confidence. As seen in Figure 30, the model can successfully identify boats of different directions and distances.

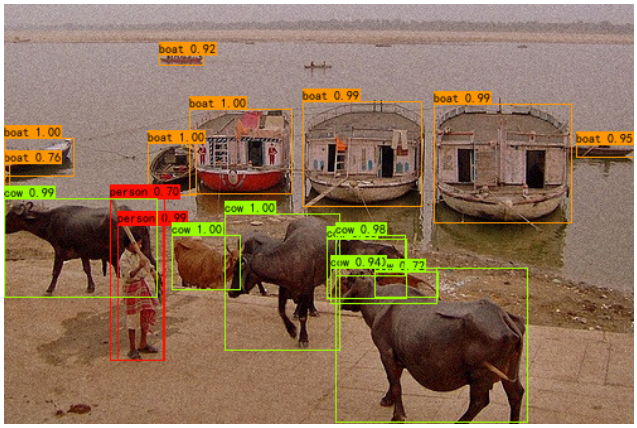


Figure 30. VOC07 dataset: detection of boats, cows, and persons.

While the images mostly show correct detection, limitations are inevitable due to not-so-prominent features or the lack of consideration for global features, resulting in some misclassifications, although the confidence is not high. Figure 31 mistakenly identifies part of a sofa as a chair, while Figure 32 incorrectly identifies stairs as a sofa, because their local features all conform to another object. Most of the object detection results in Figure 33 are accurate, but there are still some errors with small objects. For example, part of a folder is wrongly identified as a mobile phone.

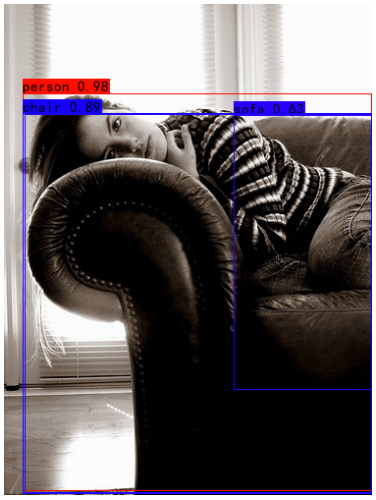


Figure 31. VOC07 dataset: detection of person, chair, sofa.

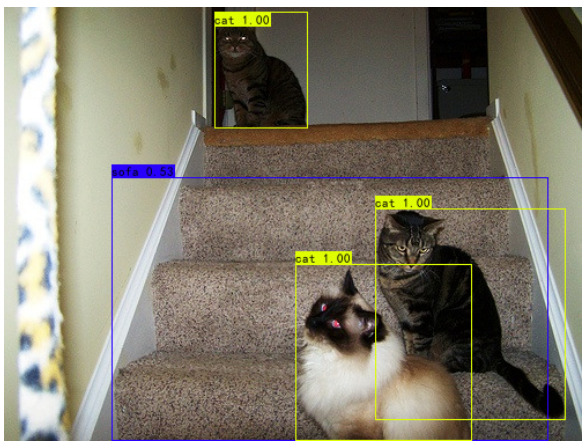


Figure 32. VOC07 dataset: detection of cats and step.

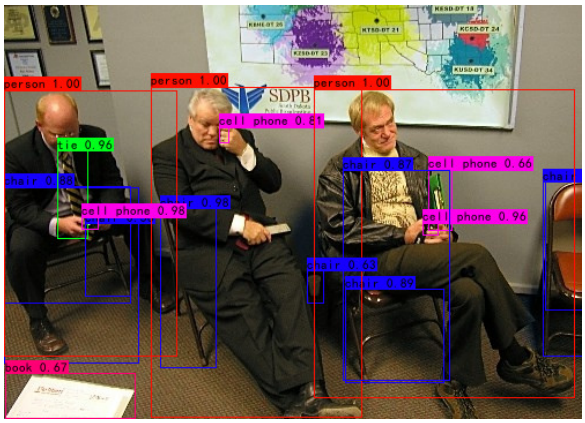


Figure 33. VOC07 dataset: detection of persons, chairs, cell phones, tie, and book.

4.3. Particle Swarm Optimization (PSO) Trajectory Planning

Here, we present the hyperparameters used in the Particle Swarm Optimization (PSO) algorithm, organized in Table 7 for clarity.

Table 7. PSO hyperparameters and their values.

Hyperparameter	Value
Control Period	0.05
Number of Iterations	200
Number of Particles	20
Dimension of the Problem	3
Inertia Weight	0.7
Cognitive Coefficient	4
Social Coefficient	4

Firstly, we use Matlab to model the robotic arm with DH parameters and then use the Monte Carlo method to solve the workspace, as shown in Figure 34. Figure 35 shows the planned route of a randomly selected trajectory.

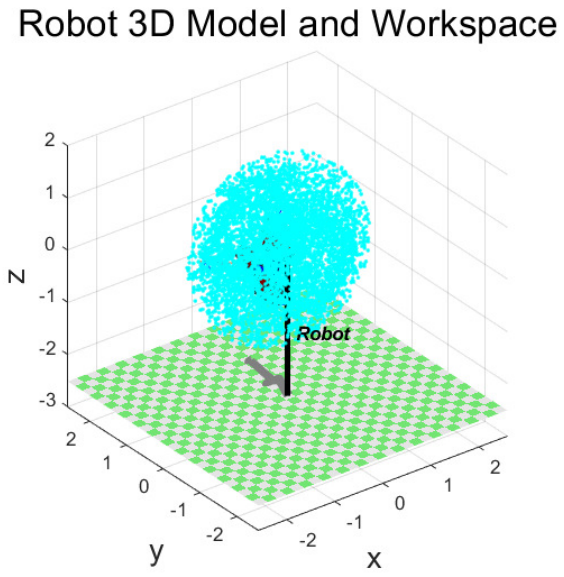


Figure 34. Robot arm 3D model and workspace.

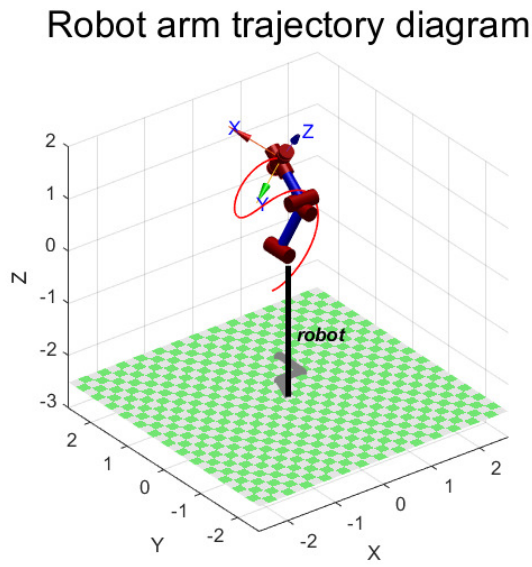


Figure 35. The route of a random trajectory.

These charts provide a detailed demonstration of the effects of applying PSO technology to optimize the trajectory of robotic joint movements. Figure 36 shows the changes in the historical best fitness of the population for each joint as the number of iterations increases during the optimization process. By comparing the position, velocity, and acceleration curves before and after optimization, we can specifically analyze the impact of the PSO algorithm. The comparison before and after PSO algorithm optimization is shown in Figure 37.

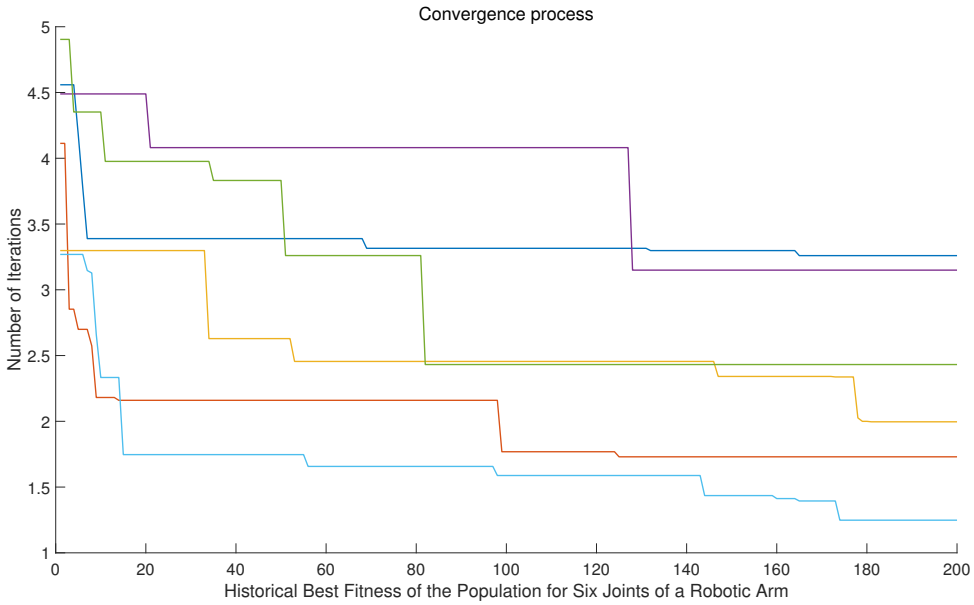


Figure 36. The change chart of the best fitness in population history.

In the pre-optimization position curves, the joints exhibited significant nonlinear changes throughout the time range, particularly in the 4 to 6 s interval, where large fluctuations occurred. Post-optimization, the position curves became noticeably smoother, especially in the 1 to 4 s range, where the changes in joint positions were more gradual, showing higher control precision and lower positional deviations.

On the velocity curves, there were large fluctuations in joint speeds before optimization, particularly for joints 3 and 5, which showed sharp peaks and rapid changes around 2

to 4 s. These abrupt changes in velocity could lead to high dynamic stresses and energy consumption in an unoptimized system. After optimization, the velocity curves showed more consistent and stable changes, with sharp peaks effectively suppressed, thus reducing dynamic loads and potential mechanical fatigue risks.

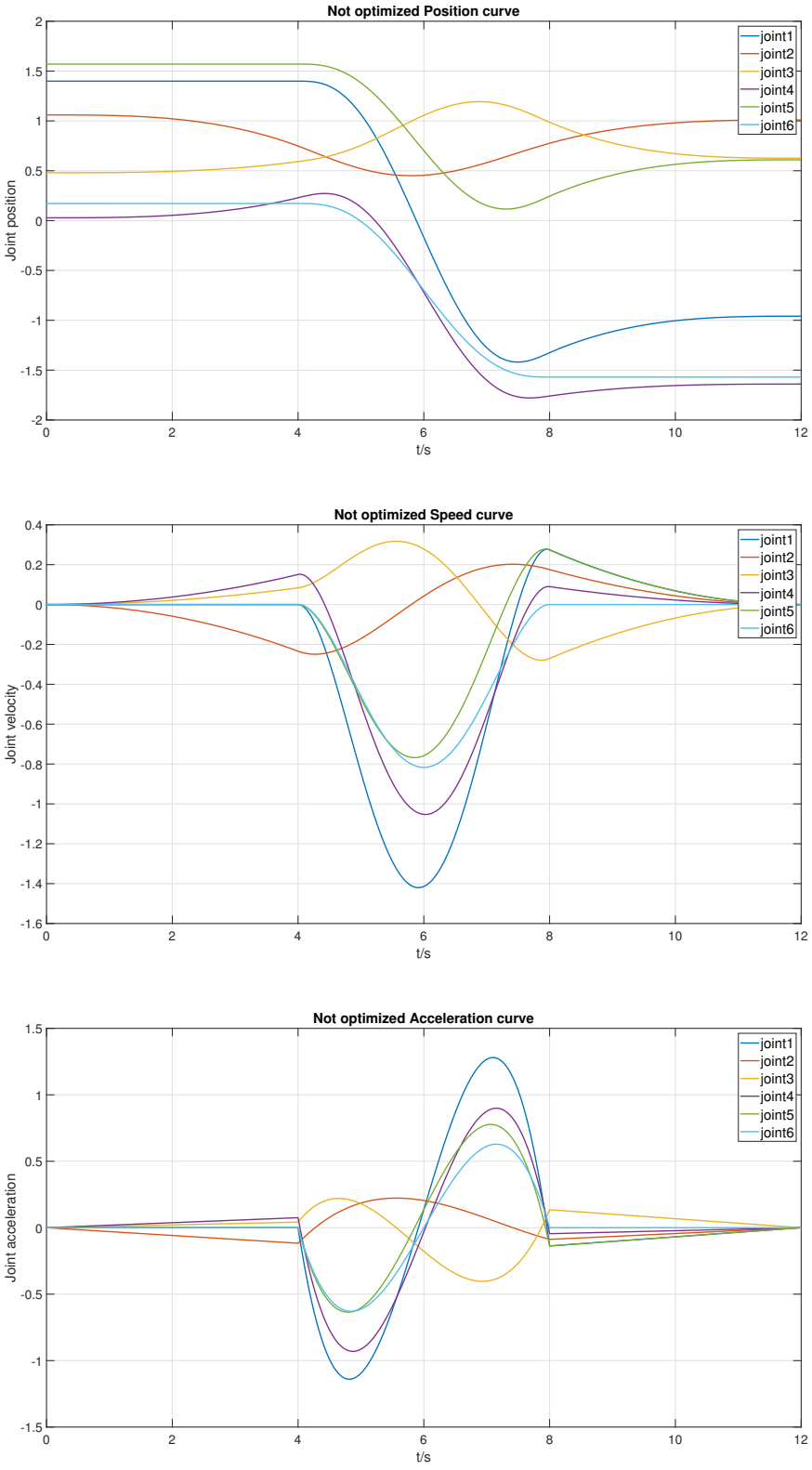


Figure 37. *Cont.*

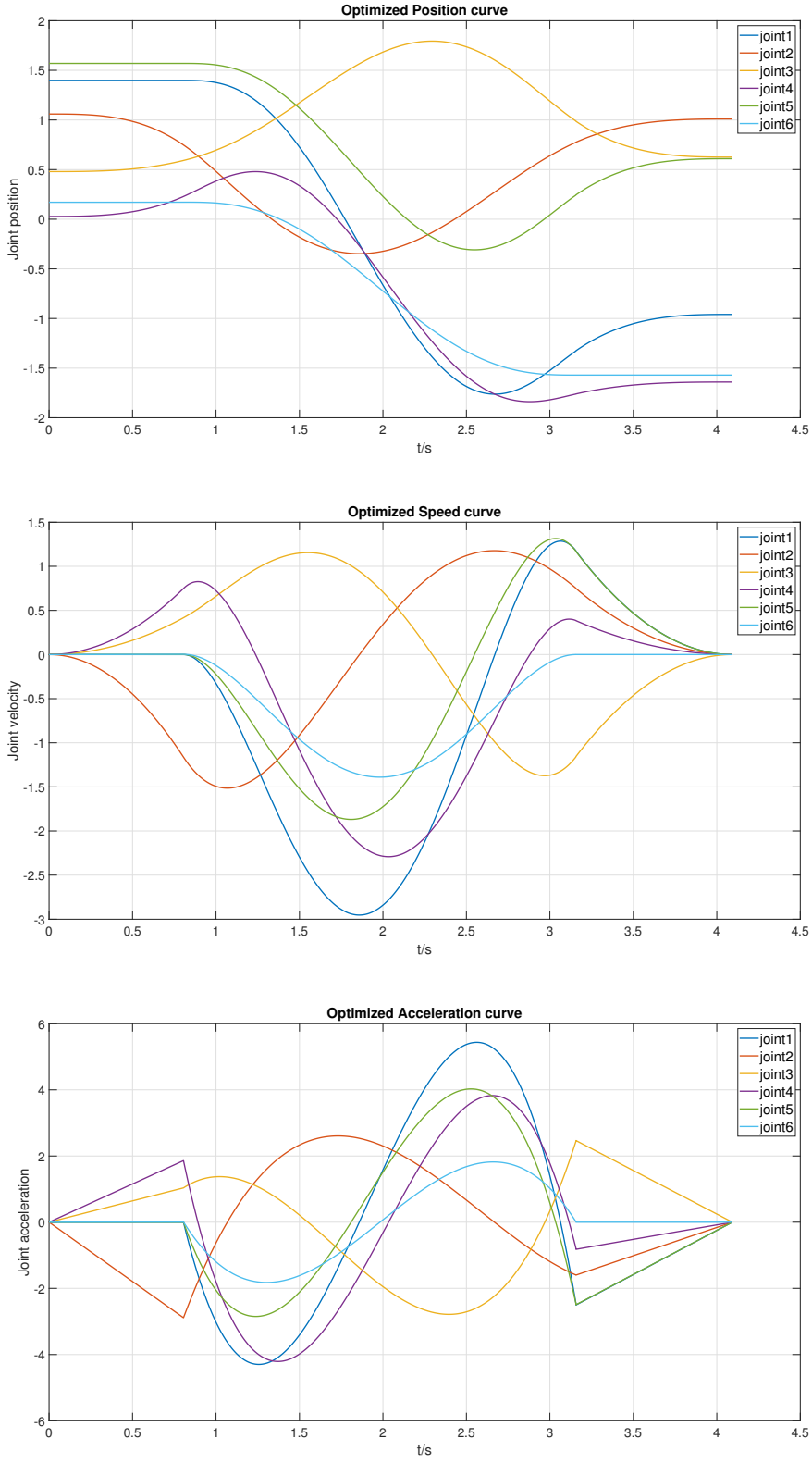


Figure 37. The comparison before and after PSO algorithm optimization.

Changes in the acceleration curves were even more apparent. Before optimization, the joint acceleration curves were filled with sharp peaks, such as joint 4, which suddenly increased to about 1.5 m/s² at 3 s. Such sudden changes in acceleration pose a challenge to the robot's mechanical structure, increasing the risk of wear and failure. After optimization,

the maximum amplitude of acceleration significantly decreased, as the same joint at the same time point dropped to about 0.5 m/s², greatly improving the smoothness and safety of the movement.

In summary, PSO optimization significantly enhances the overall performance of robotic joint movements, reducing fluctuations and extremes at various stages to achieve smoother and more consistent dynamic behavior. This not only improved the operational efficiency of the robot but also helped to reduce maintenance costs and extend equipment life.

4.4. BILSTM-KAN Prediction

Mean Squared Error (MSE) is a commonly used metric for evaluating the performance of a regression model. It measures the average of the squares of the errors—that is, the average squared difference between the actual and predicted values. The formula for MSE is given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{35}$$

where n is the number of samples and y_i and $\hat{y}_i$ are the actual and predicted values, respectively. MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero indicate a better fit to the data. Since it squares the errors before averaging, MSE gives more weight to larger errors, making it sensitive to outliers.

R-squared (R^2), also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance for a dependent variable that is explained by an independent variable or variables in a regression model. The formula for R^2 is

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{36}$$

where y_i, $\hat{y}_i$, and $\bar{y}$ are the actual, predicted, and meaning of the actual values, respectively. R^2 ranges from 0 to 1, with higher values indicating a greater proportion of variance explained by the model. An R^2 of 1 indicates that the regression predictions perfectly fit the data. However, R^2 can also be negative when the model is worse than a horizontal line (mean of the actual values), implying that the model does not follow the trend of the data at all.

Figure 38 illustrates the MSE loss values over epochs, providing insight into the model's performance throughout the training process.

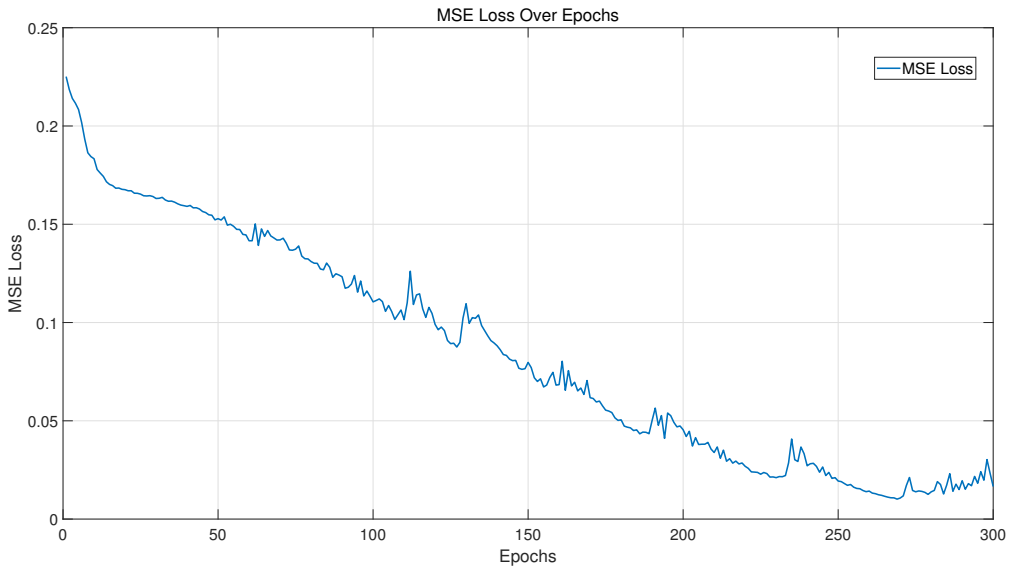


Figure 38. MSE loss during training.

The following provides a detailed analysis and discussion of the trends in MSE and R^2 observed during the training process of the model as shown in Figures 39 and 40. These metrics reflect the model's performance and learning effectiveness from two key dimensions throughout the training period.

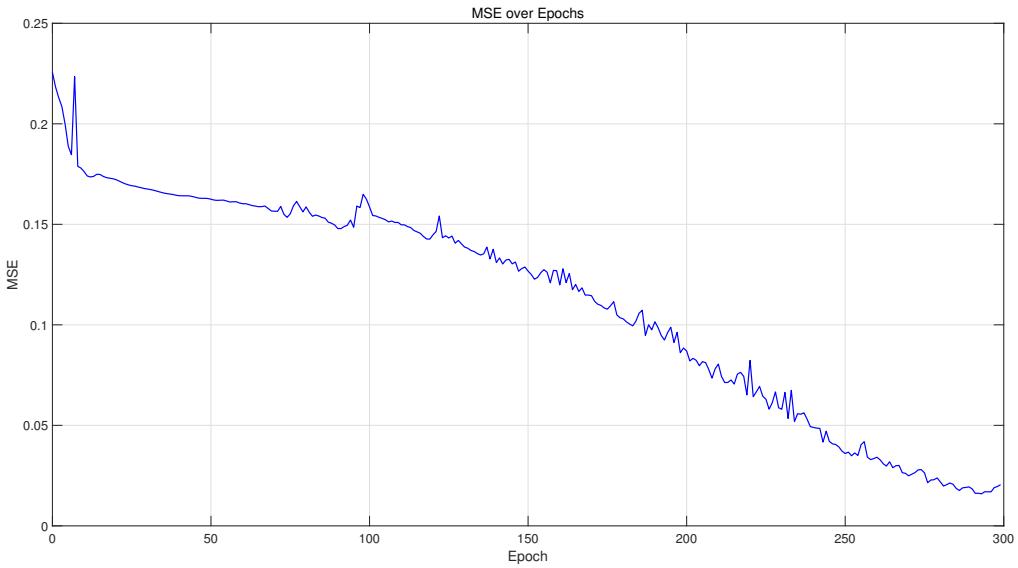


Figure 39. The MSE variation curve over 300 epochs.

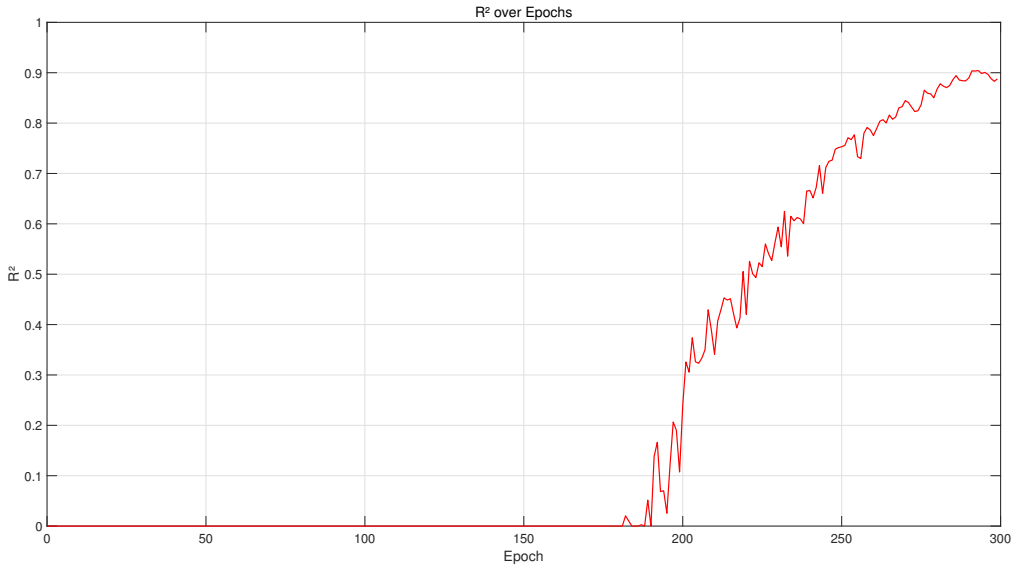


Figure 40. The R^2 variation curve over 300 epochs.

Initially, the trajectory of MSE indicates the model's rapid learning capability at the early stages, with MSE quickly dropping from approximately 0.25 to around 0.15. This significant reduction signifies that the model effectively captures the basic patterns and relationships within the data. As training progresses, MSE continues to slowly decrease and finally stabilizes around 0.019 by the 300th epoch. This trend suggests that the model gradually adapts to the complexity of the data, reducing prediction errors through internal parameter adjustments and demonstrating good generalization ability.

Next, observing the trend in R^2, the model exhibits initial instability. R^2 values are extremely low at the beginning of training, with several significant fluctuations, possibly due to suboptimal model parameter initialization or initial adaptation issues with the training data distribution. By the 150th epoch, the R^2 value continues to increase and

stabilizes at above 0.9 by the end of training, demonstrating that the model can explain the majority of variance in the data, showing excellent predictive power. After comparing the performance of the BILSTM+KAN model with other models (TCN, LSTM, transformer) as shown in Figure 41, we found that the BILSTM+KAN model demonstrated significant advantages. Firstly, the BILSTM+KAN model exhibited a rapid convergence speed in the early stages of training, with MSE quickly decreasing, showcasing its strong learning capability. Additionally, the R^2 score of this model rose rapidly during training, eventually approaching 1, indicating that it achieved an excellent fit to the task and could effectively explain the variability in the data as shown in Figure 42. In contrast, the other models did not perform as well in terms of MSE and R^2, especially the transformer model, which had higher MSE and lower R^2 scores.

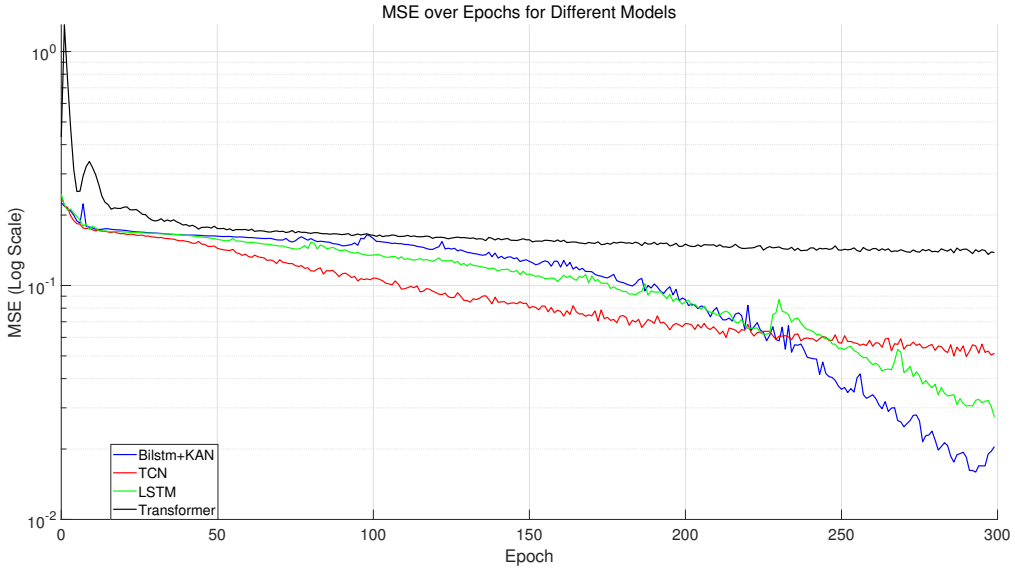


Figure 41. MSE over epochs for different models (TCN, LSTM, and transformer).

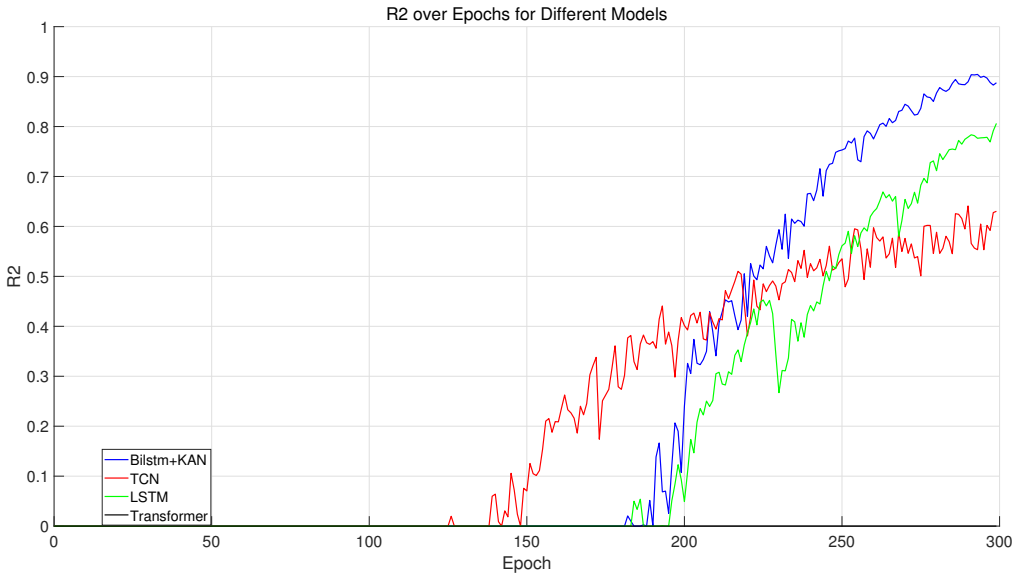


Figure 42. R^2 over epochs for different models (TCN, LSTM, and transformer).

As shown in Figure 43, after incorporating the KAN, all models showed improved MSE performance. Compared to other models, BILSTM+KAN demonstrated the best MSE performance, indicating that the BILSTM architecture may be particularly well suited to in-

tegration with KANs. Although transformer+KAN also improved, its overall MSE remains relatively high, suggesting that further adjustments or optimizations might be necessary.

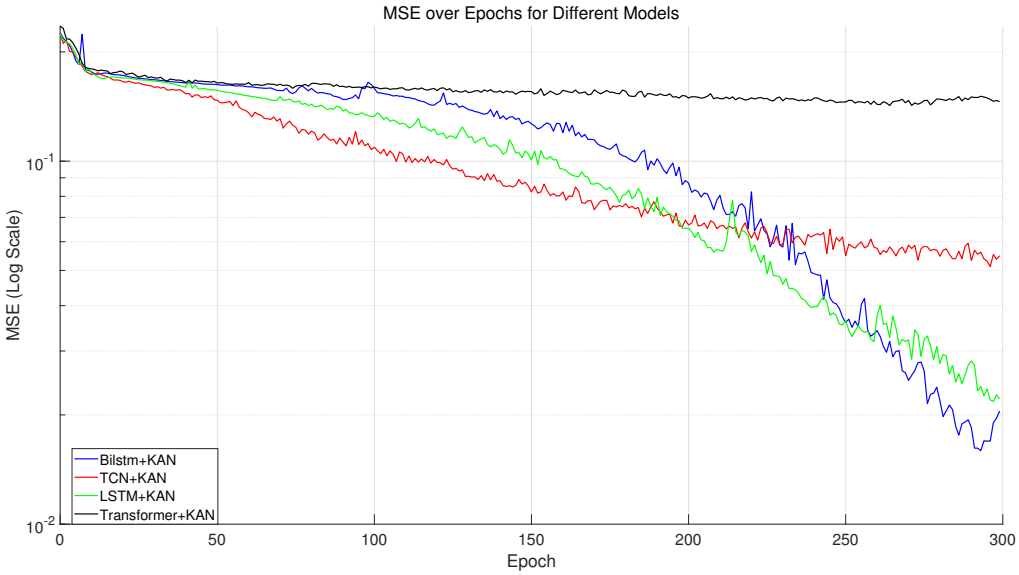


Figure 43. MSE over epochs for different models (TCN, LSTM, and transformer) with KAN.

Similarly to the MSE results, the R^2 scores for all models increased after the addition of a KAN in Figure 44. This suggests that KANs enhanced the models' generalization abilities, enabling them to better explain data variability. While transformer+KAN did see an improvement in the R^2 score, the increase was smaller compared to LSTM and BILSTM, which may be due to the transformer's inherent complex attention mechanisms. The addition of a KAN might not have provided significant additional benefits to the transformer model.

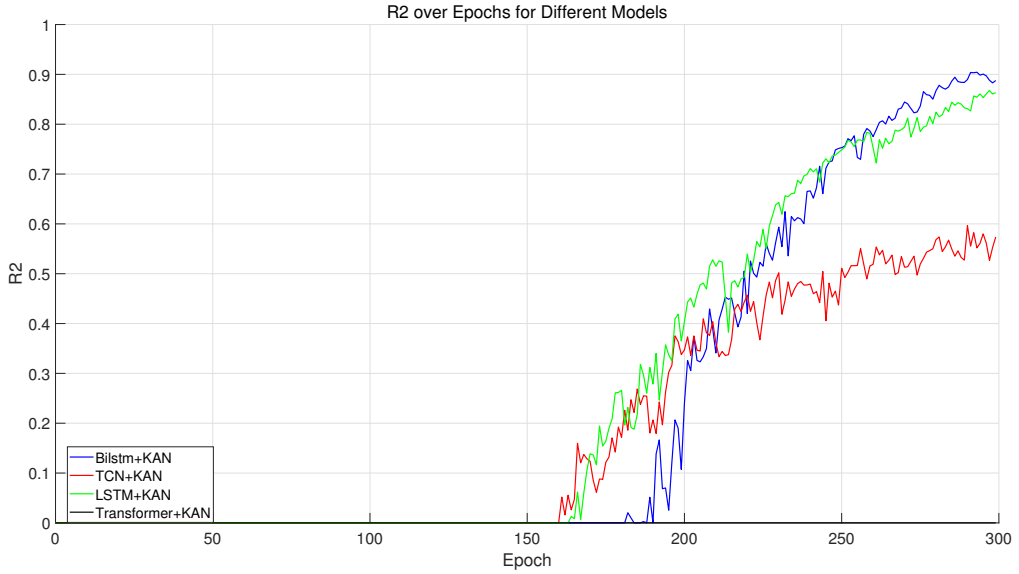


Figure 44. R^2 over epochs for different models (TCN, LSTM, and transformer) with KAN.

In the prediction results for the X, Y, and Z coordinates as shown in Figure 45, the model demonstrated a high consistency with the actual motion trajectories. Although there were slight overreactions in predicting some peaks and troughs, this sensitivity actually underscores the BILSTM-KAN's robust capability in capturing dynamic changes.

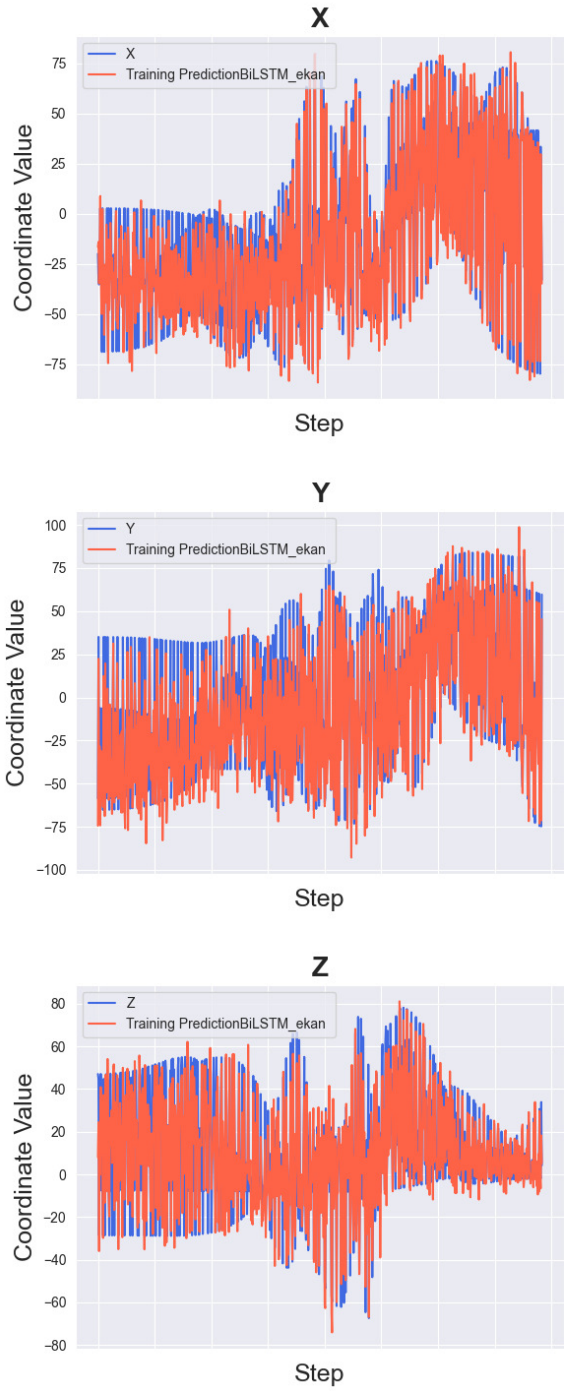


Figure 45. Training predictions for BILSTM with KAN on X, Y, and Z coordinates.

From the graphs, it is evident that the predicted trajectories closely follow the observed values, particularly in maintaining the overall motion trends. While there are minor fluctuations between the predicted and actual values in the latter half of the sequence, these remain within an acceptable error margin. These fluctuations are primarily due to the model's heightened sensitivity to complex changes in the data over short periods, highlighting its potential for predicting fine dynamics.

5. Conclusions

In this study, we explored the application of deep learning technologies in vision servo systems, particularly in target recognition and trajectory prediction. Initially, we

employed the DETR model for efficient target detection using the VOC2007 dataset. The DETR model, with its transformer-based structure, offers an end-to-end solution for target recognition, eliminating the complex post-processing steps like non-maximum suppression that are common in traditional target detection algorithms. Experimental results showed an mAP score of 84.47, which is higher than related papers in recent years that also used the VOC2007 dataset.

To generate a suitable trajectory dataset for training machine learning models, we first modeled the robotic arm based on DH parameters and calculated its workspace. We then used the PSO algorithm and 3-5-3 polynomial interpolation to optimize the motion trajectories of the robot arm. The PSO algorithm simulates group behavior during the search process, sharing information between individuals to find the optimal solution, thereby creating smooth and feasible motion paths.

A key innovation in this study was the application of the BILSTM-KAN model to predict the robot's end position. This model integrates BILSTM with the Kolmogorov–Arnold network, the latter being a novel network structure designed to replace the MLP. The KAN significantly improves network parameter efficiency and model interpretability by placing learnable activation functions on fixed node activation functions. This combined model not only captures long-term dependencies in time series data but also enhances prediction accuracy and efficiency through its optimized network structure. The experiments revealed lower MSE scores and higher R^2 scores compared to traditional time series predictions and other models combined with the KAN.

Finally, the robotic arm was modeled and simulated in Gazebo within ROS, validating the algorithms. The experimental results demonstrated the system's rapid ability to locate and track target objects.

6. Possibilities and Limitations in Industrial Applications

The methods and techniques developed in this study have significant prospects for improving the operational capabilities of industrial robots, especially in fields such as manufacturing, assembly, and quality control that require high precision and adaptability. The integration of deep learning techniques, such as the DETR model for object recognition and the BILSTM-KAN model for trajectory prediction, lays the foundation for developing more autonomous, efficient, and flexible robot systems. These systems have the potential to reduce human errors, improve productivity, and adapt faster to changes in new tasks or operating environments.

However, several limitations must be addressed to fully realize the potential of these advances in industrial scenarios. Firstly, simulated environments often cannot explain the variability of sensor calibration and mechanical alignment in real-world settings. For example, due to sensor drift or wear, there may be differences in joint angle readings of robot arms, which are usually not modeled in simulations. The internal parameters of the camera, such as focal length, principal point position, and lens distortion coefficient, may vary in actual environments due to factors such as temperature changes and mechanical vibrations.

Secondly, the robustness of these models in various unpredictable industrial environments presents another challenge. Industrial environments typically involve interactions with various objects and materials under different lighting and environmental conditions. As shown in this study, models trained in controlled or simulated environments may not run as expected when directly transferred to real-world settings. Adaptation to real-world changes requires extensive testing and may require the retraining or fine-tuning of models with real-world data, which may require significant resources.

Although this article validated the performance of various algorithms and integrated them into the control system, no data were collected to measure and evaluate the overall performance improvement. In future work, we will focus on the impact of performing this on the entire control system and make improvements.

7. Further Work

As we conclude our findings, it becomes evident that, while significant strides have been made in the application of deep learning to visual servoing, there are several avenues for enhancement and optimization. Herein, we outline potential future research directions that could address current limitations and expand the capabilities of our models:

- **Improving Computational Efficiency with KAN:** The integration of KANs has proven to be beneficial for enhancing model functionality by facilitating the handling of complex nonlinear relationships. However, this integration has led to increased computational demands. To address this, future investigations could focus on optimizing the structure of KAN layers, perhaps by minimizing the complexity where feasible without significantly compromising performance. Further research could also explore the potential of parallel processing techniques and algorithmic improvements such as network pruning and sparse connectivity. These strategies are anticipated to reduce computational overhead, thus speeding up both the training and inference phases significantly.
- **Enhancing Recognition of Small Objects in Complex Backgrounds:** Recognizing small objects within complex backgrounds remains a substantial challenge due to the intricate nature of the visual data and limited visibility. Future work could enhance feature extraction capabilities through advanced techniques like scale-invariant feature transforms or employing deep learning architectures specifically tailored for small object detection, such as YOLOv4 or SSD. Additionally, specialized data augmentation methods that simulate small object scenarios in cluttered environments could be developed to train the system more effectively. These methods would prepare the model to perform with higher accuracy and reliability in real-world settings where small object recognition is crucial.
- **Transitioning from Simulation to Real-World Applications:** To effectively validate the practical utility of the model in real-world scenarios, we plan to conduct a series of progressively advancing experiments to ensure that the model's performance in actual operations aligns with the results obtained in simulations. This process will be divided into three main stages:
  - **Preliminary Validation Stage: Closed Testing in a Laboratory Environment**
    In this stage, we will set up a physical testing platform in a laboratory environment to simulate real-world operating conditions. Specifically, we will use high-precision cameras and industrial robotic arms to simulate various possible scenarios, such as lighting changes, camera noise, and dynamic obstacles.
    We will first test the model's basic task execution capabilities under these conditions, such as object recognition, trajectory tracking, etc. Through data collection, we will real-time gather and analyze the model's performance differences across different scenarios.
  - **Initial Deployment Stage: Semi-Realistic Scenario Testing in a Controlled Environment**
    After laboratory validation, we will deploy the model in a controlled but more realistic environment, such as an industrial production line or automated warehouse. This stage of testing will introduce more complex variables, such as objects of different materials, noise interference in the environment, and unpredictable moving objects.
    We plan to combine operational data to monitor the model's performance under different conditions and use these data to fine-tune and enhance the model. Simultaneously, we will collaborate with on-site operators to gather their feedback, assessing the model's practicality and ease of use.
  - **Field Validation Stage: Comprehensive Testing in Real-World Applications**
    Finally, we will conduct comprehensive field tests in real-world application scenarios, which may include smart manufacturing, autonomous driving, or

unmanned warehousing. At this stage, key performance indicators (e.g., task success rate, response time, system stability, etc.) will become the core criteria for evaluating whether the model has the potential for large-scale application.

Author Contributions: Conceptualization, Z.H. and B.H.S.A.; methodology, Z.H., D.Z., and B.H.S.A.; resources, B.H.S.A. and Z.H.; writing—original draft preparation, B.H.S.A. and Z.H.; writing—review and editing, B.H.S.A. and Z.H.; supervision, B.H.S.A.; visualization, B.H.S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chaumette, F.; Hutchinson, S. Visual servo control. I. Basic approaches. *IEEE Robot. Autom. Mag.* **2006**, *13*, 82–90. [CrossRef]
2. Hutchinson, S.; Hager, G.; Corke, P. A tutorial on visual servo control. *IEEE Trans. Robot. Autom.* **1996**, *12*, 651–670. [CrossRef]
3. Shi, H.; Wu, H.; Xu, C.; Zhu, J.; Hwang, M.; Hwang, K.S. Adaptive Image-Based Visual Servoing Using Reinforcement Learning With Fuzzy State Coding. *IEEE Trans. Fuzzy Syst.* **2020**, *28*, 3244–3255. [CrossRef]
4. Zhu, N.; Xie, W.F.; Shen, H. Position-Based Visual Servoing of a 6-RSS Parallel Robot Using Adaptive Sliding Mode Control. *ISA Trans.* **2024**, *144*, 398–408. [CrossRef]
5. Gubbi, M.R.; Lediju Bell, M.A. Deep Learning-Based Photoacoustic Visual Servoing: Using Outputs from Raw Sensor Data as Inputs to a Robot Controller. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; pp. 14261–14267. [CrossRef]
6. Thuilot, B.; Martinet, P.; Cordesses, L.; Gallice, J. Position Based Visual Servoing: Keeping the Object in the Field of Vision. In Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292), Washington, DC, USA, 11–15 May 2002; IEEE: Piscataway, NJ, USA, 2002; Volume 2, pp. 1624–1629. [CrossRef]
7. Martinet, P.; Gallice, J. Position Based Visual Servoing Using a Non-linear Approach. In Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289), Kyongju, Republic of Korea, 17–21 October 1999; IEEE: Piscataway, NJ, USA, 1999; Volume 1, pp. 531–536. [CrossRef]
8. Dong, G.; Zhu, Z. Position-Based Visual Servo Control of Autonomous Robotic Manipulators. *Acta Astronaut.* **2015**, *115*, 291–302. [CrossRef]
9. Park, D.H.; Kwon, J.H.; Ha, I.J. Novel Position-Based Visual Servoing Approach to Robust Global Stability Under Field-of-View Constraint. *IEEE Trans. Ind. Electron.* **2012**, *59*, 4735–4752. [CrossRef]
10. Lippiello, V.; Siciliano, B.; Villani, L. Position-Based Visual Servoing in Industrial Multirobot Cells Using a Hybrid Camera Configuration. *IEEE Trans. Robot.* **2007**, *23*, 73–86. [CrossRef]
11. Parsapour, M.; RayatDoost, S.; Taghirad, H.D. Position Based Sliding Mode Control for Visual Servoing System. In Proceedings of the 2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM), Tehran, Iran, 13–15 February 2013; pp. 337–342. [CrossRef]
12. Ribeiro, E.G.; Mendes, R.Q.; Terra, M.H.; Grassi, V. Second-Order Position-Based Visual Servoing of a Robot Manipulator. *IEEE Robot. Autom. Lett.* **2024**, *9*, 207–214. [CrossRef]
13. Deng, L. Comparison of Image-Based and Position-Based Robot Visual Servoing Methods and Improvements. Ph.D. Thesis, University of Waterloo, Waterloo, ON, Canada, 2004.
14. Yang, K.; Bai, C.; She, Z.; Quan, Q. High-Speed Interception Multicopter Control by Image-Based Visual Servoing. *arXiv* **2024**, arXiv:2404.08296.
15. Albekairi, M.; Mekki, H.; Kaaniche, K.; Yousef, A. An Innovative Collision-Free Image-Based Visual Servoing Method for Mobile Robot Navigation Based on the Path Planning in the Image Plan. *Sensors* **2023**, *23*, 9667. [CrossRef]
16. Zhang, Y.; Yang, Y.; Luo, W. Occlusion-free Image-Based Visual Servoing using Probabilistic Control Barrier Certificates. *IFAC-PapersOnLine* **2023**, *56*, 4381–4387. [CrossRef]
17. Zhu, T.; Mao, J.; Han, L.; Zhang, C. Fuzzy Adaptive Model Predictive Control for Image-Based Visual Servoing of Robot Manipulators with Kinematic Constraints. *Int. J. Control Autom. Syst.* **2024**, *22*, 311–322. [CrossRef]
18. Peng, X.; Li, J.; Li, B.; Wu, J. Constrained Image-Based Visual Servoing of Robot Manipulator with Third-Order Sliding-Mode Observer. *Machines* **2022**, *10*, 465. [CrossRef]

19. Ramani, P.; Varghese, A.; Balachandar, N. Image Based Visual Servoing for Tele-Operated Ground Vehicles. *AIP Conf. Proc.* **2024**, *2802*, 110001. [CrossRef]
20. Tsai, D.; Dansereau, D.G.; Peynot, T.; Corke, P. Image-Based Visual Servoing With Light Field Cameras. *IEEE Robot. Autom. Lett.* **2017**, *2*, 912–919. [CrossRef]
21. McFadyen, A.; Jabeur, M.; Corke, P. Image-Based Visual Servoing With Unknown Point Feature Correspondence. *IEEE Robot. Autom. Lett.* **2017**, *2*, 601–607. [CrossRef]
22. Harish, Y.V.S.; Pandya, H.; Gaud, A.; Terupally, S.; Shankar, S.; Krishna, K.M. DFVS: Deep Flow Guided Scene Agnostic Image Based Visual Servoing. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 9000–9006. [CrossRef]
23. Machkour, Z.; Ortiz-Arroyo, D.; Durdevic, P. Classical and Deep Learning Based Visual Servoing Systems: A Survey on State of the Art. *J. Intell. Robot. Syst.* **2022**, *104*, 11. [CrossRef]
24. Rekavandi, A.M.; Rashidi, S.; Boussaid, F.; Hoefs, S.; Akbas, E.; bennamoun, M. Transformers in Small Object Detection: A Benchmark and Survey of State-of-the-Art. *arXiv* **2023**, arXiv:2309.04902.
25. Terven, J.; Córdova-Esparza, D.M.; Romero-González, J.A. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 1680–1716. [CrossRef]
26. Zhang, H.; Li, F.; Liu, S.; Zhang, L.; Su, H.; Zhu, J.; Ni, L.M.; Shum, H.Y. DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection. *arXiv* **2022**, arXiv:2203.03605.
27. El Ahmar, W.; Massoud, Y.; Kolhatkar, D.; AlGhamdi, H.; Alja'afreh, M.; Hammoud, R.; Laganiere, R. Enhanced Thermal-RGB Fusion for Robust Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Vancouver, BC, Canada, 18–22 June 2023; pp. 365–374.
28. Reis, D.; Kupec, J.; Hong, J.; Daoudi, A. Real-Time Flying Object Detection with YOLOv8. *arXiv* **2024**, arXiv:2305.09972.
29. Wang, Z.; Li, Y.; Chen, X.; Lim, S.N.; Torralba, A.; Zhao, H.; Wang, S. Detecting Everything in the Open World: Towards Universal Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 11433–11443.
30. Katara, P.; Harish, Y.V.S.; Pandya, H.; Gupta, A.; Sanchawala, A.; Kumar, G.; Bhowmick, B.; Krishna, M. DeepMPCVS: Deep Model Predictive Control for Visual Servoing. In Proceedings of the 2020 Conference on Robot Learning, Virtual, 16–18 November 2020; Volume 155, pp. 2006–2015.
31. Li, J.; Peng, X.; Li, B.; Sreeram, V.; Wu, J.; Chen, Z.; Li, M. Model Predictive Control for Constrained Robot Manipulator Visual Servoing Tuned by Reinforcement Learning. *Math. Biosci. Eng.* **2023**, *20*, 10495–10513. [CrossRef]
32. Fu, G.; Chu, H.; Liu, L.; Fang, L.; Zhu, X. Deep Reinforcement Learning for the Visual Servoing Control of UAVs with FOV Constraint. *Drones* **2023**, *7*, 375. [CrossRef]
33. Lee, A.X.; Levine, S.; Abbeel, P. Learning Visual Servoing with Deep Features and Fitted Q-Iteration. *arXiv* **2017**, arXiv:1703.11000.
34. Adrian, N.; Do, V.T.; Pham, Q.C. DFBVS: Deep Feature-Based Visual Servo. In Proceedings of the 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), Mexico City, Mexico, 22–26 August 2022; pp. 1783–1789. [CrossRef]
35. Liu, J.; Li, Y. An Image Based Visual Servo Approach with Deep Learning for Robotic Manipulation. *arXiv* **2019**, arXiv:1909.07727.
36. He, Y.; Gao, J.; Chen, Y. Deep Learning-Based Pose Prediction for Visual Servoing of Robotic Manipulators Using Image Similarity. *Neurocomputing* **2022**, *491*, 343–352. [CrossRef]
37. Lazo, J.F.; Lai, C.F.; Moccia, S.; Rosa, B.; Catellani, M.; de Mathelin, M.; Ferrigno, G.; Breedveld, P.; Dankelman, J.; De Momi, E. Autonomous Intraluminal Navigation of a Soft Robot using Deep-Learning-Based Visual Servoing. In Proceedings of the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 23–27 October 2022; pp. 6952–6959. [CrossRef]
38. Copot, C.; Shi, L.; Smet, E.; Ionescu, C.; Vanlanduit, S. Comparison of Deep Learning Models in Position Based Visual Servoing. In Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany, 6–9 September 2022; pp. 1–4. [CrossRef]
39. Abdulhafiz, I.; Nazari, A.A.; Abbasi-Hashemi, T.; Jalali, A.; Zareinia, K.; Saeedi, S.; Janabi-Sharifi, F. Deep Direct Visual Servoing of Tendon-Driven Continuum Robots. In Proceedings of the 2022 IEEE 18th International Conference on Automation Science and Engineering (CASE), Mexico City, Mexico, 22–26 August 2022; pp. 1977–1984. [CrossRef]
40. Jin, Z.; Wu, J.; Liu, A.; Zhang, W.A.; Yu, L. Policy-Based Deep Reinforcement Learning for Visual Servoing Control of Mobile Robots With Visibility Constraints. *IEEE Trans. Ind. Electron.* **2022**, *69*, 1898–1908. [CrossRef]
41. Felton, S.; Fromont, E.; Marchand, E. Deep Metric Learning for Visual Servoing: When Pose and Image Meet in Latent Space. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 741–747. [CrossRef]
42. Asayesh, S.; Darani, H.S.; Chen, M.; Mehrandezh, M.; Gupta, K. Toward Scalable Visual Servoing Using Deep Reinforcement Learning and Optimal Control. *arXiv* **2023**, arXiv:2310.01360.
43. Everingham, M.; Van Gool, L.; Williams, C.K.I.; Winn, J.; Zisserman, A. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. Available online: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html (accessed on 20 July 2024).
44. Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. End-to-End Object Detection with Transformers. *arXiv* **2020**, arXiv:2005.12872.

45. Wang, W.; Tao, Q.; Cao, Y.; Wang, X.; Zhang, X. Robot Time-Optimal Trajectory Planning Based on Improved Cuckoo Search Algorithm. *IEEE Access* **2020**, *8*, 86923–86933. [CrossRef]
46. Liu, Z.; Wang, Y.; Vaidya, S.; Ruehle, F.; Halverson, J.; Soljačić, M.; Hou, T.Y.; Tegmark, M. KAN: Kolmogorov-Arnold Networks. *arXiv* **2024**, arXiv:2404.19756.
47. Cao, J.; Pang, Y.; Han, J.; Li, X. Hierarchical Shot Detector. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019.
48. Zhu, Y.; Zhao, C.; Wang, J.; Zhao, X.; Wu, Y.; Lu, H. CoupleNet: Coupling Global Structure With Local Parts for Object Detection. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017.
49. Termritthikun, C.; Jamtsho, Y.; Ieamsaard, J.; Muneesawang, P.; Lee, I. EEEA-Net: An Early Exit Evolutionary Neural Architecture Search. *Eng. Appl. Artif. Intell.* **2021**, *104*, 104397. [CrossRef]
50. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In *Computer Vision—ECCV 2016*; Springer: Cham, Switzerland, 2016; pp. 21–37.
51. Dvornik, N.; Shmelkov, K.; Mairal, J.; Schmid, C. BlitzNet: A Real-Time Deep Network for Scene Understanding. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.



https://dspace.org/

Motion prediction and object detection for image-based visual servoing systems using deep learning

Hao, Zhongwen; Zhang, Deli; Honarvar Shakibaei Asli, Barmak

2024-09-01
MDPI
https://doi.org/10.3390/electronics13173487
Hao Z, Zhang D, Honarvar Shakibaei Asli B. (2024) Motion prediction and object detection for image-based visual servoing systems using deep learning. Electronics, Volume 13, Issue 17, September 2024, Article number 3487

Downloaded from a DSpace repository. https://dspace.org/