The Iranian Journal of Mechanical Engineering Transactions of ISME
Journal homepage: https://jmee.isme.ir/
Vol. 27, No. 2, 2025, pp. 6-34

Research Paper

DOI: https://doi.org/10.30506/jmee.2025.2037549.1353

Integrating Adaptive Sliding Mode Control and Deep Learning for Autonomous Vision-driven Fruit Sorting Robot

Hassan Sayyaadi[*]
Professor

Sara Adeli[†]
M.Sc. Student

This article presents a novel application of deep learning in automated fruit-sorting robotics, improving real-time object recognition and handling. By integrating advanced neural networks, the robot achieves higher accuracy in identifying various fruits, addressing fruit variability and enhancing sorting precision. This innovation, combining deep learning and visual servoing, represents a significant advancement in automated fruit-sorting technology, with promising benefits for agricultural processes. The project aims to control a fruit-sorting robot using image processing data, merging sliding mode control and NN-based (neural network-based) techniques for automation. Utilizing the latest YOLO (You Only Look Once) model, the system classifies and positions fruits rapidly and accurately, making it suitable for real-time applications. After identifying fruit types and positions, a controller is designed for the pick-and-place process. Sliding mode control manages uncertainties and guides manipulator movements precisely, while a neural network controls joint angles for smooth and accurate fruit manipulation. Comparative tests on a simulated robot revealed that the NN-based controller excels in accuracy and speed, adapting to different fruit configurations effectively. The sliding mode controller, though robust and stable, is sensitive to uncertainties, affecting sorting precision. The hybrid system, integrating both controllers, enhances adaptability by combining the NN-based approach's precision with the stability of sliding mode control, optimizing fruit-sorting performance across diverse scenarios. Results emphasize selecting the appropriate controller to balance precision and speed based on specific application needs.

Keywords: Fruit detection, NN-based control, Adaptive sliding mode control, Visual servoing

[*] Corresponding Author, Professor, Department of Mechanical Engineering, Sharif University of Technology, Tehran, Iran, sayyaadi@sharif.edu

[†] M.Sc., Student, Department of Mechanical Engineering, Sharif University of Technology, Tehran, Iran, Saraadeli20001378@ yahoo.com

Received: 2024/08/04, Revised: 2024/01/25, Accepted: 2024/04/30

1 Introduction

In recent years, the intersection of robotics and artificial intelligence has paved the way for innovative solutions in various fields, including agriculture and automation. One such groundbreaking application is the development of a "Deep Learning-based Enhanced Visual Servoing System" for automated fruit sorting robots. Traditional fruit sorting methods often rely on manual labor or rule-based systems, which can be time-cons
uming and prone to errors. The integration of deep learning techniques into visual servoing, a method used to control the motion of a robot based on visual feedback, presents an opportunity to revolutionize the efficiency and accuracy of fruit sorting processes.

Nowadays, the applications of the visual servoing system are very wide, including research and industrial applications. In their study, Kumar and his colleagues [1], developed an image processing algorithm designed for a robot to classify electronic components such as capacitors and resistors. This work utilizes a feature extraction algorithm in conjunction with an Artificial Neural Network (ANN) classifier. The system outputs the object type along with its coordinates. The results demonstrate that the feature extraction accuracy is 83.64%, the classifier accuracy is 99.33%, and the overall system accuracy, combining feature extraction and classification, is 82.72%. Similarly, in another study, B. Iscimen and his team [2], focused on classifying and removing dining-related objects from a table. Despite the greater variety of objects examined, this model achieved an accuracy of 98.3%, outperforming the previous study. The improved accuracy can be attributed to the diversity of the extracted features, which enhanced the model's performance. Naoshi Kondo has made a robot for fruit grading system using a visual system composed of color cameras and lighting equipment [3]. The system uses multiple color cameras to check the target in all directions and suck the apple through the sucker. In 2016, M.M.Sofo released a real-time apple sorting system that can distinguish multiple apples based on color, size and composition. The machine achieved the efficiency of sorting 15 apples per second in real time using two channels [4]. In [5], the application of a 4DOF fruit sorting robot based on color and size in a packaging system is presented. The sorting is made possible by image processing where color is recognized by HSV (Hue, Saturation, and Value color model) analysis, and the diameter is known in the grayscale image and setting the thresholding. The fruit to be sorted is red and green tomatoes and red and green grapes. The experiments were conducted to show the effectiveness of the proposed method. In another study, a novel decoupled image-based visual servoing is proposed to control a 4-DOF robot arm for grasping products [6]. Area, orientation angle, and centroid features extracted from the image are used as input to control the velocities of the robot arm. A multi-threshold algorithm is presented to detect and classify objects and extract the image features. Another research [7] proposes a robotic vision system that distinguishes the color of objects and their position coordinates, then sorts the objects (products) onto the correct branch of the conveyor belt according to their color in real-time. The system was built based on the HVS mode algorithm for sorting product based on color. Furthermore, the system can distinguish the shape of the object, determine its position, pick up the object, and place it on the correct branch of the conveyor belt. In [8], with the aim of sorting damaged and healthy fruits, with the help of extracting three features of edge, color and texture by Sobel filter, HSV color space and entropy filter, and according to the color difference in the damaged parts, the classification has been done. In [9], similar to the previous article, HSV is used to detect defective objects, but by replacing the Faster R-CNN network instead of CNN (Convolutional Neural Network), the processing time is less than the previous work and the detection accuracy is higher. In [10], the HSV method has shown good accuracy in modeling, but in reality, the accuracy has decreased due to the presence of environmental noise.

In the field of control, the study indicates that decentralized controllers are commonly used in industrial applications due to their simplicity and generally acceptable performance [11]. However, for applications requiring minimal tracking error, neural network-based controllers are preferred. Given the inherent inaccuracies in the dynamic models of robots, neural networks are essential for precise path tracking. Neural networks can either learn the entire system dynamics or compensate for model uncertainties. In this study, training is conducted using the back-propagation algorithm, based on the difference between actual and estimated torque. The results demonstrate that employing neural networks significantly reduces tracking error compared to other controllers.

In [12], the accurate and rapid automatic assembly of fragile fuel cell components into large stacks presents a significant challenge. High-voltage fuel cells require the precise assembly of thousands of delicate parts, where any misalignment can cause short circuits. To address this, various neural networks with different architectures were developed to solve a regression problem, modeling deviations in the pick-and-place process to ensure precision and robustness. Practical tests revealed an overall tolerance of 0.254 mm in the pick-and-place cycle, slightly exceeding the required 0.25 mm tolerance, indicating an accuracy issue with the gripper mechanism. An ANN was developed using training data from these tests, with inputs including the detected position of the fuel cell components (x, y, z) and their length, and outputs being the placement deviations. The optimal network, featuring three hidden layers and 40 neurons, showed the lowest generalization error. Implementing this network on the robot improved the accuracy of the assembly process. In [13], a delta robot utilizing an Artificial Neural Network is designed for pick-and-place, packing, and assembly tasks, aiming to achieve high accuracy at minimal cost. The ANN controlling the robot comprises 6 input neurons, 12 hidden neurons, and 3 output neurons. The network inputs are the object's position in three directions (x, y, z) and the error variables for each direction, while the outputs correspond to the required rotations in the robot's three joints.

This research focuses on leveraging the power of deep learning algorithms, such as convolutional neural networks, to enhance the visual perception capabilities of a fruit-sorting robot. By training the system on large datasets of diverse fruit images, the robot can learn to detect fruits based on their size, shape, color, and other relevant features. The integration of NN-based control into the visual servoing framework enables the robot to dynamically adapt its movements in real-time, improving its ability to grasp and manipulate fruits with precision. The proposed Deep Learning-based Enhanced Visual Servoing System aims to address the challenges associated with conventional fruit sorting methods by providing a more intelligent and adaptable solution. Through in this research, the goal is to contribute to the advancement of automation in agriculture, ultimately optimizing fruit sorting processes, reducing labor costs, and increasing overall efficiency in the agricultural industry.

In the following, section (2) presents the methodology, including preparing dataset for fruit detection model, training and verification of YOLOv9 model with prepared dataset, motion planning using LSPB (Linear Segments with Parabolic Blends) method, derivation of dynamic equations and state space form of PUMA 560 robot, adaptive sliding mode control design, and NN-based control of PUMA robot. In section (3), the model validation and the results are analyzed and discussed. Finally, in section 4, all the conclusions are provided.

2 Material and methods

This project utilizes the YOLOv9 algorithm to classify and locate four types of fruits—tomatoes, strawberries, plums, and peaches. The goal is to control the PUMA 560 robot, which has six degrees of freedom, using two methods: adaptive sliding mode and NN-based control. The entire process is simulated to evaluate performance.

2.1 Dataset preparation for fruit detection model

First, a suitable dataset must be prepared for training and testing. The dataset for this project comprises images of tomatoes, strawberries, plums, and peaches. Specifically, 84 images of tomatoes, 95 images of strawberries, 87 images of plums, 75 images of peaches, and 18 mixed images of these fruits were collected from the Internet, totaling 359 images. These images were augmented using techniques such as rotation, and cropping, increasing the dataset size to 380 images. Of these, 258 images are used as training data, while 122 images are reserved for testing. Since the YOLOv9 algorithm is a supervised learning method, the collected images need to be labeled, indicating the type and position of the fruits in each image. We used a software called CVAT to label the images, accessible online at cvat.ai. To label the images, a new task is created on the site, and the images are uploaded. For each image, we select the fruit type from the four available labels—tomato, strawberry, plum, and peach—and draw a bounding box around the fruit. The software then generates an .xml file for each image.

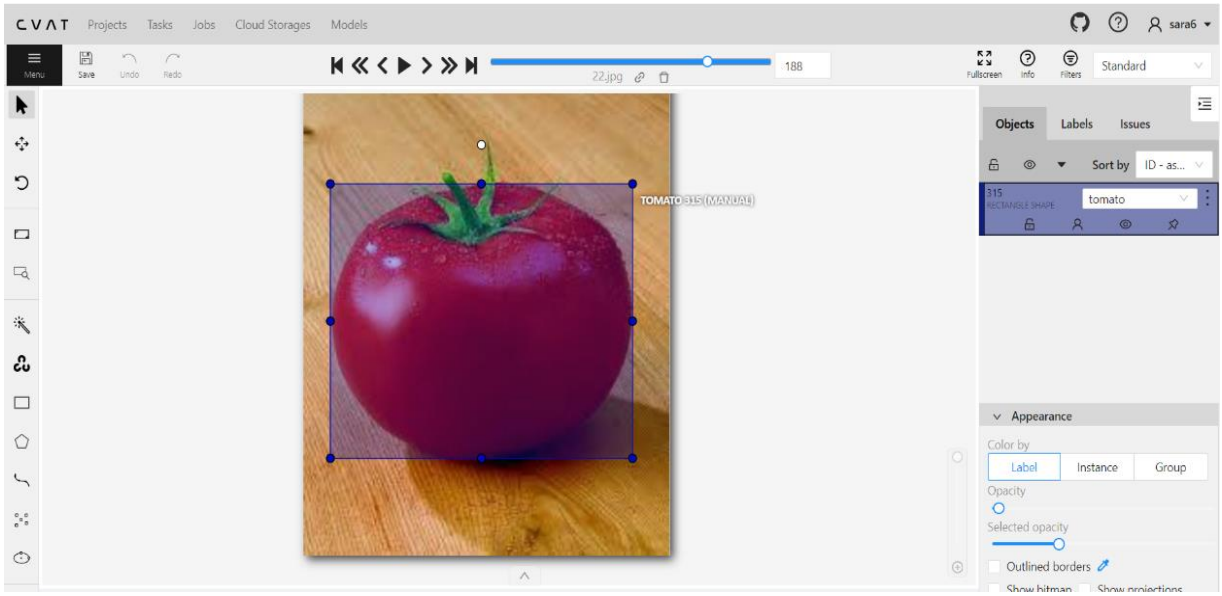


Figure 1 Tomato labeling in CVAT (type determination and bounding box drawing)

Figure 2 The dataframe contains detailed information about the dataset

	name	label	width	height	xmin	ymin	xmax	ymax	class
0	1m (10)	plum	265	190	364.679245	184.421053	562.716981	368.842105	2
1	1m (10)	plum	265	190	159.396226	202.105263	338.113208	373.894737	2
2	1m (10)	peach	265	190	193.207547	73.263158	384.000000	224.842105	3
3	1m (1)	strawberry	275	184	384.000000	193.043478	507.345455	331.304348	1
4	1m (1)	strawberry	275	184	549.236364	213.913043	640.000000	352.173913	1
...	...	...	...	...	...	...	...	...	...
470	69	strawberry	259	195	427.490347	145.230769	632.586873	354.461538	1
471	69	strawberry	259	195	256.988417	118.153846	442.316602	312.615385	1
472	68	strawberry	275	183	44.218182	94.426230	323.490909	424.918033	1
473	63	strawberry	275	183	51.200000	57.704918	549.236364	474.754098	1
474	65	strawberry	247	204	75.141700	89.411765	479.352227	400.000000	1

475 rows × 9 columns

This file contains information such as the fruit type, image dimensions (width and height), and the coordinates of the bounding box (x_{min}, x_{max}, y_{min}, y_{max}). This process allows the images to serve as features and the .xml files to provide the necessary labels for classification and positioning. An illustration of tomato image labeling in CVAT is shown in Figure (1).
Next, we parse the .xml files using a for loop to extract information such as the image name, corresponding label, image width and height, and the minimum and maximum x and y coordinates of the bounding box. This data is then stored in a dataframe, with the corresponding class labels assigned as 0 for tomato, 1 for strawberry, 2 for plum, and 3 for peach, as shown in Figure (2). Also a total of 474 fruits were detected across 380 images; This higher count is due to some images containing more than one fruit. In this project, all images have been resized to a width of 640 pixels and a height of 480 pixels to optimize them for algorithm training. The dataset has been split into training and validation sets with an 85% to 15% ratio, respectively.

2.2 Training and verification of YOLOv9 model with prepared dataset

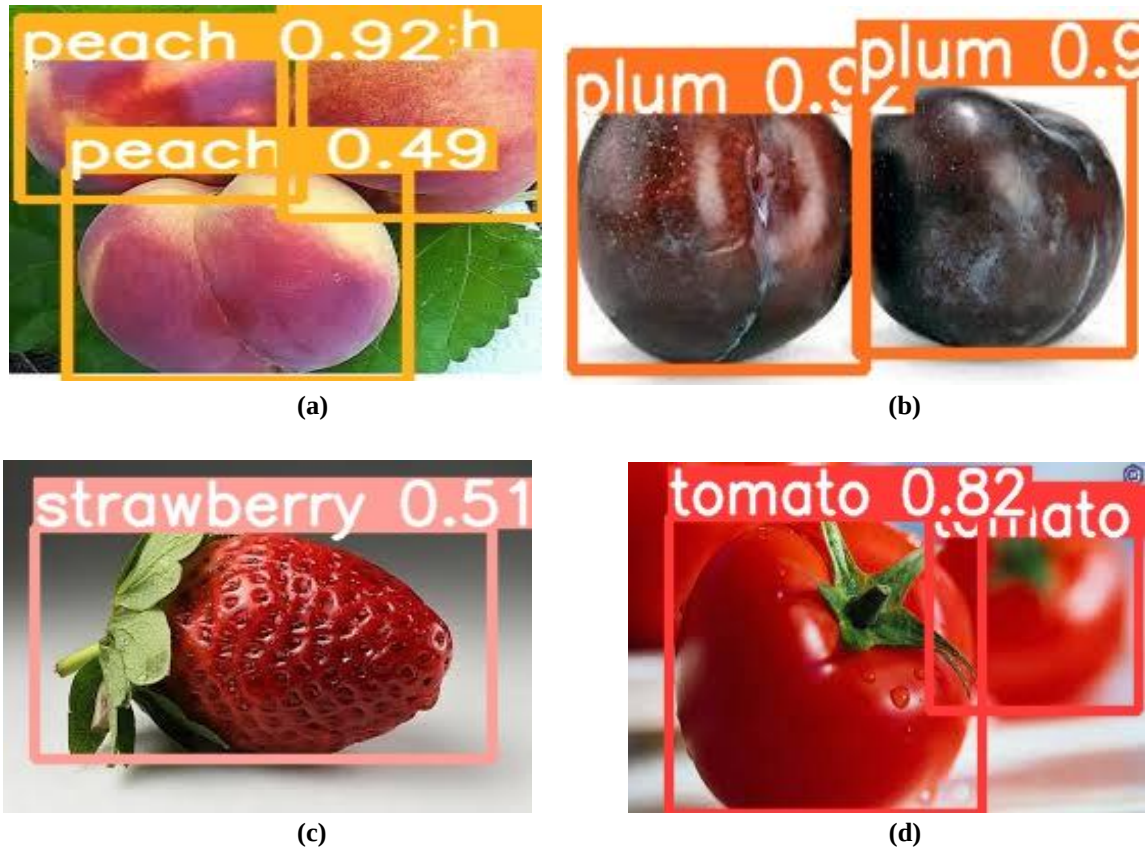
YOLOv9 is an advanced object detection algorithm widely used in machine vision and image processing. Renowned for its efficiency in real-time object detection, YOLOv9 builds upon the strengths of its predecessors, particularly YOLOv8, offering enhanced performance and accuracy. YOLOv9 utilizes a CNN-based neural network architecture tailored for effective object detection and localization in images. The architecture comprises three key components: Backbone, Neck, and Head. The Backbone of YOLOv9 efficiently extracts features from input images using a multi-layered CNN. This process progressively reduces spatial dimensions while enhancing feature map depth. The Neck component integrates and refines features derived from the Backbone to enhance object detection accuracy. The Head of YOLOv9 handles object classification and localization tasks, ultimately generating predictions in the form of bounding boxes and class probabilities [14].
The architecture of YOLOv9 is meticulously designed to push the boundaries of efficiency and accuracy in object detection. Its modular design incorporates distinct components for the backbone, neck, and detection head, each optimized for advanced feature extraction, multi-scale fusion, and precise detection. At its core, the model features the CSPDarkNet backbone, composed of 53 convolutional layers. This backbone leverages CSP (Cross Stage Partial) networks, which split feature maps into two parts, one undergoes transformation, while the other bypasses processing before merging, enhancing gradient flow, reducing redundant computations, and improving model representation capabilities.
The backbone begins with initial convolutional layers tailored to extract primary features. Layer 0 applies a convolutional operation with 3 input channels, 64 output channels, a kernel size of 3×3, and a stride of 2, while Layer 1 processes 64 input channels to produce 128 output channels with the same kernel size and stride. It further integrates RepNCSPELAN4 modules, which progressively downsample and refine features through configurations of input-output channels, residual paths, and base channels. These modules balance efficiency and accuracy in feature extraction. Additionally, Aggregated Downsampling layers are included for aggregated downsampling, maintaining spatial hierarchies across resolutions essential for detecting objects at various scales.
YOLOv9's neck integrates a PAN (Path Aggregation Network), which improves the bidirectional flow of information between scales and enhances multi-scale feature fusion by supplementing traditional bottom-up and top-down approaches. This is complemented by three levels of FPN (Feature Pyramid Network) layers, which combine features of varying resolutions to merge spatial detail with semantic information. Fusion techniques like concatenate layers and upsampling align and combine feature maps effectively across stages.
The detection head delivers precise predictions using anchor-free mechanisms and supports multi-scale outputs through DDetect modules with feature channel configurations of [256, 512,

512]. These modules eliminate the need for predefined anchor boxes, streamlining predictions while still leveraging effective loss functions like Sigmoid Cross Entropy Loss for classification and CIoU Loss for bounding box regression. The head supports four object classes (nc=4), with anchor alignment ensuring compatibility across feature maps.

In terms of computational efficiency, YOLOv9 operates with 621 layers, a parameter count of 25,440,156, and a computational complexity of 103.2 GFLOPs. The training process benefits from hyperparameter optimization, including a learning rate initialized at 0.01 and annealed via a cosine decay scheduler. A batch size of 64, weight decay of 0.0005, and momentum of 0.9 are set to ensure convergence and robust generalization. Regularization through Dropblock (rate = 0.1) reduces overfitting, while augmentations such as Mosaic, MixUp, and HSV diversify the training data.

The model incorporates advanced techniques like QAT (Quantization-Aware Training) to prepare for efficient deployment in edge environments. QAT anticipates the numerical precision constraints of edge devices, balancing accuracy and lightweight operation. During inference, NMS (Non-Maximum Suppression) with an IoU (Intersection over Union) threshold of 0.6 filters overlapping predictions for high-quality detections. The training resolution is set at 640 × 640 pixels, offering an optimal balance between computational demand and spatial detail. Activation functions such as Leaky ReLU (Rectified Linear Unit) or Swish enable effective nonlinear transformations within the model.

By integrating advanced methods like feature concatenation, progressive downsampling, and multi-scale fusion alongside optimizations in hyperparameters and training processes, YOLOv9 achieves a remarkable balance of precision, computational efficiency, and real-time performance. These characteristics make it exceptionally suited for diverse and complex object detection scenarios.



(a) **(b)**

(c) **(d)**

Figure 3(a) Detected peaches and bounding boxes with 92% and 49% confidence **(b)** Detected plum and bounding box with 90% confidence **(c)** Detected strawberry and bounding box with 51% confidence **(d)** Detected tomato and bounding box with 82% confidence.

Leveraging the established model from the literature and the data prepared earlier, we proceed with training. The process involves saving the model's weights and other outcomes. In the subsequent section, we will analyze these results and evaluate the model's performance. Next, utilizing the weights from the trained model, we identify fruits in the test data that the model has not encountered during training. The output for each image includes a bounding box and the identified type of fruit. Figure (3) provides examples of these model predictions, highlighting both the fruit type and the model's confidence in its diagnoses.

2.3 Motion planning using LSPB method

To ensure a seamless transition of the robot from its starting point to the designated endpoint, a meticulous approach governs the motion of its links, prioritizing smooth continuity. Stringent adherence to the prescribed limits of the Puma robot necessitates that the angles, speeds, and accelerations of the links remain well within the permissible range. Facilitating this precision, we employ a triangular acceleration profile, seamlessly integrating it into the speed and position profiles. The motion planning is governed by the following equations for three distinct phases (Eq. (1)-(13)):
Phase 1 ($t < t_j$):

$$a(t) = J_{max} \cdot t \tag{1}$$

$$v(t) = \frac{1}{2} J_{max} \cdot t^2 \tag{2}$$

$$q(t) = q_0 + \frac{1}{6} J_{max} \cdot t^3 \tag{3}$$

In these equations, a represents the angular acceleration, v denotes the angular velocity, and q indicates the angular position of the robot's links. J_{max} corresponds to the maximum allowable jerk, while q_0 and q_1 signify the initial and final positions, respectively. The parameter t_j, known as the jerk-limited time, is calculated using Eq. (4).

$$t_j = \sqrt[3]{\frac{|q_1 - q_0|}{2J_{max}}} \tag{4}$$

Phase 2 ($t_j < t < t_{j1}$):

$$a(t) = J_{max} \cdot t_j - J_{max} \cdot (t - t_j) \tag{5}$$

$$v(t) = \frac{1}{2} J_{max} \cdot t_j^2 + J_{max} \cdot t_j \cdot \left(t - t_j\right) - \frac{1}{2} J_{max} \cdot \left(t - t_j\right)^2 \tag{6}$$

$$q(t) = q_0 + \frac{1}{6} J_{max} \cdot t_j^3 + \frac{1}{2} J_{max} \cdot t_j^2 \cdot \left(t - t_j\right) - \frac{1}{2} J_{max} \cdot t_j \cdot \left(t - t_j\right)^2 - \frac{1}{6} J_{max} \cdot (t - t_j)^3 \tag{7}$$

In these equations t_{j1} can be calculated using Eq. (8) and (9).

$$t_{j1} = t_j + t_k \tag{8}$$

$$t_k = 2t_j \tag{9}$$

Phase 2 ($t_{j1} < t < t_{k1}$):

$$a(t) = -J_{max} \cdot t_j + J_{max} \cdot (t - t_{j1}) \tag{10}$$

$$\begin{aligned} v(t) = {} & \frac{1}{2}J_{max} \cdot t_j^2 + J_{max} \cdot t_j \cdot (t_{j1} - t_j) - \frac{1}{2}J_{max} \cdot (t_{j1} - t_j)^2 - J_{max} \cdot t_j \cdot (t - t_{j1}) \\ & + \frac{1}{2}J_{max} \cdot (t - t_{j1})^2 \end{aligned} \tag{11}$$

$$\begin{aligned} q(t) = {} & q_0 + \frac{1}{6}J_{max} \cdot t_j^3 + \frac{1}{2}J_{max} \cdot t_j^2 \cdot (t_{j1} - t_j) + \frac{1}{2}J_{max} \cdot t_j \cdot (t_{j1} - t_j)^2 - \frac{1}{6}J_{max} \\ & \cdot (t_{j1} - t_j)^3 \\ & + \left[\frac{1}{2}J_{max} \cdot t_j^2 + J_{max} \cdot t_j \cdot (t_{j1} - t_j) - \frac{1}{2}J_{max} \cdot (t_{j1} - t_j)^2\right] \cdot (t - t_{j1}) \\ & - \frac{1}{2}J_{max} \cdot t_j \cdot (t - t_{j1})^2 + \frac{1}{6}J_{max} \cdot (t - t_{j1})^3 \end{aligned} \tag{12}$$

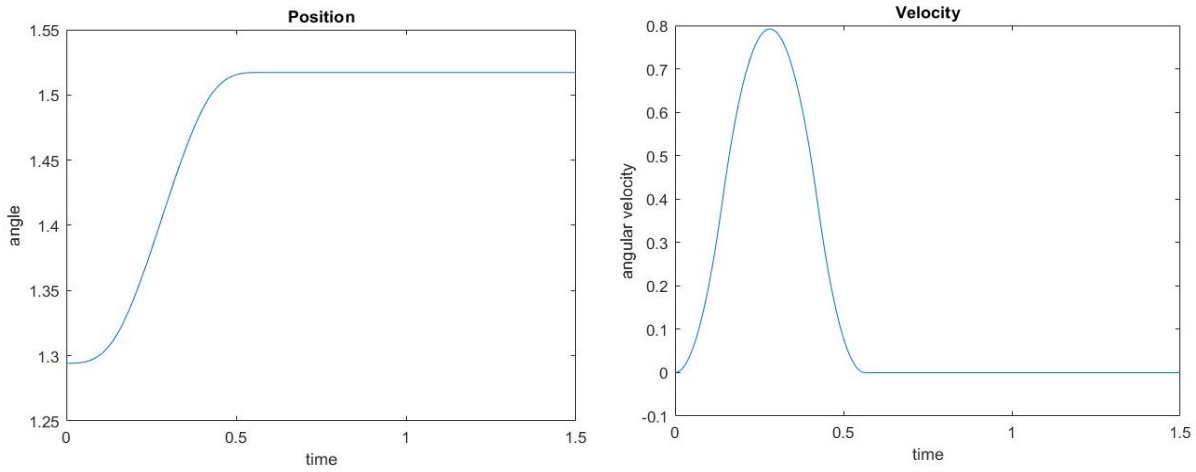
In these equations t_{k1} can be calculated using Eq. (13).

$$t_{k1} = t_j + t_{j1} \tag{13}$$

The determination of the initial and final positions is contingent upon the specific locations of the gripper at both the beginning and end of the task, with a deliberate emphasis on maintaining zero initial and final velocity and acceleration. Figure (4) illustrates the envisaged profiles, taking into consideration a precisely allotted time frame of 1.5 seconds spanning the entire process from product pickup to drop.

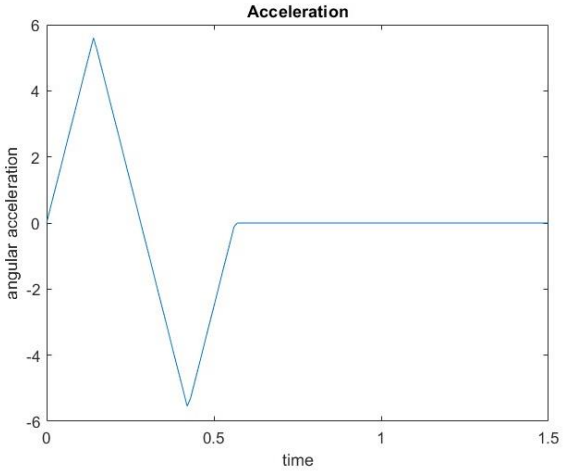
2.4 Derivation of dynamic equations and state space form of PUMA 560 robot

The study of robot's kinematics is divided into two sections: forward kinematics and inverse kinematics. Inverse kinematics is used to calculate the desired joint variables based on the path designed for the end effector. In this project, given the objective of designing a control system for a dynamic system, it is assumed that the desired joint variables have been obtained using inverse kinematics. In this section, the robot's forward kinematics is introduced using the Denavit-Hartenberg (D-H) method for application in future calculations. Figure (5) illustrates the coordinate frames required for the D-H method applied to the robot. Table (1) provides the corresponding transformations of the D-H method for this robot.



(**a**)

(**b**)



(**c**)

Figure 4 (a) Link angle (rad), **(b)** Link angular velocity (rad/s), **(c)** Link angular acceleration (rad/s2)

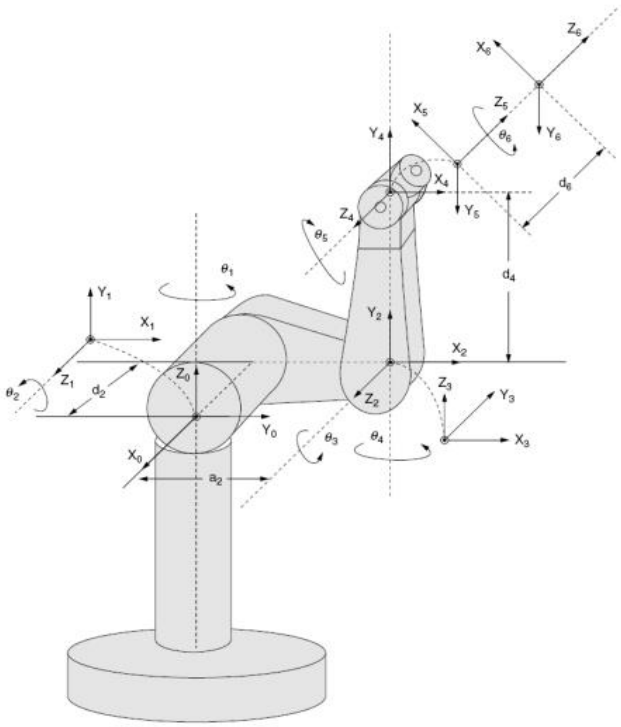


Figure 5 Puma robot with Denavit-Hartenberg coordinate [15].

Table 1 Denavit-Hartenberg Table [15].

Link	θ_z	θ_x	d_x	d_z
1	θ_1	$-\frac{\pi}{2}$	0	0
2	θ_2	0	a_2	d_2
3	θ_3	$\frac{\pi}{2}$	a_3	0
4	θ_4	$-\frac{\pi}{2}$	0	d_4
5	θ_5	$\frac{\pi}{2}$	0	0
6	θ_6	0	0	d_6

Using Table (1) and successive standard transformations, the homogeneous transformation matrix of the robot is obtained as follows:

$$ {}^{0}_{6}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T \times {}^{3}_{4}T \times {}^{4}_{5}T \times {}^{5}_{6}T = \begin{bmatrix} {}^{0}_{6}R & {}^{0}_{6}d \\ 0 & 1 \end{bmatrix} \tag{14} $$

That ${}^{0}_{6}R$ and ${}^{0}_{6}d$ are given by the following matrices:

$$ {}^{0}_{6}R = \begin{bmatrix} N_x & B_x & T_x \\ N_y & B_y & T_y \\ N_z & B_z & T_z \end{bmatrix}; \qquad {}^{0}_{6}d = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \tag{15} $$

The components of the matrices in Eq. (15) are thoroughly explained in the primary reference [15].

The Jacobian matrix describes the relationship between the joint velocities and the linear and angular velocities of the end-effector or each link of a robot. It is divided into two parts as shown in Eq. (16): linear and angular. The linear part is obtained by differentiating the positional components of the transformation matrix with respect to time as shown in Eq. (17), while the angular Jacobian is implicitly derived from the differentiation of the rotation matrix as shown in Eq. (18).

$$ J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} \tag{16} $$

$$ J_v = \frac{\partial {}^{0}_{6}d}{\partial q} \quad . \quad q = [\theta_1 . \theta_2 . \theta_3 . \theta_4 . \theta_5 . \theta_6] \tag{17} $$

$$ J_w = \frac{\partial {}^{0}_{6}R}{\partial q} \quad . \quad q = [\theta_1 . \theta_2 . \theta_3 . \theta_4 . \theta_5 . \theta_6] \tag{18} $$

A suitable method for deriving the dynamic equations of multilink systems is the Euler-Lagrange method, which is based on the energy of the system and models the dynamics of the system without the appearance of internal forces in the equations. The Euler-Lagrange equations are given in the Eq. (19) and (20).

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i \tag{19}$$

$$L = T - U \tag{20}$$

In the above equations, the kinetic energy of the system is calculated using the Jacobian matrix and the centers of mass of the robot's links. The linear velocity of each link's center of mass is determined using the Jacobian matrix, while the angular velocity of the links is derived using the angular Jacobian in relation to the velocity of the joint variables. Combining these velocities allows for the computation of the system's kinetic energy.

As gravitational potential energy is the primary form of potential energy influencing the dynamics of robots, it is calculated using the height of the center of mass of each link relative to the gravitational field.

Utilizing the Euler-Lagrange method, we derive the dynamic equation of the system in its generalized form, as represented by Eq. (21):

$$M(q)\ddot{q} + N(q.\dot{q})\dot{q} + G(q) = \tau \tag{21}$$

In Eq. (21), the variables N and G denote the non-linear components of the equation. In accordance with Eq. (22), N encompasses expressions associated with both Coriolis and centrifugal accelerations.

$$N(q.\dot{q}) = B(q)[\dot{q}\ \dot{q}] + C(q)[\dot{q}]^2 \tag{22}$$

In the Puma robot, the last three degrees of freedom correspond to the wrist mechanism and are primarily responsible for controlling the orientation of the end-effector. As a result, these three degrees of freedom do not influence the path-following functionality of the robot. Consequently, only the first three degrees of freedom are considered, leading to matrices M, B and C having dimensions of 3×3, while G and τ have dimensions of 3×1.

The complete specifications for the M and G matrices are exhaustively detailed in the original reference [15,16]. The determination of the matrix N involves navigating through various choices for array distribution. Commonly, in the design of the control law, the strategic selection of N is a prevailing practice, ensuring that $\dot{M} - 2N$ attains skew symmetry. This intricate derivation is facilitated through the application of existing relationships and the utilization of the matrix M as depicted in Eq. (23).

$$N_{ij} = \frac{1}{2}\sum_{k=1}^{n}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_k + \frac{1}{2}\sum_{k=1}^{n}\left(\frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i}\right)\dot{q}_k \tag{23}$$

In the state space representation, the system's state variables consist of the joint variables and their derivatives. The state-space representation of the system is expressed as follows:

$$x = [q_1.\ q_2.q_3.\dot{q}_1.\dot{q}_2.\dot{q}_3]^T \tag{24}$$

$$\dot{x} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ M^{-1}(\tau - N[x_4\ x_5\ x_6]^T - G) \end{bmatrix} \tag{25}$$

For the purpose of path-following, the system outputs are defined as described in Eq. (26).

$$y = [x_1.x_2.x_3]^T \tag{26}$$

2.5 Adaptive sliding mode control design

To design the controller for trajectory tracking of the robot, it is essential to determine the angular acceleration of the robot's links based on the state representation given in Eq. (25). From Eq. (24), the fourth, fifth, and sixth state variables of the system correspond to the angular velocities of the robot's links. Consequently, according to Eq. (25), the derivatives of these variables are equivalent to the angular accelerations of the robot's links. Thus, the angular accelerations of the robot's links are expressed as described in Eq. (27).

$$\ddot{q} = M^{-1}(\tau - Nq^T - G) \tag{27}$$

In Eq. (27), M^{-1} is represented in the form of matrix B, as shown in Eq. (28). Similarly, τ is defined as the input to the three robot links, expressed as a vector in Eq. (29). Furthermore, $M^{-1}(-Nq^T - G)$, representing the nonlinear component of the system dynamics, is formulated as a vector according to Eq. (30).

$$M^{-1} = B = \left[b_{ij}\right] \;\; ; \qquad i.j = 1.2.3 \tag{28}$$

$$\tau = [u_1\ u_2\ u_3] \tag{29}$$

$$M^{-1}(-Nq^T - G) = [f_1\ f_2\ f_3] \tag{30}$$

An effective strategy for crafting sliding mode control for multi-input, multi-output systems—where inputs align with monitored outputs—entails expressing the dynamic equations through Eq. (31) and extending the single-input, single-output systems methodology to this broader context.

$$\ddot{q}_i = f_i + \sum_{j=1}^{3} b_{ij}\, u_j \quad ; \; i.j = 1.2.3 \tag{31}$$

However, the challenge lies in determining the uncertainty limits, a prerequisite for this approach, outlined in Eq. (32) - (33). In Eq. (32), $\hat{f}$ represents the estimated nonlinearity of the robot's dynamic model, while f denotes its actual value. The difference between these two values reflects the parameter uncertainty in the robot's dynamics, with its limit defined by F. Similarly, in Eq. (33), B represents the inverse of the actual inertia matrix, and $\hat{B}$ is its estimated value in the model. The uncertainty in determining this matrix is indicated by Δ, with its upper bound specified by D.

Given the intricacies associated with the existence of the inverse of the M matrix and its inherent parametric uncertainties, establishing these limits becomes a formidable task. Consequently, our focus shifts to formulating a sliding mode control specifically tailored for serial robot problems. This design perspective places emphasis on the energy considerations and the positive definiteness of the M matrix.

$$\left|\hat{f}_i - f_i\right| \leq F_i \quad ; \; i = 1.2.3 \tag{32}$$

$$B = (I + \Delta)\hat{B} \qquad \left|\Delta_{ij}\right| \leq D_{ij} \;\; ; \; i.j = 1.2.3 \tag{33}$$

To fulfill this objective, the sliding surface vector is defined according to Eq. (36), in terms of the error between the actual angles of the robot links ($\vec{q}$) and their desired values ($\vec{q}_d$), as defined in Eq. (34), and $\dot{q}_r$, as specified in Eq. (35). In Eq. (36), Λ represents a positive definite symmetric matrix, and the formulation is as follows:

$$\tilde{q} = \vec{q} - \vec{q}_d \tag{34}$$

$$\dot{q}_r = \dot{q}_d - \Lambda\tilde{q} \tag{35}$$

$$s = \dot{\tilde{q}} + \Lambda\tilde{q} = \dot{q} - \dot{q}_r \tag{36}$$

In the next step, the control law should be designed in such a way that the condition of reaching the sliding level is met. Also, the goal is to track the desired path in such a way that the desired speed and acceleration are also limited. In this regard, some or all parameters of the robot are unknown; For this purpose, we are looking for a controller that can control the system in a way that follows the desired path in the best way by estimating the uncertain parameters at every moment. The basis of adaptive control method is the same, and the design of the controller is done in two stages: extracting a control rule to track the desired path and extracting the estimation rule to estimate the unknown parameters.
In this problem, the considered robot has 34 uncertain parameters. If we consider the nominal value of these parameters as vector a and their estimated value as vector $\hat{a}$ then the estimation error is defined as Eq. (37):

$$\tilde{a} = \hat{a} - a \tag{37}$$

In addition to the unknown parameters of the robot, we are looking to adaptively adjust the coefficient of the sign function, i.e. k, which is used in the control law and indicates the uncertainty range of the parameters. The process is the same as the previous section. The real value is defined by k, the estimated value by $\hat{k}$, and the error value is defined in Eq. (38) as follows.

$$\tilde{k} = \hat{k} - k \tag{38}$$

To design the control law at first the Lyapunov function, due to the positive definiteness and symmetry of the matrices M, Γ1 and Γ2, is considered as Eq. (39), where Γ1 is the adaptation gain for the uncertain parameters of the robot in the form of a positive definite square matrix with dimensions 34*34 and Γ2 is the adaption gain of coefficient k which is a positive scalar.

$$V = \frac{1}{2}\, s^T M s + \frac{1}{2}\, \tilde{a}^T {\Gamma_1}^{-1} \tilde{a} + \frac{1}{2}\, {\Gamma_2}^{-1} \tilde{k}^2 \tag{39}$$

The derivative of Lyapunov function is obtained as Eq. (40):

$$\dot{V} = s^T M\dot{s} + \frac{1}{2}\, s^T \dot{M} s + \hat{\dot{a}}^T {\Gamma_1}^{-1}\tilde{a} + + {\Gamma_2}^{-1}\tilde{k}\hat{\dot{k}} \tag{40}$$

By substituting $\dot{s}$ with $\ddot{q} - \ddot{q}_r$, as shown in Eq. (36), the derivative of the Lyapunov function can be expressed in the form of Eq. (41).

$$\dot{V} = s^T M(\ddot{q} - \ddot{q}_r) + \frac{1}{2} s^T \dot{M} s + \hat{\dot{a}}^T {\Gamma_1}^{-1}\tilde{a} + + {\Gamma_2}^{-1}\tilde{k}\hat{\dot{k}} \tag{41}$$

Based on Eq. (21) and (36), $M\ddot{q}$ can be substituted in the form of $\tau - Ns - N\dot{q}_r - G$, thereby yielding the derivative of the Lyapunov function as expressed in Eq. (42).

$$\dot{V} = s^T(\tau - Ns - N\dot{q}_r - G - M\ddot{q}_r) + \frac{1}{2} s^T \dot{M} s + \hat{\dot{a}}^T {\Gamma_1}^{-1}\tilde{a} + + {\Gamma_2}^{-1}\tilde{k}\hat{\dot{k}} \tag{42}$$

Finally, the derivative of the Lyapunov function can be expressed as shown in Eq. (43).

$$\dot{V} = s^T \left(\frac{\dot{M}}{2} - N\right) s + s^T(\tau - N\dot{q}_r - G - M\ddot{q}_r) + \hat{\dot{a}}^T {\Gamma_1}^{-1}\tilde{a} + + {\Gamma_2}^{-1}\tilde{k}\hat{\dot{k}} \tag{43}$$

Based on Eq. (43) and the antisymmetric property of $\frac{\dot{M}}{2} - N$, which ensures that $s^T \left(\frac{\dot{M}}{2} - N\right) s = 0$, the derivative of the Lyapunov function is derived in the form presented in Eq. (44).

$$\dot{V} = s^T(\tau - N\dot{q}_r - G - M\ddot{q}_r) + \hat{\dot{a}}^T {\Gamma_1}^{-1}\tilde{a} + + {\Gamma_2}^{-1}\tilde{k}\hat{\dot{k}} \tag{44}$$

In this part, considering that the matrices $M(q)$, $N(q.\dot{q})$ and $G(q)$ are dependent on uncertain parameters, as a result, we can define the Q matrix, which is a three-component vector, as follows in Eq. (45) and linearize it with respect to the vector of uncertain parameters, i.e. a.

$$Q = M\ddot{q}_r + N\dot{q}_r + G \tag{45}$$

In order to linearize Q with respect to a, considering that the relationship between Q and a is linear, we can calculate the Jacobian of the vector Q with respect to a, then we will have Eq. (46):

$$Q = M\ddot{q}_r + N\dot{q}_r + G = Y(q.\dot{q}.\dot{q}_r.\ddot{q}_r)a \tag{46}$$

where Y is obtained from Eq. (47) in the form of a 3*34 matrix.

$$Y_{ij} = \frac{\partial Q_i}{\partial a_j} \quad ; \quad i = 1.2.3 \quad j = 1.2.\ldots.34 \tag{47}$$

To establish the stability of the designed controller, it is essential to ensure that the Lyapunov function is positive definite and its derivative is negative definite. So, for the negative definiteness of the derivative of the Lyapunov function, we consider the control law as Eq. (48):

$$\tau = Y\hat{a} - K_D s - \hat{k}sign(s) \tag{48}$$

Now, by substituting Eq. (48) in Eq. (44) and also according to Eq. (46), we will have $\dot{V}$ as Eq. (49):

$$\dot{V} = s^T(\widehat{M}\ddot{q}_r + \widehat{N}\dot{q}_r + \hat{G} - K_D s - \hat{k}sign(s) - M\ddot{q}_r - N\dot{q}_r - G) + \dot{\hat{a}}^T\Gamma^{-1}\tilde{a} + \Gamma_2{}^{-1}\tilde{k}\dot{\hat{k}} \tag{49}$$

Considering that $\widetilde{M} = \widehat{M} - M$ and $\widetilde{N}$ and $\tilde{G}$ which have similar equations and also considering Eq. (46), we can simplify $\dot{V}$ in the form of Eq. (50) as follows:

$$\dot{V} = s^T Y\tilde{a} - s^T K_D s - s^T ksign(s) + \dot{\hat{a}}^T\Gamma^{-1}\tilde{a} + \Gamma_2{}^{-1}\tilde{k}\dot{\hat{k}} \tag{50}$$

Now, if we consider the adaption law of the estimated parameters and coefficient k as Eq. (51) – (52) respectively, then Eq. (50) is simplified to the form of Eq. (53).

$$\dot{\hat{a}} = -\Gamma Y^T s \tag{51}$$

$$\dot{\hat{k}} = \Gamma_2\|s\|_1 \tag{52}$$

$$\dot{V} = -s^T K_D s - k\|s\|_1 \tag{53}$$

Now, considering the positive definiteness of K_D and the positiveness of k, we can conclude that $\dot{V}$ is negative semi-definite as illustrated in Eq. (54).

$$\dot{V} \leq -s^T K_D s \leq 0 \tag{54}$$

According to Eq. (54), we can obtain Eq. (55) for $\ddot{V}$ as follows:

$$\ddot{V} \leq -2s^T K_D \dot{s} \tag{55}$$

If s and $\dot{s}$ are limited, then $\ddot{V}$ will also be limited, and according to Barbalat's lemma, if $\dot{V}$ is uniformly continuous, then $\dot{V}$ will tend to zero in infinite time, and as a result, according to Eq. (54), s will also tend to zero; In this way, according to Eq. (36), $\tilde{q}$ and $\dot{\tilde{q}}$ which are the tracking errors will also tend to zero. So now it is enough to show that s and $\dot{s}$ are bounded.
Given that $V > 0$ and $\dot{V} \leq 0$, it follows that V remains bounded. Consequently, in accordance with Eq. (39), s, $\tilde{a}$ and $\tilde{k}$ are constrained. This limitation extends to $\hat{a}$, $\hat{k}$, q and $\dot{q}$. Moreover, the closed-loop system can be expressed in the form of Eq. (56):

$$M\dot{s} + (N + K_D)s = Y\tilde{a} \tag{56}$$

which due to the positive definiteness of M, its inverse exists and is bounded, so $\dot{s}$ is also bounded, and according to the discussions, the proof of Lyapunov stability ends.

2.6 NN-based control of PUMA robot

In this section, the design of a control network for managing the robot's position is presented. The process begins by implementing the adaptive sliding mode control on a simulated robot. By applying various desired inputs, key output parameters such as joint torques, positions, velocities, and accelerations are extracted. These outputs serve as the training dataset for optimizing the performance of the control network. Subsequently, a multi-layer feed-forward neural network is designed. The input to the neural network comprises nine features, which include the desired angle, velocity, and acceleration of the joint, the current angle and velocity, the previous step angle and velocity, a time parameter, and the applied torque from the previous time step. These features collectively provide a comprehensive representation of the robot's current state and the control history, allowing the network to predict the appropriate torque for the given desired inputs.

The architecture of the NN is a fully connected feed-forward network designed for regression tasks. It begins with an input layer of 9 neurons corresponding to the 9 input features. The activation function used in this layer is ReLU, which introduces the non-linearity necessary for the network to model complex relationships. The input layer is followed by two hidden layers, each consisting of 15 neurons and employing the ReLU activation function to handle non-linear mapping efficiently. The final output layer has a single neuron representing the predicted torque value, and no activation function is applied here as the output is a continuous variable. The neural network diagram is depicted in Figure (6).

The model is compiled using the Adam optimizer, which is well-suited for large networks and provides efficient learning. The loss function chosen is MSE (Mean Squared Error), which is commonly used for regression problems to minimize the difference between the predicted and actual torque values. Before training, the input data is standardized to ensure consistency and faster convergence during optimization. The dataset used for training the network consists of 8 sets, each comprising 385 to 394 timesteps. These are split into training and validation datasets with an 85% and 15% ratio, respectively. Additionally, one test dataset comprising 394 timesteps is used for evaluation. The training process involves a dataset generated from simulations of an adaptive sliding mode controller applied to a robot model, as previously described.

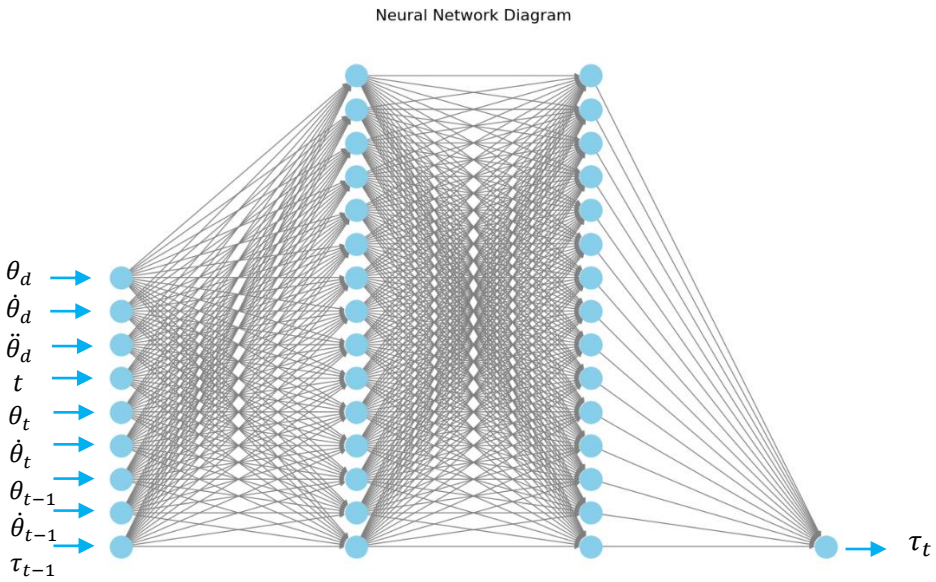


Figure 6 Control neural network with 9 inputs and one output and two hidden layers with 15 neurons and RELU activation function

Key outputs such as joint torques, positions, velocities, and accelerations are extracted and used for training. The network is trained over 1000 epochs with a batch size of 32. Validation is performed on a separate test dataset to evaluate the model's generalization performance. The model's accuracy is quantified using the MSE on the test data, and its predictions are visually compared against actual torque values through time-series plots.
In summary, the NN-based controller predicts the torque required to achieve the desired joint behavior. By leveraging state information, historical inputs, and dynamic properties, the network adapts effectively to varied control scenarios, enabling the robot to operate precisely in real-world environments. The architecture, optimization techniques, and input selection ensure robustness and accuracy in controlling the robot.

3 Results and discussion

In this section, we present and analyze the obtained results through a dual-phase examination. First, we present and analyze the outcomes of our image processing endeavors using YOLOv9 for fruit detection. The efficacy of this advanced object detection model is meticulously evaluated, highlighting its precision and robustness in identifying various fruit types. Following this, we delve into a comprehensive exploration of the results, with particular emphasis on the performance evaluation of the designed controllers. This dual focus not only underscores the effectiveness of our image processing techniques but also provides a thorough assessment of the controllers' operational performance, thereby offering valuable insights into the overall system's capability and reliability.

3.1 YOLOv9 performance in fruit detection: results and analysis

Training our YOLOv9 CNN network involved a robust dataset comprising 258 diverse images, with 219 images for training and 39 for validation, spanning four distinct fruit types: tomato, strawberry, plum, and peach. Additionally, 122 images were used for testing. The model was trained over 25 epochs with a batch size of 16, yielding impressive results that demonstrate the model's strong performance in fruit detection. The subsequent paragraphs will delve into a detailed discussion of these results.
The analysis of the YOLOv9 performance metrics for fruit detection reveals a comprehensive improvement in model accuracy over the training period. Figure (7) illustrates the trends of box_loss, cls_loss, and dfl_loss for both the training and validation datasets, each representing a critical component of the YOLO-based fruit detection model. Box_loss, also known as Bounding Box Regression Loss, measures the accuracy of the predicted bounding box positions and dimensions, such as center coordinates, width, and height, compared to the ground truth. To penalize any misalignment, it uses the IoU loss function. Cls_loss, or Classification Loss, evaluates how effectively the model predicts the correct class for an object within a detected bounding box. This is achieved using a cross-entropy loss function, which compares the predicted class probabilities with the ground truth labels to ensure accurate object classification. Meanwhile, dfl_loss, or Distribution Focal Loss, refines bounding box coordinates using probability distributions. It optimizes fine-grained predictions to align the predicted box coordinates more closely with the ground truth. Together, these loss functions guide the training process, enhancing the model’s performance in object recognition and localization. According to the Figure (7), the train/box_loss graph displays a consistent decrease in bounding box regression loss for the training data, indicating a progressive enhancement in the model's ability to predict bounding boxes accurately. Similarly, the train/cls_loss graph shows a significant drop during the initial epochs, followed by a gradual decline, suggesting that the model is rapidly learning to classify different fruit types correctly. Additionally, the train/dfl_loss graph highlights a decreasing trend in distribution focal loss, signifying growing confidence in the model's predictions.

In Figure (7), on the validation side, the val/box_loss graph, despite some fluctuations, shows an overall downward trend, implying that the model is effectively generalizing to unseen data. The val/cls_loss graph mirrors the training classification loss, decreasing significantly and indicating strong generalization capabilities. The val/dfl_loss graph, while exhibiting some variability, also trends downward, further confirming the model's improved performance on validation data.

At the end of the 25th epoch, the final values for box loss, class loss, and dfl loss are 0.417, 0.3792, and 0.9488, respectively. These metrics underscore the model's robustness and accuracy in detecting and classifying four distinct fruit types—tomato, strawberry, plum, and peach—demonstrating a solid performance throughout the training process.

Figure (8) illustrates the graphs for precision, recall, mAP_50 (mean Average Precision), and mAP_50:0.95 for the testing data. Precision and recall are computed based on true positives (TP), false negatives (FN), false positives (FP), and true negatives (TN), as defined by Eq. (57) and Eq. (58), respectively. mAP_50 represents the average area under the precision-recall curve, depicted in Figure (9). mAP_50 is calculated using a threshold of 0.5, while mAP_50:0.95 employs a range of thresholds from 0.5 to 0.95.

$$Precision = \frac{TP}{TP + FP} \tag{57}$$

$$Recall = \frac{TP}{TP + FN} \tag{58}$$

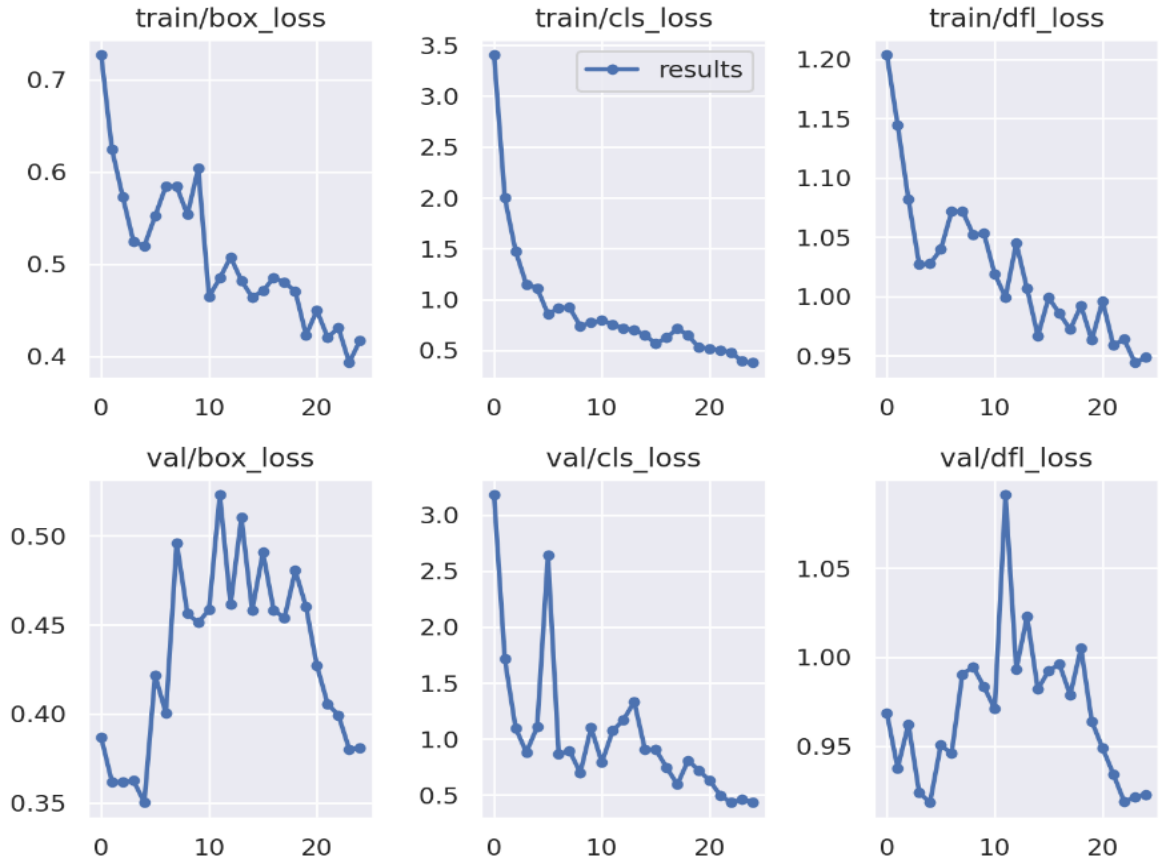


Figure 7 Training and Validation Loss Curves for YOLOv9 Fruit Detection Model: Tracking box loss, classification loss, and DFL loss over 25 epochs, showing significant convergence and performance improvements in both training and validation phases

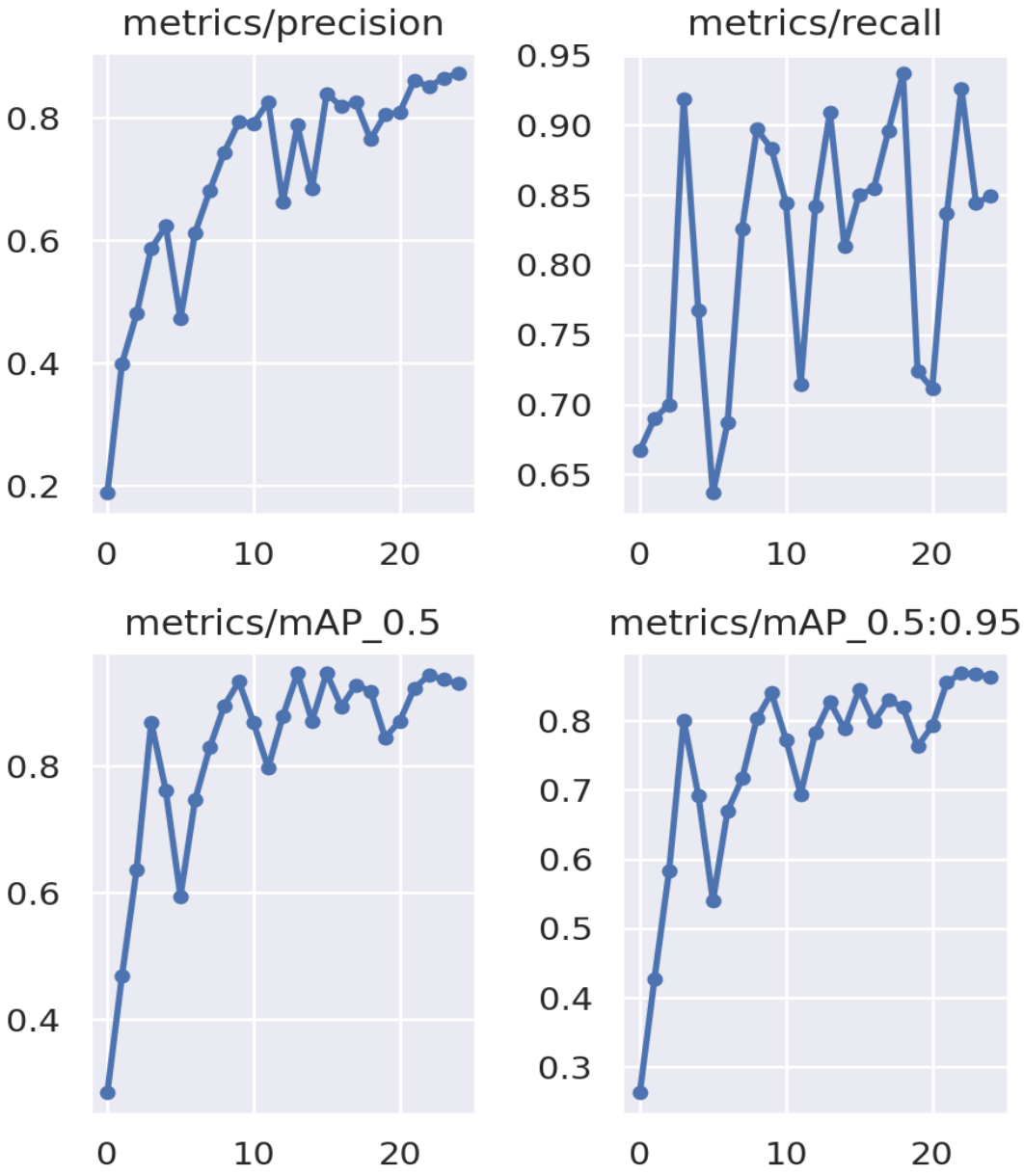


Figure 8 Performance evaluation of the YOLOv9 algorithm, depicted through four metrics: Precision (top-left) shows an increasing trend, indicating improved accuracy in identifying true positives; Recall (top-right) exhibits fluctuations, reflecting variability in detecting all actual fruit instances; mAP 0.5 (bottom-left) demonstrates a generally rising trend, indicating an improved precision-recall balance at a 0.5 threshold; and mAP 0.5:0.95 (bottom-right) shows a steady increase in mean Average Precision across thresholds from 0.5 to 0.95, indicating enhanced robustness in detection across various confidence levels.

The analysis of the YOLOv9 performance metrics for fruit detection highlights substantial improvements and robust model performance over the training period. The metrics/precision graph demonstrates a steady increase, ultimately exceeding 0.9, which indicates a significant reduction in false positive errors and showcases the model's accuracy in identifying the correct fruit types. Similarly, the metrics/recall graph, though initially variable, eventually stabilizes at a high value around 0.9, suggesting that the model effectively detects most fruit instances with minimal false negatives.

The metrics/mAP_0.5 graph, which shows the mean Average Precision at an IoU threshold of 0.5, exhibits an increasing trend and stabilizes between 0.8 and 0.9. This indicates strong performance in terms of both precision and recall at this threshold, reinforcing the model's

reliability. Furthermore, the metrics/mAP_0.5:0.95 graph, representing the mean Average Precision across various IoU thresholds from 0.5 to 0.95, also shows an upward trend, stabilizing around 0.8. This consistency across different overlap thresholds signifies a well-rounded and versatile model capable of robust detection under varying conditions.

At the end of the 25th epoch, the precision, recall, mAP_0.5, and mAP_0.5:0.95 values are 0.873, 0.849, 0.943, and 0.862, respectively. These metrics underscore the model's high accuracy and efficiency in detecting and classifying the four types of fruits—tomato, strawberry, plum, and peach—demonstrating the effectiveness of the YOLOv9 algorithm in this application.

The analysis of the Precision-Recall (PR) curves, illustrated in Figure (9), for fruit detection using the YOLOv9 algorithm reveals noteworthy performance distinctions among the four fruit classes. The tomato class (Blue Curve) stands out with an almost flawless PR curve and a mean Average Precision of 0.995. This exceptional score underscores the model's remarkable accuracy in detecting tomatoes, with very few false positives and false negatives. Similarly, the strawberry class (Orange Curve) shows robust performance, with a precision-recall curve closely mirroring that of tomatoes and mAP of 0.970, reflecting high precision and recall with minimal detection errors.

The peach class (Red Curve) also performs admirably, maintaining a high precision-recall curve across most recall values. The mAP of 0.959 indicates a strong detection capability with relatively few misclassifications. In contrast, the plum class (Green Curve) exhibits a lower mAP of 0.848, revealing greater variability in its precision-recall curve. This variability suggests that the model faces more challenges in accurately detecting plums, leading to a higher incidence of false positives and false negatives.

Overall, the combined performance across all fruit classes, represented by the bold blue curve, demonstrates an impressive mAP of 0.943 at an IoU threshold of 0.5. This high overall mAP value indicates that the YOLOv9 model is highly effective in detecting and classifying the different fruit types with excellent precision and recall, though there is room for improvement in detecting certain fruit types, such as plums.

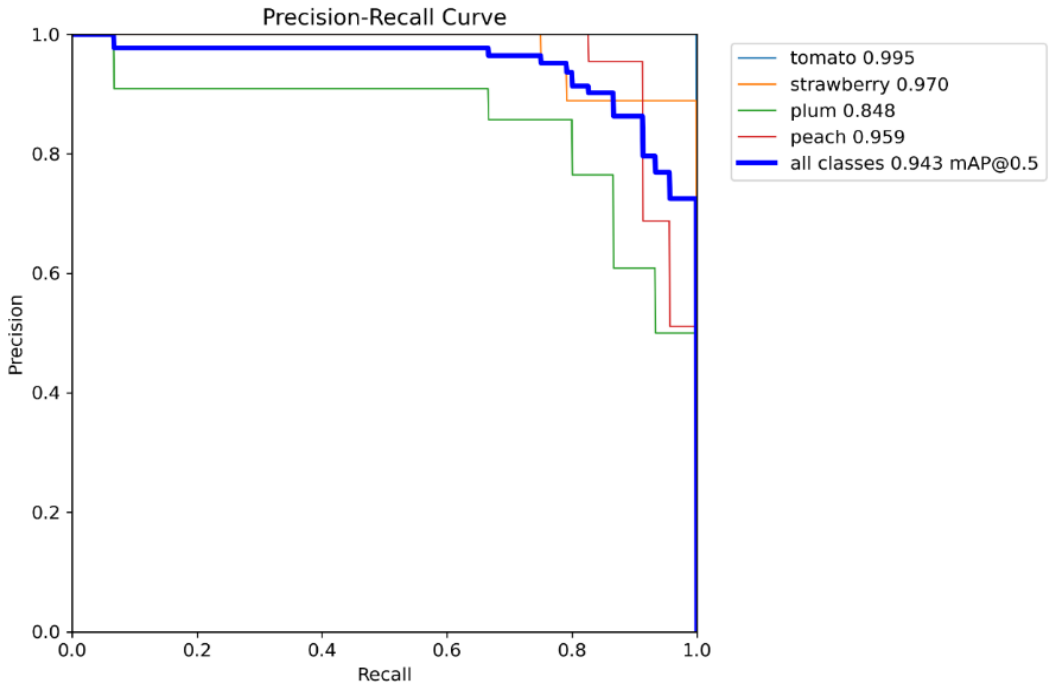


Figure 9 Precision-recall curve for fruit detection: The graph shows precision-recall curves for detecting various fruits: tomato (light blue), strawberry (orange), plum (green), and peach (red). The blue curve represents combined performance with an mAP of 0.943 at 0.5 IoU.

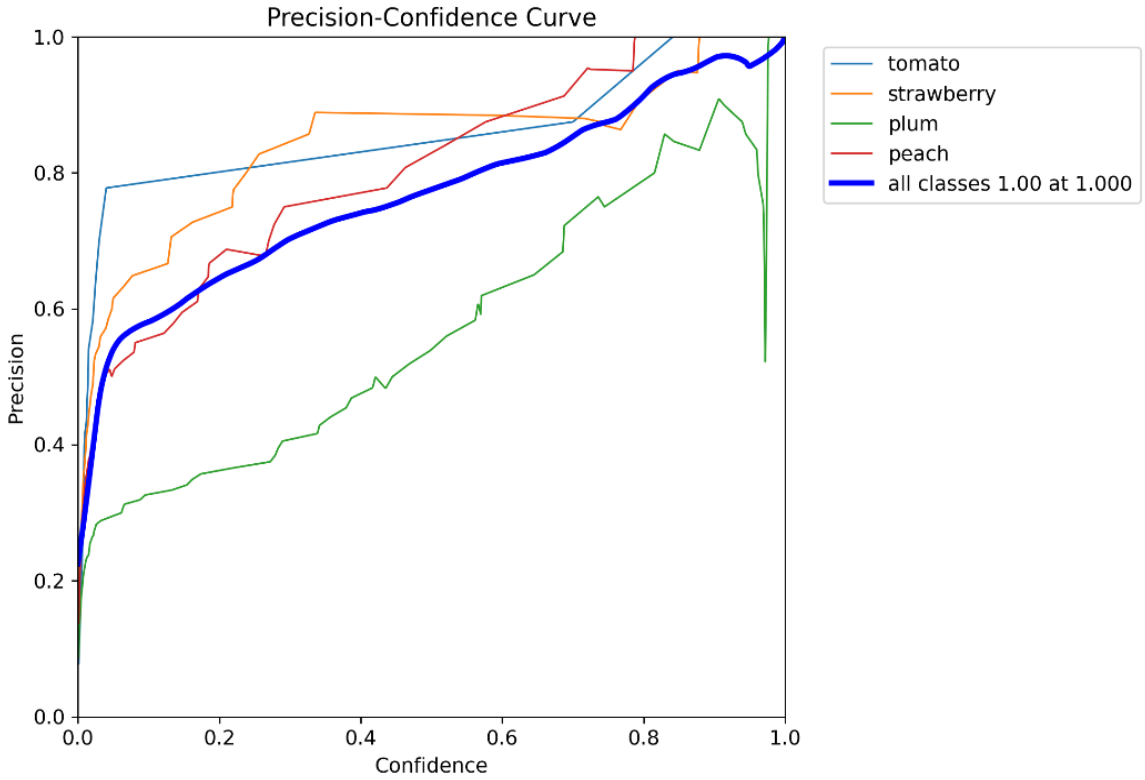


Figure 10 Precision-confidence curve for fruit detection: Detailed performance comparison across fruit classes (tomato, strawberry, plum, and peach). The curve illustrates the relationship between precision and confidence, highlighting the overall model performance (all classes) achieving 1.00 precision at 1.000 confidence.

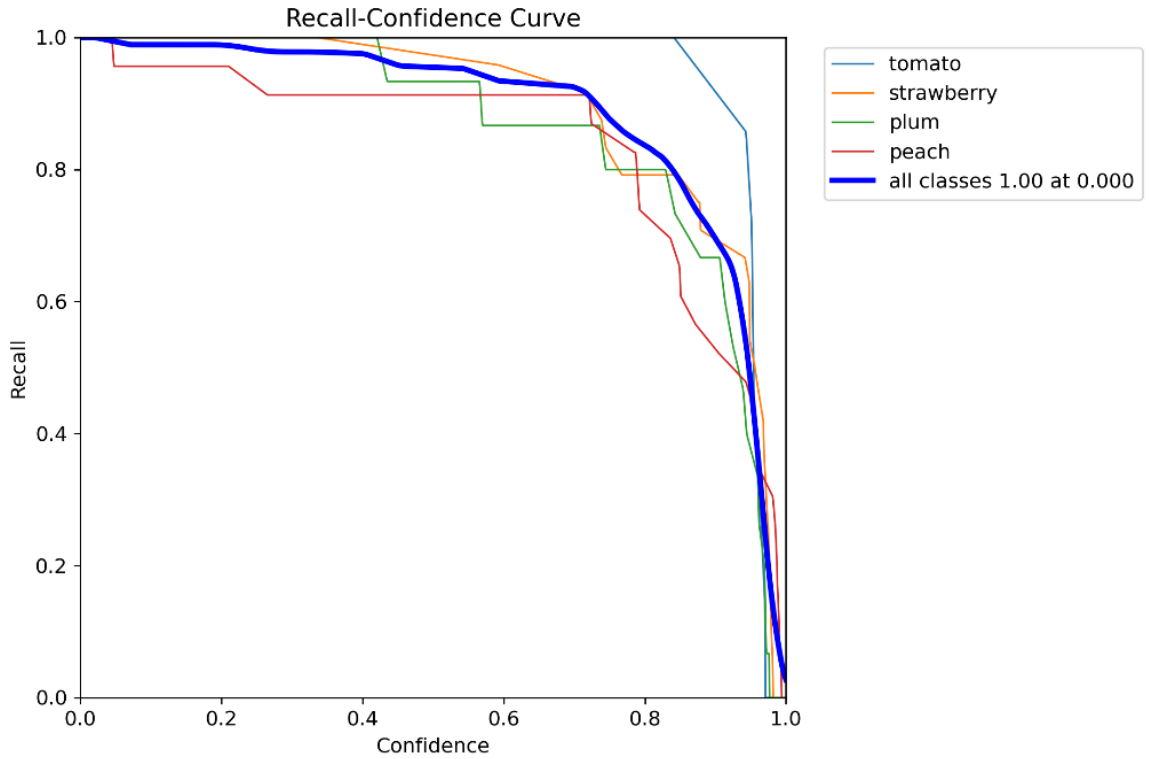


Figure 11 Recall-confidence curve for fruit detection: The graph shows the recall-confidence curves for the YOLOv9 algorithm detecting various fruits: tomato (light blue), strawberry (orange), plum (green), and peach (red). The blue curve represents combined recall for all classes, achieving a perfect recall of 1.00 at a confidence level of 0.000. This indicates how recall varies with confidence thresholds for each fruit type.

Figure (11) is a Recall-Confidence Curve for a YOLOv9 algorithm used for fruit detection, illustrating how recall varies with confidence for tomatoes, strawberries, plums, and peaches. Recall for tomatoes is very high across almost all confidence levels, especially up to about 0.8, indicating that the model detects most tomatoes even at lower confidence thresholds. As expected, recall drops sharply at the highest confidence levels due to the trade-off with precision. Strawberries show a similar pattern, maintaining high recall across most confidence levels, with a noticeable drop around 0.7. Plums exhibit generally high recall, but slightly lower compared to tomatoes and strawberries, with a significant drop-off as confidence increases. Peaches maintain high recall across most confidence levels, though with more fluctuations and a sharp decline at higher confidence levels. The overall performance, represented by the thick blue line, shows high recall up to a confidence level of about 0.8, after which it declines sharply. This trend indicates that the model effectively detects most fruits at lower confidence levels but with potentially lower precision. Key insights include the high recall for tomatoes and strawberries across most confidence levels and the need for improvement in detecting plums and peaches at higher confidence levels.

Figure (12) is an F1-Confidence Curve for a YOLOv9 algorithm used for fruit detection, illustrating how the F1-score varies with confidence for tomatoes, strawberries, plums, and peaches. For tomatoes, the F1-score is consistently high across a wide range of confidence levels, peaking above 0.9 around a confidence level of 0.8, indicating excellent balance between precision and recall. Strawberries exhibit the highest F1-scores, often exceeding 0.9, with a smooth curve indicating stable performance and a peak around 0.9 confidence. Conversely, the F1-score for plums is lower, peaking just above 0.7, with a jagged curve suggesting variability and challenges in detection. Peaches show a high F1-score, reaching above 0.8, though with more fluctuations compared to strawberries. The overall performance, represented by the thick blue line, peaks at an F1-score of 0.89 at a confidence level of 0.713, demonstrating a good overall balance of precision and recall. Key insights reveal that the model excels in detecting strawberries and tomatoes, with high and stable F1-scores, while performance for peaches is good but less stable. Plums have the lowest F1-score, indicating lower precision and recall.

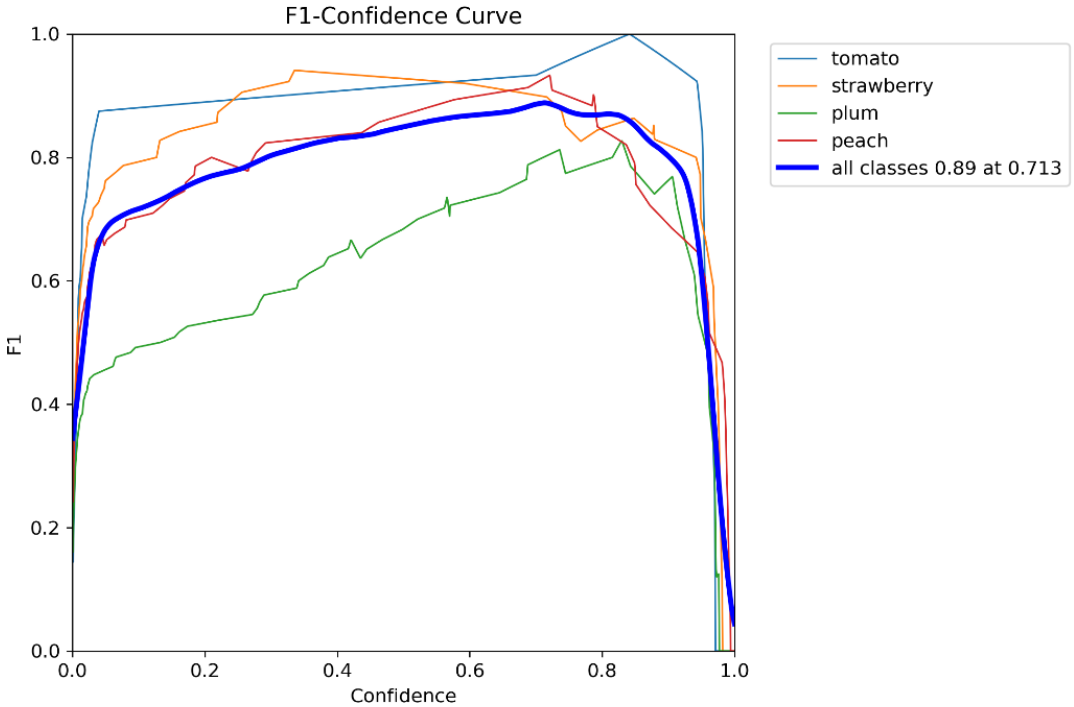


Figure 12 F1-Confidence curve for fruit detection: The graph shows the F1-confidence curves for detecting different fruits: tomato (light blue), strawberry (orange), plum (green), and peach (red). The blue curve indicates combined performance for all classes, with a peak F1 score of 0.89 at a confidence level of 0.713. This highlights the algorithm's precision and recall balance across varying confidence thresholds.

The confusion matrix for the YOLOv9 algorithm's fruit detection performance, depicted in Figure (13), reveals perfect classification for tomatoes and strawberries (1.00), but lower accuracy for plums (0.93) and peaches (0.91), with significant misclassifications for plums with the background (0.63) and peaches with plums (0.22). While the model excels in detecting tomatoes and strawberries, it struggles with plums and peaches, particularly in distinguishing them from each other and the background. To enhance performance, data augmentation with more diverse examples of plums and peaches, fine-tuning model hyper parameters, and implementing advanced post-processing techniques can be helpful.

In conclusion, the YOLOv9 CNN network demonstrated robust performance in detecting and classifying four distinct fruit types—tomato, strawberry, plum, and peach—across a comprehensive dataset. Trained over 25 epochs, the model achieved high precision and recall metrics, especially for tomatoes and strawberries, as evidenced by the precision-recall and F1-confidence curves. While the model excelled in identifying tomatoes and strawberries with near-perfect accuracy, it faced challenges with plums and peaches, particularly in differentiating them from each other and the background. These insights highlight the model's overall effectiveness and suggest areas for further improvement, such as enhancing the dataset with more diverse examples and refining model parameters.

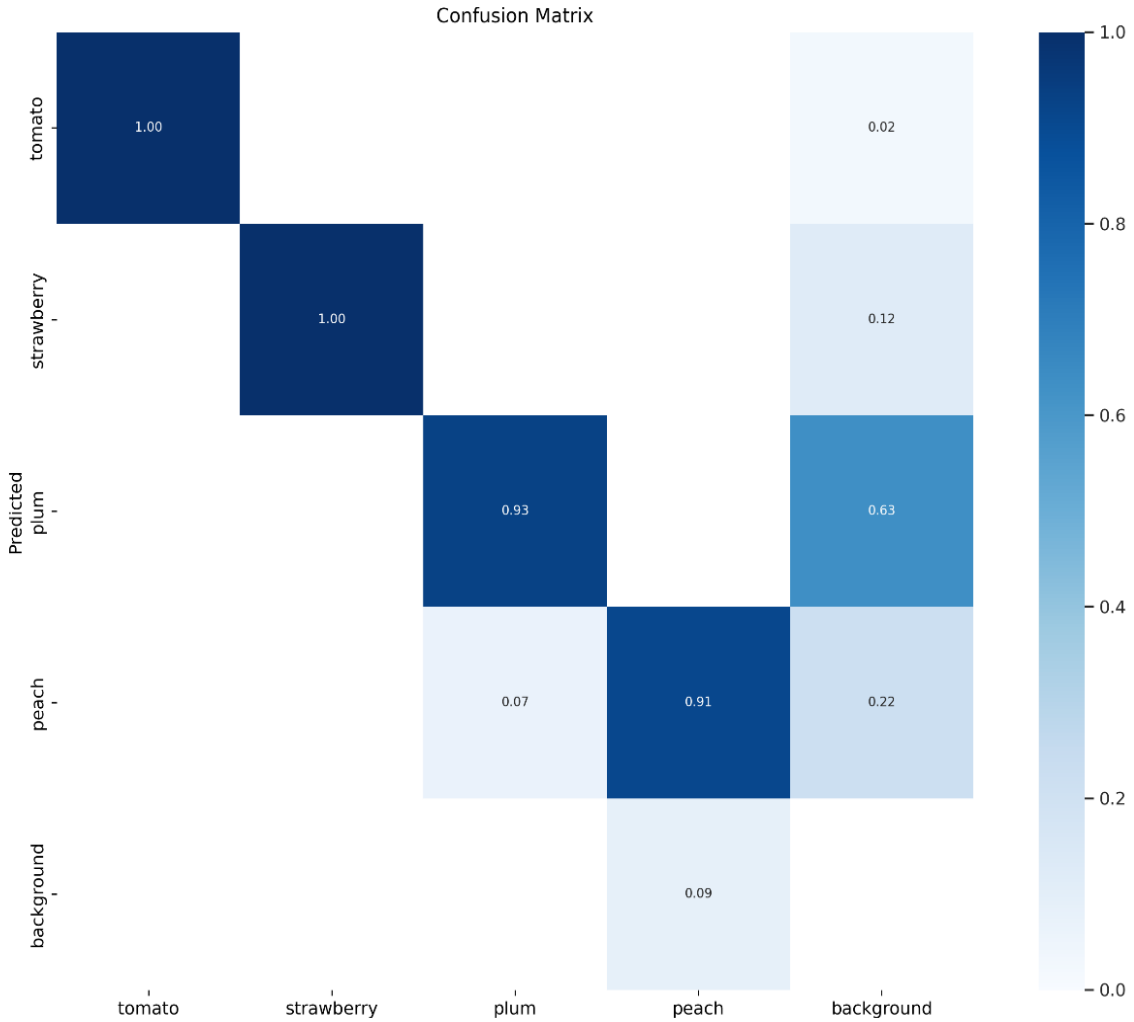


Figure 13 Confusion matrix for fruit detection using YOLOv9 algorithm: This matrix displays the performance of the YOLOv9 algorithm in correctly identifying various fruits (tomato, strawberry, plum, peach) and distinguishing them from the background. Diagonal values represent correct classifications, while off-diagonal values indicate misclassifications. The color intensity reflects the frequency of predictions, with darker shades indicating higher accuracy.

3.2 Comparative Analysis of Sliding Mode and NN-Based Controllers for PUMA 560

To assess the effectiveness of the developed controllers and facilitate a comprehensive comparative analysis, the designed controllers are rigorously evaluated using a detailed simulation within the MATLAB environment, specifically utilizing Simscape Multibody for accurate physical modeling. This simulation framework allows for a thorough examination of the controllers' performance in managing the PUMA 560 robotic arm's dynamics and tasks. The simulated representation of the robot provides a visual and analytical basis for comparing the sliding mode controller and the neural network-based controller. By leveraging the advanced capabilities of MATLAB and Simscape Multibody, the evaluation process ensures that both controllers are tested under a variety of conditions, enabling an in-depth understanding of their strengths, weaknesses, and overall effectiveness in achieving the desired control objectives.
In Figure (14), we observe the detailed simulation setup involving two PUMA 560 robotic arms, each functioning as a distinct subsystem within the overall model. This comprehensive simulation includes the implementation of a sliding mode controller and a neural network-based controller, alongside a motion planner. Each controller is applied to one of the robotic arms, allowing for a direct comparison of their performance in managing the PUMA 560's movements and tasks. The setup aims to provide insights into the efficiency, accuracy, and robustness of both control strategies under various operating conditions.
Upon conducting the simulation, notable parallels in behavior between the neural network-based controller and the sliding mode controller become evident, highlighting the efficacy of both control strategies in managing the PUMA 560 robotic arm. The corresponding Mean Squared Error values for the estimated torque of the control network across the three links of the robot are outlined in Table (2).

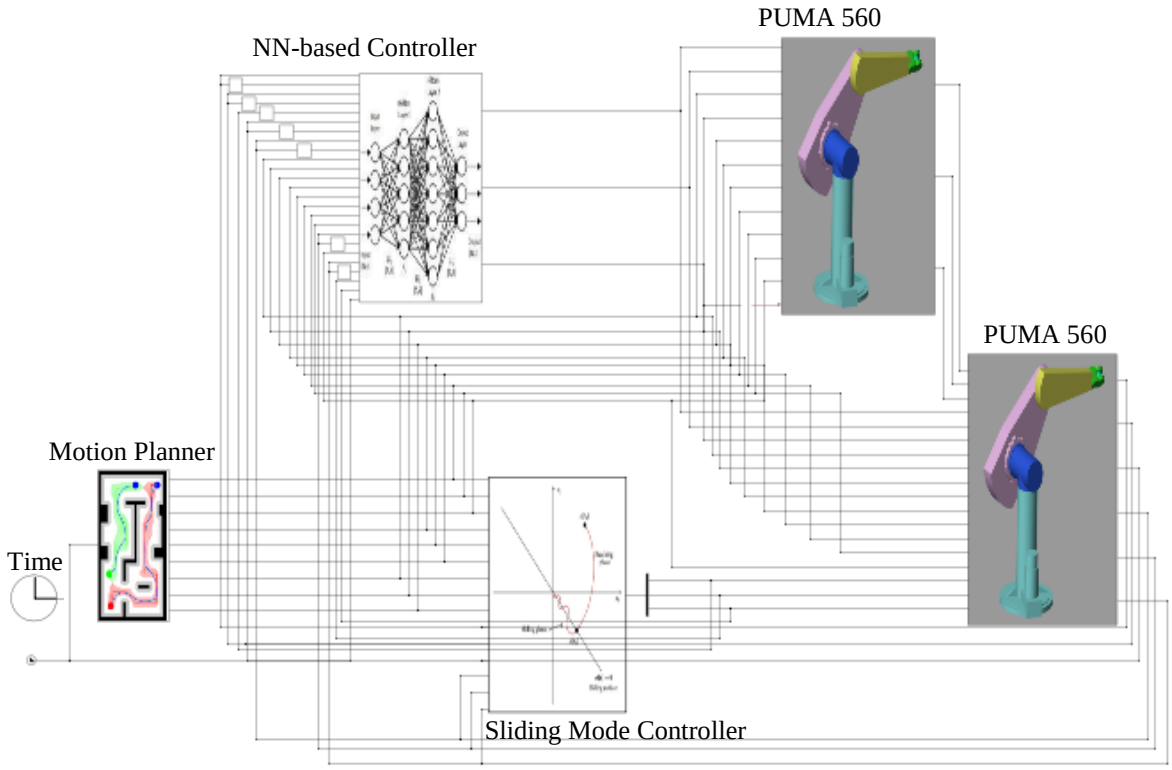


Figure 14 The whole system: Simulated modules in Simscape-MATLAB- **Motion Planner:** Generates desired motion profiles ($q_d . \dot{q}_d . \ddot{q}_d$) for 3 links based on time input. **NN-Based Controller:** Computes torque (τ_t) using desired motion ($q_d . \dot{q}_d . \ddot{q}_d$), current and previous link states for 3 links ($q_t . \dot{q}_t . q_{t-1} . \dot{q}_{t-1}$), and previous torque ($\tau_{t-1}$). **Sliding Mode Controller:** Produces control effort (τ_t) using desired motion ($q_d . \dot{q}_d . \ddot{q}_d$) and current link states ($q_t . \dot{q}_t$).

These results prominently reveal minimal test errors for all three links, underscoring the model's exceptional accuracy in estimating the requisite torque. Specifically, the low MSE values indicate that NN-based controller is highly precise in predicting the necessary torque to achieve the desired movements and tasks. Furthermore, Figure (15) provides a detailed visual representation of the actual and estimated torque of the network across the three links of the robot. This figure illustrates the close alignment between the actual and predicted torque values, further validating the robustness and reliability of the controller. The minimal errors and close match in torque estimations across all three links highlight the potential of this controller to deliver precise and efficient performance, making it suitable for advanced robotic applications.

Table 2 Mean squared error comparison of estimated torques for robot links using NN-based controller: analysis across training, validation, and test datasets

Link No.	MSE $(Train\ data)$	MSE $(Validation\ data)$	MSE $(Test\ data)$
1	0.0226	0.0292	0.0247
2	1.3022	0.4494	4.7708
3	0.0150	0.0167	0.0799

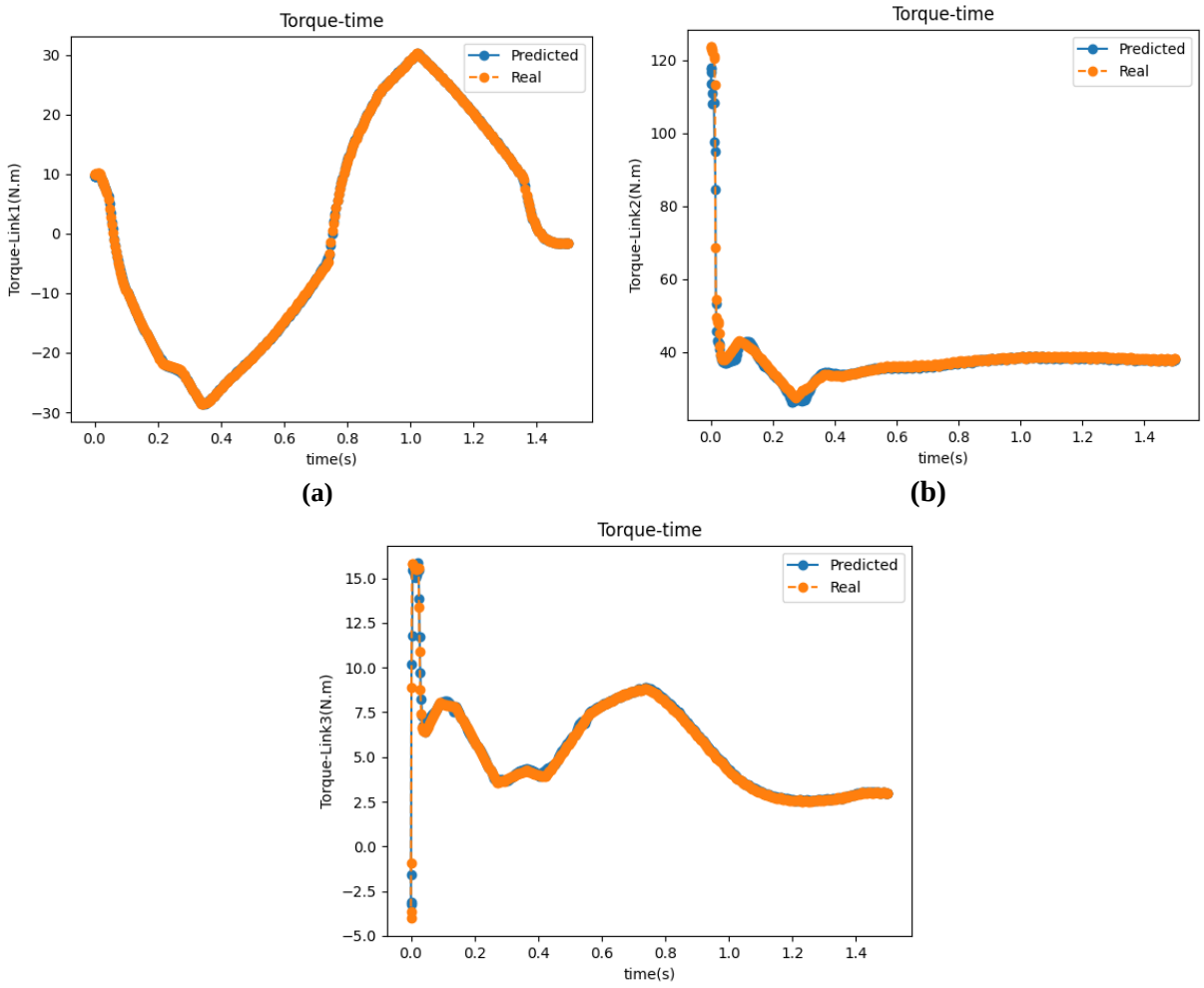


Figure 15 Comparison of Predicted and Real Torque for Robot's Link Using NN-Based Controller: This graph illustrates the predicted torque (blue) and the real torque (orange) for the robot's links over time, showcasing the high accuracy of the NN-based controller in estimating the required torque. **(a)** Link 1 **(b)** Link 2 **(c)** Link 3

Finally, the results of applying these two controllers on the PUMA 560 robot are presented in Figure (16). As depicted in the graphs, both controllers have successfully followed the desired path, demonstrating their effectiveness in trajectory tracking. However, it appears that the sliding mode control has performed slightly better in terms of accuracy and robustness compared to the NN-based control.

Despite this, it is important to consider that the data required to determine the optimal path, based on the type of product, is obtained from image processing results. Consequently, the use of the NN-based control facilitates a more efficient and expeditious transfer of this information within the Python programming environment. This integration enables the robot under neural network control to exhibit heightened speed and efficiency in product classification tasks. The neural network-based controller, therefore, provides significant advantages in scenarios where rapid processing and real-time adaptation are critical, making it a valuable approach for dynamic and complex environments.

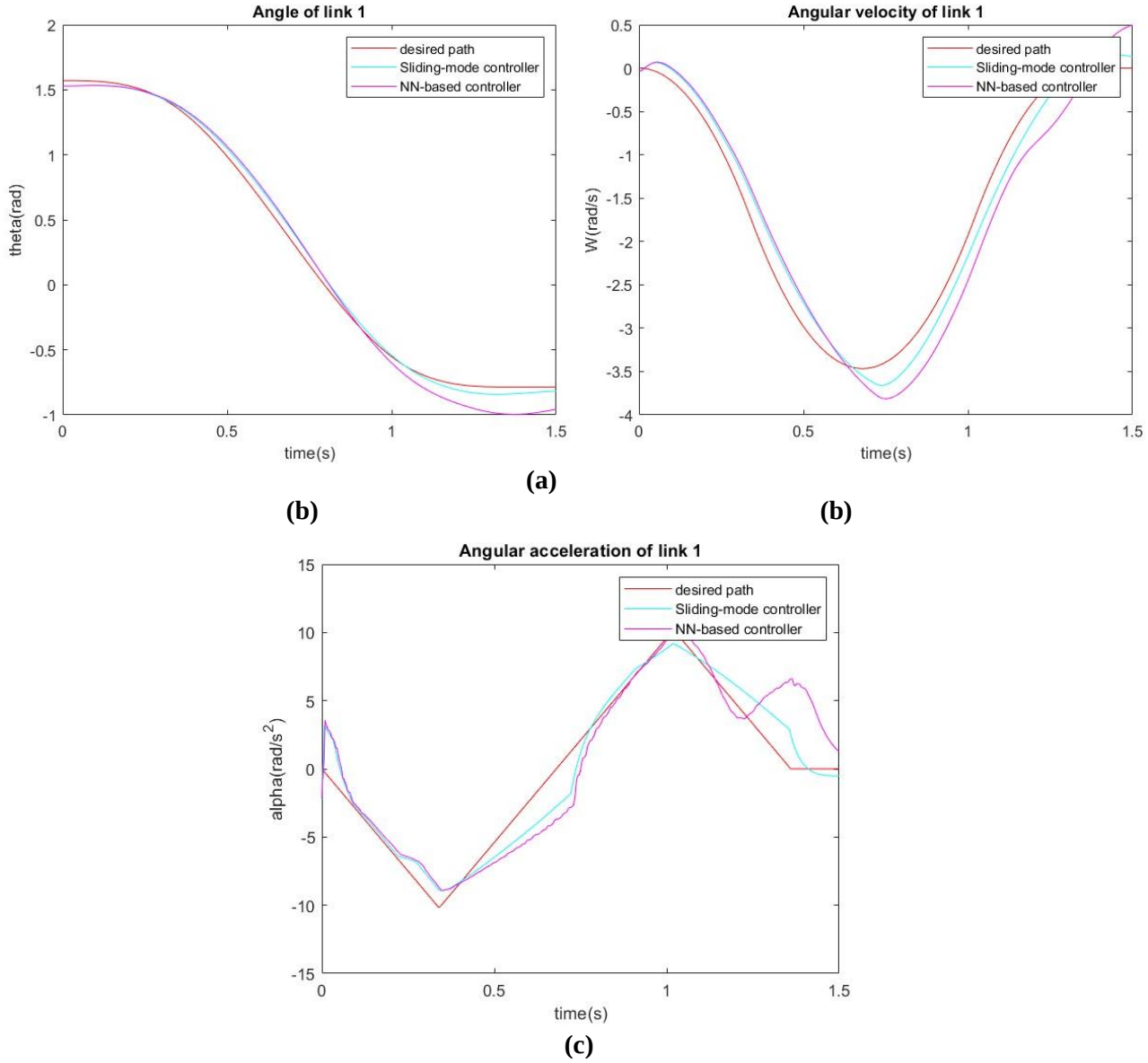


Figure 16 The tracking performance of the sliding mode and NN-based controllers for one of the robot's links in terms of **(a)** position **(b)** velocity and **(c)** acceleration. Each of the three graphs presents three distinct trajectories: the desired path (red), the path followed by the sliding mode controller (blue), and the path followed by the NN-based controller (purple). Both controllers demonstrate efficient tracking of the desired path across all three metrics, highlighting their ability to accurately and robustly control the robot's movements.

4 Conclusions

In conclusion, the implementation of the YOLOv9 algorithm for fruit detection on a conveyor belt in a production line has proven to be highly effective and efficient. The model's ability to accurately identify and classify four types of fruit demonstrates its robustness in real-world applications. Its simple real-time image processing ensures fast decision-making, optimizing efficiency and minimizing errors in the production line. This technology holds promise for enhancing automation and quality control processes in the fruit processing industry, thereby boosting productivity.

Moreover, the YOLOv9 algorithm has shown consistency and accuracy in improving the reliability and speed of fruit detection in production environments. In this project, the YOLOv9 algorithm achieved an accuracy of 0.873 in fruit identification, which is acceptable. While the model excelled in identifying tomatoes and strawberries with near-perfect accuracy, it encountered challenges with plums and peaches, particularly in distinguishing them from each other and the background. These findings highlight the overall effectiveness of the model and suggest areas for further improvement, such as enhancing the dataset with more diverse examples and refining model parameters.

Additionally, the study provided valuable insights into the performance of both sliding mode and neural network-based controllers in the context of robot control. The results indicate that the sliding mode controller demonstrates superior efficacy in guiding the automated fruit-sorting robot along desired trajectories compared to its neural network-based counterpart. However, the NN-based controller offers commendable speed advantages, enabling swift and efficient information transfer within the robotic system. Thus, while sliding mode control excels in precision, neural network-based control proves to be a promising alternative, particularly in scenarios requiring rapid decision-making and response times, such as the fruit sorting task.

References

[1] V. E. Balas, R. Kumar, and R. Srivastava, "*Recent Trends and Advances in Artificial Intelligence and Internet of Things*", in Intelligent Systems Reference Library (ISRL), Vol. 172, Cham: Springer International Publishing, 2020, doi: 10.1007/978-3-030-32644-9.

[2] B. Iscimen, H. Atasoy, Y. Kutlu, S. Yildirim, and E. Yildirim, "Smart Robot Arm Motion using Computer Vision," *Elektronika ir Elektrotechnika*, Vol. 21, No. 6, pp. 3-7, 2015, doi: 10.5755/j01.eee.21.6.13749.

[3] N. Kondo, "Robotization in Fruit Grading System," *Sensing and Instrumentation for Food Ruality and Safety*, Vol. 3, No. 1, pp. 81-87, 2009, doi: 10.1007/s11694-008-9065-x.

[4] M. M. Sofu, O. Er, M. C. Kayacan, and B. Cetişli, "Design of an Automatic Apple Sorting System using Machine Vision," *Computers and Electronics in Agriculture*, Vol. 127, pp. 395-405, 2016, doi: 10.1016/j.compag.2016.06.030.

[5] T. Dewi, P. Risma, and Y. Oktarina, "Fruit Sorting Robot Based on Color and Size for an Agricultural Product Packaging System," *Bulletin of Electrical Engineering and Informatics*, Vol. 9, No. 4, pp. 1438-1445, 2020, doi: 10.11591/eei.v9i4.2353.

[6] V. D. Cong, "Visual Servoing Control of 4-DOF Palletizing Robotic arm for Vision Based Sorting Robot System," *International Journal on Interactive Design and Manufacturing*

(IJIDeM), Vol. 17, No. 2, pp. 717-728, Apr. 2023, doi: 10.1007/s12008-022-01077-8.

[7] W. T. Abbood, O. I. Abdullah, and E. A. Khalid, "A Real-time Automated Sorting of Robotic Vision System Based on the Interactive Design Approach," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, Vol. 14, No. 1, pp. 201-209, 2020, doi: 10.1007/s12008-019-00628-w.

[8] H. M. T. Abbas, U. Shakoor, M. J. Khan, M. Ahmed, and K. Khurshid, "Automated Sorting and Grading of Agricultural Products Based on Image Processing," *8th International Conference on Information and Communication Technologies (ICICT)*, pp. 78-81, 16-17 November 2019, Karachi, Pakistan, doi: 10.1109/ICICT47744.2019.9001971.

[9] P. Chen, and V. Elangovan, "Object Sorting using Faster R-CNN," *International Journal of Artificial Intelligence & Applications (IJAIA)*, Vol. 11, No. 5/6, pp. 27-36, 2020, doi: 10.5121/ijaia.2020.11603.

[10] S. Sanwar, and M. I. Ahmed, "Automated Object Sorting System with Real-time Image Processing and Robotic Gripper Mechanism Control," *Journal of Engineering Advancements*, Vol. 04, No. 03, pp. 70-79, 2023, doi: 10.38032/jea.2023.03.003.

[11] A. Visioli, and G. Legnani, "On the Trajectory Tracking Control of Industrial SCARA Robot Manipulators," *IEEE Transactions on Industrial Electronics*, Vol. 49, No. 1. pp. 224-232, 2002, doi: 10.1109/41.982266.

[12] P. Bobka, F. Gabriel, and K. Dröder, "Fast and Precise Pick and Place Stacking of Limp Fuel Cell Components Supported by Artificial Neural Networks," *CIRP Annals*, Vol. 69, No. 1, pp. 1-4, 2020, doi: 10.1016/j.cirp.2020.04.103.

[13] M. E. Mostafa, A. E. Mostafa, H. H. Ammar, and R. Shalaby, "ANN-Based Modeling and Control of a Pick and Place Manipulato", In: X. S. Yang, R. S. Sherratt, N. Dey, A. Joshi, (eds) *Proceedings of Eighth International Congress on Information and Communication Technology, ICICT*, Lecture Notes in Networks and Systems, Springer, Singapore, Vol. 694, pp. 189-204, 2023, https://doi.org/10.1007/978-981-99-3091-3_15.

[14] S. Patil, S. Waghule, S, Waje, P, Pawar, and S, Domb, "Efficient Object Detection with YOLO: A Comprehensive Guide," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 519-531, 2024, doi: 10.48175/IJARSCT-18483.

[15] F. Piltan, S. Emamzadeh, Z. Hivand, F. Shahriyari, and M. Mirazaei "PUMA-560 Robot Manipulator Position Sliding Mode Control Methods using MATLAB / SIMULINK and Their Integration into Graduate / Undergraduate Nonlinear Control, Robotics and MATLAB Courses," *International Journal of Robotic and Automation, (IJRA)*," Vol. 6, No. 3, pp. 106-150, 2012.

[16] B. Armstrong, O. Khatib, and J. Burdick, "The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 Arm," in *Proceedings, 1986 IEEE International Conference on Robotics and Automation, Institute of Electrical and Electronics Engineers*, pp. 510-518, San Francisco, CA, USA, 1986, doi: 10.1109/ROBOT.1986.1087644.

Nomenclatures

English symbols

a	Actual value of uncertain parameters
â	Estimated value of uncertain parameters
ã	Estimation error of uncertain parameters
B(q)	Coriolis term
B	Perturbed matrix
$\widehat{B}$	Estimated matrix
C	Centrifugal term
D	Bound for perturbation elements
F	Absolute error bound
f_i	Actual value of a parameter
$\hat{f}_i$	Estimated value of a parameter
G	Gravitational term
I	Identity matrix
k	Actual value of uncertainty range coefficient
$\hat{k}$	Estimated value of uncertainty range coefficient
$\tilde{k}$	Estimation error of uncertainty range coefficient
M	Inertia matrix
N	Coriolis and centrifugal term
Q	Control input
q	Actual state vector (Joint position)
q_d	Desired state vector
$\tilde{q}$	Error state vector
$\dot{q}_r$	Reference velocity
s	Sliding surface
V	Lyapunov function
Y	Regressor matrix

Greek symbols

Γ_1	Adaption gain of uncertain parameters
Γ_2	Adaption gain of uncertainty range coefficient
Δ	Perturbation matrix
Λ	Gain matrix
τ	Joint torques