International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 15, Issue 6 (Nov. – Dec. 2026), PP. 116-127

Towards an Automated Waste-Sorting System: Computer Vision and Neural-Network Inverse Kinematics for a 5-DOF Arm

Sy-Dung Vu[1], Phan-Duy-Tan Nguyen[1], Duc-Hieu Nguyen[1], Kim-Thanh Tran-Thi[2], Mai-Phuong Dao-Thi[3], N.-K. Nguyen[1,*]

[1]*Faculty of Automation, Artificial Intelligence and Intelligent Systems, Electric Power University, Hanoi, Vietnam*

[2]*Center for Practical Training and Laboratory, Electric Power University, Hanoi, Vietnam*

[3]*Faculty of Automation, School of Electrical and Electronic Engineering, Hanoi University of Industry, Hanoi, Vietnam*

[*]*Corresponding Author*

Abstract: *Automated waste sorting with a robotic manipulator requires two capabilities that belong to different fields: recognizing and localizing the object in the image, and converting a target Cartesian position into joint angles. This paper reports the construction and separate evaluation of both modules for a 5-DOF manipulator, and verifies the inverse-kinematics module on a continuous trajectory. For the recognition module a dedicated dataset, EPU_waste, of 2,955 images and 4,068 annotated instances in three classes was built, and two instance-segmentation models, YOLOv8s-seg and YOLO11s-seg, were trained and compared on it. YOLOv8s-seg reaches a recall of 0.97 against 0.88 and runs at 56.5 frames per second on a Quadro RTX 3000 GPU, and was therefore selected. For the inverse-kinematics module a 7–15–15–4 feedforward network with 424 weights was trained by the Levenberg–Marquardt algorithm over ten independent runs, giving a joint angle error of 2.2164 ± 0.1811° and a position error of 9.96 mm. On a continuous circular trajectory, the network gives a position RMS error of 3.461 mm with a standard deviation of only 1.30 mm, so the error is largely a near-constant systematic offset.*

Keywords:*inverse kinematics; artificial neural network; Levenberg–Marquardt; instance segmentation; YOLO; waste classification.*

Date of Submission: 02-11-2026 Date of acceptance: 12-11-2026

I. INTRODUCTION

At many treatment facilities household, waste is still sorted by hand, which is slow, repetitive work carried out in unhygienic surroundings. Replacing the human operator with a manipulator requires the machine to perform two operations in sequence, and the two belong to different fields. The first is a perception task: the system must decide which class an object belongs to and where it lies in the frame. The second is a kinematic task: the target position must be converted into a set of joint angles that brings the gripper to it.

Progress on the first operation has been rapid in recent years. Single-stage detectors of the YOLO family [1] run in real time, and their segmentation variants return, for every object, not only a bounding box and a class label but a mask at pixel level [2]. For grasping, a mask is more useful than a box: the centroid of a mask estimates the center of the object far better than the center of a box whose area may be mostly background.

The second operation is older and, for an open-chain manipulator, is in principle well understood, since a closed-form solution follows from the Denavit-Hartenberg parameters [3]. In practice, however, the closed-form solution of a 5-DOF arm is multi-valued, so one branch has to be selected explicitly, and it is ill-conditioned near both the inner and the outer limit of the reachable region. These two properties are the reason for using a feedforward neural network as an approximate solver [4]: once trained, the network returns an answer in fixed time, and that answer is single-valued.

This paper reports the construction and evaluation of both modules for a 5-DOF manipulator, with two contributions:

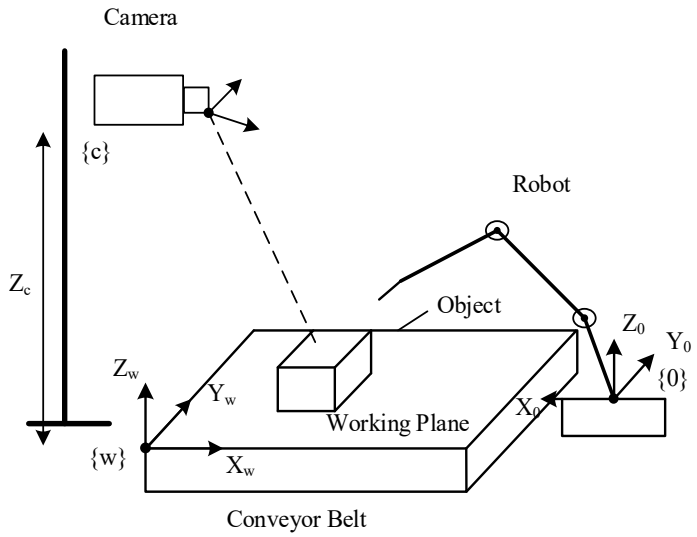
(i) an instance-segmentation dataset built specifically for waste sorting, named EPU_waste, together with a controlled comparison of two segmentation architectures on it;

(ii) a neural-network inverse-kinematics solver for the 5-DOF arm, assessed over ten independent training runs rather than one, so that the accuracy reported carries a measure of its dispersion.

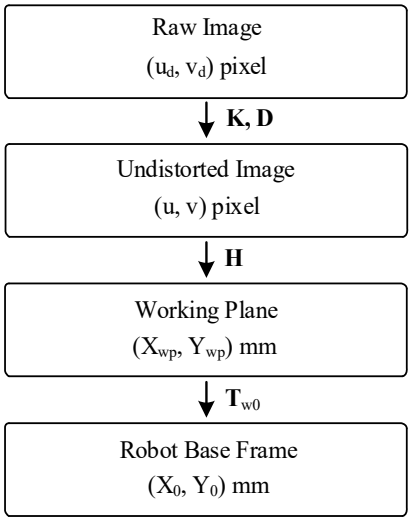
II. IMAGE PROCESSING MODULE

2.1 Coordinate transformation chain

An Eye-to-Hand configuration is used: the camera is fixed to a frame outside the manipulator and views the whole working plane. The geometric relation between camera and robot base therefore stays constant throughout operation and has to be calibrated only once. Fig. 1(a) shows the arrangement together with the four frames involved: the image frame, the camera frame {c}, the working-plane frame {w} attached to the calibration board, and the robot base frame {0}.



(a) Eye-to-Hand Configuration and Four Coordinate Frames



(b) Coordinate Transformation Chain

Fig. 1. Eye-to-Hand Setup and Coordinate Transformation: (a) System Configuration and Four Coordinate Frames; (b) Four-Stage Transformation from Pixel to Robot Base Coordinates.

The segmentation mask obtained in Section 3 tells which class an object belongs to and which image region it occupies, but its coordinates are pixel coordinates. Fig. 1(b) shows the four stages that convert them into physical units in the base frame. The raw image coordinates (ud, vd) are first undistorted using the calibrated intrinsic matrix K and the distortion coefficients D, giving (u, v). The representative point of an object is the centroid of its mask, that is, the mean coordinate of the pixels belonging to the mask. Because the objects lie on a known plane, the mapping from the image plane to the working plane is a projective homography:

$$\lambda p = Hm, \quad p = \left[X_{\mathrm{wp}}, Y_{\mathrm{wp}}, 1\right]^{\mathrm{T}}, \quad m = [u, v, 1]^{\mathrm{T}} \tag{1}$$

The matrix H is defined up to a scale factor, so although it has nine entries it carries only eight degrees of freedom; each point correspondence supplies two equations, and four coplanar correspondences are therefore sufficient to solve for it. The last stage carries coordinates from the board frame {w} into the base frame {0} by a rotation and a translation in the plane:

$$\begin{aligned} X_0 &= X_{\mathrm{wp}} \cos\beta - \sigma Y_{\mathrm{wp}} \sin\beta + t_x \\ Y_0 &= X_{\mathrm{wp}} \sin\beta + \sigma Y_{\mathrm{wp}} \cos\beta + t_y \end{aligned} \tag{2}$$

where β is the angle between the two frames, (tx, ty) is the translation vector, and σ takes the value +1 or −1 according to whether the two frames have the same or the opposite orientation. Flipping an axis changes the handedness of the frame, so σ must be fixed explicitly and cannot be inferred from the rotation alone.

The intrinsic parameters of the camera were calibrated from 25 board images at a resolution of 640 × 480, giving a mean reprojection error of 0.489 pixel. The homography was estimated from 54 corners of a ChArUco board [5], [6]; all 54 were retained as inliers after outlier rejection, with an RMS residual of 0.131 mm and a maximum of 0.308 mm over a working area of 161.2 × 138.7 mm, the scale at the center of the area being 0.637 mm per pixel. An independent check with a 300 mm gauge returned 299.37 mm, an error of 0.21%.

The mapping in (1) holds only for points lying on the calibrated plane. For an object of appreciable height, parallax displaces the mask centroid away from the projection of the object center; for the present geometry, with the camera at a height of 312.3 mm and the optical axis tilted by 2.34°, the displacement reaches

about 5 mm for an object 13 mm high at the far corner of the working area. A parallax correction based on object height, and the extraction of object orientation from the mask, are under development and lie outside the scope of this paper.

2.2 Instance segmentation module

2.2.1 Object detection using the YOLO model

The quality of the training data plays an important role in determining the performance of deep learning models. In practical waste recognition and classification tasks, publicly available datasets such as TACO [7] and OpenImages may not adequately represent the conditions encountered in a specific deployment environment, including variations in illumination, viewing angle and the physical condition of the waste objects. For this reason, we developed EPU_waste using images collected under conditions similar to those of our experimental setup. The resulting dataset comprises 2,955 images. The number of annotated instances in each waste category is reported in Table 1.

Table 1. Number of annotated instances per waste category in the training dataset.

Class	Metal	Paper	Plastic
Number of labels	1,418	1,538	1,112
Total	4,068		

Within the scope of this study an experimental model was built to evaluate the application of YOLO to waste recognition and classification. The dataset includes three waste classes (metal, paper/cardboard, plastic) with relatively distinct morphological features, as shown in Fig. 2.



Fig. 2. Examples of images from the EPU_waste dataset.

Table 2 presents the basic characteristics of each waste class in the dataset, which serve as a basis forfeature analysis and for the construction of a computer vision model for waste recognition and classification.

Table 2. Characteristics of each waste class in the EPU_waste dataset.

Class	**Label**	**Recognized characteristics**
0	Metal	Aluminium cans, metal bars, tin cans; light-reflecting surfaces.
1	Paper	Paper boxes, newspapers, cardboard and similar items.
2	Plastic	Plastic bottles and containers; transparent or multi-coloured.

The three classes were chosen for their relatively distinct morphological features: metal objects (aluminium cans, tin cans, metal bars) present specular, light-reflecting surfaces; paper objects (boxes,

newspapers, cardboard) are matte and often planar; plastic objects (bottles, containers) are transparent or multi-colored. Representative images are shown in Fig. 2.

The images were collected in a single experimental environment, with a largely uniform background and lighting, each frame containing one or a few well-separated objects. All masks were annotated on the Roboflow platform by one team following a single boundary convention. These two properties govern how the results of Section 2.3 should be read and are revisited in Section 2.5.

2.2.2 Model and evaluation metrics

Fig. 3 illustrates the data processing procedure and the general architecture of the YOLO model used in this work. The dataset first passes through a preprocessing phase that includes image size normalization and data augmentation. The images are then converted into tensors and fed to the network. The backbone extracts object features, the neck combines features at different scales, and the head produces the predictions. For the segmentation variant the output comprises the location, class label, confidence and segmentation mask of each object.

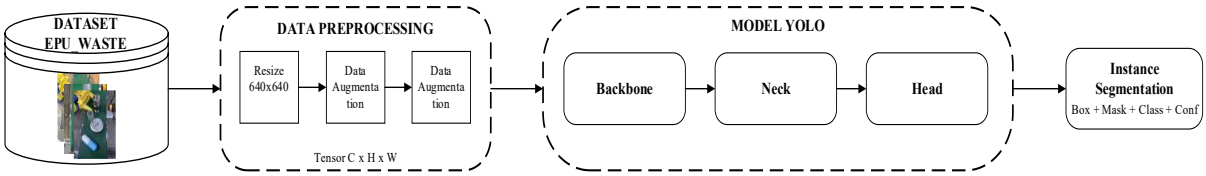


Fig. 3. Training and inference pipeline of the YOLO model.

Precision, Recall, mAP@50 and mAP@50-95 [8], [9], together with inference time (ms) and frames per second (FPS), were adopted as the primary evaluation metrics. Model selection was based on the ability to achieve high detection accuracy while meeting the real-time processing requirement of an automated waste classification system.

The core metrics are defined in Equations (3)-(6):

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Rcall = \frac{TP}{TP + FN} \tag{4}$$

$$AP = \int_0^1 P(R)dR \tag{5}$$

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{6}$$

A confusion matrix was also employed to give a visual representation of the classification errors, as shown in Fig. 4. The matrix consists of the following components [10]:

Confusion matrix

		Prediction	
		Positive (P)	Negative (N)
Ground truth	Positive (P)	True Positive (TP)	False Negative (FN)
	Negative (N)	False Positive (FP)	True Negative (TN)

Fig. 4. Structure of the confusion matrix used in model performance evaluation.

where:

- TP (true positive) is the number of positive instances correctly classified by the model;

- FP (false positive) is the number of negative instances misclassified as positive;
- FN (false negative) is the number of positive instances misclassified as negative;
- P and R denote Precision and Recall respectively;
- N is the number of object categories.'

2.3 Experimental results of model training

The experiments were conducted on a computer equipped with an Intel Core i7-10850H CPU, an NVIDIA Quadro RTX 3000 GPU with 6 GB of VRAM and 32 GB of RAM, in a Python environment. The models were trained with a batch size of 16, a learning rate of 0.001 and 200 epochs. Input images were resized to 640 × 640 pixels. At inference the confidence threshold was set to 0.5 and the IoU threshold for non-maximum suppression to 0.45. Results on the held-out EPU_waste test split are given in Table 3.

Table 3. Recognition results on the EPU_waste test split.

Model	Precision (P)	Recall (R)	mAP@50	mAP@50–95
YOLO11s-seg	0.95	0.88	0.93	0.91
YOLOv8s-seg	**0.96**	**0.97**	**0.98**	**0.96**

As shown in Table 3, YOLOv8s-seg achieves higher values than YOLO11s-seg on all metrics, but the magnitude of the difference is not uniform, and only part of it is large enough to be conclusive from a single training run per model. In Precision the advantage is 0.01, which lies within the run-to-run variation to be expected from random initialization and is therefore not interpreted here. The contrast in Recall is of a different order: YOLOv8s-seg detects about 97% of the instances in the test split while YOLO11s-seg reaches 88%. The errors of YOLO11s-seg thus stem primarily from missed detections rather than from false positives, and its F1 score is limited to 0.914 against 0.965.

This ordering is the reverse of the one reported on COCO, where YOLO11s attains 47.0 mAP@0.5:0.95 against 44.9 for YOLOv8s [11]. The discrepancy suggests that architectural improvements tuned for a large general-purpose dataset do not necessarily transfer to a small domain-specific one. In an automated sorting line a low recall translates directly into unclassified waste passing the station, which is the costlier of the two error types. On this criterion YOLOv8s-seg is the more suitable model.

One feature of Table 3 calls for an explanation. For both models mAP@0.5:0.95 lies only about 0.02 below mAP@0.5, whereas on large-scale datasets the gap is typically 0.15 to 0.30. This unusually small gap follows from properties of the dataset rather than from the capability of the models. First, all masks were annotated under a single convention on a single platform, so the labels carry none of the dispersion that arises when several annotators work independently; the model learns that convention and reproduces it closely, even at an IoU threshold of 0.95. Second, the images were collected under uniform conditions and the objects have comparatively simple boundaries, so the segmentation task here is appreciably easier than on field imagery. The mAP@0.5:0.95 of Table 3 therefore measures agreement between the model and the annotation convention of this dataset; it serves to compare the two models with each other, but should not be set directly against values published on COCO.

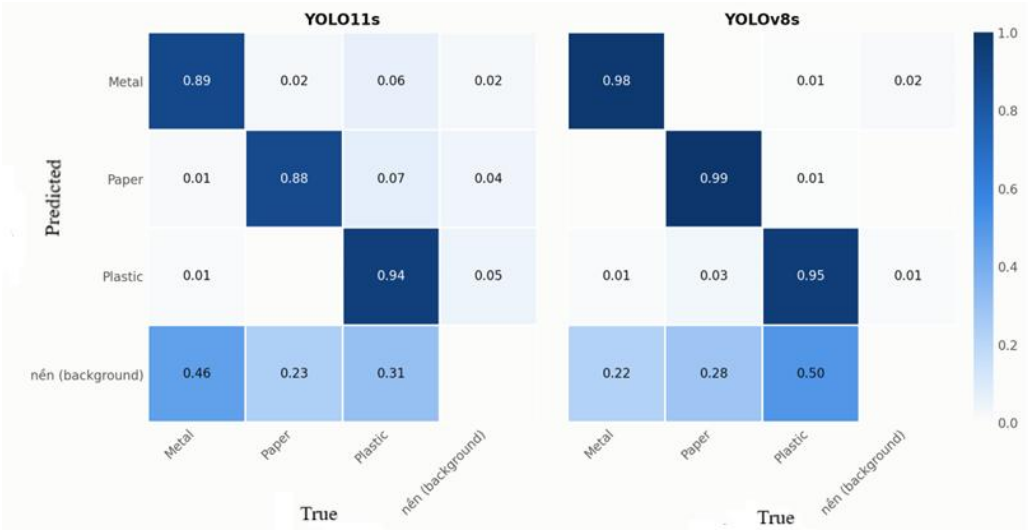


Fig. 5. Confusion matrices of the two YOLO models on the EPU_waste test split.

As Fig. 5 shows, the concentration of samples along the main diagonal indicates that both models identify the majority of instances in the three classes. The diagonal is stronger for YOLOv8s-seg, which reaches class accuracies of 0.98, 0.99 and 0.95 for metal, paper and plastic respectively, against 0.89, 0.88 and 0.94 for

YOLO11s-seg. The principal source of confusion is between paper and plastic, which is consistent with their structural similarity once crushed or deformed. YOLO11s-seg mislabels 6–7% of metal and paper instances as plastic. The false negative rate of both models remains below 5%.



Fig. 6. Segmentation results on the EPU_waste test split.

2.4 Response time

Table 4 reports the processing time of the two models. On the Quadro RTX 3000 GPU YOLOv8s-seg requires 17.70 ms per image (56.5 FPS) against 19.75 ms (50.6 FPS) for YOLO11s-seg, a margin of about 10%. On the Intel Core i7-10850H CPU YOLOv8s-seg remains faster, but the margin falls to about 3% (137.07 ms against 140.86 ms, i.e. 7.3 and 7.1 FPS). Preprocessing and postprocessing take 2–3 ms for both models, so the difference arises almost entirely in the inference step, which consumes roughly 96% of the total on the CPU and 73–77% on the GPU. The GPU raises the overall rate by a factor of seven to eight. Both models exceed 50 FPS on the GPU and so satisfy the real-time constraint of a sorting conveyor; the 7 FPS obtained on the CPU is adequate only for still-frame operation.

Table 4. Inference Speed of Models on EPU_waste (Batch = 1, Image Size = 640)

Model	Device	Hardware	Pre-processing (ms)	Inference (ms)	Post-processing (ms)	Total (ms)	FPS
YOLO11s-seg	CPU	Intel Core i7-10850H 2.70 GHz	3.11	135.04	2.72	140.86	7.1
YOLOv8s-seg	CPU	Intel Core i7-10850H 2.70 GHz	2.70	131.82	2.55	**137.07**	**7.3**
YOLO11s-seg	GPU	Quadro RTX 3000 (FP32)	2.09	15.18	2.47	19.75	50.6
YOLOv8s-seg	GPU	Quadro RTX 3000 (FP32)	2.20	12.91	2.59	**17.70**	**56.5**

Being both more accurate in recall and faster, YOLOv8s-seg is adopted for the remainder of this work.

2.5 Scope of the conclusions

Three limitations bound the conclusions just drawn. First, the dataset was collected in a single environment with three waste classes and limited diversity of background, illumination and object deformation; the results therefore characterize the intended deployment environment rather than waste classification in general. Second, each model was trained once, so differences below roughly 0.02 cannot be separated from the variation caused by random initialization, and only the gap in recall, 0.09, is large enough to support a

conclusion. Third, all metrics are measured against one annotation convention, so they compare the two models fairly with each other but say nothing about the absolute error with respect to the true object boundary. Extending the dataset to several backgrounds and lighting conditions, and repeating the training with several random seeds, are the two steps required before any more general statement can be made.

III. NEURAL-NETWORK INVERSE KINEMATICS MODULE

3.1 Manipulator parameters and the Denavit–Hartenberg table

The manipulator has five revolute joints. The first four determine the position of the gripper centre and are the subject of this section; the fifth only rotates the end link about its own axis and therefore does not affect position. Link dimensions and joint limits are listed in Table 5, and the frames attached to the links are shown in Fig. 7.

Table 5. Link dimensions and joint limits of the manipulator.

Parameter	Symbol	Value	Joint	Lower limit (rad)	Upper limit (rad)
Base height	d_1	220 mm	θ_1	−2.9322	2.9322
Upper arm	a_2	250 mm	θ_2	−0.1571	2.7611
Forearm	a_3	185 mm	θ_3	−1.5359	1.4486
Wrist offset	d_5	150 mm	θ_4	−0.1379	3.2795

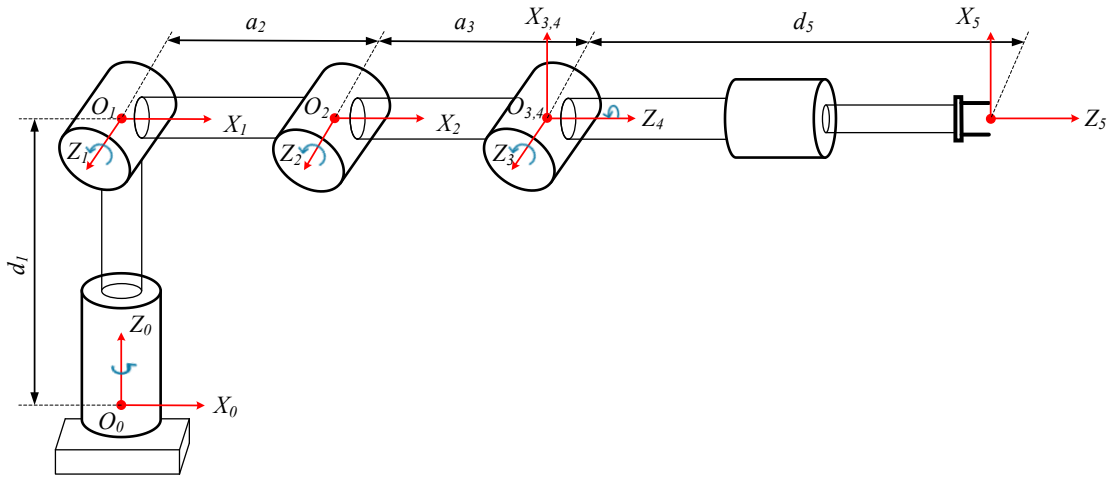


Fig. 7. Frames attached to the links of the 5-DOF manipulator.

The kinematic model follows the Denavit–Hartenberg convention [3]. The four parameters of each link are given in Table 6.

Table 6. Denavit–Hartenberg parameters of the manipulator.

Link	θ_i	d_i	a_i	α_i
1	θ_1	d_1 = 220 mm	0	90°
2	θ_2	0	a_2 = 250 mm	0
3	θ_3	0	a_3 = 185 mm	0
4	θ_4	0	0	90°
5	θ_5	d_5 = 150 mm	0	0

Where:

- θ_i is the rotation about z_{i-1} that brings x_{i-1} parallel to x_i;
- d_i is the translation along z_{i-1};
- a_i is the link length, that is, the translation along x_i;
- α_i is the twist about x_i that brings z_{i-1} parallel to z_i.

3.2 Forward kinematics

The homogeneous transformation between two consecutive frames is the product of the four elementary transformations corresponding to the four parameters in Table 6,

$$A_i = \mathrm{Rot}_z(\theta_i) \cdot \mathrm{Trans}_z(d_i) \cdot \mathrm{Trans}_x(a_i) \cdot \mathrm{Rot}_x(\alpha_i) \tag{7}$$

and the matrix relating the end-link frame to the base frame is the product of those five matrices:

$$T = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5 \tag{8}$$

Carrying out the multiplication and simplifying, the position of the gripper center takes a separable form, exactly the structure imposed by the two 90° twists in Table 6: the first joint merely rotates the whole mechanism about the vertical axis, and what remains is a planar three-link problem. Writing

$$s = \theta_2 + \theta_3 + \theta_4 \tag{9}$$

for the orientation angle of the end link in that plane, the horizontal reach r and the height z of the gripper center are

$$\begin{aligned} r &= a_3 \cos(\theta_2 + \theta_3) + a_2 \cos\theta_2 + d_5 \sin s \\ z &= d_1 + a_3 \sin(\theta_2 + \theta_3) + a_2 \sin\theta_2 - d_5 \cos s \end{aligned} \tag{10}$$

and the position in the base frame follows from a rotation of θ_1 about the vertical axis:

$$P_x = r\cos\theta_1 , \qquad P_y = r\sin\theta_1 , \qquad P_z = z \tag{11}$$

The three expressions (9)–(11) are also the formulae used to generate the training data in Section 3.4.

3.3 Inverse kinematics and design of the input vector

The inverse problem is not single-valued, because the three position equations (9)–(11) constrain four joint variables and one degree of freedom is left over; it is parameterized here by the approach angle φ = s. Once φ has been fixed the problem reduces to two links and has a solution only when the distance from the origin of the second link to the wrist point lies between $|a_2 - a_3|$ = 65 mm and $a_2 + a_3$ = 435 mm. Even then two branches remain, corresponding to the two signs of $\theta_3 = \pm \arccos c_3$ (elbow up and elbow down); near the two reach limits just mentioned the arccosine is ill-conditioned, so a small perturbation of the input produces a large deviation in θ_3.

This property dictates the design of the network input. A network receiving only (P_x, P_y, P_z) would be forced to learn a one-to-many mapping, and a feedforward regressor trained under a least-squares criterion on such a mapping converges to the mean of the admissible branches, a mean that is in general not a solution at all. The input vector is therefore extended to seven components,

$$x = \left[P_x, P_y, P_z, \varphi, \theta_1^c, \theta_2^c, \theta_3^c\right]^T , \qquad y = [\theta_1, \theta_2, \theta_3, \theta_4]^T \tag{12}$$

Supplying φ removes the redundancy, while the three context components θ_1^c, θ_2^c, θ_3^c describe the configuration the arm currently occupies and select the branch nearest to it. The idea of adding the current joint configuration to the input vector is taken from Almusawi et al. [4], who draw the context from the preceding point of a real trajectory; here the context is generated by adding uniformly distributed random noise of amplitude ± 0.15 rad to the true joint angles, and this amplitude is a free parameter of the data-generation stage rather than a value taken from a source.

3.4 Dataset, network structure and the Levenberg–Marquardt algorithm

Input–output pairs are generated from the forward kinematics (9)-(11): a set of joint angles is drawn within the limits of Table 5, the corresponding Cartesian point is computed, and the pair is retained if it satisfies the reach condition of Section 4.3 and lies inside the working region. The resulting 20,000 samples were split once into 16,000 for training, 2,000 for validation and 2,000 for testing, and the same split was reused in every run so that the results remain comparable with one another. Inputs and outputs are both scaled to [−1, 1]. The network configuration is summarized in Table 7.

Table 7. Network configuration and training parameters.

Item	Value
Architecture	7 – 15 – 15 – 4 (424 weights and biases)
Transfer functions	tansig, tansig, purelin
Input/output scaling	min–max to [−1, 1]
Training algorithm	Levenberg–Marquardt [12], [13], [14]
Weight initialization	Nguyen–Widrow [15]
Data split	16,000 / 2,000 / 2,000 (fixed indices)
Stopping criterion	100 consecutive validation failures
Independent runs	10 (different random seeds)

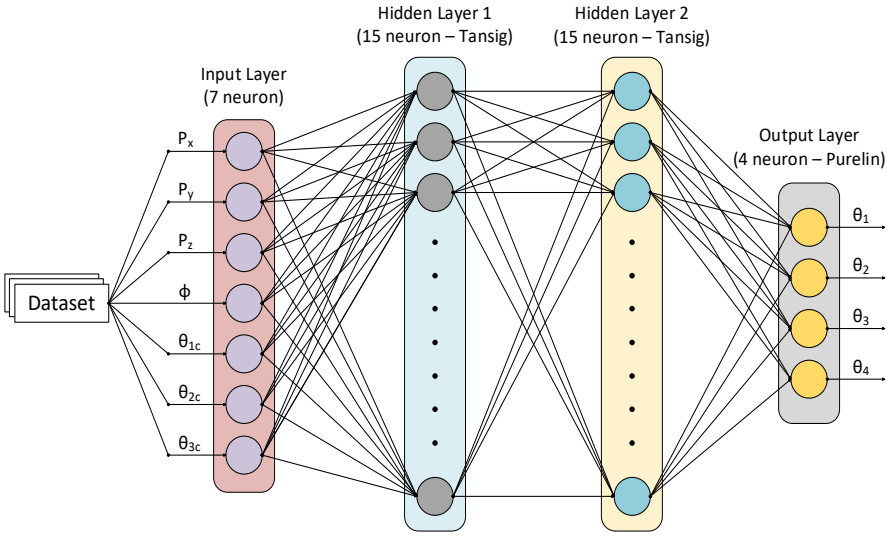


Fig. 8. Architecture of the 7–15–15–4 network.

Let $w \in \mathbb{R}^{424}$ denote the vector collecting all weights and biases of the network, and $e(w) \in \mathbb{R}^{4m}$ the vector of differences between the network output and the desired output over m training samples. The objective to be minimised is

$$xE(w) = \frac{1}{2} e^{\mathrm{T}}(w) e(w) \tag{13}$$

The Levenberg–Marquardt algorithm [12], [13] updates the weights according to

$$w_{k+1} = w_k - \left(J_k{}^{\mathrm{T}} J_k + \mu_k I\right)^{-1} J_k{}^{\mathrm{T}} e(w_k) \tag{14}$$

where $J_k = \partial e / \partial w^{\mathrm{T}}$ is the $4m \times 424$ Jacobian evaluated at w_k, I is the identity matrix of the same order and $\mu_k > 0$ is a regularization coefficient. The product $J^{\mathrm{T}}J$ is the Gauss–Newton approximation of the Hessian: it requires only first derivatives yet still carries second-order curvature information about the error surface, so near a minimum the update (14) converges appreciably faster than a gradient-descent step.

The coefficient μ_k mediates between two regimes. As $\mu_k \to 0$, expression (14) becomes a pure Gauss–Newton step, quadratically convergent but trustworthy only while the Hessian approximation remains good. For large μ_k, $(J^{\mathrm{T}}J + \mu I)^{-1} \approx \mu^{-1} I$ and (14) degenerates into a gradient-descent step with a small step size, slow but always in a descent direction. After each update, μ is divided by a factor β (here $\beta = 10$) if E has decreased, moving the algorithm towards the Gauss–Newton regime; if E has increased, the step is rejected and μ is multiplied by β. The objective therefore never increases, while the second-order rate is still exploited whenever it is available.

The cost per iteration is governed by forming and solving the system (14): forming $J^{\mathrm{T}}J$ takes $O(4m \cdot n^2)$ operations and solving the linear system $O(n^3)$, with $n = 424$. The $O(n^3)$ term is why the algorithm suits networks of a few hundred to a few thousand weights and cannot be used for deep networks with millions of parameters. For a network of the present size, Hagan and Menhaj [14] showed Levenberg–Marquardt to be faster than gradient descent by several orders of magnitude. Training was repeated ten times from different random initializations, so that the accuracy reported is a distribution rather than a single draw.

3.5 Results

The evaluation metric is the root-mean-square error of the joint angles on the test split,

$$\mathrm{RMSE} = \sqrt{\frac{1}{4m} \sum_{k} \sum_{j} \left(\hat{\theta}_{jk} - \theta_{jk}\right)^2} \tag{15}$$

where m is the number of test samples. Over the ten runs the network attains $2.2164 \pm 0.1811°$, a coefficient of variation of 8.17%. A single run of this same network may return any value between 1.89° and 2.48°, so an accuracy figure quoted from one run can be in error by as much as 13%. The error decomposed by joint is given in Table 8.

Table 8. Per-joint RMSE of the inverse-kinematics network, averaged over ten runs.

Quantity	θ_1	θ_2	θ_3	θ_4	Overall
RMSE (deg)	2.0471	1.5736	3.0527	1.9032	2.2164

The third joint is consistently the worst, about 1.9 times worse than the second. This follows directly from the structure of the closed-form solution rather than from the training process: θ_3 is obtained as $\pm \arccos c_3$,

whose derivative diverges at the two reach limits noted in Section 4.3, so geometry alone supplies less information about θ_3 than about any of the other joints. The network compensates by leaning more heavily on the context input θ_3^c, and in doing so inherits a larger share of the noise deliberately injected into that component.

So that the dispersion just quoted can be judged directly, Table 9 lists all ten runs.

Table 9. Summary of the ten independent runs of the inverse-kinematics network.

Run	Test RMSE (deg)	Deviation from mean (deg)	Best epoch	Total epochs
1	2.1917	−0.0247	1,341	1,441
2	2.3753	+0.1589	813	913
3	2.4805	+0.2641	6,921	7,021
4	1.8926	−0.3238	5,764	5,864
5	2.3282	+0.1118	1,306	1,406
6	2.1849	−0.0315	3,435	3,535
7	2.2627	+0.0463	5,968	6,068
8	2.3275	+0.1111	1,503	1,603
9	1.9654	−0.2510	10,892	10,992
10	2.1551	−0.0613	1,370	1,470
Mean	**2.2164**	—	**3,931**	**4,031**
Standard deviation	**0.1811**	—	—	—

Run 1 departs least from the mean, by only 0.0247°, and was therefore chosen as the representative run for Fig. 9 and for all later illustrations. The best run (run 4, 1.8926°) was deliberately not chosen, so that the figures reflect the typical behavior of the method rather than its most favorable outcome.

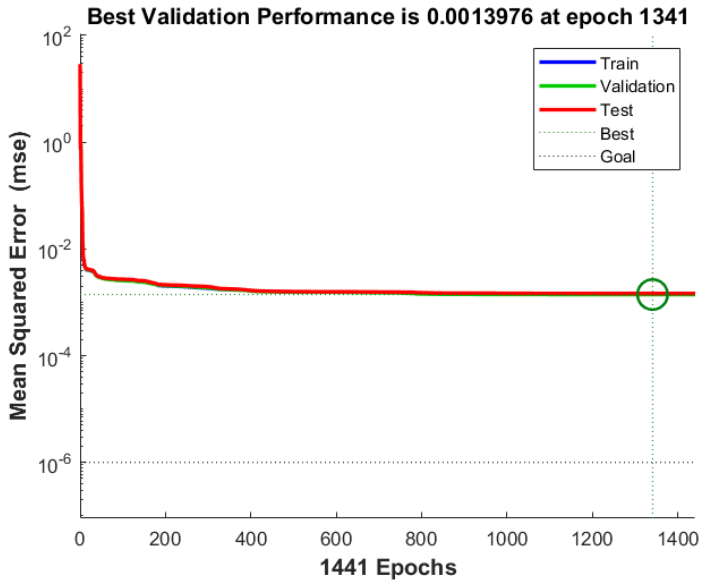


Fig. 9. Convergence of the Levenberg–Marquardt algorithm in the representative run (run 1).

3.6 Verification on a continuous trajectory

The results above were measured on a test split of discrete points spread over the working region. To verify the network in continuous operation, the representative network was embedded in a Simulink model and made to track a horizontal circular trajectory centered at r = 350mm, z = 300mm, of radius 50mm and period 10s:

$$P_x^d = 350 + 50\sin\left(\frac{2\pi}{10}t\right)$$
$$P_y^d = 50\cos\left(\frac{2\pi}{10}t\right) \quad (16)$$
$$P_z^d = 300$$

At each step the three context inputs are taken from the network output at the preceding step through a unit-delay block, exactly as the arm operates when traversing a sequence of successive points. The instantaneous position error was recorded after discarding the first 0.2 s so as to remove the start-up transient; the result is shown in Fig. 10.

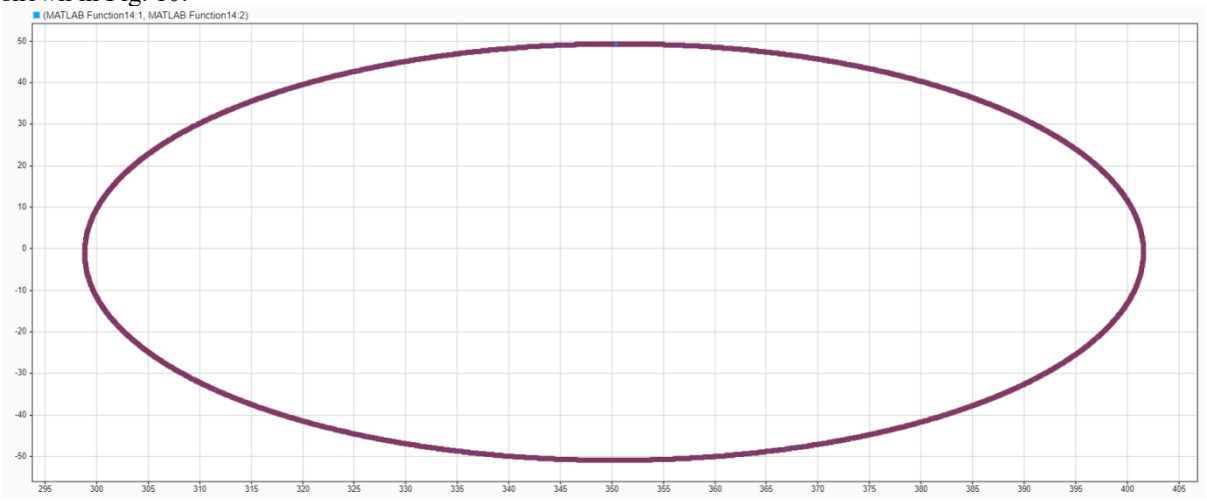


Fig. 10. Position error against time while tracking the circular trajectory.

Once the first 0.2 s have been discarded, the RMS error is 3.461 mm, the mean 3.208 mm and the maximum 4.992 mm. The mean lies close to the RMS, so the standard deviation is only 1.30 mm: the curve sits at a nearly constant level rather than oscillating about the target point. These values are smaller than the 9.96 mm measured on the test split, because the circle examined lies well inside the working region whereas the test split spans the whole of it, including points close to the reach limits.

IV. CONCLUSION

Two functional modules of an automated waste-sorting manipulator have been built and evaluated. On a dataset of 2,955 images collected for the purpose, YOLOv8s-seg outperforms YOLO11s-seg chiefly in recall (0.97 against 0.88) while also running about 10% faster on the GPU, and was selected on that basis. This ordering is the reverse of the ordering published on COCO, which suggests that architectural improvements validated on a large general-purpose dataset should not be assumed to transfer to a small specialized one, although the conclusion is tied to the dataset and imaging conditions of this study and rests on a single run per model, so that only the gap in recall is treated as firm. A 7–15–15–4 network trained by the Levenberg–Marquardt algorithm solves the inverse kinematics of the 5-DOF arm with a joint angle error of 2.2164 ± 0.1811° over ten independent runs, corresponding to 9.96 mm at the gripper. Supplying the approach angle as an input both removes the redundancy of the mapping and, by preserving the sum of the three planar joint angles, cancels part of the Cartesian error that independent angular errors would otherwise have produced.

The two modules are at present evaluated separately; the physical integration of camera, controller and manipulator, together with a measurement of the end-to-end placement error of the complete system, is the next step. The work remaining includes extending the dataset to several backgrounds and lighting conditions, repeating the segmentation training with several random seeds, and replacing the artificial context noise with the configuration at the preceding trajectory point, as in the original formulation of Almusawi et al.

ACKNOWLEDGMENTS

This research is funded by Electric Power University under research project code ĐTNH.33/2026.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 779–788.

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), 2017, pp. 2961–2969.

[3] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," ASME J. Appl. Mech., vol. 22, pp. 215–221, 1955.

[4] A. R. J. Almusawi, L. C. Dülger, and S. Kapucu, "A new artificial neural network approach in solving inverse kinematics of robotic arm (Denso VP6242)," Comput. Intell. Neurosci., vol. 2016, Art. no. 5720163, 2016.

[5] Z. Zhang, "A flexible new technique for camera calibration," IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 11, pp. 1330–1334, 2000.

[6] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," Pattern Recognit., vol. 47, no. 6, pp. 2280–2292, 2014.

[7] P. F. Proença and P. Simões, "TACO: Trash annotations in context for litter detection," arXiv:2003.06975, 2020.

[8] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in Computer Vision – ECCV 2014, Lecture Notes in Computer Science, vol. 8693, Cham, Switzerland: Springer, 2014. https://doi.org/10.1007/978-3-319-10602-1_48

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) challenge," Int. J. Comput. Vis., vol. 88, pp. 303–338, 2010. https://doi.org/10.1007/s11263-009-0275-4

[10] Z. Li, "Research on the application of machine vision in industrial robots," Explor. Sci. Acad. Conf. Ser., vol. 20, pp. 164–172, Apr. 2026, doi: 10.70267/mseac.2026164172.

[11] Ultralytics, "YOLOv8 vs YOLO11 comparison," Ultralytics Docs, 2024. [Online]. Available: https://docs.ultralytics.com/compare/yolov8-vs-yolo11. [Accessed: 19-Oct-2026].

[12] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," J. Soc. Ind. Appl. Math., vol. 11, no. 2, pp. 431–441, 1963.

[13] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," Quart. Appl. Math., vol. 2, no. 2, pp. 164–168, 1944.

[14] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," IEEE Trans. Neural Netw., vol. 5, no. 6, pp. 989–993, 1994.

[15] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in Proc. Int. Joint Conf. Neural Netw. (IJCNN), San Diego, CA, USA, 1990, pp. 21–26.