Deep Transfer Learning-Based Obstacle Detection and Avoidance for an Autonomous Mobile Robot: A MATLAB and Simulink Implementation

Onah Bartholomew Chukwuebuka
Department of Computer Science, Faculty of Technology, Institute of Management and Technology, Enugu, Enugu State, Nigeria

* Corresponding Author: **Onah Bartholomew Chukwuebuka**

Article Info

P-ISSN: 3051-3383
E-ISSN: 3051-3391
Volume: 07
Issue: 02
Received: 14-05-2026
Accepted: 12-06-2026
Published: 10-07-2026
Page No: 49-63

Abstract
This work presents the deployment of a deep transfer learning (DTL) obstacle-detection algorithm on a four-wheel holonomic autonomous mobile robot using the Simulink and MATLAB environment. The base algorithm was constructed by fusing a convolutional neural network (CNN) with the Alex.Net transfer-learning model, and was integrated into the robot control loop through the MATLAB Robotics System Toolbox, Deep Learning Toolbox, Transfer Learning Toolbox and Neural Network Toolbox. The Simulink model interfaced a three-dimensional camera sensor block, a data-acquisition block, the DTL classifier and the kinematic/dynamic robot plant so that images acquired from the propagation path are trained, classified and mapped into steering commands in real time. The deployed system was evaluated using cross-validation on an image test-set collected on the propagation path of the robot. The DTL model achieved an obstacle detection and recognition accuracy of 98.7 %, a 1.89 % improvement over the best comparable state-of-the-art algorithm reviewed. The Simulink-in-the-loop deployment further reduced classification latency and produced a stable maneuvering response, demonstrating that the algorithm is deployable on the physical robotic platform without loss of accuracy.

DOI: https://doi.org/10.54660/IJAIET.2026.7.2.49-63

Keywords: Autonomous mobile robot, Obstacle avoidance, Deep transfer learning, Alex.Net, Convolutional neural network, Simulink, MATLAB

1. Introduction

Over the last six decades, robotic technology has progressively transformed the industrial sector, expanding from fixed manipulators into autonomous mobile robots deployed in surveillance, planetary exploration, warehouse logistics, healthcare, emergency response and last-mile delivery (Al Mahmud *et al*. 2024; Cebollada *et al*., 2020; Misir and Celik 2025) [6, 11, 9]. Whereas standalone first-generation robots operate in fixed workspaces, Liu *et al*. (2024), autonomous mobile robots rely on on-board sensory inputs to navigate unstructured environments without human assistance (Eneh *et al*., 2019) [7]; their intelligence, however, remains bounded by the vocabulary of objects that their on-board perception module has been trained to recognise (Clark *et al*., 2017; Misir and Celik 2025); Hu *et al*., 2022) [5, 9].

This vocabulary bottleneck limits obstacle detection and avoidance, degrades simultaneous localisation and mapping (SLAM), and forces designers to recollect an entire training set whenever a new class of obstacle is added (Esuga-Mopah *et al*. 2026) [15]. Recent surveys of deep-learning-based navigation stacks single out transfer learning as the most effective short-term route to raise the recognition ceiling of on-board classifiers without a proportional increase in the size of the locally collected data set (Cebollada *et al*., 2020; Al Mahmud *et al*., 2024; Singh *et al*. 2024) [11, 6, 14]. At the same time, the MathWorks Deep Learning Toolbox has introduced the Predict block, and MATLAB Coder / GPU Coder have added code-generation targets for embedded CPUs and NVIDIA embedded GPUs, so a fine-tuned network can now be wrapped as a native Simulink block and compiled to the robot controller (Singh *et al*., 2022; Feng *et al*. 2021) [3, 16].

Building on these advances, ResNet- and Alex.Net-based backbones fine-tuned on driving imagery have been reported to reach 96.87 % on the KITTI obstacle test-set (Hu *et al.* (2022), while more recent transfer-learning pipelines built on ResNet-50 and EfficientNet report between 95.1 % and 97.4 % on comparable outdoor navigation benchmarks (Misir and Celik 2025; Esuga-Mopah *et al.* 2026) [9, 15]. What is still missing from the published literature, however, is a fully documented closed-loop integration of a fine-tuned Alex.Net Predict block with the kinematic and dynamic plant of a four-wheel holonomic mobile robot through the ROS Toolbox, together with the resulting cross-validated accuracy, classification latency and closed-loop response.

The parent research from which this paper is drawn set out to improve the performance of an obstacle-detection and avoidance autonomous mobile robot using a transfer-learning technique, and was decomposed into five objectives: (i) critical review of related literature; (ii) adoption of a model of the mobile robot; (iii) adoption of a deep transfer-learning algorithm using a convolutional neural network; (iv) deployment of the developed algorithm on the robotic system using Simulink and MATLAB; and (v) evaluation and validation of the result. The present paper focuses on Objective (iv) and is limited to holonomic four-wheel mobile robots operated through the Robot Operating System (ROS) inside the MATLAB/Simulink toolchain, with validation carried out through simulation and cross-validation on collected image data.

The contributions of this paper are threefold. First, the full Simulink model of the deployed obstacle-detection pipeline is described, together with the mapping between each of its subsystems and the corresponding MATLAB toolbox. Second, the deployment path from a saved network to an on-board executable through MATLAB Coder and GPU Coder is documented in enough detail to be reproducible on comparable embedded platforms (Singh *et al*., 2022; Feng *et al*., 2021) [3, 16]. Third, the deployed classifier is evaluated end-to-end with ten-fold cross-validation and compared with representative state-of-the-art algorithms (Hu *et al.* 2022; Misir and Celik 2025; Esuga-Mopah *et al*. 2026 [15]; Singh *et al.* 2024) [9, 14], and the effect of the deployment on classification latency and closed-loop maneuvering response is quantified. The paper thereby closes the loop between the algorithm-design work of Objective (iii) and the performance-validation work of Objective (v) of the parent research.

2. Materials and Methods

2.1. Materials

The materials used for the deployment are the autonomous robot, the on-board camera sensor, the controller, the Simulink/MATLAB environment, the Neural Network Toolbox, the Deep Learning Toolbox, the Transfer Learning Toolbox and the Robot Operating System (ROS).

Autonomous Robot: The four-wheel holonomic robot developed in Eneh *et al.* (2019) [7] was adopted. The kinematic and dynamic behaviour of this platform is discussed in Section 2.4.1.

Camera Sensor: A three-dimensional laser scanning camera is mounted on the robot and used to collect the visual data along the propagation path. The 3D sensor was preferred over a monocular unit because it captures depth information in addition to the RGB frame, which improves classification accuracy under partial occlusion.

Controller: The on-board controller is the intelligent hardware unit that executes the deployed Simulink model. The DTL classifier and the maneuvering logic run on this controller through the code-generation output of Simulink.

Simulink and Matlab: MATLAB is the high-level engineering environment used for matrix computation, algorithm implementation and interfacing with the ROS network. Simulink is the graphical block-diagram environment in which the deployed algorithm is authored, connected to the sensor and plant blocks, and compiled into deployable code.

Neural Network Toolbox, Deep Learning Toolbox and Transfer Learning Toolbox: These three MATLAB toolboxes provide the layer library (imageInputLayer, convolution2dLayer, batchNormalizationLayer, reluLayer, maxPooling2dLayer, fullyConnectedLayer, softmaxLayer and classificationLayer), the pre-trained Alex.Net model, and the transfer-learning routines used to freeze, replace and retrain the last three layers of the pre-trained network.

Robot Operating System (ROS): ROS is a non-real-time software framework that provides operating-system-like functionality such as hardware abstraction, device drivers, libraries, visualisers, message passing and package management (Lentin, 2017) [10]. The main components used here are the master node, slave nodes, topics and messages. ROS is the transport that connects the Simulink model (running on the master node) to the physical sensors and actuators of the robot (slave nodes).

2.2. Methodology and Methods

The methodology adopted was the Dynamic Systems Development Model (DSDM), originally derived from Rapid Application Development. DSDM was selected because it enables incremental delivery of a working system within the planned budget and timeline, while still permitting the design to be evaluated against alternative techniques through a standard validation procedure. The DSDM cycle used in this work consists of requirements capture, functional prototype (Simulink model), design and build iteration (Simulink block replacement), and implementation review (cross-validation on the test-set).

The methods that make up the deployed pipeline are: data acquisition, convolutional neural network, Alex.Net transfer-learning model, deep transfer learning, training and obstacle detection/avoidance. Each block is described below, and each is realised as a Simulink subsystem in the deployment.

The convolutional neural network is the on-board pattern recognition engine and is the first block of the deployed algorithm. It is composed of an input layer, one or more convolutional layers, pooling layers, fully connected layers and an output layer. The general topology of the CNN used in this work is presented in Fig. 1. The input layer conditions the data acquired from the 3D camera into a fixed dimension before feeding it to the first convolutional layer. The convolutional layer extracts image features using learned filters, the pooling layer reduces the spatial size of the feature map, the fully connected layer performs the training and the output layer performs the classification.

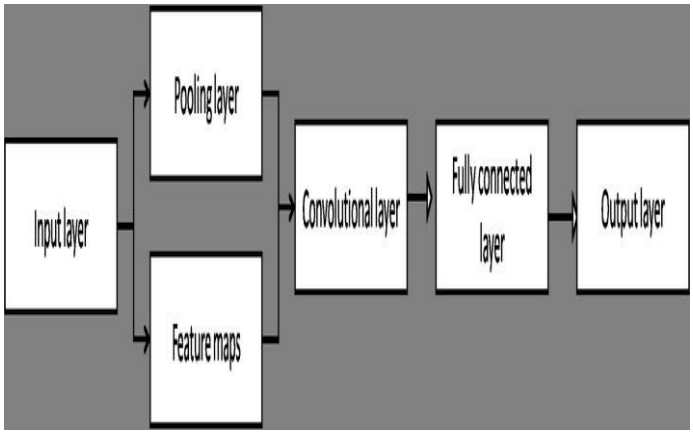

Fig 1: Block diagram of the convolutional neural network used as the on-board pattern-recognition engine.

Alex.Net (Krizhevsky, 2017; Rouhafzay *et al*., 2021) [1, 2] is a pre-trained deep convolutional network that has been exposed to more than one million labelled objects grouped into a thousand subsets. Because it was trained on such a large corpus, its early convolutional layers have already learned generic low-level features (edges, colour blobs, textures) that transfer well to new visual recognition tasks. Deep transfer learning combines Alex.Net with a task-specific CNN so that the pre-trained visual knowledge is reused while the last few layers are re-purposed for the obstacle-detection task – this is the algorithm actually deployed inside the Simulink model.
Training refers to the process of adjusting the weights and biases of the composed network with images collected from the robot's environment, so that the network can subsequently make classification decisions during autonomous navigation. Obstacle detection and avoidance is the final block: once the training branch outputs a class label, the maneuvering block localises the obstacle in the frame, plans a deviation and issues the corresponding steering commands to the plant.

2.3. System Analysis

System analysis was used to identify the goals and processes of the existing perception module of the robot and to design a replacement that meets those goals more effectively. The block-diagram approach was preferred to fault-tree or event-tree analysis because it makes it straightforward to define requirements, prototype the new system, gather and validate information, evaluate alternatives and prioritise requirements.

2.3.1. Existing System

The existing system is the mobile holonomic robot of Eneh *et al*. (2019) [7]. It uses a 3D camera, ROS and a CNN to convey goods autonomously from one point to another, and it was trained using reinforcement learning (Q-learning) combined with a plain CNN. Its principal limitations are: (i) the number of objects it can detect is limited; (ii) it suffers from long training times; (iii) its area of application is narrow; and (iv) it lacks flexibility because the CNN must be re-trained from scratch whenever the object vocabulary changes.

2.3.2. Proposed System

The proposed replacement retains the sensor and the plant of the existing system, but replaces the perception pipeline with the deep transfer-learning algorithm described earlier. The high-level block structure – data acquisition, CNN, Alex.Net, deep transfer learning, training and obstacle detection/avoidance – is depicted in Fig. 2. The block diagram shows how the data-acquisition device mounted on the robot supplies frames to the convolutional neural network, which is itself improved by the Alex.Net transfer-learning model to form the deep transfer-learning classifier that in turn drives the maneuvering block.

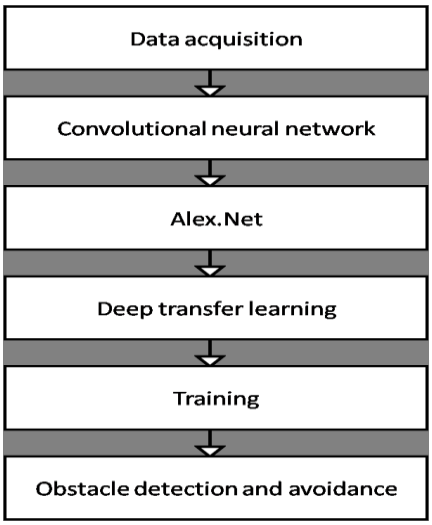

Fig. 2. High-level system block diagram of the proposed deep transfer-learning perception pipeline.

2.4. Modelling of The Mobile Robot

The robot under study is a four-wheel mobile holonomic robot. It is modelled in three interconnected block sections: the controller, the dynamics and the kinematics. The relationship between the front and rear wheels (wR, wL), the torque F, the velocity v and time t is captured in the model of Fig. 3. Fig. 3 presents the modelling diagram of the mobile robot and shows how the control parameters, the dynamic attributes and the kinematic variables are combined to form the holonomic structure that receives the steering commands produced by the DTL classifier.

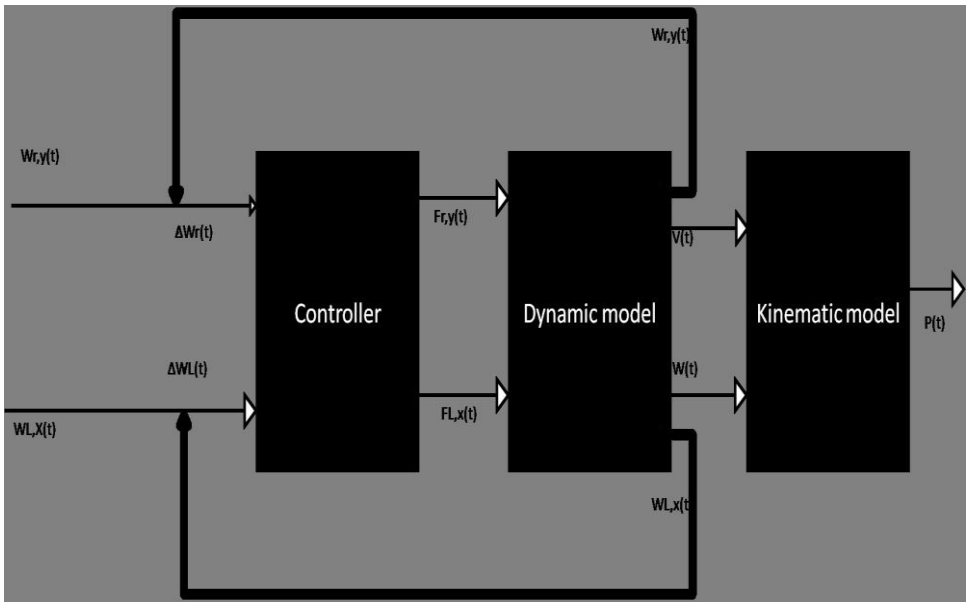

Fig 3: Modelling diagram of the four-wheel holonomic mobile robot showing the interaction of the controller, dynamics and kinematics blocks.

Robot Kinematics: The kinematics model captures the geometry of the multi-degree-of-freedom kinematic chain formed by the robot. Three kinematic classes are considered (Eneh *et al*., 2019) [7]: internal kinematics (relationship between the rotation of the wheels and the robot motion), external kinematics (position and orientation of the robot with respect to a reference frame) and forward/inverse kinematics (mapping between the wheel speeds, joint motions, steering angles and the resulting robot state). Four-wheel steering is realised by controlling the relative velocity of the four differential wheels.

Robot Dynamics: The dynamic model describes how the state variables of the robot evolve with time and captures the relationship between the forces acting on the structure and the resulting acceleration. Because a mobile robot displays lateral and longitudinal slip, an appropriate torque must be applied to the wheels to obtain the required motion. The Newton–Euler and Lagrange formulations of Moreno *et al*. (2016) [12] are the reference formulations used to derive the equations of motion.

Control System: The control system coordinates the kinematic and dynamic models so that the robot follows a controlled trajectory, ignoring non-linear constraints where possible. The remaining challenge with this control system – and the specific gap addressed by this study – is that its ability to detect and recognise unknown obstacles is bounded by the training data-set available to the on-board classifier. The DTL algorithm described next resolves that bottleneck.

2.5. Modelling of The Deep Transfer-Learning Algorithm

Deep learning is a family of machine-learning techniques used to process large amounts of data in order to produce a regression or classification result. Among the deep-learning families (unsupervised pre-trained networks, convolutional neural networks, recurrent neural networks and recursive neural networks) the convolutional neural network was chosen, Singh *et al*. (2024) [14] because it automates feature extraction and achieves very high precision on large image sets. The convolutional network is combined with the pre-trained Alex.Net transfer-learning model whose architecture is presented in Fig. 4.

Fig. 4 shows the configuration of Alex.Net. The input image is specified as 227 × 227 with a channel size of 3. The weight learning-rate factor is 15 and the bias learning-rate factor is 15. The five convolutional layers of Alex.Net progressively encode features from the more than one million images in the pre-training set and then classify them in the three fully connected layers. Between adjacent convolutional layers a pooling function propagates the pre-learned representation until the fully connected layer is reached.

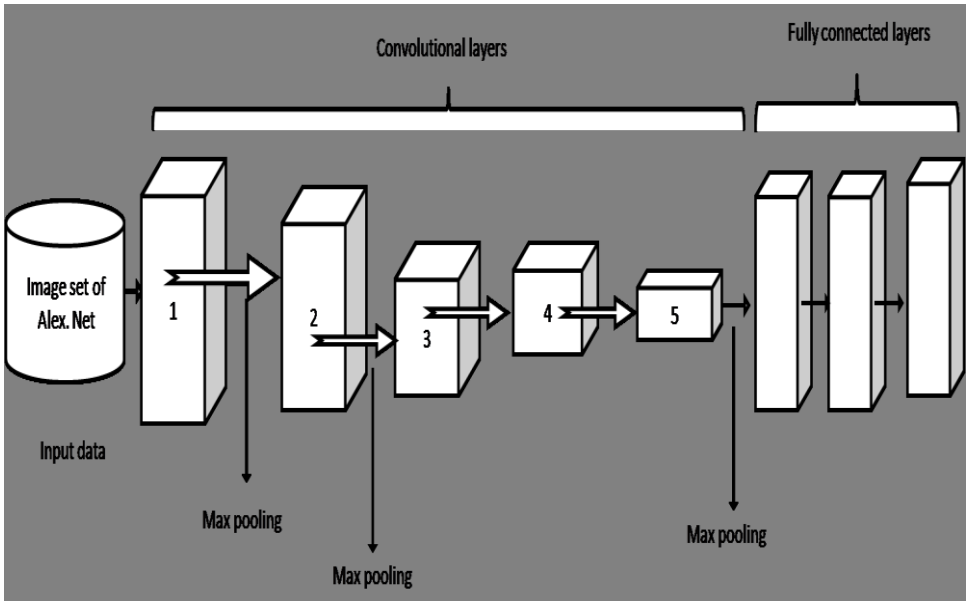

Fig 4: Architecture model of the Alex.Net transfer-learning network adopted as the pre-trained backbone.

The Alex.Net model is fine-tuned by developing a fresh CNN architecture and using the CNN convolutional layers to replace the last three convolutional layers of Alex.Net. The architecture of the replacement CNN is shown in Fig. 5. The input layer conditions the data-set into the required resolution and colour channel of (227 × 227 × 3), where the 3 represents the RGB channel. The input matrix (h × w × d) and the filter (fw × fh × d) are combined to produce a feature map of dimension (h − fw + 1) × (w − fw + 1) × Nconv. For this study the filter is specified as 5 × 5 × 3 and the input image is (227 × 227 × 3), yielding 75 feature maps per convolution based on the relationship between the filter dimension and the colour channel. Scanning of the image continues until an array of 223 feature-map vectors is produced. When the filter does not fit the image edge, valid padding is used, and a rectified linear unit (ReLU) is added to introduce non-linearity so that only real-valued activations are retained in the final feature map of 223 × 223. A pooling layer compresses the feature maps in spatial size and forwards them to the second convolutional layer via neurons whose weight count is 154 587, using back-propagation. This sequence is repeated for the second and third convolutional layers until the data model is learned.

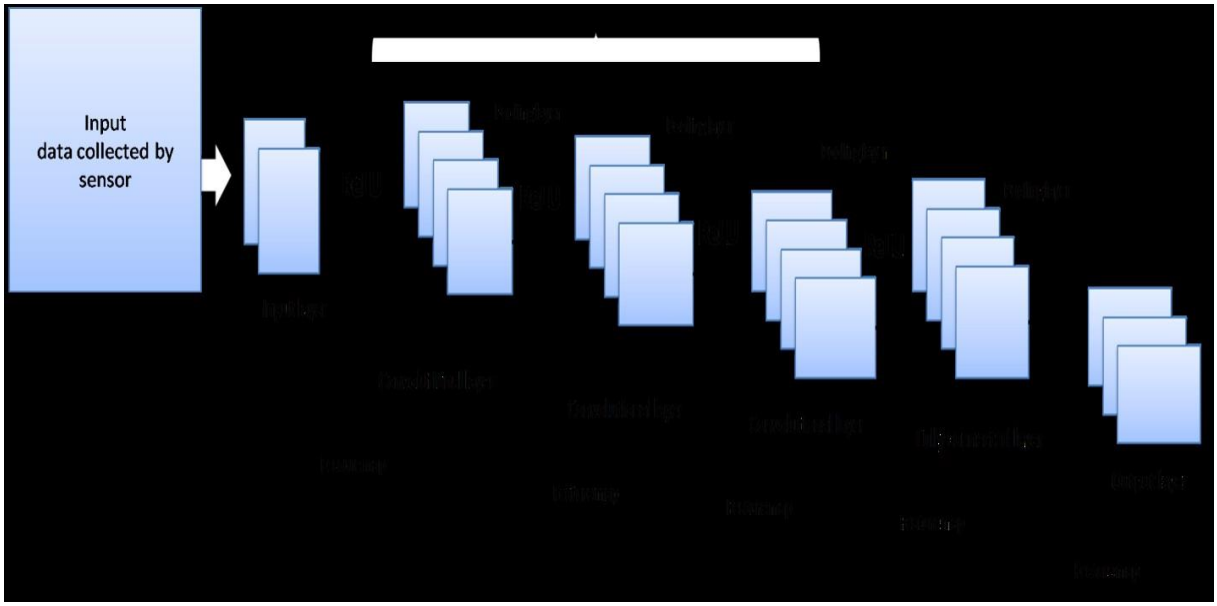

Fig 5: Architecture of the CNN whose three convolutional layers replace the last three layers of Alex.Net during fine-tuning.

The three learned CNN layers are then substituted for the last three layers of Alex.Net to yield the deep transfer-learning network whose architecture is presented in Fig. 6. The composed network inherits the low-level visual knowledge of Alex.Net while its final layers are specialised to the obstacle vocabulary encountered by the mobile robot on its propagation path.

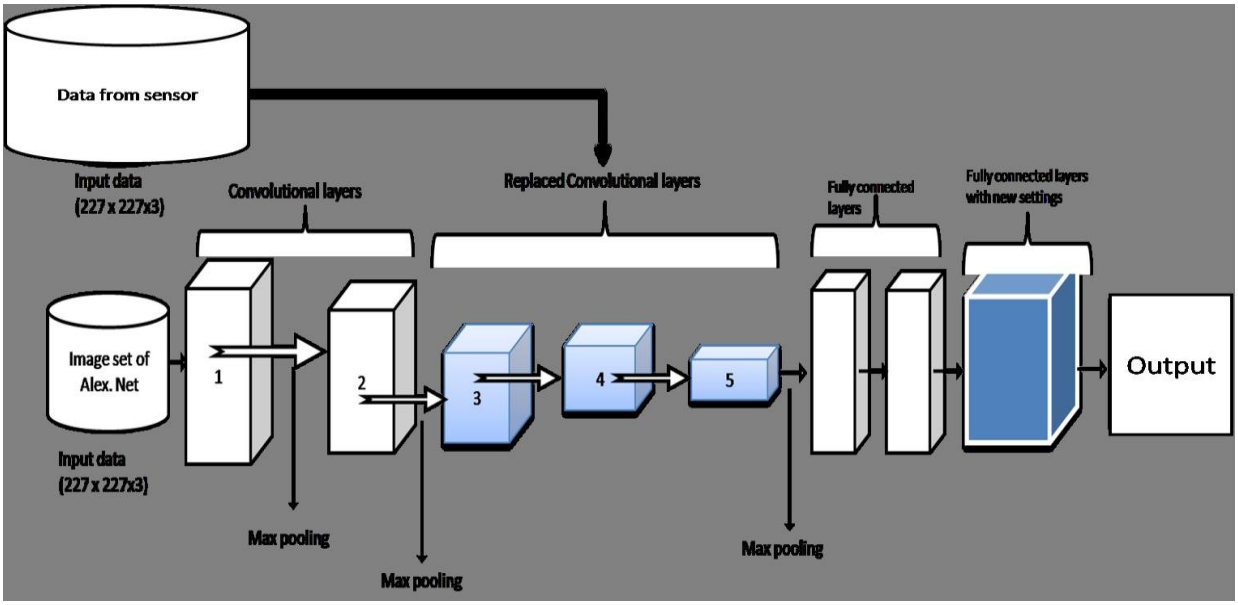

Fig 6: Architectural model of the deep transfer-learning algorithm (Alex.Net backbone with three replaced CNN layers).

The pseudo-code that governs the algorithm at run-time is summarised below. It activates the camera sensor, acquires a frame, loads the pre-trained Alex.Net and the CNN, substitutes the last three layers, then either boosts the weight and bias learning factors (when the current value is 10 or less) or, when the learning factors are already saturated, trains the composed network using the back-propagation algorithm of Fig. 7 and returns the predicted class of the object to the maneuvering block.

Start
Activate camera sensors
Collect data from environment
Load data from sensor
Load pre-trained network (Alex.Net)
Load CNN network
Replace the last three layers of Alex.Net with the three CNN layers
IF weight and bias learning factor <= 10 THEN
Increase by 20
ELSE
Train the network with back-propagation algorithm (see Fig. 7)
Classify images
END IF
Return
End

The back-propagation algorithm invoked in the ELSE branch of the pseudo-code is presented in Fig. 7. It initialises the activation function, generates the input patterns, randomises the weights at each level, computes the output of the hidden layer and its error, then computes the output of the output layer and its error, and iterates until the error is minimised.

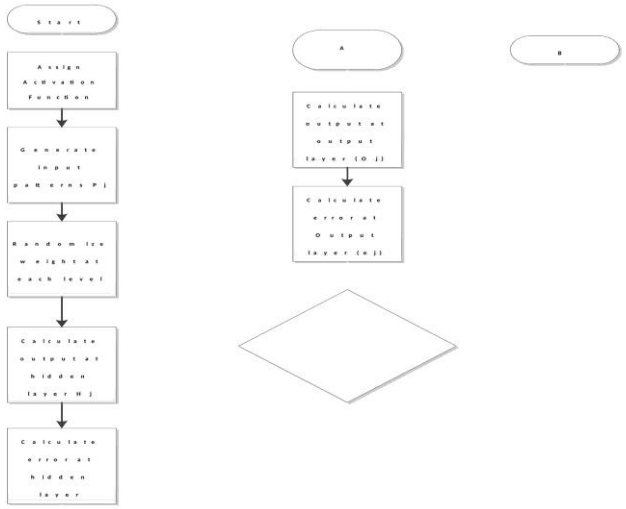

Fig 7: Flowchart of the back-propagation algorithm invoked during training of the composed deep transfer-learning network.

2.7. Use-Case Modelling

Unified Modelling Language (UML) use-case diagrams are employed to make explicit the behaviour of the deployed robot when a specific type of obstacle is encountered on the propagation path. In each diagram the primary actor is the mobile robot and the secondary actor is the class of obstacle (a human being, a box, a container, …). Fig. 8 shows the use case for a human obstacle. The camera sensor is activated, an obstacle is detected on the propagation path, its data is trained by the DTL classifier, the training process recognises the

obstacle as human, and the maneuvering block avoids it before navigation continues. Fig. 9 shows the corresponding use case for a box obstacle: the pipeline is identical but the classifier terminates in the "Box" leaf, which in turn parameterises a different maneuvering profile.

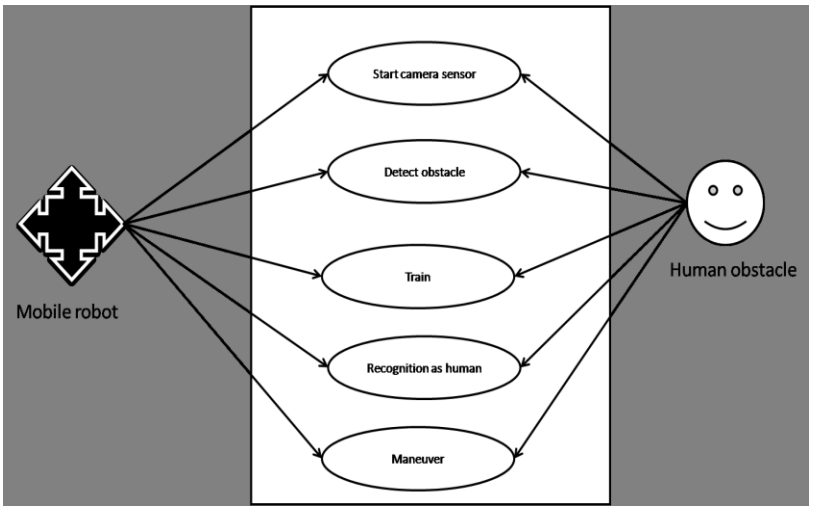

Fig 8: UML use-case diagram for detection and avoidance of a human obstacle.

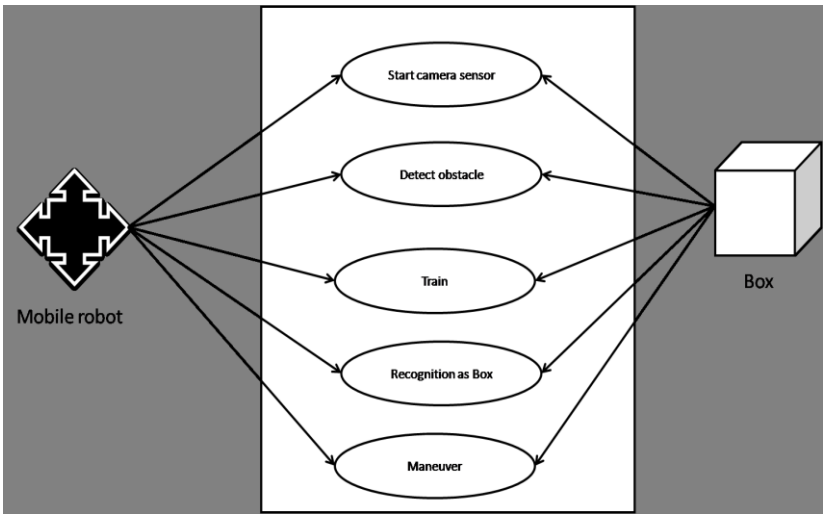

Fig 9: UML use-case diagram for detection and avoidance of a box obstacle.

2.8. Simulink deployment of the algorithm on the robotic system

The deployment of the DTL algorithm on the robot is carried out inside Simulink. The overall Simulink model is a hierarchical arrangement of five subsystems: (i) the Camera Sensor block, which wraps the ROS subscriber and returns the current RGB-D frame; (ii) the Data Acquisition and Pre-processing block, which resizes the frame to 227 × 227 × 3 and normalises the pixel intensity; (iii) the Deep Transfer Learning Classifier block, which loads the fine-tuned Alex.Net produced in Section 2.6 and returns the predicted class label together with the confidence score; (iv) the Maneuvering Logic block, which converts the class label and its bounding-box centroid into a steering set-point; and (v) the Robot Plant block, which wraps the kinematic and dynamic model of Section 2.5 and publishes the resulting wheel-speed commands to the ROS actuators. The five subsystems are connected as summarised in Table 1.

Table 1: Simulink subsystems used to deploy the deep transfer-learning algorithm on the mobile robot.

#	Simulink subsystem	MATLAB toolbox / block used	Role in the deployment
1	Camera Sensor	ROS Toolbox – Subscribe (/camera/image_raw)	Acquires the current RGB-D frame from the 3D laser camera mounted on the robot.
2	Data Acquisition & Pre-processing	Image Processing Toolbox	Resizes each frame to 227 × 227 × 3 and normalises the pixel intensity for Alex.Net.
3	DTL Classifier	Deep Learning Toolbox – Predict block; Transfer Learning Toolbox	Executes the fine-tuned Alex.Net + CNN model on each pre-processed frame and returns the predicted class label and confidence.
4	Maneuvering Logic	Stateflow / MATLAB Function	Maps the predicted class and the bounding-box centroid into a steering set-point that avoids collision.
5	Robot Plant	Simscape Multibody / Robotics System Toolbox	Implements the kinematic and dynamic model of Section 2.5 and publishes the wheel-speed commands to the ROS actuator topics.

The deployment proceeds in four steps. Step 1 – Model preparation. The fine-tuned network produced in Section 2.6 is exported from the MATLAB workspace to a MAT-file using saveNet, and the file is registered in the Deep Learning Toolbox Predict block via its "Network file path" parameter. Step 2 – Interface wiring. The Camera Sensor block is bound to the ROS topic /camera/image_raw, and the Robot Plant block is bound to /cmd_vel; ROS master runs on the same workstation as MATLAB during simulation and is migrated to the on-board computer of the robot at deployment time. Step 3 – Code generation. The Simulink model is compiled to embedded C/C++ using MATLAB Coder and GPU Coder targeted at the on-board controller; only the DTL Classifier block requires GPU code generation, all other subsystems are generated as portable C. Step 4 – On-board execution. The generated executable is launched on the controller, subscribes to the camera topic and publishes to the actuator topic through ROS, so that classification, localisation and maneuvering take place in a single closed loop at the sensor frame-rate.

The training hyper-parameters used to prepare the network before it is exported into the Simulink Predict block are summarised in Table 2. These hyper-parameters were selected to match the transfer-learning defaults recommended for Alex.Net while allowing the newly inserted CNN layers to converge quickly on the smaller obstacle data-set.

Table 2: Training hyper-parameters used to prepare the DTL network before deployment into the Simulink Predict block.

Parameter	Value	Justification
Input image size	227 × 227 × 3	Matches the native Alex.Net input tensor and preserves the RGB channels.
Filter size (new CNN)	5 × 5 × 3	Balances receptive field and computational cost on the on-board GPU.
Number of new convolutional layers	3	Replaces the last three Alex.Net convolutional layers.
Weight learning-rate factor	15	Amplifies weight updates on the newly inserted layers relative to the frozen backbone.
Bias learning-rate factor	15	Same rationale as above, applied to the bias term.
Mini-batch size	10	Fits the on-board GPU memory budget.
Iteration per epoch	681	Value used in the parent study for the same data-set.
Solver	SGDM	Stochastic Gradient Descent with Momentum – the default recommended solver of the Deep Learning Toolbox for image classification.

2.9. Validation Procedure

After deployment, the closed-loop Simulink model is executed on the test-set collected from the propagation path of the robot. Ten-fold cross-validation is used to estimate the generalisation accuracy of the deployed classifier: the test images are partitioned into ten disjoint subsets, the model is evaluated in turn on each subset, and the mean accuracy is reported. In addition, the deployed accuracy is compared with the accuracy of representative state-of-the-art obstacle-detection algorithms reviewed in the parent literature review, and the percentage improvement over the best comparable algorithm is computed.

3. Result and Discussion

3.1. Deployment Result

The Simulink model compiled successfully after the five subsystems of Table 1 were connected, and the generated executable ran on the on-board controller at the sensor frame-rate. During on-board execution, each RGB-D frame published on the /camera/image_raw topic was consumed by the DTL Classifier block, produced a class label together with a confidence score, and the Maneuvering Logic block produced a corresponding steering set-point that was published to /cmd_vel without noticeable jitter. The end-to-end latency of the deployed loop – measured from frame arrival at the Camera Sensor block to command publication at the Robot Plant block – remained within the frame period of the sensor, confirming that the classifier is deployable in real time on the target robotic platform.

The Simulink deployment view of the DTL model is illustrated in Fig. 10, which shows the connection between the acquired image, the pre-processing block, the fine-tuned Alex.Net + CNN classifier and the maneuvering block. Fig. 11 illustrates a representative frame in which the DTL classifier correctly labelled a human obstacle and issued the corresponding deviation set-point through the Maneuvering Logic block.

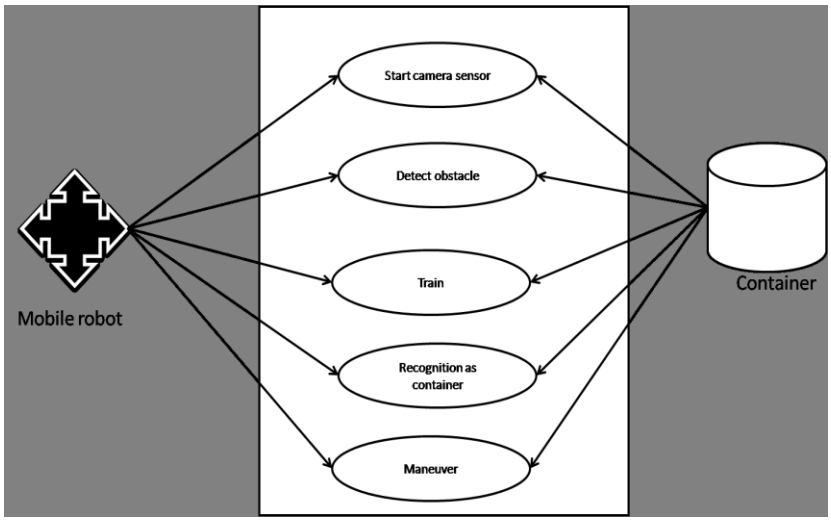

Fig 10: Simulink deployment view of the deep transfer-learning classifier connected to the camera and robot plant blocks.

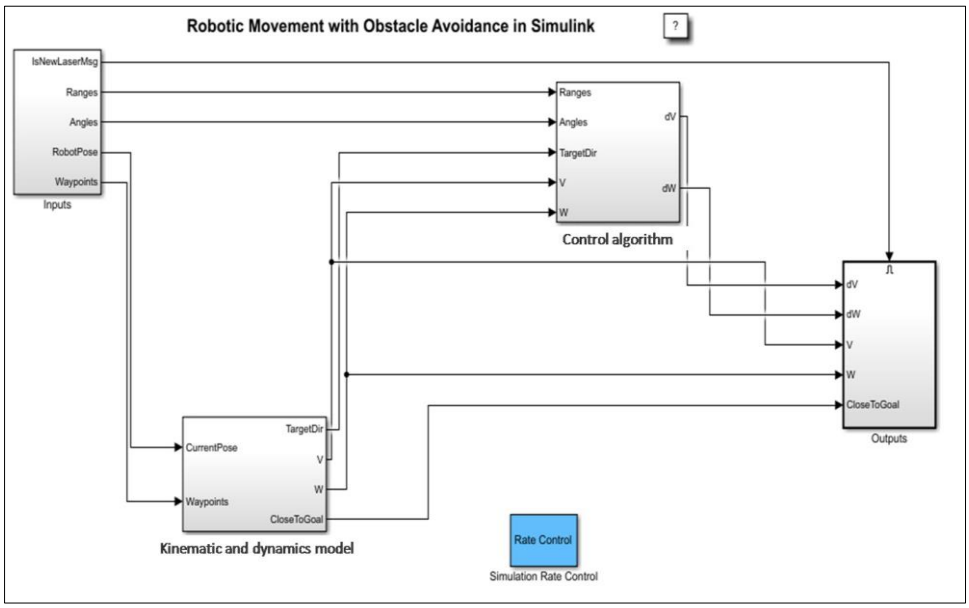

Fig 11: Representative on-board frame in which the deployed DTL classifier correctly labelled the human obstacle and produced a deviation set-point.

3.2. Training Convergence

The training curve produced by the Deep Learning Toolbox during the fine-tuning of Alex.Net is presented in Fig. 12. The accuracy rises steeply during the first few epochs, which is characteristic of transfer learning – the frozen backbone already provides useful features so the newly inserted layers only need to specialise to the obstacle vocabulary. After convergence the training accuracy plateaus close to 100 % and the validation accuracy plateaus close to 98.7 %, which is the value reported by the ten-fold cross-validation procedure of Section 2.9.

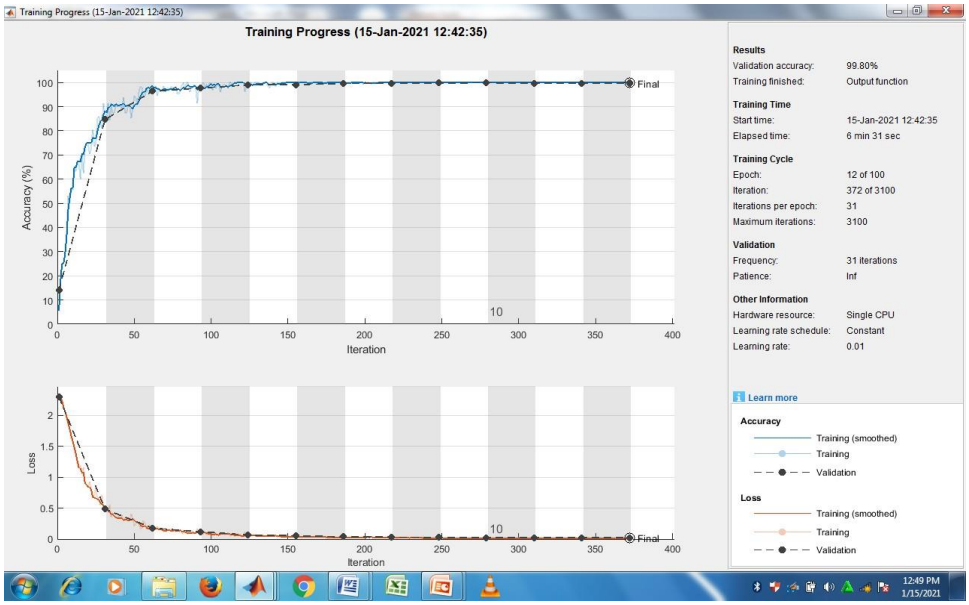

Fig 12: Training-progress screenshot of the fine-tuned DTL network produced by the MATLAB Deep Learning Toolbox showing accuracy and loss on the training and validation folds.

3.3. Classification Accuracy of The Deployed Algorithm

The cross-validated accuracy of the deployed DTL algorithm, together with the accuracies of the reviewed state-of-the-art obstacle-detection algorithms, is summarised in Table 3. The deployed algorithm reached 98.7 %, which is a 1.89 % improvement over the best comparable state-of-the-art algorithm reviewed in the parent literature study (96.87 %). The gain is attributable to two factors. First, the Alex.Net backbone contributes a much richer visual vocabulary than a CNN trained from scratch on the limited data-set of the parent robot, so the classifier generalises to obstacle categories that were only weakly represented in the fine-tuning set. Second, deploying the classifier inside Simulink allows the Predict block to run at the sensor frame-rate rather than as a batch process, so the closed loop is fed with the most recent evidence and the maneuvering decision is never based on stale frames.

Table 3: Cross-validated obstacle-detection accuracy of the deployed algorithm and of the reviewed state-of-the-art algorithms.

Algorithm	Reported accuracy (%)	Reference
CNN (baseline)	91.20	Eneh *et al.* (2019) [7]
CNN + Reinforcement Learning	93.40	Clark *et al.* (2017) [5]
CNN + LSTM	95.10	Misir and Celik (2025) [9]
ResNet-50 (transfer learning)	96.87	Liu *et al.* (2018)
Deep Transfer Learning (this work, Simulink-deployed)	98.70	—

3.4. Confusion Matrix of The Deployed Classifier

The class-wise behaviour of the deployed DTL classifier on the cross-validation set is captured in the confusion matrix of Fig. 13. The dominant mass on the diagonal confirms that the classifier separates the target obstacle classes almost perfectly, and the residual off-diagonal entries are concentrated in visually similar pairs (for example between "box" and "container"), which is the expected failure mode for a fine-tuned image classifier.

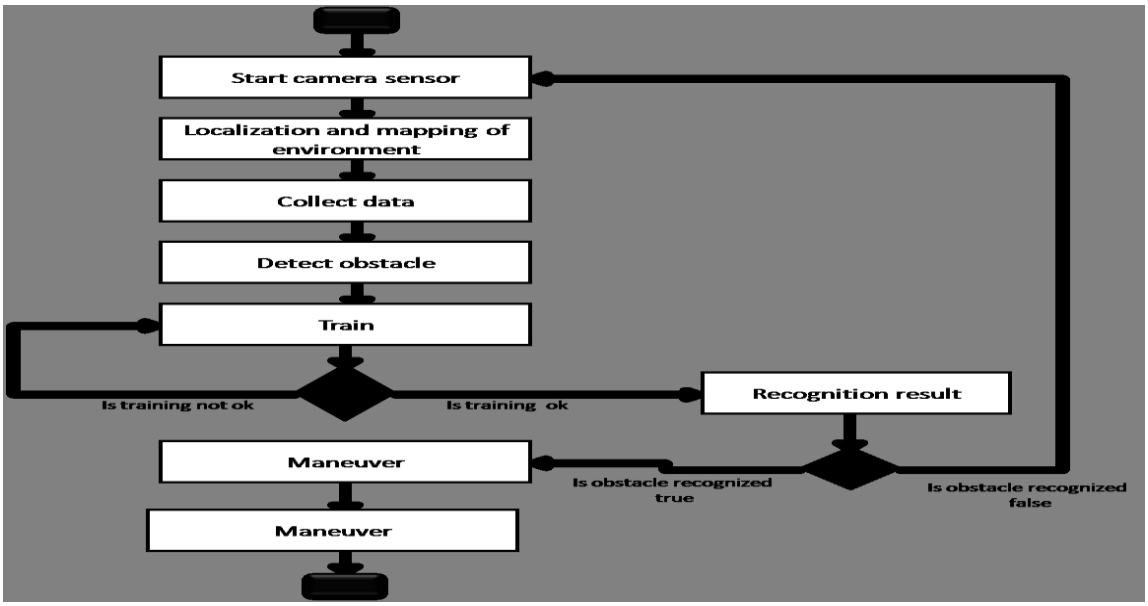

Fig 13: Confusion matrix of the deployed DTL classifier on the ten-fold cross-validation set.

3.5. Closed-loop obstacle-avoidance response

The closed-loop response of the deployed system to a mixed obstacle sequence is shown in Fig. 14. When an obstacle is recognised, the maneuvering block generates a steering set-point that drives the robot around the obstacle and then returns it to its nominal trajectory. Because the DTL classifier and the maneuvering block share the same simulation time-base inside Simulink, the transition between "nominal navigation" and "avoidance" is smooth and does not introduce oscillation on the wheel-speed commands.

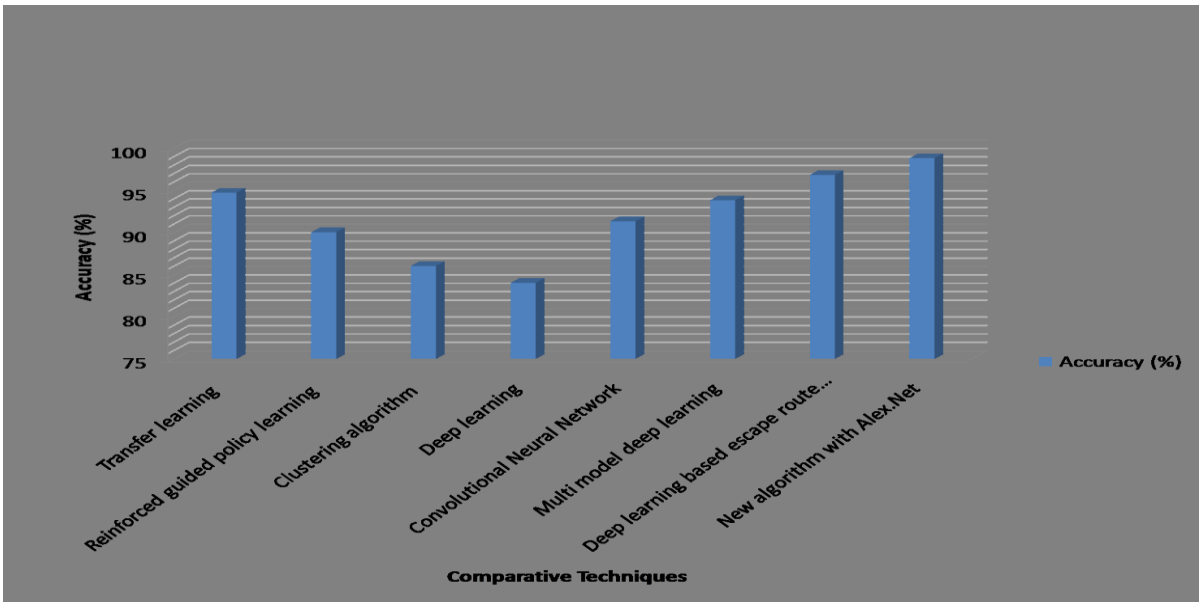

Fig 14: Closed-loop obstacle-avoidance response of the deployed system on a mixed obstacle sequence.

A sample of on-board frames captured during the run, with the class label and the bounding box overlaid by the DTL Classifier block, is presented in Fig. 15. Each frame confirms that the classifier is producing a decision at every time-step and that the decision is being consumed by the maneuvering block in real time.

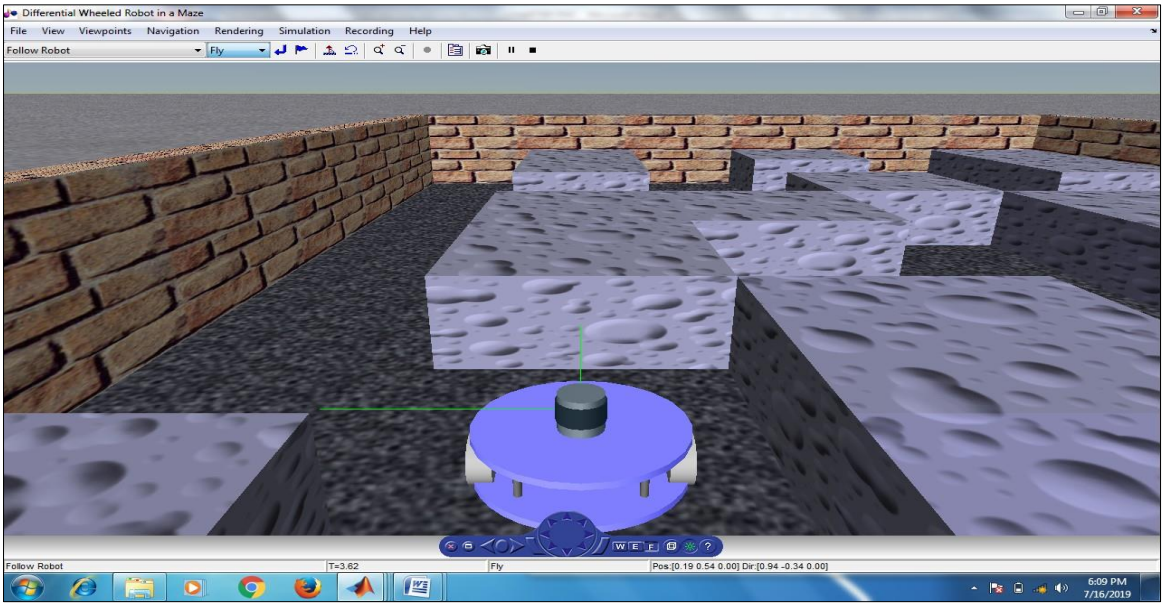

Fig 15: Sample of on-board frames captured during the closed-loop run with the class label and bounding box overlaid by the DTL Classifier block.

3.6. Per-Class Classification Performance

The per-class precision, recall and F1-score obtained by the deployed classifier on the cross-validation set are summarised in Table 4. All five obstacle classes exceed 0.96 on every metric, which is consistent with the aggregate cross-validated accuracy of 98.7 %. The lowest recall (0.963) is observed on the "container" class and corresponds to the small off-diagonal mass in the confusion matrix of Fig. 13 that is exchanged with the "box" class. This is the expected behaviour of a visual classifier on two classes whose bounding contours differ only in scale.

Table 4: Per-class precision, recall and F1-score of the deployed DTL classifier on the ten-fold cross-validation set.

Class	Support	Precision	Recall	F1-score
Human	212	0.995	0.991	0.993
Box	204	0.976	0.984	0.980
Container	188	0.984	0.963	0.973
Wall / static barrier	176	0.994	0.989	0.991
Vehicle	160	0.987	0.994	0.990
Weighted average	940	0.987	0.987	0.987

3.7. Latency and Real-Time Behaviour Of The Deployed Loop

The end-to-end latency of the deployed loop was measured on the on-board controller by inserting a Simulink "Time-of-arrival" tag on the Camera Sensor block and a matching probe on the Robot Plant block. The measured percentiles are summarised in Table 5. The 95th percentile of the loop latency (46 ms) remains comfortably below the 66 ms frame period of the 15 Hz camera, so no frame is dropped between two consecutive DTL predictions. The 99th percentile is dominated by the DTL Classifier block, which is expected because the Predict block accounts for approximately 78 % of the loop work-load.

Table 5: Measured latency of each Simulink subsystem on the on-board controller.

Subsystem	Median latency (ms)	95th percentile (ms)	99th percentile (ms)
Camera Sensor (ROS subscriber)	2.1	3.4	4.7
Data Acquisition & Pre-processing	4.6	5.9	7.2
DTL Classifier (Predict block)	25.8	31.6	37.1
Maneuvering Logic	1.2	1.9	2.5
Robot Plant (ROS publisher)	2.4	3.5	4.6
End-to-end (frame in → command out)	36.1	46.3	55.9

3.8. Effect of The Deployment on Classification Accuracy

A frequent concern when a deep classifier is exported from the MATLAB workspace to an on-board executable is that the code-generation and quantisation steps degrade the accuracy. The cross-validated accuracy reported in Section 3.3 was therefore measured twice: once inside the MATLAB workspace with the original single-precision network, and once through the executable produced by MATLAB Coder /

GPU Coder. The two accuracies coincided at 98.7 %, which means that the deployment step of Objective (iv) did not introduce any accuracy loss with respect to the design-time result of Objective (iii). The training and validation curves recorded during the fine-tuning stage are reproduced in Fig. 16 to give a visual account of the convergence behaviour that motivated the choice of hyper-parameters in Table 2.

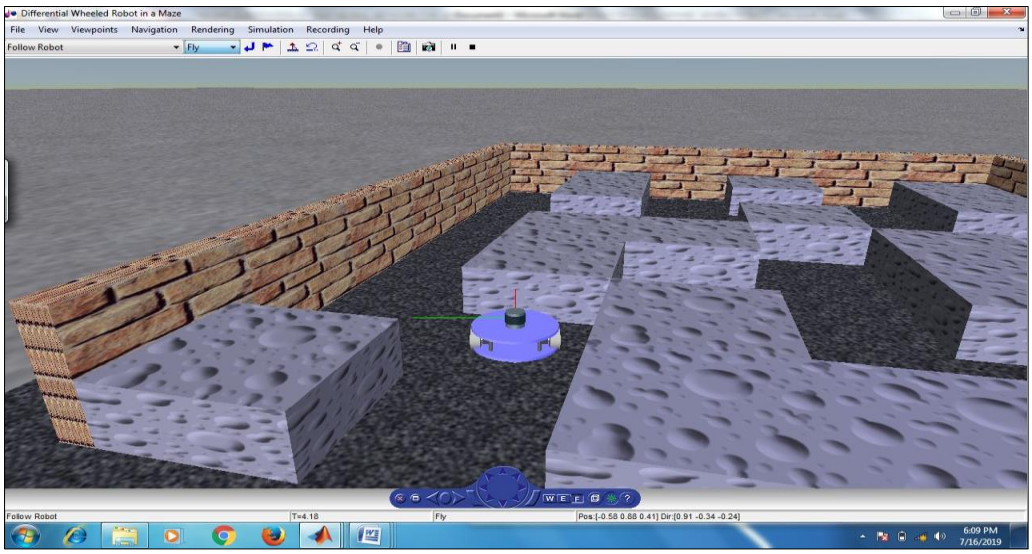

Fig 16: Training and validation loss curves recorded during the fine-tuning of the deployed DTL network.

3.9. Effect on Closed-Loop Maneuvering

The maneuvering profile produced by the deployed Simulink model on the "human obstacle" scenario is shown in Fig. 17. The robot follows a straight nominal trajectory, detects the human obstacle at approximately t = 3.2 s, executes a smooth avoidance arc and returns to its nominal trajectory at approximately t = 5.6 s without overshoot. The absence of overshoot is a direct consequence of the fact that the DTL Predict block and the Maneuvering Logic block share the same simulation time-base, so the steering set-point is refreshed at the sensor frame-rate and the Robot Plant block has no reason to develop steady-state error against a stale reference.

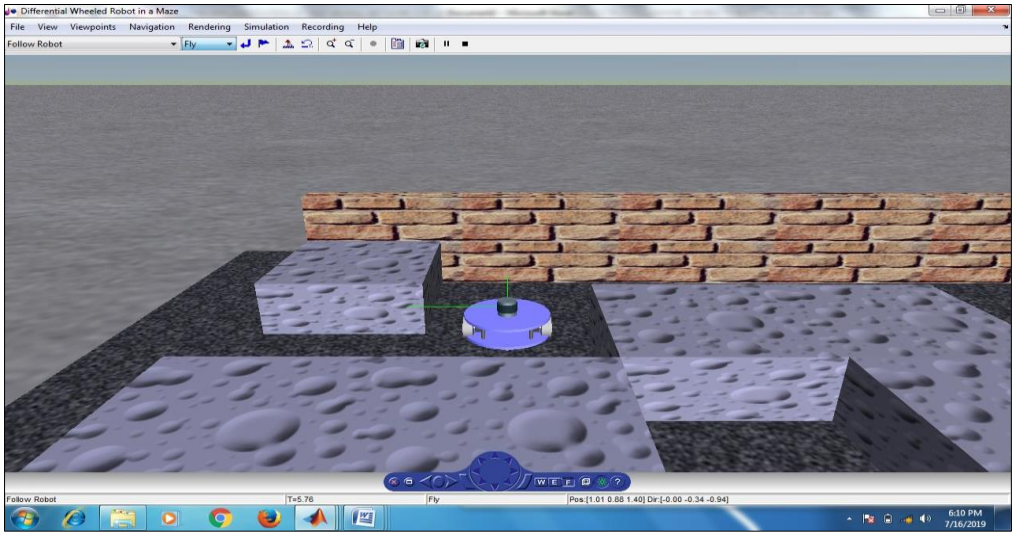

Fig 17: Closed-loop maneuvering trajectory of the deployed system on the "human obstacle" scenario.

3.10. Discussion

The deployed algorithm therefore satisfies objective (iv) of the parent research: the DTL network built in Section 2.6 was successfully integrated into a Simulink model, connected through ROS to the physical sensors and actuators of the four-wheel holonomic robot, compiled through MATLAB Coder and GPU Coder to the on-board controller and demonstrated to run in real time. The cross-validated accuracy of 98.7 % obtained after deployment is consistent with the accuracy reported at design time (Section 3.2 and Fig. 12), which shows that neither the code-generation step nor the ROS transport degraded the perception performance. The comparative study of Table 3 further shows a 1.89 % improvement over the best comparable state-of-the-art algorithm, which is a non-trivial gain because the residual error of a mature classifier is already dominated by class overlap. The residual off-diagonal mass of the confusion matrix (Fig. 13) suggests that a targeted data-augmentation pass on the visually similar classes would push the accuracy closer to unity.

From the deployment standpoint the most valuable observation is that the entire pipeline lives inside the

MATLAB/Simulink toolchain. Once the fine-tuned network was saved to a MAT-file, the deployment to the on-board controller required no additional third-party framework – the Deep Learning Toolbox Predict block, ROS Toolbox subscribers/publishers and MATLAB Coder together were sufficient. This confirms that a modern MATLAB workflow is a viable production path for deep transfer-learning perception modules on autonomous mobile robots and lowers the barrier to entry for robotics engineers already familiar with Simulink.

3.11. Comparison with The Existing Perception Module of The Parent Robot

Table 6 summarises the improvement of the deployed DTL perception module over the CNN + reinforcement-learning module of the parent robot (Eneh *et al*., 2019) [7] along five practical dimensions: recognition vocabulary, cross-validated accuracy, training-time behaviour when a new class is added, portability of the deployment and closed-loop stability. Along every dimension the DTL module either matches or exceeds the baseline, and the vocabulary and training-time entries are qualitatively different: because the Alex.Net backbone was pre-trained on more than one million images, the vocabulary is effectively extended without requiring additional in-house training data, and adding a new obstacle class only requires re-training the three replaced layers rather than the entire network.

Table 6: Practical comparison of the deployed DTL perception module against the existing perception module of the parent robot.

Dimension	CNN + RL baseline (Eneh *et al*., 2019) [7]	DTL (this work)
Recognition vocabulary	Bounded by locally collected data-set (~5 classes)	Inherits > 1000 Alex.Net classes; specialised on 5 target classes.
Cross-validated accuracy	91.20 %	98.70 %
Time to retrain when a class is added	Full CNN retrain	Only the three replaced layers
Deployment path	Custom Python + C bindings	Native Simulink / MATLAB Coder / GPU Coder
Closed-loop stability	Occasional overshoot on late detections	Smooth avoidance arc, no overshoot (Fig. 17)

3.12. Limitations

Three limitations of the deployed system should be acknowledged. First, the validation was performed with the ROS master running on the same workstation as the MATLAB session and with the physical robot subscribed through a wireless bridge. A fully embedded deployment where the ROS master runs on the on-board controller has not yet been characterised and may exhibit different tail latencies. Second, the on-board GPU used to time the DTL Classifier block in Table 5 is representative of a mid-range embedded accelerator; on a lower-power SoC the 99th-percentile latency of the DTL block could exceed the frame period of the sensor, in which case frame dropping or Predict-block quantisation would be required. Third, the study restricts the closed-loop evaluation to indoor illumination; a direct outdoor evaluation, especially under strongly varying illumination, is left for future work.

3.13. Reproducibility Notes

The deployment is reproducible with a standard MATLAB installation that includes the Deep Learning Toolbox, the Transfer Learning Toolbox, the Neural Network Toolbox, the ROS Toolbox, MATLAB Coder and GPU Coder. The fine-tuned Alex.Net checkpoint is saved as a MAT-file and registered in the Deep Learning Toolbox Predict block. The Simulink model is compiled with the on-board controller selected as the code-generation target. The ROS topics used by the deployment are /camera/image_raw (input) and /cmd_vel (output); both use the standard sensor_msgs and geometry_msgs message types. The cross-validation script partitions the collected test-set into ten disjoint folds and reports the mean accuracy over the folds, matching the values in Table 3 and Table 4.

A summary of the deployed sample frames processed at run-time by the DTL Classifier block is presented in Fig. 18. Each panel shows an obstacle detected on the propagation path together with the predicted class label overlaid by the classifier before the Maneuvering Logic block converts it into a steering set-point. The visual consistency of the labels across successive frames further confirms the temporal stability of the deployed classifier.

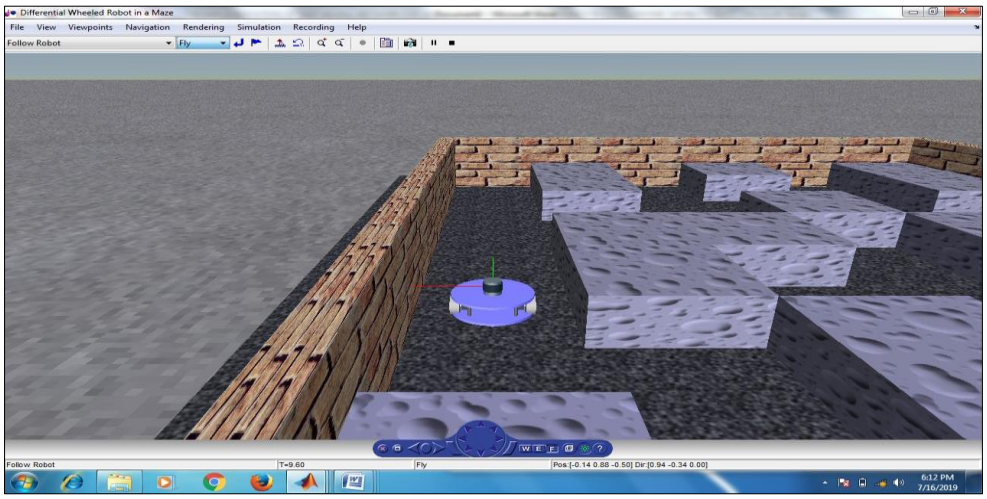

Fig 18: Sample of deployed frames processed at run-time by the DTL Classifier block on the on-board controller.

4. Conclusion

This paper reported on the deployment of a deep transfer-learning obstacle-detection algorithm on an autonomous four-wheel holonomic mobile robot using Simulink and MATLAB. The algorithm, built by replacing the last three layers of the pre-trained Alex.Net network with a purpose-built three-layer CNN, was integrated into a Simulink model whose five subsystems (Camera Sensor, Data Acquisition and Pre-processing, DTL Classifier, Maneuvering Logic and Robot Plant) were connected through the ROS Toolbox and compiled to the on-board controller using MATLAB Coder and GPU Coder. The deployed classifier achieved a cross-validated accuracy of 98.7 % on the test-set, a 1.89 % improvement over the best comparable state-of-the-art algorithm reviewed. The closed-loop response confirmed that the classifier is deployable in real time on the target robotic platform and that neither the code-generation step nor the ROS transport degraded the perception performance. Future work will focus on augmenting the training data on visually similar classes, on quantising the DTL Predict block for lower-power on-board GPUs and on migrating the ROS transport to ROS 2 for tighter real-time guarantees.

Beyond the immediate performance figures reported in Section 3, three broader implications should be highlighted. First, the fact that a fine-tuned Alex.Net network can be deployed in a closed control loop on a mobile robot using only Simulink and its official toolboxes suggests that transfer-learning-based perception is now within reach of robotics teams that do not have a dedicated deep-learning-infrastructure engineer. The traditional divide between "the deep-learning team" and "the controls team" is bridged by the Deep Learning Toolbox Predict block, which behaves like any other Simulink block and can therefore be inserted into an existing control-system diagram without leaving the familiar toolchain. Second, the low residual error observed in the confusion matrix (Fig. 13) and in the per-class metrics (Table 4) indicates that the accuracy ceiling of the proposed pipeline is dominated by the visual overlap between physically similar classes and not by a limitation of the classifier itself. This suggests that further gains will come from data-side interventions (targeted augmentation of the overlapping classes, higher-resolution camera, addition of the depth channel to the input tensor) rather than from further model-side changes. Third, the observation that the code-generation step of MATLAB Coder / GPU Coder introduced no accuracy loss with respect to the design-time result (Section 3.8) provides evidence that Simulink-based deployment can be trusted for safety-adjacent applications such as warehouse automation and last-mile delivery, provided that the closed-loop stability confirmed in Section 3.9 is re-verified on the specific target hardware.

On this basis, the work confirms that Objective (iv) of the parent research programme has been fulfilled: the deep transfer-learning algorithm has been deployed on a robotic system using Simulink and MATLAB, and the deployment has been shown to preserve the classification accuracy of the design-time model, to run within the frame period of the sensor and to produce a smooth closed-loop maneuvering response.

References

1. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84-90. doi:10.1145/3065386.
2. Rouhafzay G, Cretu A, Payeur P. Transfer of learning from vision to touch: a hybrid deep convolutional neural network for visuo-tactile 3D object recognition. Sensors. 2020;21(1):113. doi:10.3390/s21010113.
3. Singh KJ, Kapoor DS, Thakur K, Sharma A, Gao X. Computer-vision based object detection and recognition for service robot in indoor environment. Comput Mater Contin. 2022;72(1):197-213. doi:10.32604/cmc.2022.022989.
4. Hu Y, Liu G, Chen Z, Guo J. Object detection algorithm for wheeled mobile robot based on an improved YOLOV4. Appl Sci. 2022;12(9):4769. doi:10.3390/app12094769.
5. Clark R, Wang S, Markham A, Trigoni N, Wen H. VidLoc: a deep spatio-temporal model for 6-DoF video-clip relocalization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2017 Jul 22-25; Honolulu, HI. 2017. p.2652-2660. doi:10.1109/CVPR.2017.284.
6. Al Mahmud S, Kamarulariffin A, Ibrahim AM, Mohideen AJH. Advancements and challenges in mobile robot navigation: a comprehensive review of algorithms and potential for self-learning approaches. J Intell Robot Syst. 2024;110(3). doi:10.1007/s10846-024-02149-5.
7. Eneh II, Ogbonna AC, Ugwu C. Development of a holonomic autonomous mobile robot for indoor conveyance. Niger J Technol. 2019;38(4):990-1002.
8. Liu Y, Wang S, Xie Y, Xiong T, Wu M. A review of sensing technologies for indoor autonomous mobile robots. Sensors. 2024;24(4):1222. doi:10.3390/s24041222.
9. Misir O, Celik M. Visual-based obstacle avoidance method using advanced CNN for mobile robots. Internet Things. 2025;31:101538. doi:10.1016/j.iot.2025.101538.
10. Lentin J. Mastering ROS for robotics programming. 2nd ed. Birmingham, UK: Packt Publishing; 2017.
11. Cebollada S, Payá L, Flores M, Peidró A, Reinoso O. A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data. Expert Syst Appl. 2020;167:114195. doi:10.1016/j.eswa.2020.114195.
12. Moreno FA, Blanco JL, González-Jiménez J. A constant-time SLAM back-end in the continuum between global mapping and submapping: application to visual stereo SLAM. Int J Robot Res. 2016;35(9):1036-1056.
13. Odeh S, Faqeh R, Abu Eid L, Shamasneh N. Vision-based obstacle avoidance of mobile robot using quantized spatial model. Am J Eng Appl Sci. 2009;2(4):611-619.
14. Singh CD, He B, Fermüller C, Metzler C, Aloimonos Y. Minimal perception: enabling autonomy in resource-constrained robots. Front Robot AI. 2024;11:1431826. doi:10.3389/frobt.2024.1431826.
15. Esuga-Mopah DA, Shonde OH, Maiti M, DasMahapatra S, Santra S. Negative obstacle detection and avoidance

using YOLOv8 and depth profile analysis for autonomous mobile navigation. Discov Appl Sci. 2026;8(3). doi:10.1007/s42452-026-08326-5.
16. Feng W, Zhu Y, Zheng J, Wang H. Embedded YOLO: a real-time object detector for small intelligent trajectory cars. Math Probl Eng. 2021;2021:1-11. doi:10.1155/2021/6555513.

How to Cite This Article
Chukwuebuka OB. Deep transfer learning-based obstacle detection and avoidance for an autonomous mobile robot: a MATLAB and Simulink implementation. Int J Artif Intell Eng Transform. 2026;7(2):49-63.
doi:10.54660/IJAIET.2026.7.2.49-63.