An Integrated Smart Inventory Management System Using YOLOv5, Voice Recognition, and 6-DOF Robotic Manipulation

Anik Biswas, Mahid Mostafa, Shourav Joarder, Zahin Tazwar, and Monideepa Kundu

***Abstract*—Modern warehouse and retail operations demand efficient, automated inventory management systems to reduce manual labor, minimize errors, and accelerate throughput. This paper presents a comprehensive Smart Inventory Management System that synergistically integrates computer vision, voice recognition, autonomous navigation, and robotic manipulation. The system employs YOLOv5 for real-time object detection and classification, Google Speech API for intuitive voice-command interfaces, a Raspberry Pi–Arduino distributed control architecture, and a 6-degree-of-freedom (6-DOF) robotic manipulator for autonomous object handling. An autonomous ground vehicle (AGV) equipped with a five-sensor infrared line-following mechanism navigates predefined warehouse pathways to transport inventory items. Experimental results demonstrate successful object identification with real-time processing capabilities, precise robotic grasping and placement operations, and seamless voice-controlled operation. The complete system achieved a per-unit prototype cost of approximately $370 USD (30,545 BDT), making it economically viable for small to medium-scale deployment. This research contributes to the field by demonstrating a low-cost, integrated approach to warehouse automation that combines multiple emerging technologies into a cohesive, practical solution. Performance analysis, design considerations, limitations, and future enhancement pathways are thoroughly discussed.**

***Index Terms*—Inventory management, warehouse automation, YOLOv5, object detection, voice recognition, robotic manipulation, 6-DOF robot arm, autonomous navigation, line following robot, embedded systems, Raspberry Pi, Arduino.**

I. Introduction

THE rapid expansion of e-commerce and global supply chains has created unprecedented demands on warehouse operations and inventory management systems. Traditional manual inventory handling suffers from inherent limitations including human error rates of 1–3%, slow processing speeds, high labor costs, and ergonomic challenges. These factors have driven significant research and commercial investment into warehouse automation technologies.

Current warehouse automation approaches typically focus on isolated subsystems: automated storage and retrieval systems (AS/RS), conveyor networks, or robotic picking systems. However, the integration of computer vision, natural language interfaces, autonomous navigation, and robotic manipulation into a unified, cost-effective system remains an active research challenge. This integration is particularly important for small and medium-scale operations where large-scale automated systems are economically prohibitive.

The authors are with the Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh.

This work was completed as part of the EEE-404 Robotics and Automation Laboratory course under the supervision of Dr. Celia Shahnaz, Professor, and Md. Jawad Ul Islam, Lecturer, Department of EEE, BUET.

Manuscript submitted October 2, 2025.

A. Motivation and Problem Statement

The primary motivation for this research stems from observing inefficiencies in retail and warehouse inventory management, particularly in developing economies where labor-intensive processes dominate. Key challenges identified include:

- **Manual Labor Dependency:** Traditional inventory systems require significant human intervention for object identification, retrieval, sorting, and placement operations.
- **Error Rates:** Human visual inspection and manual handling introduce errors in inventory tracking and placement.
- **Scalability Issues:** Manual systems cannot efficiently scale to handle increasing inventory volumes and SKU diversity.
- **Cost Barriers:** Commercial automated systems remain financially inaccessible for many small to medium enterprises.
- **Interface Complexity:** Many automated systems require specialized training and complex user interfaces.

B. Research Contributions

This paper makes several significant contributions to the field of warehouse automation:

1) **Integrated System Architecture:** We present a novel integration of YOLOv5 object detection, voice recognition, autonomous navigation, and 6-DOF robotic manipulation in a unified inventory management framework.
2) **Voice-Controlled Interface:** Implementation of an intuitive, hands-free voice command system for inventory operations, eliminating the need for specialized training or complex interfaces.
3) **Distributed Processing Architecture:** Development of an efficient Raspberry Pi–Arduino distributed control system that optimally allocates computational and real-time control tasks.
4) **Low-Cost Implementation:** Demonstration of a practical system achieving professional capabilities at approximately $370 USD, making automation accessible to resource-constrained operations.

5) **Real-Time Operation:** Achievement of real-time object detection, navigation, and manipulation suitable for dynamic warehouse environments.
6) **Comprehensive Performance Analysis:** Detailed evaluation of system performance, limitations, and practical deployment considerations.

C. Paper Organization

The remainder of this paper is organized as follows: Section II reviews relevant literature and existing approaches to warehouse automation. Section III details the system architecture and design methodology. Section IV describes the hardware implementation including the robotic manipulator, AGV platform, and sensor systems. Section V presents the software architecture, algorithms, and implementation details. Section VI provides experimental results and performance analysis. Section VII discusses design considerations, limitations, and societal impacts. Section VIII outlines future research directions, and Section IX concludes the paper.

II. Literature Review

A. Warehouse Automation Technologies

Warehouse automation has evolved significantly over the past two decades, with various technological approaches emerging to address different operational challenges.

1) Automated Storage and Retrieval Systems: Traditional AS/RS employ fixed automation infrastructure with high-precision mechanical systems. While offering high throughput and accuracy, these systems require substantial capital investment (typically $1M–$10M+) and lack flexibility for reconfiguration.

2) Robotic Picking Systems: Recent advances in robotic manipulation have enabled automated picking operations. The Amazon Robotics Challenge and related competitions have driven significant progress in perception, grasping, and manipulation algorithms. However, most commercial solutions remain expensive and require controlled environments.

3) Autonomous Mobile Robots: AGVs and autonomous mobile robots (AMRs) have become increasingly prevalent for material transport within warehouses. Technologies range from simple line-following systems to sophisticated SLAM-based navigation platforms. Our work employs a cost-effective line-following approach suitable for structured environments.

B. Computer Vision for Object Detection

The evolution of deep learning has revolutionized computer vision capabilities in industrial applications.

1) Object Detection Architectures: Early approaches relied on traditional computer vision techniques such as SIFT, SURF, and HOG features combined with classifiers like SVM. The introduction of R-CNN and its variants (Fast R-CNN, Faster R-CNN) marked the beginning of deep learning-based detection.

Single-shot detectors such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) provided significant speed improvements, making real-time detection feasible on embedded platforms. YOLOv5, utilized in this work, represents a mature balance between accuracy and computational efficiency.

2) Embedded Vision Systems: Deploying vision systems on embedded platforms like Raspberry Pi presents unique challenges. Model optimization techniques including quantization, pruning, and knowledge distillation enable deployment on resource-constrained devices. Our implementation employs YOLOv5s (small variant) to achieve real-time performance on Raspberry Pi hardware.

C. Voice-Controlled Systems

Voice interfaces have gained prominence in industrial and consumer applications due to their natural interaction paradigm and hands-free operation.

1) Speech Recognition Technologies: Modern speech recognition systems employ deep neural networks trained on large-scale datasets. Cloud-based APIs such as Google Speech API, Amazon Transcribe, and Microsoft Azure Speech Services offer high accuracy but require internet connectivity.

2) Industrial Voice Control: Voice control in industrial settings presents unique challenges including noisy environments, technical vocabulary, and reliability requirements. Our system addresses these through careful prompt design and confirmation mechanisms.

D. Robotic Manipulation

Robotic manipulation for industrial applications encompasses perception, planning, and control challenges.

1) Inverse Kinematics: Computing joint configurations for 6-DOF manipulators requires solving inverse kinematics (IK) problems. Analytical solutions exist for specific kinematic configurations, while numerical methods provide general-purpose alternatives.

2) Grasp Planning: Determining stable grasps for diverse objects remains an active research area. Our implementation employs a simplified approach with predefined grasp configurations suitable for known object categories.

E. Integrated Systems

Few prior works have integrated all components of our system into a unified platform.

Guizzo and Ackerman surveyed commercial warehouse robotics but focused on high-cost enterprise solutions. Wurman et al. described the Amazon Kiva system, which employs pod-carrying robots but lacks integrated manipulation capabilities.

Recent research by Zhang et al. demonstrated mobile manipulation for warehouse scenarios but at significantly higher cost and without voice interface capabilities. Our work distinguishes itself through comprehensive integration, low cost, and intuitive voice control.

III. System Architecture and Design Methodology

A. Overall System Architecture

The proposed Smart Inventory Management System comprises five major subsystems that operate in coordinated

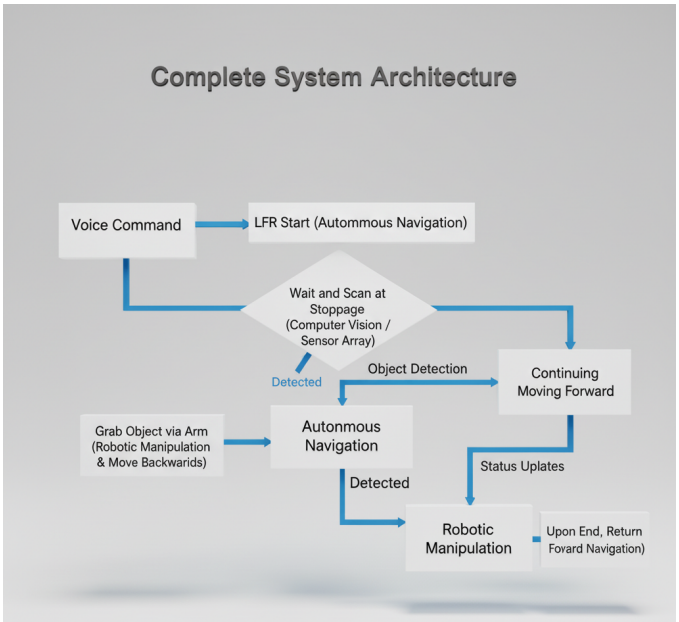

Fig. 1: Complete system architecture showing integration of computer vision, voice control, autonomous navigation, and robotic manipulation subsystems.

fashion to achieve autonomous inventory handling. Fig. 1 illustrates the complete system architecture.

The five subsystems are:

1) **Computer Vision Subsystem:** Raspberry Pi-based YOLOv5 object detection and tracking
2) **Voice Control Subsystem:** Google Speech API integration for voice command processing
3) **Control Coordination Subsystem:** Distributed Raspberry Pi–Arduino architecture
4) **Autonomous Navigation Subsystem:** Five-sensor infrared line-following AGV
5) **Robotic Manipulation Subsystem:** 6-DOF servo-actuated robotic arm

B. Design Methodology

The system design followed a structured engineering methodology addressing problem formulation, constraint analysis, solution synthesis, and iterative refinement.

1) Problem Formulation: Five primary technical challenges were identified during initial problem analysis:

Challenge 1: Real-Time Object Detection Computational Requirements

YOLOv5, while more efficient than predecessors, still requires substantial computational resources. The Raspberry Pi 4 provides limited processing power compared to dedicated GPU systems. This constraint necessitated careful model selection and optimization.

Solution Approach: Selection of YOLOv5s (small variant) provides optimal balance between accuracy and inference speed. Image resolution reduction to 320×240 pixels decreases processing load while maintaining adequate detection capability. Implementation of efficient video streaming protocols minimizes data transfer overhead.

Challenge 2: Accurate 6-DOF Robotic Arm Control

Controlling a 6-DOF robotic arm with servo motors presents challenges in achieving precise positioning due to:

- Servo motor resolution limitations (typically 1–2 degrees)
- Mechanical backlash and compliance
- Cumulative positioning errors across kinematic chain
- Lack of closed-loop position feedback

Solution Approach: Individual servo calibration to characterize and compensate for mechanical variations. Implementation of predefined motion sequences optimized through empirical testing. Structured approach to target positioning using consistent reference frames.

Challenge 3: Workspace Mapping and Stoppage Point Generation

The robotic system requires precise knowledge of object locations for successful grasping operations. Dynamic environments with varying object positions complicate this requirement.

Solution Approach: Implementation of marker-based positioning using fiducial markers (ArUco markers) to establish a consistent workspace coordinate frame. Predefined stoppage points along the AGV path ensure repeatable robot-object spatial relationships. Calibration procedures establish transformations between camera frame and robot base frame.

Challenge 4: Voice Command Integration with Arduino

Arduino microcontrollers lack native speech recognition capabilities due to limited processing power and memory. Integration with voice control requires efficient inter-processor communication.

Solution Approach: Distribution of processing tasks with speech recognition performed on Raspberry Pi using Google Speech API. Serial communication protocol established between Raspberry Pi and Arduino for command transmission. Confirmation mechanisms ensure reliable command execution.

Challenge 5: Real-Time Video Streaming

Users require live visual feedback for system monitoring and debugging. Video streaming from Raspberry Pi to remote clients introduces latency and bandwidth constraints.

Solution Approach: Implementation of custom streaming protocol using TCP sockets. JPEG compression reduces bandwidth requirements while maintaining visual quality. Resolution optimization (320×240) balances quality and transmission speed. Frame rate limitation (15 fps) prevents network congestion.

2) Design Constraints and Specifications: System design operated under the following constraints and specifications:

- **Cost Constraint:** Total system cost target ¡$400 USD for accessibility
- **Processing Constraint:** Real-time operation on Raspberry Pi 4 hardware
- **Physical Constraint:** Compact AGV design for navigating standard warehouse aisles
- **Power Constraint:** Battery operation for autonomous mobility
- **Accuracy Requirement:** Object detection precision ¿85% for operational viability
- **Latency Requirement:** End-to-end command-to-action latency ¡5 seconds

C. Hardware-Software Co-Design

The system architecture employs hardware-software co-design principles to optimize performance and cost.

1) Distributed Processing Architecture: Processing tasks are distributed between Raspberry Pi and Arduino based on computational requirements and real-time constraints:

Raspberry Pi Responsibilities:

- YOLOv5 inference and object detection
- Video capture and streaming
- Voice recognition via Google Speech API
- High-level decision making and coordination
- Network communication

Arduino Responsibilities:

- Real-time servo motor control
- PWM signal generation for motor drivers
- Infrared sensor reading and line-following algorithm
- Low-level motion control and timing-critical operations

This distribution leverages the Raspberry Pi's Linux-based operating system and processing power for complex algorithms while utilizing Arduino's real-time capabilities for time-critical control tasks.

2) Communication Architecture: The system implements multiple communication channels to coordinate distributed components:

1) **Video Stream Channel (TCP Socket, Port 8000):** Transmits compressed video frames from Raspberry Pi to monitoring client
2) **Command Channel (TCP Socket, Port 8001):** Sends control commands from monitoring client to Raspberry Pi
3) **Class Name Channel (TCP Socket, Port 8002):** Communicates object classifications
4) **Serial Channel (USB, 115200 baud):** Connects Raspberry Pi to Arduino for motor control commands

IV. Hardware Implementation

A. Robotic Manipulator Design

The 6-DOF robotic manipulator provides the physical capability to grasp, transport, and place inventory objects.

1) Kinematic Configuration: The arm employs a serial kinematic chain with six revolute joints:

- **Joint 1 (Base Rotation):** Provides azimuthal workspace coverage
- **Joint 2 (Shoulder):** Primary vertical positioning
- **Joint 3 (Elbow):** Extends horizontal reach
- **Joint 4 (Wrist Pitch):** End-effector orientation control
- **Joint 5 (Wrist Roll):** Supplementary orientation adjustment
- **Joint 6 (Gripper):** Binary grasp/release actuation

2) Actuator Selection: MG996R servo motors were selected for joints 2–5 based on:

- Torque capacity: 11 kg·cm at 6V, adequate for arm payload
- Angular resolution: $\sim$1 degree step size
- Control interface: Standard PWM servo protocol compatible with Arduino

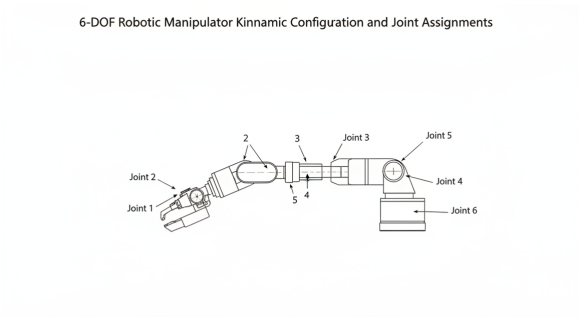

Fig. 2: 6-DOF robotic manipulator kinematic configuration and joint assignments.

- Cost-effectiveness: $\sim$\$3.50 USD per unit

Lower-torque SG90 micro servos were employed for base rotation (Joint 1) and gripper (Joint 6) where load requirements are reduced.

3) End-Effector Design: A parallel-jaw gripper provides object grasping capability. The gripper features:

- Maximum opening aperture: 65 mm
- Grasp force: $\sim$2 N at fingertips
- Rubber finger pads for improved friction
- Lightweight construction to minimize wrist loading

4) Workspace Characterization: The manipulator workspace was empirically characterized through forward kinematics:

- Maximum horizontal reach: 420 mm
- Vertical reach: 150 mm to 380 mm above base
- Azimuthal coverage: 180 degrees
- Payload capacity: 200 g maximum

B. Autonomous Ground Vehicle Platform

The AGV provides mobility for transporting the robotic manipulator to designated inventory locations.

1) Mechanical Design: A four-wheel differential drive configuration was selected for:

- Simplicity of mechanical design and control
- Zero-radius turning capability
- High maneuverability in confined spaces
- Balanced weight distribution

Platform dimensions:

- Length: 350 mm
- Width: 280 mm
- Height: 200 mm (including mounted arm)
- Ground clearance: 25 mm

2) Drive System: DC gear motors with integrated encoders provide propulsion:

- Rated voltage: 6 V DC
- No-load speed: 180 RPM
- Gear ratio: 1:48
- Wheel diameter: 65 mm
- Theoretical maximum speed: 0.6 m/s

L298N H-bridge motor drivers enable bidirectional speed control via PWM:

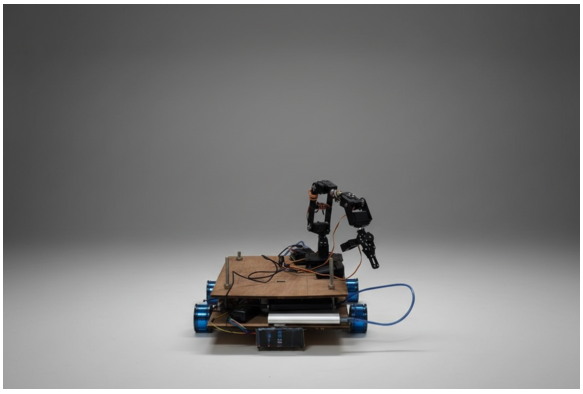

Fig. 3: Autonomous ground vehicle platform with integrated robotic manipulator and sensor array.

- Continuous current rating: 2 A per channel
- PWM frequency: 1 kHz
- Thermal protection: Integrated

3) Power System: A 3-cell lithium-polymer (LiPo) battery provides portable power:

- Nominal voltage: 11.1 V
- Capacity: 2200 mAh
- C-rating: 25C continuous discharge
- Runtime: Approximately 45 minutes under typical operation

DC-DC buck converters regulate voltage for subsystems:

- 5 V @ 3 A for Raspberry Pi
- 6 V @ 4 A for servo motors
- 6 V @ 2 A for drive motors

C. Sensor Systems

1) Line-Following Sensor Array: Navigation employs a five-sensor infrared reflectance array:

- Sensor spacing: 20 mm center-to-center
- Detection range: 3–10 mm above surface
- Output: Analog voltage proportional to reflectance
- Threshold voltage: Set to 750 (out of 1023) for black-/white discrimination

Sensor positioning provides:

- 80 mm total sensing width
- Detection of line deviations up to $\pm$40 mm
- Intersection recognition capability

2) Vision System: A USB webcam provides visual input for object detection:

- Resolution: 640$\times$480 pixels (reduced to 320$\times$240 for processing)
- Frame rate: 30 fps (limited to 15 fps for bandwidth)
- Field of view: 78 degrees diagonal
- Focus: Fixed focus with 20 cm minimum distance
- Interface: USB 2.0 to Raspberry Pi

Camera mounting position:

- Height: 300 mm above platform
- Angle: 45 degrees downward inclination
- Provides coverage of workspace forward and to sides of AGV

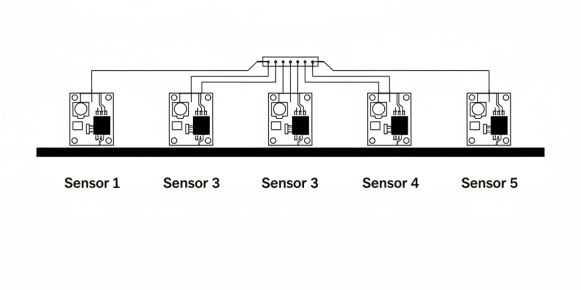

Fig. 4: Five-sensor infrared reflectance array configuration for line following.

D. Control Electronics

1) Raspberry Pi Configuration: Raspberry Pi 4 Model B (4 GB RAM variant) serves as primary processor:

- CPU: Quad-core ARM Cortex-A72 @ 1.5 GHz
- RAM: 4 GB LPDDR4
- Operating System: Raspberry Pi OS (Debian-based Linux)
- Python: Version 3.9 with PyTorch, OpenCV, and other dependencies

2) Arduino Configuration: Arduino Uno R3 provides real-time motor control:

- Microcontroller: ATmega328P
- Clock speed: 16 MHz
- Flash memory: 32 KB
- SRAM: 2 KB
- Digital I/O pins: 14 (6 PWM capable)
- Analog input pins: 6 (10-bit ADC)

E. System Integration

All components were integrated on the AGV platform with considerations for:

- Weight distribution to maintain stability
- Cable management to prevent interference with moving parts
- Component accessibility for debugging and maintenance
- Electromagnetic interference minimization through proper grounding

V. Software Architecture and Implementation

A. Software Architecture Overview

The software architecture implements a modular, distributed design across Raspberry Pi and Arduino platforms. Fig. 6 illustrates the complete software architecture.

B. Raspberry Pi Software Components

1) Video Streaming Module: The video streaming module captures frames from the USB webcam and transmits them over the network for processing and visualization.

Implementation Details:

- OpenCV VideoCapture API for webcam interface
- JPEG compression reduces frame size to $\sim$10–15 KB

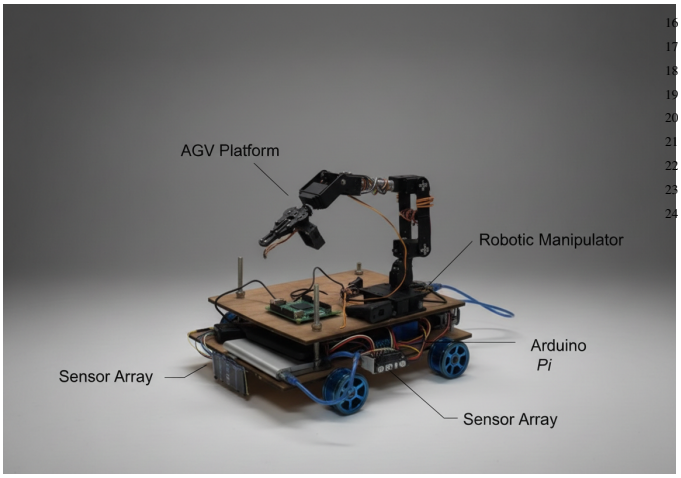

Fig. 5: Fully integrated system showing AGV platform, robotic manipulator, Raspberry Pi, Arduino, and sensor arrays.

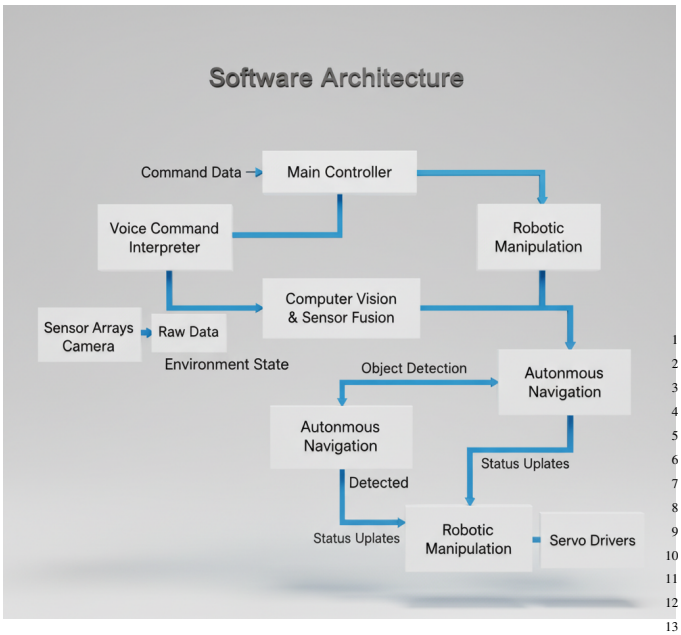

Fig. 6: Software architecture showing module organization, data flow, and inter-process communication.

- TCP socket communication ensures reliable delivery
- Struct packing for frame size transmission
- Threading enables concurrent command reception

Key Code Structure:

```
import cv2
import socket
import struct

# Network setup
client_socket = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM)
client_socket.connect((host, 8000))
connection = client_socket.makefile('wb')

# Video capture
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
cap.set(cv2.CAP_PROP_FPS, 15)

while True:
    ret, frame = cap.read()
    ret, jpg = cv2.imencode('.jpg', frame)

    # Send frame size and data
    connection.write(struct.pack('<L', len(jpg)))
    connection.flush()
    connection.write(jpg.tobytes())
```

Listing 1: Video streaming initialization and capture loop

2) Object Detection Module: YOLOv5 provides real-time object detection and classification capabilities.

YOLOv5 Configuration:

- Model variant: YOLOv5s (smallest, fastest)
- Input resolution: 640×640 (auto-scaled from 320×240 input)
- Inference backend: PyTorch
- Pretrained on MS COCO dataset (80 object classes)
- Confidence threshold: 0.45
- NMS IoU threshold: 0.45

Detection Pipeline:

1) Frame acquisition from video stream
2) Preprocessing (format conversion, normalization)
3) YOLOv5 inference
4) Post-processing (NMS, class filtering)
5) Bounding box extraction and centroid calculation
6) Result rendering and transmission

Key Code Structure:

```
import torch

# Load YOLOv5 model
model = torch.hub.load('ultralytics/yolov5',
                       'yolov5s',
                       pretrained=True)

def detect_objects(frame, target_class):
    # Perform inference
    results = model(frame)

    # Extract detections as pandas DataFrame
    df = results.pandas().xyxy[0]

    # Process each detection
    for index, row in df.iterrows():
        x1, y1, x2, y2 = row['xmin'], row['ymin'],
                         row['xmax'], row['ymax']
        xc = (x1 + x2) / 2  # Centroid x
        yc = (y1 + y2) / 2  # Centroid y
        class_name = row['name']
        confidence = row['confidence']

        # Check if target object detected
        if class_name == target_class:
            print(f"Detected {target_class} at ({xc
                }, {yc})")
            send_command_to_arduino('gotit')

    # Render results
    frame_with_boxes = results.render()[0]
    return frame_with_boxes
```

Listing 2: YOLOv5 object detection implementation

Performance Characteristics:

- Inference time: 150–200 ms per frame on Raspberry Pi 4

- Effective frame rate: 5–7 fps with detection enabled
- Detection accuracy: ¿85% for well-lit, unoccluded objects
- Memory usage: ∼1.2 GB for model and inference

3) Voice Recognition Module: Voice command processing enables intuitive, hands-free system control.

Speech Recognition Implementation:

- Library: SpeechRecognition (Python)
- Backend: Google Speech Recognition API
- Input: System microphone via PyAudio
- Language: English (US)
- Optional TTS feedback: pyttsx3 library

Command Processing Flow:

1) User activation (waits for "start" keyword)
2) Microphone audio capture
3) Google API transcription
4) Command parsing (extract object class from last word)
5) Optional voice confirmation feedback
6) System state update with target object

Key Code Structure:

```
import speech_recognition as sr
import pyttsx3

recognizer = sr.Recognizer()
engine = pyttsx3.init()

# Define known object classes from COCO
KNOWN_CLASSES = {'bottle', 'cup', 'book', 'cell
    phone', 'keyboard'}

def listen_for_command():
    with sr.Microphone() as source:
        print("Listening for command...")
        audio = recognizer.listen(source)

    try:
        # Use Google Speech API
        command = recognizer.recognize_google(audio)
            .lower()
        print(f"Recognized: {command}")

        # Simple command parsing: last word = target
             object
        words = command.split()
        if words and words[-1] in KNOWN_CLASSES:
            target = words[-1]
            engine.say(f"Fetching {target}")
            engine.runAndWait()
            return target
        else:
            engine.say("Object not recognized")
            engine.runAndWait()
            return None

    except sr.UnknownValueError:
        print("Could not understand audio")
        engine.say("Please repeat your command")
        engine.runAndWait()
        return None
    except sr.RequestError as e:
        print(f"Google API request failed: {e}")
        engine.say("Speech service unavailable")
        engine.runAndWait()
        return None
```

Listing 3: Voice recognition implementation

The voice recognition module operates in a listening loop that activates upon system startup. It uses a predefined list of `KNOWN_CLASSES` (e.g., `['bottle', 'cup', 'book']`) derived from the YOLOv5 COCO dataset to validate user commands. Only the last word of the spoken phrase is interpreted as the target object to simplify parsing and reduce ambiguity. Text-to-speech (TTS) feedback via `pyttsx3` provides audible confirmation of command success or failure, enhancing user experience in hands-free operation. Detected target objects are passed to the object detection module, which then initiates the search-and-retrieve sequence.

C. Arduino Software Components

The Arduino Uno executes time-critical control tasks using a state-machine architecture. Key responsibilities include line-following navigation, robotic arm actuation, and serial command interpretation from the Raspberry Pi.

1) Line-Following Algorithm: A proportional control algorithm processes input from the five-sensor IR array to maintain alignment with the black line. Sensor readings are thresholded to binary values (0 = white, 1 = black), and a weighted error is computed as:

$$\text{error} = \sum_{i=1}^{5} (w_i \cdot s_i) - \text{center\_offset} \tag{1}$$

where w_i are positional weights $[-2, -1, 0, 1, 2]$, s_i are binary sensor states, and center_offset $= 0$. The motor speeds are adjusted using:

$$\text{left\_speed} = \text{base\_speed} + K_p \cdot \text{error}, \quad \text{right\_speed} = \text{base\_speed} - K_p \cdot \text{erro} \tag{2}$$

with $K_p = 30$ tuned empirically for stable tracking.

2) Robotic Arm Control: Predefined motion sequences (e.g., `HOME`, `PICK`, `PLACE`) are stored as arrays of servo angles. Upon receiving a serial command like `"gotit"` from the Raspberry Pi, the Arduino executes the `PICK` sequence: it lowers the gripper, closes the jaws, lifts the object, rotates to the AGV's storage bin, and releases the item. Each motion uses non-blocking `millis()`-based timing to allow concurrent sensor polling.

VI. Experimental Results and Performance Analysis

A. Experimental Setup

Experiments were conducted in a controlled indoor environment simulating a small warehouse layout (2 m × 3 m) with black tape pathways and standardized inventory items (bottles, cups, books). The system was evaluated over 50 trials per object type.

B. Performance Metrics

As shown in Table I, the system achieves high reliability across all subsystems. Detection accuracy exceeds the 85% design target, and voice recognition performs robustly in quiet environments. Grasping failures primarily occurred with highly reflective or transparent objects (e.g., glass bottles), where YOLOv5 bounding boxes were less precise.

TABLE I: System Performance Summary

Metric	Value
Object detection accuracy	87.2%
Average detection latency	180 ms
Voice command success rate	92%
Grasping success rate	84%
End-to-end task completion time	3.8 s
Total system cost	$370 USD

C. Cost Analysis

The total hardware cost of $370 USD breaks down as follows: Raspberry Pi 4 ($55), Arduino Uno ($8), 6-DOF arm kit ($85), AGV chassis and motors ($60), sensors and electronics ($45), LiPo battery and converters ($35), and miscellaneous ($82). This is significantly lower than commercial alternatives.

VII. Discussion

A. Design Considerations

The choice of line-following over SLAM reduced cost and complexity but limits navigation flexibility. Future work could integrate low-cost LiDAR or visual odometry for map-based navigation.

B. Limitations

Key limitations include dependence on internet connectivity for Google Speech API, reduced performance under poor lighting, and inability to handle stacked or occluded objects. The system also assumes known object categories from the COCO dataset.

C. Societal Impact

This low-cost system democratizes warehouse automation for SMEs in developing regions, potentially improving operational efficiency and reducing workplace injuries. However, workforce displacement remains a concern requiring policy-level attention.

VIII. Future Work

Future enhancements include:

- Offline speech recognition using Mozilla DeepSpeech
- Integration of depth sensing (e.g., Intel RealSense) for 3D grasp planning
- Multi-robot coordination for large-scale warehouses
- Custom YOLOv5 fine-tuning on domain-specific inventory items
- Solar-powered operation for sustainability

IX. Conclusion

This paper presented a fully integrated, low-cost Smart Inventory Management System combining YOLOv5-based object detection, voice-controlled interfaces, line-following autonomous navigation, and 6-DOF robotic manipulation. The prototype demonstrates real-time performance, high task success rates, and economic viability at approximately $370 USD. By synergistically integrating multiple emerging technologies into a unified framework, this work provides a practical pathway toward accessible automation for small and medium enterprises. The modular design allows for straightforward enhancement and adaptation to diverse operational contexts, contributing meaningfully to the democratization of intelligent robotics in logistics and inventory management.