Table of Contents

# 1 High-Level Design (HLD) - Vision-Based Pick and Place System

## 1.1 Document Information

* **Project:** Vision-Based Pick and Place Robotic System
* **Version:** 1.0
* **Date:** 2025-10-18
* **Status:** Draft
* **Authors:** System Architecture Team

## 1.2 Table of Contents

1. [Introduction](#Xe3d0fc0bea9a42ce7605565d0964033d7f6ee47)
2. [System Context](#Xe5c860d979428bcf2624da057da1929d80871fa)
3. [Architectural Principles](#X2d4c67915ea765e5ebcfa398fd064889bdfcdc8)
4. [System Architecture Overview](#X5225e65baddbf51b8bbe314745a3788ec51ba65)
5. [Subsystem Design](#Xf582c70afb959b64f1e2a318f5634fc6c19c30f)
6. [Data Architecture](#X9e55b3d3cefb61f21823ded8c92669578c6fb2b)
7. [Interface Design](#X65d82ab3bda015097769ee9dfc0978926e97b3f)
8. [Deployment Architecture](#X6dc515e58e002e3afd4d65f6646e7e576dc6b74)
9. [Security Architecture](#X5e8e6d18f1f6945ae9f8a2d48e614b6f2995483)
10. [Scalability & Performance](#Xdad7deb0a1a6f2c3700d79f5d1018b166961932)
11. [Appendices](#X261d6f9e0e4ee8186dd98e2029c7ef8cfa788fa)

## 1.3 1. Introduction

### 1.3.1 1.1 Purpose

This High-Level Design (HLD) document describes the system architecture for a vision-based pick-and-place robotic system. It provides: - Architectural overview and design principles - Subsystem decomposition and responsibilities - Interface specifications between subsystems - Data flow and storage architecture - Deployment and infrastructure design

### 1.3.2 1.2 Scope

**In Scope:** - System architecture (software, hardware, network) - Subsystem decomposition - Interface definitions (APIs, messages, protocols) - Data architecture (databases, caching, logging) - Deployment models (single-machine, distributed)

**Out of Scope:** - Detailed class diagrams (see Low-Level Design) - Algorithm implementation details - Hardware schematics (see Electrical Design docs) - Test plans (see Testing & Validation Plan)

### 1.3.3 1.3 Definitions & Acronyms

| Term | Definition |
| --- | --- |
| HLD | High-Level Design |
| ROS2 | Robot Operating System 2 |
| DDS | Data Distribution Service (ROS2 middleware) |
| IK | Inverse Kinematics |
| FK | Forward Kinematics |
| F/T | Force/Torque |
| TF | Transform (coordinate frame transformations) |
| URDF | Unified Robot Description Format |

## 1.4 2. System Context

### 1.4.1 2.1 System Boundary

┌────────────────────────────────────────────────────────────────┐  
│ EXTERNAL ACTORS │  
├────────────────────────────────────────────────────────────────┤  
│ • Operator (HMI interaction) │  
│ • Integrator (configuration, calibration) │  
│ • Engineer (development, debugging) │  
│ • Manager (monitoring, reporting) │  
│ • Maintenance Tech (diagnostics, repair) │  
└────────────────────────┬───────────────────────────────────────┘  
 │  
 ▼  
┌────────────────────────────────────────────────────────────────┐  
│ VISION-BASED PICK-PLACE SYSTEM │  
│ ┌──────────────────────────────────────────────────────────┐ │  
│ │ Vision Motion Grasp Task │ │  
│ │ Perception Planning Planning Orchestration │ │  
│ └──────────────────────────────────────────────────────────┘ │  
│ ┌──────────────────────────────────────────────────────────┐ │  
│ │ Control & Monitoring Security Data │ │  
│ │ Execution & Logging & Auth Management │ │  
│ └──────────────────────────────────────────────────────────┘ │  
└────────────────────────┬───────────────────────────────────────┘  
 │  
 ▼  
┌────────────────────────────────────────────────────────────────┐  
│ EXTERNAL SYSTEMS │  
├────────────────────────────────────────────────────────────────┤  
│ • Warehouse Management System (WMS) │  
│ • Manufacturing Execution System (MES) │  
│ • Cloud Analytics Platform (AWS/Azure) │  
│ • Time-Series Database (InfluxDB Cloud) │  
└────────────────────────────────────────────────────────────────┘

### 1.4.2 2.2 System Interfaces

| **Interface** | **Type** | **Protocol** | **Description** |
| --- | --- | --- | --- |
| HMI (Web) | Input | HTTP/WebSocket | Operator control panel |
| Robot API | Bidirectional | EtherCAT/Modbus | Motor control, status |
| Camera | Input | USB 3.0 | RGB-D image stream |
| F/T Sensor | Input | Ethernet/UDP | Force-torque data |
| WMS/MES | Bidirectional | REST API | Task orders, completion reports |
| Cloud Analytics | Output | HTTPS/MQTT | Telemetry, logs |

### 1.4.3 2.3 Stakeholders

| **Stakeholder** | **Interest** | **Key Concerns** |
| --- | --- | --- |
| End User (Operator) | Ease of use, reliability | Simple UI, fast error recovery |
| System Integrator | Easy deployment, configuration | Documentation, calibration tools |
| Software Developer | Code quality, modularity | Clean architecture, testability |
| Project Manager | On-time delivery, ROI | Progress tracking, risk management |
| Safety Officer | Compliance, worker safety | ISO 10218, E-stop, audit trails |

## 1.5 3. Architectural Principles

### 1.5.1 3.1 Design Principles

1. **Modularity:**
   * Each subsystem is independently deployable and testable
   * Clear interfaces between modules (ROS2 topics, services, actions)
   * Loose coupling, high cohesion
2. **Scalability:**
   * Horizontal scaling: Add more robots without redesigning system
   * Vertical scaling: Upgrade compute (e.g., Jetson Xavier → Orin)
3. **Real-Time Performance:**
   * Control loops run at deterministic frequencies (1 kHz for motion control)
   * Vision pipeline optimized for low latency (<50ms)
4. **Fault Tolerance:**
   * Graceful degradation (e.g., if F/T sensor fails, continue with vision-only)
   * Automatic retry mechanisms for transient errors
   * Comprehensive error logging
5. **Security:**
   * Authentication required for configuration changes
   * Encrypted communication for sensitive data (TLS)
   * Audit trail for all critical operations
6. **Maintainability:**
   * Self-documenting code (docstrings, type hints)
   * Automated testing (unit, integration, system)
   * Version control for all artifacts (code, configs, models)
7. **Usability:**
   * Intuitive UIs (web dashboard, RViz)
   * Guided wizards for complex tasks (calibration)
   * Clear error messages with actionable guidance

### 1.5.2 3.2 Architectural Patterns

| **Pattern** | **Application** | **Benefit** |
| --- | --- | --- |
| **Layered Architecture** | Separate concerns (perception, planning, control) | Clear separation, testability |
| **Event-Driven (Pub-Sub)** | ROS2 topics for sensor data | Decoupling, flexibility |
| **Microservices** | Independent ROS2 nodes | Scalability, fault isolation |
| **State Machine** | Task orchestration | Clear control flow, error handling |
| **Repository Pattern** | Data access (PostgreSQL, Redis) | Abstraction, testability |
| **Adapter Pattern** | Hardware abstraction (ros2\_control) | Portability across robots |

## 1.6 4. System Architecture Overview

### 1.6.1 4.1 Logical Architecture (Layered View)

┌──────────────────────────────────────────────────────────────┐  
│ LAYER 6: PRESENTATION │  
│ Web UI, RViz2, Grafana, Foxglove, Mobile App │  
└──────────────────────────────────────────────────────────────┘  
 ▲ (HTTPS, WebSocket)  
┌──────────────────────────────────────────────────────────────┐  
│ LAYER 5: APPLICATION / BUSINESS LOGIC │  
│ Task Orchestrator (FSM/BT), Workflow Manager, Analytics │  
└──────────────────────────────────────────────────────────────┘  
 ▲ (ROS2 Services/Actions)  
┌──────────────────────────────────────────────────────────────┐  
│ LAYER 4: DOMAIN / CORE LOGIC │  
│ Vision Pipeline, Grasp Planner, Motion Planner (MoveIt2) │  
└──────────────────────────────────────────────────────────────┘  
 ▲ (ROS2 Topics/Services)  
┌──────────────────────────────────────────────────────────────┐  
│ LAYER 3: MIDDLEWARE (ROS2) │  
│ DDS (CycloneDDS), TF2, Image Transport, ros2\_control │  
└──────────────────────────────────────────────────────────────┘  
 ▲ (ROS2 APIs)  
┌──────────────────────────────────────────────────────────────┐  
│ LAYER 2: DEVICE ABSTRACTION │  
│ Camera SDK, Motor Drivers, Sensor Drivers, GPIO │  
└──────────────────────────────────────────────────────────────┘  
 ▲ (USB, EtherCAT, Ethernet)  
┌──────────────────────────────────────────────────────────────┐  
│ LAYER 1: HARDWARE │  
│ Robot, Camera, F/T Sensor, Compute (Jetson, NUC), Network │  
└──────────────────────────────────────────────────────────────┘

### 1.6.2 4.2 Subsystem Decomposition

| **Subsystem** | **Responsibilities** | **Key Components** |
| --- | --- | --- |
| **Vision Perception** | Detect objects, estimate poses, generate point clouds | YOLO detector, Pose estimator, PCL processor |
| **Grasp Planning** | Compute optimal gripper poses | Grasp sampler, Collision checker, Quality scorer |
| **Motion Planning** | Plan collision-free trajectories | MoveIt2, OMPL, IK solver |
| **Control & Execution** | Execute trajectories with feedback | ros2\_control, PID controllers, Trajectory interpolator |
| **Task Orchestration** | High-level task sequencing, error handling | State machine (BT.CPP), Event dispatcher |
| **Monitoring & Logging** | Telemetry, logging, alerting | Prometheus, Grafana, ELK stack, rosbag2 |
| **Security & Auth** | User authentication, access control | OAuth2, JWT, Firewall |
| **Data Management** | Persistent storage, caching | PostgreSQL, Redis, InfluxDB |
| **Configuration** | System configuration, parameter management | YAML files, ROS2 parameter server |

## 1.7 5. Subsystem Design

### 1.7.1 5.1 Vision Perception Subsystem

**Purpose:** Acquire images, detect objects, estimate 6DoF poses, generate point clouds

**Architecture:**

┌─────────────────┐  
│ RealSense D435 │ (Hardware)  
└────────┬────────┘  
 │ USB 3.0 (RGB-D stream)  
 ▼  
┌──────────────────────────────────┐  
│ Camera Driver Node │  
│ (realsense2\_camera) │  
│ - Publishes: /camera/color/image\_raw │  
│ : /camera/depth/image\_rect │  
│ : /camera/color/camera\_info │  
└────────┬─────────────────────────┘  
 │ ROS2 Topics (image\_transport)  
 ▼  
┌──────────────────────────────────┐  
│ Object Detection Node │  
│ - Subscribes: RGB image │  
│ - Runs: YOLOv8 inference │  
│ - Publishes: Detection2DArray │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Pose Estimation Node │  
│ - Subscribes: RGB-D, Detections │  
│ - Runs: PnP / Deep pose model │  
│ - Publishes: PoseArray │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Point Cloud Processor Node │  
│ - Subscribes: Depth image │  
│ - Runs: Deprojection, filtering │  
│ - Publishes: PointCloud2 │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Coordinate Transform Node │  
│ - Subscribes: Poses (cam frame) │  
│ - Uses: TF2 (cam → robot frame) │  
│ - Publishes: Poses (base frame) │  
└──────────────────────────────────┘

**Interfaces:** - **Input:** Camera USB stream (RGB 1920×1080 @ 30fps, Depth 1280×720 @ 30fps) - **Output:** - /vision/detected\_objects (vision\_msgs/Detection2DArray) - /vision/object\_poses (geometry\_msgs/PoseArray) - /vision/point\_cloud (sensor\_msgs/PointCloud2)

**Performance:** - Detection latency: <50ms (YOLOv8 on Jetson Xavier) - Pose estimation latency: <100ms - Total pipeline latency: <150ms (end-to-end)

### 1.7.2 5.2 Grasp Planning Subsystem

**Purpose:** Compute grasp poses given object pose and geometry

**Architecture:**

┌──────────────────────────────────┐  
│ Grasp Planner Node │  
│ - Service: /compute\_grasp │  
│ Request: object\_pose, point\_cloud │  
│ Response: grasp\_pose, quality │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Grasp Sampling Module │  
│ - Samples candidate grasps │  
│ - Methods: Centroid, GPD, GraspNet │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Collision Checking Module │  
│ - Checks gripper-object collision│  
│ - Uses: FCL (Flexible Collision Library) │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Grasp Ranking Module │  
│ - Computes quality metrics │  
│ - Metrics: Force closure, reachability │  
└────────┬─────────────────────────┘  
 │  
 ▼  
 Best Grasp

**Interfaces:** - **Input:** - Object pose (geometry\_msgs/PoseStamped) - Point cloud (sensor\_msgs/PointCloud2) - **Output:** - Grasp pose (geometry\_msgs/PoseStamped) - Quality score (float, 0-1)

**Performance:** - Grasp computation: <200ms - Success rate: >90% (for known objects)

### 1.7.3 5.3 Motion Planning Subsystem (MoveIt2)

**Purpose:** Plan collision-free trajectories from current state to goal

**Architecture:**

┌──────────────────────────────────┐  
│ MoveIt2 Move Group Interface │  
│ - Action: /move\_group/goal │  
│ Goal: target\_pose │  
│ Result: trajectory │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Planning Scene Manager │  
│ - Maintains collision objects │  
│ - Subscribes: /point\_cloud (obstacles) │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ IK Solver (KDL/TRAC-IK) │  
│ - Computes joint angles for pose│  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Path Planner (RRT\*, PRM) │  
│ - Searches collision-free path │  
│ - Library: OMPL │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Trajectory Generator │  
│ - Time-parameterizes path │  
│ - Respects velocity/accel limits│  
└────────┬─────────────────────────┘  
 │  
 ▼  
 Joint Trajectory

**Interfaces:** - **Input:** - Target pose (geometry\_msgs/PoseStamped) - Planning scene (obstacles) - **Output:** - Joint trajectory (trajectory\_msgs/JointTrajectory)

**Performance:** - Planning time: <500ms (typical) - Success rate: >95% (in uncluttered workspace)

### 1.7.4 5.4 Control & Execution Subsystem (ros2\_control)

**Purpose:** Execute trajectories with real-time feedback control

**Architecture:**

┌──────────────────────────────────┐  
│ Controller Manager │  
│ - Loads/unloads controllers │  
│ - Runs at 1 kHz │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Joint Trajectory Controller │  
│ - Subscribes: /joint\_trajectory │  
│ - Interpolates waypoints │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ PID Controller (per joint) │  
│ - Computes torque command │  
│ - Feedforward + feedback │  
└────────┬─────────────────────────┘  
 │  
 ▼  
┌──────────────────────────────────┐  
│ Hardware Interface │  
│ - Writes: motor commands (EtherCAT) │  
│ - Reads: encoder positions │  
└────────┬─────────────────────────┘  
 │  
 ▼  
 Robot Motors

**Interfaces:** - **Input:** - Joint trajectory (trajectory\_msgs/JointTrajectory) - **Output:** - Motor commands (EtherCAT PDO) - Joint states (sensor\_msgs/JointState)

**Performance:** - Control loop: 1 kHz - Trajectory tracking error: <2mm (Cartesian space)

### 1.7.5 5.5 Task Orchestration Subsystem

**Purpose:** High-level task sequencing, state management, error handling

**State Machine Diagram:**

┌──────────┐  
 │ IDLE │◄──────────────┐  
 └─────┬────┘ │  
 │ (Start) │  
 ▼ │  
 ┌──────────┐ │  
 │ SCAN │ │  
 └─────┬────┘ │  
 │ (Image captured) │  
 ▼ │  
 ┌──────────┐ │  
 │ DETECT │ │  
 └─────┬────┘ │  
 │ (Object found) │  
 ▼ │  
 ┌─────────────┐ │  
 │ PLAN\_GRASP │ │  
 └─────┬───────┘ │  
 │ (Grasp computed) │  
 ▼ │  
 ┌─────────────┐ │  
 │ PLAN\_PICK │ │  
 └─────┬───────┘ │  
 │ (Trajectory ready) │  
 ▼ │  
 ┌──────────────┐ │  
 │ EXECUTE\_PICK │ │  
 └─────┬────────┘ │  
 │ (Grasped) │  
 ▼ │  
 ┌──────────────┐ │  
 │ PLAN\_PLACE │ │  
 └─────┬────────┘ │  
 │ (Trajectory ready) │  
 ▼ │  
 ┌───────────────┐ │  
 │ EXECUTE\_PLACE │ │  
 └─────┬─────────┘ │  
 │ (Released) │  
 ▼ │  
 ┌──────────┐ │  
 │ VERIFY │ │  
 └─────┬────┘ │  
 │ (Success) │  
 └────────────────────┘  
  
 (Any error) → ┌───────┐  
 │ ERROR │  
 └───┬───┘  
 │ (Retry / Abort)  
 └─→ IDLE or previous state

**Interfaces:** - **Input:** - User commands (start, stop, reset) - Sensor events (object detected, grasp success/fail) - **Output:** - High-level commands to subsystems (scan, plan, execute) - System state (published to /system/state)

### 1.7.6 5.6 Monitoring & Logging Subsystem

**Architecture:**

┌────────────────────────────────────────┐  
│ Data Collection Agents │  
│ - ROS Topic Logger (rosbag2) │  
│ - Prometheus Exporters (node, GPU) │  
│ - Application Loggers (Python, C++) │  
└────────┬───────────────────────────────┘  
 │  
 ▼  
┌────────────────────────────────────────┐  
│ Storage Layer │  
│ - Time-Series: InfluxDB (metrics) │  
│ - Logs: Elasticsearch (text search) │  
│ - Bags: Local SSD (ROS bag files) │  
└────────┬───────────────────────────────┘  
 │  
 ▼  
┌────────────────────────────────────────┐  
│ Visualization & Alerting │  
│ - Grafana (dashboards) │  
│ - Kibana (log search) │  
│ - Alertmanager (threshold alerts) │  
└────────────────────────────────────────┘

**Key Metrics:** - **Performance:** Cycle time, throughput (picks/hour) - **Quality:** Grasp success rate, placement accuracy - **System Health:** CPU, GPU, RAM usage, temperatures - **Errors:** Error counts by type, mean time between failures (MTBF)

## 1.8 6. Data Architecture

### 1.8.1 6.1 Data Flow Diagram

Sensors (Camera, F/T)  
 │  
 ├─→ Vision Pipeline → Object Poses ─┐  
 │ │  
 └─→ Motor Encoders → Joint States ──┤  
 │  
 ▼  
 ┌─────────────────────┐  
 │ Core Processing │  
 │ (Planning, Control)│  
 └─────────┬───────────┘  
 │  
 ┌──────────────────────────┼──────────────────────┐  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────┐ ┌─────────────────┐ ┌──────────────────┐  
│ PostgreSQL │ │ Redis │ │ InfluxDB │  
│ (Persistent) │ │ (Cache) │ │ (Time-Series) │  
├─────────────────┤ ├─────────────────┤ ├──────────────────┤  
│ • Tasks │ │ • Session state │ │ • Sensor data │  
│ • Configs │ │ • Recent poses │ │ • Metrics │  
│ • Users │ │ • Temp results │ │ • Performance │  
└─────────────────┘ └─────────────────┘ └──────────────────┘

### 1.8.2 6.2 Database Schemas

#### 1.8.2.1 PostgreSQL (Relational Data)

**Table: tasks** | Column | Type | Description | |————–|———–|————-| | task\_id | SERIAL | Primary key | | object\_type | VARCHAR | Object class | | status | ENUM | pending, in\_progress, completed, failed | | start\_time | TIMESTAMP | Task start | | end\_time | TIMESTAMP | Task completion | | result | JSONB | Task outcome (success, error message) |

**Table: users** | Column | Type | Description | |————–|———–|————-| | user\_id | SERIAL | Primary key | | username | VARCHAR | Unique username | | password\_hash| VARCHAR | bcrypt hash | | role | ENUM | operator, engineer, admin | | created\_at | TIMESTAMP | Account creation |

#### 1.8.2.2 InfluxDB (Time-Series Data)

**Measurement: sensor\_data** - **Tags:** sensor\_id, type (camera, force, encoder) - **Fields:** value (float), unit (string) - **Timestamp:** nanosecond precision

**Measurement: performance\_metrics** - **Tags:** metric\_name (cycle\_time, success\_rate) - **Fields:** value (float) - **Timestamp:** nanosecond precision

## 1.9 7. Interface Design

### 1.9.1 7.1 ROS2 Topic Interfaces

| **Topic** | **Message Type** | **Publisher** | **Subscriber(s)** | **Hz** |
| --- | --- | --- | --- | --- |
| /camera/color/image\_raw | sensor\_msgs/Image | realsense2\_camera | vision\_pipeline | 30 |
| /vision/detected\_objects | vision\_msgs/Detection2DArray | object\_detector | grasp\_planner | 10 |
| /vision/object\_poses | geometry\_msgs/PoseArray | pose\_estimator | grasp\_planner, task\_orchestrator | 10 |
| /joint\_states | sensor\_msgs/JointState | hardware\_interface | moveit2, RViz | 100 |
| /joint\_trajectory | trajectory\_msgs/JointTrajectory | moveit2 | controller\_manager | on-demand |
| /system/state | std\_msgs/String | task\_orchestrator | monitoring, UI | 10 |

### 1.9.2 7.2 ROS2 Service Interfaces

| **Service** | **Type** | **Server** | **Purpose** |
| --- | --- | --- | --- |
| /compute\_grasp | custom\_msgs/ComputeGrasp | grasp\_planner | Compute grasp pose |
| /plan\_trajectory | moveit\_msgs/GetMotionPlan | moveit2 | Plan pick/place trajectory |
| /start\_task | std\_srvs/Trigger | task\_orchestrator | Start pick-place workflow |

### 1.9.3 7.3 ROS2 Action Interfaces

| **Action** | **Type** | **Server** | **Purpose** |
| --- | --- | --- | --- |
| /move\_group | moveit\_msgs/MoveGroup | moveit2 | Execute motion plan |
| /execute\_trajectory | control\_msgs/FollowJointTrajectory | controller\_manager | Execute trajectory with feedback |

### 1.9.4 7.4 REST API Interfaces

**Base URL:** https://<robot\_ip>:8000/api/v1

| **Endpoint** | **Method** | **Description** | **Auth** |
| --- | --- | --- | --- |
| /status | GET | Get system status | None |
| /start | POST | Start pick-place task | JWT |
| /stop | POST | Stop current task | JWT |
| /tasks | GET | List all tasks | JWT |
| /tasks/{id} | GET | Get task details | JWT |
| /config | GET | Get system config | JWT (admin) |
| /config | PUT | Update system config | JWT (admin) |

## 1.10 8. Deployment Architecture

### 1.10.1 8.1 Single-Machine Deployment (Development)

┌────────────────────────────────────────────────────────┐  
│ Intel NUC (Ubuntu 22.04 RT) │  
│ ┌──────────────────────────────────────────────────┐ │  
│ │ Docker Container: ros2\_workspace │ │  
│ │ - Vision Pipeline (CPU-based YOLO) │ │  
│ │ - MoveIt2, ros2\_control │ │  
│ │ - Task Orchestrator │ │  
│ │ - Monitoring (Prometheus) │ │  
│ └──────────────────────────────────────────────────┘ │  
│ ┌──────────────────────────────────────────────────┐ │  
│ │ Docker Container: data\_services │ │  
│ │ - PostgreSQL, Redis, InfluxDB │ │  
│ └──────────────────────────────────────────────────┘ │  
│ ┌──────────────────────────────────────────────────┐ │  
│ │ Docker Container: visualization │ │  
│ │ - Grafana, Kibana │ │  
│ └──────────────────────────────────────────────────┘ │  
└────────────────────────────────────────────────────────┘  
 │ USB 3.0 │ EtherCAT  
 ▼ ▼  
 RealSense D435 Servo Drives → Robot

**Advantages:** - Simple setup, single machine to manage - Low cost, suitable for prototyping

**Disadvantages:** - Limited compute for vision (CPU-only) - Single point of failure

### 1.10.2 8.2 Distributed Deployment (Production)

┌────────────────────────────────┐  
│ NVIDIA Jetson Xavier NX │  
│ - Vision Pipeline (GPU) │  
│ - YOLOv8 (TensorRT) │  
│ - Pose Estimation │  
└───────────┬────────────────────┘  
 │ ROS2 DDS (UDP multicast)  
 ▼  
┌────────────────────────────────┐  
│ Intel NUC (RT Linux) │  
│ - MoveIt2, ros2\_control │  
│ - Task Orchestrator │  
│ - Controller Manager (1kHz) │  
└───────────┬────────────────────┘  
 │ EtherCAT (real-time)  
 ▼  
┌────────────────────────────────┐  
│ Servo Drives (EtherCAT) │  
│ - Joint 1-6 drivers │  
└────────────────────────────────┘  
  
┌────────────────────────────────┐  
│ Edge Server (x86, optional) │  
│ - PostgreSQL, Redis, InfluxDB │  
│ - Grafana, Kibana │  
│ - REST API (FastAPI) │  
└────────────────────────────────┘

**Advantages:** - GPU acceleration for vision (Jetson) - Real-time control isolated on NUC - Scalable (add more Jetsons for multi-camera)

**Disadvantages:** - More complex networking (DDS configuration) - Higher cost

### 1.10.3 8.3 Cloud-Connected Deployment

┌───────────────────────────────────────────────────┐  
│ AWS / Azure Cloud │  
│ - MLflow (model registry) │  
│ - S3 / Blob Storage (rosbag backups) │  
│ - CloudWatch / Application Insights (monitoring) │  
│ - Grafana Cloud (dashboards) │  
└────────────────┬──────────────────────────────────┘  
 │ HTTPS, MQTT  
 ▼  
┌───────────────────────────────────────────────────┐  
│ Edge Gateway (on-premises) │  
│ - Data uplink (buffered, resilient) │  
│ - Local cache (Redis) │  
│ - Security (firewall, VPN) │  
└────────────────┬──────────────────────────────────┘  
 │ Internal network  
 ▼  
 ┌─────────────────────────┐  
 │ Robot Controller │  
 │ (same as distributed) │  
 └─────────────────────────┘

**Advantages:** - Centralized analytics across multiple sites - Cloud storage for long-term data retention - Remote monitoring

**Disadvantages:** - Requires reliable internet connection - Data privacy concerns (check regulations)

## 1.11 9. Security Architecture

### 1.11.1 9.1 Security Layers

┌──────────────────────────────────────────────┐  
│ Layer 5: Application Security │  
│ - Input validation, SQL injection prevention│  
└──────────────────────────────────────────────┘  
 ▲  
┌──────────────────────────────────────────────┐  
│ Layer 4: Authentication & Authorization│  
│ - OAuth2, JWT tokens, RBAC │  
└──────────────────────────────────────────────┘  
 ▲  
┌──────────────────────────────────────────────┐  
│ Layer 3: Data Encryption │  
│ - TLS 1.3 (HTTPS, gRPC), AES-256 (storage) │  
└──────────────────────────────────────────────┘  
 ▲  
┌──────────────────────────────────────────────┐  
│ Layer 2: Network Security │  
│ - Firewalls, VLANs, Intrusion Detection │  
└──────────────────────────────────────────────┘  
 ▲  
┌──────────────────────────────────────────────┐  
│ Layer 1: Physical Security │  
│ - Locked cabinets, E-stop, safety zones │  
└──────────────────────────────────────────────┘

### 1.11.2 9.2 Security Controls

| **Threat** | **Mitigation** |
| --- | --- |
| Unauthorized access | OAuth2 authentication, JWT tokens |
| Data breach | TLS encryption (in transit), AES encryption (at rest) |
| Denial of Service | Rate limiting (100 req/min), firewall rules |
| Code injection | Input validation, parameterized queries |
| Privilege escalation | RBAC (operator, engineer, admin roles) |
| Insider threat | Audit logging (immutable), separation of duties |

## 1.12 10. Scalability & Performance

### 1.12.1 10.1 Scalability Dimensions

| **Dimension** | **Approach** | **Limit** |
| --- | --- | --- |
| **Concurrent Robots** | ROS2 DDS namespaces, multi-master | 10 robots/network |
| **Throughput** | Parallel planning, GPU acceleration | 60 picks/min (2× current) |
| **Data Volume** | InfluxDB downsampling, log rotation | 1 TB/month |
| **Users** | Load balancer (NGINX), stateless API | 100 concurrent |

### 1.12.2 10.2 Performance Requirements

| **Metric** | **Target** | **Current** | **Gap** |
| --- | --- | --- | --- |
| Cycle Time | 2 sec | 3 sec | -1 sec (optimize trajectory) |
| Vision Latency | <50ms | 45ms | ✅ Met |
| Control Loop | 1 kHz | 1 kHz | ✅ Met |
| API Response | <100ms | 80ms | ✅ Met |
| Uptime | 99.5% | 98.2% | +1.3% (improve error handling) |

### 1.12.3 10.3 Performance Optimization Strategies

1. **Vision Pipeline:**
   * TensorRT quantization (FP16 → 2× speedup)
   * Reduce input size (640×640 → 512×512)
2. **Motion Planning:**
   * Pre-computed IK solutions (lookup table)
   * Trajectory caching for repeated tasks
3. **Control:**
   * RT-Preempt kernel tuning (CPU isolation)
   * Feedforward compensation (reduce PID load)

## 1.13 11. Appendices

### 1.13.1 Appendix A: Technology Versions

| Component | Version |
| --- | --- |
| ROS2 | Humble (LTS) |
| Ubuntu | 22.04 LTS |
| Python | 3.10 |
| Docker | 24.0.5 |
| PostgreSQL | 15.3 |
| InfluxDB | 2.7.1 |
| Grafana | 10.0.3 |

### 1.13.2 Appendix B: Key Design Decisions

| **Decision** | **Rationale** |
| --- | --- |
| ROS2 over ROS1 | Real-time support, better security, active development |
| CycloneDDS over FastDDS | Better performance on small networks |
| PostgreSQL over MySQL | JSONB support, extensibility |
| YOLOv8 over Faster R-CNN | Real-time inference, good accuracy |
| Docker over bare metal | Reproducibility, easy deployment |

### 1.13.3 Appendix C: Glossary

| Term | Definition |
| --- | --- |
| OMPL | Open Motion Planning Library |
| PCL | Point Cloud Library |
| TF2 | ROS2 transform library |
| URDF | Unified Robot Description Format |
| DDS | Data Distribution Service |

## 1.14 Document Approval

| **Role** | **Name** | **Signature** | **Date** |
| --- | --- | --- | --- |
| System Architect | TBD |  |  |
| Lead Engineer | TBD |  |  |
| Project Manager | TBD |  |  |

**Document Status:** ✅ Complete **Last Updated:** 2025-10-18 **Next Review:** 2025-11-18 **Version Control:** Git (branch: main)