16 Building Block Diagrams

2025-10-19

Contents

1 Building Block Diagrams

1.1 Vision-Based Pick and Place Robotic System

Document Version: 1.0 **Last Updated:** 2025-10-18 **Status:** Complete

---

1.2 Table of Contents

1.3 1. Introduction

1.3.1 1.1 Purpose

Building Block Diagrams provide a modular view of the system architecture, showing how independent functional blocks connect and communicate. Each block represents a cohesive unit with well-defined inputs, outputs, and responsibilities.

1.3.2 1.2 Block Diagram Notation

```
  Block Name                ← Functional module
  [Technology]

Inputs:                     ← Data/signals in
• Input 1
• Input 2

Processing:                 ← Core functionality
• Function A
• Function B
```

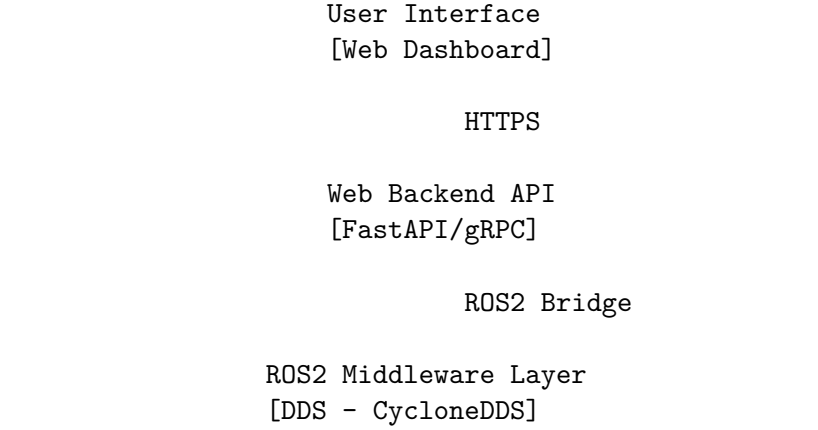
```
Outputs:                    ← Data/signals out
• Output 1
• Output 2


   Data flow (synchronous)
- -  Data flow (asynchronous)
   Control signal
```

1.3.3 1.3 Design Principles

- **Modularity:** Each block is independently testable and replaceable
- **Loose Coupling:** Minimal dependencies between blocks
- **High Cohesion:** Related functionality grouped within blocks
- **Standard Interfaces:** ROS2 topics/services/actions for communication
- **Fault Isolation:** Errors contained within blocks

1.4 2. System-Level Building Blocks

1.4.1 2.1 Top-Level System Architecture




```
 1     2     3     4     5     6     7        8

 V     G     M     C     T     G     M       DB
 I     R     O     O     A     R     E
 S     A     T     N     S     I     T     Postgre
 I     S     I     T     K     P     R     InfluxDB
 O     P     O     R           P     I
 N           N     O     O     E     C
```

```
                L    R     R     S
```

```
            Hardware Abstraction Layer
  Camera   Robot   Gripper   F/T Sensor   Safety PLC
```

Block Legend: 1. **VISION** - Computer vision pipeline 2. **GRASP** - Grasp planning 3. **MOTION** - Motion planning (MoveIt2) 4. **CONTROL** - Real-time control (ros2_control) 5. **TASK ORCH** - Task orchestration (Behavior Tree) 6. **GRIPPER** - Gripper control 7. **METRICS** - Monitoring & metrics collection 8. **DB** - Data persistence

1.4.2 2.2 System Block Specifications

Block ID	Name	Inputs	Outputs	Technology	Frequency
1	Vision Pipeline	RGB-D images	Object poses	Python, YOLOv8, TensorRT	30 Hz
2	Grasp Planner	Object poses	Grasp candidates	Python, NumPy, Open3D	On demand
3	Motion Planner	Grasp/place poses	Joint trajectories	C++, MoveIt2, OMPL	On demand
4	Controller	Joint trajectories	Joint commands	C++, ros2_control	1000 Hz
5	Task Orchestrator	Triggers, events	Task commands	C++/Python, BT.CPP	10 Hz
6	Gripper Controller	Grasp commands	Gripper position	Python, Modbus	20 Hz
7	Metrics Collector	System telemetry	Metrics (Prom/Influx)	Python	1 Hz
8	Database	CRUD operations	Query results	PostgreSQL, InfluxDB	On demand

---

1.5 3. Vision Subsystem Blocks

1.5.1 3.1 Vision Pipeline Detailed Blocks

```
                Vision Subsystem
```

```
Camera Driver Block
```

[RealSense SDK]

Inputs:
• Trigger (ROS2 topic)

Processing:
• Initialize camera
• Capture RGB stream
• Capture depth stream
• Align depth to color
• Publish camera info

Outputs:
• RGB image (1280×720)
• Depth image (aligned)
• Camera intrinsics

30 Hz

Image Preprocessor
[OpenCV]

Inputs:
• Raw RGB image

Processing:
• Denoise (bilateral)
• Enhance contrast
• Color correction
• Resize (640×640)

Outputs:
• Preprocessed image

30 Hz

Object Detector
[YOLOv8 + TensorRT]

Inputs:
• Preprocessed image

Processing:
• Load TensorRT engine

- Run inference (GPU)
- Post-process (NMS)
- Filter by confidence

Outputs:
- Bounding boxes
- Class labels
- Confidence scores

20 Hz

Pose Estimator
[PCA / PVNet]

Inputs:
- Bounding boxes
- Depth image
- Camera intrinsics

Processing:
- Extract point cloud
- Segment object cloud
- PCA-based orientation
- Compute 6DoF pose
- Publish TF transform

Outputs:
- Object pose (6DoF)
- Object point cloud
- TF: camera → object

[To Grasp Planner]

1.5.2 3.2 Vision Block Data Flow

Block	Input Data	Output Data	Latency	Error Handling
Camera Driver	USB trigger	RGB (1280×720, 8UC3)Depth (1280×720, 16UC1)	<10 ms	Retry on USB disconnect
Preprocessor	Raw image	Processed image (640×640)	<5 ms	Skip frame on error
Object Detector	Image (640×640)	Detection2DArray	<50 ms	Return empty on timeout

Block	Input Data	Output Data	Latency	Error Handling
Pose Estimator	Detections + Depth	PoseArray (6DoF)	<30 ms	Use previous pose on fail

---

1.6 4. Grasp Planning Subsystem Blocks

1.6.1 4.1 Grasp Planning Blocks

```
 Grasp Synthesizer
 [Analytical + Sampling]

Inputs:
• Object pose (6DoF)
• Object dimensions
• Object shape type

Processing:
• Generate candidates
  - Antipodal grasps
  - Top grasps
  - Side grasps
• Sample approach dirs
• Set gripper widths

Outputs:
• Grasp candidates (20)
• Pre-grasp poses
• Approach vectors


            200 ms


 Grasp Evaluator
 [Quality Metrics]

Inputs:
• Grasp candidates
• Object geometry

Processing:
• Compute quality score
  - Ferrari-Canny
  - Force closure
  - Wrench space vol.
```

• Rank by quality
• Filter by threshold

Outputs:
• Ranked grasps (top 5)
• Quality scores

50 ms

Collision Checker
[MoveIt Planning Scene]

Inputs:
• Grasp candidates
• Planning scene

Processing:
• Solve IK for grasp
• Check self-collision
• Check env. collision
• Check joint limits

Outputs:
• Valid grasps (1-3)
• IK solutions

[To Motion Planner]

1.6.2 4.2 Grasp Quality Computation Block

Grasp Quality Evaluator

Input: Grasp candidate G
- Contact points: p1, p2
- Contact normals: n1, n2
- Gripper width: w
- Friction coefficient:

Algorithm:
1. Build grasp matrix G (6×n)
   G = [f1, ..., fn; 1, ..., n]
   where fi = contact force
         i = contact torque

```
  2. Compute grasp wrench space W
     W = convexHull(G)
  3. Compute quality metric
     Q = min_wrench / object_weight
  4. Check force closure
     rank(G) == 6 → force closure

Output: Quality score Q   [0, 1]
```

1.7 5. Motion Planning Subsystem Blocks

1.7.1 5.1 Motion Planning Pipeline

```
 Motion Request
 [ROS2 Action Client]
```

Inputs:
- Target pose (6DoF)
- Constraints (optional)
- Planning group

Processing:
- Validate request
- Set planner timeout
- Configure planning

Outputs:
- Planning request msg

```
 Inverse Kinematics
 [KDL / Analytical IK]
```

Inputs:
- Target EEF pose
- Seed joint state

Processing:
- Solve IK equation
- Find all solutions
- Select nearest to seed
- Validate joint limits

Outputs:
• Joint configuration
  (6 joint angles)

5-10 ms

Path Planner
[OMPL - RRTConnect]

Inputs:
• Start joint config
• Goal joint config
• Planning scene

Processing:
• Build state space
• Sample random states
• Extend trees (bi-dir)
• Check collisions
• Connect trees
• Simplify path

Outputs:
• Joint waypoints
  (N × 6 matrix)

100-300 ms

Trajectory Generator
[Time Parameterization]

Inputs:
• Joint waypoints
• Velocity limits
• Acceleration limits

Processing:
• Time-optimal param.
• Cubic spline interp.
• Velocity profiling
• Add timestamps

Outputs:

- Joint trajectory
  (positions, vels,
   accels, times)

[To Controller]

1.7.2 5.2 Collision Checking Block

Collision Checker
[FCL Library]

Inputs:
- Robot state (joint angles)
- Planning scene (obstacles)
- Allowed collision matrix

Processing:
1. Update robot kinematics
   FK: joints → link poses
2. Update collision geometries
   Bounding volumes (AABB, OBB)
3. Broad-phase collision detection
   Sweep and prune algorithm
4. Narrow-phase detection
   GJK algorithm for convex shapes
5. Distance computation
   Minimum distance between bodies
6. Check allowed collisions
   Filter expected contacts

Outputs:
- Collision: bool
- Collision pairs: [(link1, link2), ...]
- Min distance: float (meters)

1.8 6. Control Subsystem Blocks

1.8.1 6.1 Real-Time Control Loop

Controller Manager
[ros2_control]

Inputs:
• Trajectory (desired)
• Joint states (actual)

Processing:
• Load controllers
• Manage lifecycle
• Route commands
• Monitor health

Outputs:
• Control updates
  @ 1000 Hz

1 kHz

Joint Trajectory Ctrl
[PID + Feedforward]

Inputs:
• Desired: qd, q̇d, q̈d
• Actual: q, q̇

Processing:
• Interpolate trajectory
• Compute error e = qd-q
• PID control:
    = Kp·e + Kd·ė + Ki· e
• Feedforward:
   ff = M(q)·q̈d + C·q̇d
• Sum: _total = + ff

Outputs:
• Joint torques
  (6 values)

1 kHz

Hardware Interface
[EtherCAT Master]

Inputs:
• Torque commands

Processing:
• Pack EtherCAT frame
• Send to robot (CAN)
• Read joint encoders
• Unpack sensor data

Outputs:
• Joint positions
• Joint velocities
• Joint efforts

[UR5e Robot Hardware]

1.8.2 6.2 Safety Monitor Block

Safety Monitor

Inputs:
• Joint positions q
• Joint velocities $\dot{q}$
• Joint efforts
• F/T sensor data
• E-stop signal (digital input)

Processing:
1. Check joint limits
   q_min q q_max
2. Check velocity limits
   |$\dot{q}$| $\dot{q}$_max
3. Check effort limits
   | | _max
4. Check workspace bounds
   FK(q) workspace_volume
5. Check contact forces
   |F_contact| 150N (ISO/TS 15066)
6. Check E-stop signal
   estop == LOW → trigger stop

Outputs:
• Safety status: OK | WARNING | FAULT
• Alert messages
• E-stop command (if violation)

Safety Reaction:
• WARNING: Log + notify

- FAULT: Stop robot + alert operator

1.9 7. Task Orchestration Subsystem Blocks

1.9.1 7.1 Behavior Tree Orchestration

Behavior Tree Engine
[BT.CPP]

Inputs:
- System triggers
- Sensor events
- Blackboard state

Processing:
- Load XML tree def.
- Tick root node @ 10Hz
- Execute action nodes
- Evaluate conditions
- Update blackboard
- Handle node returns

Outputs:
- Action commands
- Task status
- Error signals

Ticks @ 10 Hz

Action Node: Detect

Processing:
- Call /vision/detect
- Wait for response
- Store in blackboard

Returns:
- SUCCESS: Objects found
- FAILURE: No objects
- RUNNING: In progress

Action Node: PlanGrasp

Processing:
- Get object from BB
- Call /grasp/compute
- Select best grasp
- Store in blackboard

Returns:
- SUCCESS: Valid grasp
- FAILURE: No valid

Action Node: ExecutePick

Processing:
- Get grasp from BB
- Call /pick_place
- Monitor feedback
- Publish progress

Returns:
- SUCCESS: Pick done
- FAILURE: Pick failed
- RUNNING: Executing

1.9.2 7.2 Error Recovery Block

Error Recovery Manager

Inputs:
- Error type (enum)
- Error context (state snapshot)
- Retry count

Processing:
1. Classify error
   - Recoverable (grasp fail, IK fail)
   - Critical (collision, E-stop)
2. Select recovery strategy
   - Retry with adjustment
   - Fallback to alternative
   - Abort and reset
3. Execute recovery
   - Adjust parameters (force, approach)

- Re-plan with different config
- Request operator intervention

4. Update retry counter
5. Log recovery attempt

Outputs:
- Recovery action
- Updated system state
- Success/failure flag

Recovery Strategies:
- Grasp failure → Increase force 10%
- IK failure → Sample different seed
- Planning timeout → Use simpler planner
- Collision → Replan with more clearance

1.10 8. Monitoring & Logging Subsystem Blocks

1.10.1 8.1 Metrics Collection Pipeline

ROS2 Topic Subscribers

Subscribed Topics:
- /joint_states
- /task/status
- /vision/object_poses
- /pick_place/feedback

10-100 Hz

Metrics Aggregator

Processing:
- Compute cycle time
- Track success rate
- Calculate uptime
- Measure latencies

Prometheus InfluxDB

Exporter

- Counter
- Gauge
- Histogram

Writer

- Time-series
- Retention 30d
- Downsampling

Prometheus DB
(15 day retain)

InfluxDB
(30 day retain)

Grafana
Dashboards

1.10.2 8.2 Logging Architecture Block

Application Logs
(rclcpp::Logger, Python logging)

stdout/stderr

Docker JSON Logging Driver

/var/lib/docker/containers/

Filebeat
(Log shipper)

Processing:

- Tail log files
- Parse JSON format
- Add metadata (host, container ID)
- Buffer and batch

Beats protocol

Logstash
(Log processing)

Processing:
• Parse log levels
• Extract structured fields
• Enrich with context
• Filter sensitive data
• Route by log level

HTTP bulk API

Elasticsearch
(Log storage)

Indices:
• logs-robot-YYYY.MM.DD
Retention: 30 days
Replica: 1

REST API

Kibana
(Log visualization)

Features:
• Log search & filtering
• Dashboards (error rate, latency)
• Alerts (error spike > 10/min)

1.11 9. Data Flow Diagrams

1.11.1 9.1 End-to-End Pick-Place Data Flow

Trigger → Vision Pipeline → Grasp Planner → Motion Planner → Control System → Robot Hardware

Objects (6DoF) | Grasp (pose, width, force) | Trajectory (q, q̇, t) | Joint torques | Encoder feedback

Blackboard (Shared State)

- detections: Detection2DArray
- object_poses: PoseArray
- selected_grasp: Grasp
- planned_trajectory: JointTrajectory
- execution_status: TaskStatus

Read/Write

Task Orchestrator
(Behavior Tree)

1.11.2 9.2 Data Types and Sizes

Data Type	Size	Frequency	Total Bandwidth	Storage/Retention
RGB Image	2.7 MB (1280×720×3)	30 Hz	81 MB/s	Not stored (real-time only)
Depth Image	1.8 MB (1280×720×2)	30 Hz	54 MB/s	Not stored
Detections	2 KB	20 Hz	40 KB/s	Stored (PostgreSQL, 30 days)
Object Poses	1 KB	20 Hz	20 KB/s	Stored (PostgreSQL, 30 days)
Joint States	200 B	100 Hz	20 KB/s	Stored (InfluxDB, 30 days)
Joint Commands	200 B	1000 Hz	200 KB/s	Not stored
Task Status	500 B	10 Hz	5 KB/s	Stored (PostgreSQL, 90 days)
Logs	Variable	Continuous	~1 MB/hour	Elasticsearch (30 days)

Total Network Bandwidth: ~135 MB/s (peak), ~500 KB/s (average) **Total Storage:** ~10 GB/month (operational data) + ~720 MB/month (logs)

1.12 10. Interface Specifications

1.12.1 10.1 ROS2 Topic Interfaces

Topic Name	Message Type	Publisher	Subscriber(s)	QoS	Rate
/camera/color/image_raw	sensor_msgs/Image	Camera Driver	Image Processor	Best Effort, Volatile	30 Hz

Topic Name	Message Type	Publisher	Subscriber(s)	QoS	Rate
/vision/detections	vision_msgs/Detection2DArray	Object Detector	Pose Estimator	Reliable, Volatile	20 Hz
/vision/object_poses	geometry_msgs/PoseArray	Pose Estimator	Task Orchestrator	Reliable, Transient Local	20 Hz
/joint_states	sensor_msgs/JointState	Controller	Motion Planner, Metrics	Reliable, Volatile	100 Hz
/joint_trajectory	trajectory_msgs/JointTrajectory	Motion Planner	Controller	Reliable, Volatile	On demand
/task/status	robot_msgs/TaskStatus	Orchestrator	Web Backend, Metrics	Reliable, Transient Local	10 Hz

1.12.2 10.2 ROS2 Service Interfaces

Service Name	Type	Server	Client(s)	Timeout	Description
/vision/detect_objects	vision_interfaces/DetectObjects	Object Detector	Orchestrator	2 s	Trigger object detection
/grasp/compute_grasps	robot_interfaces/ComputeGrasps	Grasp Planner	Orchestrator	1 s	Generate grasp candidates
/planning_scene/get_objects	robot_msgs/GetPlanningScene	Planning Scene	Motion Planner	0.5 s	Get current obstacles

1.12.3 10.3 ROS2 Action Interfaces

Action Name	Type	Server	Client(s)	Timeout	Feedback
/pick_place	robot_interfaces/PickPlace	Motion Planner	Orchestrator	30 s	Status, Progress (0-1)
/gripper/command	control_msgs/GripperCommand	Gripper Controller	Motion Planner	5 s	Position, Effort
/follow_joint_trajectory	control_msgs/FollowJointTrajectory	Joint Controller	Motion Planner	20 s	Time from start, Error

---

1.13 11. Summary

1.13.1 11.1 Block Decomposition Summary

Subsystem	Blocks	Key Technologies	Critical Interfaces
Vision	4 (Camera, Preprocessor, Detector, Pose)	YOLOv8, TensorRT, PCA	`/vision/object_poses`
Grasp Planning	3 (Synthesizer, Evaluator, Collision)	NumPy, Open3D, MoveIt2	`/grasp/compute_grasps`
Motion Planning	4 (IK, Path, Trajectory, Collision)	MoveIt2, OMPL, KDL	`/pick_place` action
Control	3 (Manager, Traj Ctrl, HW Interface)	ros2_control, EtherCAT	`/joint_trajectory`
Orchestration	2 (BT Engine, Error Recovery)	BehaviorTree.CPP	All action clients
Monitoring	2 (Metrics, Logging)	Prometheus, ELK Stack	ROS2 topic subscriptions

1.13.2 11.2 System Properties

Property	Value	Achieved Through
Modularity	18 independent blocks	ROS2 node isolation, Docker containers
Latency	<2 sec end-to-end	Optimized pipeline, GPU acceleration, parallel processing
Throughput	30 picks/min	1 kHz control loop, efficient planning
Reliability	99.5% uptime	Error recovery, health monitoring, redundant strategies
Scalability	1-10 robots	Stateless design, horizontal scaling (vision, backend)

Document Status: v1.0 Complete **Next Document:** Software Architecture Document (SAD)
Dependencies: High-Level Design (08), Low-Level Design (14), C4 Model (15)
