Table of Contents

# 1 Testing & Validation Plan

## 1.1 Vision-Based Pick and Place Robotic System

## 1.2 Document Control

| **Item** | **Details** |
| --- | --- |
| **Document Title** | Testing & Validation Plan |
| **Version** | 1.0 |
| **Date** | 2025-10-18 |
| **Status** | Draft |
| **Author(s)** | QA Lead, Test Engineer |
| **Approvers** | Tech Lead, Project Manager |

## 1.3 1. Introduction

### 1.3.1 1.1 Purpose

This document defines the comprehensive testing and validation strategy for the vision-based pick-and-place robotic system, ensuring: - **Functional correctness:** System performs as specified - **Performance:** Meets cycle time, accuracy, throughput targets - **Safety:** Complies with ISO 10218, ISO/TS 15066 - **Reliability:** 99.5% uptime target - **User acceptance:** Satisfies end-user requirements

### 1.3.2 1.2 Scope

**In Scope:** - Unit testing (individual modules/functions) - Integration testing (module-to-module interfaces) - System testing (end-to-end workflows) - Performance testing (latency, throughput, stress) - Safety testing (E-stop, collision detection, force limiting) - Acceptance testing (customer sign-off) - Regression testing (after changes)

**Out of Scope:** - Penetration testing (covered in Security Plan) - Long-term reliability testing (>6 months, post-deployment) - Field testing at customer sites (deployment phase)

### 1.3.3 1.3 Test Levels

Level 1: Unit Tests (Developer-driven, pytest, gtest)  
 ↓  
Level 2: Integration Tests (Module interfaces, ROS2 launch tests)  
 ↓  
Level 3: System Tests (End-to-end workflows, simulation + hardware)  
 ↓  
Level 4: Acceptance Tests (Customer requirements, real environment)

## 1.4 2. Test Strategy

### 1.4.1 2.1 Testing Pyramid

┌─────────────────────┐  
 │ Acceptance (10%) │ ← Few, slow, expensive  
 ├─────────────────────┤  
 │ System (20%) │  
 ├─────────────────────┤  
 │ Integration (30%) │  
 ├─────────────────────┤  
 │ Unit (40%) │ ← Many, fast, cheap  
 └─────────────────────┘

**Rationale:** More low-level tests (fast feedback), fewer high-level tests (high confidence).

### 1.4.2 2.2 Test Approach

| **Test Level** | **Approach** | **Environment** | **Tools** |
| --- | --- | --- | --- |
| **Unit** | White-box, code coverage | Dev machine | pytest, gtest, coverage.py |
| **Integration** | Black-box, interface contracts | Dev + CI/CD | ROS2 launch\_testing, mock services |
| **System** | Black-box, end-to-end scenarios | Simulation (Gazebo) + Real hardware | Manual + automated scripts |
| **Performance** | Benchmark-driven, metrics | Real hardware | JMeter, Locust, custom profilers |
| **Safety** | Compliance-driven, audit | Real hardware + safety setup | Manual inspection, cert tools |
| **Acceptance** | Requirement-driven, UAT | Customer environment | Customer-defined tests |

## 1.5 3. Unit Testing

### 1.5.1 3.1 Objectives

* Verify individual functions/classes work correctly
* Achieve >80% code coverage
* Fast execution (<5 min for full suite)
* Run on every commit (CI/CD)

### 1.5.2 3.2 Test Cases (Examples)

#### 1.5.2.1 3.2.1 Vision Pipeline

**Test: Object Detection** - **Input:** Image with 1 red cube - **Expected:** Bounding box at (x=320, y=240, w=100, h=100), confidence >0.9 - **Assertion:** python detections = detector.detect(image) assert len(detections) == 1 assert detections[0].class\_name == "cube" assert detections[0].confidence > 0.9

**Test: Pose Estimation** - **Input:** RGB-D image, object mask - **Expected:** 6DoF pose (x,y,z,qx,qy,qz,qw) - **Assertion:** python pose = estimator.estimate\_pose(image, mask) assert abs(pose.position.z - 0.5) < 0.01 # Object at 50cm height

#### 1.5.2.2 3.2.2 Grasp Planning

**Test: Grasp Sampling** - **Input:** Object pose, point cloud - **Expected:** List of 10 grasp candidates - **Assertion:** python grasps = planner.sample\_grasps(pose, cloud) assert len(grasps) >= 10 assert all(g.quality > 0.5 for g in grasps)

#### 1.5.2.3 3.2.3 Motion Planning

**Test: IK Solver** - **Input:** Target pose (x=0.5, y=0.2, z=0.3, roll=0, pitch=π/2, yaw=0) - **Expected:** Joint angles [θ1, θ2, θ3, θ4, θ5, θ6], IK success=True - **Assertion:** python joint\_angles, success = ik\_solver.solve(target\_pose) assert success == True assert len(joint\_angles) == 6 # Verify FK(IK(pose)) == pose fk\_pose = fk\_solver.compute(joint\_angles) assert np.allclose(fk\_pose, target\_pose, atol=0.001)

### 1.5.3 3.3 Coverage Goals

| **Module** | **Target Coverage** | **Current** | **Gap** |
| --- | --- | --- | --- |
| vision\_pipeline | 85% | TBD | TBD |
| grasp\_planner | 80% | TBD | TBD |
| motion\_planner (custom code) | 90% | TBD | TBD |
| task\_orchestrator | 75% | TBD | TBD |
| **Overall** | **80%** | **TBD** | **TBD** |

### 1.5.4 3.4 Test Execution

**Command:**

colcon test --packages-select vision\_pipeline grasp\_planner  
colcon test-result --all --verbose

**CI/CD Integration:** - GitHub Actions runs tests on every PR - Fail CI if coverage <80% - Report coverage to Codecov.io

## 1.6 4. Integration Testing

### 1.6.1 4.1 Objectives

* Verify modules communicate correctly (ROS2 topics, services, actions)
* Test data flow between subsystems
* Detect interface mismatches early

### 1.6.2 4.2 Test Cases (Examples)

#### 1.6.2.1 4.2.1 Vision → Grasp Planning

**Test:** Detected object pose flows to grasp planner - **Setup:** Launch vision\_pipeline and grasp\_planner nodes - **Action:** Publish mock RGB-D image with object - **Expected:** Grasp planner receives /vision/object\_poses message within 200ms - **Assertion:** ```python # launch\_testing syntax def test\_vision\_to\_grasp(): vision\_node = launch\_node(‘vision\_pipeline’) grasp\_node = launch\_node(‘grasp\_planner’)

pub = Publisher('/camera/color/image\_raw', Image)  
 sub = Subscriber('/grasp/candidates', GraspArray)  
  
 pub.publish(mock\_image)  
 msg = sub.wait\_for\_message(timeout=1.0)  
 assert msg is not None  
 assert len(msg.grasps) > 0

#### 4.2.2 Motion Planning → Control  
  
\*\*Test:\*\* Trajectory execution action completes  
- \*\*Setup:\*\* Launch moveit2 and ros2\_control nodes  
- \*\*Action:\*\* Send `FollowJointTrajectory` action goal  
- \*\*Expected:\*\* Action succeeds, robot reaches goal within tolerance  
- \*\*Assertion:\*\*  
```python  
client = ActionClient('/joint\_trajectory\_controller/follow\_joint\_trajectory', FollowJointTrajectory)  
goal = FollowJointTrajectory.Goal(trajectory=test\_trajectory)  
future = client.send\_goal\_async(goal)  
result = future.result(timeout=10.0)  
assert result.error\_code == FollowJointTrajectory.Result.SUCCESSFUL

### 1.6.3 4.3 Test Environment

**Simulation:** - Use Gazebo for realistic robot/environment simulation - Mock camera publishes synthetic images - Fast iteration, no hardware risk

**Hardware-in-Loop (Optional):** - Real camera, simulated robot (or vice versa) - Validate sensor drivers, communication latency

## 1.7 5. System Testing

### 1.7.1 5.1 Objectives

* Validate end-to-end workflows (idle → scan → detect → pick → place → home)
* Test in realistic scenarios (cluttered workspace, varying lighting)
* Verify all requirements met

### 1.7.2 5.2 Test Scenarios

#### 1.7.2.1 5.2.1 Nominal Pick-Place (Sunny Day)

**Scenario:** Single object, ideal conditions 1. **Pre-conditions:** - Robot at home position - 1 red cube (50×50×50mm) on table at (x=0.5, y=0.2, z=0.05) - Camera operational, lighting uniform (2000 lumen) 2. **Steps:** - Press “Start” button - System scans workspace (camera captures image) - Vision detects cube (bounding box, pose) - Grasp planner computes top-down grasp - Motion planner generates pick trajectory - Robot executes pick (gripper closes, force=20N) - Motion planner generates place trajectory - Robot executes place at target (x=0.3, y=-0.2, z=0.05) - Robot returns home 3. **Expected Results:** - ✅ Cycle time: <10 seconds - ✅ Object successfully placed at target - ✅ Placement error: <5mm - ✅ No collisions detected

#### 1.7.2.2 5.2.2 Multi-Object Sequential Picking

**Scenario:** 5 objects, pick all sequentially 1. **Pre-conditions:** 5 colored cubes randomly placed on table 2. **Steps:** For each object: scan → detect → pick → place 3. **Expected Results:** - ✅ All 5 objects picked and placed - ✅ Total cycle time: <60 seconds - ✅ Success rate: 100% (0 failures)

#### 1.7.2.3 5.2.3 Error Recovery (Grasp Failure)

**Scenario:** Intentional grasp failure, test retry logic 1. **Pre-conditions:** - Slippery object (low friction) - Grasp force reduced to 50% (to induce failure) 2. **Steps:** - Robot attempts pick - F/T sensor detects drop (force spike → 0N) - System logs error: “Grasp failed” - System retries with increased force (100%) - Second attempt succeeds 3. **Expected Results:** - ✅ Failure detected within 500ms - ✅ Retry succeeds - ✅ Event logged with timestamp

#### 1.7.2.4 5.2.4 Occlusion Handling

**Scenario:** Partially occluded object 1. **Pre-conditions:** Object A partially hidden behind object B 2. **Steps:** - Scan workspace - Vision detects visible objects (A partially visible, B fully visible) - System picks B first (higher confidence) - Re-scan after picking B - Now A fully visible, system picks A 3. **Expected Results:** - ✅ Both objects eventually picked - ✅ No collisions with occluding objects

## 1.8 6. Performance Testing

### 1.8.1 6.1 Objectives

* Measure and validate performance metrics
* Identify bottlenecks
* Ensure real-time constraints met

### 1.8.2 6.2 Test Cases

#### 1.8.2.1 6.2.1 Cycle Time Test

**Objective:** Measure average cycle time - **Setup:** 100 pick-place cycles (single object) - **Metrics:** - Mean cycle time - P50, P95, P99 percentiles - Standard deviation - **Pass Criteria:** Mean <2.5 sec, P95 <3.0 sec

**Test Procedure:**

cycle\_times = []  
for i in range(100):  
 start = time.time()  
 execute\_pick\_place()  
 end = time.time()  
 cycle\_times.append(end - start)  
  
mean\_time = np.mean(cycle\_times)  
p95\_time = np.percentile(cycle\_times, 95)  
assert mean\_time < 2.5, f"Mean cycle time {mean\_time}s exceeds 2.5s"  
assert p95\_time < 3.0, f"P95 cycle time {p95\_time}s exceeds 3.0s"

#### 1.8.2.2 6.2.2 Vision Latency Test

**Objective:** Measure vision pipeline latency - **Setup:** 1000 images processed - **Metrics:** - Detection latency (image → bounding boxes) - Pose estimation latency (image → 6DoF pose) - **Pass Criteria:** Detection <50ms, Pose <100ms

#### 1.8.2.3 6.2.3 Control Loop Jitter Test

**Objective:** Measure real-time control loop stability - **Setup:** 1 hour continuous operation, log loop timings - **Metrics:** - Mean loop time (should be 1ms for 1kHz) - Jitter (std dev) - Max jitter - **Pass Criteria:** Mean=1ms ±0.1ms, Max jitter <2ms

**Tool:** cyclictest (Linux RT benchmark)

sudo cyclictest -p 90 -t 1 -n -a 1 -D 3600 -m -q

#### 1.8.2.4 6.2.4 Throughput Test

**Objective:** Max picks per hour (continuous operation) - **Setup:** 1-hour run, unlimited objects (refill bin as needed) - **Metrics:** Total picks in 1 hour - **Pass Criteria:** ≥1800 picks/hour (30 picks/min)

### 1.8.3 6.3 Load Testing (API)

**Objective:** Test REST API under concurrent load - **Tool:** Locust (Python load testing framework) - **Scenario:** 100 concurrent users, each calling /start API - **Pass Criteria:** 95% requests complete <100ms, 0% errors

## 1.9 7. Safety Testing

### 1.9.1 7.1 Objectives

* Verify compliance with ISO 10218 (robot safety), ISO/TS 15066 (collaborative robots)
* Validate E-stop functionality
* Test collision detection, force limiting

### 1.9.2 7.2 Test Cases

#### 1.9.2.1 7.2.1 Emergency Stop (E-Stop) Test

**Test:** E-stop response time - **Setup:** Robot in motion (50% of max speed) - **Action:** Press E-stop button - **Measurement:** Time from button press to motor stop (oscilloscope) - **Pass Criteria:** <100ms

**Test:** E-stop recovery - **Setup:** E-stop triggered, robot halted - **Action:** Release E-stop, press “Reset”, select “Return Home” - **Expected:** Robot returns to home position safely - **Pass Criteria:** No unintended motion, user acknowledges before resuming

#### 1.9.2.2 7.2.2 Collision Detection Test

**Test:** Detect unexpected contact - **Setup:** Robot moving toward pick position - **Action:** Place foam block in path (simulated collision) - **Expected:** F/T sensor detects force spike (>150N), robot stops - **Pass Criteria:** Stop within 100ms, no damage to robot/object

#### 1.9.2.3 7.2.3 Force Limiting Test (ISO/TS 15066)

**Test:** Verify force limits in collaborative mode - **Setup:** Robot approaches human (mannequin with force sensor) - **Action:** Robot contacts mannequin during motion - **Measurement:** Peak contact force (N) - **Pass Criteria:** Force <150N (ISO/TS 15066 limit for transient contact)

#### 1.9.2.4 7.2.4 Safety Zone Test

**Test:** Human enters safety zone, robot slows/stops - **Setup:** Robot operating at 100% speed, camera detects humans - **Action:** Human enters outer zone (slow zone) - **Expected:** Robot slows to 50% speed - **Action:** Human enters inner zone (stop zone) - **Expected:** Robot stops completely - **Pass Criteria:** Speed reduction smooth, stop <100ms

### 1.9.3 7.3 Safety Certification

**Process:** 1. Conduct all safety tests 2. Document results in safety report 3. Submit to TÜV/UL for certification audit 4. Obtain CE marking (EU) or UL listing (US)

## 1.10 8. Acceptance Testing

### 1.10.1 8.1 Objectives

* Validate system meets customer requirements
* Obtain customer sign-off for deployment
* Basis for contract completion

### 1.10.2 8.2 Acceptance Criteria

| **Criterion** | **Target** | **Measurement Method** | **Pass/Fail** |
| --- | --- | --- | --- |
| Cycle Time | ≤2 sec/object | 100-pick test, average | TBD |
| Throughput | ≥28,000 picks/day | 8-hour run, extrapolate to 24h | TBD |
| Grasp Success Rate | ≥99% | 1000-pick test, count failures | TBD |
| Placement Accuracy | ±0.1mm | CMM measurement (10 placements) | TBD |
| Uptime | ≥99.5% | 1-week continuous run, track downtime | TBD |
| Safety Compliance | ISO 10218, ISO/TS 15066 | Certification audit | TBD |

### 1.10.3 8.3 User Acceptance Test (UAT) Procedure

1. **Preparation (Week 1):**
   * Install system at customer site
   * Calibrate camera-robot transform
   * Load customer objects (train detection model if needed)
2. **Training (Week 1):**
   * Train operators (2 days)
   * Train maintenance staff (1 day)
3. **UAT Execution (Week 2):**
   * Customer runs system for 40 hours (1 week)
   * Customer observes performance, logs issues
   * Project team fixes critical bugs (on-site support)
4. **UAT Sign-Off (End of Week 2):**
   * Customer reviews acceptance criteria table
   * If all “Pass”, customer signs acceptance document
   * If any “Fail”, create punch list, remediate, retest

### 1.10.4 8.4 Acceptance Test Report Template

# Acceptance Test Report  
  
## Test Summary  
- \*\*Date:\*\* [YYYY-MM-DD]  
- \*\*Location:\*\* [Customer Site]  
- \*\*Testers:\*\* [Names]  
  
## Results  
| Criterion | Target | Actual | Pass/Fail | Notes |  
|-----------|--------|--------|-----------|-------|  
| Cycle Time | ≤2 sec | 1.8 sec | ✅ Pass | Average of 100 picks |  
| ... | ... | ... | ... | ... |  
  
## Issues Found  
| Issue ID | Severity | Description | Status |  
|----------|----------|-------------|--------|  
| UAT-001 | High | Camera loses connection after 4 hours | Fixed |  
| UAT-002 | Low | Dashboard slow to load (5 sec) | Open |  
  
## Conclusion  
[Pass / Fail / Conditional Pass]  
  
## Signatures  
- Customer Representative: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_  
- Project Manager: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_

## 1.11 9. Regression Testing

### 1.11.1 9.1 Objectives

* Ensure new changes don’t break existing functionality
* Run after every significant code change or bug fix

### 1.11.2 9.2 Regression Suite

**Composition:** - All unit tests (full suite) - Critical integration tests (vision → planning → control) - 1 end-to-end system test (smoke test: single pick-place)

**Execution:** - Automated (CI/CD pipeline) - Run on every merge to main branch - Takes ~30 minutes (parallelized)

**Pass Criteria:** - 100% of unit tests pass - All critical integration tests pass - Smoke test completes successfully

## 1.12 10. Test Environment & Tools

### 1.12.1 10.1 Test Environments

| **Environment** | **Purpose** | **Hardware** | **Software** |
| --- | --- | --- | --- |
| **Dev** | Unit, integration tests | Developer laptop | Ubuntu 22.04, ROS2, Gazebo |
| **CI/CD** | Automated regression | GitHub Actions runners | Docker containers |
| **Sim** | System tests (simulation) | x86 server (16 cores, 32GB RAM) | Gazebo, RViz2 |
| **Lab** | System tests (real hardware) | Full robot cell (UR5e, camera, NUC) | Production-identical setup |
| **Customer** | Acceptance tests | Customer site | Customer environment |

### 1.12.2 10.2 Test Tools

| **Tool** | **Purpose** | **Language** | **License** |
| --- | --- | --- | --- |
| **pytest** | Unit tests (Python) | Python | MIT |
| **gtest** | Unit tests (C++) | C++ | BSD |
| **coverage.py** | Code coverage (Python) | Python | Apache 2.0 |
| **gcov/lcov** | Code coverage (C++) | C++ | GPL |
| **launch\_testing** | ROS2 integration tests | Python | Apache 2.0 |
| **JMeter** | API load testing | Java | Apache 2.0 |
| **Locust** | API load testing | Python | MIT |
| **cyclictest** | Real-time jitter testing | C | GPL |
| **Gazebo** | Robot simulation | C++ | Apache 2.0 |
| **RViz2** | Visualization, debugging | C++ | BSD |

## 1.13 11. Test Data Management

### 1.13.1 11.1 Test Data Sets

| **Dataset** | **Description** | **Size** | **Location** |
| --- | --- | --- | --- |
| **Synthetic Images** | Rendered cubes, boxes (Blender) | 1000 images | /test\_data/synthetic/ |
| **Real Images** | Lab-captured RGB-D | 500 images | /test\_data/real/ |
| **Edge Cases** | Occlusions, poor lighting | 100 images | /test\_data/edge\_cases/ |
| **Point Clouds** | Pre-captured scenes | 200 PCD files | /test\_data/point\_clouds/ |

### 1.13.2 11.2 Test Data Versioning

* **Tool:** DVC (Data Version Control)
* **Storage:** AWS S3 bucket (or local NAS)
* **Versioning:** Tag datasets with git commit hash

**Example:**

dvc add test\_data/  
git add test\_data.dvc .gitignore  
git commit -m "Add test dataset v1.0"  
git tag test-data-v1.0  
dvc push

## 1.14 12. Defect Management

### 1.14.1 12.1 Defect Lifecycle

[Found] → [Logged] → [Triaged] → [Assigned] → [Fixed] → [Verified] → [Closed]

### 1.14.2 12.2 Defect Severity Levels

| **Severity** | **Definition** | **SLA** | **Example** |
| --- | --- | --- | --- |
| **Critical** | System unusable, safety risk | Fix within 24h | E-stop not working |
| **High** | Major feature broken | Fix within 1 week | Object detection fails |
| **Medium** | Minor feature broken | Fix within 2 weeks | Dashboard slow to load |
| **Low** | Cosmetic, minor annoyance | Fix in next release | Typo in UI |

### 1.14.3 12.3 Defect Tracking Tool

**Tool:** Jira / GitHub Issues **Fields:** - **ID:** BUG-XXX - **Summary:** Short description - **Severity:** Critical / High / Medium / Low - **Priority:** P0 (urgent) to P3 (low) - **Assignee:** Developer responsible - **Status:** Open / In Progress / Resolved / Closed - **Found in Version:** v1.0 - **Fixed in Version:** v1.1

## 1.15 13. Test Metrics & Reporting

### 1.15.1 13.1 Key Metrics

| **Metric** | **Target** | **Current** | **Trend** |
| --- | --- | --- | --- |
| **Code Coverage** | >80% | TBD | TBD |
| **Test Pass Rate** | 100% (all must pass) | TBD | TBD |
| **Defect Density** | <1 bug per 1000 LOC | TBD | TBD |
| **Mean Time to Detect (MTTD)** | <1 day (find bugs fast) | TBD | TBD |
| **Mean Time to Resolve (MTTR)** | <5 days (fix bugs fast) | TBD | TBD |

### 1.15.2 13.2 Test Reports

**Weekly Test Summary:** - Tests executed: 1250 - Tests passed: 1248 (99.8%) - Tests failed: 2 (0.2%) - Integration test: vision → grasp (timeout) - System test: multi-object (1 out of 5 objects missed) - New bugs found: 3 (2 High, 1 Low) - Bugs fixed this week: 5

**Release Test Report:** - All acceptance criteria met: ✅ - Critical bugs: 0 - High bugs: 1 (known issue, workaround documented) - **Recommendation:** Approve for release

## 1.16 14. Test Schedule

| **Phase** | **Duration** | **Start** | **End** | **Deliverables** |
| --- | --- | --- | --- | --- |
| **Unit Tests** | Ongoing (every commit) | Week 7 | Week 22 | Test code, coverage reports |
| **Integration Tests** | 2 weeks | Week 17 | Week 18 | Integration test suite |
| **System Tests (Sim)** | 2 weeks | Week 19 | Week 20 | Test results, bug reports |
| **System Tests (Hardware)** | 2 weeks | Week 21 | Week 22 | Hardware test report |
| **Performance Tests** | 1 week | Week 22 | Week 22 | Performance benchmarks |
| **Safety Tests** | 1 week | Week 22 | Week 22 | Safety certification docs |
| **Acceptance Tests** | 2 weeks | Week 25 | Week 26 | UAT report, sign-off |

## 1.17 15. Roles & Responsibilities

| **Role** | **Responsibilities** |
| --- | --- |
| **QA Lead** | Define test strategy, review test plans, approve releases |
| **Test Engineer** | Write test cases, execute tests, log bugs |
| **Developer** | Write unit tests (code coverage), fix bugs |
| **Automation Engineer** | Build CI/CD pipelines, automate regression tests |
| **Safety Auditor** | Conduct safety tests, obtain certifications |
| **Customer Representative** | Define acceptance criteria, execute UAT, sign-off |

## 1.18 16. Risks & Mitigation

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| Insufficient test coverage (<80%) | Bugs slip to production | Enforce coverage gates in CI/CD, prioritize critical paths |
| Real hardware unavailable for testing | Delays, reliance on simulation | Procure hardware early, use hardware-in-loop (HIL) setup |
| Test data not representative | False confidence | Collaborate with customer, use real production objects |
| Acceptance tests fail at customer site | Project delay, reputation damage | Pre-acceptance testing in lab with customer objects, buffer time |
| Safety certification rejected | Cannot deploy | Engage safety consultants early, pre-audit with TÜV |

## 1.19 17. Appendices

### 1.19.1 Appendix A: Test Case Templates

**Unit Test Template (pytest):**

def test\_object\_detection():  
 """  
 Test that object detector correctly identifies a single cube.  
 """  
 # Arrange  
 detector = ObjectDetector(model\_path="yolov8.onnx")  
 image = load\_test\_image("single\_cube.jpg")  
  
 # Act  
 detections = detector.detect(image)  
  
 # Assert  
 assert len(detections) == 1  
 assert detections[0].class\_name == "cube"  
 assert detections[0].confidence > 0.9

**Integration Test Template (ROS2 launch\_testing):**

import launch\_testing.actions  
import pytest  
  
def generate\_test\_description():  
 return launch.LaunchDescription([  
 launch.actions.Node(package='vision\_pipeline', executable='detector'),  
 launch\_testing.actions.ReadyToTest()  
 ])  
  
def test\_vision\_publishes\_detections(launch\_service, proc\_info, proc\_output):  
 sub = Subscriber('/vision/detections', Detection2DArray)  
 msg = sub.wait\_for\_message(timeout=5.0)  
 assert msg is not None

### 1.19.2 Appendix B: Test Data Samples

* test\_data/synthetic/cube\_001.jpg: Red cube, centered, well-lit
* test\_data/edge\_cases/occluded.jpg: Partially occluded object
* test\_data/point\_clouds/bin\_pile.pcd: 20 objects in random pile

### 1.19.3 Appendix C: References

* [ISO 10218-1:2011 (Robot Safety)](https://www.iso.org/standard/51330.html)
* [ISO/TS 15066:2016 (Collaborative Robots)](https://www.iso.org/standard/62996.html)
* [ROS2 Testing Guide](https://docs.ros.org/en/humble/Tutorials/Intermediate/Testing/Testing-Main.html)

**Document Status:** ✅ Complete **Last Updated:** 2025-10-18 **Next Review:** After Development Phase (Week 22) **Approvals:** Pending QA Lead, Tech Lead Sign-Off