

# Standard Operating Procedure

---

**Document Number:** ROBOT-SYS-001

**Standard Operating Procedure (SOP):** Real-Time Robotics  
Controller Task Scheduler

**Revision:** 1.0

**Last Updated:** [DATE]

---

## Contents

---

1. Introduction
  - 1.1 Purpose
  - 1.2 System Overview
  - 1.3 Regulatory Compliance
2. System Specifications
  - 2.1 Task Definitions and Parameters
  - 2.2 System Timer Configuration
3. Operational Protocols
  - 3.1 Standard Operating Procedures
    - 3.1.1 Task Execution Cycle
    - 3.1.2 Processor Assignment Policy
  - 3.2 Performance Optimization
    - 3.2.1 Overlap Avoidance
    - 3.2.2 Priority Scheduling
4. Emergency Operations
  - 4.1 Task Starvation and Data Staleness
  - 4.2 Recovery From Timeout Skew
5. Maintenance Requirements
  - 5.1 Task Reset Logic
  - 5.2 Priority Conflict Testing

- 6. Quality Assurance
    - 6.1 Execution Integrity Checks
    - 6.2 Latency Verification
  - 7. Security Protocols
    - 7.1 Resource Isolation
    - 7.2 Task Conflict Resolution
  - 8. Environmental Considerations
    - 8.1 Real-Time Synchronization Margin
    - 8.2 Resource Load Handling
  - 9. Training Requirements
    - 9.1 System Familiarity
    - 9.2 Debugging and Profiling
  - 10. Document Control
    - 10.1 Revision History
    - 10.2 Authorization
  - 11. Process Flows and State Transitions
    - 11.1 Task Workflow Summary
    - 11.2 Arbitration and Processor Grant Logic
    - 11.3 Timeout Triggering Logic
- 

# **1. Introduction**

---

## **1.1 Purpose**

---

To manage real-time task execution in a robotics controller by coordinating periodic processes across sensor, motor, and communication modules.

## **1.2 System Overview**

---

- Executes five periodic tasks (T1-T5) with distinct timing and resource needs.
- Schedules tasks by priority and processor availability.

- Uses shared arbitration to ensure fair, conflict-free task execution.

### 1.3 Regulatory Compliance

---

- Complies with embedded robotics system timing requirements.
  - Ensures no starvation and bounded response latency.
- 

## 2. System Specifications

---

### 2.1 Task Definitions and Parameters

---

- **T1 (Motor control)**: Period 40ms; Execution 1ms (pri10), 5ms (pri7)
- **T2 (Sensor read)**: Period 100ms; Execution 10ms (pri4), 5ms (pri8), 5ms (pri4)
- **T3 (Sensor control)**: Offset 50ms; Execution 8ms (pri5), 12ms (pri8)
- **T4 (Comm read)**: Period 200ms; Execution 10ms (pri9), 20ms (pri2), 3ms (pri3)
- **T5 (Comm process)**: Period 400ms; Execution 2ms (pri3), 12ms (pri1), 10ms (pri6)

### 2.2 System Timer Configuration

---

- Timer values used to generate timeouts:
    - `timeout40` , `timeout50` , `timeout100` , `timeout100offset` ,  
`timeout200` , `timeout400`
  - Timers trigger request logic in each module.
-

## 3. Operational Protocols

---

### 3.1 Standard Operating Procedures

---

#### 3.1.1 Task Execution Cycle

- Tasks begin in `state = 0`
- On timeout, `start` flag is activated
- Progresses through multiple execution states until `finish = TRUE`
- Resets to idle state after each full cycle

#### 3.1.2 Processor Assignment Policy

- Each task submits a `request` signal based on its priority
- Central scheduler grants processor access to task with highest priority
- Tie cases (e.g., T2 vs T3) resolved using `last24` flag toggle logic

### 3.2 Performance Optimization

---

#### 3.2.1 Overlap Avoidance

- Task activations blocked if prior activation not yet complete
- `error` condition raised if timeout triggers while task is active

#### 3.2.2 Priority Scheduling

- Grant assigned to task with maximum request value
  - Conflicts resolved with fixed arbitration rules and toggled preference bit
-

## 4. Emergency Operations

---

### 4.1 Task Starvation and Data Staleness

---

- T3 includes `activation_count` tracking number of times it's triggered
- May decline to execute if T2 has not generated sufficient new data

### 4.2 Recovery From Timeout Skew

---

- On invalid activation conditions, task remains idle
- Task resumes execution in next cycle if valid trigger conditions restored

---

## 5. Maintenance Requirements

---

### 5.1 Task Reset Logic

---

- On `finish`, reset internal `state` to 0
- Counters and latches (e.g., `activation_count`, `timeoutlatch`) also reset

### 5.2 Priority Conflict Testing

---

- Validate arbitration fairness with priority collision simulations
- Specifically check interactions between T2/T3 and T4/T5

---

## 6. Quality Assurance

---

### 6.1 Execution Integrity Checks

---

- Assert single task execution per processor at any time

- Check `error` flag during all timeout conditions

## 6.2 Latency Verification

---

- Calculate response time: `MIN[start, finish]` and `MAX[start, finish]`
  - Check how quickly T3 responds after T2 finishes
- 

# 7. Security Protocols

---

## 7.1 Resource Isolation

---

- One processor-granted task at a time
- Scheduler prevents concurrent use of shared processor

## 7.2 Task Conflict Resolution

---

- `last24` toggled based on most recent winner between T2/T3 or T4/T5
  - Ensures both tasks get fair turns over time
- 

# 8. Environmental Considerations

---

## 8.1 Real-Time Synchronization Margin

---

- Comm and sensor operations aligned via timeout offsets
- Avoids sensor/motor control timing mismatches

## 8.2 Resource Load Handling

---

- Cooldown periods implied by timeout intervals and activation counters
  - Prevents overloading processor under bursty input
-

## 9. Training Requirements

---

### 9.1 System Familiarity

---

- Operators must understand timing, state transitions, and arbitration logic
- Maintenance staff must interpret latch/counter behavior

### 9.2 Debugging and Profiling

---

- Review logs from timeout-based activations
- Profile arbitration performance and latency across T1-T5

---

## 10. Document Control

---

### 10.1 Revision History

---

- Rev 1.0 – Initial SOP derived from formal model (robot.txt)

### 10.2 Authorization

---

- Approved by: Embedded Systems Lead
- Reviewed by: Real-Time Safety Board

---

## 11. Process Flows and State Transitions

---

### 11.1 Task Workflow Summary

---

- All tasks follow: `idle → executing → finished → idle`
- Reentry triggered by timer events

### 11.2 Arbitration and Processor Grant Logic

---

- Scheduler compares request values across tasks

- Chooses winner or toggles `last24` to resolve ties

### **11.3 Timeout Triggering Logic**

---

- Timeouts fire at fixed modular intervals (40ms, 50ms, etc.)
- Used to synchronize task startups and control latency margins