Flyby-F11 Autonomy Architecture: Ontology-Constrained Reinforcement Learning

Finley Holt

2025-12-26

Table of contents

1 Flyby-F11 Autonomy Architecture: Ontology-Constrained Reinforcement Learning

Platform: Flyby Robotics F-11 Developer Quadcopter with Jetson Orin NX 16GB **Mission Context**: GPS-denied, communications-limited autonomous missions (MCTSSA collaboration) **Core Innovation**: Hybrid ontology-constrained RL for safe, adaptive, explainable autonomy

---

1.1 Executive Summary

This document defines the architectural approach for autonomous navigation on the Flyby-F11 platform. Based on comprehensive literature review (SYNTHESIS.qmd), we implement a **hybrid architecture** combining:

1. **Formal Ontological Knowledge** - SUMO-based ontology for safety constraints, domain vocabulary, and semantic reasoning
2. **Multi-Agent Reinforcement Learning** - Hierarchical RL agents optimizing within ontology-defined safe action spaces
3. **Automatic Goal Generation Model (AGGM)** - Runtime adaptation to unseen situations through ontological reasoning

This approach addresses critical failures in pure learning systems (37-65% collision rates in UAV-ON benchmark) while maintaining adaptability that pure rule-based systems lack.

1.1.1 Why This Approach?

Evidence from Literature (detailed synthesis): - **Safety**: Ontologies reduce collision rates by constraining RL action spaces before execution - **Adaptability**: AGGM enables runtime goal generation for unseen situations without retraining - **Explainability**: Semantic rule tracing provides audit trails required for defense applications (MCTSSA) - **Efficiency**: Ontological abstractions improve sim-to-real transfer and sample efficiency - **Edge-Compatible**: <100ms reasoning latency achievable on Jetson platforms

---

1.2 Architecture Overview

1.2.1 Single-Reasoner Architecture (Vampire Only)

Architectural Decision (Phase 3 Evaluation, 2024-12-25)

After comprehensive empirical benchmarking (Phase 3 Evaluation Report), we determined that a **single-reasoner architecture using Vampire** is optimal:

Finding	Impact
Vampire ~50ms p95 latency	Acceptable for 20Hz navigation loop
OWL reasoners cannot express safety axioms	ELK/Reasonable rejected (DL limitations)
Prolog 4,700x faster but adds complexity	Rejected to avoid translation/maintenance burden
KIF/SUMO remains single source of truth	0% translation loss

Key Insight: Ontological reasoning belongs in the navigation layer (20Hz), not the flight control layer (400Hz). Real-time safety (<10ms) is handled by the classical control layer, not symbolic reasoning.

1.2.2 Tiered Safety Architecture

```
TIER 1: Classical Control (<1ms)
  PX4/ArduPilot attitude control
  Motor mixing, PID loops
  Sensor filtering, state estimation

TIER 2: Pre-computed Safety (<10ms)
  Obstacle costmaps (baked from depth)
  Geofence boundary polygons
  Velocity limits, acceleration constraints

TIER 3: Tactical Reasoning (~50ms) - VAMPIRE
  "Am I violating a no-fly zone?"
  "Is battery critical for return?"
  "Does this waypoint sequence satisfy constraints?"
  Runs at 20Hz navigation rate

TIER 4: Mission Planning (~100ms-1s) - VAMPIRE
  Route verification and regulatory compliance
  Mission feasibility analysis
  Pre-flight or during mission replanning
```

1.2.3 Unified Compute Architecture

With the single-reasoner decision, there is no mode swapping between planning and execution:

```
             UNIFIED EXECUTION MODE

Vision/Perception      Vampire Reasoning
(~12-14 GB)            (~50-100 MB)
```

```
YOLO11 (TensorRT)       Vampire Theorem Prover
~2-3 GB                 ~14 MB (typical)
- Object detection      - Full FOL reasoning
- 20-50ms latency       - Safety queries: ~50ms
                        - Planning queries: ~100ms
Segmentation Model      - Query caching for repeated
~1-2 GB
- Terrain types         Perception → TPTP Bridge
- Traversability        ~50 MB
                        - Vision facts → TPTP format
VLM (Optional)          - Spatial relations computed
~3-5 GB (7B model)      - Event detection
- Scene understanding
- Semantic grounding    ROS 2 vampire_bridge Package
                        - Query service (100ms timeout)
Depth Processing        - Result caching (LRU + TTL)
~500 MB                 - 20Hz safety monitoring
- Obstacle maps
- Clearance calc

ROS 2 Middleware + System Overhead: ~2 GB
```

Key Advantages: - **No translation needed**: KIF/SUMO is the single source of truth - **Full expressivity**: All safety axioms preserved (0% loss) - **Simplified architecture**: One reasoner for planning AND runtime - **Acceptable latency**: ~50ms fits within 20Hz navigation loop

1.2.4 Three-Level Hierarchy (Execution Mode)

```
LEVEL 1: MISSION PLANNING (10-second horizon)

 Vampire (TPTP)            RL Agent: Mission
 - Mission rules                - Select waypoints
 - Constraints                  - Adapt mission
 - Airspace rules               - Resource planning


                   Waypoint sequence




LEVEL 2: BEHAVIOR SELECTION (1-second horizon)

 Vampire (TPTP)            RL Agent: Behavior
 - Behavior rules               - Navigate
 - Transitions                  - Loiter
 - Safety rules                 - Land
```

```
                        - Avoid

                   Behavior + parameters



 LEVEL 3: TRAJECTORY OPTIMIZATION (100-ms horizon)

   Vampire (TPTP)             RL Agent: Trajectory
   - Actuator limits              - Velocity commands
   - Safety margins               - Obstacle avoid
   - Clearances                   - Energy optimize


                   [vx, vy, vz, yaw]



                     PX4/ArduPilot
                     (Control Loop)
```

1.2.5 Integration Points

Single Reasoner (No Mode Swapping): - Vampire handles both planning and runtime tactical queries - Different query types distinguished by timeout (planning: 5s, tactical: 100ms) - Query caching reduces repeated query overhead (LRU + TTL) - KIF/SUMO remains single source of truth with 0% translation loss

Ontology → RL (Constraint Enforcement): - Filters action spaces: Vampire proves which actions satisfy constraints - Shapes rewards: penalties for ontology-detected violations - Structures state: semantic abstractions from TPTP facts

RL → Ontology (Experience-Based Learning): - Optimizes within safe boundaries defined by ontology - Discovers efficient policies through exploration - Adapts parameters based on environmental feedback

Bidirectional (AGGM Runtime Reasoning): - Forward reasoning: select goals from ontology-defined goal set - Backward reasoning: create new goals to return to known safe states - Importance weighting: prioritize safety-critical observations

---

1.3 Component Details

1.3.1 1. Ontological Knowledge Base

Foundation: SUMO upper-level ontology (design/verification) + SWI-Prolog (runtime inference) **Reasoning Strategy**: Two-phase architecture detailed in ONTOLOGY_FOUNDATION.qmd

1.3.1.1 Planning Phase: SUMO + Heavyweight Reasoners

SUMO Ontology (SUO-KIF format): - Full first-order logic with n-ary relations - ~25,000 terms, ~80,000 axioms - Supports ternary+ relations: `(orientation ?OBJ1 ?OBJ2 ?DIRECTION)` - Used for mission modeling, verification, safety proofs

Reasoning Engines: - **Vampire**: Automated theorem proving for safety property verification - **Clingo**: Answer set programming for optimal path planning - **E-Prover**: Alternative FOL reasoner for constraint checking

Four-Layer Structure:

```
Upper Level: SUMO
- Physical objects (Object, Agent)
- Processes (Motion, Action)
- Relations (orientation, between, during)
- Functions (distance, measure, direction)



Domain Level: IEEE AUR + UAV Extensions
- UAV (extends TransportationDevice)
- FlightPhase (Takeoff, Transit, Landing)
- SpatialRelation (hasSafeSeparation)
- EnvironmentalCondition (Wind, Visibility)



Application Level: Flyby-F11 Missions
- SurveyMission, InspectionMission
- GeofenceBoundary, NoFlyZone
- ISR payloads (Gremsy VIO, RESEPI, Raptor)
- NDAACompliance, FAAPart107Rules



Instance Level: Runtime State
- currentMission: Inspection123
- detectedObstacle: Tree47 at [x,y,z]
- batteryLevel: 68% (12 min remaining)
```

1.3.1.2 Execution Phase: Compiled Prolog Rules

SWI-Prolog Runtime (~50-100 MB): - Compiled from SUMO axioms relevant to mission - Optimized for <10ms query latency - Engine architecture: ~20KB per concurrent query thread - TCMalloc for reduced memory footprint

Example Compiled Rules (Prolog syntax):

```
% Collision avoidance (from SUMO spatial reasoning)
canExecute(moveToward(Pos)) :-
  forall(obstacle(Obs),
         distance(currentPosition, Obs, Dist),
         Dist > safetyMargin).

% Energy management (from SUMO resource ontology)
mustReturnToHome :-
  estimatedEnergyRemaining(Energy),
  energyToReturnHome(Required),
  safetyReserve(Reserve),
  Energy < (Required + Reserve).

% Geofence enforcement (from SUMO spatial containment)
canExecute(Action) :-
  resultingPosition(Action, Pos),
  isWithinGeofence(Pos).

% Flight phase transitions (from SUMO process ontology)
canTransitionTo(landing) :-
  isLandingZone(currentPosition),
  altitude(Alt), Alt < landingInitiationAltitude,
  horizontalVelocity(Vel), Vel < maxLandingVelocity.

% Spatial relations (ternary - preserved from SUMO)
between(Drone, Obj1, Obj2) :-
  position(Drone, [X1, Y1, Z1]),
  position(Obj1, [X2, Y2, Z2]),
  position(Obj2, [X3, Y3, Z3]),
  % Check if Drone is geometrically between Obj1 and Obj2
  is_on_line_segment([X1,Y1,Z1], [X2,Y2,Z2], [X3,Y3,Z3]).
```

SUMO → Prolog Translation Strategy: 1. **Manual translation** of critical safety axioms (reviewed and verified) 2. **Semi-automatic** for common patterns (spatial relations, temporal logic) 3. **Testing**: Equivalence checking between SUMO proofs and Prolog queries 4. **Validation**: Mission scenarios tested in both planning and execution modes

1.3.2 2. Multi-Agent Reinforcement Learning

Paradigm: Hierarchical RL with experience sharing **Integration**: Each agent operates within ontology-constrained action/state spaces

1.3.2.1 Mission Planner Agent (Level 1)

MDP Formulation: - **State** (ontological): - Mission progress: `{waypointsCompleted, currentObjective, remainingObjectives}` - Resources: `{batteryLevel, timeElapsed, payloadStatus}` - Environment: `{weatherCondition, airspaceRestrictions, gpssAvailability}`

- **Action Space** (ontology-filtered):

  - `selectNextWaypoint(waypoint_id)` - only from valid waypoints
  - `adaptMissionPlan(new_sequence)` - only safe alternatives
  - `abortMission(reason)` - when constraints violated

- **Reward Function**:

```
R = w1 * missionCompletion
  + w2 * efficiencyBonus (time, energy)
  + w3 * safetyMargin (distance to constraints)
  - w4 * constraintViolationPenalty
```

- **Algorithm**: SAC (Soft Actor-Critic) for continuous action parameters
- **Training**: NPS computing cluster with mission scenario diversity

1.3.2.2 Behavior Selector Agent (Level 2)

MDP Formulation: - **State** (ontological): - Vehicle: `{altitude, velocity, orientation, flightMode}` - Context: `{currentWaypoint, obstaclesNearby, terrainType}` - Mission: `{activeBehavior, missionPhase}`

- **Action Space** (ontology-validated):
  - `navigate(speed, altitude)` - transit to waypoint
  - `loiter(radius, duration)` - station-keeping
  - `land(descentRate)` - controlled descent
  - `avoid(direction, magnitude)` - collision avoidance maneuver
  - `returnToHome()` - emergency return

- **Reward Function**:

```
R = w1 * behaviorAppropriateness (context match)
  + w2 * smoothTransitions (continuity)
  + w3 * constraintSatisfaction (safety)
  - w4 * behaviorSwitchPenalty (stability)
```

- **Algorithm**: PPO (Proximal Policy Optimization) for stable learning
- **Training**: Behavior trees provide structured exploration

1.3.2.3 Trajectory Optimizer Agent (Level 3)

MDP Formulation: - **State** (sensor-based): - Observations: `{EKF_pose, ISR_payload_data, YOLO_detections, MAVLink_telemetry}` - Dynamics: `{position, velocity, acceleration, attitude}` - Setpoints: `{targetPosition, targetVelocity, targetHeading}`

- **Action Space** (actuator-constrained):
  - Velocity command: `[vx, vy, vz, yaw_rate]` within $\pm$2.5 m/s limits

- **Reward Function**:

```
R = w1 * setpointTracking (error minimization)
  + w2 * trajectorySmooth (jerk minimization)
  + w3 * energyEfficiency (acceleration cost)
```

```
    + w4 * obstacleClearance (safety margin)
    - w5 * collisionPenalty
```

- **Algorithm**: TD3 (Twin Delayed DDPG) for low-level control
- **Training**: Continuous in simulation with domain randomization

1.3.3 3. Automatic Goal Generation Model (AGGM)

Purpose: Adapt to unseen situations without retraining **Source**: Literature review paper 02 (Ghanadbashi & Golpayegani, 2022)

1.3.3.1 Six-Stage Process

```
Stage 1: OBSERVE
- Multi-sensor fusion → ontological concepts
- Ontology schema: L^t = {C^t, M^t} (concepts, relations)
```

```
Stage 2: EVALUATE
- Q-value from RL policy
- State distance: |s^t - s^(t-1)|
- Importance weighting: iw_c(x) for each concept
```

```
Stage 3: IDENTIFY SIGNIFICANT CHANGE (triggers)
Case 1: Q-value discrepancy (unexpected reward)
Case 2: State distance threshold (environment change)
Case 3: High-importance concept detected (safety)
```

```
Stage 4: REASON (forward/backward)
- Forward: infer goal from predefined goal-set via SWRL
- Backward: create new goal to reach known safe state
- Priority: F(B, J) balances task vs. state-similarity
```

```
Stage 5: GENERATE ACTION
- Ontology-constrained action space (only valid actions)
- RL policy selects action within constraints
- Multi-agent coordination if needed
```

```
Stage 6: EXECUTE
- Translate to MAVSDK/MQTT commands
- Monitor execution through telemetry
- Update belief state for next cycle
```

Example Scenario: Unexpected obstacle during transit

1. **Observe**: ISR payload detects unknown object at 8m distance in flight path
2. **Evaluate**:
   - Q-value drops (expected reward for "move forward" now low)
   - State distance increases (new obstacle concept added)
   - High importance weight (safety-critical observation)
3. **Identify Change**: Case 3 triggered (high-importance safety concept)
4. **Reason**:
   - Forward: No predefined goal matches "unexpected obstacle in path"
   - Backward: Create new goal "reach clear airspace" (return to obstacle-free state)
   - Priority: Safety (J) > task completion (B)
5. **Generate Action**: RL policy selects `avoid(left, 5m)` from ontology-validated actions
6. **Execute**: Send velocity command to PX4, monitor clearance until safe

1.3.4 4. Safety Constraint Enforcement

Critical Requirement: Collision avoidance, geofence compliance, energy management **Method**: Ontology-based action filtering + runtime monitoring

1.3.4.1 Pre-Flight Safety Checks

```
# Ontology evaluates readiness before takeoff
canTakeOff ←
  batteryLevel > minimumForMission
  hasValidLocalization    # GPS or EKF-based position estimate
  ¬hasActiveWarning
  isWithinAuthorizedAirspace
  weatherCondition == Acceptable
```

1.3.4.2 In-Flight Safety Monitoring

```
# Continuous evaluation at 10 Hz
every 100ms:
  # Collision avoidance
  for obstacle in detectedObstacles:
    if distance(uav, obstacle) < safetyMargin:
      trigger_emergency_avoidance(obstacle)

  # Geofence enforcement
  if not isWithinGeofence(current_position):
    trigger_return_to_geofence()

  # Energy management
```

```
if batteryLevel < criticalThreshold:
  trigger_immediate_landing()
if estimatedEnergyRemaining < energyToReturnHome + reserve:
  trigger_return_to_home()

# Sensor health / localization
if not hasValidLocalization:
  trigger_emergency_land()  # lost position estimate
```

1.3.4.3 Action Filtering Workflow

```
RL Policy outputs action → Ontology Reasoner
                              ↓
                    Evaluate canExecute(action)
                              ↓


                  Valid              Invalid

                             Query: suggestAlternative(action)

                    ↓                     ↓
              Execute action      Try alternative
                                   (or safe fallback)
```

1.3.5 5. Perception-to-Reasoning Bridge

Challenge: Vision models output sub-symbolic representations (bounding boxes, masks, embeddings), but reasoning engines need symbolic facts (relations, concepts, predicates) **Solution**: Symbolic abstraction layer that grounds perceptions to ontological concepts

1.3.5.1 Architecture: From Sensors to Symbols

```
                    RAW SENSORS

  ISR Payload     LiDAR           IMU            GPS
 (VIO/Raptor)    (RESEPI)       (Onboard)      (Optional)


      v              v              v                 v

              SUB-SYMBOLIC PROCESSING

  YOLO11       Point Cloud     EKF State      Position
  (objects)     (terrain)      (pose)         Estimation
  TensorRT      Processing     ArduPilot      MAVLink
  20-50ms       50-100ms       400 Hz         10 Hz
```

```
                    v

     SYMBOLIC ABSTRACTION LAYER (ROS 2 Grounding Nodes)

  ObjectGroundingNode:
    Input:  DetectionArray (YOLO bboxes)
    Output: objectType(obj_123, 'person').
            position(obj_123, [x, y, z]).
            confidence(obj_123, 0.92).
            inRegion(obj_123, zone_A).

  TerrainGroundingNode:
    Input:  Segmentation masks
    Output: terrainType(region_5, 'water').
            traversable(region_5, false).
            terrainSlope(region_5, 15.3).

  SpatialRelationGroundingNode:
    Input:  Depth map + object positions
    Output: distance(drone, obj_123, 5.2).
            between(drone, obj_45, obj_67).
            northOf(obj_123, waypoint_A).
            clearance(forward, 8.5).

  EventDetectionNode:
    Input:  Object tracking history
    Output: enters(obj_123, no_fly_zone).
            loitering(obj_45, duration(30)).
            violates(mission, constraint_7).

  VLMGroundingNode (Optional):
    Input:  RGB frame + depth
    Prompt: "Describe objects and spatial relations as
             Prolog facts using SUMO ontology"
    Output: Structured facts for complex scene understanding


                             v

             PROLOG KNOWLEDGE BASE (SWI-Prolog)

  % Dynamic facts (asserted by perception grounding nodes)
  objectType(obj_123, person).
```

```
position(obj_123, [45.2, -122.1, 100]).
distance(drone, obj_123, 5.2).
terrainType(region_5, water).
clearance(forward, 8.5).

% Static rules (compiled from planning phase)
safeToFlyOver(Region) :-
    terrainType(Region, Type),
    not(Type = water),
    not(Type = urban).

mustAvoid(Object) :-
    objectType(Object, person),
    distance(drone, Object, Dist),
    Dist < 50.  % meters

hasSafeClearance(Direction) :-
    clearance(Direction, Dist),
    Dist > 3.0.  % minimum safety margin

violatesSafetyConstraint(Constraint) :-
    mustAvoid(Object),
    distance(drone, Object, Dist),
    Dist < 10.  % critical threshold


                             v

         RL AGENTS + BEHAVIOR TREE EXECUTOR

Query examples:
- safeToFlyOver(current_region)? → bool
- mustAvoid(X)? → [obj_123, obj_45]
- violatesSafetyConstraint(C)? → abort/replan
- hasSafeClearance(forward)? → bool

Action selection constrained by Prolog query results
```

1.3.5.2 Why This Multi-Layer Approach?

Vision Models Alone Cannot: 1. **Reason relationally**: YOLO says "person at (x,y)", not "person between drone and target" 2. **Check constraints**: Can't encode "must avoid urban areas when communications-denied" 3. **Apply temporal logic**: Can't reason "if in no-fly zone for >5 seconds, abort mission" 4. **Understand mission context**: Don't know what "safe", "compliant", or "appropriate" means

The Symbolic Layer Provides: 1. **Semantic grounding**: Maps pixels → concepts from SUMO ontology 2. **Spatial relations**: Computes n-ary relations (between, northOf, inside) 3. **Constraint evaluation**: Checks compiled mission rules against perceived world state 4. **Event recognition**: Detects complex events (entering zones, loitering, violations) 5. **Explainability**: Every decision traceable to symbolic facts and rules

1.3.5.3 Multi-Sensor Fusion Example

Scenario: Approaching person during autonomous navigation

```
EKF Pose:           Thermal/RGB:        YOLO11:
Velocity            Range at            Detection
[2.0, 0, 0]       bearing 0°:       class: person
m/s forward         5.2 meters          conf: 0.92
                                        bbox: [x,y,w,h]


                          v

                 Grounding Nodes (ROS 2)

                   ObjectGroundingNode:
                   objectType(obj_123,
                              person).
                   position(obj_123,
                           [5.2, 0, 0]).

                   SpatialRelationNode:
                   distance(drone,
                           obj_123, 5.2).
                   timeToContact(obj_123,
                                2.6).



                           v

                 Prolog KB Query

                 ?- mustAvoid(X).
                 X = obj_123.

                 ?- violatesSafetyConstraint
                    (C).
                 C = proximity_alert.


                           v
```

```
AGGM Triggered
Case 3: High-importance
        safety concept

Backward reasoning:
→ Goal: reach safe distance
→ Priority: Safety (J) > B


            v

RL Policy (Constrained)

Query Prolog for valid
actions:
- Forward: INVALID (collision)
- Stop: VALID
- Avoid left: VALID
- Avoid right: VALID

Select: avoid(left, 5m)


            v

Execute Action
[vx=0, vy=2.0, vz=0]

Monitor: distance(drone,
         obj_123, Dist)
Until: Dist > 10
```

1.4 Implementation Roadmap

Detailed Plan: See SYNTHESIS.qmd - Recommended Next Steps

1.4.1 Phase 1: Ontology Foundation & Two-Phase Architecture (Weeks 1-6)

1.4.1.1 Part A: Planning Mode Infrastructure (Weeks 1-3)

- Create Podman container with SUMO ontology (SUO-KIF format), Vampire, Clingo, E-Prover
- All dependencies fully documented in `Containerfile.planning`
- Develop UAV domain ontology in SUMO (flight phases, spatial relations, sensors)
- Create Flyby-F11 application ontology (missions, constraints, NDAA compliance)
- Implement safety axioms in SUO-KIF, verify with Vampire

Deliverables: - `/flyby-f11/ontology/Containerfile.planning` (self-contained planning environment) - `/flyby-f11/ontology/planning_mode/sumo_base.kif` - `/flyby-f11/ontology/planning_mode/uav_domain.kif` - `/flyby-f11/ontology/planning_mode/flyby_mission.kif` - `/flyby-f11/ontology/planning_mode/safety_axioms.kif` - Verification scripts (Vampire proofs for safety properties)

1.4.1.2 Part B: Execution Mode Infrastructure (Weeks 4-6)

- Create Podman container with SWI-Prolog (ARM build for Jetson compatibility)
- All dependencies documented in `Containerfile.execution`
- Develop SUMO → Prolog translation tools (manual + semi-automatic)
- Compile critical safety axioms to Prolog rules
- Benchmark Prolog inference latency and memory footprint (both x86 dev + ARM Jetson)

Deliverables: - `/flyby-f11/ontology/Containerfile.execution` (self-contained execution environment) - `/flyby-f11/ontology/execution_mode/compiled_rules.pl` - `/flyby-f11/scripts/sumo_to_prolog_translator.py` - Translation validation tests (SUMO proofs Prolog queries) - Performance benchmarks (query latency <10ms, memory <100MB)

1.4.2 Phase 2: Perception-to-Reasoning Bridge (Weeks 7-10)

1.4.2.1 Part A: Symbolic Abstraction Layer (Weeks 7-8)

- Create Podman container with ROS 2 Humble + GPU passthrough
- All ROS 2 and perception dependencies in `Containerfile.ros2`
- Create ROS 2 grounding nodes package
- Implement ObjectGroundingNode (YOLO → Prolog facts)
- Implement TerrainGroundingNode (Segmentation → traversability facts)
- Implement SpatialRelationGroundingNode (depth + position → n-ary relations)
- Implement EventDetectionNode (tracking → temporal events)

Deliverables: - `flyby_f11_ros2_ws/Containerfile.ros2` (ROS 2 + GPU passthrough) - `flyby_f11_ros2_ws/Containerfile.vision` (TensorRT + vision models) - `flyby_f11_ros2_ws/src/perception` - Unit tests for each grounding node - Integration tests (vision models → Prolog assertions)

1.4.2.2 Part B: Phase Transition Manager (Weeks 9-10)

- Implement mission planner node (runs heavyweight reasoners in planning container)
- Implement phase transition controller (container orchestration: planning → execution)
- Develop memory profiling and monitoring tools
- Test full planning → execution workflow with container switching

Deliverables: - `flyby_f11_ros2_ws/src/mission_planner/` - `flyby_f11_ros2_ws/src/phase_transition_ma` - `podman-compose.yml` (orchestrates planning + execution containers) - Memory allocation tests (16GB budget validation) - Latency benchmarks (phase transition time)

1.4.3 Phase 3: Multi-Agent RL (Weeks 9-14)

- Define MDPs for mission planner, behavior selector, trajectory optimizer
- Implement AGGM (6-stage process) for each agent
- Develop training infrastructure (Gymnasium environments, reward shaping)

- Train policies with experience sharing (NPS cluster)

Deliverables: - `flyby_f11_ros2_ws/src/ontology_rl/` - Trained policies (mission/behavior/trajectory agents) - Ablation study results (with/without ontology constraints)

1.4.4 Phase 4: Benchmark Evaluation (Weeks 15-18)

- Develop scenarios (waypoint nav, obstacle avoid, GPS-denied, mission adapt)
- Implement baselines (pure RL, pure rules, LLM-based AOA)
- Collect metrics (SR, OSR, DTS, SPL, collision rate, efficiency)
- Statistical analysis (significance testing, Pareto frontiers)

Deliverables: - Benchmark suite with evaluation scripts - Comparative results (tables, plots, failure analysis) - Technical report / conference paper draft

1.4.5 Phase 5: Hardware Validation (Weeks 19-24)

- **Project-drone testing** (Weeks 19-21): Indoor waypoint nav, obstacle avoid, GPS-denied
- **Flyby-F11 integration** (Weeks 22-23): Outdoor survey/inspection missions
- **Stress testing** (Week 24): Sensor failures, environmental challenges, edge cases
- Iterative refinement based on flight logs

Deliverables: - Flight test videos with ontology decision overlays - Hardware performance benchmarks (Jetson utilization) - Safety assessment report (collision-free hours)

1.4.6 Phase 6: Documentation (Weeks 25-26)

- System architecture documentation
- Research paper for ICRA/IROS/RSS
- Open-source repository release
- MCTSSA demonstration package

1.5 Key Innovations and Contributions

1.5.1 1. Two-Phase Compute Architecture for Edge Ontological Reasoning

Challenge: Heavyweight ontological reasoners (Vampire, full SUMO) require >8GB RAM and multi-second inference times, incompatible with real-time UAV control and concurrent vision processing on 16GB unified memory **Solution**: Novel two-phase architecture separating planning and execution - **Planning Mode**: 100% compute for heavyweight reasoning (SUMO + Vampire/Clingo) during mission receipt/replanning - **Execution Mode**: Compiled Prolog rules (~100MB) coexist with vision models (~12GB) for real-time inference - **Phase Transition**: Automatic model swapping via memory manager (unload reasoners → load YOLO/segmentation) - **Performance**: <10ms Prolog query latency, supports concurrent 10Hz reasoning + 20Hz vision processing

Expected Impact: - First demonstration of full SUMO ontology verification + real-time execution on edge hardware - Enables rigorous safety proofs (Vampire) without sacrificing runtime performance - Memory-constrained platforms can now run both complex reasoning and modern vision models

1.5.2 2. Perception-to-Reasoning Bridge for Symbolic Grounding

Challenge: Vision models output sub-symbolic representations (bounding boxes, embeddings) while ontological reasoners require symbolic facts (predicates, relations) **Solution**: ROS 2 grounding nodes that translate perceptions to ontology-aligned Prolog facts - ObjectGroundingNode: YOLO detections → `objectType(obj_123, person)` - SpatialRelationGroundingNode: Depth maps → n-ary relations `between(drone, obj1, obj2)` - EventDetectionNode: Tracking history → temporal predicates `enters(obj, zone)` - VLMGroundingNode (optional): Scene understanding → structured symbolic facts

Expected Impact: - Closes the "semantic gap" between perception and reasoning - Enables SUMO's rich spatial reasoning (ternary+ relations) from sensor data - Provides explainability: every decision traces through symbolic facts to axioms

1.5.3 3. Real-Time Ontological Reasoning on Edge Hardware

Challenge: Traditional ontology reasoners require server-class compute **Solution**: Hybrid SUMO (planning) + SWI-Prolog (execution) approach - SUMO: Full expressivity for verification (n-ary relations, FOL) - Prolog: Lightweight runtime (<100MB, <10ms queries) - Translation validation: Equivalence testing between SUMO proofs and Prolog - Target: 10 Hz reasoning loop concurrent with 20 Hz vision processing

Expected Impact: First demonstration of SUMO-grade reasoning on ARM edge platform for UAV control

1.5.4 4. AGGM for GPS-Denied Navigation

Challenge: Pure RL fails in unseen situations (UAV-ON: 7.30% success rate) **Solution**: Backward reasoning to create goals returning to known safe states - Ontology defines state similarity metrics - Priority function balances safety (J) vs. task completion (B) - Runtime goal generation without retraining

Expected Impact: >50% success rate in novel environments (vs. 7.30% baseline), <10% collision rate (vs. 37-65% baseline)

1.5.5 5. Explainable Autonomy for Defense Applications

Challenge: Black-box RL policies unacceptable for MCTSSA collaboration **Solution**: Semantic decision trace through ontological reasoning - Every action justified by SWRL rule firing - Safety constraints auditable and verifiable - Mission-intent interpretation transparent to operators

Expected Impact: First explainable autonomous UAV system meeting NDAA compliance and defense safety standards

1.5.6 6. Hierarchical Multi-Agent RL with Ontological Structure

Challenge: Monolithic policies don't scale to complex missions **Solution**: Specialized agents at each hierarchy level (mission/behavior/trajectory) - Experience sharing via shared replay buffer - Policy distillation from expert to novice agents - Meta-learning across mission types

Expected Impact: Improved sample efficiency (faster training convergence), modular extensibility (add new missions/behaviors without full retraining)

1.6 Success Criteria

1.6.1 Simulation Benchmarks (Phase 4)

- ☐ Success Rate (SR) > 50% in novel environments (UAV-ON-style)
- ☐ Collision Rate < 10% (vs. 37-65% for unconstrained baselines)
- ☐ Safety Constraint Violations = 0 (hard requirement)
- ☐ Explainability: 100% of actions traceable to ontology rules

1.6.2 Hardware Performance (Phase 5)

- ☐ Ontology reasoning latency < 100ms at 10 Hz
- ☐ Jetson Orin NX resource utilization < 70% (CPU/GPU/memory)
- ☐ Real-time control loop: 10 Hz for mission, 100 Hz for trajectory
- ☐ Successful GPS-denied navigation for >5 minutes using EKF-only localization

1.6.3 Flight Testing (Phase 5)

- ☐ 10+ successful autonomous missions (waypoint navigation)
- ☐ 5+ successful obstacle avoidance scenarios (dynamic obstacles)
- ☐ 3+ successful mission adaptations (unexpected events)
- ☐ Zero collisions in controlled testing environment
- ☐ MCTSSA demonstration: communications-denied mission completion

1.6.4 Research Impact (Phase 6)

- ☐ Conference paper accepted (ICRA/IROS/RSS)
- ☐ Open-source release with 100+ GitHub stars
- ☐ Contribution to UAV-ON benchmark (ontology-constrained baseline)
- ☐ MCTSSA collaboration continuation (follow-on projects)

1.7 References

Detailed Literature Review: SYNTHESIS.qmd

Core Papers: 1. Hare & Tang (2024) - Multi-agent ontology-driven RL for personalized systems 2. Ghanadbashi & Golpayegani (2022) - AGGM for unseen situations (traffic control) 3. Aguado et al. (2024) - Survey of ontology-enabled robot dependability 4. UAV Collision Avoidance Ontologies - Domain application review 5. Xiao et al. (2025) - UAV-ON benchmark for object-goal navigation 6. Hu et al. (2025) - Survey of hybrid decision-making for autonomous vehicles

Platform Details: SYSTEM_CONSTRAINTS.qmd

Ontology Specification: ONTOLOGY_FOUNDATION.qmd

1.8 Appendix: Alignment with Project-Drone

Development Strategy: Shared autonomy components developed on accessible hardware

```
project-drone (Development Platform)
  Hardware: Jetson Orin Nano Super 8GB (67 TOPS)
  Sensors: T265 visual odometry, D455 depth camera
  Purpose: Algorithm development, rapid prototyping
  Shared packages:
      autonomy_core/ (mission planning, waypoint navigation)
      behavior_trees/ (BT mission logic, ontology-aware nodes)
      perception_pipeline/ (vision models, semantic fusion)
      px4_interface/ (MAVSDK bridge, flight abstraction)

                   ↓ (symlinks when ready)

flyby-f11 (Deployment Platform)
  Hardware: Jetson Orin NX 16GB (50 TOPS, 2x memory)
  ISR Payloads: Gremsy VIO | RESEPI LiDAR | NextVision Raptor
  Purpose: Mission-specific deployment (MCTSSA)
  Packages:
      flyby_f11_bringup/ (mission launch configurations)
      flyby_f11_sensors/ (platform-specific drivers)
      flyby_f11_mission/ (MCTSSA mission logic)
      ontology_interface/ (reasoner, perception mapper, action validator)
```

Benefit: Ontology-constrained RL developed incrementally on project-drone, then deployed to flyby-f11 when hardware available. Shared packages ensure consistency while platform-specific packages handle hardware differences.

Document Version: 2.0 **Last Updated**: 2024-12-26 **Status**: Architecture updated based on Phase 3 evaluation. Phases 1-3 complete, Phase 4+ updated for single-reasoner (Vampire) architecture.

Changelog: - v2.0 (2024-12-26): Updated to single-reasoner architecture (Vampire only) based on Phase 3 evaluation - v1.0 (2024-12-25): Initial architecture with two-phase compute (SUMO+Vampire planning, SWI-Prolog execution)