Spring Test Plan

PatrolKnight

AI-Augmented Patrol Scheduler for Autonomous Security Robots

Team E

Author: **Hongyi (Sam) Dong**
Andrew ID: samd
samd@andrew.cmu.edu

Author: **Po-Ting (Koko) Ko**
Andrew ID: potingk
potingk@andrew.cmu.edu

Author: **Chenxi (Leo) Xu**
Andrew ID: chenxix2
chenxix2@andrew.cmu.edu

Author: **Pin-Wen (Maddie) Wang**
Andrew ID: pinwenw
pinwenw@andrew.cmu.edu

Author: **Tzu-Cheng (Joshua) Tsai**
Andrew ID: joshuat3
joshuat3@andrew.cmu.edu

Sponsor: **Knightscope**

February 2026

Carnegie Mellon University
Robotics Institute

KNIGHTSCOPE

Contents

1 Introduction

This document outlines the strategic plan for the PatrolKnight capstone project in preparation for the Spring Validation Demo (SVD) in April. It details the testing protocols essential for ensuring a successful demonstration and high-quality delivery. To maintain development momentum, each Progress Review (PR) is treated as a critical milestone, ensuring the incremental completion of tasks across core subsystems, including path-planning, perception, SLAM, behavior control, visual servo, perception, and UI/UX design.

2 Logistics

2.1 Equipment

We will need three robots for the whole project, F1tenth, low speed vehicle (LSV) provided by KnightScope and will be sent to CMU, and K7 robot which we need to fly to California for testing and Demo. All the demo in CMU will be preformed via LSV, which has the same devices (see appendix B) as K7 robot.

2.2 Location

The work and testing on this project can mostly be done in AI Maker Space in Tepper.

3 Schedule

Table 1: Spring Semester Demonstration and Milestone Verification Plan

Event	Date	Milestones	Tests	Requirements
PR1	02/13	• High-fidelity simulation of autonomous trajectory tracking. • Path planning module generates multi-waypoint global paths. • Mapping module produces collision-free occupancy grid costmaps.	T.P.01 T.P.02 T.P.03 T.P.04	M.F.1.1 M.F.7 M.F.6
PR2	02/27	• System integration on F1Tenth platform (Mapping, Planning, and Pure Pursuit). • Validation of F1Tenth kinematics within the simulation environment. • Implementation of toggleable operational modes (e.g., Standard vs. Alert/Surveillance).	T.P.05 T.P.06 T.P.07	M.P.1.2 M.F.4
PR3	03/20	• Real-time object detection and classification for anomalous entities/hazards. • Visual Servoing utilizing PID control for target centering and tracking. • Full software stack integration (excluding high-level detection inference). • Initial simulation testing on the LSV (Low-Speed Vehicle) platform.	T.P.08 T.P.09 T.P.10 T.P.11	M.P.5.1
PR4	04/03	• Integration of trained object detection models into the primary software pipeline. • Full hardware-in-the-loop (HIL) integration for the LSV platform.	T.P.12	M.F.2
SVD	04/17	• UI/UX validation: Real-time visualization of waypoint selection and path computation. • Successful autonomous navigation of the LSV along calculated trajectories.	T.P.13	M.P.6.2
SVD Encore	04/24	Same as SVD.		

4 Tests

T.P.01 - Pure Pursuit Unit Test	
Objectives	
To validate the functional correctness and numerical stability of the `pure_pursuit_node` in isolation. The test ensures correct path subscription, look-ahead point selection, steering computation, and bounded output behavior under controlled simulated inputs.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), ROS 2 CLI (for topic publishing and monitoring)
Elements	Local Planning Subsystem (`pure_pursuit_node`)
Personnel	Sam Dong
Location	AI Maker Space
Procedure	
1. Launch the `pure_pursuit_node` in a standalone terminal. 2. Verify successful startup and confirm parameter logging (look-ahead distance, wheelbase, update rate, maximum steering angle). 3. Publish a test `nav_msgs/Path` message to `/path` representing a known trajectory (e.g., straight line or constant-radius arc). 4. Publish a valid `geometry_msgs/PoseStamped` to `/state` with consistent position and orientation. 5. Monitor the output steering command on `/steering` (type: `std_msgs/Float32`) using `ros2 topic echo`. 6. Vary vehicle pose slightly (lateral offset and heading error) and observe corresponding steering corrections.	
Verification Criteria	
- Node subscribes to `/path` and `/state` without runtime errors. - If `/state` is received before `/path`, the node logs a warning (e.g., "No path received yet") and does not publish control output. - After both topics are available, the node publishes steering commands upon each new `/state` update. - For a straight path with vehicle aligned to the centerline, the steering command is approximately 0°. - Steering output is bounded by `max_steer_deg`. - Steering response is continuous and stable under small pose perturbations (no oscillatory or discontinuous behavior).	

Table 2: Pure Pursuit Unit Test

T.P.02 - Path Planning Unit Test	
Objectives	
To validate the functional correctness and algorithmic behavior of the `path_planning` node in isolation. The test verifies correct subscription to localization and waypoint topics, successful trajectory computation (Dijkstra / TSP / Hybrid A*), and proper publication of a feasible `nav_msgs/Path` message prior to full system integration.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), Rviz2 (for visualization), ROS 2 CLI (for topic publishing)
Elements	`path_planning` node, Mock Data Publishers (Costmap, Odometry, Waypoints)
Personnel	Leo & Maddie
Location	AI Maker Space
Procedure	
1. Launch the `path_planning` node in a standalone terminal. 2. Publish a dummy occupancy grid to `/pf/cost_map` (type: `nav_msgs/OccupancyGrid`). 3. Publish a valid start pose to `/pf/pose/odom` (type: `nav_msgs/Odometry`). 4. Publish a target `geometry_msgs/PoseArray` to `/interface/waypoints` via CLI or test script. 5. Monitor the output `/planning/path` topic using `ros2 topic echo`. 6. Visualize the costmap, start pose, waypoints, and planned path in Rviz2. 7. Repeat with multiple waypoint configurations (single goal, multi-goal sequence, obstacle-constrained case).	
Verification Criteria	
- Node subscribes to `/interface/waypoints`, `/pf/cost_map`, and `/pf/pose/odom` without runtime errors. - Upon receiving waypoints, the node logs a planning status message (e.g., "Planning Started"). - A valid `nav_msgs/Path` message is published to `/planning/path`. - The generated path is collision-free and consistent with the provided costmap constraints. - For multi-waypoint missions, the path traverses all user-defined waypoints in correct sequence. - Path length is consistent with expected optimal behavior (shortest feasible path under map constraints). - No TF errors, frame mismatches, or disconnected topics occur during execution.	

Table 3: Path Planning Node Unit Test

T.P.03 - Mapping Unit Test	
Objectives	
Implement a collision-free map converter that generates an occupancy grid cost map to avoid possible obstacle collision in the mission planning process. It also labels the map grid outside the drivable area for the user interface so that waypoints are added correctly.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), ROS 2 CLI (for topic publishing)
Elements	Nav2 module, GIMP map editing tool
Personnel	Joshua Tsai
Location	AI Maker Space
Procedure	
1. For the input binary map, use GIMP map editing tool to add, remove, or adjust map points. 2. Based on the vehicle configuration and safe margin, determine `robot_radius` and `inflation_radius` parameters. 3. Launch the `nav2_bringup` module in a standalone terminal. 4. Confirm the node starts without errors and logs its configured parameters. 5. Publish a pseudo transformation between map, odom, and base_link. 6. Examine the output cost map topic `/global_costmap` (type: `nav_msgs/OccupancyGrid`).	
Verification Criteria	
- Run `convert_map` script and obtain the correct map files. - Tune parameters and obtain the desired cost map output. - Labels non-drivable grids as specific values. - Run `show_map` script to visualize both the input and output occupancy grid maps. - Edit the existing map to obtain a static map without unreliable points.	

Table 4: Mapping Node Unit Test

T.P.04 - User Interface Node Unit Test	
Objectives	
To verify the `interface_bridge` and frontend application correctly subscribe to ROS2 map and localization topics, render robot state in real time, allow waypoint editing (add/drag/delete), and publish mission waypoints back to ROS 2 in isolation before full system integration.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), Web Browser (Chrome/Firefox), ROS2 Commnad line, Rviz2 (optional cross-check)
Elements	`interface_bridge` node (ROS2 Topic and WebSocket conversion), Frontend React Application, Mock Publishers (`/map`, `/pf/pose/odom`, `/planning/path`)
Personnel	Po-Ting Ko
Location	AI Maker Space
Procedure	
1. Start the frontend application and open it in a web browser. 2. Launch the `interface_bridge` node in a standalone terminal. 3. Publish a dummy occupancy grid to `/map` (type: `nav_msgs/OccupancyGrid`). 4. Publish a dummy robot pose to `/pf/pose/odom` (type: `nav_msgs/Odometry`). 5. Verify the map and robot pose are rendered correctly on the interface. 6. Add, drag, and delete waypoints directly on the map canvas. 7. Click the "Publish Mission" button to send waypoints as a `geometry_msgs/PoseArray` type to publish `/interface/waypoints` topic. 8. Monitor the published topic with mock planner node. Connect all published waypoints and publish as `/planning/path` topic. 9. Visualize the waypoints and mission path on user interface. 10. Optionally visualize waypoints in Rviz2 for cross-validation.	
Verification Criteria	
- Web interface successfully connects to `interface_bridge` via WebSocket. - Map is rendered with correct resolution, origin, and orientation. - Robot pose updates in real time with correct frame alignment. - Waypoint editing (add/drag/delete) behaves deterministically without UI crash. - Clicking "Publish Mission" results in a valid `PoseArray` message. - Waypoints in ROS 2 match exactly the positions defined in the UI (no frame mismatch). - No dropped WebSocket connections or ROS topic errors during operation.	

Table 5: User Interface Node Unit Test

T.P.05 - F1Tenth Simulation Integration Test	
Objectives	
To verify full-system integration of the F1Tenth simulation stack with the developed path planning, pure pursuit, and user interface modules. The goal is to confirm end-to-end mission execution: waypoint input → path generation → trajectory tracking → simulated motion.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), F1Tenth Simulation Package, Rviz2, Web Browser, ROS 2 CLI
Elements	F1Tenth Simulation Stack (Map Server, Particle Filter, Kinematics), `path_planning` node, `pure_pursuit` node, `interface_bridge` node, Frontend UI Application
Personnel	Po-Ting Ko
Location	AI Maker Space
Procedure	
1. Install and configure all required F1Tenth simulation dependencies. 2. Launch the F1Tenth system stack (map server, particle filter, kinematics). 3. Launch the simulation environment and visualize the map and vehicle model in Rviz2. 4. Launch the `path_planning`, `pure_pursuit`, and `interface_bridge` nodes. 5. Start the frontend user interface in a web browser. 6. Verify that the UI correctly renders the map and current robot pose. 7. Add mission waypoints using the UI and publish them to `/interface/waypoints`. 8. Confirm that `path_planning` subscribes and publishes a valid `/planning/path` topic. 9. Confirm that `pure_pursuit` subscribes to robot pose and path, and publishes control commands to the vehicle topic. 10. Observe the simulated vehicle following the planned trajectory in Rviz2. 11. Verify that the executed trajectory and planned path are visualized in the UI.	
Verification Criteria	
- All nodes launch without missing dependencies or TF errors. - Rviz2 correctly displays map, vehicle model, and trajectory markers. - UI map and robot pose are consistent with Rviz2 visualization (frame alignment validated). - Waypoints published from UI are correctly received by planner. - Planner generates a feasible path that avoids obstacles. - Pure pursuit generates smooth control commands without oscillation or divergence. - Simulated vehicle successfully tracks the path within acceptable tracking error. - No dropped ROS connections or WebSocket failures during runtime.	

Table 6: F1Tenth Simulation Full-System Integration Test

T.P.06 - Real-World F1Tenth Hardware Integration Test	
Objectives	
To validate that the integrated software stack (Pure Pursuit, Path Planning, Mapping, UI) functions correctly on the physical F1Tenth platform. **Primary Goal:** Ensure the algorithms fit within the hardware's computational constraints (CPU, RAM, Latency) while maintaining safe, real-time autonomous driving performance.	
Equipment	Physical F1Tenth Car (Orin), Remote Laptop.
Elements	**Modules:** F1Tenth Stack, Global Planner, Pure Pursuit, Interface Bridge. **Hardware:** VESC, Servo, Lidar.
Personnel	Sam Dong, Po-Ting Ko, Joshua Tsai
Location	Physical Test Track / Laboratory (Flat surface with defined boundaries).
Procedure	
1. **Bench Test (Wheels Up):** Place car on blocks. Launch all drivers and algorithms. 2. **Resource Monitoring:** Open terminal and run `htop` or `jtop`. Monitor CPU and RAM usage while all nodes (Mapping, Planning, Control, UI) are active. 3. **Latency Check:** Verify frequency of `/drive` topic is stable (e.g., greater than 20Hz) and Lidar scan delay is less than 50ms under load. 4. **Static Steering Test:** Send waypoints via UI. Confirm physical steering wheels turn to match the Pure Pursuit calculation without jitter. 5. **Slow Speed Auto:** Place car on track. Set speed limit to 1.0 m/s. Activate autonomous mode. 6. **Full Integration Run:** Command a multi-lap mission via UI. Monitor "Time to Plan" vs "Time to Execute" to ensure the hardware isn't lagging behind the car's motion.	
Verification Criteria	
- **Hardware Fit:** CPU usage stays below 90% (no thermal throttling); RAM usage below 85%. - **Control Safety:** No "jerk" movements when traverse all the waypoints. - **Path Accuracy:** Physical cross-track error less than 15cm compared to planned path. - **UI Sync:** Remote UI updates robot pose without freezing/crashing the onboard ROS bridge.	

Table 7: Real-World Hardware Integration Test Plan

T.P.07 - Behavior UI/UX Unit Test	
Objectives	
To validate that the Behavior User Interface correctly integrates camera streaming, object detection visualization, interactive object selection, and target ID publishing for downstream visual servo control.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), Object Detection & Tracking Node, Behavior UI Module.
Elements	**Software Modules:** Detection Node, Visual Servo Control Node. **Interfaces:** `/camera/image`, `/detections`, `/target_id`.
Personnel	Maddie Wang
Location	AI Maker Space / Laboratory Environment
Procedure	
1. Launch camera driver and confirm live image stream is visible in the Behavior UI. 2. Launch object detection node and verify bounding boxes appear correctly. 3. Confirm each detected object is assigned a unique and stable ID across frames. 4. Click on a detected object within the UI. 5. Verify the selected object's bounding box changes color immediately. 6. Confirm the selected object ID is published on `/target_id`. 7. Launch the visual servo node and verify it subscribes to the selected ID. 8. Confirm steering commands are generated based only on the selected object. 9. Rapidly switch selection between objects and verify system stability. 10. Temporarily remove the selected object from the camera view and observe behavior.	
Verification Criteria	
- Camera feed displays continuously without frame drops or freezing. - Bounding boxes align correctly with detected objects. - Object IDs remain consistent across frames during tracking. - Bounding box color changes within 100 ms of user click. - Only one object can be selected at any time. - Selected ID is correctly transmitted on `/target_id`. - Visual servo reacts exclusively to the selected ID. - If selected object disappears, system halts tracking or issues warning safely. - No UI freeze, ROS bridge failure, or control instability during rapid switching.	

Table 8: Behavior UI/UX Unit Test

T.P.08 - Object Detection & Tracking Test	
Objectives	
To evaluate the YOLOv12 architecture's efficacy in person-class detection and its consistency in multi object tracking (MOT) across diverse motion scenarios.	
Equipment	Development Workstation (Ubuntu 22.04, PyTorch, YOLOv12 source code), high-resolution validation datasets.
Elements	Local Perception Subsystem
Personnel	Sam Dong, Leo Xu, Maddie Wang
Location	AI Maker Space
Procedure	
1. Deploy the YOLOv12 repository and load pre-trained weights. 2. Download 10 diverse video from Pexels, categorized by: a. Single-person linear motion b. Multi-person scenes (2–5 subjects) with trajectory crossings. 3. Run the YOLOv12 tracking pipeline using a persistent tracker. 4. Capture frame-by-frame data including bounding box coordinates, confidence scores, and an unique track identities. 5. Review output visualizations to assess bounding box stability and ID persistence.	
Verification Criteria	
- The system must maintain a False Positive Rate (FPR) of $< 5\%$ across all test sequences. - Track IDs must remain unique. - The model inference must has an average latency of < 80 FPS on the tested hardware. - Successful export of track telemetry in a format suitable for downstream integration.	

Table 9: Object Detection & Tracking Test

T.P.09 - Visual Servo Control Test	
Objectives	
To verify that the `visual_servo` node correctly subscribes to detected object image coordinates, computes heading error relative to the image center, and generates steering control using a PID controller for target alignment.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), F1Tenth Simulation or Camera Feed Source, Rviz2 (optional), ROS 2 CLI
Elements	`visual_servo` node, Object Detection Publisher (bounding box or pixel center), PID Controller Module, Vehicle Control Topic (e.g., `/drive`)
Personnel	Po-Ting Ko
Location	AI Maker Space
Procedure	
1. Launch the `visual_servo` node in a standalone terminal. 2. Publish mock object center coordinates (e.g., `geometry_msgs/Point`) representing detected object pixel position. 3. Define image width and compute image center (cx, cy). 4. Compute horizontal pixel error: $e = x_{object} - x_{center}$. 5. Convert pixel error into heading error (normalized by focal length or image width). 6. Apply PID controller to generate steering command. 7. Publish steering control to vehicle topic (`/steering` (type: `std msgs/Float32`)). 8. Vary object position across image (left, center, right) and observe response. 9. Monitor control output using `ros2 topic echo` and Rviz2 visualization.	
Verification Criteria	
- Node subscribes to object coordinate topic without crashing. - Pixel error is correctly computed relative to image center. - Steering command sign is correct (left error → left turn, right error → right turn). - PID output responds smoothly without excessive oscillation. - When object is centered, steering command converges to zero. - Control response remains stable under rapid target movement. - No integral windup or instability observed during sustained offset.	

Table 10: Visual Servo Control Node Test

T.P.10 - Planning Mode Selector Unit Test	
Objectives	
Implement a planning mode selector that switches between Destination mode and Following mode based on user preference. When the following mode is enabled, the system continuously tracks and follows the designated objects until the mission is complete. It then switches back to the destination mode to finish the remaining patrol route.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), ROS 2 CLI (for topic publishing)
Elements	`planning_mode_selector` node
Personnel	Joshua Tsai
Location	AI Maker Space
Procedure	
1. Record visited waypoints and remaining waypoints. 2. When receiving user command from `Behavior UIUX`, switch to Following mode. 3. Receive the location of designated objects, send to `visual_servo` node to generate the control command. 4. Receive end signal when the following task is completed (user command, out of patrol area, lose object signal). 5. Send remaining waypoints to the planner module to generate the new destination path. 6. Switch to Destination mode.	
Verification Criteria	
- Visited waypoints and remaining waypoints are successfully separated and recorded. - Successfully receive and send topics from perception, localization, planning, visual_servo, and UIUX modules. - Determine the correct decision when the following task is completed. - Successfully switch between Following mode and Destination mode.	

Table 11: Planning Mode Selector Unit Test

T.P.11 - LSV (Golf Car) Simulation Test	
Objectives	
Validate the baseline Knightscope Autoware planning pipeline in the LSV simulation environment and confirm that it can generate a feasible planned path to a given goal. This test also aims to familiarize the team with Autoware usage, including launch procedures, required inputs, and common tools, establishing a reliable baseline for future planning feature expansion.	
Equipment	Development PC (Ubuntu 22.04 + ROS 2 Humble), ROS 2 CLI (for topic publishing)
Elements	Autoware.universe, `mission_planner` module
Personnel	Joshua Tsai
Location	AI Maker Space
Procedure	
1. Set up the LSV simulation environment and bring up the Autoware stack required to run planner. 2. Load the provided map and verify that the planning modules can access the map data. 3. Start the nodes to provide the planner with a valid vehicle pose (or a simulator pose source). 4. Configure a navigation goal through the interface and trigger the planner execution. 5. Visualize and monitor planner outputs (global path) using RViz and topic inspection tools. 6. Observe the planner behavior and confirm that outputs remain stable and continuous.	
Verification Criteria	
- The planning-related Autoware modules launch successfully in simulation without critical errors. - The planner receives required inputs (map, vehicle pose, goal pose) and produces valid planning outputs. - Planner outputs are continuous and stable during the run, with no invalid messages. - The team can consistently reproduce the workflow to run the planner in the simulator.	

Table 12: LSV (Golf Car) Simulation Test

T.P.12 - LSV (Golf Car) Full-System Integration Test	
Objectives	
To integrate and validate the autonomous software stack on the physical LSV platform. **Critical Goal:** Verify outdoor localization (GPS/LiDAR), path planning modules (Global planner), and safety override systems function correctly before attempting high-speed autonomous missions.	
Equipment	LSV Chassis (Golf Car), Industrial PC (Ubuntu + ROS 2), 3D LiDAR (Velodyne/Ouster), GPS/RTK Unit, CAN Bus Interface, Physical E-Stops.
Elements	**Hardware:** Drive-by-Wire System (Throttle/Brake/Steer). **Software:** Localization (NDT/EKF), Global Planner, MPC/Pure Pursuit, Safety Watchdog.
Personnel	Sam Dong, Po-Ting Ko, Joshua Tsai, Maddie Wang, Leo Xu
Location	Outdoor Test Track / Closed Parking Lot (Flat, open asphalt).
Procedure	
1. **Safety Pre-Check:** Inspect tire pressure, verify physical E-Stops cut power to the motor, and ensure the Safety Driver has full manual brake override authority. 2. **DBW Check:** With wheels jacked up (or in neutral), test steering, throttle, and brake response via joystick (Joy_node) to confirm CAN bus communication. 3. **Sensor Initialization:** Drive manually to verify GPS/RTK fix status (Fixed/Float) and 3D LiDAR point cloud registration in Rviz. 4. **Localization Lock:** Initialize NDT/EKF localization. Confirm the `/tf` tree connects `map` → `odom` → `base_link` consistently. 5. **Low-Speed Auto:** Align vehicle to a straight path. Engage autonomous mode with speed capped at 1.5 m/s. 6. **Takeover Test:** While moving, Safety Driver intentionally taps the brake. Confirm software disengages immediately (state transition to Manual). 7. **Waypoint Mission:** Execute a predefined loop trajectory. Monitor lateral error.	
Verification Criteria	
- **Safety:** Physical brake pedal press immediately disables autonomous control (Lat ¡ 10ms). - **Localization:** GPS/LiDAR fusion remains stable (no "jumps" ¿ 0.5m) during motion. - **Control:** Steering wheel turns smoothly without violent oscillation (verify PID/MPC gains). - **Endurance:** Vehicle completes 3 laps of the test track without intervention or system crash. - **Latency:** Control loop frequency maintains ¿ 30Hz on the CAN bus.	

Table 13: LSV (Golf Car) Physical Integration Test Plan

T.P.13 - Spring Validation Demo	
Objectives	
To validate the full-stack integration of the autonomous patrol system, specifically the system's ability to switch from global path following to active personnel tracking and successfully rejoin the original mission route.	
Equipment	Low Speed Vehicle, Intel RealSense Depth Camera D456, Valeo Mobility kit- 2MP fisheye ETH Camera, PC with ROS2 installed, Neousys IPC Computer, Ouster Dome Lidar, Velodyne Lidar
Elements	Perception Subsystem, Global Planning Subsystem, Visual Servo Subsystem, Local Planning Subsystem, SLAM Subsystem, UI/UX Subsystem, Behavior Control Subsystem
Personnel	Sam Dong, Po-Ting Ko, Joshua Tsai, Maddie Wang, Leo Xu
Location	Tepper Parking Lot
Procedure	
1. Open the UI on the PC and read a pre-stored map. 2. The user selects the rough start position on the UI and submit. 3. User drag and drop waypoints and submit after finishing. 4. UI sends the waypoints to the ROS2 Global Planning node, and receives a smoothed shortest path. 5. UI visualize the path and LSV starts moving along that path. 6. During patrol, the user is able to see a blue dot, representing the current position of LSV, moving along the path. 7. On a side window, users can monitor live video streams with bounding boxes and person IDs. 8. Users are able to click the bounding box at any time of the patrol, LSV will start tracking and following that personnel. 9. After the user exits the tracking mode, LSV will return to its original route.	
Verification Criteria	
- Open UI on PC and read pre-stored map. - User select the rough start position on UI and submit. - User drag and drop waypoints and submit after finishing. - UI send the waypoints to ROS2 Global Planning node, and receive a smoothed shortest path. - UI visualize the path and LSV start moving along that path. - During patrol, user is able to see a blue dot, representing the current position of LSV, moving along the path. - On a side window, user can monitor live video stream with bounding boxes and person IDs. - User is able to click bounding box at any time of the patrol, LSV will start tracking and following that personnel. - After the user exit the tracking mode, LSV will return to its original route.	

Table 14: Spring Validation Demo

Appendix A

Mandatory Requirements		
M.F.1	Plan Paths	
	M.P.1.1	Generate three candidate patrol routes (e.g., max-score route, shortest-distance route, max-coverage route) within 5 seconds after each cost-map update, which occurs every 1 minute.
	M.P.1.2	Compute a new route that returns the K7 to its original mission path, upon confirmation that an alarm is a false positive, within 5 seconds.
M.F.2	Predict User Behavior with AI Prediction Model	
	M.P.2.1	Generate 3 non-repeated waypoints based on user's input and possible hazard locations in 10 seconds.
	M.P.2.2	Predict hotspots with higher visiting frequency or having emergency probability lower than 80%
M.F.3	Decide Route	
	M.P.3.1	If the operator does not select any newly suggested routes, maintain the currently active route with a decision latency of less than 5 seconds.
	M.P.3.2	When the operator selects a new route, update the active route within 1 second.
M.F.4	Handle Emergency	
	M.P.4.1	When an emergency is detected, generate at least 3 candidate routes that direct the K7 through the emergency site from its current position and present these routes for operator selection.
	M.P.4.2	Initiate at least 10 FPS live video streaming automatically when the K7 enters a radius of 30 meters around the detected abnormal site.
	M.P.4.3	Provide the operator with the ability to either confirm the emergency or mark it as a false alarm within 2 seconds after live video or sensor evidence becomes available.
M.F.5	Follow Suspicious Object	
	M.P.5.1	Detect in-frame suspicious object every second.
	M.P.5.3	Report suspicious object to user within 0.5 second.
	M.P.5.2	Follow the selected object after receiving alert from the user within 5 seconds.
M.F.6	Interact With User	
	M.P.6.1	Allow at most 1 human operator to drop, move, and delete waypoints directly on the map via the UI.
	M.P.6.2	Allow the human operator to select 1 candidate route as the final patrol route.
M.F.7	Visualize Maps and Routes	
	M.P.7.1	Present more than 3 candidate route on the UI as a distinct gray line on the map.
	M.P.7.2	Visualize the current selected route in blue.

Table 15: Mandatory Performance Requirements