A Systematic Approach to Autonomous Unmanned System Experimentation

Ryan Halterman and Chris Scrapper
SPAWAR Systems Center, Pacific
San Diego, CA

ABSTRACT

Regimented experimentation of autonomous unmanned vehicles is an important enabler for performance assessment, issue identification, and root-cause analysis. As autonomous capabilities improve, these are increasingly problems of needles and haystacks; large amounts of test data are often required to fully identify and analyze erratic events. Acquisition and analysis of this test data can be arduous, error prone, and inefficient.

Driven by a philosophy that test methods should be repeatable, reproducible, and as automated as possible, we have developed an experimentation regime for expeditionary autonomous unmanned vehicles. This regime represents a method of decomposing functionality, building up evidentiary information, and analyzing component results on overall system performance despite the challenge of testing to sufficient statistical significance. We have also developed a set of methods and tools that automate data collection and analysis to facilitate experimentation. This paper describes this test and experimentation philosophy, regime, and associated tools developed under the Office of Naval Research, Code 30 Ground Vehicle Autonomy program.

Keywords: autonomous vehicles, test automation, experimentation, mobile robotics

1. INTRODUCTION

As autonomous systems become more capable and robust, uncovering and diagnosing remaining issues becomes increasingly challenging. Koopman and Wagner[1] argue that vehicle-level testing, alone, is not enough to ensure safety of autonomous vehicles. They propose that supplemental approaches to validation are required such as simulation, formal proofs, fault injection, and human reviews. While system-level testing is likely not sufficient to ensure system reliability, it plays a significant piece of the overall test and validation scheme. Some of the challenges to testing autonomous vehicles they identified include high reliability requirements due to ultimate absence of a human driver, high operational requirements due to complex environment, frequently non-deterministic algorithms, inductive learning algorithms, and fail-operational (rather than fail-stop) system requirements. The challenges of exhaustively testing autonomous vehicles notwithstanding, a regimented experimentation strategy is useful for systematically exposing new issues and ensuring that regressions are not introduced.

The Office of Naval Research, Code 30 Ground Vehicle Autonomy program seeks to develop autonomous capabilities for expeditionary ground vehicles. This suite of hardware and software systems is intended to provide a low-cost, platform and mission agnostic, autonomy appliqué for a large range of vehicles, with potential operational scenarios including on-road, unimproved, and off-road missions in either daytime or nighttime. The desire for expeditionary operation places limits on the assumptions that can be applied to its development. In particular, it cannot be assumed that a system will have previously traversed any given route. This limits the usefulness of pre-generated, high fidelity maps as often employed on autonomous systems and implies that all perception must be performed *in situ*. The intended user is the Marine Expeditionary Force, with particular use cases including Logistics Resupply Connector (LOGCON), Route Reconnaissance and Clearance (R2C), and Advanced Scout. The tenets of this effort include: (1) improved mission efficiency and effectiveness, (2) facilitating integration with existing force structure, (3) expanded area of operations, (4) providing core autonomous capabilities for multiple missions, (5) reducing overall life-cycle cost (installation, procurement, O&M), (6) reducing logistical footprint, and (7) reducing dependency on high-bandwidth communications.

Unmanned Systems Technology XIX, edited by Robert E. Karlsen, Douglas W. Gage, Charles M. Shoemaker,
Hoa G. Nguyen, Proceedings of SPIE Vol. 10195, 101950M · © 2017 SPIE
CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2266798

1.1 Integration Strategy

The experimentation strategy described herein was incorporated as a part of the overall integration strategy for the Ground Vehicle Autonomy program. This integration strategy was previously[2,3] described, but is reviewed here to provide context. The major activities within it are depicted in Figure 1. This model defines a continuous cycle for development and testing that allows for simultaneous development, test, and production release of the system. This implies that at any given point in time several versions of a maturing technology may exist in different development threads with feedback links from one thread to another.

Within the integration strategy, a singular baseline system is maintained as the canonical system. Potentially improving capabilities are developed and vetted in relation to the baseline over the course of overlapping waves. Within each wave, four overlapping activity threads are performed: conduct analysis, evolve system architecture, integrate capability enhancements, and validate system.

Conduct analysis is concerned with the development and experimentation of individual capabilities and the assessment of their contribution to overall system improvement. Each capability under development effectively follows its own conduct analysis thread. In this way, there are multiple parallel conduct analysis threads. The conduct analysis thread culminates in an integration field test and a technology readiness assessment designed to determine the suitability for baseline integration of each capability under test. The period of the integration strategy, as executed under the Ground Vehicle Autonomy program, is approximately six months and is typically driven by the frequency of these integration tests.

The remaining activity threads concern the singular baseline system. Evolve system architecture considers holistic changes to the system architecture to facilitate improvements to maintainability and to facilitate future capability development needs.

Integrate capability enhancements seeks to incorporate new capabilities that have passed all hurdles of the conduct analysis thread and that have been deemed ready to include in the baseline system. Multiple new capabilities are generally integrated each wave and the integrate capability enhancements thread ensures that there are no conflicts between them or the existing baseline. Integrate capability enhancements culminates in a system verification test and a new stable baseline release.

Validate system provides extended testing of the newly released baseline. It includes continual system testing in operational environments and seeks to uncover hidden and erratic issues.

2. EXPERIMENTATION STRATEGY

The experimentation strategy developed for the Ground Autonomy Program is founded on the ideas that the test suite should address both subsystem components and the complete system and that individual tests should be repeatable, reproducible, and as automated as possible. These ideas stem from the need operate as efficiently as possible so as to maximize time and funding. Even without limits therein, achieving a statistical level of certainty regarding system operation is a monumental task. The experimentation strategy is based on three primary steps: decomposing functionality, building evidentiary information, and analyzing component results. In accordance with the overarching integration strategy[2,3] executed under the Ground Vehicle Autonomy program, these steps are continually refreshed.

2.1 Decompose functionality

In order to define a regime for developing, integrating, and testing a complex, multi-faceted system, we conceptualized a reference model architecture (shown in Figure 2) that represents a hierarchal decomposition of the functional elements that comprise the Ground Vehicle Autonomy system. Each row in the reference model diagram may be viewed as a level of autonomy. At the lowest level are the most basic control and perception operations. Capability at this level effectively provides nothing more than tele-operation. The highest level considers collaborative behaviors and tactics that involve autonomy of multiple (potentially cross-domain) autonomous platforms.

This model serves several purposes across the spectrum of development, integration, and testing of the system. It serves as a foundational vernacular that allows team members to discuss capabilities and issues across the

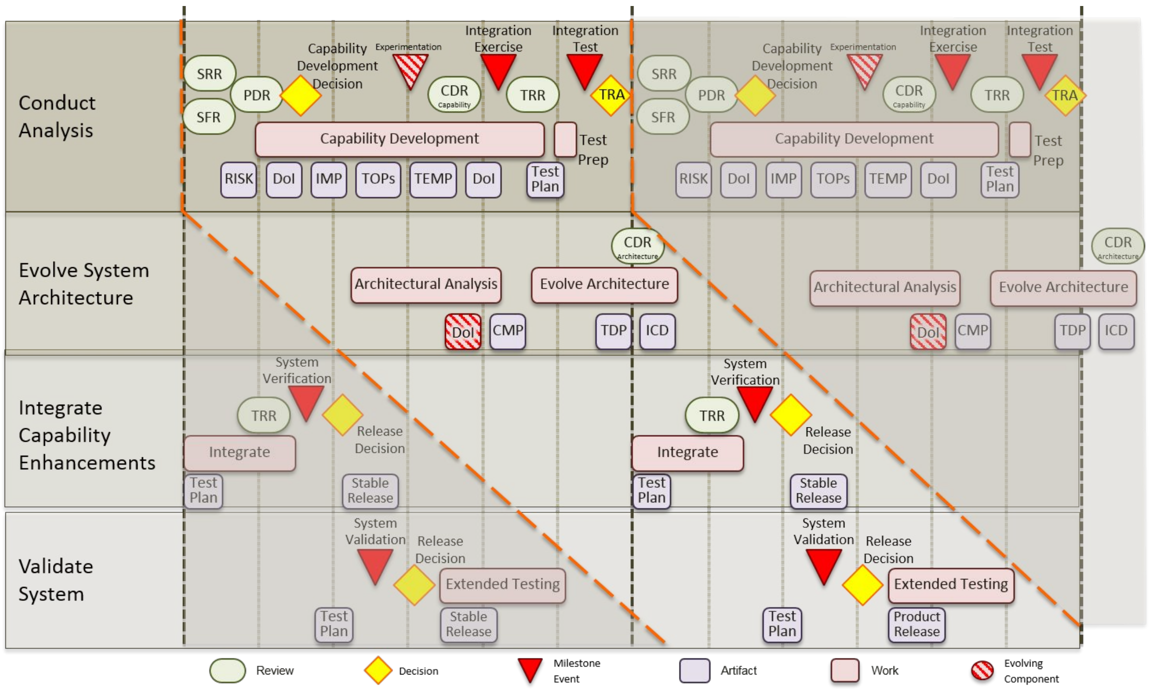


Figure 1. Integration Strategy chart showing overlapping waves (columns) and overlapping activity threads (rows)

system in a common vocabulary and with common reference points. This provides improved clarity and specificity in technical discussions – even when involving team members across functional areas. By simply referencing, for example, “Maneuver Level sensor processing”, a perception developer can seed a discussion in which diverse participants from the planning team, the world modeling team, the integration team, or the management team immediately understand what part of the system is under consideration and which touch points relate to their particular areas of control.

The Reference Model also serves as a basis to analyze risks, issues, and capability prioritization by aligning knowledge requirements. In this function, it simplifies analysis of capability dependency and establishes a dependency graph indicating which lower-level capabilities are required to facilitate a desired higher-level capability.

Finally, the Reference Model serves as a high level elemental test coverage matrix. It permits a view into which capabilities require the development of additional test methods and emphasizes the utility of methods that stress the system – and its subcomponents – at a variety of levels. For example, when evaluating the ability of a stereo-vision based perception system to detect, classify, and track obstacles in the environment, it is useful to first evaluate its ability to produce usable images under varying conditions, place voxels in a scene through accurate camera calibration and stereo correlation, and discriminate a drivable support surface from potential obstacles. In this way, root cause analysis for discovered issues can leverage test traceability throughout their respective processing chains. Additionally, a compendium of highly-automated tests that cover all aspects of the reference model facilitates regression testing as the system evolves.

2.2 Build evidentiary information

With the conceptual system decomposition and appropriate test method coverage, we continuously exercise the system. Koopman and Wagner[1] estimate that, for a hypothetical fleet of one million vehicles operated one hour per day, over a billion hours of testing would be required to verify a rate of one catastrophic computing failure every 1000 days. This illustrates the challenge inherent in autonomous system development and experimentation. With a system that currently falls well short of that failure rate, the proverbial system issue needles are still relatively prevalent in the haystack of the test suite.

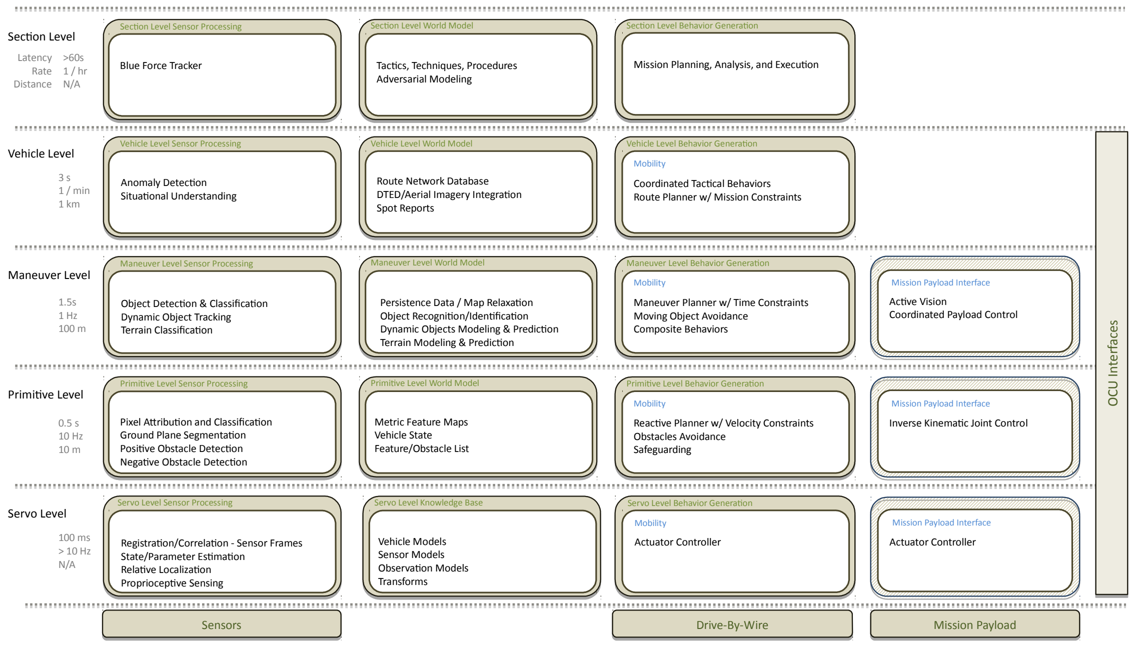

Section Level
Latency >60s
Rate 1 / hr
Distance N/A
Section Level Sensor Processing
Blue Force Tracker
Section Level World Model
Tactics, Techniques, Procedures
Adversarial Modeling
Section Level Behavior Generation
Mission Planning, Analysis, and Execution
Vehicle Level
3 s
1 / min
1 km
Vehicle Level Sensor Processing
Anomaly Detection
Situational Understanding
Vehicle Level World Model
Route Network Database
DTED/Aerial Imagery Integration
Spot Reports
Vehicle Level Behavior Generation
Mobility
Coordinated Tactical Behaviors
Route Planner w/ Mission Constraints
Maneuver Level
1.5s
1 Hz
100 m
Maneuver Level Sensor Processing
Object Detection & Classification
Dynamic Object Tracking
Terrain Classification
Maneuver Level World Model
Persistence Data / Map Relaxation
Object Recognition/Identification
Dynamic Objects Modeling & Prediction
Terrain Modeling & Prediction
Maneuver Level Behavior Generation
Mobility
Maneuver Planner w/ Time Constraints
Moving Object Avoidance
Composite Behaviors
Mission Payload Interface
Active Vision
Coordinated Payload Control
Primitive Level
0.5 s
10 Hz
10 m
Primitive Level Sensor Processing
Pixel Attribution and Classification
Ground Plane Segmentation
Positive Obstacle Detection
Negative Obstacle Detection
Primitive Level World Model
Metric Feature Maps
Vehicle State
Feature/Obstacle List
Primitive Level Behavior Generation
Mobility
Reactive Planner w/ Velocity Constraints
Obstacles Avoidance
Safeguarding
Mission Payload Interface
Inverse Kinematic Joint Control
Servo Level
100 ms
> 10 Hz
N/A
Servo Level Sensor Processing
Registration/Correlation - Sensor Frames
State/Parameter Estimation
Relative Localization
Proprioceptive Sensing
Servo Level Knowledge Base
Vehicle Models
Sensor Models
Observation Models
Transforms
Servo Level Behavior Generation
Mobility
Actuator Controller
Mission Payload Interface
Actuator Controller
OCU Interfaces
Sensors
Drive-By-Wire
Mission Payload

Figure 2. Reference Model Architecture

Continuous subsystem testing is performed by functional area leads who shepherd capabilities within a given subsystem (functional area) through experimentation and the integration strategy. Large scale in-field system testing occurs approximately every six months. These integration exercises are used as the final wicket that must be crossed prior to a capability integration decision.

2.3 Analyze component results

According to the integration strategy implemented under the Ground Vehicle Autonomy program, capabilities are added to, modified within, and removed from the baseline system in response to evidentiary information obtained through the continuum of testing. If a proposed change (capability addition, modification, subtraction) improves performance relative to the baseline across the compendium of tests, it is considered for inclusion. Changes which maintain a comparable level of performance may be integrated into the baseline if they bring other less tangible benefits (e.g. simplicity, maintainability, etc.). Changes which under perform the baseline system are not permitted for integration. In this way, effectively monotonic capability improvements are assured in the baseline system while allowing varied capability exploration in feature branches. Figure 3 illustrates this improvement over the course of several waves of the integration strategy. For a given test route, an increase in the max and mean distance traveled autonomously is accompanied by a decrease in the number of operator interventions (safety stops), planner timeouts, and unexpected behaviors. This performance evaluation stems from an analysis of the evidentiary information accumulated through system and component level tests.

3. TEST METHODS AND TOOLS

This section discusses some of the test methods and software tools developed by the Ground Vehicle Autonomy team to expedite experimentation and testing of the autonomous system and its subcomponents at various levels of the reference model – from servo and primitive level elemental tests through vehicle level system tests. Test methods developed under the Ground Vehicle Autonomy program were designed to be repeatable and

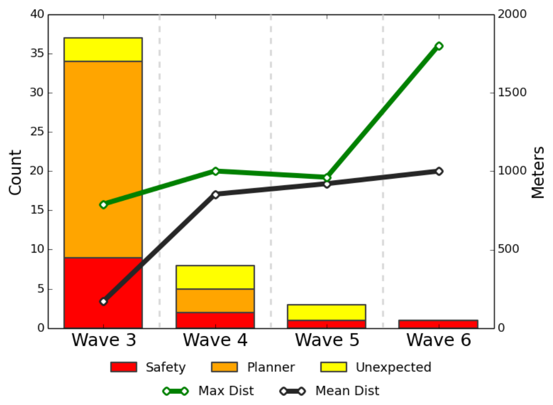


Figure 3. System performance over time showing an increase in the distance traveled autonomously and a decrease in the number of events for a given test route.

reproducible to the degree possible. In this way, they are modeled after the American Society for Testing and Materials (ASTM) standard test methods for response robots.[4] Most have data collection and analysis tools that simplify test conduct.

3.1 Servo Level Tests

3.1.1 Relative Localization Accuracy

This experiment quantifies the error accumulation of the relative localization state estimation algorithm over large off-road distances. Data is collected off-road on a controlled path. Data analysis reports error in terms of positional and heading error as a percentage of distance traveled as well as a direct plot of 2D position overlaid on aerial imagery for qualitative analysis. These results provide insight into the rate of relative position error experienced during GPS outages. The data can also be used to evaluate the rate of growth of error under varying operating environments.

Test Metrics 1. Position error, defined by 3D positional error as percentage of distance traveled. 2. Heading error, defined by the heading error as percentage of distance traveled. 3. Mean and standard deviation of error at end of run, as percentage of distance traveled.

Test Preparation The GPS data has been disconnected from the relative localization state estimator system, but is still being recorded by the system. State estimation system has been initialized.

Test Procedures 1. On the test log, record header information, e.g., environmental conditions, tester, configuration, etc. 2. Place the vehicle in the starting position and heading at the beginning of the course. 3. Ensure that the localization system has been properly initialized and has accurate heading information. 4. Drive the course according to the localization test profile. Record the following using system logging: relative and absolute position estimate output by localization system. Absolute GPS position and heading information. Raw sensor data from IMU, GPS, gyroscope, wheel odometry.

Data Analysis Procedures 1. Sub-sample data at lengths of 100 m. Plot horizontal position error as a percentage of distance traveled and calculate the mean and standard deviation. 2. Plot the absolute path versus the relative aligned paths for both the controlled and user selected routes. Plot percentage of position error over distance traveled. Plot heading error with respect to time.

Test Profile The course driven must include conditions likely to occur during an off-road mission scenario. These conditions include the following: A minimum total distance of 2 km. A minimum drive time of 20 minutes. Maintaining a stationary position for 3 minutes. Traveling in reverse for at least 25 km. 5 instances of heavy acceleration and 5 instances of heavy braking. Course elements with pitch and/or roll exceeding 10 degrees for at least 30 seconds. A cumulative elevation change of at least 50m. Injected GPS loss for 2 minutes during vehicle motion. Injected gyroscope loss for at least 2 minutes during vehicle motion.

3.1.2 Low-level Controller Step Response

This experiment measures the low-level controller's ability to faithfully execute velocity and steering commands. The test records the vehicle's response to commanded velocity and steering angle step commands. This allows for simple measurements of the transient and steady state characteristics. The test does not require autonomy and is run independently from most of the vehicle subsystems.

Test Metrics 1. Steady state error. 2. Percent overshoot. 3. Settling time.

Test Preparation 1. Identify a smooth road with a relative constant slope. 2. Define the start of the experiment with a visual marker.

Test Procedures 1. Drive the vehicle to a predefined velocity (could be 0 m/s). 2. Apply a step velocity or steering command. The step command should occur at the same location for each trial. 3. Measure the velocity or steering response of the vehicle. 4. Repeat the experiment 10 times and average the results. 5. Perform steps (1)-(4) for different step sizes and road inclinations.

Data Analysis Procedures 1. Plot the commanded velocity and the measured velocity. 2. Plot the error signal. 3. Measure the steady state error, percent overshoot, and settling time.

3.1.3 Low-level Controller Constant Velocity

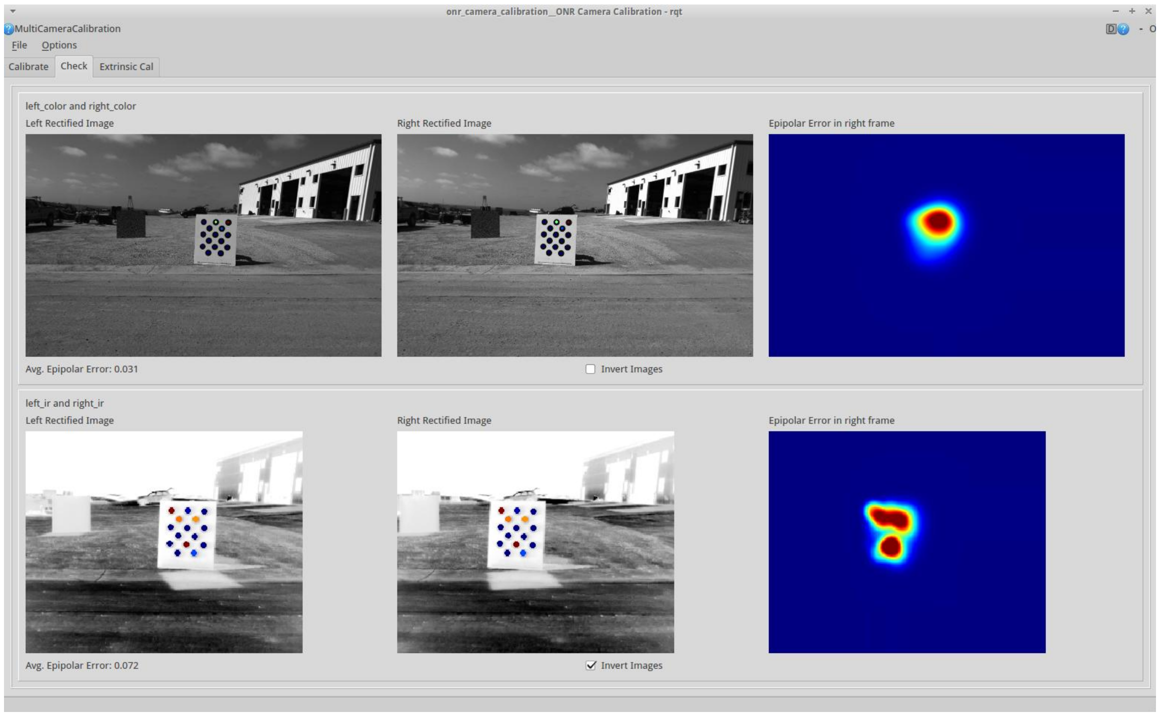
The low-level control constant velocity test, evaluates the controller's ability to maintain a constant velocity while traversing hills. The test does not require autonomy and is run independently from most of the vehicle subsystems. Steering actuation is disengaged in order to allow for manual control of the steering wheel. A constant command velocity is then sent to the low-level controller and the resulting error signal is measured as the vehicle is driven over a hilly road.

Test Metrics Mean sum of squared velocity error.

Test Preparation 1. Identify a road with several rolling hills of various inclinations. 2. Define the start of the experiment with a visual marker.

Test Procedures 1. Place the vehicle at a predefined starting location. 2. Sample the commanded velocity and measured velocity at a uniform rate. 3. Apply a constant velocity command. 4. Drive the vehicle over a set of hills. 5. Measure the velocity response of the vehicle. 6. Stop the vehicle at a predefined location. 7. Repeat the experiment 10 times and average the results.

Data Analysis Procedures 1. Plot the commanded velocity and the measured velocity. 2. Plot the error signal and vehicle pitch. 3. Compute the sum of squared errors and divide by the total drive distance.



Avg. Epipolar Error: 0.031

Avg. Epipolar Error: 0.072

Figure 4. Camera epipolar error tool

3.2 Elemental Perception Tests

3.2.1 Camera Epipolar Check

This test evaluates the quality of the stereo camera calibration by measuring the epipolar reprojection error. If a stereo calibration exhibits large reprojection error, the stereo correlation suffers and the resultant disparity map is often sparse and erroneous.

Test Metrics Epipolar reprojection error sampled across the stereo field of view.

Test Preparation Stereo cameras are calibrated using a structured target to calculate camera intrinsic and extrinsic parameters.

Test Procedures With the system stationary and the epipolar reprojection error tool running, place a structured stereo calibration target within the stereo field of view. When the tool detects the board, dots will be drawn on the targets on the calibration board. Dark blue dots correspond to low epipolar error, while dark red dots correspond to higher epipolar error. Once the target has been detected, move the target to a new location within the stereo field of view. Repeat until the entire field of view has been sampled.

Data Analysis Procedures Plot the epipolar reprojection error magnitude for sampled locations across the stereo field of view. The analysis tool automatically generates this plot as shown in Figure 4.

3.2.2 Stereo Range Accuracy

This test quantifies the range accuracy of the stereo system for an object directly in front of the vehicle and in the periphery of the horizontal field of view. The range accuracy of the stereo system is evaluated using a rigid board printed with high-texture orange and black patterns, see Figure 5. Range accuracy affects the ability of the navigation system to effectively plan to avoid non-traversable regions in the environment. Greater accuracy may minimize potentially hazardous collisions and may permit longer range planning.

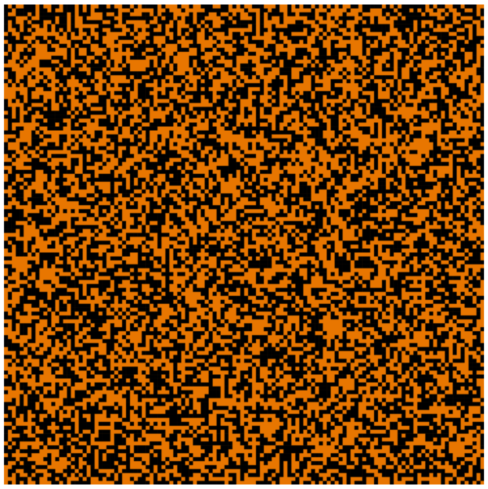


Figure 5. Stereo range accuracy high texture target

Test Metrics Object range error defined by the difference between the average pixel disparity of the object (in world coordinates) and the measured object distance from the vehicle. Object range variance defined by the standard deviation of the target pixel disparity.

Test Preparation Stereo cameras are calibrated using a structured target to calculate camera intrinsic and extrinsic parameters. Camera to vehicle calibration may be generated by design information or automated calibration. Test location has appropriate markers for placing and aligning the vehicle as well as location markers for all target locations.

Test Procedures Using location markers move the vehicle to its static location and orientation. On the test log, record header information, e.g., environmental conditions, tester, configuration, etc. Using the target test stands, place three identical targets vertically in front of the vehicle, at 35m and at 0°azimuth, 20m and -45° azimuth, and 5m and 45° azimuth. Record five seconds of calibrated images. Record the bagfile name and corresponding test information.

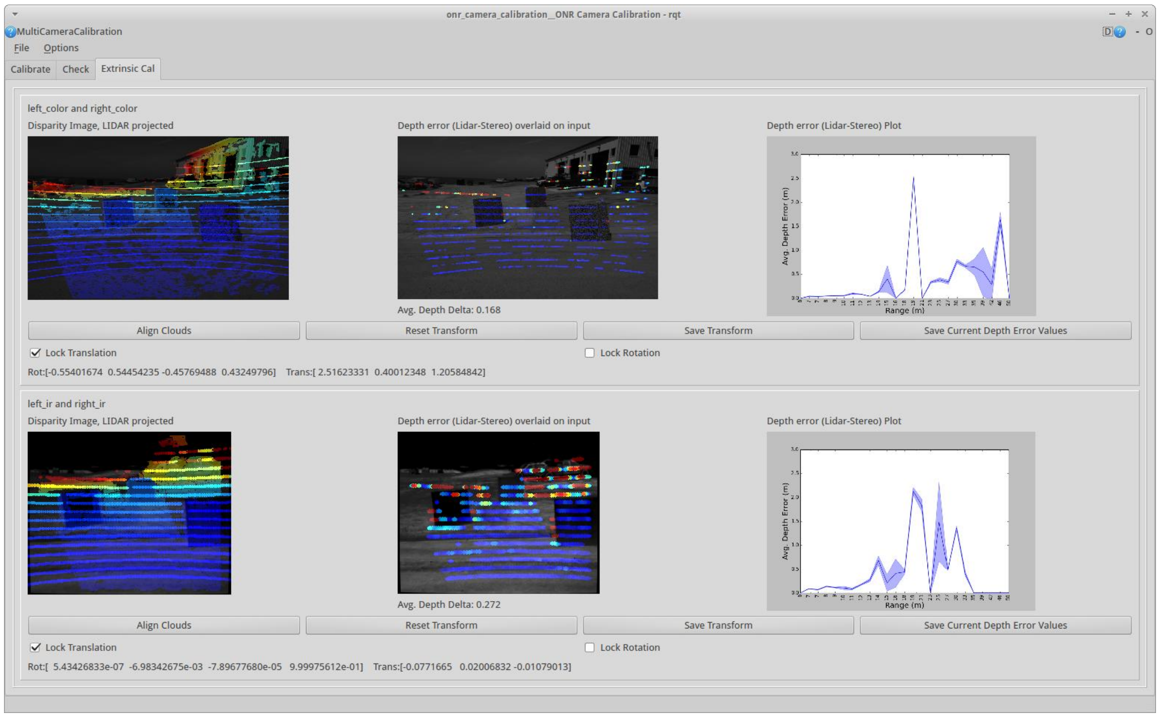
Data Analysis Procedures Plot range error vs. range for all pixels with ground truth. Plot standard deviation of error versus range for all pixels with ground truth. A co-located multi-line LIDAR sensor is used to define range ground truth. The analysis tool automatically correlates LIDAR points with stereo pixels and estimates the resultant stereo range error. Figure 6 shows this tool and example range-depth error plots for a color and an infrared stereo system.

3.3 Elemental Planning Tests

3.3.1 Maneuverability Course

The maneuverability course tests the vehicle's ability to navigate various environments, e.g., on-road, unimproved, and off-road, while avoiding natural and artificial objects such as bushes and barrels. For ease of transportation and storage, we primarily use collapsible barrels similar to that in Figure 7.

Features of the maneuverability course include: gate and dodge, slalom, sharp and/or blind corners, and open ranges and corridors. The gate and dodge element forces the vehicle through a narrow channel where it must immediately dodge a barrel afterwards. The slalom forces the vehicle to navigate back and forth through several barrels. Sharp and/or blind corners force the vehicle to take wide turns or perform k-point turns. Open ranges and corridors allow the vehicle travel at higher speeds.



left_color and right_color

Disparity Image, LIDAR projected

Depth error (Lidar-Stereo) overlaid on input

Depth error (Lidar-Stereo) Plot

Avg. Depth Delta: 0.168

[✓] Lock Translation   [ ] Lock Rotation

```
Rot:[-0.55401674  0.54454235 -0.45769488  0.43249796]   Trans:[ 2.51623331  0.40012348  1.20584842]
```

left_ir and right_ir

Disparity Image, LIDAR projected

Depth error (Lidar-Stereo) overlaid on input

Depth error (Lidar-Stereo) Plot

Avg. Depth Delta: 0.272

[✓] Lock Translation   [ ] Lock Rotation

```
Rot:[  5.43426833e-07  -6.98342675e-03  -7.89677680e-05   9.99975612e-01]   Trans:[-0.0771665   0.02006832 -0.01079013]
```

Figure 6. Stereo extrinsic calibration check

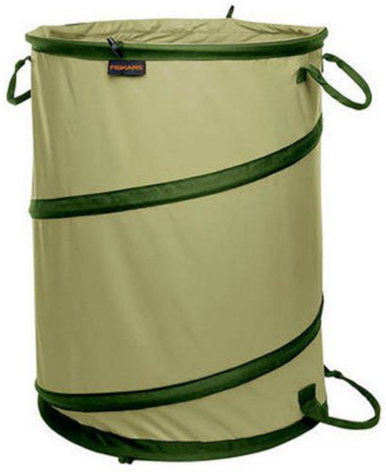


Figure 7. Collapsible barrel obstacles

The maneuverability course is primarily used to test the reactive planner, which computes the desired vehicle trajectory over a 5-second time horizon. The reactive planner accepts a target trajectory from the maneuver planner and generates speed and steering commands for the low-level controller using a receding horizon model predictive (RHMP) control framework in which the trajectory is optimized over a cost function based on a feed-forward simulation. This system therefore depends on accurate cost maps with labeled objects, accurate localization of the robot, and accurate servo control.

This test is performed according to the procedure outlined below. This procedure currently relies on manual logs; we intend to incorporate it into the Mole test tool (see section 3.4.1) to facilitate automated data collection, metric generation, and data archival.

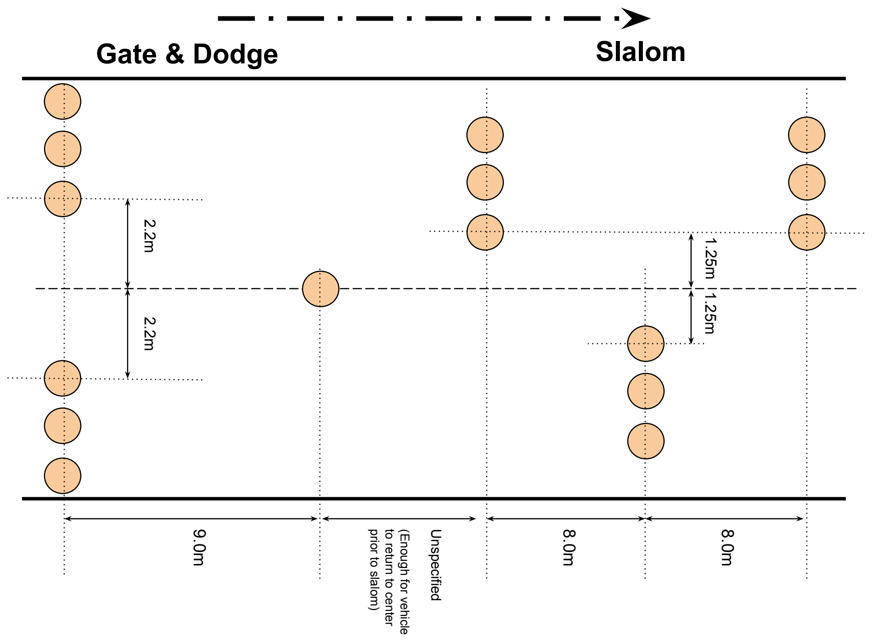

Gate & Dodge
Slalom
2.2m
2.2m
1.25m
1.25m
9.0m
Unspecified (Enough for vehicle to return to center prior to slalom)
8.0m
8.0m

Figure 8. Maneuverability Course. Dash-dot arrow indicates direction of travel. Variant reverses direction of travel.

Test Metrics 1. Maximum speed. 2. Average speed. 3. Number of safety stops. 4. Number of planner timeouts. 5. Number of permissible obstacle interactions. 6. Number of premature "goal reached" stops. 7. Average number of events per run, i.e., (#3 + #4 + #5 + #6) / (number of trials). 8. Success rate, i.e., (number of completions) / (number of trials).

Test Preparation Prepare test apparatus as in Figure 8.

Test Procedures 1. On the test log, record header information, e.g., vehicle id, environmental conditions, test administrator, system configuration, etc. 2. Drive the test course in autonomy mode. 3. Record the following data using system logging: local odometry, stereo disparity, LIDAR point clouds, and cost maps. 4. If human intervention is required, on the log form, annotate: location of human intervention, reason for human intervention, type of obstacles and conditions for stopping, odometer reading at time of intervention, and end and restart time of autonomy.

Data Analysis Procedures Tabulate the number and types of events defined above. Extract the autonomous drive times and compute the maximum and average speeds.

3.3.2 Congested L

The Congested L test exercises the vehicle's ability to negotiate a blind corner in a tight corridor. This scenario is challenging both from a perception point of view as well as a path planning point of view. The perception challenge is that the opening may not be immediately apparent to the system and radial noise (due to stereo disparity mismatches) may cause the opening to appear blocked. The path planning challenge is that the tight turn makes it difficult for vehicle to escape the corridor with a single turn.

In such environments, it is not always possible to perform simple one-directional turns. Instead, the system must perform a multi-point turn in which the vehicle must back up several times. Furthermore, in a congested

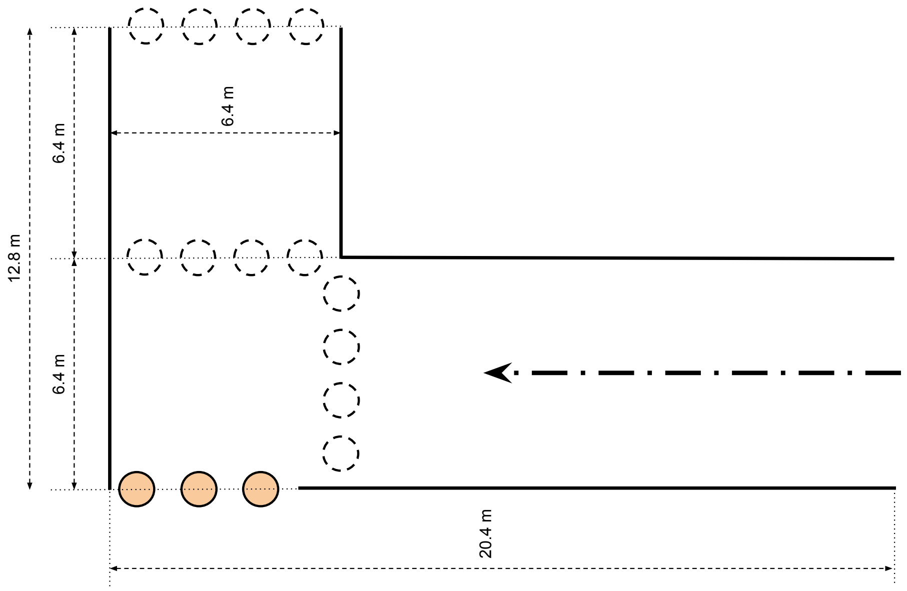


Figure 9. Congested L. Dash-dot arrow indicates direction of travel. Dashed circles indicate possible variants through placement of additional obstacles to force congested maneuvering (e.g. multi-point turns).

environment the vehicle may need to make a decision as to which path to take. This experiment will verify the functionality and effectiveness of the autonomous system's path planning algorithms in these types of environments.

This test is performed according to the procedure outlined below. This procedure currently relies on manual logs; we intend to incorporate it into the Mole test tool (see section 3.4.1) to facilitate automated data collection, metric generation, and data archival.

Test Metrics Number of completions in identified congested areas.

Test Preparation Prepare test apparatus as shown in Figure 9.

Test Procedures 1. On the test log, record header information, e.g., environmental conditions, tester, configuration, etc. 2. Drive the test course in autonomy mode. 3. If human intervention is required, record the following: approximate location of human intervention, reason for human intervention, type of obstacles and conditions for stopping, odometer reading at time of intervention, end and restart time of autonomy.

Data Analysis Procedures Tabulate the number and types of unsuccessful attempts through congested environments.

3.4 System Level Tests

System level tests are used to evaluate the performance and effectiveness of the autonomy system working as a whole. These tests are conducted within operationally-relevant scenarios, which are meant to model a variety of conditions and, in particular, identify possible points of system failure. There are three milestone system-level tests within the integration strategy[2,3] for the Ground Vehicle Autonomy program: Integration Tests, System Verification tests, and System Validation tests. Integration Tests are conducted at the end of each development wave (nominally six months) and are used to assess the level of maturity attained by developing technologies and to determine their readiness for integration into the baseline system. Subsystems that are deemed ready for integration are further vetted to ensure proper code compliance and system reliability. All subsystems slated for integration are then tested in a System Verification test to ensure that all requirements identified in the vetting process have been fulfilled. After completion of the System Verification test, a Stable Build of the autonomy system is released, which becomes the new baseline system. Whereas System Verification tests at the system-level, System Validation tests at the user-defined mission-level. Specifically, a System Validation test evaluates the effectiveness of the system to operate within a realistic mission and provides opportunity for user feedback.

Both the Integration Test and the System Verification test include a Nominal Autonomy test to assess the overall performance of the system. A Nominal Autonomy test consists of sending the vehicle on a defined course through a relevant expeditionary environment (e.g. dirt road through a canyon, mock village, etc.), through which it must navigate autonomously to reach the desired goal point. Embedded within the test scenario are several challenging factors, including some of the elemental tests described above. For the System Validation test, the vehicle is again put into an operationally-relevant scenario. However, the focus of this test is at the mission-level as opposed to the system-level. Thus the System Validation test adds another level of complexity by introducing external human/vehicle players, adversarial threats, and mission objectives that must be met. In what follows, we now describe in detail the Nominal Autonomy test.

3.4.1 Nominal Autonomy

The nominal autonomy test is composed of one or more routes through a relevant expeditionary environment. The objective of the test is to complete the course with as few human interventions as necessary. The test captures information on the vehicle's ability to autonomously navigate over extended drive times through different terrain types and under different complicating environments. It gauges the amount of human intervention needed to successfully complete an extended route in a challenging off-road environment (e.g. congested buildings and man-made structures, tall grass, dirt trails, blind turns, inclines, side slopes, dense canopies, obscuring foliage, etc.).

These courses contain a number of natural and artificial challenges that test the system's aptitude for navigating a previously unseen environment. In some cases, the natural environment of the test course can be used to define these test elements. In other cases, artificial test apparatuses are added to the environment to further stress system performance.

Continuous Nominal Autonomy The nominal autonomy test is further divided into a continuous case and a segmented case. In continuous nominal autonomy, the objective point is distant. The system autonomously transits the several kilometer-long test course and events of interest are logged. These events include those that directly relate to performance metrics such as operator intervention (i.e. safety stop), planner timeouts (i.e. inability of the system to determine a suitable path), undesired behavior (e.g. unnecessarily hugging the path edge, unnecessary acceleration or deceleration, etc.), maintenance time (i.e. time spent restarting or correcting capabilities under test), and time and distance spent in autonomous mode. Also logged are events that serve to facilitate root cause analysis such as time and location of autonomous engagement, shifting into forward or reverse, and manually entered notes at any point of interest. When an event – such as operator intervention – occurs, the safety driver resets the system to the next position with nominal functionality of perception and planning components and the trial continues.

The metrics evaluated for continuous nominal autonomy include the number of operator interventions and undesired events, the time in maintenance mode, the time and distance in autonomous mode, and the average autonomous speed.

Segmented Nominal Autonomy In segmented nominal autonomy, a course is divided into smaller segments. Each segment is designed around a core challenge that is designed to stress the system in a specific way similar to an equestrian event. Often, these challenges are designed with a known system weakness in mind. Some of the employed challenges include initializing the system facing the opposite direction of travel for the segment, blocking off the nominal path for a segment, emplaced obstacles such as maneuver course elements, circuitous

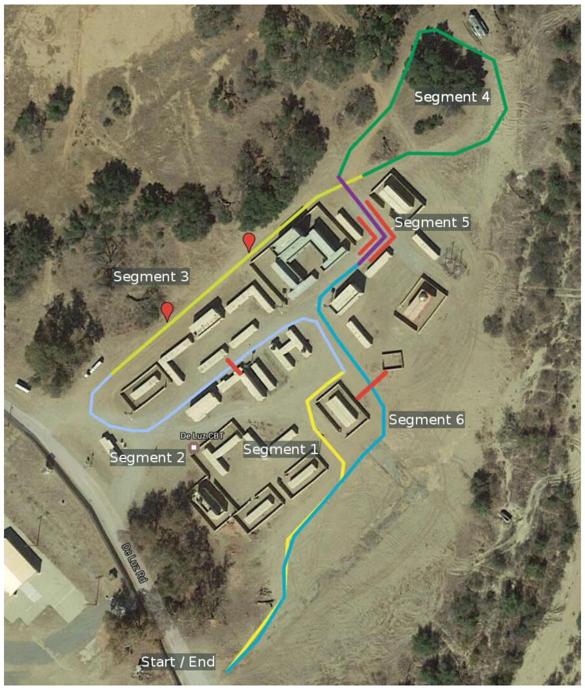


Figure 10. Segmented nominal autonomy course. Red lines indicate walls placed to constrain or block the nominal path. Red pins indicate location of maneuver course elements (e.g. gate-and-dodge and slalom). Map imagery: Google, DigitalGlobe

paths, artificially constraining a path with barricades forcing multi-point turns and congested behaviors, steep or rough terrain, etc.

The segments are continuously run in order with a termination condition of time or number of segments attempted. The same events are logged as in continuous nominal autonomy, but the primary metric is the number of segments completed within time or attempted segments constraint. When conducting the test, if an operator intervention occurs, the current segment is terminated and system is reset to the beginning of the next segment. An example segmented nominal autonomy course is shown in Figure 10. It includes several of the above challenges such as reverse initialization, blocked paths, emplaced obstacles, and constrained path.

Mole Test Tool To facilitate and expedite data collection, metric generation, and analysis of nominal autonomy tests, we developed a web based tool – called Mole – that runs locally on the system. It is accessible both from within the autonomous vehicle and wirelessly from chase vehicles or fixed ground stations. While the original concept was simply to facilitate nominal autonomy, it was designed (and has proven useful) as a flexible continuous event capture system that can run and provide useful data and analysis capability in any test scenario. As shown in Figure 11, the Mole data capture piece is comprised of three principal parts: the event generator, a REST-ful API backend, and the web front end dashboard. In addition to the data capture piece, Mole includes an automated report generation capability. This tool has significantly improved the efficiency and effectiveness of system testing.

Mole is designed around the concept of customizable event triggers. Event triggers define system data fields to be monitored, logic statements that operate on those fields, and metadata sources associated with each trigger. The functionality for these triggers is implemented in the Event Generator. It handles dynamic Robot Operating System (ROS) subscriptions to topics of interest, monitors the appropriate fields within those topics, evaluates trigger logic (consisting of a Python boolean statement), creates events through the web API, and uploads useful metadata to associate with those events. The metadata can include any secondary field or image that gives context to the state of the system at the time of the event. These may include items such as camera imagery, stereo disparity image, system costmap, GPS location, etc. Given the potential for latency in recording events,

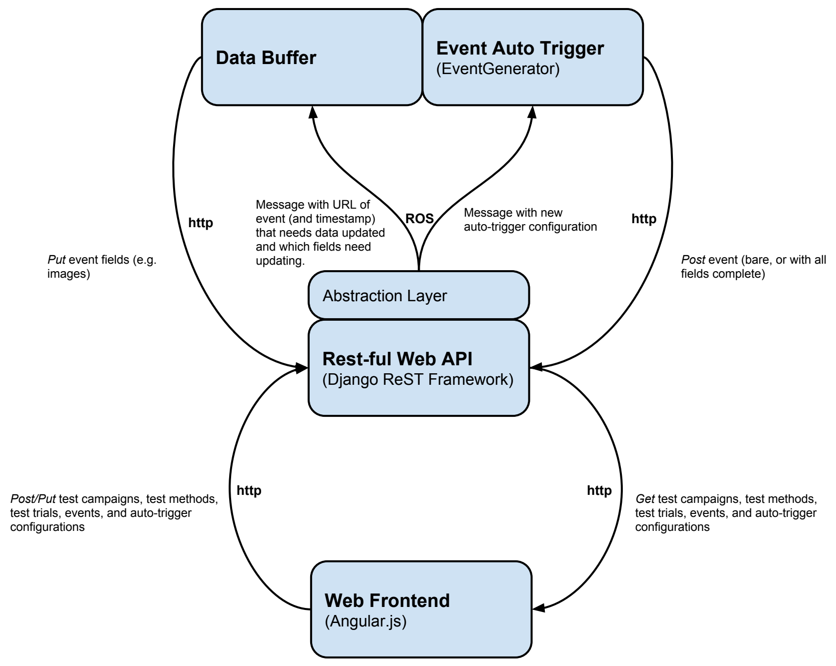

Data Buffer
Event Auto Trigger
(EventGenerator)
http
Message with URL of event (and timestamp) that needs data updated and which fields need updating.
ROS
Message with new auto-trigger configuration
http
Put event fields (e.g. images)
Post event (bare, or with all fields complete)
Abstraction Layer
Rest-ful Web API
(Django ReST Framework)
http
Post/Put test campaigns, test methods, test trials, events, and auto-trigger configurations
http
Get test campaigns, test methods, test trials, events, and auto-trigger configurations
Web Frontend
(Angular.js)

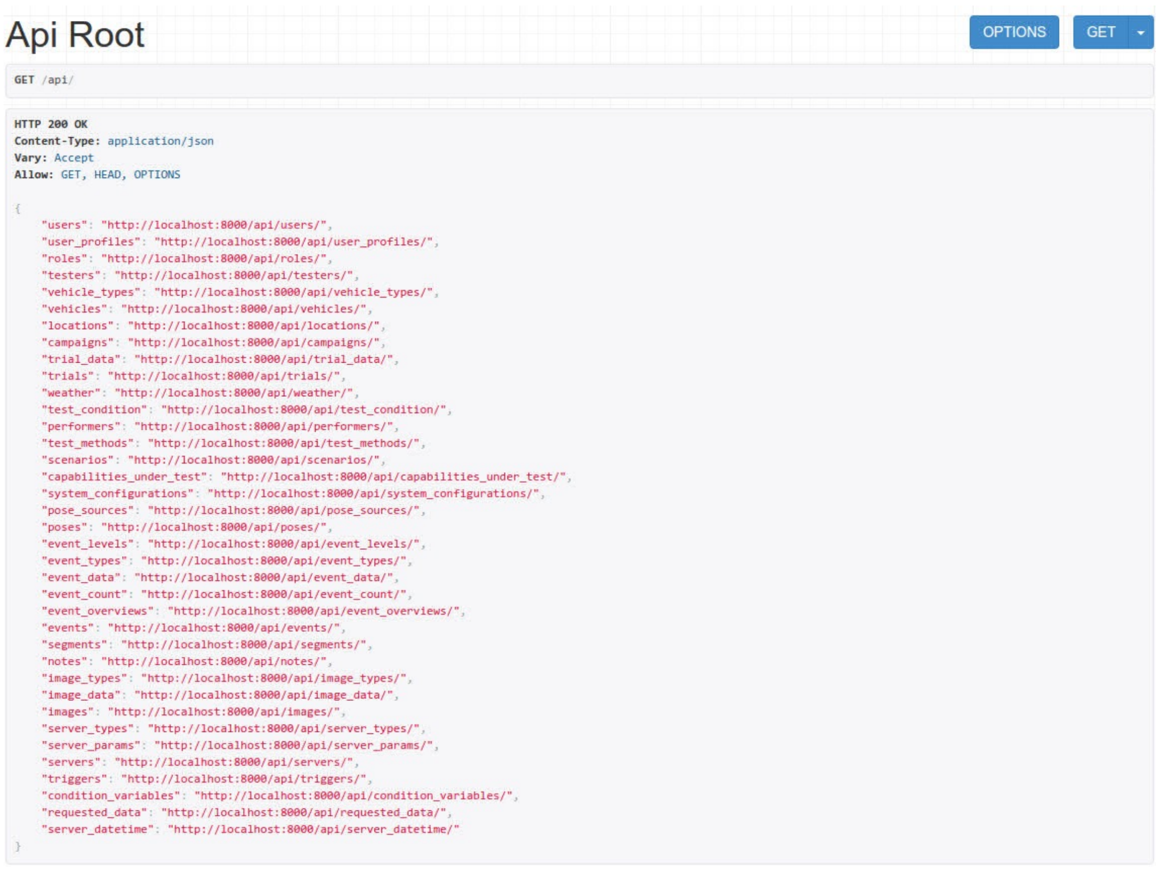
Figure 11. Mole Data Capture Architecture

the Event Generator maintains a rolling cache of this metadata. When an even occurs, this cache is queried for the requested data nearest in time to the event's occurrence. Definitions of the triggers and associated metadata are stored in a database. On startup (and upon configuration change) they are advertised by the REST API backend to the Event Generator.

Mole configuration is entirely database-defined. Vehicles and capabilities under test, test scenarios, event types, event triggers, associated metadata, secondary servers (e.g. for map imagery, video, or system monitoring), and several other parameters are configurable and stored within a database. This allows for great flexibility and ease of switching between test scenarios while providing a consistent ontology that facilitates automated report generation. A REST-ful API provides access to all aspects of Mole configuration and logged data. It is built in Python using Django[5] and the Django Rest Framework[6] (DRF). A browseable API, provided by DRF, greatly simplifies development of both frontend and backend capabilities (see Figure 12).

A web-based test dashboard is developed using AngularJS[7] for interface elements and data binding and Leaflet[8] for overhead maps. A screenshot of the web dashboard for a segmented nominal autonomy trial is shown in Figure 13. It shows the test information in the upper left, aerial imagery with event locations annotated in the lower left, and an event timeline showing events and their associated metadata along the right side. Through this dashboard a test administrator can begin and end trials, monitor the test progress, re-classify or delete events, input manual events, add notes to existing events, and upload additional imagery (e.g. exterior view from a cell phone).

Automated report generation permits analysis immediately following test conclusion. Metrics and figures are automatically generated across any selected trials within a test campaign. A report aggregates and formats the metrics, figures, test information, and additional boilerplate content. The report itself is templated and can be output in a number of formats. The most frequently used of these are pdf document and html. Custom metrics and figures can be easily produced and included within any report.



```
GET /api/

HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
    "users": "http://localhost:8000/api/users/",
    "user_profiles": "http://localhost:8000/api/user_profiles/",
    "roles": "http://localhost:8000/api/roles/",
    "testers": "http://localhost:8000/api/testers/",
    "vehicle_types": "http://localhost:8000/api/vehicle_types/",
    "vehicles": "http://localhost:8000/api/vehicles/",
    "locations": "http://localhost:8000/api/locations/",
    "campaigns": "http://localhost:8000/api/campaigns/",
    "trial_data": "http://localhost:8000/api/trial_data/",
    "trials": "http://localhost:8000/api/trials/",
    "weather": "http://localhost:8000/api/weather/",
    "test_condition": "http://localhost:8000/api/test_condition/",
    "performers": "http://localhost:8000/api/performers/",
    "test_methods": "http://localhost:8000/api/test_methods/",
    "scenarios": "http://localhost:8000/api/scenarios/",
    "capabilities_under_test": "http://localhost:8000/api/capabilities_under_test/",
    "system_configurations": "http://localhost:8000/api/system_configurations/",
    "pose_sources": "http://localhost:8000/api/pose_sources/",
    "poses": "http://localhost:8000/api/poses/",
    "event_levels": "http://localhost:8000/api/event_levels/",
    "event_types": "http://localhost:8000/api/event_types/",
    "event_data": "http://localhost:8000/api/event_data/",
    "event_count": "http://localhost:8000/api/event_count/",
    "event_overviews": "http://localhost:8000/api/event_overviews/",
    "events": "http://localhost:8000/api/events/",
    "segments": "http://localhost:8000/api/segments/",
    "notes": "http://localhost:8000/api/notes/",
    "image_types": "http://localhost:8000/api/image_types/",
    "image_data": "http://localhost:8000/api/image_data/",
    "images": "http://localhost:8000/api/images/",
    "server_types": "http://localhost:8000/api/server_types/",
    "server_params": "http://localhost:8000/api/server_params/",
    "servers": "http://localhost:8000/api/servers/",
    "triggers": "http://localhost:8000/api/triggers/",
    "condition_variables": "http://localhost:8000/api/condition_variables/",
    "requested_data": "http://localhost:8000/api/requested_data/",
    "server_datetime": "http://localhost:8000/api/server_datetime/"
}
```

Figure 12. Mole browseable API root

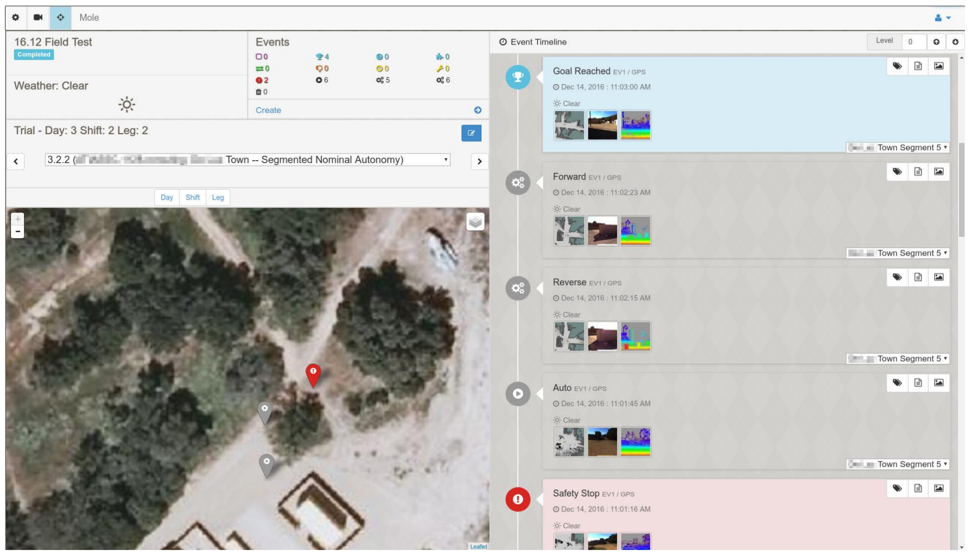


Figure 13. Mole web front end screenshot

4. CONCLUSION

We have described an approach to autonomous unmanned system experimentation that is built on a foundation of decomposing functionality, acquiring evidentiary information, and analyzing component results. It employs – wherever possible – automated, repeatable, reproducible test methods. It is facilitated by a number of software tools. In combination with an integration strategy that defines clear paths for inserting technology into the baseline system, it enables systematically exposing new issues, ensuring that regressions are not introduced, efficiently performing root cause analysis, and managing overall technical risk.

REFERENCES

[1] Koopman, Philip, and Michael Wagner. "Challenges in autonomous vehicle testing and validation." SAE International Journal of Transportation Safety 4.2016-01-0128 (2016): 15-24.

[2] Scrapper, Chris, Ryan Halterman, and Judith Dahmann. "An implementer's view of the evolutionary systems engineering for autonomous unmanned systems." Systems Conference (SysCon), 2016 Annual IEEE. IEEE, 2016.

[3] Dove, Rick, Bill Schindel, and Chris Scrapper. "Agile Systems Engineering Process Features Collective Culture, Consciousness, and Conscience at SSC Pacific Unmanned Systems Group." INCOSE International Symposium. Vol. 26. No. 1. 2016.

[4] Jacoff, Adam, et al. "Standard test methods for response robots." ASTM International Committee on Homeland Security Applications (2010).

[5] Django (Version 1.7) [Computer Software]. (2016). Retrieved from https://djangoproject.com

[6] Django REST framework (Version 3.2.2) [Computer Software]. (2016). Retrieved from https://django-rest-framework.org

[7] AngularJS (Version 1.4.8) [Computer Software]. (2016). Retrieved from https://angularjs.org

[8] Leaflet (Version 0.7.3) [Computer Software]. (2016). Retrieved from http://leafletjs.org