The Edge Architecture for Semi-Autonomous Industrial Robotic Inspection Systems

Roberto Silva Filho, Bo Yu,
Ching-Ling Huang, Raju Venkataramana

AI and Learning Systems Group

General Electric Global Research Center (GE-GRC)

San Ramon, CA

{silva_filho, yu, chingling.huang, venkataramana}@ge.com

Ashraf El-Messidi, Dustin Sharber,
John Westerheide, Nasr Alkadi

Baker Hughes, a GE Company (BHGE)

Oil & Gas Technology Center

Oklahoma City, OK

{ashraf.el-messidi, dustin.sharber, john.westerheide, nasr.alkadi}@bhge.com

Abstract: Robots have been increasingly used in industrial applications, being deployed along other robots and human supervisors in the automation of complex tasks such as the inspection, monitoring and maintenance of industrial assets. In this paper, we shared our experience and presented our implemented software framework for such Edge computing for semi-autonomous robotics inspection. These systems combine human-in-the-loop, semi-autonomous robots, Edge computing and Cloud services to achieve the automation of complex industrial tasks. This paper describes a robotic platform developed, discussing the key architectural aspects of a semi-autonomous robotics system employed in two industrial inspection case studies: remote methane detection in oilfields, and flare stack inspections in oil and gas production environment. We outline the requirements for the system, sharing the experience of our design and implementation trade-offs. In particular, the synergy among the semi-autonomous robots, human supervisors, model-based edge controls, and the cloud services is designed to achieve the responsive onsite monitoring and to cope with the limited connectivity, bandwidth and processing constraints in typical industrial setting.

Keywords: semi-autonomous robotics, remote methane leak inspection, Unmanned Aerial Vehicle (UAV), HMI (Human Machine Interface).

Reference to this paper should be made as follows: Roberto Silva Filho et. al. ‘The Edge Architecture of Semi-Autonomous Industrial Robotic Inspection System’, *Int. J. Cloud Computing*,

Biographical notes: *This paper is a revised and expanded version of a conference paper entitled “Semi-Autonomous Industrial Robotic Inspection Part I: Remote Methane Detection in Oilfield” presented at* ***IEEE EDGE 2018, in San Francisco, on 07/03/2018****.*

1 Introduction

Robotics have been increasingly used to automate dirty, dull and dangerous jobs in different industrial scenarios. They can withstand harsh environmental conditions, can operate over long hours without fatigue, performing many tasks that may seem tedious, risky or unsafe to human workers. Examples include the inspection of turbine and pipeline interiors, climbing and inspecting industrial asset walls, cleaning and repairing internal components of heavy equipment. The mobile computational nature of robotic inspection systems require real-time data processing and communication which makes it a perfect application for Edge Computing [1-3]. In this paper, we present the main components of an Edge Computing Architecture [7] of a model-driven robotic UAV system applied for industrial inspections, showing its use in different scenarios.

While high levels of robotic automation and autonomy have been achieved in controlled environments as industry production lines [4], many industrial tasks still require human supervision. They require operators that constantly assess the quality of job being performed, the data being collected, and the overall progress of the robot in its tasks. In particular, the ability to detect and act upon exceptional situations is very important during robotics supervised operation. For example, Aerial Unmanned Autonomous Vehicles (or UAVs) can suffer mechanical failures, can be subject to sudden weather condition changes that may get them out of planned route or trapped into trees; UAVs also require periodic maintenance including battery recharge, parts and payload exchanges. In another example, Inspection crawler robots may have their cables stuck on engine parts, may require periodic cleaning and calibration of tools and sensors and may need to be untangled from difficult stops within assets.

In this paper, we focus on semi-automated inspection of industrial assets [5]. During these inspections, robots work under human supervision in the detection of structural and functional problems involving oil pipelines, power generation turbines, containers, reactors and others. Robots interact with assets through non-destructive probing: measuring, sensing and photographing, e.g., detecting worn materials, sensing heat, gas leaks and unusual vibrations. Every piece of equipment has its own characteristics, parameters and configuration that make each asset installation unique. While humans can easily cope with these variations, robots must be pre-configured, guided through, or taught about these differences, conditions and restrictions to perform proper inspection, and detect conditions that need intervention as repair or replacement.

Humans are also ultimately responsible for the quality of the job being performed, the data being collected, and the overall progress of the inspection work. Most importantly, the need to detect and act upon exceptional situations is one of the main reasons constant human supervision is still required. Finally, as any mechanical asset, robots need periodic maintenance including cleaning, battery replacement, part exchange, and calibration. Hence, while robotic autonomy has been increasing over the years, especially in the consumer space, industrial operators must still supervise activities of robots, ready to assist when in-situ exceptions occur.

A common problem faced by robotic inspection in industrial space is limited network connectivity. Industrial sites such as oil rigs, power plants, ship engine rooms, and wind turbines are typically located in areas of difficult access, with poor cellular connectivity, which may limit the ability of the robotic system to communicate with external computing resources. The ability to cope with intermittent or poor network connectivity is therefore essential requirement for robotics inspection systems.

Other important limiting factors in robotics systems are their reduced footprint and power. These constrains pose a limit to the amount of computing robotic systems may have. For example, Aerial UAVs must be lightweight and small enough to fit restricted industrial areas, e.g., within turbine casings and pipes, and the onboard battery power must be shared between the onboard computing and its propulsion systems. As a consequence, while certain amount of autonomy can be embedded in the vehicle itself, e.g. through GPS (Global Positioning Systems), object and movement detectors, computer vision, and other techniques, large part of computation necessary for its operation needs to be performed elsewhere: either remotely, at real-time, or pre-computed offline. A good example comes from autonomous car systems that rely on high-fidelity models of the road environment. These models are pre-computed, in the cloud, based on large amounts of data collected over time [6]. Similarly, industrial model-driven approaches, as the one described in this paper, require the development of accurate asset models, based on extensive site survey and operations planning. These models support an optimization step where the most efficient robot route is calculated using 3D models of the asset and site.

Due to these combined set of factors, robotics inspection systems require a careful balance between autonomy, task planning, and real-time controls that generally combine cloud, edge computing and humans. In this paper, we present the Edge Computing Architecture [7] of a model-driven robotic UAV system used for industrial inspections, showing its use in different scenarios.

This paper is organized as follows. In Section 2, we first present the requirements of the system in the context of industrial inspection scenarios; we then show how to address these requirements using a system that combines UAVs, base satiation, cloud services and human operators. In Section 3, we describe the basic interfaces between major system components, discussing its design trade-offs and implementation details. We then validate the approach showing how it has been used in two industrial applications: 1) the detection of methane leaks in industrial oil & gas facilities (in Section 4), and 2) the inspection of flare stacks in oil refineries (in Section 5). Finally, we conclude with a short summary of the approach and potential future work.

2 Requirements

Industrial asset inspections are critical to the long-term operation of complex industrial systems. They enable early detection of operational anomalies and decay that may lead to failures and breakdowns; they inform maintenance crews on the parts of the system that must be serviced or replaced, thus preventing unforeseen downtime; Inspections are also required after major overhauls, as a verification step, before bringing the asset back into production. Robotics-assisted inspections provide repeatability and automation to these tasks. In order to be successful, industrial assets industrial inspections typically follow three major stages: Planning, Execution and Reporting. In this section, we specify functional requirements for these three stages and the non-functional requirements (NFRs) of our design.

Inspection Planning

Adequate planning supports the efficient execution of inspection procedure, minimizing its downtime, while supporting accuracy and thoroughness of the process by identifying critical areas of interest. E.g. in typical overhaul of flare stacks and power plant turbines must be brought offline, cooled then taken apart in operations that may last from hours or days. This downtime disturbs regular production and incurs in considerable costs. In order to achieve maximum efficiency and mitigate asset downtime, industrial inspections require careful preparation and planning.

Robotics systems must operate under well specified guidance of software control systems that account for different environmental characteristics and task goals. They require careful preparation and planning before their deployment in different missions. During inspection planning stage, a detailed execution plan is produced. The plan varies, depending on the type of asset, the data to be captured (e.g., videos, photos, ultra-sound, sensors), the type of robots to be used (e.g., UAVs, crawlers, submarines), and the specific challenges of the industrial location where the inspection will be performed (e.g., known obstacles, hovering area, magnetic characteristic, predominant wind, wireless interferences). In particular, the inspection planning system UI shall provide means for operators to experiment with certain parameters and eventually produce the following artefacts as the inspection plan:

- **Environment model**: Similar to a road map in an automotive GPS navigation system, the environment model captures and represents the relevant characteristics of the space where the robots will operate and traverse. It may include data such as weather and environmental conditions, e.g., predominant wind and precipitation, terrain, area of operation, no-fly zones.

- **Asset model**: This is high-fidelity digital representation of assets and reside within the overall environment model representations. They define the exact location (with respect to the environment), and the dimensions (boundaries and structure) of the industrial assets that will be inspected. This can be 2D or 3D model, depending on the type of inspection to be performed.

- **Execution plan**: specifies the trajectory to be followed by the robot around the assets within the environment. It also includes tasks and maintenance actions to be performed by each robot along the trajectory and the assets. In particular, the plan defines a sequence of tasks with data to be collected and the robots/tools to be used in each step.
    - **Data capturing configuration:** identifies the sensors to be used, the type and quality attributes for the data to be captured in each step of the workflow model;
    - **Robot and tools schedule:** define the robots to be used in the inspection and the tools they must carry. Note that multiple robots can be used in an inspection site, allowing for different task scheduling including serial and parallel involving robots, their sensors and tools. Multiple robots also allow divide-and-conquer of tasks by location and allow more efficient use of sensors and tools. E.g., one robot can

perform RGB video inspection, while another one can perform infra-red (IR) video inspection of a site. These decisions and schedules must be part of the plan.

Once the execution plan details are specified, further optimization is typically performed to achieve certain objectives, e.g., minimizing the overall inspection time, energy, the materials required, or a combination of multiple factors. These include computationally-intensive simulations considering: the environment, asset models, and the characteristics of each robot. Typically, this plan is reviewed and approved by the human supervisor before it gets approved for execution.

Inspection Execution

During the inspection execution, robots follow the inspection plan in a semi-autonomous way, working along with human supervisors and auxiliary systems. As in any model-driven approach, unforeseen conditions and exceptions must be detected and gracefully handled. The handling of unforeseen exceptions is typically performed by human workers. In particular the system Human-Machine Interface (or HMI) plays a key role in the communication and resolution of exceptional conditions.

- **Supervised plan execution**: During normal plan execution, situational awareness is key for the proper monitoring of the process. During this stage, system HMI must keep the operator informed of the location, progress and quality of the work of the robots. In particular, the system operator should be constantly informed of the following:
    - **Robot location**, with the precise position of the robots with respect to the environment and the assets being inspected.
    - **Plan progress**, which tracks the current progress of the plan and status of the robots, including information like finished tasks, latest checkpoint, time-to-finish and elapsed time;
    - **Data quality**, which allows the operator to detect any problems with the data being collected, e.g. sensor precision, image sharpness, exposure, and other data characteristics. A typical HMI provides a timeline of recent data samples captured along the inspection for quick visual inspection. Additionally, the system may use automatic data verification algorithms for flagging quality issues, and to detect sensor value anomalies.

- **Exception handling**: During supervised execution, exceptions may be detected by the system or by the operator via existing monitoring mechanisms When an exception is detected, the human supervisor must be able to intervene on the operation of the robot performing corrective actions. In order to facilitate the detection of problems during the inspection, the system HMI must support:
    - **Exception notification** mechanisms to immediately inform the operator of data and plan execution issues, e.g. by means of visual, audible and tactile alerts;
    - **Manual override or emergency takeover** allows the human supervisor to take full operation control of the robots at any time during the inspection;
    - **Macro functions** allow users to perform commonly used or preprogrammed tasks. These operations facilitate the resolution of common exceptional situations. For example:
        - Autonomously navigate back home or starting location;
        - Autonomously keep position;
        - Gracefully (land and) shutdown;
        - Pause, roll back, resume last task;
        - Pause for battery exchange or repair;
        - Resume from a certain task or checkpoint in the plan.

Inspection Reporting

An important part of the inspection process is the generation of reports that summarize the findings during the process. In order to minimize data loss and mitigate common mistakes, it is important to integrate certain reporting capabilities into the inspection execution stage. The earlier the report is produced, the better chance it has to represent the findings of the inspection. Reporting also benefits from further data refinement, processing and exploration. For example, during the inspection, the user may decide to re-visit defective parts to gather close-up photos and measurements. Hence, reporting can be facilitated by augmenting the execution stage with features such as:

- **Operator-driven field notes**, taken as the inspection takes place.
- **Ad-hoc data collection fly-overs**: Optional data collection (outside of original plan) gathering additional data that may be crucial in determining the asset condition.
- **On-the-fly interpretation of collected data**: Based on the data collection, the system may allow basic processing of the asset data as it gets collected. For example, AI-based detection of features or fissures. This is typically done in the edge or, if network connectivity is available, can also be performed in the cloud if connectivity allows.

After the inspection, the system must support further summarization and analysis of the data, in particular:

- **Processing and interpretation of the collected data:** Inspection and refinement of collected data from the robots, for example zoom-in of important picture notes, cropping of videos, drawing on top of pictures. Relevant data statistics out of raw sensor data can be attached. Based on the data collection, basic interpretation of the data and asset status along with recommended actions / next steps.
- **On-site report generation**: The faster one can generate reports, the cheaper the inspection process it is for customers and inspectors. Hence, whenever possible, the final report should be generated on-premise, with minimum use to cloud services by using cached and pre-loaded templates.

Non-Functional Requirements

Besides satisfying the above functional requirements for user supervision and control of one or multiple drones in inspection planning, execution and reporting, we argue that the inspection system must support the following non-functional requirements.

- **Multi-robot management**: The system shall have the ability to simultaneously supervise and control one or multiple robots of different types (drones, crawlers, rollers, etc.). As the complexity of inspection tasks increases, multiple robots may be used in sequential or parallel activities, performing similar or complementary tasks. In these situations, the system must support operators in their situational awareness and exception handling. There is usually a central control station that communicates with all robots at the same time to coordinate, dispatch inspection tasks and avoid interference/collision among robots. For example:
    - Robots can be used in parallel, collaboratively inspecting multiple and complementary parts of an asset, or independently, working in different tasks of the plan at the same time.
    - Robots can be used in sequence, with specialized tools and sensors, performing multi-pass inspections. E.g. infra-red (IR) camera scans followed by RGB scans of an asset.
    - Hybrid approaches, incorporating both parallel and sequential plan execution are also possible.

In these situations, the system HMI must support the operator with situational awareness and exception handling involving multiple robots. If exceptions shall occur, requiring human intervention, other robots may either continue or pause their activities until the operator is able to resolve the exception, as a way to prevent parallel exceptions.

- **Customizability**: The capabilities required by inspection applications may vary according to the mission, types of robots, assets and different phases. Instead of a one-size-fits-all approach, the system must provide a flexible platform that allows the customization and rapid prototyping for different inspection scenarios. For example:
    - Inspection of flare stacks in oil refineries require the flight of UAVs over points of interest to take IR and RGB photos, with a route in a 3D space around an asset in a restricted area [8].
    - UAV gas leak inspection requires the planning and flight of semi-autonomous airplanes over oil pad/wells and along gas pipelines, the real-time visualization of gas leak sensor, and the ability to navigate the plane around a long asset over a fixed altitude 2D plane [9].
    - Inspection of power generation turbine internals using crawler robots requires the real-time visualization of ultrasound images, navigation over a tridimensional asset, and inspection of videos and images [5].

- **Interoperability with different robotics platforms:** In order to support the proposed variety of robots and inspection scenarios, the system must be able to interface with and control robots of different vendors. For example, supervision of sensor data and remote control must be able to operate and communicate with robots, in similar ways over common interfaces and protocols like ROS (Robot Operating System) [10] and MavLink (Micro Air Vehicle Link). The system shall also provide support for different mobile platforms (iOS, Android) as well as the ubiquitous Web browser UI (which can run on most of PC and mobile devices). The control of robots largely benefits from the mobility and portability of today's wireless phones and tablet computing platforms. Moreover, Web UI allows the collaboration with remote stake holders.

- **Mobility & Portability:** The inspection of industrial assets is usually performed at the asset site. Hence, robotic inspection systems must be inherently mobile and portable. They must also cope with limited network connectivity of these sites and can perform the inspection plan offline.

3 Design

In this section, we present our robotic inspection platform and HMI, showing how it supports the aforementioned requirements. The main system components include: a base station providing edge data storage, media services (video, photos) and onsite plan execution; a mobile HMI on a tablet computer, coupled with a remote controller; a collection of robots that execute the inspection; and cloud services providing additional processing power & capabilities.

Logical System Architecture

Fig. 1 presents the general architecture of the robotics inspection system. The system is composed of several sub-systems:

- **Mobile devices:** Apps are developed based on cross-platform development for iOS, Android, and Web, in particular we use Apache Cordova [12] and Ionic framework [13]. Depending on the application, the mobile Apps use a selected set of HMI components to support the three stages from inspection planning, execution to reporting. These HMI components will be later described in more details.

- **Base Station:** Several integrational supporting servers located on this Base Station (or commonly called the Ground Control Station in UAV framework). The Base Station is a common portable, onsite computing support to work with robots from different vendors with platform-specific adapters, e.g., ROS [10] or MavLink [11] protocols. Additional media servers on the Base Station can support multimedia streaming including video and audio communications from the robots to the onsite human worker and additional remote observers (e.g., the manager in office). Depending on the inspection application, the Base Station can also offer additional computing power and the stability to support the HMI on mobile devices. E.g., sensor data processing, aggregation, and analytics can be done in Base Station before uploading to the Cloud.

- **Cloud Services:** Cloud services are used in support of large dataset collection and processing. They support relational and file data storage, provide image recognition, data analytics, and other computational-intensive services. Cloud services are typically consumed by system components via REST APIs [14].

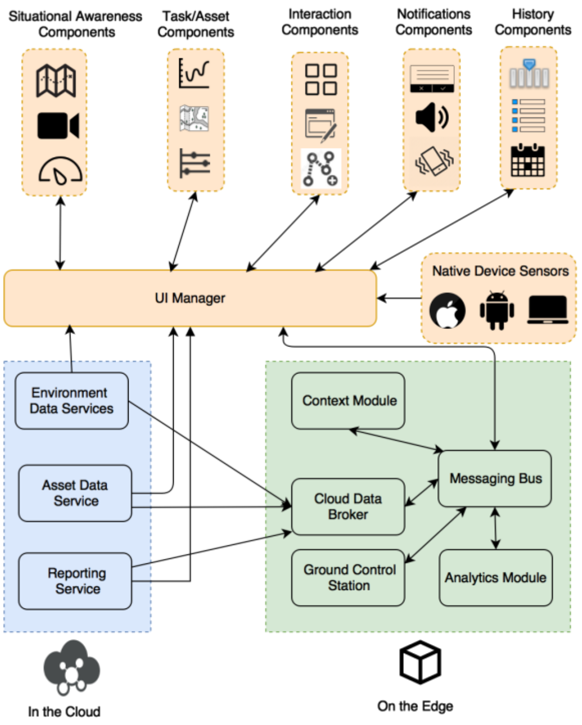


Fig. 1 General logical system architecture.

Physical System Architecture

Fig. 2 shows one realization of the logical system architecture, based on a prototype robotic inspection system from [8], where drones are used to photograph assets. In this example, the RC controller is integrated with the mobile device. The Base Station is a laptop or a portable PC to support more processing power and server as the onsite communication hub.

The inspection planning is done using the Base Station laptop, where the user can specify the flight route for robots, the assets to cover and the inspection tasks to perform. The execution stage is supported by the mobile device HMI (integrated with the RC controller) running an application that allows users to monitor the inspection progress, review photos taken along the flight route and take emergency control of the robots. Additional image analytics is available is the cloud and can be accessed when network connectivity is available. After the inspection, the reporting is performed with the help of the same Base Station laptop.

The mobile characteristic of the system requires both local and wide area network connectivity. Robotics protocols can run on wireless networking layers like Wi-Fi. Meanwhile, intermittent connectivity with cloud services, via the cellular network, to the cloud is assumed. When the sensor data or photos are collected from the asset, the data exploration or image analytics happens along the way from the robot’s onboard computing, mobile device, Base Station, and then to the Cloud Services. Due to required real-time monitoring by the human supervisor and the limited to no connectivity, the data processing is preferred to happen progressively on the robot, then on Base Station or mobile device, and then on Cloud Services if possible. This approach provides the responsiveness of onsite processing, at the edge. However, due to the limited computing power onsite, the data is usually processed preliminarily at the edge to satisfy real-time monitoring and then comprehensively on the cloud to achieve the best results. As a balance of cloud vs. edge computing, the overall architecture and functionality of each components and HMI must be designed with flexibility for the inspection application to work smoothly in the field.

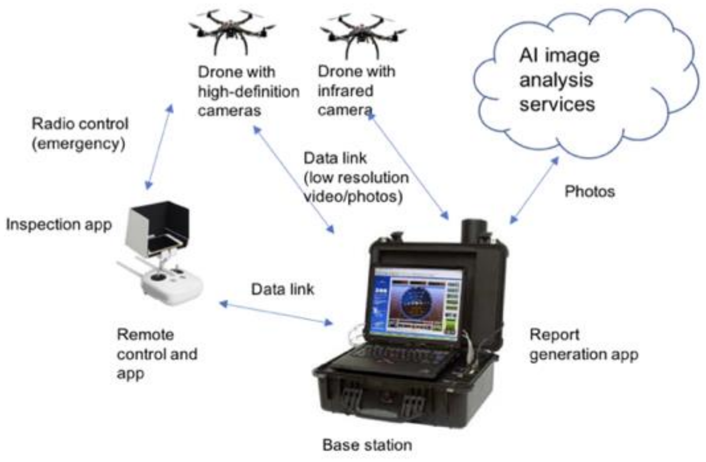

Drone with high-definition cameras
Drone with infrared camera
AI image analysis services
Radio control (emergency)
Data link (low resolution video/photos)
Photos
Inspection app
Data link
Remote control and app
Report generation app
Base station

Fig. 2 One realization of the architecture and entities in Fig 1.

Human-Machine Interface Components

Our human-in-the-loop robotic inspection platform assumes constant operator supervision. As such, it must provide adequate situational awareness through a set of user interfaces tailed for inspection, planning and reporting. In particular, we have developed a set of HMI components to support five main scenarios as depicted at the top in Fig. 1. Please also see Appendix A for examples of discussed HMI components.

- **Situation Awareness Components** provide human supervisor with location and operational insights. For example,
    - **Status bar** for showing the status of robots, flight metrics, battery levels, etc.
    - **Map view** to show the real-time positions of the robot with respect to the assets and overall environment, usually rendered in 2D or 3D view.
    - **Live video** to provide the real-time video feed from the robot’s point of view (POV). The user can maneuver the robot and switch between video feeds when multiple cameras are mounted on the robot.

- **Task / Asset Analytics Components** provide information related to the inspection tasks or the inspection workflow status. The user utilizes these components to make sure that the robot performs according to the plan. For example,
    - **Inspection progress bar** with main inspection checkpoints, or as a route around the asset, where different colors indicate completed tasks, remaining tasks, missed spots, etc.
    - **Heatmap** showing real-time visualization of collected sensor data, quality, warnings, etc. This heatmap can be overlaid on top of the map view or rendered in a separate view.
    - **Analytics model visualization** showing the statistics and parameters behind analytics models and data visualization.

- **History components** allow users to go back in time for data quick data and task quality analysis. For example,
    - **Timeline bar** shows the recently captured sensor data, photos or video, so the user can navigate along the timeline, browse the data, and keep track of key inspection events. These data points may be organized as a sequential timeline view and tagged with the tasks performed. If network connectivity is available to access Cloud Services, additional asset records from historian or database can be shown, e.g., past maintenance records, previous repairing, replaced components.

- **Notification components** allow the HMI to produce multi-modal alarms and notifications to the inform the user of exceptions and important conditions. For example,
    - **Color-coded notification panel** to visually indicate important events with different levels of severity and health warning of the robots and assets.
    - **Audible alerts or text-to-speech notifications** to deliver the notification via audio in the case when the user cannot look at mobile device and has to work on the job with both hands.
    - **Vibrations** as a subtle way to communicate with the human worker. A combination of above method can be used, depending on the type of inspections.

- **Interaction components** support additional functions for user to quickly re-configure the robot, handle exceptions, add to the inspection plan to collect more data points, etc. For example,
    - **Quick action bars** to support the App setting and implementation of major configurations. Additional macro functions to control the robots can be used throughout various scenarios to allow the fast response to emergent conditions like "interrupt and resume" of current task and "go back to home" to return home location for battery change.
    - **Robot path editor or configurator** to support onsite ad-hoc movement planning for robots like UAV flight route update.

Base Station Components

The base station computing platform aggregates a variety of hardware and software edge components to coordinate interactions and communications among UAVs, user interfaces, and the cloud services.

- **Ground Control Station (GCS)** handles the two-way communication with UAVs via standard robotics protocols, such as ROS [10] or MavLink [11]. On one hand, user requests from mobile apps or results from the **Analytics Modul**e are translated into standard robotics messages and sent to UAVs via radio link. On the other hand, parameters and sensor data are received from UAVs and pushed to **Messaging Bus** for further consumption.

- **Messaging Bus** provides the real-time communication channel that can be subscribed and published from mobile apps or other components in the base station. The different interface components, as we described earlier, may subscribe to different types of events to support situation awareness in real-time monitoring. They may send control commands to the UAVs via the messaging bus as well. The messaging service is also consumed by other components inside the base station to coordinate behaviors. For instance, the **Analytics Module** listens to the sensor data collected from the UAVs, builds the online analytics model, then publishes the model parameters back to the messaging bus, which is further received by the **Reporting Module** to generate reports and save to the cloud.

- **Analytic Module** offloads the computing overhead from the UAVs to the edge box. It ingests and integrates raw sensor data from multiple UAVs and builds the online analytics model that is used to dynamically optimize UAV flight path, update sampling rate, and coordinate the UAV swarm.

- **Cloud Data Broker** serves as the broker between the base station and the cloud services. It subscribes to the real-time data stream from the messaging bus, synthesize it, and periodically saves the data to the cloud at the background. In addition, it provides the local data persistence layer to provide fault tolerance due to unstable Internet connections.

- **Context Module** provides the context model to capture the relevant information for user interaction, and the inference engine to reason about the user's current activity and situation, which is consumed by the UI manager to provide UI adaption and customization. In the current implementation, the Context Module is driven by a Finite State Machine to infer the current phase in the inspection process based on a set of UAV and environmental variables. In the future work, we plan to experiment with more complex context awareness models [15] to provide more fine-grained customization capabilities.

Cloud Services

Our inspection platform utilizes a set of cloud services to produce information consumed by the HMI as well as the base station components. In general, we summarize them into three categories:

- **Asset Data Services** are the gateway to existing data repositories of the inspected assets themselves. This usually includes the basic asset information, historical data regarding its operation and inspection schedules.
- **Environment Data Services** are external services to fetch environmental data, such as weather condition, wind direction and speed. This data is critical in UAV inspection to optimize the parameters and flight path, as well as to adjust the parameters in the analytics model.
- **Reporting Data Service** is used to store the inspection results to the cloud and also allows the mobile apps to retrieve historical reports.

In the next two sections, we show how our robotic architecture has been used in support of two case studies: methane leak inspection (Section 4) and flare stack inspection (Section 5).

4 Case Study I: Methane Leak Inspection

Raven [9] is a UAV inspection platform developed and funded by Baker Hughes, a GE Company (BHGE), with the help of GE Digital Research team in San Ramon. Raven uses UAVs to automate the field data collection/detection of methane leaks in oil & gas facilities. The use of UAVs automates the otherwise labor-intensive task performed by human inspectors that currently have to walk along facilities and pipelines, using Optical Gas Imaging (OGI) devices seeking for potential methane gas leaks. This method is time-consuming and the data provided by the measurement devices is not very precise: it only provides a binary (yes/no) qualitative indication of leakage with no specific measurement of its intensity. Additionally, the OGI devices are expensive and heavy, making it difficult for operators to deploy them at-scale. Raven was designed to replace this procedure with UAVs equipped with sensors, mobile Ground Control Stations (GCS) and smart mobile applications. The Raven inspection process depicted in in Fig. 3.

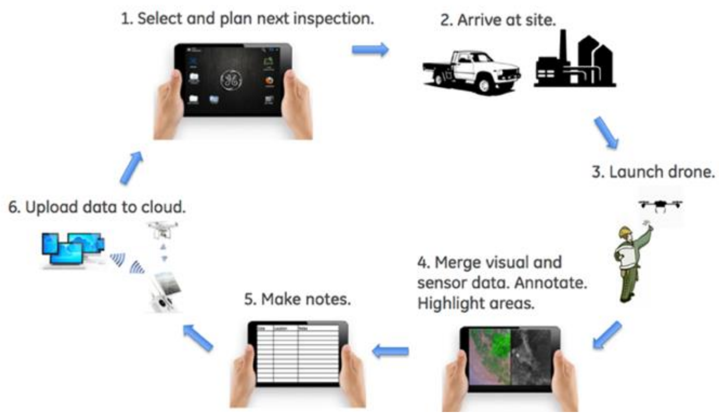

1. Select and plan next inspection.
2. Arrive at site.
3. Launch drone.
4. Merge visual and sensor data. Annotate. Highlight areas.
5. Make notes.
6. Upload data to cloud.

Fig. 3 Workflow for Raven remote methane inspection in oilfield.

In Step 1 of Fig. 3, the field engineer browses existing oil facilities, reviewing past inspection reports and decide which assets to inspect. Based on the site map and layout, the engineer plans the flight route for the UAV to collect methane data and detect leaks. Upon arriving at the site in Step 2, the engineer reviews the inspection plan and connects the mobile device with the UAV via the GCS (Ground Control Station). Typically, supporting servers and GCS are located in the engineer's truck. Once the mobile device connects with the UAV, the inspection plan will be uploaded to UAV's navigation controller. In Step 3, the engineer checks the status of UAV before launching it.

In Step 4 (execution), the engineer uses the mobile device to monitor the inspection progress, the current methane readings, and the status of the UAV. In Step 5, the field engineer can review the summary of collected data in statistics and visualization, and put in additional remarks about this inspection. Finally, in Step 6, a report will be auto-generated and all the data will be uploaded to the Cloud Services for further processing and archiving. These 6 steps complete a typical methane inspection cycle for Raven.

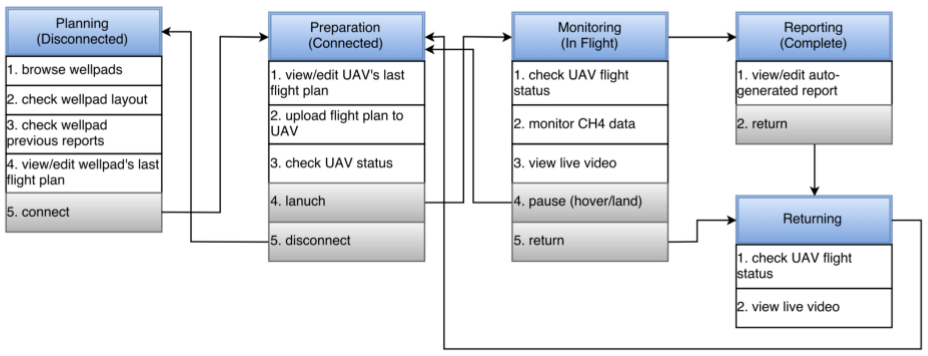


Fig. 4 Mobile App workflow for Raven inspection.

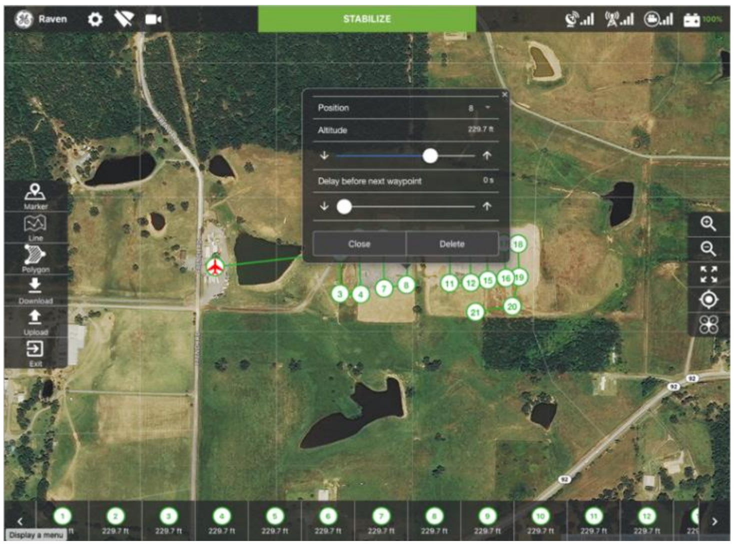
The Raven mobile application workflow is depicted in Fig. 4 in more details. In the "Planning" stage, the field engineer browses existing oil facilities in cloud database and decide which oil pad / wells to inspect. The engineer can read past inspection reports of this site. Then, based on the site map and layout, the engineer plans the flight route with a sequence of way-points for the UAV to visit, hover for a defined duration and collect methane data to detect leaks. At this planning stage, the mobile device is not connected with the UAV and thus this can be done before visiting the site. However, this planning stage can also happen at the site.

Typical Methane Inspection Scenario

This section describes the user experience of Raven methane inspection app, highlighting how the system supports users along the main inspection stages of: planning, execution and reporting.

Planning: During inspection planning, the engineer browses all the oil facilities registered in the system cloud database. Once the site is selected, the user is provided with a site map on top of which the plan is constructed. As shown in Fig. 5, the inspection plan consists of sequence of geo-located way-points. The user adds one way-point at a time by clicking on the map and specifying the way-point's altitude (height), and the hovering delay before moving to next way-point. This delay translates into the number of sensor data points collected at this way-point. The engineer can use finger gestures to move and adjust each way-point on the map. Once complete, the plan is compiled into a set of UAV instructions. In particular, way points are translated into GPS coordinates. Optionally, the way-points can be generated by specifying an area in the map. The system will then create an optimal schedule to cover the selected area.

The flight plan is designed in such a way that allows for sufficient coverage over the entire facility infrastructure. Although there are known higher risk pieces of equipment where leaks generally occur, they can be found at any point of the industrial system assets where gas travels. As such, the flight path planning is a critical step in the overall process in ensuring that a thorough survey is conducted.



Position 8
Altitude 229.7 ft
Delay before next waypoint 0 s

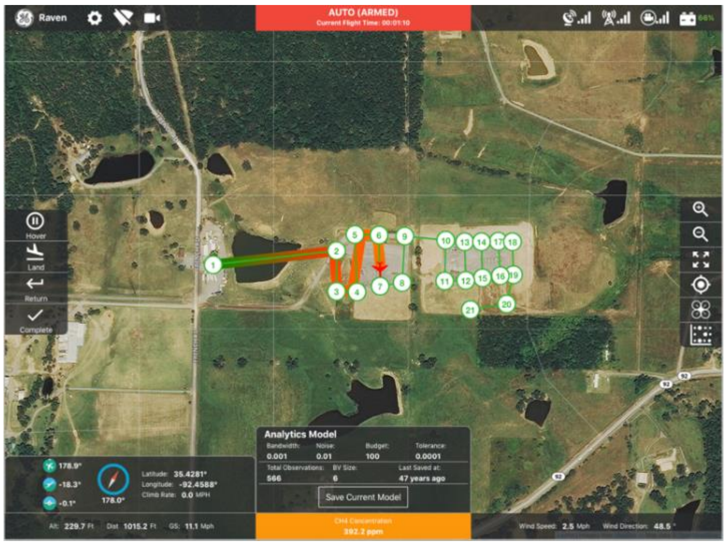
Fig. 5 Plan the UAV's flight route as a sequence of way-points.

The planned way-points are also listed in the bottom of the screen in visiting sequence order. By clicking on these existing way-points, the engineer can adjust these parameters until the overall flight route is ready. On the right side of the screen, there are a few buttons to change the zoom level and perspective of the background map. Once the flight route is ready, the field engineer can upload the flight plan to UAV's onboard navigation controller. The field engineer can also download the existing flight plan from the UAV for further editing (and then upload the flight plan back to the UAV for execution).

Another important inspection parameter is speed. The appropriate speed of UAV along the inspection path is dictated by the measurement frequency of the sensor. Flying too slow may significantly increase survey flight time, resulting in a lower time and cost-savings to the operator, while flying too fast would result in sparse measurement points/data points across the flight path, ultimately leading to an inaccurate final output. Because the methane detector utilizes a laser-based sensor, it operates by taking an average concentration measurement across the length of the laser beam. Therefore, flying as low as safely possible is key in increasing the overall measurement accuracy. The flight altitude is typically governed by the tallest structure on-site, as operators generally require that the UAV flies at least 20 feet higher than the tallest structure in the inspection area.

Execution: During inspection execution, there is a slightly change in the HMI as seen in Fig. 6, the app uses the same map view but now uses the waypoints to track the inspection progress. The top of the screen now shows situation awareness components like UAV status bar with indicators of critical UAV system operating conditions. The UAV current mode is shown in the top, middle banner, along with satellite signal, communication signal and battery level. On the top left, there are a few buttons for app settings, UAV connectivity and video streaming mode.

In the middle portion of the screen, the sensor data heatmap and flight route are both overlaid on top of the satellite map. The flight route consists of a sequence of way-points with their sequence numbers. The heatmap shows color-coded methane concentration based on sensor data. In this example, red means high concentration of methane (>400 ppm), yellow means medium concentration (200 ~ 400 ppm) and green means low concentration (<200 ppm). The left side of the screen shows some Macro buttons for emergency UAV takeover functions, e.g., "hover in mid-air", "land immediately", and "return to the start location".



AUTO (ARMED)
Current Flight Time: 00:01:10

Analytics Model
Bandwidth: 0.001 Noise: 0.01 Budget: 100 Tolerance: 0.0001
Total Observations: 566 BV Size: 6 Last Saved at: 47 years ago
Save Current Model

178.9° -18.3° -0.1° 178.0°
Latitude: 35.4281° Longitude: -92.4588° Climb Rate: 0.0 MPH
Alt: 229.7 Ft Dist: 1015.2 Ft GS: 11.1 Mph
CH4 Concentration 392.2 ppm
Wind Speed: 2.5 Mph Wind Direction: 48.5 °

Fig. 6 Monitor the inspection and review captured data in real time.

The left bottom portion shows more details of the UAV's current status, including coordinates, flight heading, speed and climb rate. The right bottom portion shows the environment condition like site wind speed and direction. The center bottom portion shows the methane analytics model that takes collected sensor data as inputs in real-time and then statistically fits the methane distribution in a Gaussian normal distribution. Latest methane sensor value is displayed here with colors matched with those on the heatmap. The outputs of the analytics model, as the probability distribution parameters, are also recorded and will be uploaded to the cloud services after the inspection.

In the middle bottom of the display the user can save the outputs of the analytics model, as the parameterization of the probability distribution. When selected, the data is uploaded to Cloud Services for archiving. The field engineer can also view and archive the live video streamed from UAV's onboard camera. An infrared camera can also be utilized to allow the ground user to look at equipment thermal signatures, understand its health, and potentially identify any malfunctions.

In Fig. 7, this snapshot of the Raven mobile application shows the view when UAV almost completes the inspection plan and on the its route back to the start location. The field engineer can view the live video streamed from UAV's onboard camera. The video is shown on the right bottom corner. This video can also be saved for archiving and future reference. The same status bar on top show UAV's status and other indicators as usual. The detected methane concentration can be seen color-coded and overlaid on top of the flight route, along the way-points. On the left of the screen, the engineer can select to go to the reporting view (as shown in Fig. 8) or move on to the next flight plan.

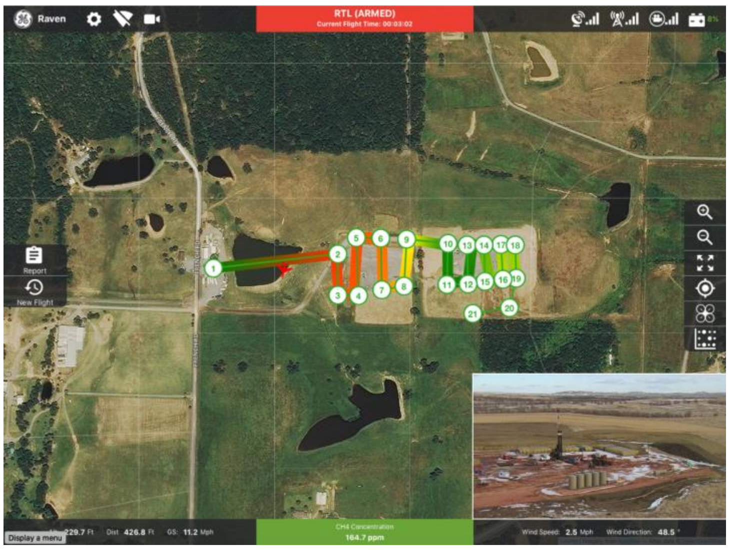
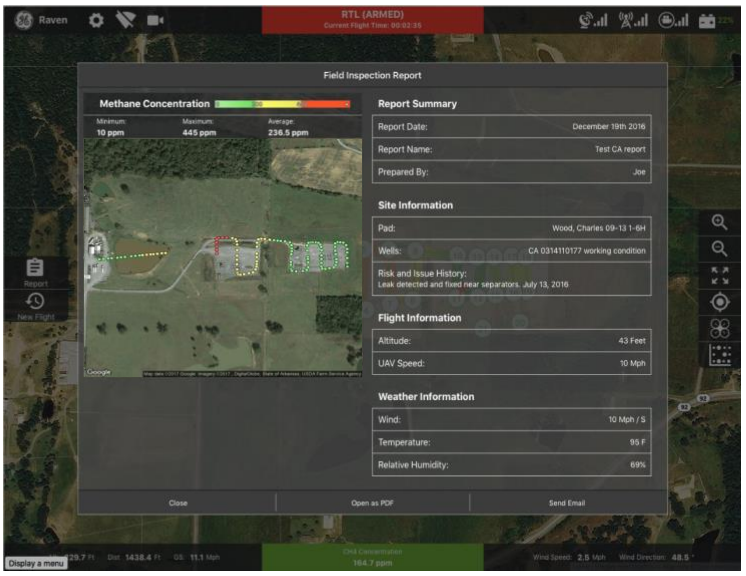


Fig. 7 View the real-time video feed (right bottom corner) from onboard camera.



Field Inspection Report

Methane Concentration

Minimum:	Maximum:	Average:
10 ppm	445 ppm	236.5 ppm

Report Summary

Report Date:	December 19th 2016
Report Name:	Test CA report
Prepared By:	Joe

Site Information

Pad:	Wood, Charles 09-13 1-6H
Wells:	CA 0314110177 working condition
Risk and Issue History: Leak detected and fixed near separators. July 13, 2016	

Flight Information

Altitude:	43 Feet
UAV Speed:	10 Mph

Weather Information

Wind:	10 Mph / S
Temperature:	95 F
Relative Humidity:	69%

Close | Open as PDF | Send Email

Fig. 8 Review report before submitting to Cloud Services.

Reporting: Fig. 8 shows one example of Raven reporting view on the HMI. At this stage, the UAV has completed the inspection plan, with all data collected and analytics model parameters ready to be uploaded to Cloud Services. This reporting view also includes the summary and extra information about the oil facility, UAV's flight status and weather condition during the performed inspection. The engineer can choose to add remarks and review the methane data points rendered on the map. The engineer can also view the PDF version of the same report and, if network connectivity is available, send the report to the manager, site maintenance or other interested parties. This final reporting view completes one inspection plan and the engineer can execute another inspection plan to collect further data points or move on to the next site.

Methane Sensing and Modelling

An important outcome of the inspection process is a geo-located heatmap produced by the app. As the UAV is inflight, the methane concentration values from the laser-based methane sensor are paired with the GPS coordinates. In the HMI, the inspector/operator at the ground-level is able to quickly identify where potential leaks may be. While the HMI relies on Google Earth imagery as the base layer for planning the initial route, the live video feed will point the inspector to the exact location of the UAV, so the concentration hotspots are easily identifiable. After the flight, all of the collected data are processed into a high-resolution heatmap, e.g., see Fig. 9.

Additionally, during post-flight processing, the collected imagery is converted into a new/updated 2D image of the facility. This allows for more accurate leak localization to be conducted. Because of the standardized approach put in-place by regulatory agencies (EPA and others), it is critical for newly developed leak detection technologies to accurately determine both leak location and leak rate.

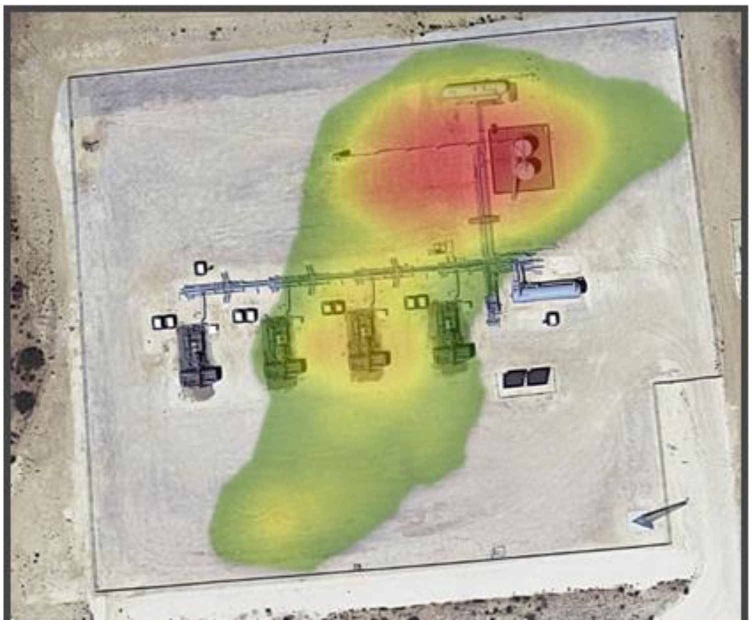


Fig. 9 Post-flight high-resolution methane concentration heat map.

Because the atmospheric methane plumes are significantly altered by wind dynamics, dispersion modelling is necessary in order to trace the detected plumes back to their sources on the ground. For this analysis, weather condition data is necessary. To capture higher granularity of weather conditions during the inspection, an external weather station is placed on-site and captures wind speed, wind direction, and temperature measurements at 4Hz. The geotagged methane data is fused with the weather station data to perform inverse dispersion modelling. Additionally, within a CFD simulation environment, individual leak rates may be estimated.

At the conclusion of the inverse dispersion modelling analysis, a high-resolution concentration heatmap is generated and overlaid onto the facility image. This heatmap indicates the origin of the detected leaks with a higher degree of confidence. While a 2D heatmap is provided in HMI immediately after the flight, it typically takes hours to run the dispersion analytics and output the high precision heatmap due to the intensive computing involved.

Design Trade-offs and Implementation Details

The lack of network connectivity, and limitations in processing power lead us to adopt different design decisions in the implementation of Raven system. The lack of connectivity made us move processing to the drone and to the base station. Connectivity with the cloud, however, is still required for sharing report results, for accessing historical data, and for obtaining information such as weather, and the map itself.

Real-time video was particularly challenging as it requires high bandwidth, which can interfere with drone controls responsiveness. The solution to the problem was to separate video from controls, using independent communication channels. The base station is important as a central point for data gathering and for controlling and coordinating the activity of multiple UAVs. As much as possible, sensor summarization is done in the UAV, having only small data transmitted between UAV and base station during inspection. This optimization is supported by a dedicated UAV on-board computer.

The HMI components and mobile apps are implemented using Cordova/Ionic framework [13] for JavaScript, which allows the porting the application to different mobile platforms including: iOS, Android and Web browser. The base station services are implemented as Docker [16] containers. In particular, Redis [17] Pub/Sub messaging bus is used to handle the communication between components and UAV. The messaging bus exposes WebSockets and REST API interfaces for client-side communication. The Ground Control Station communicate with the UAV using an open source MavLink implementation called MavProxy [18].

The Analytics module uses the 2D Gaussian Process [19] to model the methane distribution. The context module and reporting module are written in Node.js [20], due to the need for efficient non-blocking I/O model. The container-based architecture allows for easy scaling of the base station from a micro-controller in prototyping to a more powerful PC when analytics module was introduced, and eventually to a cluster when UAV swarm is supported.

5 Case Study II: Flare Stack Inspection

Many industrial assets are large in size and many parts of the assets are of difficult access for maintenance crew, e.g., flare stacks, wind turbines, power lines, power towers. Traditional inspection procedures are carried out by climbing on the asset and manually taking close-up pictures and videos of the asset parts. It also involves collecting environmental data such as humidity, temperature, gas concentration, etc. Inspection results are then analysed and summarized to determine the right corrective maintenance procedures for the industrial assets. With proper design, planning and execution, these tasks can be achieved by the UAVs as demonstrated in our first case study.

However, current inspection practices with the UAV typically require two people, a pilot and an inspector. The pilot operates the UAV to fly around the asset, get close to the asset and hovers at certain points of interests. The inspector remotely controls the UAV onboard camera gimbal, shutter and other sensors, collecting data and taking high-resolution photos and measurements of these points of interests. Depending on the types of these points of interests, either RGB or Infrared (IR) cameras are used for different types of leak detection and fatigue diagnosis. The inspector also monitors the live camera view during UAV flight and makes sure the camera gimbal is properly aimed at the target, pictures are correctly taken and exposed, and the sensor data is within acceptable ranges. This is important to assure the quality of photos collected and, as a consequence, the overall quality of the inspection.

After the UAV lands, the inspector manually downloads photos from the UAV, usually by taking out the SD memory card from the camera or by connecting a cable from a computer to the camera. The inspector then goes through the pictures taken during the flight. If some important shots are blurry, improperly exposed or even missing target in the frame, the measurements have to be re-taken, requiring the pilot to fly the UAV again to cover a subset of the points of interests originally planned. At the end of the inspection, all satisfactory photos and sensor data are saved in a computer and brought back to office where a final inspection report is composed, with the detected problems description and the picture as evidence. This report will also be saved as a maintenance record and as a future reference of the asset / site condition.

Current inspection processes are often ad-hoc, with limited planning before visiting a site and flying the UAV. Inspectors use their own judgment and experience to guide the pilot through different points of interest on or around the asset to take photos and measurements. It might take several iterations of UAV flights until all the photos are deemed satisfactory (i.e., clearly show the points of interests and correctly exposed) for the intended inspection purposes. This is usually a time-consuming process, depending on the number of points of interests, size of the asset, weather condition and the manual skills of both pilot and inspector. This inefficiency also incurs more costs in terms of human labor and resources.

In the Flare Stack IaaS (Inspection as a Service) project, we automate both planning and execution stages of the inspection, while still applying human supervision in order to handle exceptions and data collection deep dive activities on the way.

Planning: At the planning stage, the site information and detailed asset 3D model are collected for each site through a detailed site surveillance. Often times, an existing 3D CAD model of the asset can be used in lieu of the actual asset. The engineer then annotates this model with geo-located points of interest. The planning system uses this model to auto-generate the flight path of the UAV and the correct photo sensing tasks to be performed at each point. Each point of interests contains the center of the photo frame, size of the frame, shooting angle of the camera, and the type of the camera (RGB or IR). Inspection usually involves multiple passes over the same points. For example, the capture of RGB and IR images requires switching of cameras on the UAV. This usually results in two UAV flight plans, one for RGB camera photography and another for IR photography to cover corresponding points of interest.

Execution: During the execution and monitoring stage, the field engineer takes the inspection plan, which contains the flight path, points of interest, etc., and uploads it to the UAV's onboard navigation and camera control units. With the help of multiple GPS and differential GPS (DGPS) units, the UAV will fly according to the specified path around the asset. The UAV hovers at the right locations and adjust the camera gimbal so that these points of interests can be photographed properly and from the specified angle. For example, the UAV can first fly the RGB flight plan with the RGB camera installed. Then a second run will be the UAV flight with the IR camera installed.

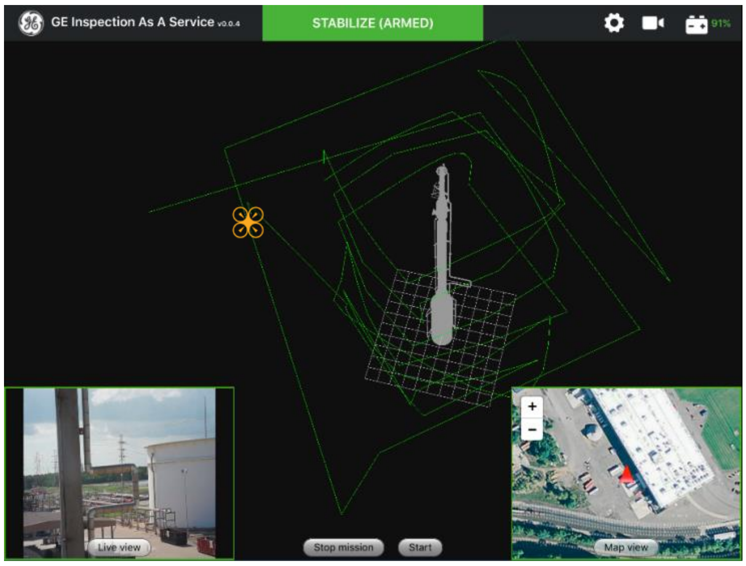


Fig. 10 Review the inspection progress with 3D asset model and UAV flight path in the main screen. Real-time video on the left corner while real-time UAV map coordinates on the right

.

Fig. 10 shows one snapshot of the Flare Stack IaaS mobile application used by the engineer to control the UAV and monitor the inspection progress. With the help of this mobile application, only one engineer is required to perform the inspection, instead of two people (originally, one pilot and another inspector), since the UAV flight and the camera shooting control are both automated based on the inspection plan and real-time controlled by UAV's onboard computing units.

The Flare Stack IaaS app follows similar guidelines as Raven. For example, at the top of the screen is the status bar similar to that in Raven mobile application in Fig. 6, which shows UAV's state, battery level and other buttons for setting and video streaming. The middle portion of the mobile application is the 3D model view and the UAV flight path. In this example, it shows a flare stack with scale grids at the base, colored in grey. The green lines are the flight trajectory of the UAV. The real-time position of the UAV is also rendered on this 3D model view based on UAV's latest GPS coordinate with reference to known flare stack coordinates. The engineer can zoom in/out and rotate the 3D model view and explore from any angle. At the center bottom, there are a few mission control buttons to load mission plan, start, pause, stop, etc., to control the UAV's inspection progress.

At the right bottom corner of Fig. 10, there is a map view of the UAV real-time coordinate and heading on top of the satellite image. This map is to provide a broader view and context of the surroundings and to complement the 3D model view. The engineer can also change the zoom level and explore the map. Every 1 second, if UAV is outside of the scope of the map view due to user's manipulation, the map will be re-drawn with UAV at the center and at the default zoom level.

At the left bottom corner of Fig. 10, the app displays a live video streamed from one camera onboard of the UAV. This camera is not the RGB or IR camera used for inspection photography. This camera view is meant for monitoring and navigation purposes only. This live camera view can be switched to the full screen mode (by clicking on the second button at top right corner) when the engineer needs to see the asset in more details. The photography of points of interest will be done by separate professional cameras to offer better image quality and frame stabilization. Both map view and live camera view can also be hidden (by clicking on the button "Live View" or "Map View") so that the engineer can focus on the main 3D model view and the inspection progress control.

Reporting: Finally, the reporting stage happens after the UAV lands and the data collection is done. Currently, the field engineer imports the photos from the RGB / IR cameras to a computer and subsequently uploads the pictures to an online web reporting tool and fills out the forms to complete the inspection. The Flare Stack app automates this process by automatically downloading the images from the UAV to the mobile application, where the engineer can review the shots, add additional remarks, and finally submit the data to the cloud services where the final report gets generated.

6 Conclusion and Future Work

The ideas of using robotics for industrial inspection have been explored and deployed as early as 1990s, e.g., in nuclear power plants [21], sewage pipes [22], and construction sites [23]. As CPU/GPU computing capabilities, sensing, communication technologies and the precision of robotics control continued to advance over the past decades, there are more and more robotics are automated and used in the very dynamic industrial environments. Some recent examples can be found in [24-32]. Different types of robotics are proposed and deployed for the monitoring or inspection of various pipelines inside industrial assets [24,25], the structure of bridges [26,27] and oil & gas facilities [28]. Specifically, Unmanned Aerial Vehicles (UAVs) are used to inspect industrial facilities, buildings and assets [29-32] due to its lower costs and mobility to navigate inside 3-D space. As one can observe in these recent works, different inspection techniques, sensed data processing, and robotic self-localization are required and specifically developed for the intended inspection purposes. Robotics for industrial inspection remains a hot research area, incorporating real-time control, 3-D motion planning, sensing and communications.

While fully autonomous robotics applications are still constrained to few research settings, supervised semi-autonomic operation of robots in industrial applications are going mainstream. They promote overall cost reduction, efficiency, accuracy and safety of human workers. In this paper, we present this kind of semi-autonomous robotic inspection system as a special case of Edge Computing that requires an orchestration of the robotic autonomy, task planning, and real-time controls involving Cloud, Edge services and proper HMI for human supervision & emergency takeover. We outline the functional requirements for the planning, execution, and reporting stages, as well as non-functional requirements for human-in-the-loop, semi-autonomous industrial robotic inspection systems. We present our design and current implementation, showing how it has been used in the detection of methane leaks in industrial oilfield facilities and the inspection of Flare Stack in oil refineries. This kind of industrial inspection system provides higher resolution and accuracy if compared to the existing manual inspection process. We envision this specific kind of Edge Computing in industrial applications to gain popularity in coming years.

In spite of recent advances, there are still limitations and future work available for such semi-autonomous inspection systems. For example, robots need to act more autonomously and allow fault-tolerance when it is operating outside of video or line-of-sight monitoring of the human operator. Another example is how AR (Augmented Reality), VR (Virtual Reality) or MR (Mixed Reality) can help the human operator gain better situation awareness during inspection and how NUI (Natural User Interface) allows better collaboration among human and robots. The wireless communication range and battery lifetime can also limit the operating range of the robot, especially for large industrial facilities. Last but not the least, AI has been used in real-time sensing, identification, classification and robotics control so that it is possible to learn the environment and correct the pre-conceived model during the inspection and adapt the inspection plan. However, AI requires both data and computation too. Some of these can be solved, remedied or enhanced by faster mobile communication (like 5G), high capacity battery design or more powerful CPU/GPU computation hardware.

Nonetheless, by treating such robotic inspection systems as Edge computing, it still needs more qualitative and qualitative analysis to understand the design trade-offs between Edge computing and Cloud computing. Today most of such Edge-Cloud integrations still require case-by-case designs and are tailored towards the specific applications. There is no general guideline for Edge and Cloud to properly share the computation loads, considering the network bandwidth (e.g., 4G vs 5G) and the amount of data flows required. Towards the full autonomy of such robotic inspection systems, an Edge-driven approach and design perspective are probably required, instead of those of Edge-Cloud integration.

Acknowledgements

We would like to acknowledge the research collaboration and contribution by Oklahoma State University (OSU) team: Dr. Jamey Jacob, Dane Johnson, Nate Lannan, Taylor Mitchell, and Rakshit Allamraju.

References

[1] P. Corcoran and S. K. Datta, “Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network,” *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 73–74, Oct. 2016.

[2] “What is Edge Computing?,” *GE Digital*, 09-Jun-2017. [Online]. Available: https://www.ge.com/digital/blog/what-edge-computing. [Accessed: 04-Jan-2018].

[3] ETSI, “Mobile-Edge Computing – Introductory Technical White Paper,” Sep. 2014.

[4] “Brilliant Manufacturing,” *GE Digital*, 23-May-2016. [Online]. Available: https://www.ge.com/digital/brilliant-manufacturing. [Accessed: 04-Jan-2018].

[5] “Mobile Robotics for Inspection.” [Online]. Available: http://inspection-robotics.com/category/products/mobile-robotics/. [Accessed: 04-Jan-2018].

[6] E. Guizzo, “How Google’s Self-Driving Car Works,” *IEEE Spectrum: Technology, Engineering, and Science News*, 18-Oct-2011. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works. [Accessed: 04-Jan-2018].

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[8] “Avitas Systems, a GE Venture.” [Online]. Available: http://www.avitas-systems.com/. [Accessed: 04-Jan-2018].

[9] “Meet Raven - YouTube.” [Online]. Available: https://www.youtube.com/watch?v=GHDyHALZ3EQ. [Accessed: 04-Jan-2018].

[10] M. Quigley *et al.*, “ROS: an open-source Robot Operating System,” in *ICRA workshop on open source software*, 2009, vol. 3, p. 5.

[11] “MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS.” [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed: 04-Jan-2018].

[12] “Apache Cordova.” [Online]. Available: https://cordova.apache.org/. [Accessed: 04-Jan-2018].

[13] Ionic, “Ionic Framework,” *Ionic Framework*. [Online]. Available: https://ionicframework.com/. [Accessed: 04-Jan-2018].

[14] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” in *Proceedings of the 22Nd International Conference on Software Engineering*, New York, NY, USA, 2000, pp. 407–416.

[15] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context Aware Computing for The Internet of Things: A Survey,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 414–454, First 2014.

[16] “Docker,” *Docker*. [Online]. Available: https://www.docker.com/. [Accessed: 11-Jan-2018].

[17] “Redis.” [Online]. Available: https://redis.io/. [Accessed: 11-Jan-2018].

[18] “MAVProxy — MAVProxy 1.6.2 documentation.” [Online]. Available: http://ardupilot.github.io/MAVProxy/html/index.html. [Accessed: 11-Jan-2018].

[19] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Mass: The MIT Press, 2005.

[20] Node.js Foundation, "Node.js," *Node.js*. [Online]. Available: https://nodejs.org/en/. [Accessed: 04-Jan-2018].

[21] L. Briones, P. Bustamante, M. A. Serna, "Wall-climbing robot for inspection in nuclear power plants," *Proc. IEEE Int. Conf. Robotics and Automation, pp. 1409-1414, 1994.*

[22] H.-B. Kuntze, H. Haffner, "Experiences with the development of a robot for smart multisensoric pipe inspection," *Proc. IEEE Int. Conf. Robotics and Automation, pp.1773-1778, 1998.*

[23] M. Abderrahim, C. Balaguer, A. Gimenez, J. M. Pastor, V. M. Padron, "ROMA: A climbing robot for inspection operations," *Proc. IEEE Int. Conf. Robotics and Automation, May, 1999.*

[24] S. Kim, C. H. Kim, Y.-G. Bae, S. Jung, "Development of spiral driven type mobile robot for NDT inspection in small pipes of thermal power plants," *13th Int. Conf. Control, Automation and System (ICCAS) 2013, pp. 924-928, 2013.*

[25] A. Yamashita, K. Matsui, R. Kawanishi, T. Kaneko, T. Murakami, H. Omori, T. Nakamura, H. Asama, "Self-localization and 3-D model construction of pipe by earthworm robot equipped with omni-directional rangefinder," *Robotics and Biomimetics (ROBIO) 2011, pp. 1017-1023, 2011.*

[26] N. H. Pham, H. M. La, "Design and implementation of an autonomous robot for steel bridge inspection," *Communication Control and Computing (Allerton) 2016, 54th Annual Allerton Conference, pp. 556-562, 2016.*

[27] J. K. Oh, G. Jang, S. Oh, J. H. Lee, B. J. Yi, Y. S. Moon, J. S. Lee, Y. Choi, "Bridge inspection robot system with machine vision," *Automation in Construction, Elsevier, Vol. 18, Issue 7, pp. 929-941, 2009.*

[28] D. A. Anisi, J. Gunnar, T. Lilleghagen, C. Skourup, "Robot automation in oil and gas facilities: indoor and onsite demonstrations," *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS) 2010, pp. 4729-4734, 2010.*

[29] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, R. Siegwart, "A UAV system for inspection of industrial facilities," *IEEE Aerospace Conference 2013, March, 2013.*

[30] S. Omar, P. Gohl, M. Burri, M. Achtelik, R. Siegwart, "Visual industrial inspection using aerial robots," *Proc. Int. Conf. Applied Robotics for Power Industry 2014, Oct., 2014.*

[31] C. Echmann, C.-M. Kuo, C.-H. Kuo, C. Boller, "Unmanned aircraft systems for remote building inspection and monitoring," *6th European Workshop on Structural Health Monitoring, Jan., 2012.*

[32] M. Stokkeland, K. Klausen, T. A. Johansen, "Autonomous visual navigation of unmanned aerial vehicle for wind turbine inspection," *2015 Int. Conf. Unmanned Aircraft Systems (ICUAS), June, 2015.*

Appendix A: Examples of HMI Components

To further illustrate the idea, here we describe the general interfaces and examples of the HMI components (discussed in Section 2) according to the following template:

Name	**The name of the component**
Parameters	Input configuration parameters and events
Commands	Indicate actions that can be performed using the control
Notifications	Events produced by the component in response to changes in its state or data
Example	Sample screenshot showing the appearance of the component

Table 2: Template to describe UI toolkit components API

Situation Awareness Components

Name	**Status Bar**
Parameters	StatusMessage: Object representing the health condition and severity. BatteryStatus: Object representing the available battery percentage. NetworkStatus: Object representing the network connectivity strength. GPSStatus: Object representing the GPS coordinates, signal quality, number of satellites, etc. VideoQualityStatus: Object representing the video quality, e.g., frames rate and image resolution.

Commands	OpenAppConfiguration
	ChangeNetworkConnection {On, Off}
	ChangeGPSPositioning {On, Off}
	ChangeVideoStreaming {On, Off}

Notifications	Allow listeners to be notified of the status when it changes:
	StatusMessage: {Message: String, Severity: Int}
	BatteryStatus: {Voltage: Float, Percentage: Float}
	GPSStatus: {Longiture: Double, Latitude: Double, Active: Boolean, SignalQuality: Float, NumberOfSatellites: Int}
	VideoQualityStatus: {Framerate: Int, Resolution: String}

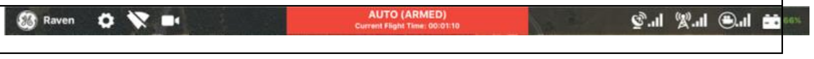
Dependencies	This control listens to UI manager change notifications including: BatteryStatusChange, NetworkStatusChange, GPSStatusChange, VideoQualityStatusChange and others.
Example	 Raven AUTO (ARMED) Current Flight Time: 00:01:10 66%

Table 3: Template for "Status Bar" API

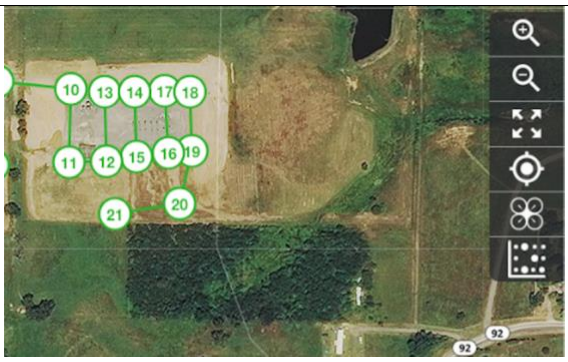
Name	**Map View**
Parameters	MapRegion: A geographic region to be displayed in the map, usually marked by the 4 GPS coordinates as corners of this rectangle. MapZoom: The zoom level of the map. MapLayerList: The list of layers representing paths, objects and other artifacts in a map. MapLayer: Description of individual layer with its elements. MapType: roadmap, satellite image, hybrid, terrain.
Commands	ShowLayer: Render an existing layer on the map HideLayer: Hide an existing layer on the map ZoomLevel: Change the current zoom level of the map ChangeMapType: Change the type of the map RenderPosition: Show the positions of robot and other assets (typically GPS coordinates) on the map AutoCenter: Center the map around the current robot's position with a default zoom level
Notifications	AreaSelection: Coordinates of sub-area in the map AreaZoom: Zoom level modified MapSelection: List of selected by objects in a map
Example	

Table 4: Template for "Map View" API

Name	**Live Video**
Parameters	VideoSource: Source selection of video feed from multiple cameras. VideoQoS: Framerate and resolution specification. VideoFrame: Position and size on the screen.
Commands	StartVideo: Start a video streaming session

	PauseVideo: Pause an existing video streaming session StopVideo: Stop an existing video streaming session ChangeVideoQoS: Change video QoS of an existing video session
Notifications	VideoQoSChange: provides updates on the quality of service including frame rate and resolution
Example	

Table 5: Template for "Live Video" API

Task / Asset Analytics Components

Name	**Inspection Progress Bar**
Parameters	WayPoint: The coordinate of a point on the map (including latitude, longitude and altitude) where robot should stop and perform inspection or sensing. WayPointList: A list of ordered waypoints which represent the route for the robot to traverse on the map. CurrentWayPoint: the waypoint which the robot is currently performing inspection. All waypoints before this point have been covered.
Commands	CreateWayPoint: Add one waypoint into the list of waypoints ReadWayPoint: Go over the list of waypoints and read the waypoint values UpdateWayPoint: Update an existing waypoint in the list DeleteWayPoint: Delete an existing waypoint from the list
Notifications	WayPointChange: {waypointValue: WayPoint, command: String} where command is either CREATE, READ, UPDATE, DELETE WayPointCovered: a waypoint that robot just visits and finishes the inspection at that waypoint DeviationFromWayPoints: any route deviation from the waypoints and notes

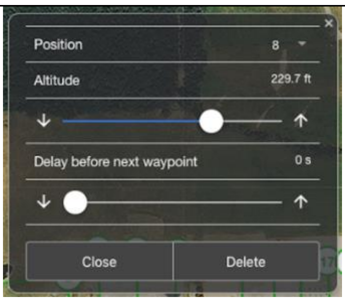
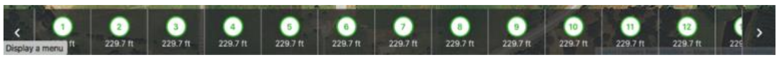
Example	 Position 8 Altitude 229.7 ft Delay before next waypoint 0 s  Display a menu 1 2 229.7 ft 3 229.7 ft 4 229.7 ft 5 229.7 ft 6 229.7 ft 7 229.7 ft 8 229.7 ft 9 229.7 ft 10 229.7 ft 11 229.7 ft 12 229.7 ft

Table 6: Template for “Inspection Progress Bar” API

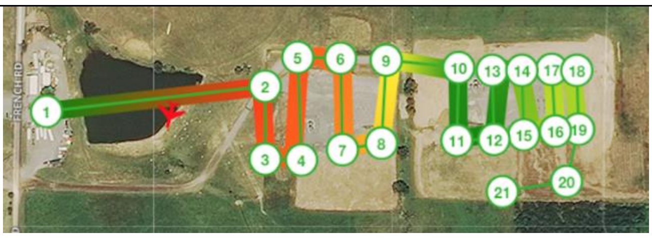
Name	**Heatmap**
Parameters	HeatMapData: Path with inspection status or sensing values for each segment of the map. A segment is a line connecting two waypoints. HeatMapColorRange: Provides the color range and the mapping from inspection status or sensing values into the color representation.
Commands	ShowHeatMap: Render the heatmap layer on top of the map HideHeatMap: Hides the heatmap layer on the map ConfigureMap: defines color range and color mapping from inspection status or sensing data
Notifications	HeatMapDataChange: updated values of the heatmap
Example	 FRENCH RD 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

Table 7: Template for “Heatmap” API

Name	**Analytics Model Visualization**
Parameters	ModelData: A list of model parameters, for the inspection status or sensed data, indexed by map coordinates or other IDs.
Commands	ShowModelDetail: Show details for selected model dataset HideModelDetail: Hide display of model information
Notifications	ModelDataChange: Information about individual changes in the model parameters.

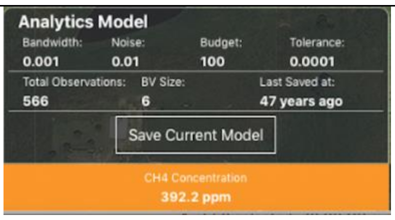
	Alarm/Warning: Pre-defined alarm or warning triggered by the model parameters.
Example	 Analytics Model Bandwidth: 0.001 Noise: 0.01 Budget: 100 Tolerance: 0.0001 Total Observations: 566 BV Size: 6 Last Saved at: 47 years ago Save Current Model CH4 Concentration 392.2 ppm

Table 8: Template for "Analytics Model Visualization" API

History Components

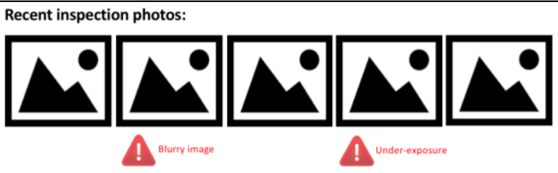
Name	**Timeline**
Parameters	Snapshots: Samples or summary of Inspection status or sensor readings, usually cross-referenced with the waypoints along the route. Additional information can be provided by the analytics model to indicate the data quality of the data or interpret the inspection results.
Commands	ShowShapshotDetail: Show detailed information around this snapshot data. ReferencedWayPoint: Show waypoint information around this snapshot data.
Notifications	Alarm/Warning: About data quality, inspection anomaly, or other conditions
Example	

Table 9: Template for "Timeline" API

Notification Components

Name	**Color Coded Notification Panel**
Parameters	ColorCodedNotification: {message: String, Color: int}
Commands	ShowColorNotificaiton HideColorNotification
Notifications	ColorNotificaitonChange
Example	N/A

Table 10: Template for "Color Coded Notification Panel" API

Name	**Audible Alerts / Voice Notifications**

Parameters	SoundAlert : {waveFile: Stirng}
	SpeechAlert: {text: String}

Commands	Sound, Text-to-Speech
Notifications	N/A
Example	N/A

Table 11: Template for “Audible Alerts / Voice Notifications” API

Name	Vibration
Parameters	VibrationAlert: {duration: long, frequency: float}
Commands	Vibrate
Notifications	VibrationNotification: notifies listeners of a vibration event
Example	N/A

Table 12: Template for “Vibration” API

Interaction Components

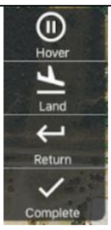
Name	**Quick Action Bar**
Parameters	A list of icons for emergency takeover or mission control commands
Commands	Hoover: stops and glides the drone/robot on the current position Land: land the drone/robot on current location GoPrevioius: go back (return) to the previous waypoint GoForward: go to the next waypoint Finish: Finish operation, landing and saving data.
Notifications	
Example	

Table 13: Template for “Quick Action Bar” API

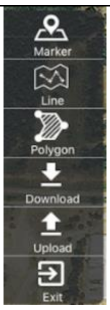
Name	**Flight Path Editor or Configurator**
Parameters	WayPoint: indicates a map position with geo-located coordinates WayPointGraph: a flight path, with points and transitions (lines) Line: a connection between two waypoints, which is part of a graph
Commands	AddMarker RemoveMarker AddLine RemoveLine AddPolygon UploadPath DownloadPath
Notifications	PathChangeNotification: information about what changed in the path model
Example	

Table 14: Template for “Flight Path Editor or Configurator” API