Semi-Autonomous Industrial Robotic Inspection: Remote Methane Detection in Oilfield

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. They usually operate along with other robots and human supervisors in complex tasks such as industrial assets inspection, monitoring and maintenance. Even though fully autonomous robotics applications are still work-in-progress, supervised semi-autonomic operation of robots in industrial applications are going mainstream. They promote overall cost reduction, efficiency, accuracy and safety of human workers. 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 is a first in series where we describe a robotic platform developed within BHGE and GE-GRC, discussing its use in one example of industrial inspection case study for remote methane inspection in oilfield. 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).

I. INTRODUCTION

Robotics have been increasingly used to automate dirty, dull and dangerous jobs in the industrial domain. 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 inside parts of heavy equipment. The mobile computational nature of robotic inspection systems, makes it a good example of the Edge Computing [1]–[3] in the industrial setting [2].

While high levels of robotic automation and autonomy have been achieved in settings such as industry production lines [4], more complex industrial tasks still require human supervision. A typical example is the inspection of industrial assets [5]. During these inspections, robots work with humans in the detection of structural and functional problems involving oil pipelines, power generation turbines, containers, reactors and others. Robots interact with assets through nondestructive 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.

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 why constant human supervision is still required. Finally, as any mechanical asset, robots still need periodic maintenance including cleaning, battery replacement, part exchange, and calibration. Hence, while robotic autonomy has been increasing over the years, especially in consumer space, industrial operators must still supervise activities of robots, ready to assist when in-situ exceptions occur.

Another 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 network connectivity. The ability to cope with intermittent or poor network connectivity is therefore essential during robotics inspection.

However, constraints in footprint and power limit the amount of computing robotic systems may have. For example, UAVs (Unmanned Aerial Vehicles) must be lightweight and small enough to fit restricted industrial areas, e.g., within turbine casings and pipes; while the onboard battery power must be shared between the UAV computing and propulsion systems. As a consequence, while certain amount of autonomy can be embedded in the robot itself, e.g. through GPS sensing, object and movement detectors, computer vision, and other techniques, large part of computation necessary for its operation needs to be performed elsewhere.

For example, autonomous cars rely on high-fidelity models of the road environment computed based on large amounts of data in the cloud [6]; while industrial model-driven approaches, as those described in this paper, require the development of accurate asset models, based on extensive site survey and

operations planning, followed by the optimization step where a 3D model of the asset and site are used to plan and guide the robots through the most efficient route.

Hence, robotics inspection requires a careful balance between autonomy, task planning, and real-time controls involving cloud, edge computing and proper interaction with humans. In this paper, we present an Edge Computing architecture [7] of a model-driven robotic UAV system used for industrial inspections.

We first present the requirements of the system in the context of industrial inspection scenarios; we then show how to address these requirements by a human-in-the-loop, edge computing system combining UAVs, base satiation, cloud services and human operators. We describe the basic interfaces of the system, discussing the design trade-offs and implementation details we adopted. We then validate the approach showing how it has been used in an industrial application: the detection of methane leaks in industrial oil & gas facilities.

II. REQUIREMENTS

Asset inspections are critical to the proper operation of industrial systems. Inspections enable early detection of operational anomalies and asset decay that may lead to future breakdowns; they inform maintenance crews on the parts of the system that must be serviced or replaced, thus preventing failure of the asset; Inspections are also required after major overhauls, before bringing the asset back into production. Robotics-assisted inspections of industrial assets typically consist of three stages: Planning, Execution and Reporting. In this section, the authors specify functional requirements for these three stages and the non-functional requirements (NFRs) of our design.

A. Inspection Planning

Many industrial assets must be shut down or overhauled before an inspection can occur. E.g. flare stacks and power plant turbines must be shut down, 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.

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 in the inspection (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). Depending on the applications, different models are produced as part of the plan:

- **Environment model**: Similar to a road map in a 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**: These are high-fidelity digital representation of assets and reside within the overall environment model representations. They define the exact location, within the environment, 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, tasks and maintenance events to be followed by each robot around the assets, within the overall environment. In particular, the plan contains: 1) **Inspection workflow model**, which describes the tasks to be performed at each step of the inspection, identifying tools and activities required in each step. 2) **Data capturing configuration**, which identifies the sensors to be used, the type and quality attributes for the data to be captured in each step of the workflow model. 3) as well as **Robot and tools schedule**, as it is usually the case that more than one robot can be used during an inspection. Note that multiple robots allow for divide-and-conquer use of sensors and tools to reduce inspection time.

Once the execution plan details are identified, further optimization is typically performed, e.g., to minimize the overall inspection time, energy or the materials required. Hence, computational intensive simulations based on the environment and asset models are typically used to adjust the execution plan to the environment, assets and the characteristics of each robot. Typically, this plan is reviewed and approved by the human supervisor before its execution by robots.

B. Inspection Execution

During the inspection execution, robots work along with human supervisors and auxiliary systems in the enactment of the inspection plan. As in any model-driven approach, any unforeseen conditions and exceptions must be detected and gracefully handled by the human workers. In particular the system Human-Machine Interface (or HMI) plays a key role in this stage to support the plan execution supervision and exception handling.

- **Supervised plan execution**: During this stage, the 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: 1) **Robot's location**, with the precise location of the robots with respect to the environment and the assets being inspected. 2) **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; and 3) **Data quality**, which provides a way for the operator to assess the quality of the data collected, e.g. by visualization of sensor data, images, with the timestamp and location on the assets.
- **Exception handling**: More importantly, human workers must be able to intervene on the operation of the robot during its execution in order to handle exceptional situations and perform corrective actions. The system HMI must support: 1) **Exception notification** by means of visual, audio, vibration or tactile alerts. 2) **Manual override or emergency takeover** for the human

supervisor allowing full operator control of the robots; and 3) **Macro functions** allowing for quick execution of common corrective tasks, e.g., autonomously navigate the robot back home or default starting location; autonomously keep robot's current position; gracefully shutdown; pause, roll-back, resume last task; among others.

C. Inspection Reporting

The inspection report summarizes the findings with collected data during the process, and indicates potential corrective actions to the assets. Reporting often requires further data refinement, processing and exploration around the findings. For example, during an inspection, defective asset parts may be detected which may require in-depth analysis though the capture of close-up photos, sensors and measurements.

In our architecture, we optimize the process of reporting by supporting different activities during inspection execution. These activities are performed adaptively based on the current inspection findings. In particular, the system must support:

- **Incorporation of field notes**: Operator-driven field notes, during or after inspection, with the attachment of extra data points (e.g. photos, sensor data).
- **Ad-hoc data collection tasks**: Optional inspection tasks for assets (outside of original plan) as a way to collect further data that may be crucial in determining the asset condition.
- **Interpretation of the collected data**: Based on the data collection, basic interpretation of the data and asset status along with recommended actions / next steps. This is typically done in the edge or, if network connectivity is available, can also be performed in the cloud.
- **On-site report generation**: The faster one can generate reports, the cheaper it is for customers and inspectors. Whenever possible, the final report should be generated on premisse, with minimum use to cloud services by using cached and pre-loaded templates.

D. Non-Functional Requirements

Besides satisfying the above functional requirements, the inspection system must support the following.

Multi-robot management: As the complexity of inspection tasks increases, multiple robots may be used in sequential or parallel schedules, performing similar or complementary tasks. 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 also 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, that require 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. The system must provide a common 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: Finally, 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) [11].

Mobility: 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.

III. DESIGN

In this section, we present our robotic inspection platform architecture 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.

A. Logical System Architecture

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

- **Mobile devices (orange boxes):** Apps are developed based on cross-platform development for iOS, Android, and web browser in environment like 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.
- **Base Station (green boxes):** 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 Base Station can support multimedia like video, 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 (blue box):** Many cloud services are used. They can archive uploaded data in database or file repositories, provide image recognition, data analytics, and other computational-intensive services. Cloud services are used through REST APIs [14].

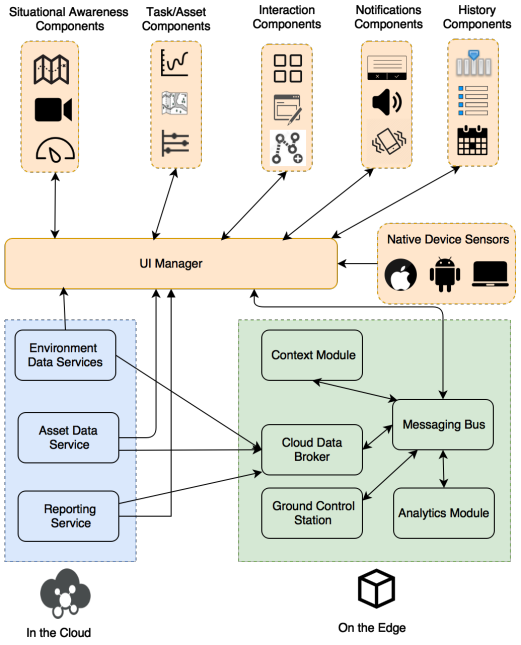

Fig. 1 General logical system architecture.

B. Physical System Architecture

Fig. 2 shows one realization of the 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) where the user can use application 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.

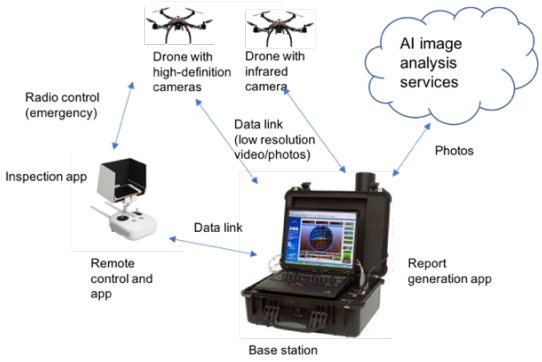

Fig. 2 One realization of the architecture and entities in Fig 1.

The mobile characteristic of the system requires onsite area wireless networking and control among robots, Base Station and mobile devices. Robotics protocols can run on wireless networking layers like Wi-Fi. Meanwhile, intermittent connectivity, 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 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.

C. 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, used during inspection, planning and reporting. In particular, we classify the HMI components as the follow five types, as depicted at the top in Fig. 1.

Situation Awareness Components provide human supervisor with the awareness of the inspection. For example, 1) **Status bar** for showing the status of robots, flight metrics, battery levels, etc. 2) **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. 3) **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 are those that provide information related to the inspection tasks or the work flow for the robot to perform. The human worker utilizes these components to make sure that the robot performs according to the plan. For example, 1) **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. 2) **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. 3) **Analytics model visualization** showing the statistics and parameters behind analytics models and data visualization.

History components are those provide the history data overview during the inspection. For example, **Timeline** bar shows the recently captured sensor data, photos or video, so the user can nevigate 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 are multiple-modal alarms and notifications to the human supervisor. For example, 1) **Color-coded notification panel** to visually indicate important events with different levels of severity and health warning of the robots and assets. 2) **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. 3) **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, 1) **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. 2) **Robot path editor** to support onsite ad-hoc movement planning for robots like UAV flight route update.

D. Base station components

The base station sitting on the edge plays the central role within our robotic inspection platform. It consists of a variety of 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 Module** 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/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.

E. Cloud services

As discussed earlier, assets inspection utilizes a set of cloud services to provide information that can be consumed by the user interfaces as well as the base station components. In general, we summarize them into three categories: 1) **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. 2) **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. 3) **Reporting Data Service** is used to store the inspection results to the cloud and also allows the mobile apps to retrieve historical reports.

IV. CASE STUDY: METHANE DETECTION IN OIL FIELD

In this section, we show how our robotic architecture has been used in support of 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 focusing on UI/UX and system architecture.

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 currently performed by human inspectors that need to manually inspect facilities and pipelines with using optical gas imaging (OGI) devices. This method is time-intensive and the data provided by the measurement device is qualitative in nature. i.e. it only provides a binary (yes/no) indication of leakage with no measurement of its intensity. Additionally, the OGI devices are

expensive and difficult for operators to deploy at-scale. Raven replaces this procedure with the process depicted in in Fig. 3.

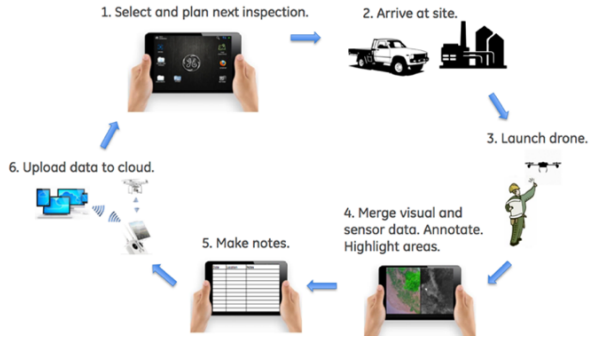

Fig. 3 Workflow for Raven remote methane inspection in oilfield.

In Step 1 of Fig. 3, the field engineer browses existing oil facilities, reviewing past inspection reports and decide which one 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. 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, 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.

A. Methane inspection scenario

This section shows in detail, the use of Raven methane inspection app, hightlighting the main features of the system during planning, execution and reporting stages.

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. 4, 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 hoving 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 the way-points on the map, which later will be converted into GPS coordinates. The way-points can also be generated automatically by specifying the area to cover.

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 system 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. Another component in creating the flight plan is assessing the appropriate speed of the UAV during the survey.

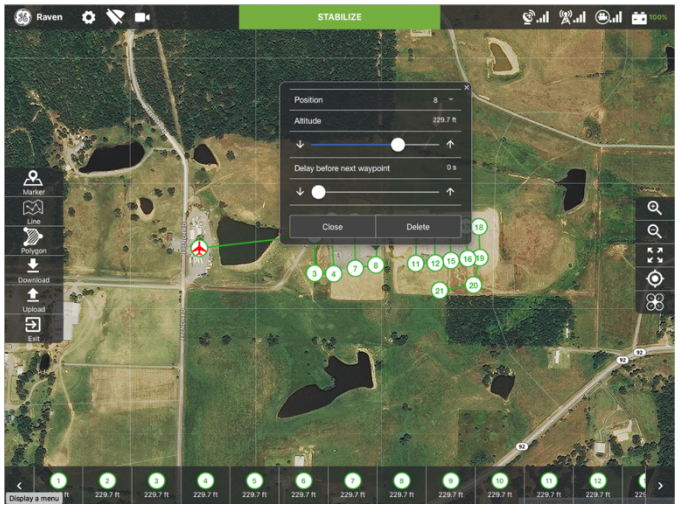

Fig. 4 Plan the UAV's flight route as a sequence of way-points.

The appropriate speed 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 system features 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.

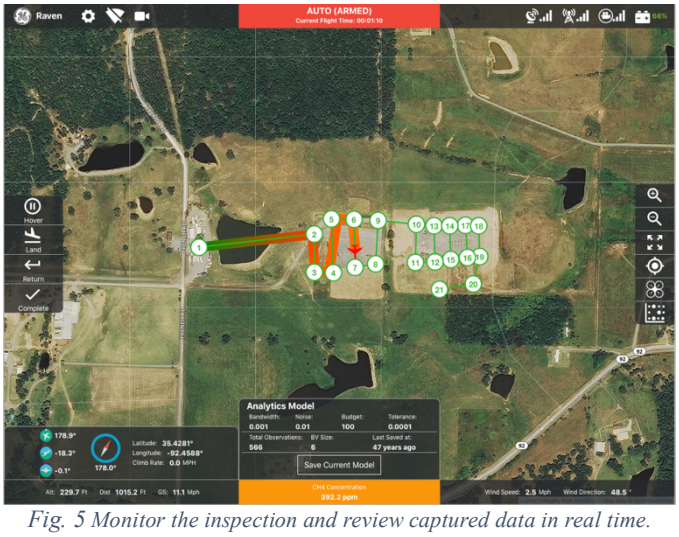

Fig. 5 Monitor the inspection and review captured data in real time.

Execution: During inspection execution, as seen in Fig. 5, in the same HMI allows the field engineer to monitor the inspection process. The top of the screen shows the situation awareness components and the status bar. The status bar indicates UAV's state in the middle banner, along with satellite signal, communication signal and battery level. On the top left, there are a few buttons for App setting, connecting to the UAV, and video streaming from UAV. The heatmap and flight route are overlaid on top of the satellite map. The heatmap shows color-coded methane concentration based on sensor data. The left side of the screen shows some Macro buttons for emergency UAV takeover functions.

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 status of the methane analytics model. Latest methane sensor value is displayed here with colors matched with those on the heatmap. The outputs of the analytics model, as the parameterization of the probability distribution, are also recorded and then 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.

Fig. 6 Review report before submitting to Cloud Services.

Reporting: Fig. 6 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.

B. Methane sensing and modeling

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. 7.

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.

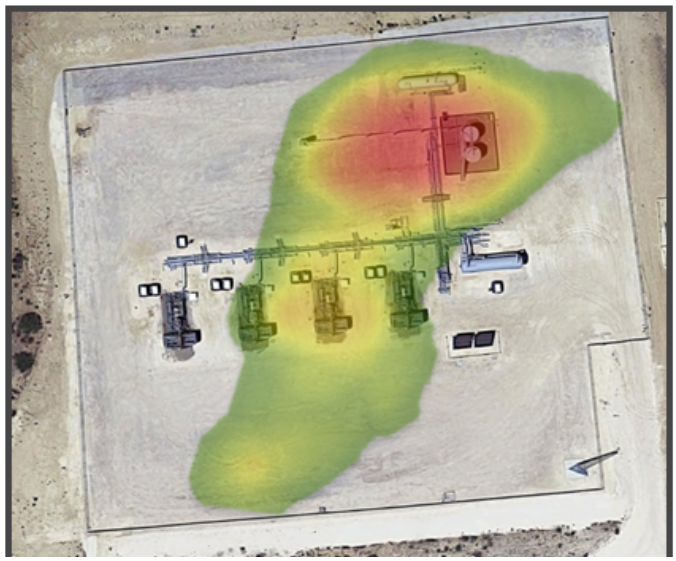
Fig. 7 Post-flight high-resolution methane concentration heat map.

Because the atmospheric methane plumes are significantly altered by wind dynamics, dispersion modeling 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 modeling. Additionally, within a CFD simulation environment, individual leak rates may be estimated.

At the conclusion of the inverse dispersion modeling 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.

C. 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 in using Cordova/Ionic framework [13] based on 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.

V. Conclusions

Semi-autonomous robotic inspection system is 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 and emergency takeover. In this paper, we outline the functional requirements for the planning, execution, and reporting stages, as well as non-functional requirements for human-in-the-loop industrial robotic inspection systems. We present our design and current implementation, showing how it has been used in the inspection and detection of methane leaks in industrial oilfield facilities. This UAV inspection system provides higher resolution and accuracy if compared to the existing manual inspection process based on hand-held optical gas imaging devices. It produces more precise heatmaps and automates the reporting and data sharing processes. We envision this specific kind of Edge Computing in industrial applications to gain popularity in coming years.

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].