SigSent: An Intelligent, Multi-Terrain, Hexapod Sentinel

Joshua Lee Franco, John Millner,
Jeff Strange Jr., Richard Wales

Dept. of Electrical and Computer Engineering,
University of Central Florida,
Orlando, Florida, 32816-2450

Security guards routinely work in situations best described by the "three Ds of robotics"—dull, dirty, and dangerous. Sentinel robots can multiply the presence of security guards, and by doing so, reduce the costs and risks associated with a conventional security strategy. With the ability to either walk as a hexapod or skid-steer drive with four wheels, SigSent can traverse across varied terrain (e.g., sand, grass, etc.). SigSent's walking gait can be refined through a genetic algorithm. SigSent features multiple human-robotic interaction capabilities including automated pedestrian detection, voice command, and teleoperation with a joystick controller. SigSent can also operate autonomously by following predefined surveillance routines with GPS waypoint navigation and local path planning. Additionally, the appropriate mode of locomotion is suggested to operators through terrain classification using the NeuroEvolution of Augmenting Topologies framework.

***Index Terms*—Intelligent robots, Artificial neural networks, Genetic algorithms, Human-robot interaction, Security management.**

Fig. 1: SigSent in Walking Mode

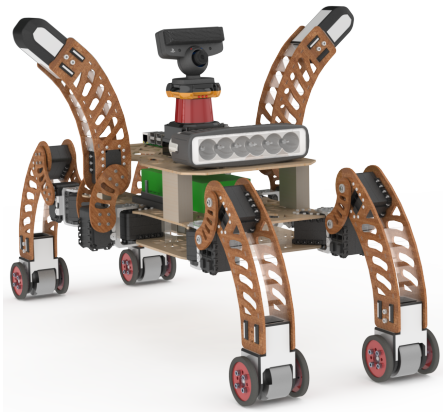

Fig. 2: SigSent in Driving Mode

I. Introduction

The goal of SigSent is to enable security professionals to patrol outdoor areas remotely, reducing the risk of harm to human sentries. SigSent should also encourage proactive security policies by freeing security professionals from repetitive tasks.

By learning to work in a mixed terrain environment, the robot can effectively perform its job as a sentinel regardless of its surrounding landscape.

Operators can either directly control the robot's actions through teleoperation or indirectly manage the robot using an autonomous sentry mode. The SigSent bot streams a video feed of its first-person perspective, enabling remote surveillance. With multiple SigSent units, a single operator alone could surveil a much larger area. While patrolling, SigSent also alerts the operator upon detecting human activity. This can reduce the cognitive demand on security professionals arising from constant simultaneous supervision of multiple locations.

SigSent is designed to interact with people it encounters. The robot is designed such that it can pursue an intruder if deemed necessary by its operator. Through Sigsent's onboard speaker, an operator can relay vocal instructions to an encountered pedestrian. By strobing its lightbar, SigSent can disorient trespassers. SigSent's camera can be used to record video of events for later action by law enforcement.

SigSent is able to fulfill many of the time-intensive duties of security guards, enabling guards to perform higher-level tasks with less occupational hazard.

II. Mechanical Design

SigSent's design is an extension of the conventional hexapod robot. It possesses 6 legs, with 3 degrees of freedom (DOF) controlled by Dynamixel AX-12A servomotors, which sets it apart from other known implementations that may feature only 1 degree of freedom for each leg [1]. In addition, SigSent features four motorized wheels attached to its front and rear legs. Each wheel is driven by an AX-4114C brushless DC outrunner motor and controlled by a Favourite LittleBee electronic speed controller. The ESCs were flashed with BLHeli_S firmware to enable bidirectional operation. This allows SigSent to either walk on six legs across rough terrain as a hexapod or drive

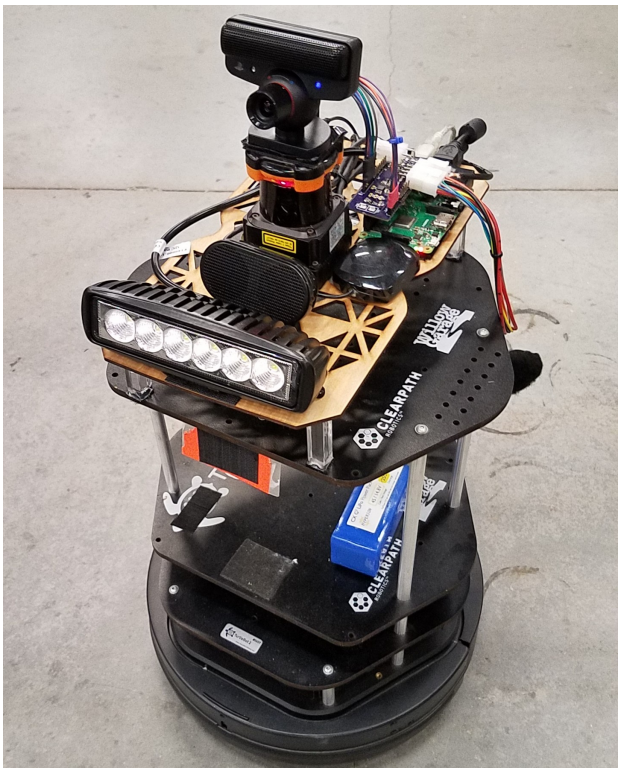

Fig. 3: Testing Platform of a Turtlebot 2 with the SigSent Sensor Array

over smoother surfaces efficiently in a four-wheeled vehicle configuration, as illustrated by Figures 1 and 2 respectively.

SigSent is designed to support a tripod gait with its weight only supported by three of its legs on the ground. It is also designed to drive at speed with its middle legs lifted off the ground.

SigSent's chassis was designed with SolidWorks and manufactured by a combination of laser cutting and 3D printing in the Texas Instruments Innovation Lab.

In order to better parallelize our team's tasks, a test platform was used to prototype the high level goals of the vehicle while the SigSent platform was designed and constructed. This platform, a ClearPath Robotics TurtleBot 2, on loan from the Robotics Club at UCF was modified to include SigSent's modular sensor array to allow for easy switching between the two platforms as seen in Figure 3.

III. Electrical Design

A. Printed Circuit Boards (PCBs)

SigSent incorporates 5 custom-designed PCBs which host its motion-controlling microcontroller, fuel gauge, inertial measurement unit, Raspberry Pi microcomputer integration, and servomotor voltage regulation. All the PCBs feature double-sided construction. The boards' components

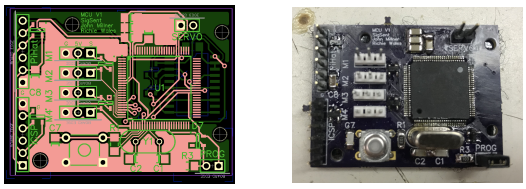

Fig. 4: MCU board layout in Diptrace and fully assembled

were primarily surface mounted with the use of reflow soldering. PCBs were designed with the use of Diptrace and were manufactured by either OSH park or JLCPCB depending on the board. The PCB design in Diptrace, and the fully assembled board for SigSent's motion-controlling microcontroller board is shown in Figure 4

B. Power System

SigSent is powered by an attached Turnigy Multistar 4 cell, 10Ah LiPo battery. Overcurrent protection is provided by a 60A fuse off the battery. Current, charge, and voltage is monitored by an LTC2943 fuel gauge and 300 $\mu\Omega$ sense resistor to be output to the robot's microcomputer. High current or out-of-bounds voltages are highlighted to the robot's operator through the basestation's GUI. Power is distributed through a set of busbars. Each of SigSent's legs have an associated custom-designed board which provides 12V regulated power to the 3 respective servos. The servo regulator boards were designed with the use of Texas Instruments's WEBENCH Power Architect tool [2]. A Mini-Box M3-ATX picoPSU powers SigSent's sensors and on-board computers.

C. Sensors

SigSent is able to observe its surrounding environment through the use of a Sony PlayStation Eye. The Eye incorporates both a 480p RGB camera sensor with a 60 hz frame rate and a quadrophonic microphone array [3]. The camera sensor is used to provide the robot's perspective for teleoperation. The microphone is used to relay the sounds of the robot's environment to the operator through headphones attached to the basestation. The Eye interfaces with the Raspberry Pi through USB 2.0.

In order to determine the distance to objects in the surrounding environment and map its surroundings, SigSent utilizes a Hokuyo UTM-30LX 2D LIDAR (from the Robotics Club at UCF). This LIDAR features long range detection up to 30m with a 270°field of view. The UTM-30LX utilizes a 905nm wavelength semiconductor laser diode, which enables outdoor operation. The LIDAR scans at 40 hz and possesses a resolution of 5 cm at 30m [4]. It also interfaces with the Pi through USB 2.0.

SigSent localizes itself globally through the use of an integrated GPS module. The module, a SparkFun board incorporating a Venus638FLPx IC, is attached to an active omnidirectional GPS antenna through an SMA connector. The module can update at up to 20 hz, possesses a 2.5m accuracy, supports WAAS, and maintains a -168dBm sensitivity. [5] The module interfaces with the Pi over its I^2C bus.

Additional motion feedback and detection is provided by the inertial measurement unit (IMU) mounted to SigSent's abdomen. The IMU is a custom-designed board incorporating an InvenSense MPU-9250 which communicates with the Pi over its I^2C bus. The MPU-9250 comprises a 16-bit resolution 3-axis gyroscope, a 16-bit resolution 3-axis accelerometer, and a 14-bit resolution 3-axis magnetometer. Additionally, the MPU-9250 possesses a proprietary motion processing engine which fuses sensor data to produce a pose quaternion [6].

IV. Software Design

SigSent's microcomputer, a Raspbery Pi 3 Model B+, utilizes the Ubuntu MATE 16.04 LTS Xenial Xerus operating system . A majority of its code base is implemented through the Robot Operating System (ROS), and its intelligence training system runs independently in Python scripts as needed.

A. Robot Operating System

The Robot Operating System (ROS), originally developed by Willow Garage and currently managed by the Open Source Robotics Foundation, was used to accelerate the development of SigSent. ROS is a system that manages programs and their interactions with each other. With ROS, transferring data from one program to another is standardized into pre-agreed messages through a TCP/IP network that allows for modular and maintainable code, different sensors and actuators, and distributed computing. These standardized messages and the open source community behind ROS have allowed for many algorithms and procedures to be genericized, which enabled the SigSent team to quickly develop a working robot without redundant development of well-defined concepts. For this project, ROS used the version Kinetic Kame, the up-to-date LTS version of ROS over the more recent Lunar Loggerhead, or the new ROS2.0 release since documentation and stability was prioritized over bleeding edge functionality.

1) Odometry

SigSent is able to know its pose (its position and orientation) in the world by implementing an Unscented Kalman Filter(UKF) that fuses GPS, IMU, and estimated position of the legs (while in walk mode) or the estimated RPM of the wheels (while in drive mode). A UKF was implemented over the more traditional Extended Kalman Filter(EKF) to ensure a more stable output after comparing the two. The UKF provided by the ROS package `robot_localization` allowed for significant customization. Through experimentation, parameters were tuned to achieve more stable poses for path planning.

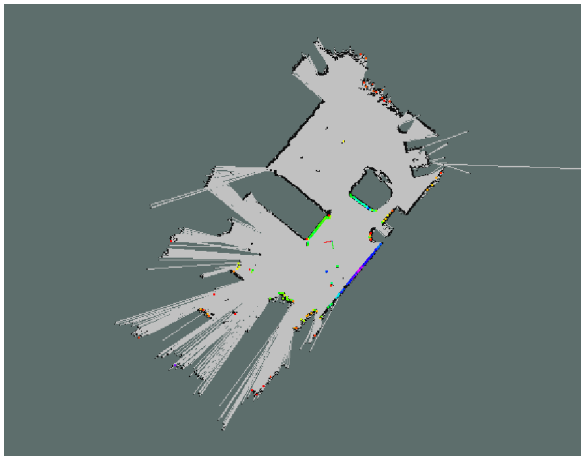

Fig. 5: An occupancy grid created with SLAM [7]

2) Localization and Mapping

SLAM or Simultaneous Localization And Mapping receives radial distance data gathered by a laser scan of the Hokuyo UTM-30LX LIDAR sensor and creates an interpreted map in the format of a 2D occupancy grid. This map is used to inform almost every part of the path planning algorithm. An example is shown in Figure 5. SigSent implements SLAM through the ROS package `gmapping`, an implementation of a Rao-Blackwellized particle filter developed by the OpenSLAM community.

Being a 2D LIDAR, the UTM-30LX can only scan radial distances at a single inclination. Due to this, objects such as chairs and tables often go undetected. Modern 3D LIDARs provide a much more informative perspective of the world and enable more detailed maps, but are prohibitively expensive.

3) Path Planning

Path Planning is achieved through the implementation of local and global path planning routines and utilization of the ROS package `move_base` first developed by Eitan Marder-Eppstein and David V. Lu in 2009. `move_base` manages the local and global path planners as well as the costmaps that the path planners use.

SigSent uses `NavFN`, an implementation of Dijkstra's algorithm, as its global path planner. Other algorithms and heuristics such as A* and D* lite were considered but were difficult to implement and tune compared to `NavFN`. The global planner is primarily in control of the "global path", which is the main path followed from the robot's

initial position to its end goal. The global path describes macroscopic poses for the robot to reach its goal.

The local planner determines and describes how the robot will achieve the paths set by the global planner. SigSent utilizes the `DWA_local_planner` ROS package, an implementation of the Dynamic Window Approach. `DWA_local_planner` accounts for the robot's footprints, trajectories, and motion by simulating potential poses in the future and determining the optimal outcome. `DWA_local_planner` is highly configurable and significant time was spent tuning its various parameters to ensure that SigSent was able to safely and efficiently navigate in both open spaces and tight, highly dynamic ones (e.g., a classroom filled with students).

Costmaps denote the "cost" for a robot to enter areas of the map, where a lower cost denotes the robot being safer or more efficient to move there over another area of the map. Costmaps for SigSent are generated through a layer plug-in approach used by the ROS package `costmap_2d` which uses 4 layers of plug-ins to determine the varying costs of the local and global costmaps. The obstacle layer detects potential obstacles (including walls) and other things that the robot could not enter without impacting. The inflation layer places an exponentially decaying cost around obstacles in order to dissuade the robot from getting near obstacles unless it must to avoid other, closer obstacles. The third and fourth layers in combination provide human detection and movement prediction. Developed by David V. Lu, the `social_proxemic_layer` identifies people by detecting the legs of a walking or stationary person. The layer places a Gaussian distribution around identified people depending on their predicted vector. The second layer, `social_passing_layer`, creates a much higher cost in the predicted path of a human in order to keep the robot out of their way. The robot can move around detected humans without stopping and recalculating its overall path.

4) GPS Waypoint Manager

The GPS Waypoint Manager is a custom node created for ROS which receives commands from the GUI or a script and directs the robot to patrol a sequence of GPS waypoints continuously until stopped. The program first sets up the `GPS_goal` node created by Daniel Snider which translates "global" GPS messages to coordinates of the local map created by `gmapping`. From there, the GPS Waypoint Manager loops through a given ROS message of GPS waypoints and sends the waypoints to the `gps_goal` node. After a goal has been sent, the node then monitors `move_base`'s status until it has confirmed that SigSent has reached its immediate goal. Once confirmed, `gps_goal` sends the next goal. If SigSent fails to reach its goals after some number of attempts designated by the waypoint manager, or if a diagnostic warning is received, `gps_goal` raises a flag, prompting human intervention.

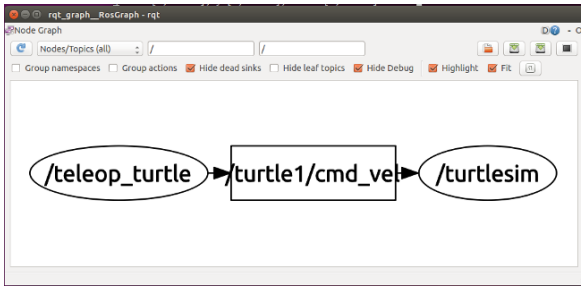

Fig. 6: An example ROS topic graph [8]

5) System Monitoring

Several systems were implemented to improve monitoring of SigSent and to diagnose any issues encountered. Monitoring and diagnosing problems is made significantly easier by ROS's built-in info logging features. Different levels of importance such as "info", "warning", "error", and "fatal" help filter errors and more clearly indicate their relative importance. ROS provides easy viewing of these through `rqt_console`.

`rqt_graph` and `rqt_tf_tree` are used to visualize the system's structure and its transforms. These tools help users better understand how nodes and messages are interacting throughout the system. Analyzing these visuals helps ensure proper system connectivity and communication. An example `rqt_graph` visualization is shown in Figure [8].

In order to properly visualize sensor data, path planning, and costmaps, RVIZ (ROS Visualization) was integrated with a custom configuration file illustrating SigSent's LIDAR data, global and local path planning, global and local costmaps, camera feeds, pose, and odometry. The CLI command `rostopic echo /[topic]` is used for quick and insightful diagnosis of problems that occur during development or in the field.

Battery voltage and current are monitored to ensure that SigSent's battery is never drained excessively deep or dangerously overdrawn in the field.

B. Genetic Algorithms

1) NEAT

SigSent features basic terrain classification through a trained artificial neural network (ANN) output from the NeuroEvolution of Augmenting Topologies (NEAT) framework [9]. IMU data can be gathered while driving and walking over various terrains. The data is then labeled with the respective mode of locomotion and whether the terrain was rough or smooth. NEAT then uses this training set to evolve a neural network that would correctly classify what kind of terrain the robot is traveling over.

NEAT uses a genetic algorithm (GA) to perform the neuroevolution. An initial set of ANNs, called the population, is created with randomly generated individuals. These neural nets have 31 inputs: linear acceleration and angular velocity in three dimensions from the five most recent IMU readings, and a binary input specifying what movement mode the robot is currently in. The singular output on the net is a boolean detailing whether the terrain is smooth or not. These individuals are evaluated and given a fitness value that measures their performance. For our purposes, the fitness would be increased whenever the ANN correctly classifies an IMU data point given to it.

The population's ANNs are modified to search for the best-fit individual. The genetic operators that perform this action are aptly named mutation and crossover. Mutations occur with a relatively small probability and directly change something about an individual. In the case of the neural network, it could add or remove nodes and connections between them. Crossover serves as the main operator to explore the search space. The genomes of two parent networks are combined to form a child network that now contains all disjoint and excess nodes between them.

Each iteration of this process is known as a generation. After training, the ANN is exported to a file to be loaded by its respective ROS node at runtime. The classifier then subscribes to the IMU data topic and calculates what terrain type it believes to be traveling over.

2) Gait Optimization

A unique GA was written in Python to optimize SigSent's walking gait. The same GA process described above was used with specialized functions to perform the genetic operations on the population. The algorithms for the operators followed the procedures outlined in work by Manglik et al [10].

Mutation and crossover are restricted to only produce individuals whose fitness evaluation scores higher than their parents. Also, mutation and crossover get repeated on the individual if the generated offspring is not stable. Manglik's paper describes stability as having the robot's center of gravity lie within a defined polygon encompassing its abdominal structure [10]. To mimic this process, we would check if any of the servos have an angle too extreme to keep stable.

Mutation has a probability of occurring on a specific individual and also has a probability of modifying a servo's angle state. If while iterating through the individual's servos, none are selected, a servo at random is chosen to ensure at least one gets modified. This yields an expected number of mutated servos, E_m, where $E_m = P \times p(m) \times S \times p(s)$ where P, $p(m)$, S, and $p(s)$ is the population, probability of mutation, number of servo states per gait, and probability of servo selection respectively.

In crossover, selected parents have a probability to swap state values of randomly chosen legs at steps in the gait. Individuals are chosen for crossover using tournament selection, a greedy method where k number of random individuals are chosen from the population and the best individual in the random pool is returned. The Cartesian product of the two gaits' sets of legs is iterated over so that all pairwise combinations of the legs are candidates for swapping state values. Crossover's expected value is similar to mutation's, given as $E_c = P \times p(c) \times L \times p(l)$ where $p(c)$ is the probability of crossover, L is the number of legs, and $p(l)$ is the probability of a leg being selected.

V. Embedded System

To reduce the computational load on the Raspberry Pi and provide real-time control of locomotion, an ATmega2560 microcontroller unit (MCU) was integrated into SigSent. The Pi handles the high-level information such as the direction to travel, the method of locomotion, and the speed to travel. The microcontroller translates that abstraction into coordinated movement amongst the six limbs and four wheels.

Data is transferred from the Pi to the MCU over a Serial Peripheral Interface (SPI) bus. Depending on the data to be sent, a header is generated informing the MCU on how to parse the messages that follow. Bitmasks are defined on the two devices that signify message types and preset constants. Figure 7 displays the bit encoding structure constituting a specific message header.

Fig. 7: Components of an 8-bit header message

High-level functions in Python turn message encoding into a few abstracted method calls where the program simply provides booleans on what items are enabled. The bitstring is then encoded with the specified components. The header consists of a single byte. Every command requires an additional byte then to encode its command data. The driving move command requires a third byte signifying magnitude as it is the only movement that has a speed parameter.

Packets are then sent synchronously to the MCU. Initially, a message header is expected, so the first byte received by the MCU is parsed as one. From there, booleans are set so that next time the SPI interrupt is called, the data byte is parsed for the correct information that it contains. If a byte ever fails to be parsed, the MCU aborts the current

message type it had expected and falls back to wait for a correct header to arrive instead. After successfully parsing a message, the relevant functions are called with the packets data and the SPI interrupt is set to expect a header next again.

Bitmasks were designed to maximize the hamming distance (HD) of the code set so that errors could be more easily detected. With a minimum HD of four among the command codes that are expected by the parser at the same time, an error of three flipped bits can be detected and ignored. This is important for SigSent as incorrect commands could easily lead to destructive movements. Redundant bits and code words with large distances reduce the probability that an incorrectly parsed message will be used [11].

SigSent's walking gait is hard-coded in memory on the MCU. The motors' electronic speed controllers (ESCs) are set in a brake mode to restrict the wheels from spinning while walking. When a walking command is received, the gait is iterated through. During teleoperation, SigSent's walking gait will continue to be stepped through as long as a new movement command has been received from the users joystick. If the operator decides to rotate the robot, SigSent will enter its neutral standing pose and turn. The neutral pose acts as a state that SigSent can enter before any type of change to its movement that does not follow the programmed gait sequence. By always returning to a defined pose, transitions out of the gait are guaranteed to succeed as the intermediary leg movements have already been verified to work.

In drive mode, the robot can receive messages that relay a direction and speed for the ESCs. Tank drive controls are used to skid-steer the robot by providing different speed messages to each side of the robot such that SigSent is propelled more toward one direction over the other.

VI. Human-Robot Interaction

SigSent can be controlled through a variety of interfaces. Primary interaction is facilitated by the basestation's GUI. Additionally, simple voice commands can be issued to SigSent.

A. GUI

Interaction between the operator and robot is facilitated by a custom-designed graphical user interface (GUI), created using Python language bindings for the Qt framework. The GUI's embedded widgets each showcase a unique feature of SigSent. The GUI is shown in Figure 8.

The live camera feed reads in images from SigSent and analyzes them with a basic pedestrian detection classifier from OpenCV. Pedestrians are overlaid with rectangles on the image view to illustrate their presence. The Histogram

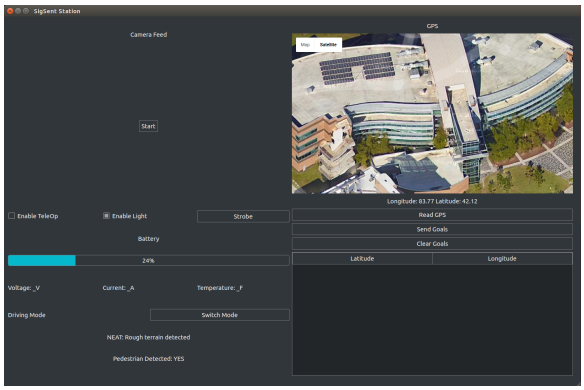

Fig. 8: Qt GUI running in a Ubuntu VM, launched with Python

of Oriented Gradients method is used to capture human-like shapes on a normalized image to account for light differences in the RGB channels [12]. A text label is highlighted in red when there are no pedestrians detected in the current frame. The label is set to green to alert the operator of a pedestrian being observed.

Teleoperation can be activated through a simple checkbox that begins sending joystick axis data to a ROS node that converts it into a geometry message which is then consumed by SigSent, causing motion. In walking mode, the joystick will only allow forward linear motion as well as angular velocity to the left or right. Drive mode includes a reverse capability. Both modes use "tank" controls where the robot must be stationary to enact a rotational turn, rather than strafing while moving forward/backward. This simplifies movement and is a safer alternative to prevent harmful rolls on the vehicle. While walking, the force on the joystick has no effect on the speed of travel. The speed is affected while driving however.

The integrated lightbar can be toggled on or off via a checkbox on the GUI. A strobe button is also provided to allow for a quick, sequential flashing of the light. This is intended to visually alert pedestrians or temporarily disorient trespassers.

A battery meter and accompanying text labels display the remaining capacity and recent readings on voltage, current, and temperature measured by the integrated fuel gauge.

Google Maps was integrated into the GUI to provide an intuitive interface to monitor the location of SigSent and place GPS waypoints for the robot to patrol. Qt is able to interact with the Javascript in the web view through linked Python code within the widget. With this, Google Maps was a natural choice due to its rich API. Extending it to provide map marker placements and retrieving GPS coordinate values along those points required marginal changes to the code. A Qt table view displays latitude and longitude values

for all of the placed waypoints. The user can then send the goals to a ROS topic as a packaged message containing an array of GPS waypoint objects, an included message type in ROS. The list can also be cleared at any time with a separate button to reset into a new path. In Figure 9, the blue markers signify locations with the given coordinates in the table below the map.

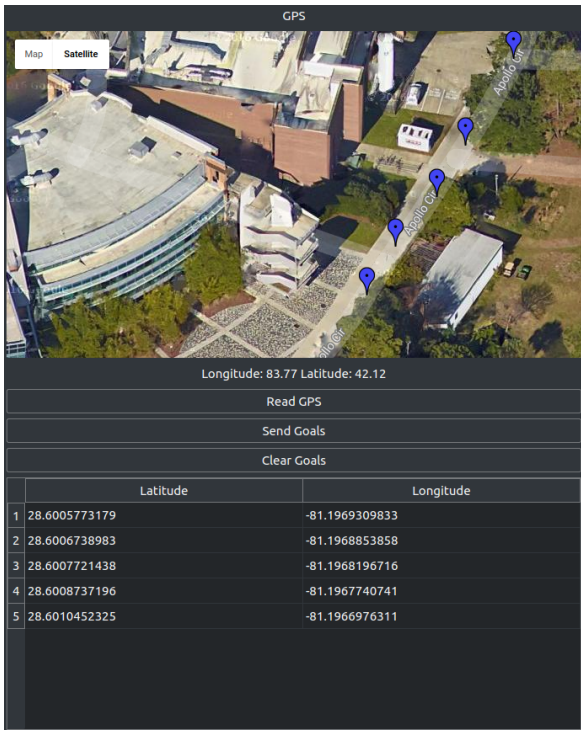

Fig. 9: GPS waypoints spread along UCF's outer ring with associated coordinate values

The interface provides a label displaying the binary output from the NEAT terrain classifier. The label states whether the terrain being traversed has been classified as smooth or rough. This will alert the operator to alter the robot's chosen mobility mode This mobility change is not automatic to ensure that the robot does not begin changing poses while it is not safe to do so, in the case of an unsafe position over the terrain or an incorrect classification.

B. Voice Command

SigSent can be directed to perform simple actions upon voice command issued by the operator through the basestation's microphone. Speech recognition is achieved by utilizing Pocketsphinx, an embedded version of CMU Sphinx, to perform keyword spotting [13]. Pocketsphinx is loaded with the collection of words to be used in commands and their pronunciations. These pronunciations are standardized for Sphinx and are pulled from the CMU Pronouncing Dictionary using the CMU Lexicon Tool [14]. Pocketsphinx is then loaded with commands to be recognized which are composed of the previously loaded words. Individual recognition thresholds are necessary for every desired command. In order to tune these thresholds for sufficient detection rate with minimal false positives, a tool created by Pankaj Baranwal was utilized which automated the process [15].

Voice commands were developed to operate SigSent's integrated lightbar and to switch modes of locomotion. Command phrases were designed to be short and phonetically dissimilar from one another to maximize the command detection rate and minimize incorrect command detection.

VII. Conclusion

Robust robotic systems are desirable in any application where autonomy can remove the need for human intervention. In the case of a security sentinel, SigSent is able to traverse various potentially dangerous environments in lieu of its operators, keeping security personnel safe inside their base of operations.

With more time, SigSent could be sleeker, lighter, and better performing. Our limited budget necessitated design compromises, including lower power servos and heavier structural materials. The computational resources used were low-cost and commensurately lacking in computational power. Our microcomputer could be supplemented with a GPU board to aid in the computer vision bottlenecks that we ran into, and to meet the computational demands of the intelligence system addition. We were able to work around much of these deficiencies by offloading computation to another server, however, this prevents the SigSent robot from operating independently of a basestation.

The AX-12A servos also did not meet our demands despite their advertised specifications. Servos that can exert a greater torque than the 1.5 Joules of the AX-12A are necessary to keep SigSent standing at its current weight of 5.8kg.

The intelligence platform was also not able to be implemented into SigSent's final design. Due to the robot not being able to currently support its weight, data could not be gathered to train the NEAT networks or optimize its walking gait.

Despite these challenges, SigSent was able to effectively complete its overall goal of an autonomous sentinel and patrol robot by successfully patrolling and pathing known and unknown GPS waypoint paths, interacting and detecting pedestrians and provided telecommunication and aversion tools for its operator to interact with suspects, as well as designing a valid platform for mixed walk/drive functionality.

SigSent's multimodality function is a potentially useful form of robotics that has not yet been fully explored. With this proof of concept project, we have found that it could have uses for military Intelligence, Surveillance, Reconnaissance (ISR), Search and Rescue, and medical delivery roles. With a larger budget and more time, SigSent could become a powerful tool for many organizations.

Additionally, more research should be placed into the human-robot interaction of this robot to determine how to best reduce cognitive load and stress for the humans in the loop.

ACKNOWLEDGMENT

SigSent would not have been possible without the considerable funding support provided by Vision Land Service located in Winter Park, Florida. We are very grateful for their generosity and appreciate their motivation to apply robotics to solve novel problems in unique fields.

Additionally, the team is dearly grateful for the support of the Robotics Club at UCF and UCF's Texas Instruments Innovation Lab who shared countless resources and facilities for our team to work with.

REFERENCES

[1] U. Saranli, M. Buehler, and D. E. Koditschek, "Design, modeling and preliminary control of a compliant hexapod robot," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 3. IEEE, 2000, pp. 2589–2596.

[2] "Webench power architect." [Online]. Available: http://www.ti.com/design-tools/webench-power-design/power-architect.html

[3] "News press releases," Apr 2007. [Online]. Available: https://web.archive.org/web/20070429055410/http://www.us.playstation.com/News/PressReleases/396

[4] "Utm-30lx." [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=169

[5] "Sparkfun venus gps with sma connector." [Online]. Available: https://www.sparkfun.com/products/11058

[6] "Mpu-9250 datasheet." [Online]. Available: https://www.invensense.com/download-pdf/mpu-9250-datasheet/

[7] "Gmapping and rplidar." [Online]. Available: http://www.geduino.org/site/archives/35

[8] "Use of simulators in robotics on the example of simulator of gazebo and the darwin-op robot playing soccer," Jun 2015. [Online]. Available: http://developers-club.com/posts/258911/

[9] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[10] A. Manglik, K. Gupta, and S. Bhanot, "Adaptive gait generation for hexapod robot using genetic algorithm," in *Power Electronics, Intelligent Control and Energy Systems (ICPEICES), IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[11] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[12] Satya Mallick, "Histogram of oriented gradients," https://www.learnopencv.com/histogram-of-oriented-gradients/, 2016, [Online; accessed 13-April-2018].

[13] "Pocketsphinx - sphinx for handhelds." [Online]. Available: http://www.speech.cs.cmu.edu/pocketsphinx/

[14] "Logios lexicon tool." [Online]. Available: http://www.speech.cs.cmu.edu/tools/lextool.html

[15] P. Baranwal, "Automatic tuning of keyword spotting thresholds pankaj baranwal medium," Jul 2017. [Online]. Available: https://medium.com/@PankajB96/automatic-tuning-of-keyword-spotting-thresholds-a27256869d31

Joshua Lee Franco is a senior Electrical, Computer, and Mechanical Engineering student at the University of Central Florida. He is he currently working as Lab Technician at the Texas Instruments Innovation Lab. He will be joining a Guidance, Navigation, and Controls group during the Summer of 2018 at Lockheed Martin with its Space System Company. He is also looking to attend graduate school in the coming fall of 2018 for a degree in Robotics.

John Millner is a senior Electrical and Computer Engineering student at the University of Central Florida. He is currently working as an undergraduate research assistant for UCF's center for microgravity research designing power systems for SurfSat. John is a former President of the Robotics Club at UCF. He has led its AUVSI RoboSub Team, and has worked on its Intelligent Ground Vehicle, International Aerial Robotics Competition, and DemoBot teams for the club.

Jeffrey Strange Jr. is a senior Electrical Engineering student at the University of Central Florida, graduating with University Honors in May of 2018. He will begin supporting the F-35 program as a Systems Engineering Associate with Lockheed Martin in Fort Worth, Texas in June of 2018. Previously, Jeff participated in the UCF Lockheed Martin College Work Experience Program, interned with Abacus Technology Corporation, and worked in UCF's Center for Microgravity Research.

Richard Wales is a senior Computer Engineering student at the University of Central Florida, graduating summa cum laude in Spring 2018. He begins a Software Development Engineer role in Seattle, Washington with Amazon in August. He worked with Dr. Sean Szumlanski as a teaching assistance for Computer Science 1 and is currently exploring research in cellular automata based file compression under Dr. Annie Wu.