Development and demonstration of autonomous behaviors for urban environment exploration

Gaurav Ahuja, Donald Fellars, Gregory Kogut, Estrellina Pacis Rius*,
Misha Schoolov, Alexander Xydes

Space and Naval Warfare Systems Center Pacific, San Diego, CA 92152, U.S.A.

ABSTRACT

Under the Urban Environment Exploration project, the Space and Naval Warfare Systems Center Pacific (SSC-PAC) is maturing technologies and sensor payloads that enable man-portable robots to operate autonomously within the challenging conditions of urban environments. Previously, SSC-PAC has demonstrated robotic capabilities to navigate and localize without GPS and map the ground floors of various building sizes.[1] SSC-PAC has since extended those capabilities to localize and map multiple multi-story buildings within a specified area. To facilitate these capabilities, SSC-PAC developed technologies that enable the robot to detect stairs/stairwells, maintain localization across multiple environments (e.g. in a 3D world, on stairs, with/without GPS), visualize data in 3D, plan paths between any two points within the specified area, and avoid 3D obstacles. These technologies have been developed as independent behaviors under the Autonomous Capabilities Suite, a behavior architecture, and demonstrated at a MOUT site at Camp Pendleton. This paper describes the perceptions and behaviors used to produce these capabilities, as well as an example demonstration scenario.

Keywords: robotics, urban environment, exploration, mapping, localization, autonomy, lidar, SLAM

1. INTRODUCTION

As global conflicts move into more urban settings, unmanned ground vehicles (UGVs) must operate in and around buildings that pose unique challenges to navigation and localization. The SSC-PAC Urban Environment Exploration (UrbEE) project is designed to enhance current teleoperated systems with semi-autonomous behaviors to more effectively operate within urban environments. This project enables intelligent navigation in the presence of unreliable GPS, regularly caused by satellite occlusion, urban canyons, multipath, etc. Since mission operations in urban areas may occur inside buildings, robots will also need to seamlessly operate in both outdoor and indoor settings without imposing additional workload on the operator. Building types can also range greatly in size, number of floors, and degree of clutter/rubble, which means indoor navigation needs to be robust to these varying structural characteristics.

The solution described in this paper focuses on the use of low-cost sensors paired with sophisticated software to localize and map in real-time throughout multiple stories of multiple buildings. A horizontally mounted Hokuyo lidar is paired with a vertically mounted Hokuyo lidar as an inexpensive way for 3D perception of the environment. In addition, a pair of stereo cameras is mounted on the vehicle to perceive stairs and control the vehicle while it is climbing up and down staircases. Algorithms and behaviors have also been implemented to detect and traverse stairs, localize, map, avoid obstacles, plan paths, and visualize the data in 3D. This solution was demonstrated on a small UGV at a MOUT site at Camp Pendleton.

Section 2 of this paper describes related research in this field and compares this research to the solutions presented in this paper. Section 3 gives an overview of the software architecture used to implement the algorithms in a modular fashion. Section 4 describes the perceptions and behaviors associated with the resulting UrbEE capabilities. Section 5 describes the algorithms involved in controlling the platform. Section 6 outlines the steps taken to visualize the maps and data in 3D. Section 7 describes the configuration of the platform and environment used for demonstration and testing.

*E-mail: estrellina.pacis@navy.mil, telephone: 1 619 553 2554, URL: http://www.spawar.navy.mil/robots/

2. RELATED WORK

The ability to navigate and explore multi-story buildings requires effective 3D perception of the environment. Where methods for solving the 3D simultaneous localization and mapping (SLAM) problem do exist,[2–4] most have yet to tackle the problem of online SLAM throughout a multi-story building and/or incorporating positive and negative obstacle avoidance. Iocchi et al.[5] describe an approach that combines 2D-SLAM algorithms with visual odometry (VO) and inertial measurement unit (IMU) data to build multi-level planar maps offline. The IMU data was used to detect plane-to-plane transitions and combined with the VO data to determine displacement between planes to align the maps from multiple levels.

Kohlbrecher et al.[6] describe a 2D-SLAM subsystem based on occupancy grids but has not yet been applied to multiple stories. Kohlbrecher describes the use of an architecture similar to the Autonomous Capabilities Suite (ACS), later described in this paper, but contains a full 6DOF navigation subsystem. This navigation subsystem is based on an extended Kalman filter similar to that of ACS, but unlike ACS, the runtime rate cannot be adjusted.

Karg et al.[7] use graph-based SLAM and Monte Carlo localization to align maps from multiple levels of a building. This approach uses a particle filter to identify similar structural properties of different stories and align the maps of the separate levels. This method, unfortunately, may have difficulties in buildings with strong symmetries. The implementation presented in this paper addresses this alignment problem by combining the different stories of a building through landmarks on adjacent levels placed at the locations where the robot detects stairs.

A completely probabilistic approach[8] has been implemented that relies exclusively on multi-level surface (MLS) maps, based on regular elevation maps, for localization. This approach uses low-cost hardware to perform online SLAM through multiple multi-story buildings, as well as positive and negative obstacle avoidance.

3. ARCHITECTURE AND IMPLEMENTATION

The UrbEE solution was implemented with the Autonomous Capabilities Suite (ACS), a robot behavior architecture developed by SSC-PAC. ACS is a modular software architecture and toolset that supports the development, testing, and deployment of autonomy. ACS provides common robot architecture features, such as hardware abstraction, the composition of behaviors into well-defined perceptual and behavorial modules, and behavior arbitration via XML-defined state machines.

3.1 Abstraction

ACS provides a three-level taxonomy for abstracting autonomy into discrete modules: Devices, Perceptions, and Behaviors. Device modules are interfaces to specific pieces of hardware. Perception modules process raw data from device modules or converts data from other perception modules into new types of data (e.g. converting point clouds into obstacle maps). Finally, behavior modules provide actuator commands to achieve a specific goal based on perceptual and device data (e.g. directing a vehicle to a specific point in space). Multiple behaviors can be used to provide a more complex behavior, which is usually referred to as a Task.

Task modules use a sequencing of behaviors and user input to achieve high level capabilities. For example, the UrbEE Exploration task module has the goal of navigating an unknown area in an efficient manner to map it with minimal operator input. Exploration, therefore, must use a variety of behaviors such as waypoint following, obstacle avoidance, navigation to unexplored (or open) space, and stair traversal. Another type of abstraction is a Comms module, which allows all device, perceptions, behaviors, and tasks to be independent of the communications protocol. UrbEE comms modules allow the system to simultaneously integrate with the robot platform (such as an iRobot *PackBot* via *Aware2*), the Multi-Robot Operator Control Unit (MOCU)[9] via the Joint Architecture for Unmanned Systems (JAUS), and to Google Earth and RViz[10] for 3D visualization. Figure 1 is a simplistic illustration of the ACS module construct.

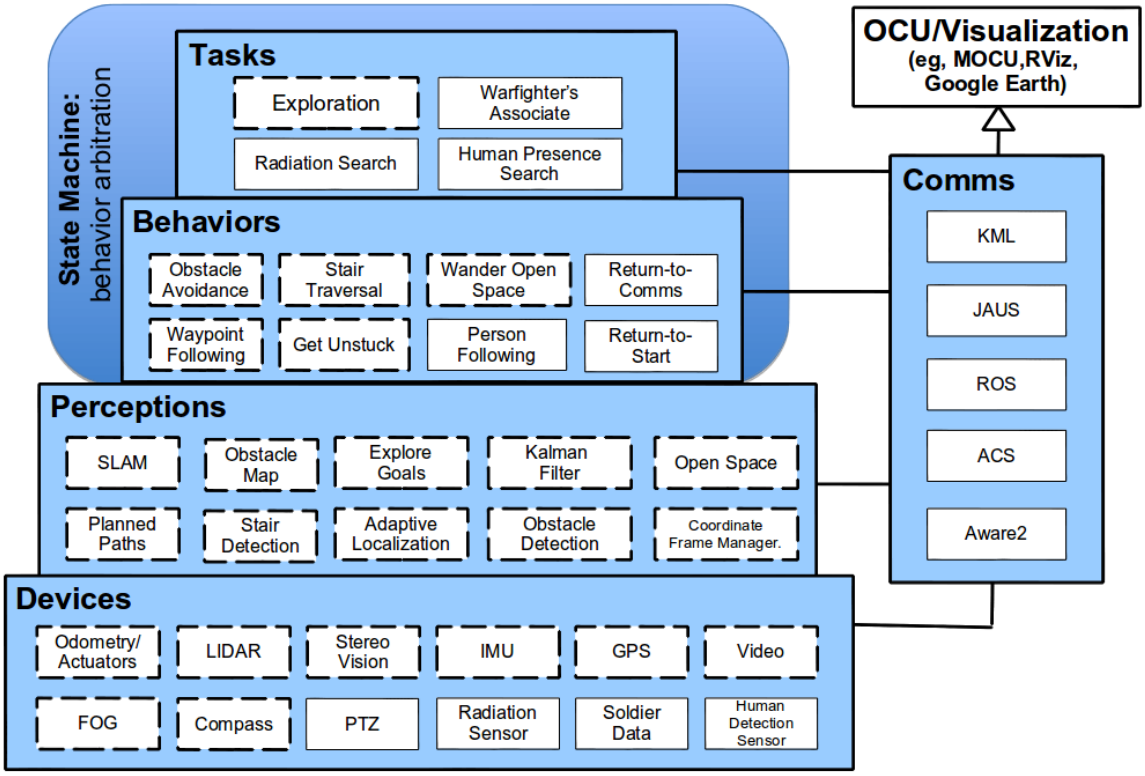


Figure 1: A simplistic illustration of the ACS module construct. Dotted boxes are example modules that are directly tied to the Exploration task described above.

All ACS modules have well-defined interfaces and communicate using a publish-subscribe pattern. The ACS messaging policy uses configuration files to define the connections between the modules, which communications protocol to use, and expand upon existing data types. In addition, ACS has a scalable build system that allows any combination of modules to be loaded at run-time.

3.2 State machine

As mentioned above, sequencing of behaviors is used to produce higher-level tasks. The use of state machines provides the behavior arbitration needed to coordinate among conflicting behavior outputs and user input. In the Exploration task example described above, the obstacle avoidance, waypoint following, and stair traversal behaviors could all produce conflicting goals at any given time. A state machine coordinates the ACS modules to achieve the desired outcome through the use of a Task-State pattern, where a Task defines the larger goal or mission of the vehicle and a State defines a particular configuration of ACS modules. Several tasks are defined under UrbEE, such as the Exploration task. Each Task has a default entry State, with State changes taking place in response to Events. For example, a stairway detected by an ACS perception module produces an Event, which may trigger a transition to a State designed to guide the vehicle safely up the stairway. The state machine definition describing these transitions is stored in an XML file provided at runtime.

3.3 Integration with other architectures

The modularity of the ACS architecture makes it easy to work with other architectures or third-party software, whether it is the loose integration of external capabilities into ACS or the use of ACS modules within third-party architectures. This flexibility allows for seamless partnerships with other government agencies, academia, and industry. A few examples of current architectures with which ACS has been integrated include multiple flavors of JAUS, Player/Stage, and the Robot Operating System (ROS).[10]

The JAUS architecture is an open architecture used by several DoD programs, including the Navy's Advanced Explosive Ordnance Disposal Robotic System (AEODRS) and the Army's Interoperability Profiles (IOPs) for UGVs. ACS has native support for JAUS through tight integration with a version of the JAUS Toolset (JTS) implementation of the Society of Automotive Engineers JAUS (SAE-JAUS). This integration allows developers to easily conform with the JAUS architecture with minimal additional JAUS implementation. While the current JAUS standard only includes minimal support of autonomous behaviors, ACS implements a number of non-standard extensions for autonomy adopted by the AEODRS program. The IOP program is also exploring further extensions to JAUS for autonomous operation.

The open-source ROS is currently the architecture of choice at many academic and R&D organizations and was used under UrbEE for visualization and testing purposes (further described in Section 6.2). ROS is also tightly integrated within ACS, as any ACS module may be used as a ROS node (and vice-versa) with minimal modification.

4. PERCEPTIONS

The exploration of multiple, multi-story buildings required the development of new 3D perceptions, such as 3D obstacle detection and stair detection. Previously developed perceptions[1] were also extended to the 3D domain. This included staying localized while traversing stairs, building and stitching multiple floor maps, re-localizing in recently explored floors, planning paths between floors, and goal planning using floor transfer points. Most of these perceptions required some minimal 3D sensing to detect positive and negative obstacles. These requirements made vision-based perceptions a good fit, but due to time constraints a vertical lidar system was used as an interim solution while vision-based algorithms were being perfected.

4.1 Positive and negative obstacle detection

This solution used a range-based 3D obstacle detection algorithm supported by the vertically mounted Hokuyo lidar. Range readings with a calculated height between a threshold distance above the ground plane and a threshold distance from the top of the robot were considered to be a positive obstacle, whereas range readings with a calculated height below the ground plane within a given threshold distance were considered negative obstacles. Each range reading identified as an obstacle was projected onto the X-Y plane as a 2D obstacle that spanned the width of the robot positioned at the location of the range reading. These 2D-obstacle representations were sent to the obstacle avoidance behavior module, allowing the obstacle avoidance module to take into account 3D obstacles without modification.

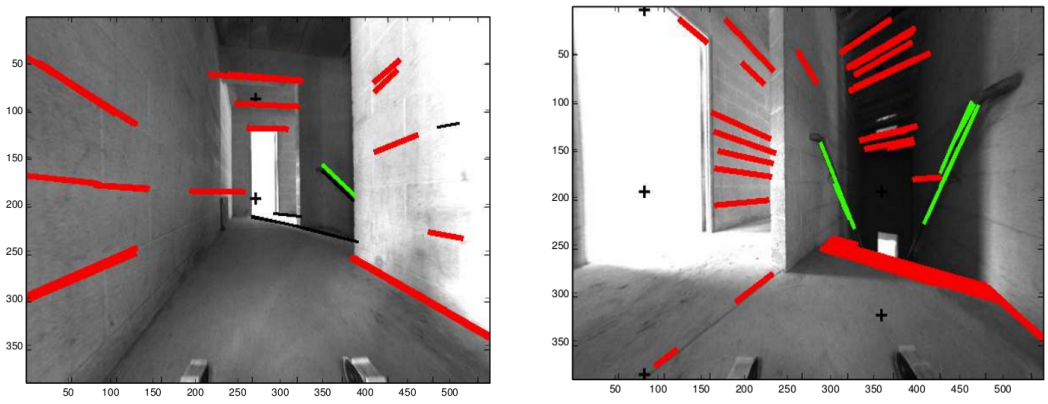
4.2 Stair detection

The exploration of a multi-story building required the development of a reliable method to detect and traverse staircases. Three algorithms of varying complexity and robustness were implemented to detect stairs. The first was developed by the Jet Propulsion Laboratory (JPL) and is based on vision data. The other two were developed by SSC-PAC and are based on lidar data.

4.2.1 JPL method

JPL was funded to develop a vision-based algorithm to detect stairs and provide drive commands that centered the robot while traversing the stairs. JPL implemented two versions of stair detection: one using a stereo pair and the other using a monocular camera. The stereo system leverages previous work that builds a 3D map from the stereo data and detects features that meet the classification requirements of a staircase based on height and slope of the map features. The monocular solution allowed the drive camera typically included on a small UGV to be leveraged, and avoids the addition of a separate camera system.

As part of this project, JPL addressed the need to detect both ascending and descending stairs. While other researchers have investigated methods for infering potential locations of descending stairs based on texture and on line-based optical flow methods,[11] JPL adopted a different approach based both on concern for the achievable



(a) (b)

Figure 2: Descending stair detection using Manhattan-world assumption: (a) descending stairs perpendicular to the robot; (b) descending stairs directly in front of the robot.

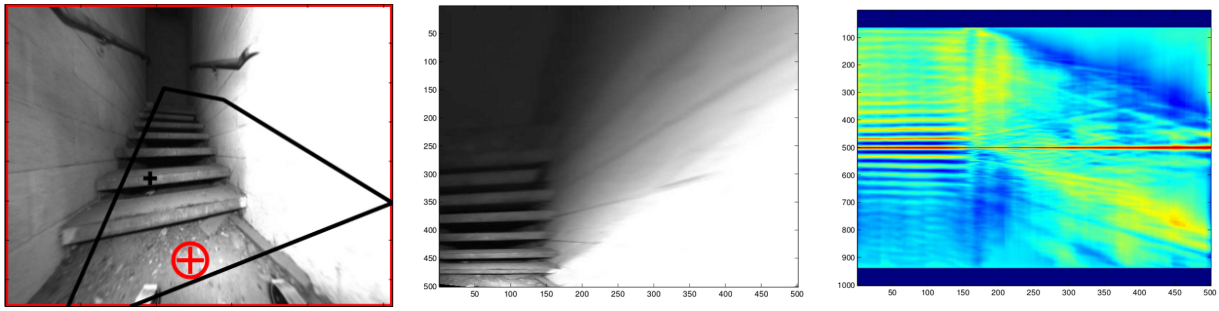
robustness as well as on particular attributes of the larger system design. For example, in the current system, stair detection is performed passively while the robot autonomously maps the environment which would make active control of the viewpoint to aid optical flow-based stair detection difficult. The ascending and descending robot control is largely unchanged from that described by Helmick et al.[12]

The approach JPL used is based on an assumption of a Manhattan-world in which the major planar surfaces are presumed to align with one of three orthogonal directions (two horizontal and one vertical). While not universally valid, the Manhattan-world assumption allows approaches amenable to real-time implementation that are robust to challenging lighting conditions and environments. For efficiency, the current implementation also assumes the robot has zero pitch and roll.

Since it is assumed that the stairwells themselves are also consistent with the two major axes of the building, the robot's orientation (with respect to the building) informs the subsequent search for ascending and descending stairwells. This is particularly important for detection of descending stairwells since the visual indications of a descending stairwell are quite limited, especially so for a small robot whose cameras are very close to the ground. For this reason, JPL's approach to descending detection has thus far been focused on detection of banisters or railings.

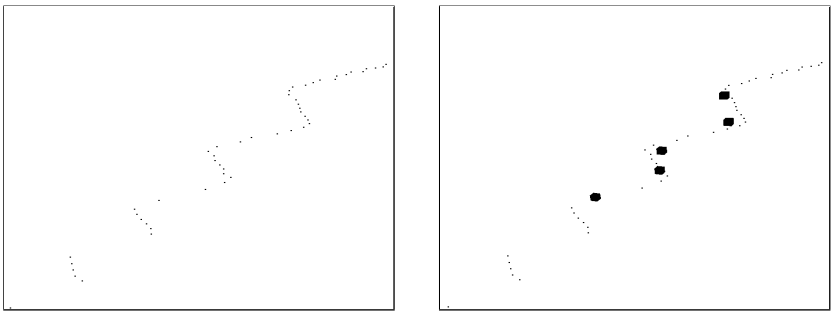
Determining the robot's relative heading is equivalent to identifying the two vanishing points 90-degress apart lying on the horizon. Any image of a horizontal line in the environment (having one of the two primary orientations) will pass through the corresponding vanishing points. Similarly, any image of non-horizontal lines that are parallel to the direction of ascending/descending stairs, such as railings, will pass through a corresponding vanishing point located above/below the corresponding horizontal vanishing point. Figures 2(a) and 2(b) show two cases of detected descending railings. There are some false positives caused by accidental alignments, but these should be transitory.

Detection of ascending stairwells also leverages estimates of vehicle orientation/vanishing point estimation. In addition to the vanishing point based cues (horizontal stair edges must themselves pass through the corresponding vanishing point), the ascending detection algorithm relies on the presence of regularly spaced features on a plane with a slope consistent with stairs (typically 25-35 degrees). Rather than rely on explicit line detection which can be problematic in environments with poor lighting or little texture, JPL's method is based on the following steps: selecting an appropriate sub-area of the image in which stairs are likely to be observed, calculating a homography that maps that sub-area to a rectangular image in which the majority of the perspective effects are eliminated, identifying significant regular horizontal features in that sub-area using auto-correlation and a FFT, identifying the bottom-center of the regular features in the image, and applying the inverse homography to map



(a) (b) (c)

Figure 3: Ascending stair detection: (a) selecting location of stairs, (b) Homography image in which the majority of the perspective effects are eliminated. (c) Using an auto-correlation and a FFT to identify significant regular horizontal features



(a) (b)

Figure 4: Point cloud data from vertically mounted Hokuyo lidar: (a) raw data from the vertical lidar. (b) five corner features extracted using edge and intersection detection.

that point to the original image where it is intersected with the known ground plane to estimate the location of the base of the stairs. Example images for each step are shown in Figures 3(a), 3(b), and 3(c).

4.2.2 SSC-PAC method 1

A second stair detection algorithm was developed by SSC-PAC and uses data from the vertically-mounted Hokuyo lidar. An ACS lidar perception module reads raw data from the Hokuyo lidar, converts the data into a point cloud, and publishes the converted data to the rest of the system via a standard ACS message. The converted point cloud is processed by the feature extractor contained within the stair detection perception module. This module analyzes the two-dimensional point cloud scan for features such as lines and corners. The first pass of the feature extractor analyzes the data for lines, and any lines that intersect within a set distance and angle of each other are identified as corners, as shown in Figure 4.

Once corner features have been identified, this data must be processed to determine if the feature set defines a staircase. To define a staircase, the stair detection algorithm uses a sequence of four points in two possible configurations: "run-rise-run" or "rise-run-rise" (Fig. 5). Several assumptions are made when identifying whether sets of four corners fit one of the two stair configurations. These assumptions come from the observation of standard staircases found in U.S. construction. Typically, all "run" sections are the same size, as are all "rise" sections. Also, a comfortable run:rise ratio is about 2:1, although this ratio can vary depending on the staircase. Finally, stairs are typically built at approximately right angles. By determining whether the corner features match these attributes, the presence of stairs can be identified in the data.

The detection of more than four corner features is handled by iterating through all possible subsets of four features. The four selected corner features are collected and the two maximally distant points are determined.

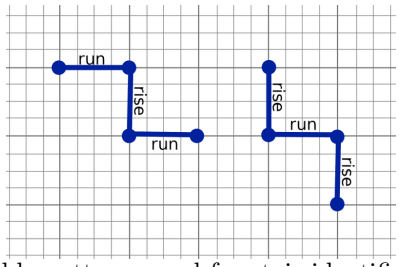


Figure 5: Two possible patterns used for stair identification with 4 points.

(a) (b)

Figure 6: Two types of staircases at the Camp Pendleton MOUT site: (a) Regular staircase and (b) open staircase.

Choosing one of these two points as a starting point, the other points in the array are sorted based on their distance from this start point. The resulting series is then processed to ensure it fits the previous assumptions. If the points fulfill all of the conditions, then a staircase is detected. Though this algorithm accurately detects solid staircases (Fig. 6(a)), it requires modification to work on open staircases (Fig. 6(b)). The current algorithm does not account for the open space between the steps.

4.2.3 SSC-PAC method 2

A second, more simplistic stair detection algorithm was written to reduce the computational requirements for detecting stairs. This algorithm took advantage of the obstacle detection module to focus on range readings that could be useful for stair detection. The presence of a staircase in front of the robot meant that there must be multiple points available below the robot for descending staircases and multiple points above the ground for ascending staircases. If there were not enough such points, there were likely no stairs present; when there were multiple positive or negative points, the actual data to be processed for stair detection was already pruned by the obstacle detection algorithm.

This stair detection method is very similar to method 1 and similar assumptions were used, but the detection method is different. It instead combines the line-detection algorithm with the simple notion of looking for "run-rise-run" or "rise-run-rise", ignoring lidar returns that did not fit with a reasonable bound for a rise or a run. This allowed detecting regular or open staircases using minimal computational resources but increased the possibility of false positives due to relaxed rules for detecting a rise or a run.

4.3 Localization

One problem encountered during testing was maintaining accurate robot localization while traversing stairs. To date, a robust solution has not yet been implemented at SSC-PAC. The localization methods currently being utilized are a 2D simultaneous localization and mapping (SLAM) system and a Kalman filter (KF).[13] The SLAM system is only effective in a flat environment and therefore is not suitable for use while traversing stairs. The KF may be usable for stair traversal localization with the addition of more accurate odometry sensors. The current

UGV test system is only equipped with a wheel displacement encoder, a sensor that is unreliable on stairs due to the greater than average amount of slippage. This error may be mitigated by assuming a uniform height and depth offset for stairs and then reinitializing the KF after completing the traversal using offsets generated by these assumptions. Future work will be done to determine the actual height and depth of the staircase and dynamically define these offsets. At the start of each floor, a new map will be created and the robot can localize using both SLAM and the KF. After returning to a previous floor, the robot will re-localize in the map it had previously created. Ideally, a better solution would be to use a 3D lidar, but this is currently impractical due to size, weight, power, and cost constraints.

4.4 Mapping

Simultaneous localization and mapping (SLAM) is a method of building a map while staying referenced inside the map.[14] A customized and extended version of the *Karto*[15] mapping libraries was developed by SRI International (SRI) for UrbEE to allow for multi-floor/multi-building mapping. This extended version included a few new techniques for correcting the mapper whenever lidar points did not register properly in the existing map, particularly in large multi-building maps. This version also provides additional information, such as a co-variance matrix for each localized pose. In addition to these new mapping features, the *Karto* localization libraries were also extended with 3D localization for multi-floor maps. Multi-floor mapping also added more complexity to the occupancy grid, such as anchor points, transfer points, and occupancy-grid transforms between floors.

Anchor points define the relationship from one map to a coordinate frame or another map. These points serve several other purposes as well. An operator can manually adjust and align two different floor maps to each other using their anchor points; this helps capture and correct the yaw drifts between floors. If given their own GPS coordinates, anchor points can also be used to transform the entire map occupancy grid to GPS coordinates. The map can then be realigned using overhead imagery, which will correct the GPS drifts in urban areas. Similarly, anchor points could be automatically generated from other references, e.g. the four corners of a building, to automatically align the different floors.

Transfer points are locations where a robot may enter or exit a floor, such as the points of origin for a staircase, or the point from where the robot was manually carried to a new, unknown area. A transfer point in one map must always be linked to a transfer point in another map, e.g., the point of origin for a staircase going up in floor1 is linked with the point of origin for a staircase going down in floor2. Currently, all linked transfer point pairs are considered bi-directional by default. For example, the transfer points from floor1 to floor2 are automatically set as available transfer points from floor2 to floor1. This bi-directionality is not always required to allow for one-way transfer points, such as the rare case of a UGV going up and down an escalator.

Several improvements were also made to the ACS modules to manage the multi-floor and multi-building maps. One of the most useful and influential changes was feeding the position estimate from the KF, instead of the raw vehicle odometry, to the SLAM algorithm. The KF has proven to have less than 2% error over distance traveled on planar surfaces, which resulted in more accurate SLAM maps and more robust localization. Based on the staircase assumptions, a few additions were also made to the KF algorithm to manage the huge drifts from climbing and descending stairs. These changes helped stabilize the x, y, z and yaw drifts that were nearly impossible to account for within the original 2D KF.

The ACS mapping perception module was also extended to handle autonomous transitions between floors and buildings by adding new ACS datatypes and lookup tables for buildings, floors, anchors, and transition points. ACS state machine handlers in the mapping module were extended to start, stop, and resume mapping on different floors and buildings based on state machine events, KF data, and coordinate-frame transform-manager data. Several other ACS modules were also extended to ACS_ROS hybrid modules; these incorporate ROS nodes for utilizing ROS utilities, which were used for data recording and 3D data visualization.

4.5 Path planning

SRI *Karto's* gradient path planning libraries were also extended to support the new mapping libraries. These extensions allowed the robot to plan paths between points on different floors of one building, as well as between

points in different buildings, but do not plan the external path between the buildings. *Karto* leaves this up to the robot's onboard autonomy. The ACS path planning perception module extends the *Karto* path planner to plan this external path provided that the robot has already travelled between the buildings. Future improvements to the ACS associated with path planning perception module include extending the 2D breadcrumbs module to 3D, which stores a trail of positions the robot has traveled with the associated distances between each position, and creating detailed stair data in the stair detection modules during stair traversal.

5. TASKS AND BEHAVIORS

5.1 Stair traversal

A simple stair traversal behavior module was developed to mount and traverse stairs. When the robot is at the base of a set of ascending stairs, the robot actuates the flippers to a mounting position roughly 45 degrees above the horizontal plane. The pitch of the robot is used to determine if it successfully mounted the first step. The flippers are then positioned back near horizontal, and the robot proceeds forward up the stairs. The measured roll from a coordinate frame manager perception is used to determine the turnspeed to keep the robot heading up the stairs so it does not turn sideways due to slipping or bouncing. The observed pitch is used to determine when the robot has finished traversing the stairs. A more sophisticated stair traversal behavior is implemented using the JPL libraries. In addition to the simple behavior described, the JPL libraries output commands to navigate to the base of the stairs and keep the robot centered on the stairs while traversing.

5.2 Exploration

As described in Section 3.1, an exploration task module utilizes numerous behaviors and perceptions described in this paper. Single-floor exploration is achieved by determining the location of unknown areas and navigating to those locations until a full map is completed.[1] Exploring across multiple stories introduces more complexities, such as the need to identify transition points, namely stairs, between each floor. These transition points can be located by the stair detection perception module, but the location may be inaccurate when detected from afar. To remove this inaccuracy, the robot will pause its current state in the exploration task and navigate to the detected stairs to confirm their presence and store their location. The confirmed stairs are given the lowest priority of explore points, which are stored in the explore goals perception module, to ensure the entire floor is explored before traversing stairs to a new floor. Once all of the unknown areas on a floor are explored based on an open space perception, the robot will navigate back to any stairs that have not been traversed and autonomously climb or descend to the next floor.

6. 3D VISUALIZATION

6.1 *Google Earth*

The robot and resulting map were visualized using *Google Earth*, a widely available cross-platform tool for visualizing the earth. While satellite imagery of the earth is a basic feature of *Google Earth*, robot position and scans of the local environment were added to the map using KML, an XML notation for displaying geographic data in *Google Earth*. KML allows for waypoints, building maps, models, and paths to be displayed on top of the satellite imagery, giving a comprehensive three-dimensional view of the robot and its environment. These visualizations are created by a KML comms module. This module writes the robot's positional and environmental data to two separate KML-formatted text files which are read by *Google Earth* over KML Network Links.

The first file generated by the KML comms module contains the robot's position and the path that the robot has traversed since visualization started (Fig. 7(a)). The robot's current position is represented by a 3D avatar of the robot in its current pose, displayed as a KML Placemark. The Placemark of the current position contains data for the robot's location, orientation, and scale, allowing for the complete physical state of the robot in its environment to be displayed on the map. Trailing it, the path of previous positions is displayed as a KML LineString. This text file is updated with new data as the robot moves.

(a) (b)

Figure 7: *Google Earth* is used to visualize the robot, its path, and the map data. (a) Closeup of robot and map on the ground floor of a building at Camp Pendleton. (b) Overview of all four floors after a complete run.

The second file generated by the KML comms module contains the KML representation of the robot's surrounding environment. A 2D scan of the robot's environment is obtained in real time using a lidar. The data from the lidar scan is then converted into a list of polygons, and those polygons are used to form KML LineStrings. On the ground floor, the resulting KML LineStrings are extended from a height of one meter down to the ground to form walls. Multi-floor maps can be visualized by varying the height of the LineStrings (Fig. 7(b)). As the robot moves to the second story of a building, the lidar data from that second floor is displayed in a different color and at a different altitude, thus differentiating individual floors.

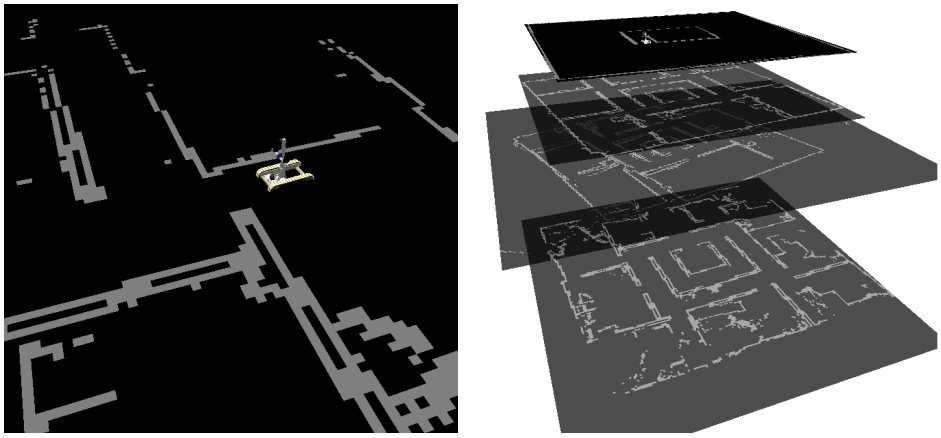
These two KML-formatted text files reside on-board the robot. Since KML Network Links are used to access these text files, the data can be displayed on any system that has a network connection to the robot. Using CGI scripting, the contents of the two text files are served to the visualization system, giving a remote operator a georeferenced view of the robot's position, environment, and path traveled.

Further enhancements are made possible by using *Google Earth's* Add Placemark or Add Path feature to specify a path for the robot to follow. The operator can click on the map to specify coordinates for the robot to visit. These coordinates are saved to a KML file as GPS coordinates, and then passed via network connection to the waypoint behavior module and explore goals perception.

6.2 ROS and RViz

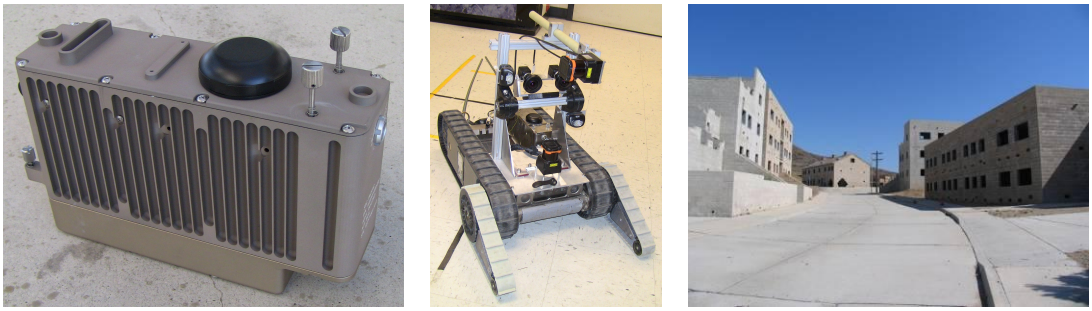
Another widely used tool for 3D visualization is ROS's RViz, described as a 3D visualization environment for robots using ROS.[10] To use this tool, several ACS modules (e.g. KalmanFilter, Mapping, CoordinateFrameManager, PathPlanning, StairDetection, and device drivers) were extended to become ACS_ROS hybrid modules. Each of these ACS_ROS modules now publish several ROS topics for visualizing inputs and outputs for each module. For purposes of visualizing the robot as it traversed a building and built the multi-floor maps, ROS messages for robot position and the environment map were published by the ACS_ROS modules. Several additional ROS messages were added to support visualizing a 3D robot in RViz. The resulting visualization of a robot executing an exploration task can be seen in Figure 8(a), and of a map with multiple floors in Figure 8(b).

Originally, RViz was intended for visualizing the robot and the maps created during multi-floor mapping. However, because of its ease of use, it became very handy to use as a debugging tool as well. A new ACS_ROS module is being created to utilize ROS interactive markers for RViz, with the intent to use it as a more interactive tool for additional behavior development work.



(a) (b)

Figure 8: ROS RViz is used to visualize the robot, it's path, and the map data. (a) Closeup of robot and map on the ground floor of a building at Camp Pendleton. (b) Overview of all four floors after a complete run.



(a) (b) (c)

Figure 9: (a) SSC-PAC autonomy payload and (b) UrbEE test platform. (c) MOUT Test site at Camp Pendleton.

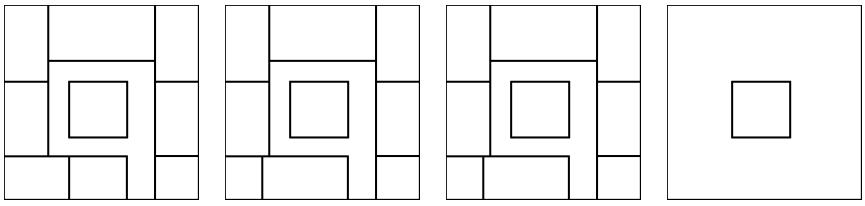
7. DEMONSTRATION

7.1 Platform

The UrbEE test platform consisted of two parts: an iRobot *Packbot Scout* as the base platform and SSC-PAC's autonomy payload (Fig. 9(a)). The autonomy payload contains a *KVH DSP-3000* fiber-optic yaw-rate gyroscope, a *MicroStrain 3DM-GX2* IMU, a Ublox GPS, and an Intel Core Duo processor board. These sensors and processor are contained within a 3 in x 6.5 in x 10 in box. The platform was equipped with two Hokuyo lidars, one mounted horizontally and one mounted vertically, and a pair of stereo cameras from Point Grey (Fig. 9(b)).

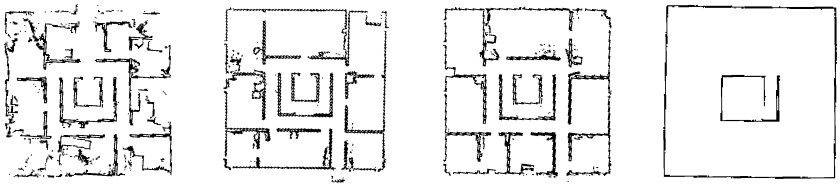
7.2 Environment

A Military Operations in Urban Terrain (MOUT) site at Camp Pendleton was used for demonstration and testing throughout the UrbEE project (Fig. 9(c)); this site is currently used for military training, making it ideal for robotic field testing. The range consists of residential sections, including a gas station, houses, apartment buildings, a school, and a playground, as well as a business district consisting of a hotel, office buildings, and a town square. There are a total of 29 buildings, ranging from 1 to 3 stories. 14 are intact and 15 have been partially damaged. In addition, there are nine ghost buildings to simulate structures that have been completely destroyed.



(a) Floor 1 (b) Floor 2 (c) Floor 3 (d) Floor 4

Figure 10: Partial ground truth illustrations of the hotel building used for testing.



(a) Floor 1 (b) Floor 2 (c) Floor 3 (d) Floor 4

Figure 11: The resulting maps from a demonstration run of the hotel building at the Camp Pendleton MOUT site.

7.3 Partial ground truth

All of the testing described in this paper has been done in the hotel building at the MOUT site, which contains three main floors, plus an accessible roof. Partial ground truth data for the building was established by measuring the dimensions of the building and each room (Fig. 10). These measurements do not take into account the 8-inch thick walls. (Full ground truth data was not established because of time constraints.)

7.4 Demonstration scenario

To demonstrate and test the algorithms mentioned in this paper, the robot was started just inside the front door of the ground floor of the hotel described in Section 7.3. Instead of using the Exploration behavior described earlier, the robot was commanded to enter the Shared Search mode in which the behaviors would decide where to go based on the same perceptions used by the Exploration task, but the operator could command the robot to move in a preferred direction if so desired. The operator could also draw a search zone on the operator control unit (OCU) to define an entire area for the robot to explore. Map data recorded from one of the demonstration runs shows a layout (Fig. 11) similar to that of the partial ground truth (Fig. 10).

8. CONCLUSION

Since 2007, SSC-PAC has been developing intelligent robotic capabilities under the UrbEE project to provide a wide-range of enhanced perceptions and behaviors that enable semi-autonomous to fully autonomous operation of small UGVs in GPS-denied urban areas. These capabilities include positive and negative obstacle avoidance, waypoint navigation, GPS-denied pose estimation and localization, path planning, large-scale multi-story mapping, autonomous exploration, stair detection and traversal, return-to-communications, and return-to-start.

In addition to the regular demonstrations and performance tests conducted at Camp Pendleton, UrbEE has participated in various user experiments, such as the Army Expeditionary Warfighter Experiment, Spiral F in January, 2010, and a joint user-study with the Army Research Laboratory (ARL) Human and Research Engineering Directorate on the effects of progressive levels of autonomy on robotic reconnaissance in September, 2009. Results from these experiments showed that a mapping capability was key to aiding the operator during reconnaissance missions. Though the purpose of the reconnaissance missions was never solely to produce a floorplan, the mapping feature greatly enhanced operator understanding of where the robot was and had been as the operator searched the building using the robot's cameras. In those experiments the robotic asset was left

behind due to the limitations of the UGV operating in a large-scale area and the increased workload placed on the operator by operating the UGV in the multi-story/multi-building environment. The work described in this paper was an effort to specifically address these technology gaps.

The modular software framework within the Autonomous Capability Suite (ACS) was used to ensure each capability developed was self-contained and vehicle independent, allowing any combination of perceptions and behaviors to be applied to a specific mission application, vehicle type, etc. This flexibility also allowed for the incorporation of new advances in computational hardware and sensors, plus expands upon previous algorithmic work. This approach has enabled SSC-PAC to deliver different subsets of UrbEE capabilities, allowing application of UrbEE software to new R&D autonomy projects, and technology transfer to EOD and Countermine Programs of Record.

ACKNOWLEDGMENTS

The authors of this paper would like to acknowledge the Office of Secretary of Defense (OSD) Joint Ground Robotics Program for their sponsorship, the Camp Pendleton MOUT site 131 Range Mayor, his staff, and the Range and Training Area Management Division for demonstration and testing support, as well as SRI International and the Jet Propulsion Laboratory for their technology contributions.

REFERENCES

[1] Ahuja, G., Fellars, D., Kogut, G., Pacis Rius, E., Sights, B., and Everett, H. R., "Test Results of Autonomous Behaviors for Urbane Environment Exploration," in [*Unmanned Systems Technology XI*], Gerhart, G. R., Gage, D. W., and Shoemaker, C. M., eds., *Proc. SPIE* **7332**(1), 73321A (2009).

[2] Newman, P., Cole, D., and Ho, K., "Outdoor SLAM Using Visual Appearance and Laser Ranging," in [*Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*], 1180 – 1187 (May 2006).

[3] Konolige, K., Agrawal, M., Bolles, R., Cowan, C., Fischler, M., and Gerkey, B., "Outdoor Mapping and Navigation Using Stereo Vision," in [*Experimental Robotics*], Khatib, O., Kumar, V., and Rus, D., eds., *Springer Tracts in Advanced Robotics* **39**, 179 – 190, Springer Berlin / Heidelberg (2008). 10.1007/978-3-540-77457-0_17.

[4] Weingarten, J. and Siegwart, R., "EKF-based 3D SLAM for Structured Environment Reconstruction," in [*Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*], 3834 – 3839 (Aug 2005).

[5] Iocchi, L., Pellegrini, S., and Tipaldi, G., "Building Multi-Level Planar Maps Integrating LRF, Stereo Vision and IMU Sensors," in [*Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*], 1 – 6 (Sep 2007).

[6] Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U., "A flexible and scalable slam system with full 3d motion estimation," in [*Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*], 155 –160 (nov. 2011).

[7] Karg, M., Wurm, K., Stachniss, C., Dietmayer, K., and Burgard, W., "Consistent Mapping of Multistory Buildings by Introducing Global Constraints to Graph-Based SLAM," in [*Robotics and Automation (ICRA), 2010 IEEE International Conference on*], 5383 – 5388 (May 2010).

[8] Pfaff, P., Kmmerle, R., Joho, D., Stachniss, C., Triebel, R., and Burgard, W., "Navigation in combined outdoor and indoor environments using multi-level surface maps," (2008).

[9] Powell, D., Gilbreath, G., and Bruch, M., "Multi-Robot Operator Control Unit for Unmanned Systems," *Defense Tech Briefs* (Aug 2008).

[10] `http://www.ros.org/`.

[11] Hesch, J., Mariottini, G., and Roumeliotis, S., "Descending-stair detection, approach, and traversal with an autonomous tracked vehicle," in [*Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*], 5525 –5531 (oct. 2010).

[12] Helmick, D., Roumeliotis, S., McHenry, M., and Matthies, L., "Multi-sensor, high speed autonomous stair climbing," in [*Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*], **1**, 733 – 742 vol.1 (2002).

[13] Pacis, E. B., Sights, B., Ahuja, G., Kogut, G., and Everett, H. R., "An Adapting Localization System for Outdoor/Indoor Navigation," in [*Unmanned Systems Technology XI*], Gerhart, G. R., Gage, D. W., and Shoemaker, C. M., eds., *Proc. SPIE* **6230**(2), 623022 (2006).

[14] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B., "Fastslam: A factored solution to the simultaneous localization and mapping problem," in [*In Proceedings of the AAAI National Conference on Artificial Intelligence*], 593–598, AAAI (2002).

[15] `http://www.kartorobotics.com/`.