Progress towards multi-robot reconnaissance and the MAGIC 2010 Competition

Edwin Olson Johannes Strom Ryan Morton Andrew Richardson

Pradeep Ranganathan Robert Goeddel Mihai Bulic

Department of Computer Science and Engineering

University of Michigan

{ebolson,jhstrom,rmorton,chardson,rpradeep,rgoeddel,mihai}@umich.edu

http://april.eecs.umich.edu

Jacob Crossman Bob Marinier

Soar Technology

{jcrossman,bob.marinier}@soartech.com

http://www.soartech.com

Abstract

Tasks like search-and-rescue and urban reconnaissance benefit from large numbers of robots working together, but high levels of autonomy are needed in order to reduce operator requirements to practical levels. Reducing the reliance of such systems on human operators presents a number of technical challenges including automatic task allocation, global state and map estimation, robot perception, path planning, communications, and human-robot interfaces. This paper describes our 14-robot team, designed to perform urban reconnaissance missions, that won the MAGIC 2010 competition.

This paper describes a variety of autonomous systems which require minimal human effort to control a large number of autonomously exploring robots. Maintaining a consistent global map, essential for autonomous planning and for giving humans situational awareness, required the development of fast loop-closing, map optimization, and communications algorithms. Key to our approach was a decoupled centralized planning architecture that allowed individual robots to execute tasks myopically, but whose behavior was coordinated

centrally. In this paper, we will describe technical contributions throughout our system that played a significant role in the performance of our system. We will also present results from our system both from the competition and from subsequent quantitative evaluations, pointing out areas in which the system performed well and where interesting research problems remain.

1 Introduction

In 2001, the United States Congress mandated that one-third of all ground combat vehicles should be unmanned by 2015. The Defense Advanced Projects Research Agency (DARPA) identified autonomy as a key technology for meeting this challenge. In order to accelerate the development of these technologies, they sponsored a series of now-famous "Grand Challenges" in which teams built autonomous cars to drive along mountainous desert roads (in 2004 and 2005) and in urban environments (in 2007) (Thrun et al., 2007; Urmson et al., 2008).

These grand challenges were successful in developing methods for robot perception, path planning, and vehicle control. In other respects, the impact of these challenges has been less than was hoped: while robots are currently deployed in military operations, they are almost always *tele-operated*: virtually everything the robot does is dictated by a human using remote control. In hindsight, we see three problems that made it more difficult than expected to apply the technologies developed for these challenges to the development of practical autonomous robots:

- An over-reliance on prior data. In the DARPA Grand Challenges, detailed maps and aerial imagery were available to teams; aside from unexpected (but rare) obstacles, vehicle trajectories could largely be planned in advance. Such information is not always available, particularly inside buildings. GPS, a staple of grand challenge vehicles, can be deliberately jammed or unavailable due to buildings or natural topographical features.

- No humans in-the-loop. In real-world operations, a human commander will often have better insight into how to efficiently complete some components of the mission (incorporating, for example, information from non-digital sources like human scouts). For a robot system to be useful, it must be continually responsive to the commander and the evolving situation. This creates significant communication, human-robot interface, and human-understandable state representation challenges,

all of which were absent from the DARPA Grand Challenges.

- No explicit cooperation between agents. While vehicles in the urban challenge had to safely handle other moving cars, there was no explicit coordination between vehicles. Individually, robots will be quite limited for the foreseeable future; it is only when deployed in large numbers that they offer significant advantages over humans for reconnaissance or search and rescue missions. However, determining how to efficiently deploy a team of robots introduces difficult task-allocation, multi-vehicle state and map estimation, and communication challenges.

Figure 1: Our robot team at the beginning of a mission in Adelaide, Australia.

The Multi-Autonomous Ground Robot International Challenge (MAGIC) was sponsored by the US and Australian militaries to address these challenges. Following in the pattern of the grand challenges that came before it, MAGIC was open to academics and industry alike, offered several million dollars in funding and prizes, and culminated in a carefully designed and instrumented competition event. This event was held in Adelaide, Australia, in November 2010. Five finalist teams explored and mapped a 500m $\times$500 m indoor and outdoor environment looking for simulated improvised explosive devices and other "objects of interest" (OOIs). Performing such a mission completely autonomously is well-beyond the state of the art, so human commanders were allowed to provide strategic assistance to the robots and to remotely intervene in the case of other types of problems. No physical interventions were allowed, and teams were assessed penalties for all interactions with the robot system. In other words, the contest was structured in a way to determine the most autonomous team capable of completing the missions.

This paper describes our system (see Fig. 1), along with a number of technical contributions that were critical to the success of our system. We analyze the performance of our system and point out the failures, some

of which are either bugs or shortcomings due to the compressed development schedule. Others, however, highlight interesting and unsolved research problems. We will also describe some of the lessons we learned about building a large team of robots, such as the importance of making as much of the system state as "visible" as possible in order to aid debugging and situational awareness.

2 MAGIC 2010 Competition

Competitions provide a method for benchmarking progress in autonomous robotics by testing in real-world environments. For example, the series of DARPA grand challenges spurred innovation in automotive automation and machine perception (Montemerlo et al., 2008; Leonard et al., 2008). Competitions are also used as an effective learning tool to train the next generation of researchers, such as in the Intelligence Ground Vehicle Competition (IGVC), where student teams build autonomous vehicles to navigate a predetermined course (IGVC, 2011). Learning Applied to Ground Robots (LAGR) was another competitive DARPA project where multiple institutions used a standardized platform to benchmark different approaches to automatic terrain classification and navigation (Jackel et al., 2006; Konolige et al., 2009). In a similar vein, the on-going European Land Robot Trials (ELROB) seek to push the abilities of individual robots executing tele-operated reconnaissance missions and unmanned resupply missions (ELROB, 2006; Langerwisch et al., 2010; Himmelsbach et al., 2009). Meanwhile, the RoboCup Rescue League competition focuses on the search and rescue domain by holding events testing mobility in tough environments and individual robot autonomy (Andriluka et al., 2009; Balakirsky et al., 2007).

In contrast, the MAGIC competition was created to foster the development of multi-robot coordination and autonomy by focusing on reconnaissance in dangerous urban environments. Robots were tasked with exploring a large indoor-outdoor environment, building a map of it, and taking action to "neutralize" any detected threats, including simulated bombs and hostile humans. The contest focused on improving the autonomy of the robots, rather than their mobility or perception abilities. Teams were scored on a combination of map quality, number of OOIs neutralized, amount of human interaction and technical sophistication.

In the MAGIC competition, robots explored the showgrounds in three phases (see Fig. 2). The course was largely unknown to the teams, except for aerial imagery provided by the organizers and a brief walk-through of the first section. In short, the goal was for a team of robots, assisted by two human operators, to explore and map the area.

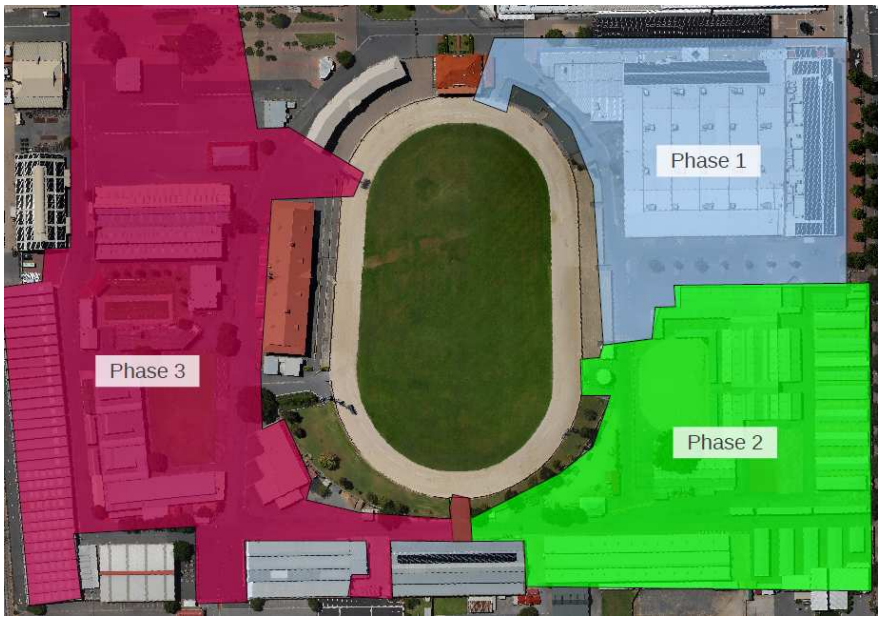


Figure 2: The 500 m × 500 m Adelaide Showgrounds were split into 3 phases for final round of the MAGIC competition. The site contained indoor and outdoor portions, an obstacle course, a sand pit, many curbs, chain-link fences, and other interesting terrain topologies.

The play field simulated a simplified battlefield with bombs, friendly and hostile humans, and other objects such as cars, doorways, and "toxic vats" (see Fig. 3). Pictures of these objects were provided in advance to help participants develop a strategy for detecting them. Ultimately, it was up to teams to decide how to divide this task between robots and the human operators.

In practice, remotely controlled Unmanned Aerial Vehicles (UAVs) are widely deployed in these types of search missions and are helpful in tracking moving objects (U.S. Army, 2010). Consequently, MAGIC organizers offered a data feed providing position estimates for humans. This feed was implemented using a ultra-wideband tracking system but was intentionally degraded to mimic the limited abilities of real-world aircraft: they cannot see through trees or under roofs, and they cannot identify whether targets are friendly or hostile. Humans could be classified upon visual inspection by the robots based on the color of their jumpsuits: red indicated a hostile person, blue indicated a non-combatant. This was a reasonable simplification designed to reduce the difficulty of the perception task.

Simulated bombs and hostile persons were dangerous to robots and civilians. If a robot came within 2.5 m of a bomb or hostile human, it would "detonate", destroying all nearby robots and civilians. Stationary bombs had a lethal range of 10 m, while hostile humans had a blast range of 5 m. In either case, the loss of any civilian resulted in a complete forfeit of points for that phase; it was thus important to monitor the position

of civilians to ensure their safety.

Two human operators were allowed to interact with the system from a remote Ground Control Station (GCS). Teams were allowed to determine when and how the humans would remotely interact with the robots. However, the total interaction time was measured by the contest organizers and used to penalize teams that required greater amounts of human assistance.

Teams were free to use any robot platform or sensor platform they desired subject to reasonable weight and dimension constraints. Teams were also free to deploy as many robots as they desired, subject to a minimum of three. In addition, teams designated some robots as "sensor" robots (that can collect data about the environment) and "disruptor" robots (which are responsible for neutralizing bombs and are not allowed to explore). This division of responsibilities increased the amount of explicit coordination required between robots: when a bomb was found by a sensor robot, it would have to coordinate the neutralization of that bomb with a second robot. Disruptors neutralized bombs by illuminating them with a laser pointer for 15 s. Hostile humans could also be neutralized, though a different type of robot coordination was required: two sensor robots had to maintain an unbroken visual lock on the target for 30 s.

In the tradition of the previous DARPA challenges, the competition was conducted on a compressed timeline: only 15 months elapsed between announcement and the challenge event, which was held in November 2010. Of the 23 groups that submitted proposals, only five teams survived the two rounds of down-selection to become finalists. This paper describes the approach of our team, Team Michigan, which won first place in the competition.

3 Team Michigan Overview

The MAGIC competition had a structure that favors large teams of robots since mapping and exploration are often easily parallelizable. A reasonable development strategy for such a competition is to start with a relatively small number of robots and, as time progressed and the capability of those robots improved, to increase the number of robots. The risk is that early design decisions could ultimately limit the scalability of the system.

Our approach was the opposite: we initially imagined a large team of robots and then removed robots as we better understood the limits of our budget, hardware, and algorithms. The risk is that trade-offs might be made in the name of scalability that ultimately prove unnecessary if a large team ultimately proves

(a) Non-combatant (blue) & mobile OOI (red)

(b) Static OOI through Doorway

(c) Toxic Vat

Figure 3: Objects of interest included (a) non-combatants and mobile OOIs, (b) static OOIs and doorways, (c) toxic vats, and cars (not shown).

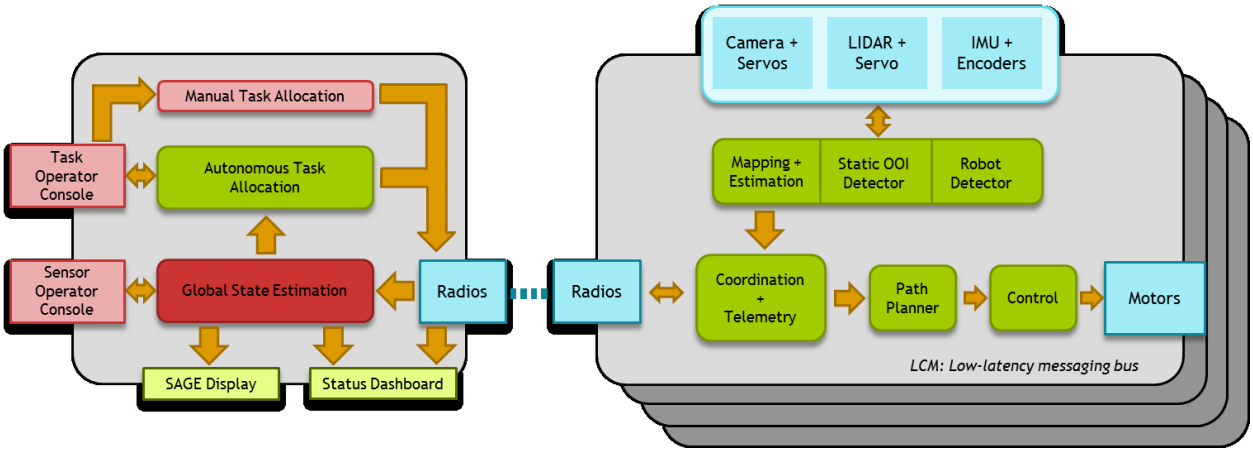


Figure 4: Software architecture. Our software system is composed of two main modules: the GCS (left) and the robots (right). Robots communicate with the GCS via radio for task assignment and telemetry. Individual software modules communicate via LCM (Huang et al., 2010).

unworkable. Early in our design process, for example, we committed to a low-cost robot design that would allow us to build 16-24 units. Our platform is less capable, in terms of handling in rough terrain and its sensor payload, than most other MAGIC teams. However, this approach also drove us to focus on efficient autonomy: neither our radio bandwidth nor the cognitive capacities of our human operators would tolerate robots that require vigilant monitoring or, even worse, frequent interaction.

Our team also wanted to develop a system that could be rapidly deployed in an unfamiliar environment. This was not a requirement for the MAGIC competition, but would be a valuable characteristic for a real-

world system. For example, we aimed to minimize our reliance on prior information (including maps and aerial imagery) and on Global Positioning System (GPS) data. During our testing, our system was routinely deployed in a few minutes from a large van by simply unloading the robots and turning on the ground control computers.

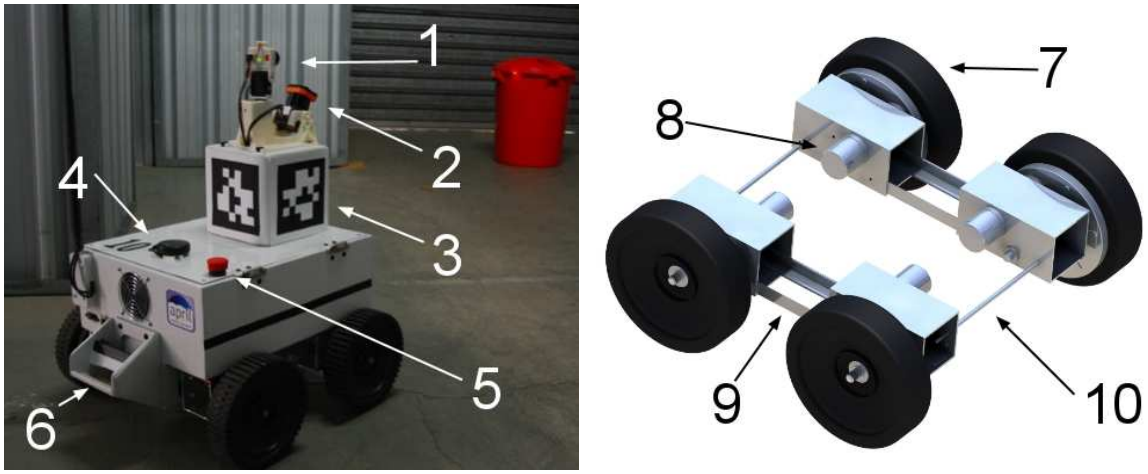
Our final system was composed of 14 robots and 2 human operators that interacted with the system via the GCS. We deployed twice as many robots as the next largest team, and our robots were the least expensive of any of the finalists at $11,500 per unit.

The architecture of our system is diagrammed in Fig. 4. At a high-level, it resembles that of other teams: each robot represents a semi-autonomous system interacting with a ground control station. However, we believe our solution has several interesting characteristics:

- Our GCS and robots use a Decoupled Centralized Architecture (DCA), in which the robots are oblivious to their role in the "grand plan", but act in a coordinated fashion due to the multi-agent plans computed on the GCS. Our approach takes this decoupling to an extreme degree: individual robots make no attempt to maintain a consistent coordinate frame.
- The GCS constructs a globally-consistent map using a combination of sensing modalities including a visual fiducial system and a new and very fast loop closing system.
- Robots are entrusted with their own safety, including not only safe path planning, but also autonomous detection and avoidance of hazardous objects.
- The GCS offers a human control of a variety of autonomous task-allocation systems via an efficient user interface.
- We considered the effects of cognitive loading on humans explicitly, which led to the development of a new event notification and visualization system.

This paper will describe these contributions, as well as provide an overview of our system as a whole. Throughout, we will describe where new methods were needed and where standard algorithms proved sufficient. Our system, while ultimately useful and the winner of the MAGIC competition, was not without shortcomings: we will describe these and describe our work to uncover the underlying causes.

Multi-robot systems on the scale of MAGIC are very new, and there are few well-established evaluation



(a) (b)

Figure 5: Robot neutralizing a static OOI with relevant components labeled (a): 1. Point Grey color camera, 2. Hokuyo LIDAR unit, 3. APRIL tag, 4. Garmin GPS, 5. Emergency stop button, and 6. 3-in-1 bumper, tip preventer, and handle. The simple suspension was key to our mechanical reliability and included (b): 7. Lawn mower wheels, 8. Motors, 9. Aluminum L brackets (hard attachment) and 10. Torsion bars (soft attachment).

metrics for such systems. We propose a number of metrics and the corresponding data for our system in the hope that it will provide a basis for future systems' evaluations.

3.1 Robot Hardware

In order to reduce the cost of our fleet of robots and to optimize our robots to the requirements of the MAGIC competition, we developed our own robot platform. For MAGIC, robots needed to be small enough to navigate through doorways and small corridors, yet large enough to mount 10 cm curbs and traverse urban terrain consisting of grass, gravel, and tarmac.

The main sensor payload of each robot consists of a Point Grey Firefly MV color camera and a planar Hokuyo UTM-30LX LIDAR range scanner. The LIDAR was actuated with an AX-12 servo, sweeping up or down every 1.25 s. Each 3D point cloud, acquired while the robot was in motion, consisted of approximately 50,000 points. The camera was independently actuated by two AX-12 servos, allowing the robot to acquire omni-directional imagery.

The robot contained additional sensors, including a custom-designed, low-cost 6-DOF Inertial Measurement Unit (IMU) and a GPS receiver. The IMU was critical for estimating the pitch and roll of the robot so that 3D point clouds were correctly registered. Additionally, the IMU was required for the robot odometry yaw

estimation. The GPS sensor, a Garmin 18x-5Hz, was ultimately not used by our system.

All the components of the robot were connected via USB or Ethernet to a dual-core 2.53Ghz Lenovo T410 ThinkPad laptop with 4GB of RAM and a 32GB SSD hard drive. We chose a laptop as our primary computation unit due to good price/performance and built-in back-up power from the laptop's battery. We ran a customized distribution of Ubuntu 10.04LTS and our software was primarily written in Java.

Our robot was constructed using rapid-manufacturing techniques, which allowed us to inexpensively but reliably duplicate our designs. Our design was rapidly iterated, with two major redesigns and fifteen iterations total. Our chassis was made from 9 mm birch plywood cut by a laser; wood proved a good choice due to its strength, low weight, and its ability to be easily modified with hand tools.

For more complex shapes, we used a Dimension 3D printer which "prints" parts by depositing layers of ABS plastic. These included the actuated sensor mount and various mounting brackets and protective cases. These printers produce parts that are strong enough to be used directly. Unfortunately, they are also fairly slow: each robot required about 20 hours of printing time for a full set of parts.

Each of our robot's four wheels was independently powered and controlled. The gears and wheel assembly used parts from a popular self-propelled lawn mower, in which power was transferred to the wheel using a gearway cut into the inner rim of the wheel. All of the power for the robot, including for the four DC brushed motors, was provided by a 24V 720Wh LiFePO4 battery; our robots had a worst-case run time of around four hours and a typical run time almost twice that. This longevity was an advantage versus some other teams, which had to swap batteries between phases.

4 Global State Estimation: Mapping and Localization

Simultaneous Localization and Mapping (SLAM) is a critical component of our system. For example, our multi-agent planning system relies on a global state estimate in order to know what parts of the environment have been seen and to efficiently coordinate exploration of unseen territory. A good navigation solution is also critical for human-robot interfaces. A number of challenges must be addressed:

- Consistency: When the same environment is observed at different times, the features of that environment must be registered correctly. In other words, loop closures must have a low false negative rate. As with most mapping systems, the false positive rate must be as close to zero as possible.

- Global accuracy: While large-scale deformations in a map are acceptable in some applications (Sibley et al., 2009), global accuracy is important when the system must coordinate with other systems that use global coordinates or when it is desirable to use aerial imagery. In the case of MAGIC, global accuracy was also explicitly an evaluation criterion.

- Rapid updates: The state estimate must be able to quickly incorporate information as it arrives from the robots.

- Minimal communication requirements: Our system had limited bandwidth, which had to support 14 robots not only mapping but also command/control, telemetry, and periodic image transmission.

- Minimize reliance on GPS: The robots must operate for long periods of time indoors, where GPS is not available. Even outdoors, GPS (especially with economical receivers) is often unreliable.

4.1 Coordinate Frames and the Decoupled Centralized Architecture

Our state estimation system is based on an approach that we call a Decoupled Centralized Architecture (DCA). In this approach, the global state estimate is computed centrally and is not shared with the individual robots. Instead, each robot operates in its own private coordinate frame that is decoupled from the global coordinate frame. Within each robot's private coordinate frame, the motion of the robot is estimated through laser- and inertially-aided odometry, but no large-scale loop closures are permitted. Because there are no loop closures, the historical trajectory of the robot is never revised and remains "smooth". As a result, fusion of sensor data is computationally inexpensive: observations can simply be accumulated into (for example) an occupancy grid.

In contrast, a global state estimate is computed at the GCS. This state includes a global map as well as the rigid-body transformations that relate each robot's private coordinate system to the global coordinate system. Loop closures are computed and result in a constantly changing posterior estimate of the map and, as a consequence, the relationship between the private and global coordinate frames. This global map is used on the GCS to support automatic task allocation and user interfaces, but is not transmitted to the robots, avoiding bandwidth and synchronization problems. However, critical information, such as the location of dangerous OOIs, can be projected into each robot's coordinate system and transmitted. This allows a robot to avoid a dangerous object based on a detection from another robot. (As a matter of implementation, we transmit the location of critical objects in global coordinates and the N global-to-private rigid-body transformations for each robot; this requires less total bandwidth than transmitting each critical object N

times.)

The basic idea of decoupling navigation and sensing coordinate frames is not new (Moore et al., 2009), but our DCA approach extends this to multiple agents. In this multi-agent context, it provides a number of benefits:

- It is easy to debug. If robots attempted to maintain their own complete state estimates, this estimate could easily deviate from the one visible to the human operator due to having different histories of communication exchanges with other robots. Such deviations could lead to difficult-to-diagnose behavior.
- It provides a simple sensor fusion scheme for individual robots, as described above.
- In systems that share submaps, great care must be exercised to ensure that information does not get incorporated multiple times into the state estimate; this would lead to over-confidence (Bahr, 2009; Cunningham et al., 2010). In our approach, only rigid-body constraints (not submaps) are shared, and duplicates are easily identified.
- Low communication requirements. Robots do not need to receive copies of the global map; they receive commands directly in their own coordinate frames.
- Little redundant computation. In some systems, robots share sub-graphs of the map and each robot redundantly computes the maximum likelihood map. In our system, this computation is done only at the GCS.

The principle disadvantage of this approach is that robots do not have access to the global map nor the navigational corrections computed by the GCS. The most significant effect is that each robot's private coordinate frame accumulates error over time. This eventually leads to inconsistent maps which can interfere with motion planning. To combat this, robots implement a limited sensor memory of about 15 seconds, after which observations are discarded. The length of this memory is a direct consequence of the rate at which the robot accumulates error; for our skid-steered vehicles, this error accumulates quickly.

With less information about the world around it, the on-robot motion planner can get caught in local minima (this is described more in Section 5.4). This limitation is partially mitigated by the centralized planner which provides robots with a series of waypoints that guide it around large obstacles. In this way, a robot can benefit from the global map without actually needing access to it.

4.2 Mapping Approach

We adopt a standard probabilistic formulation of mapping, in which we view mapping as inference on a factor graph (see Fig. 6). A factor graph consists of variable nodes and factor potentials. In our case, the variable nodes represent samples of the trajectory of each of the robots in our system. The factor potentials represent probabilities as a function of one or more variable nodes. For example, an odometry observation is represented by a factor potential connecting two temporally-adjacent samples from the trajectory of a robot. This factor potential has high probability (or low "cost") when the current values of the variable nodes agree with the odometry observation, and low probability (or high cost) when the current belief about the motion of the robot is inconsistent with the observation.

More formally, the factor potentials in our factor graph represent the probability of a given observation z_i in terms of the variable nodes x, which we write as $p(z_i|x)$. This quantity is the *measurement model*: given a particular configuration of the world, it predicts the distribution of the sensor. For example, a range sensor might return the distance between two variable nodes plus some Gaussian noise whose variance can be empirically characterized.

In practice, a single observation depends only on a small subset of the variable nodes x. When a factor graph is drawn as in Fig. 6, the dependence of factor potentials on variable nodes is represented as an edge in the graph. A typical factor graph arising from a mapping problem is very sparse: the sensor sub-systems described later in this section each result in factor potentials relating only two nodes.

A factor graph is essentially a statement of an optimization problem: it collects information about the world in the form of the observations/factor potentials (the z_i's, or collectively z). The central challenge is to infer what these observations say about the variable nodes (x). In other words, we wish to determine the posterior distribution $p(x|z)$. By applying Bayes' rule, we can rewrite our desired posterior in terms of the sensor model. (We additionally assume that we have no a priori belief about x, i.e., that $p(x)$ is uninformative, and that z_i are conditionally independent given x)

$$p(x|z) \propto p(z|x) = \prod_i p(z_i|x) \tag{1}$$

We further assume that each potential is normally distributed, so $p(z_i|x) \propto \exp\{-(x-\mu)'\Sigma^{-1}(x-\mu)\}$. The standard approach is to find x which maximizes the probability of Eqn 1 (or equivalently maximizes the negative log probability). Taking the log converts $\prod p(z_i|x)$ into a sum of quadratic terms. Computing the

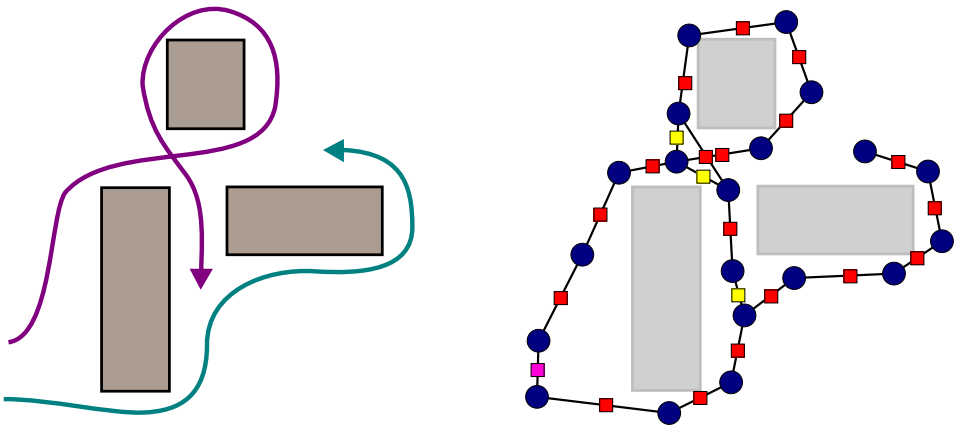


Figure 6: Factor graph resulting from a mock exploration mission. Robots follow two trajectories (one dotted, one dashed) to explore an environment (left). The resulting map (right), is represented as a factor graph in which robot poses (unknown variables, blue circles) are related through known potential functions (squares) which are determined from odometry (red), scan matching (yellow) or fiducial observation (white).

maximum (by taking the derivative) can be iteratively computed by solving a series of linear system of the form $A\Delta x = b$. Typically only 1-2 iterations are required if a good initialization is known. Crucially, the low node degree of the potentials leads to a sparse matrix A. This sparsity is the key to efficient inference (Olson et al., 2006; Grisetti et al., 2007; Kaess et al., 2007).

In an online system such as ours, the factor graph is constantly changing. The motion of robots introduces new variable nodes and additional sensor readings produce new factor potentials. In other words, we will actually consider a sequence of graphs G_j that tend to grow over time. For example, robots report their position about once every 1.25 seconds, creating a new variable node which represents the (unknown) true position of the robot at that time. Sensor data collected from that position is compared to previously acquired sensor data; if a match is found, a new factor potential is added to encode the geometric constraint that results from such a match. (This matching process will be described in more detail below.)

The factor graph approach has distinct advantages over other methods. The memory required by Kalman filter approaches grows quadratically due to the fact that it explicitly maintains a covariance matrix. Particle filters (Montemerlo, 2003) exhibit even faster growth in memory due to the growth in dimensionality of the state vector; this growth necessitates resampling methods (Grisetti et al., 2005) which introduce approximation errors that can cause failures in complex environments. Maximum-likelihood solvers, like ours, do not attempt to explicitly recover the posterior covariance. Additionally, factor graphs can be retroactively edited to recover from incorrect loop closures.

In the following sections, we first describe how sensor data was processed to be incorporated into this graph,

including our methods for filtering false positives. We then describe our optimization (inference) strategy in more detail, including several improvements to earlier methods.

4.3 Sensing Approach

Our vehicles combine sensor data from three sources: inertially-aided odometry, visual fiducials, and quasi-3D laser scan matching. As we will describe below, our system can also make use of GPS data, but generally did not do so. In each case, the processing of the sensor data has the direct goal of producing a rigid body motion estimate from one robot pose to another. In the case of odometry, these rigid body transformations describe the relationship between the i^{th} and $i-1^{th}$ poses of a particular robot. For scan matching, these constraints are created irrespective of whether the poses are from a single robot or two different ones. The observation of a visual fiducial necessarily concerns two separate robots. While scan matching constraints (edges) were computed based on robot laser data sent to the centralized ground station, information about inertially-guided odometry and visual fiducial observations were computed locally on each robot and transmitted over the radio for inclusion in the graph.

Each of the sensing modalities is integrated into the factor graph in a similar way: by introducing an edge which encodes a geometric constraint between two nodes (representing locations). The differences in the quality of data provided by each modality is taken into account during inference by considering the covariance associated with those edges.

4.3.1 Inertially-aided odometry

Our robots have four powered wheels in a skid-steer configuration. The two rear wheels have magnetic Hall-effect encoders with a resolution of about 0.7 mm per tick. These provide serviceable estimates of forward translation, but due to the slippage that results from our drive train when turning, yaw rate estimates are very poor.

To combat this, we developed the PIMU, a "pico"-sized inertial measurement unit (see Fig. 7). The PIMU includes four gyroscope axes (with two axes devoted to yaw for a slightly reduced noise floor), a three axis accelerometer, a magnetometer, and a barometric altimeter. In our mixed indoor/outdoor settings, we found the magnetometer to be too unreliable to integrate into our filter and did not use it. The barometric altimeter was included for evaluation purposes and was also unused.

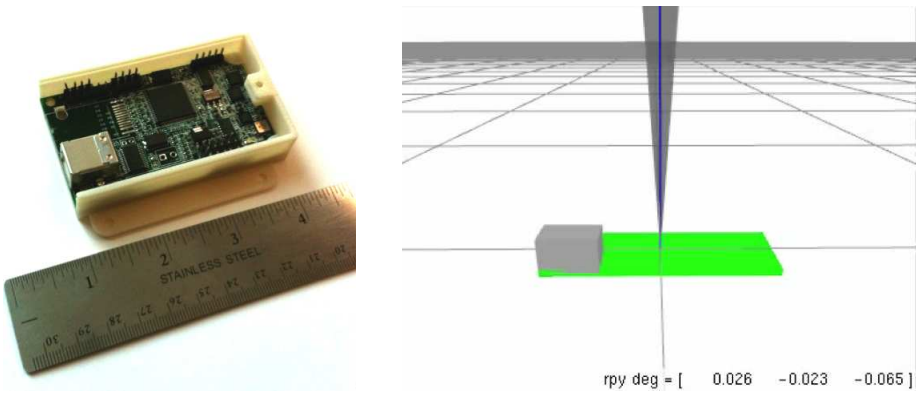


Figure 7: Inertial Measurement Unit. Left: our open-source PIMU unit combines gyroscopes and accelerometers into a compact USB-powered device. Right: The orientation of the PIMU can be constrained by integrating the observed acceleration and comparing it to the gravity vector; this constrains the gravity vector to lie within a cone centered about the observed gravity vector.

The PIMU streams data via USB to the host computer where filtering is performed. The filtering could have been performed on the PIMU (which has an ARM Cortex-M3 microprocessor), but performing this filtering on the main CPU made it easier to modify the algorithm and to reprocess logged sensor data as our filtering algorithms improved.

Because the PIMU uses low-cost MEMS-based accelerometers and gyroscopes, online calibration and estimation of bias parameters becomes important. This is particularly true of the gyroscopes, whose zero-rate output undergoes a large-magnitude random walk. The long operating time of our robots (e.g., 3.5 hours for competition) necessitates an ongoing recalibration, as a single calibration at start-up would yield unacceptable errors by the end of the mission. Similarly, our large number of robots necessitated a simple system requiring no manual intervention. We also wished to avoid an explicit zero-velocity update system in which the robot would tell the PIMU that it was stationary; not only would it complicate our system architecture (adding a communication path from our on-robot motion planner to the IMU), but such an approach is also error prone: a robot might be moving even if it is not being commanded to do so. For example, the robot could be sliding, teetering on three of its four wheels, or could be bumped by a human or other robot.

We implemented automatic zero-velocity updates, in which the gyroscopes would detect periods of relative stillness and automatically recalibrate the zero-rate offset. In addition, roll and yaw were stabilized through observations of the gravity vector through the accelerometer. The resulting inertial-aided odometry system was easy to use, requiring no explicit zero-velocity updates or error-prone calibration procedures, and the performance quite good considering the simplicity of the online calibration methods (and comparable to commercial MEMS-grade IMUs). The PIMU circuit board and firmware are now open-source projects and

can be economically manufactured in small quantities for around $250 per unit.

4.3.2 Visual Fiducials

Our robots are fitted with visual fiducials based on the AprilTag visual fiducial system (Olson, 2011) (see Fig. 8). These two-dimensional bar codes encode the identity of each robot and allow other robots to visually detect the presence and relative position of another robot. These detections are transmitted to the GCS and are used to close loops by adding more edges to the graph. These fiducials provide three significant benefits: 1) they provide accurate detection and localization, 2) they are robust to lighting and orientation changes, and 3) they include enough error checking that there is essentially no possibility of a false positive.

Fiducials provide a mechanism for closing loops even when the environment is otherwise devoid of landmarks, and was particularly useful in initializing the system when the robots were tightly clustered together. These observations only occurred sporadically during normal operation of the system, since the robots usually stayed spread out to maximize their effectiveness. Each robot has four copies of the same fiducial, with one copy on each face of the robot's "head", allowing robots to be detected from any direction. Each fiducial measures 13.7 cm (excluding the white border); the relatively large size allowed robots to detect other robots up to a range of around 5 m.

4.3.3 Laser Scan Matching

The most important type of loop closure information was based on matching laser scans from two different robot poses. Our approach is based on a correlative scan matching system (Olson, 2009a), in which the motion between two robot poses is computed by searching for a rigid-body transformation that maximizes the correlation between two laser scans. These two poses can come from the same or different robots.

Our MAGIC system made several modifications and improvements over our earlier work. First, our MAGIC robots acquire 3D point clouds rather than the 2D "cross sections" used by our previous approach. While methods for directly aligning 3D point clouds have previously been described (Hähnel and Burgard, 2002), these point clouds would need to be transmitted by radio in order to make robot-to-robot matches possible. The bandwidth required to support this made the approach impractical.

Instead, our approach first collapsed the 3D point clouds into 2D binary occupancy grids, where occupied cells correspond to areas in which the point cloud contains an approximately vertical surface. These occupancy grids were highly repeatable and differed from the terrain classification required for safe path planning (as

described in Section 5.3). Additionally, they can be efficiently compressed since these maps required radio transmission to the GCS (see Section 4.3.4).

Using this approach, our matching task was to align two binary images by finding a translation and rotation. However, correlation-based scan matching typically is formulated in terms of one image and one set of points. We obtained a set of points from one of the images by sampling from the set pixels. Our previous scan matching methods could achieve approximately 50 matches per second on a 2.4 GHz Intel processor. However, this was not sufficiently fast for MAGIC: with fourteen robots operating simultaneously, the number of potential loop closures is extremely high. Where our earlier work used two-level multi-resolution approach to achieve a computational speed-up, we generalized this approach to use additional levels of resolution. With the ability to attempt matches at lower resolutions, the scan matcher is able to rule out portions of the search space much faster. As a result, matching performance was improved to about 500 matches per second with 8 resolution levels. This matching performance was critical to maintaining a consistent map between all fourteen robots.

(a) Robot (b) Sensorhead

Figure 8: One of 24 robots built for the MAGIC 2010 competition, featuring AprilTag visual fiducials, in (a) and a close-up of the sensor package in (b), consisting of an actuated 2D LIDAR and a pan/tilt color camera.

4.3.4 Lossless Terrain Map Compression

In order for the GCS to close loops based on laser data, terrain maps must first be transmitted to the GCS. With each of fourteen robots producing a new scan at a rate of almost one per second, bandwidth quickly becomes an issue. Each terrain map consisted of a binary-valued occupancy grid with a cell size of 10 cm; the average size of these images was about 150×150 pixels.

Initially, we used the ZLib library (the same compression algorithm used by the PNG graphics format), which compressed the maps to an average of 378 bytes. This represents a compression ratio of just under 8x, which reflects the fact that the terrain maps are very sparse. However, our long-distance radios had a

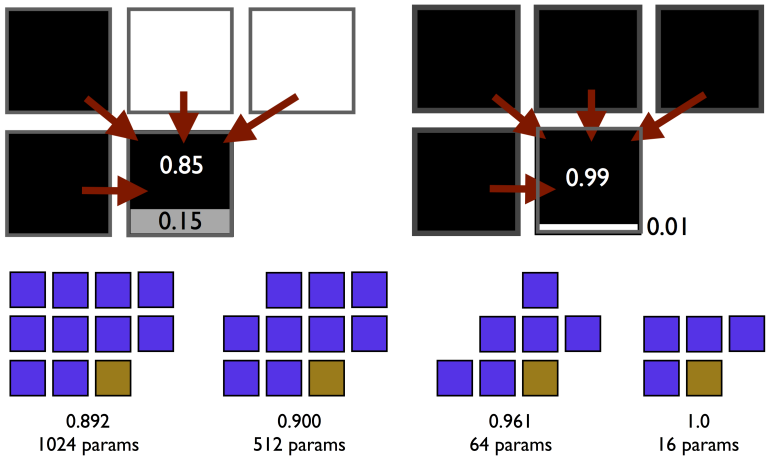


Figure 9: Graphical Model Based Compression. (Top) When compressing binary-valued terrain maps, the value of every pixel is predicted given four other pixels. For example, given two black pixels to the left and two pixels above (top left), our model predicts an 85% likelihood that the fifth pixel will be black. If all the surrounding pixels are black (top right), our model predicts a 99% likelihood that the fifth pixel will be black. We experimented with four different models that use different sets of nearby pixels to make predictions (second row); these patterns were designed to capture spatial correlations while making use only of pixels that would already have been decoded. More complex models required fewer bits to encode a typical terrain map: for example, the 10-neighbor model used only 0.892 times as many bits as the 4-neighbor baseline model. However, this relatively modest improvement comes at a dramatic cost in model complexity: the 10-neighbor model has $2^10 = 1024$ parameters that must be learned, in contrast to the baseline model's $2^4 = 16$ parameters. Consequently, our system used the 4-neighbor model.

theoretical shared bandwidth of just 115.2 kbps with even lower real-world performance. The transmission of terrain maps from fourteen robots would consume virtually all of the usable bandwidth our system could provide.

Recognizing that ZLib is not optimized for this type of data, we experimented with a variety of alternative compression schemes. Run-Length Encoding (RLE) performed worse than ZLib, but when RLE and ZLib were combined, the result was modestly better than ZLib alone. Our intuition is that ZLib must see a fair amount of data in order to build an effective compression dictionary, during which time the compression rate is relatively poor. By pre-compressing with RLE, the underlying structure in the terrain map is more concisely encoded, which allows a ZLib's dictionary to provide larger savings more quickly.

Ultimately, we employed a predictive compression method based on a graphical model that predicts the value of a pixel given its previously-encoded neighbors. This model was combined with an arithmetic encoder (Press et al., 1992). This approach yielded a compression ratio of just over 21x, reducing the average message size to 132 bytes (see Fig. 10).

The graphical model was a simple Bayes net (see Fig. 9), in which the probability of every pixel is assumed

to be conditionally dependent with its four neighbors above and to the left of it. Restricting neighbors to those above and to the left allows us to treat pixels as a causal stream—no pixels "depend" on pixels that have not yet been decoded.

Once the structure of a model has been specified, we can train the model quite easily: we use a large corpus of terrain maps and empirically measure the conditional probability of a pixel in terms of its neighbors. Because we use a binary-valued terrain map, the size of the conditional probability tables that we learn is quite small: a four-neighbor model results has just 16 parameters.

We experimented with a variety of model structures in which the value of a pixel was predicted using 4, 6, 9, and 10 earlier pixels (see Fig. 9). The number of parameters grows exponentially with the number of neighbors, but the compression improved only modestly beyond four neighbors. Consequently, our approach used a four neighbor model.

For compression purposes, the graphical model is known in advance to both the encoder and decoder. The encoder considers each pixel in the terrain map and uses the model to determine the probability that the pixel is 0 or 1 based on its neighbors (whose values have been previously transmitted). In Fig. 9 we see two examples of this prediction process; in the case of four "0" neighbors, our model predicts a 99% probability that the next pixel will also be zero. The encoder then passes the probabilities generated by the model and the actual pixel value to an arithmetic encoder.

Arithmetic encoders produce a space-efficient bitstream by encoding a sequence of probabilistic outcomes (Press et al., 1992). They achieve efficiencies very close to the Shannon limit. In other words, pixels whose value was predicted by the model require fewer bits than those that are "surprising" given the probabilistic model.

On the ground control station, the whole process is reversed: an arithmetic decoder converts a bitstream into a sequence of outcomes (pixel values). As each new pixel is decoded, the values of its previously-transmitted neighbors affect the conditional probabilities of the model.

Our graphical-model based compression performed much better than ZLib, RLE, or our RLE+ZLib hybrid, achieving an average of 132 bytes per terrain map (see Fig. 10). This is only 44% as much data required by the next best method, RLE+ZLib and 35% as much as ZLib alone. In the presence of austere communications limits, this savings was critical in supporting the simultaneous operation of 14 robots.

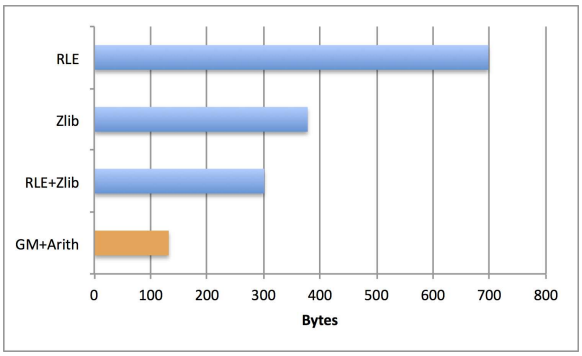


Figure 10: Terrain Map Compression Results. Our graphical-model plus arithmetic encoding approach significantly outperforms run-length encoding, ZLib, and a combination of the two. Uncompressed terrain maps averaged about 22,000 binary pixels.

4.4 GPS

Each robot was equipped with a consumer-grade Garmin GPS-18x-5 Global Positioning System receiver, which provided position updates at 5 Hz. Under favorable conditions (outdoors with large amounts of visible sky), these devices produce position estimates with only a few meters of error. In more difficult settings, such as outdoors next to a large building, we would occasionally encounter positioning errors on the order of 70 m. Upon entering a building, the receiver would produce increasingly poor position fixes until it eventually (correctly) reported that a fix had been lost.

Unfortunately, these receivers would often output wildly inaccurate confidence metrics: in particular, they would occasionally report good fixes when the error was very large. Without a reasonable way of estimating the accuracy of GPS fixes, it is difficult to incorporate them into a state estimate. Like most mapping systems, incorporating a high-confidence but inaccurate measurement wreaks havoc on the quality of the posterior map. Transition periods, where the robots entered or exited a building, were particularly problematic. As a result, we generally found that our system worked better with GPS disabled, and as such we didn't make use of it during the competition.

4.5 Loop Validation

If incorrect edges are added to the graph, such as if one location is mistaken for another similar-looking environment, the posterior map can rapidly become badly distorted. This sensitivity to false positives is not unique to our approach—most mapping methods are relatively fragile with respect to data association errors.

In our system, laser scan matching produces the greatest number of false positives. Consequently, laser scan matches are subjected to additional verification before they are added to the graph. The basic idea of loop validation is to require that multiple edges "agree" with each other (Bosse, 2004; Olson, 2008; Olson, 2009b). Specifically, consider a topological "loop" of edges: an edge between node A and B, another edge between B and C, and a third edge between C and A. If the edges are correct, then the composition of their rigid-body transformations should approximately be the identity matrix. Of course, it is possible that two edges might have errors that "cancel", but this seldom occurs.

Specifically, our method puts all newly generated (and unverified) edges into a temporary graph. We perform a depth-limited search on this graph looking for edge cycles; when a cycle is found, its cumulative rigid-body transform is compared to the identity matrix. If it is sufficiently close, the edges in the cycle are copied to the actual graph. Old hypotheses for which no cycle has been found are periodically removed from the temporary graph to reduce the computational cost of this depth-limited search and to reduce the probability that erroneous loops will be found.

4.6 Inference in the SLAM Graph

The preceding sections have described the construction of the factor graph that describes the map of the environment. This factor graph implicitly encodes a posterior distribution (the map that we want), but recovering this posterior requires additional computation. Our approach is focused around maximum likelihood inference; the actual *distribution* is not explicitly estimated. When marginals are required, we rely on standard fast approximations based on Dijkstra projections (Bosse, 2004; Olson, 2008). From an optimization perspective, our inference method most closely resembles SqrtSAM (Dellaert and Kaess, 2006), with two note-worthy modifications. First, instead of using COLAMD (Davis et al., 2004) for determining a variable order, we use the Exact Mininum Degree (EMD) ordering. Second, we added a method for improving the quality of initial solutions (as described below) which helps avoid local minima.

Exact Minimum Degree (EMD) Ordering. SqrtSAM-style methods build a linear system $Ax = b$ whose solution x is repeatedly computed to iteratively improve the state estimate. The matrix A is then factored (e.g., using a Cholesky or QR decomposition) in order to solve for x. When the matrix A and its factors are sparse, we can significantly reduce computational costs. Critically, however, the sparsity of the factors of A are dramatically affected by the order of the variables in A. Consequently, finding a "good" variable ordering (one which leads to sparse factors of A) is a key component of SqrtSAM-style methods. Of course, it takes time to compute (and apply) a variable ordering; in order for a variable ordering to be

worthwhile, it must save a corresponding amount of time during factorization.

We consider two variable orderings: COLAMD (developed by Davis (Davis et al., 2004) and recommended by Dellaert (Dellaert and Kaess, 2006)), and a classic Exact Minimum Degree (EMD) ordering. EMD generates a variable ordering by repeatedly selecting the variable with the minimum node degree; COLAMD is (roughly speaking) an approximation of EMD that can be computed more quickly.

In early versions of our system, we used COLAMD to reorder our A matrix. COLAMD works quite well in comparsion to no variable reordering; on phase2, computation drops from 6 seconds to about 0.1 second. This is interesting since COLAMD actually expects to be given the Jacobian matrix J, whereas our A matrix is actually $J^T J$.

However, we subsequently discovered that EMD produced even better variable orderings. Computing an EMD ordering is much slower than computing a COLAMD ordering, but we found that the savings during the Cholesky decomposition typically offset this cost. The performance of EMD and COLAMD are compared in Table 1 and Fig. 11), measuring the runtime and quality of the orderings on all three of the MAGIC phases and a standard benchmark dataset (CSW Grid World 3500). In each case, COLAMD computes an ordering significantly faster than EMD. However, the quality of the ordering (as measured by the amount of fill-in in the Cholesky factors) is always better with EMD; this saves time when solving the system. The critical question is whether the total time required to compute the ordering and solve the system is reduced by using EMD. In three of the four cases, EMD results in a net savings in time. These results highlight some of the subtleties involved in selecting a variable ordering method; a more systematic evaluation (potentially including other variable ordering methods) is an interesting area for future work.

Initialization. It is well-known that least-squares methods are prone to divergence if provided with poor initial estimates (Olson, 2008). This problem can be significantly mitigated with a good initial estimate. A popular method is to construct a minimum spanning tree over the graph, setting the initial value of each node according to the path from the root of the tree to the node's corresponding leaf.

Initialization is especially important when incorporating global position estimates incorporating orientation information, since linearization error can inhibit convergence even with relatively small errors; 30 or more degrees can be problematic, 70 degrees is catastrophic. Likewise, a challenge with spanning tree approaches is that two physically nearby nodes can have very different paths in the spanning tree, leading to significant errors.

Phase	Operation	EMD	COLAMD
Phase 1 (avg. degree 5.7)	Ordering Time	0.094 s	0.011 s
	L nz	517404 (0.383%)	881805 (0.652%)
	Solve Time	0.597 s	1.298 s
	Total Time	**0.756 s**	1.398 s
Phase 2 (avg.degree 2.96)	Ordering Time	0.089 s	0.0069
	L nz	190613 (0.116%)	263460 (0.1609%)
	Solve Time	0.118 s	0.165 s
	Total Time	0.249 s	**0.208 s**
Phase 3 (avg. degree 2.68)	Ordering Time	0.0025 s	0.0009 s
	L nz	16263 (0.317%)	20034 (0.391%)
	Solve Time	0.006 s	0.0146 s
	Total Time	**0.014 s**	0.0281 s
CSW Grid World 3500 (avg. degree 3.2)	Ordering Time	0.069 s	0.006 s
	L nz	183506 (0.166%)	299244 (0.271%)
	Solve Time	0.096 s	0.214 s
	Total Time	**0.195 s**	0.251 s

Table 1: Effects of variable ordering. While it takes longer to compute the MinDegree ordering, it produces sparser L factors (i.e. has smaller non-zero (nz) counts), with large impacts on the time needed to solve the system. Note that total time includes additional costs for constructing the A matrix and thus is slightly larger than the sum of ordering and solve time. Best values are **bolded**.

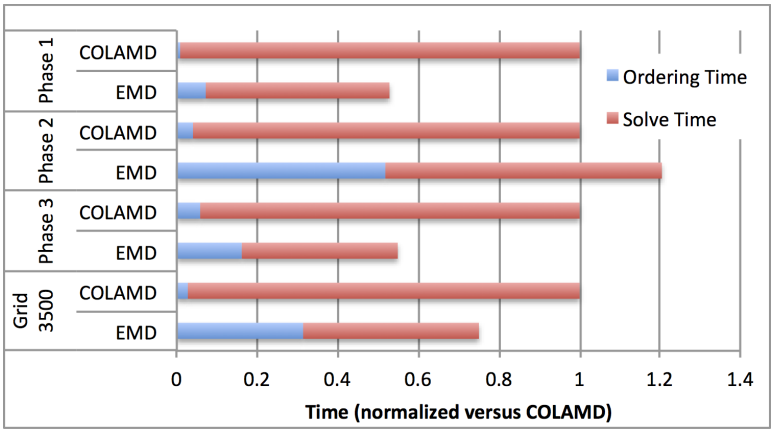


Figure 11: EMD vs. COLAMD runtimes. In each case, it is faster to compute the COLAMD ordering than the EMD ordering. However, in all but Phase 3 (which had a particularly low node degree), EMD's better ordering results in lower total computational time.

Our solution was to construct a minimum-uncertainty spanning tree from each node with global orientation information. In our system, global alignment could be specified manually at the start of a mission to allow the global state to be super imposed on satellite imagery (usually only one or two such constraints were needed). Each spanning tree represents a position prediction for every pose relative to some globally-referenced point. We then initialized each pose with a weighted average of its predicted positions from the spanning trees. Since multiple spanning trees were used, discontinuities between nearby nodes were minimized. This initialization process was very fast: it ran in a small fraction of the time required to actually solve the graph.

4.7 Graph Simplification

While the memory usage of our graph grows roughly linearly with time, the optimization time grows faster since the average node degree increases as more loop closures are identified. To prevent this, we can consider periodically applying a 'simplify' operation which would result in a SLAM graph with lower average node degree and would therefore be faster to optimize. The basic challenge in simplifying a graph is to find a new graph with fewer edges whose posterior, $p(x|z_i)$, is similar to the original graph. Both the maximum likelihood value of the new graph and the covariance should be as similar as possible.

In the case where multiple edges connect the same two nodes, the edges can be losslessly merged into a single edge (assuming Gaussian uncertainties and neglecting linearization effects). In the more general case, lossless simplifications of a graph are not usually possible.

Our basic approach is to remove edges that do not provide significantly new information to the graph. Consider an edge e connecting two vertices a and b. If the relative uncertainty between a and b is much lower when e is in the graph than without it, then e should remain in the graph. Our approach is to construct a new graph containing all of the nodes in the original graph, and to add new edges that reduce the uncertainty between the nodes they connect by at least a factor of α (see Alg. 1). In practice, we use a Dijkstra projection (Bosse, 2004; Olson, 2008) to compute upper-bounds on the uncertainty rather than the exact uncertainty.

This procedure creates a new graph with a posterior uncertainty that is similar to the input graph, but its maximum likelihood solution will be different. Our solution is to force all of the edges in the new graph to have a zero residual when evaluated at the maximum likelihood solution of the original graph. The biggest shortcoming of this method is that the effects of erroneous edges in the original graph are permanently "baked in" to the simplified graph, making it more difficult for the operator to correct loop closing errors.

Algorithm 1 Graph-Simplify(G, α)

```
{ Create a new graph G' with no edges }
G' = < nodes(G), null >
{ Iterate overall edges }
for all e ∈ edges(G) do
   Σ = dijkstraProjection(G', node1(e), node2(e))
   if trace(Σ) > α trace(covariance(e)) then
      e' = makeEdge(optimize(G), node1(e), node2(e))
      G' =< nodes(G'), edges(G') ∪ e' >
   end if
end for
return G'
```

Thus, our operators only employed the graph-simplify command when the graph was in a known-good state. As we detail in our evaluation, this operation is typically used at most 2-3 times per hour of operation.

4.8 Global Map Rasterization

The global state estimate from the SLAM optimization is used as the backbone of all autonomous and human-interface systems on the GCS. However, a disadvantage of graph-based mapping methods is that they do not directly produce an occupancy-grid representation of the environment. Such a representation is often the most convenient representation for visualization and autonomous planning. Thus, in a graph SLAM framework, an explicit rasterization step is needed which generates such an occupancy-grid from the underlying graph. There are two major challenges in such a system: first, an approach must address how multiple observations of the same terrain should be fused together; in particular, it is generally desirable to remove "trails" associated with moving objects. Second, the approach must be fast in order to support real-time planning.

The obvious approach to rasterization is to iterate over the nodes in the graph and to project the laser scan data into an occupancy grid. This naive approach can remove transient objects through a voting scheme, and is adequately fast on small maps (or when used off-line). There is little discussion of map rasterization in the literature, perhaps due to the adequacy of this simple approach in these cases. However, when applied to graphs with thousands of nodes (our graph from the first phase of MAGIC had 3876 nodes and the second had 4265 nodes), the rasterization time takes significantly longer than the optimization itself and introduced unacceptable delays into the planning process.

We devised two techniques to speed up map rasterization, which we have described more fully in a separate publication (Strom and Olson, 2011). The first technique relies on identifying redundant sensor observations

Method	Avg. Time (s)	Max. Time (s)
Naive	5.62	6.94
w/ Cover	2.75	3.37
w/ Cache & Cover	**1.24**	**3.6**

Table 2: Average & worst-case runtimes for rasterizing the 230 increasingly large posterior SLAM graphs from the phase 2 dataset that are larger than 200m x 150m. The naive method entails a simple linear rasterization pass of all the sensor data, using none of the optimizations described here. Suggested method is **bolded**.

to exclude from the final map. The second involves determining which computations used to compute the previous map could be directly reused to compute the next map, in the case where the SLAM posterior for a portion of the graph stayed fixed. On environments on the scale of a MAGIC phase (220 x 160 m), these improvements commonly result in a reduction from 5.5 seconds to less than 1.5 seconds per rasterization (see Table 2).

Consider the j-th posterior SLAM graph $G_j = \langle V_j, E_j \rangle$. Each node $x \in V_j$ has some associated sensor data which combines to form the j-th map M_j. For low-noise sensors such as laser range finders, if multiple graph nodes are observing the same area, they provide redundant coverage of a portion of M_j. We identify redundant nodes in V_j and remove them to form a covering C_j that still computes the same map M_j, but much faster. While finding a minimum covering is difficult, we can use a conservative heuristic to compute a covering based on sensor origin and field of view. While not minimal, we can, in practice, be assured that the resulting map will be structurally equivalent to the map computed from all the nodes. This technique provided a factor of 2-3 speedup for the rasterization. Additionally, we found that using a covering actually can improve the quality of the map by reducing feature blurring.

In the absence of loop closures that affect the posteriors of all the graph nodes, some parts of successive maps M_i and M_{i+1} will be very similar. To exploit this, we can build an intermediate map representation by partitioning V_i into k subsets $S_1 \cdots S_k$. A submap m_j can be formed from S_j, and then M_i can be rasterized by combining all the submaps $m_1 \cdots m_k$. If the posterior of all nodes in S_j do not change after integrating some loop closures, we can reuse m_j when computing the next map M_j. This technique allows us to trade a doubling in worst case time for an average reduction in runtime by additional factor of 2. Together, our improvements to online map rasterization were critical in enabling the high level autonomous planners to act often and with lower latency.

5 Levels of Autonomy in a Large Robot Team

The autonomy of our system is achieved through the composition of several layers. Coordination of the whole team is performed by a centralized planner which considers high-level goals such as coordinated exploration of an area. Additional autonomy on individual robots handles automatic object detection, terrain classification, and path planning. Using the DCA approach described in Section 4, our on-robot autonomy is able to reduce the bandwidth requirements of our system. Our high- and low-level autonomy also contribute to reducing the cognitive load on the operators since less interaction is required.

5.1 Task Allocation

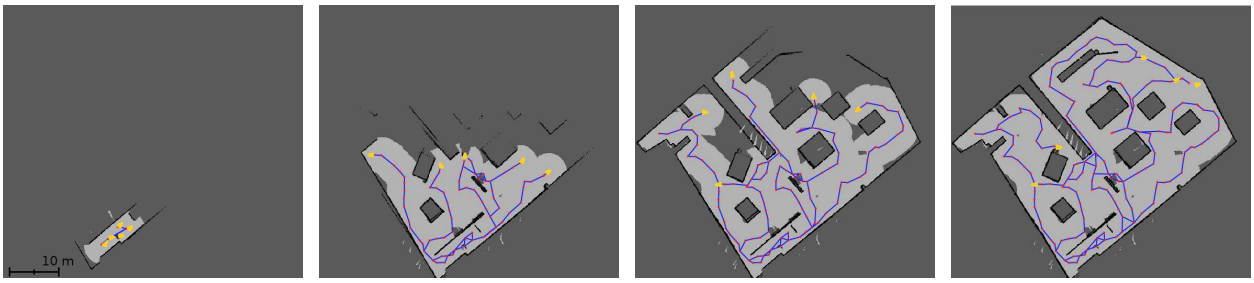
Robots are assigned tasks, such as "drive to x,y" or "neutralize OOI", from a centralized GCS. During normal operation of our system, robots are assigned tasks autonomously from one of two planners: one specializing in exploration using sensor robots, and the other managing OOI disruptions. However, humans supervising the process can override these tasks.

5.1.1 Exploration Task Allocation

The exploration planner autonomously tasks sensor robots to explore the unknown environment, expanding the global map. This process indirectly searches for OOIs as each robot's on-board autonomy is always searching for OOIs. We use a variant of existing frontier-based exploration techniques for high-level tasking of the sensor robots (Yamauchi, 1998; Burgard et al., 2000). The *frontier* consists of the reachable regions of the environment on the boundary between the known and unknown regions of the world. The planner greedily assigns each robot a goal on the frontier based on both the expected amount of information to be gained by traveling to a location and the cost to acquire that information. The expected information gain is calculated by casting out rays from a candidate goal to estimate a robot's sensor coverage of new terrain at the goal location. The acquisition cost is estimated as the distance a robot must travel to reach the goal. As a result of our cost formulation, robots tend to go first to nearby, high information gain regions before moving on to explore new areas farther away.

The planning search can be quite computationally expensive as the size of the global map increases, so at first we only search up to a set distance away, e.g. 20 m, for potential goals. This is usually sufficient to keep the robots exploring. However, when a robot explores all the frontier in its vicinity it must search further for reachable frontier. Thus, the planner constructs a *trajectory graph* (as seen in Fig. 12) to mitigate the

large-scale search cost. The trajectory graph is similar to a map of a highway system. It records a set of connected waypoints between which sensor robots are known to have previously traveled. The planner uses the trajectory graph to cheaply compute a coarse path between the robot and any long-distance goals.



(a) (b) (c) (d)

Figure 12: Robots building a trajectory graph. As the robots (yellow triangles) explore the environment (time increases from (a) to (d)), the trajectory graph (red nodes and blue traversable edges) grows into a coarse highway system. The autonomous planners use the trajectory graph to efficiently route robots long distances through explored terrain. This example was generated in a simulated 50×40 m region.

5.1.2 Disruptor Task Allocation

As sensor robots explore the environment they continuously search for static OOIs. Once found, the system must quickly and efficiently neutralize these simulated bombs as they may be blocking passageways and pose hazards to robot and friendly humans if inadvertently triggered.

Like the exploration planner, the disruptor planner sends high-level commands like "go to x,y" to the disruptor robots. Neutralization of an object is accomplished by shining a laser pointer on the target in question. The neutralization process is fully automated, though the human operator's approval is required before beginning neutralization in order to comply with the MAGIC competition rules.

The disruptor planner fulfills two planning goals: 1) immediately tasking an available disruptor robot to the neutralization region of each newly found static OOI and 2) positioning idle disruptor robots evenly throughout the environment to minimize response time. The planner utilizes the same trajectory graph used by the exploration planner when choosing paths. Unlike sensor robots, disruptor robots *only* travel on the 'highway'; complying with a MAGIC rule that disruptor robots (unlike sensor robots) are not allowed to explore new areas.

5.1.3 Human-in-the-Loop Task Allocation

Robot tasks are assigned by the autonomous planners under normal conditions. However, when tricky conditions arise, human knowledge is useful for explicitly generating tasks which will result in loop closures or for coordinating the neutralization of mobile OOIs. In these cases, the human is able to issue a command specifying a driving destination. Human operators also help ensure the safety of robots and non-combatants during an OOI neutralization.

In Section 6, we will discuss in more detail the user interfaces the human operators used to assign tasks or override autonomous task allocation. Regardless of whether a human or machine was in charge of task allocation, all tasks were executed autonomously by the robots.

5.2 In-Situ Object Detection

The contest contained OOIs throughout the environment in unknown locations. Using humans for detection by scanning 14 video streams would have resulted in an excessively high cognitive load and a strain on our communications infrastructure. Additionally, a robot driving near a static OOI has only a limited time to identify it and potentially change course to avoid detonation. Since the communication latency with the GCS and the high-level planning rate are not short enough to guarantee safe robot behavior, autonomous detection and avoidance of these hazards on the robot is *crucial*. As sensor robots explore the environment, they continuously and autonomously analyze their sensor data for the presence of static OOIs and toxic vats (refer back to Section 2 and Fig. 3 for more details on OOI specifics). Objects flagged by both the shape and color detectors are passed on to the operators as putative toxic vats and static OOIs.

5.2.1 Shape and Color Detectors

The shape detection procedure starts by segmenting the points from one 3D sweep of the actuated LIDAR sensor. The aim of the segmentation step is to identify individual objects in the scene. Segmentation is achieved by first removing the ground surface from the scene; then, individual points are grouped into cells on a grid; finally, cells with similar heights are grouped together as a single object. This height-based segmentation strategy helps to separate objects from walls even if they are very nearby.

After segmentation, we try to identify approximately cylindrical objects, since both the static OOIs and toxic vats were approximately cylindrical. First, the length and width of each object are calculated from the

maximally aligned bounding rectangle in the X-Y dimensions. Also, cylindrical objects presented a semi-circular cross section to the LIDAR scanner and a robust estimate of the radius of each object was derived from a RANSAC-based circle fit (Fischler and Bolles, 1981). The circumference of the observed portion of this circle was also calculated. Appropriate thresholds on the values of these four features (length, breadth, radius, circumference) were then learned from a training dataset. Since we did not have access to exact samples of all the objects that we needed to detect, the thresholds were then manually enlarged.

Once candidate objects are detected using shape information, we then apply color criteria. To robustly detect color, we used thresholds in YCrCb color space. This color space allows us to treat chromaticity information (CrCb) separate from luminance information (Y), thus providing a basic level of lighting invariance. Thresholds on color and luminance of objects in YCrCb space were then learned from a training set of images that were obtained under various lighting conditions.

In order to test the color of an object that was detected with the LIDAR, we must have a camera image that contains the object. A panoramic camera (or array of cameras) would make this relatively straight-forward, since color data is acquired in all directions. Our approach was to use a single actuated camera with a limited field of view instead; this approach allows us to obtain high resolution images of candidate objects while minimizing cost.

A naive camera acquisition strategy would be to pan towards objects once detected by the LIDAR. This operation could take a second or more, leading to unacceptable object classification delays. False positives from the LIDAR system exacerbated the problem, since each false positive generated a "pan and capture" operation for the camera.

We solve this problem by continuously panning our camera and storing the last set of images in memory. When the LIDAR detects a putative object, we retrieve the appropriate image from memory. As a result, we are able to test the color of objects without any additional actuation delay. This system is illustrated in Fig. 13.

5.3 Terrain Classification

Safe autonomous traversal through the environment requires the detection of obstacles and other navigational hazards. Our approach uses the 3D LIDAR data to analyze the environment in the robot's vicinity, producing a 2D occupancy grid, discretized xy-pixels, describing the drivability of each cell (see Fig. 14). This map

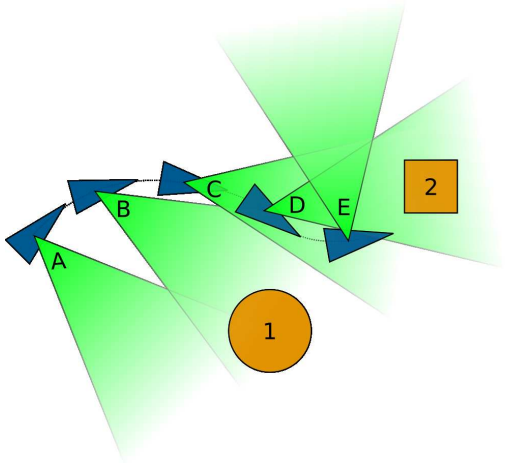
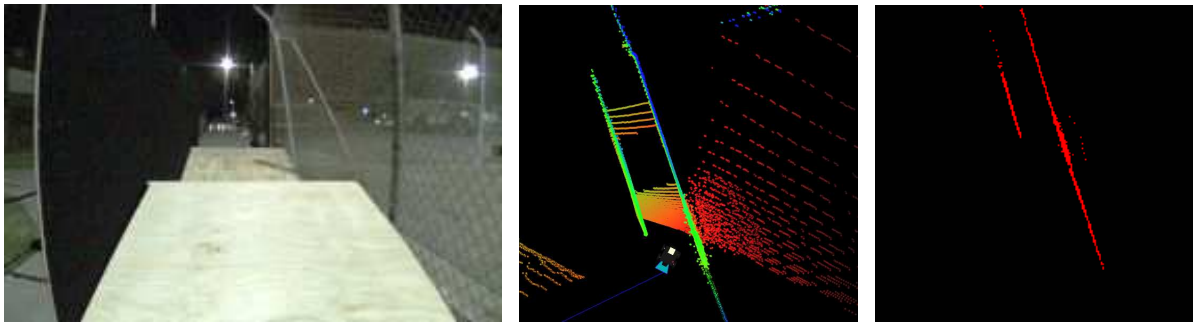


Figure 13: Five images (green) are acquired from a moving robot (blue). For object 1, the camera sharing module returns image B, as the object projects fully into image B and only partially into image A. In contrast, object 2 projects fully into both images C and D, so the newer image, D, is returned.

is generated in three stages by 1) constructing a ground model and using it to filter the 3D point cloud, 2) classifying the terrain with the filtered points, producing a single-scan drivability map, and 3) accumulating these drivability maps over time.

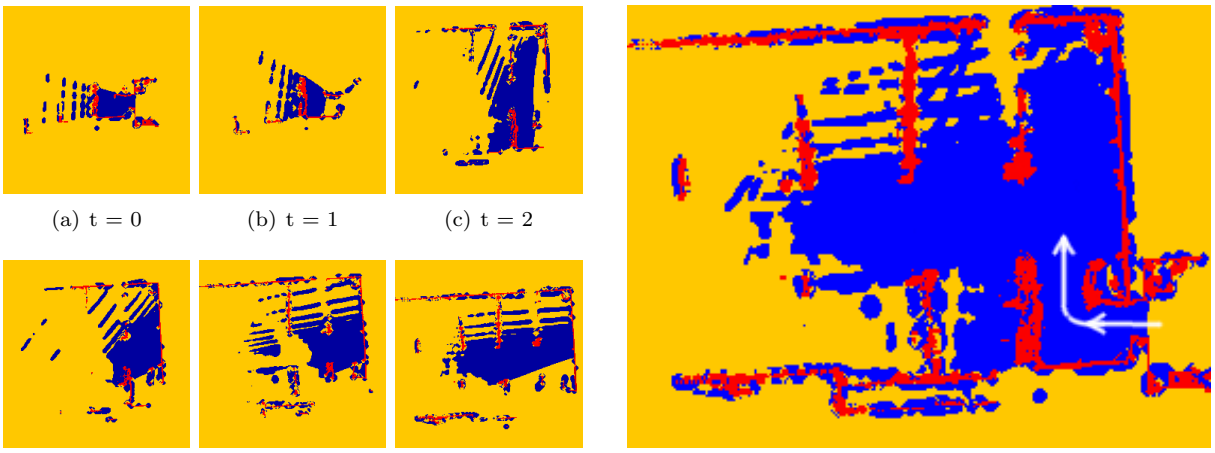
In the first stage, we construct a ground model in order to reject 3D LIDAR points that pose no threat to the robot, such as tree limbs that would pass safely overhead. To build the ground model, we note that the minimum z value observed within each cell (subject to some simple outlier rejection) is a serviceable *upper bound* of the ground height. This model assumes that we cannot see through ground, which would fail, for example, if the robot were driving on glass. These upper bounds are propagated between adjacent cells in order to compute better ground height estimates, even in regions with no observations. We use a ground model prior, corresponding to 20 degree upward slopping terrain, to signify the maximum expected slope on natural ground terrain; this method is similar to an algorithm used in the DARPA Urban Challenge (Leonard et al., 2008).

In the second step, we process the filtered points from the previous step in order to produce a drivability map for a single 3D scan. Our approach can best be described as an up detector because it detects vertical discontinuities between neighboring cells and marks only the cell with the higher z value as an obstacle (rather than both cells). This detector is based on the constant slope model, which represents drivable terrain as a linear function (Matthies et al., 1996; Schafer et al., 2008). For example, an obstacle exists between two cells, A and B, l units away, if the change in height h between them is greater than some



(a) Camera View (not used for terrain analysis) (b) 3D Point Cloud (c) Obstacle Map

Figure 14: The scene in (a) corresponds to the overhead view of the 3D LIDAR return shown in (b) (colored by height) along with a possible terrain classification shown in (c) (red for obstacle and black for drivable).



(a) t = 0 (b) t = 1 (c) t = 2

(d) t = 3 (e) t = 4 (f) t = 5

(g) Historical Composite Map

Figure 15: Each of the terrain classification maps (a)-(f) were generated by a single 3D sweep. These sequential maps are merged into a larger composite map (g), which is sent to the path planner. (red indicates an obstacle, blue for drivable terrain, and yellow for unknown)

predefined linear function, specifically,

$$h = |z_A - z_B| > al + b \tag{2}$$

The constant parameters a for slope and b for vertical offset summarize the physical capabilities of the robot. The constant slope model would mark A, B, and each intermediate unobserved cell in between the two as obstacles. However, our algorithm marks as an obstacle only the single cell, A or B, with the maximum z value, thus minimizing the dilation of obstacles out into drivable terrain, allowing operation in constrained environments.

In order to compute the h and l parameters of Eqn 2, which cannot be measured from a single cell, we again pass messages between cells. The message passing algorithm requires each cell to keep track of two values,

z_{min} and z_{max}. A message from one cell to an adjacent cell includes two values: the originating cell's (z_{min} - a) and ($z_{max} + a$). The value of l is never explicitly calculated because it is encoded in the incremental addition/subtraction of a. Upon receiving a message, the cell simply keeps the min of each received value and the respective z_{min} or z_{max}. A cell is marked as an obstacle if its z_{min} is more than b greater than one of its 4-connected neighbors' z_{max}. We found that two iterations produced maps sufficient for the terrain present in MAGIC. Additional work has been done to handle negative obstacles, which were not present (greater than 10 cm) in the competition (Morton and Olson, 2011).

In the final step, we combine drivability maps produced from individual 3D scans into a larger map (see Fig. 15). These maps were combined pessimistically: evidence of an obstacle from any individual scan results in an obstacle in the output map. This allowed handling slow moving dynamic obstacles without explicitly tracking and predicting their motion. The individual maps are generally acquired from different robot positions, and so the compositing operation takes into account the motion of the robot as measured by the robot's navigation system. Errors in the robot's position estimates result in artifacts— an obstacle appearing in two different places, for example— which can impede the performance of the path planner. To combat this, we aggregated only the most recent 12 obstacle maps, representing a memory of about 15 s. If an alignment error occurs that prevents the robot from finding a motion plan, the robot will remain stationary until the erroneous map is forgotten. Naturally, there is no appreciable alignment error while the robot remains still. The terrain classification process has an in-system runtime of approximately 300 ms for maps on the scale of 400 m^2 and 5 cm discretization.

5.4 On-Robot Path Planning

Motion planning goals are transmitted to the robot by the GCS from either an autonomous planner or the human operator, as described in Section 5.1. These goals provide no information about the path that the robot should take — the robot must perform its own path planning.

While path planning has been well studied, practical implementations must consider a number of complications. These include cluttered environments (indoor and outdoor), incomplete and noisy obstacle maps that change with every new sensor sweep, imperfect localization, the possibility that no safe path exists, and limited computational resources. The characteristics of the computed path also have an impact on the quality of future maps and the safety of the robot. A smooth path, for example, is less likely to create alignment artifacts, while the addition of path *clearance* decreases the likelihood of accidental collision due to localization errors (Wein et al., 2008; Geraerts and Overmars, 2007; Bhattacharya and Gavrilova, 2007).

Our decision to limit the sensor memory of the terrain map to the last 15 seconds, coupled with the finite range of that map, creates a particularly challenging complication. In general, we would like to compute paths with added clearance to avoid accidental collisions with the environment. One common method to add clearance to a path is to define a minimum-cost formulation with decaying cost values around obstacles, usually known as a *potential field*. However, finite terrain map history causes potential field methods to fail. This is because the distance-clearance tradeoff implied by the potential field is difficult to balance when the terrain classification of some cells is unknown. By changing the cost parameters associated with narrow passageways and the cost associated with traveling through unverified terrain, it is possible to get a minimum-cost planner to work in specific scenarios. However, finding parameters that behave reasonably over a wide range of scenarios proved very difficult. Our solution was to plan in two stages: first, we search for minimum *distance* paths, then we optimize those paths to minimize cost.

This minimum distance path is computed via Dijkstra's shortest path algorithm on the discrete, regular grid (Dijkstra, 1959). We plan on a 2D $\{X, Y\}$ configuration space generated by a circular kernel that encompasses the robot footprint, allowing us to plan without considering rotation. Various stages of computing the shortest path are shown in Fig. 16.

The minimum distance path does not account for localization errors, as discussed, nor is the resulting path generally smooth since it corresponds to motions along an 8-connected graph. We solve these issues by 1) iteratively relaxing the path to add a bounded amount of clearance and 2) iteratively smoothing the transitions between nodes on the path to reduce high frequency jitter. The result is a smooth path that maximizes clearance in small environments (e.g., doorways) but is not constrained to follow the maximal clearance path in large environments (e.g., hallways). The algorithm is illustrated in Fig. 16(d) and is further detailed in a separate publication (Richardson and Olson, 2011).

(a) Obstacle map (b) Configuration space (c) Minimum distance path (d) Smoothed result

Figure 16: Each stage of on-robot path planning. The configuration space (b) is computed by 2D max disc convolution with the obstacle map (a). The path resulting from Dijkstra's algorithm (orange) with the distance to the goal for each node examined (gray) are shown in (c). Nodes with distances greater than the solution (yellow) are not examined. Finally, the distance transform (gray) and smoothed path (blue) are shown in (d)

Figure 17: Team Michigan's user interfaces (from left): SOC, Status Dashboard, TOC, and SAGE. These are shown in greater detail in Figs. 19 and 20.

6 Human Supervision of a Large Robot Team

Our choice to deploy a relatively large number of robots had important implications for the interfaces we provided the human operators. With 14 robots, it is cognitively infeasible for two humans to micromanage the actions and state estimation for all agents simultaneously. While our autonomous systems are designed to operate without human intervention under normal operation, we expected to encounter situations which caused them to fail. Consequently, we designed a suite of user interfaces which enable the human operators to maintain situational awareness and intervene in a variety of ways when such situations occur (see Fig. 17). We designed four interfaces, two of which were interactive (matching the two operators we were allowed by the contest rules):

- Sensor Operator Console (SOC) - Allows human intervention in perception tasks such as OOI identification and data association. Controlled by sensor operator.

- Task Operator Console (TOC) - Allows team and individual robot tasking, interface to autonomous planners, and provides birds-eye-view of the environment. Controlled by task operator.

- Status Dashboard - Read-only display of each robot's status with indicator lights for various states, battery meters, and communication throughput that was especially useful for highlighting error states on individual robots.

- Situation, Actions, Goals, Environment (SAGE) Display - Read-only display with automatic analysis of the global state to provide a real-time narration of the reconnaissance mission.

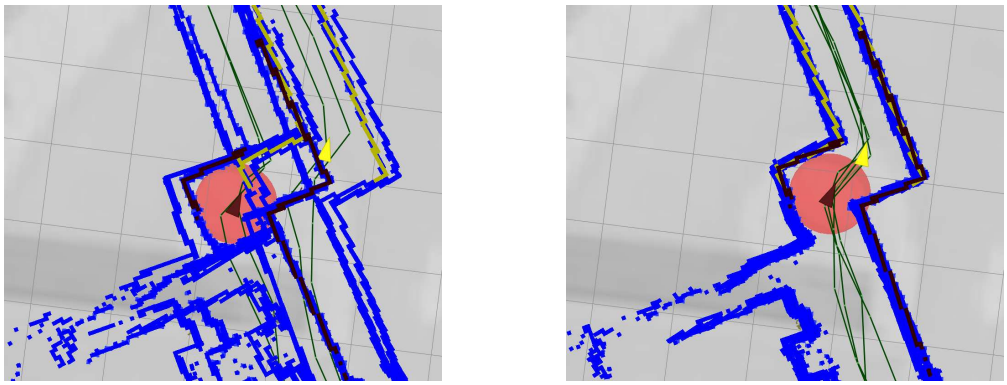
6.1 Sensor Operator Console

Online machine perception remains one of the major challenges in autonomous robotics. Automatic object detection and classification or loop-closure detection (data association) has been the topic of considerable research. While we implemented automatic systems for both of these (see Sections 4, 5.2), the state-of-the-art still cannot guarantee error-free performance for 45 hours of combined robot up-time as demanded by the scope of the competition. Therefore, we dedicated one of our two interactive user interfaces to allowing a human to monitor the perception tasks completed by the system and to intervene when errors occur.

All autonomous detections of OOIs are automatically relayed to the GCS and displayed on the SOC as a thumbnail on the map. This allows the sensor operator to see all the object detections currently under consideration by the system with a single glance and, if necessary, modify the classification. For example, a false detection of a static OOI can be flagged as such by the sensor operator, causing the autonomous system to abort the deployment of a disruptor robot already en route for neutralization.

Perceptual errors can also have catastrophic errors on the global state estimate. Our automatic loop-closure system described in Section 4 alleviates many of the small odometry errors typically experienced by the robots. However, our robots occasionally make significant odometry errors due to continuous wheels slippage, for example, due to an elevated ridge or drainage ditch. These errors tend not to be correctable by our automatic loop closer, resulting in erroneous posteriors and degrading map quality. Severe odometry errors can also contribute to erroneous loop closures which will cause the SLAM estimate to diverge. Bad loop closures also occur due to perceptual aliasing, a common occurrence due to the similar structure of many man-made environments. Thus, human monitoring of the SLAM graph can be used to detect these catastrophic errors and fix them. Using a SLAM graph as a back-end for our global state estimate makes it straightforward for a human to understand when an error occurs, simply by visualizing its topology and the evolving posteriors of all the robot trajectories. The sensor operator can view the rasterized global map to look for artifacts of bad loop-closures, or leverage experience to detect when two robots' trajectories are misaligned (see Fig. 18).

We explored several ways in which a human can interact with the SLAM graph. We found that high-level control over the automatic loop closer was the most effective way for the human to interact. When the state estimate diverges, the human can revert automatically added loop-closures through a "rewind"-type command. Subsequently, the human can suggest that the system find a loop closure through two specific robot poses. The system then produces a match confidence score, which the human can use to determine if



(a) Loop closures lacking (b) With loop closures

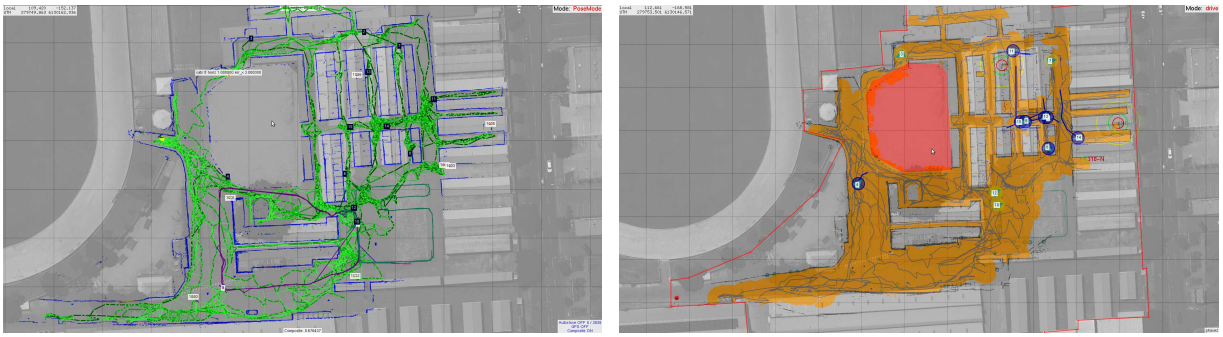
Figure 18: Our maps can typically maintain alignment by running the automatic loop closer, leading to improvement from (a), no loop closures, to (b), with loop closures. When the loop closer fails (not shown), misalignment, such as in (a), can provide a cue to a human operator that intervention may be necessary.

the constraint will alleviate the original problem faced by the loop closer. When errors occur, the typical sequence of commands employed by the sensor operator are: 1) rewind the constraints added by the loop closer, 2) suggest an additional constraint be added between two robot poses in the problem area and 3) restart the loop closer. Step 2 is particularly crucial, since it provides a different prior for the loop closer so it is less likely to make the same mistake again.

We found these primitives an effective way to fix the global state estimate, while still reducing the cognitive load on the sensor operator to a manageable level. Instead of micromanaging the state estimate of the system, the human can use their superior perception ability to detect errors on a higher level. The loop closures are still produced via the scan matcher (in fact, our system provides no mechanism for a human to manually specify an alignment between two poses); instead, the human can suggest poses for the automatic system to align, and thereby incorporate loop closures that the system otherwise finds to be too uncertain.

6.2 Task Operator Console

The reconnaissance task posed in the MAGIC contest introduces a variety of objectives that the robots must achieve, including exploration, avoiding moving OOIs, OOI identification and OOI neutralization. Together, achieving these tasks presents a dynamic, partially observable planning problem which can be difficult to handle autonomously. Furthermore, important aspects of the target area are not known in advance, making it difficult to design autonomous systems which can handle all eventualities. By allowing our second human operator to observe the task allocation process, we are able to correct mistakes by utilizing human intuition and inference.



(a) Sensor Operator Console (SOC)

(b) Task Operator Console (TOC)

Figure 19: The two interactive displays used by our human operators. The SOC allowed a human to modify override labeled OOIs and to oversee the progress of the SLAM Graph (a). The TOC allowed a human to manage the autonomy of the robots (b). (Best viewed in color)

Our system architecture allows robots to be commanded at a variety of levels of detail: fully-autonomous, where all task assignments are computed without any human intervention; semi-autonomous, such as "drive to x,y" where paths are computed autonomously, but the destination is specified by the human; and tele-operation, where the robots' motion can be micromanaged via remote-control of the drive train. The goal of our system was to spend as much time as possible in higher autonomy modes so as to "stretch" the capacity of a single human to control a larger team of robots. In addition, autonomy allowed the system to continue to make progress even when the operators were fully absorbed in a task that required their concentration (such as OOI neutralization).

A straight-forward way of building an interface for N robots is to replicate the interface for a single robot. Such an approach has the advantage of simplicity, but makes it hard for the human operator to understand the relationships between different robots. Instead, our TOC presents a bird's eye view of the environment, showing all the robots and their global position estimates on the most recent rasterized map. This interface is inspired by similar interfaces found in many successful Real Time Strategy (RTS) computer games which require simultaneous management of hundreds of agents. The operator may select multiple robots via the mouse, tasking them simultaneously. Compressed camera data is streamed only on demand for the selected robots, which helps reduce communication bandwidth.

A key tension in the user interface design was determining what to display. In particular, when should buttons and/or sliders be used to allow the parameters of the system to be adjusted? In a system as complex as ours, dedicating even a small amount of interface real estate to each option would hopelessly crowd the screen, leaving little space for "real" work. Our approach was to use a pop-up console that allowed the operators to quickly adjust the system parameters (such as telemetry data rates, detailed robot status requests, configuring

communication relays, etc.) Without a shell-like command completion facility, remembering the available options would have proved impractical. But we found that with command completion, team members were quickly able to express complex commands (i.e., commands that affect an arbitrary subset of the robots) that would have been tedious or impossible to do with on-screen buttons or sliders.

This pop-up console was designed to be verbose and self-documenting, with the natural consequence that commands were not terribly concise. We added an additional macro system, allowing an operator to bind function keys to particular commonly-used commands. For example, all robots could be assigned to their respective autonomous planner and unpaused with a single key stroke.

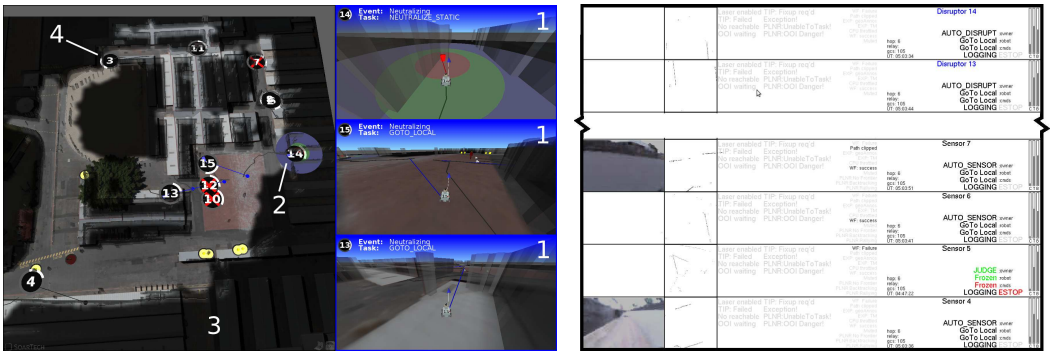
We believe that these sorts of user-interface issues are critically important to achieving good performance. We did not conduct any formal user studies to verify the efficacy of these methods, but our experience leads us to believe that our approach (avoiding on-screen controls in order to maximize the amount of screen space that can be devoted to the interface's primary function, coupled with a pop-up console with command completion) is highly efficient.

6.3 Status Dashboard

The interfaces described above were designed to streamline the process of interacting with a system that involves many agents. To further reduce cognitive load, we designed two additional displays which served to augment the operators' situational awareness in different ways.

When faults do occur, operators need a method for quickly determining the status of malfunctioning robots. We designed a second "dashboard" display to show the status for all robots, and highlight any anomalous state. For example, if a robot is not responding to waypoint commands, a warning indicator would light (much like an "engine trouble" indicator in a car). Further, diagnostic information would be displayed that would help the operator determine, for example, that the robot is suffering from an extended communications outage. The operator can then use the TOC to convert one of the robots into a communications relay.

We also used text-to-speech announcements to indicate important error states. This system monitored message rates, battery voltages, memory usage, CPU temperature, etc., and would audibly alert the operators to any anomalies. This was an enormous time saver; operators would often be alerted to an anomalous condition before it was obvious that anything was wrong by observing the robots themselves.



(a) SAGE Display (b) Status Dashboard

Figure 20: The SAGE interface (a) provided a cognitively lightweight system overview by 1) automatically narrating important events and showing the entire global state, such as 2) OOIs, 3) unexplored space and 4) robot positions. Alternatively, the human operators could quickly ascertain each robot's status via its respective row on the status dashboard (b).

6.4 SAGE Display

The Situations, Actions, Goals and Environment (SAGE) display (see Fig. 20(a)) was designed to provide mission-level situation awareness without overwhelming the user with distracting details. It borrowed from ideas frequently used in 3D games. Its default view is an isometric, top-down "bird's eye" view that provided the user with the overall situation awareness including robot positions and mission risks. SAGE also detects key events and displays zoomed in views of the event locality automatically, allowing a human operator to maintain global and local situational awareness without requiring any manual interaction. SAGE can be thought of as a live "highlights" display, narrating the operation of the system.

6.4.1 Global Situational Awareness

SAGE's global top-down view uses several design strategies to communicate key information to the user at a glance. It automatically computes its view position from the positions of active robots and shifts view positions smoothly without distracting the user or requiring intervention.

The most important information presented in the bird's eye view are the robot markers. These markers are view position invariant so they are always clearly visible. They contain information about the robot's id, orientation, state, and task. The top down view also displays information about objects that the robots detect. Most objects are displayed in small sizes and with muted colors, but dangerous objects are highlighted with richly saturated colors and rings showing the trigger, blast, and neutralization distances. Finally the overhead view provides information about information quality. Uncertainty about walls and moving objects

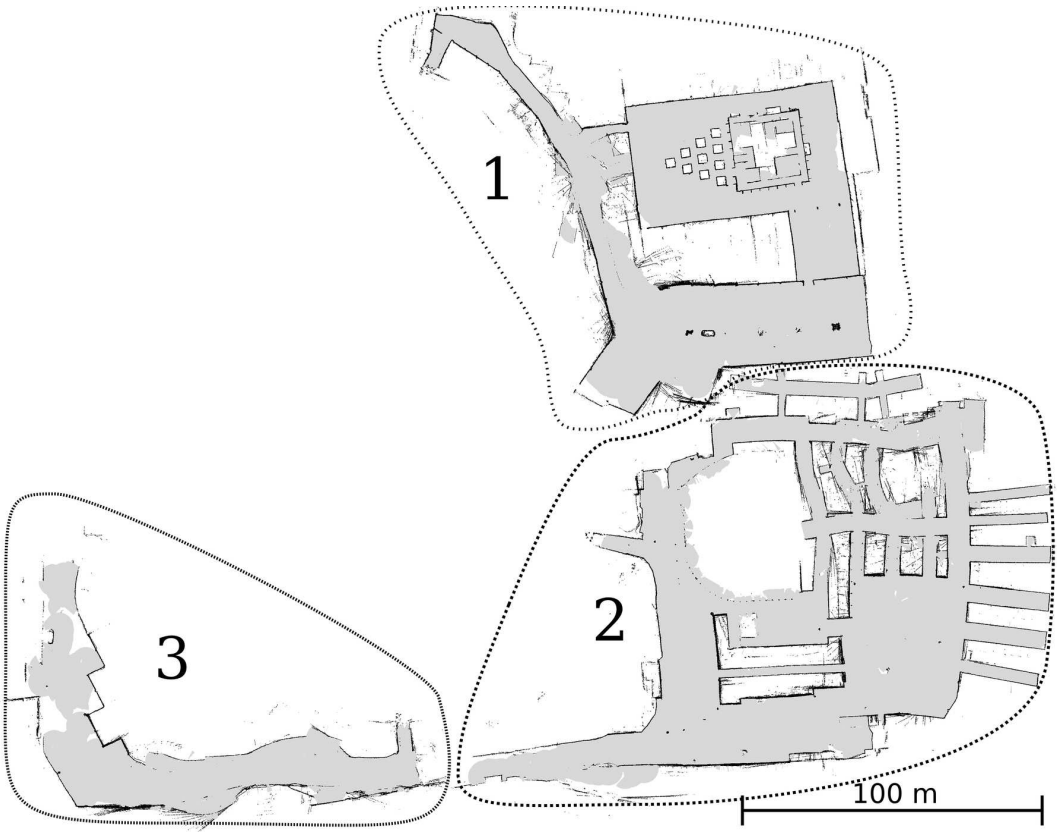
is represented as dynamic shifts in transparency, with higher levels of transparency assigned to higher uncertainties. This is especially valuable for mobile OOIs, which fade over time so as not to mislead the user into thinking they are still at a previously sensed location. Unexplored regions are represented using a "fog of war" shading similar to that used in real-time strategy games. Thus the user can see the global state of the robot team and important mission data at a glance.

6.4.2 Event Detection and Highlight Display

SAGE also draws user attention to important events. SAGE is able to automatically detect several types of events of interest to the user using the robots' status reports and simple rules. For example, SAGE recognizes when new objects are detected, when robots change to a neutralization task, and when civilians are close to dangerous OOIs. Upon recognition, SAGE automatically creates a new view that highlights this event and smoothly animates it to a position at the right of the display. These highlight windows are color coded and decorated with key information allowing the user to quickly see critical information such as the event's danger level (e.g. red color for very dangerous events), type, and associated robot. The view itself is selected to give the user appropriate context – an isometric view for new objects and a third person view behind the robots for neutralization.

To prevent events from overwhelming the user, the system prioritizes events and shows only the top four events at any time. It also merges similar events and removes events over time based on the type and severity of the event. The result is that a user has the detailed context needed without any need to manipulate interface elements, thus saving time and allowing the user to focus on the high level tasks.

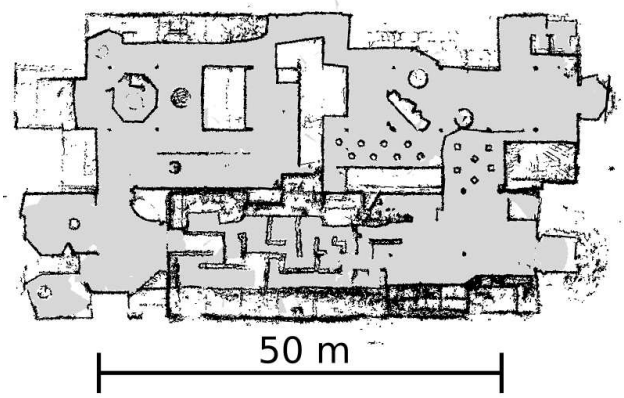
Highlighting important events enables the human operators to maintain good situational awareness throughout the challenge, and contributed to enabling only 2 human operators to control 14 robots. The operators leverage these displays to prioritize their intervention in the system. Although human intervention was heavily penalized during the competition, we were able to leverage the intuition and domain knowledge of the humans at the appropriate times to ensure our global state estimate stayed consistent and to handle the complex process of OOI neutralization. Combining low human intervention with good state estimation allowed us to address the key goals of the contest and produce a successful system.



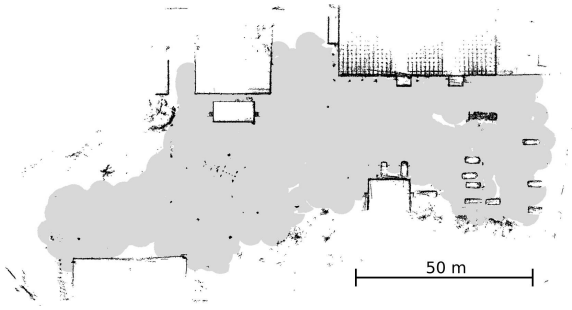
(a) Competition-day

(b) Satellite

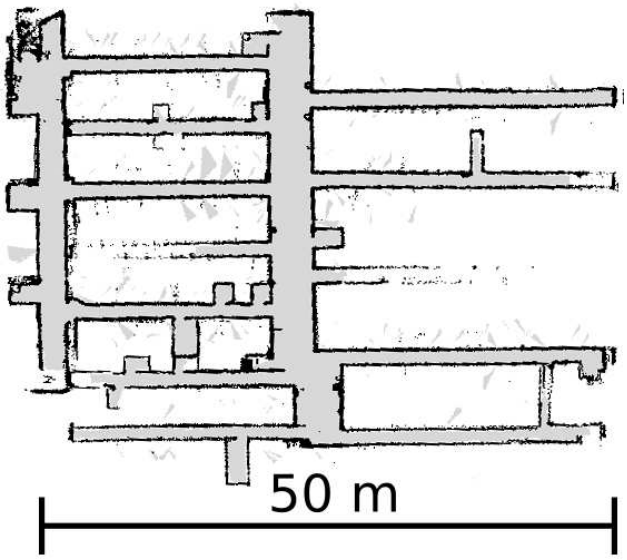
Figure 21: Combined map of all three phases produced during the MAGIC competition. (Phases indicated by dotted lines.) Some distortion is evident, which corresponds to failures of the mapping system to correct severe odometry errors that occasionally resulted from getting stuck on a curb or ledge. Full resolution maps are available from our website: `www.april.eecs.umich.edu`



(a) Old Ram Shed Challenge



(b) Keswick Barracks Map



(c) Adelaide Self-Storage

Figure 22: Additional environments in Adelaide SA that were mapped using our system.

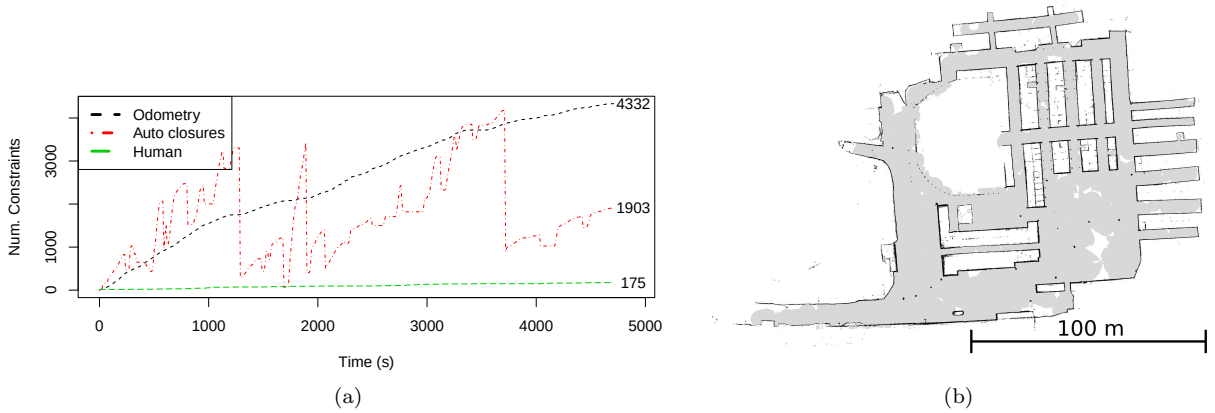
7 Evaluation

We have evaluated our reconnaissance system by analyzing our performance in the MAGIC 2010 contest and through additional experiments. Each team was allocated 3.5 hours to complete all three phases of the competition. We spent roughly 80 minutes each on phases 1 and 2 and 40 minutes on the final phase. Phase 1 consisted largely of an exhibition-style building whose interior was setup very similar to the June site-visits: it contained several 3 m x 3 m obstacles, in addition to a maze augmented with 15° plywood ramps. We successfully neutralized 5 of the 6 static OOIs in this phase. Phase 2 was both topologically and strategically more complex, with many more buildings and internal corridors, as well as mobile OOIs which patrolled outside along fixed routes. Our system was successful in neutralizing both of the mobile OOIs, which required complex coordination of two sensor robots tracking a moving target at a range of over 10 meters for at least 30 seconds. We also were able to neutralize 5 of the 6 static OOIs in this phase. Phase 3 was considerably larger in area than phases 1 and 2, although the entrance into the phase was blocked by a tricky combination of a patrolling mobile OOI and non-combatant. Ultimately, this situation proved to be too strategically challenging for our system to handle in the allocated time, highlighting the need for continued work in real-world, multi-agent robotics.

Our system evaluation focuses on two components: the performance of the mapping system and the amount of human interaction with the system over the course of the competition. Our evaluation is made possible by the comprehensive logs which we collected on each robot and the GCS during all of our testing and at the MAGIC competition. These logs contain all data returned from each of the sensors on every robot and every command issued by the GCS to each robot as well all the data passed between each module of our system infrastructure. In addition to helping us quickly develop such a complex software system, we expect that our logging infrastructure enables us to evaluate our performance during the MAGIC competition to a better degree than most of our competitors.

7.1 Map Quality

One of the main evaluation criteria for the MAGIC competition was the final quality of the map. The judges, working with the National Institute of Standards and Technology (NIST), quantitatively evaluated the quality of each team's map. This evaluation was based on fiducial markers whose positions were ground-truthed and manually extracted from each team's map (Jacoff, 2011). The map we submitted for evaluation during the competition can be seen in Fig. 21. This map, combined with our OOI detection and neutralization



(a) (b)

Figure 23: Source of edges in the SLAM Graph during an additional experiment in (a) and the resulting map in (b). Dips in the number of automatically added edges show when the operator used *simplify*, while the increasing number of human-initiated edges shows the interaction rate of the operator to mitigate odometry errors and false loop closures.

performance, earned us first place at the competition. Unfortunately, the raw data used by the judges was not made available to the teams, and so we cannot provide any additional analysis of the errors.

Instead, we have shown the corresponding satellite map, which shows the rough structure of the venue. The contest organizers modified the topology from what is shown in the overhead view by placing artificial barriers to block some hallways and building access points. It is still apparent that the maps we produced are topologically accurate and that all structural elements are present. However, local distortion is evident in some portions of the map. These represent failures of the autonomous loop closer to correct severe odometry errors, such as when a robot experiences slippage due to catching on a curb.

In environments where mobility was less of an issue, we were able to produce maps that are completely distortion free. Maps from several of our test runs outside the competition are shown in Fig. 22. Although they are on a smaller scale than the MAGIC contest, they showcase the high quality maps our system is capable of producing.

7.1.1 Human assistance during mapping

Our sensor operator assisted our system in maintaining a consistent global map. A key performance measure for our team was the amount of cognitive effort required of the human to support this process. We measure this effort in terms of the number of edges added by the human. One might reasonably want to measure this cognitive effort in terms of time, but this is difficult to do without introducing a great deal of subjectivity: the actual time spent interacting with the user interface is quite short—typically under ten seconds per

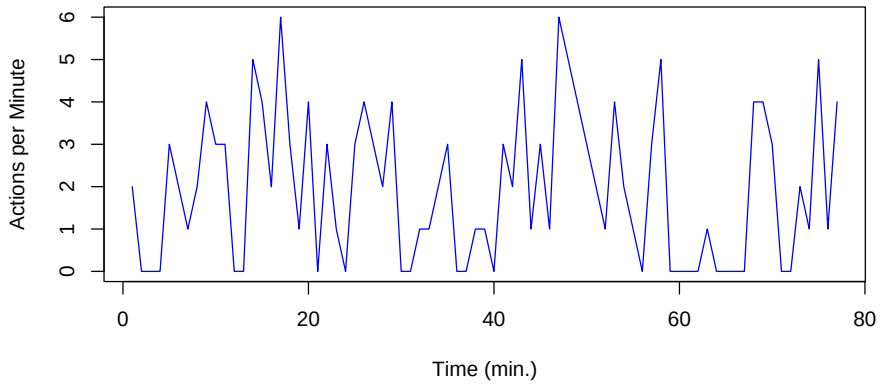


Figure 24: Actions-per-minute for the sensor operator during an additional experiment. Several extended periods of inactivity show our mapping system is capable of significant periods of full autonomy.

edge—and wouldn't reflect the effort spent watching the displays for problems (which doesn't require any measurable action).

The data necessary to evaluate this metric was not collected during our competition run, thus we replicated the run by playing back the raw data from the competition log and having our operator re-enact his performance during the competition. These conditions are obviously less stressful than competition, but are still representative of human performance. We used phase 2, which included a significant amount of odometric error requiring operator attention and included over 70 minutes of activity.

The result of this experiment was a map that was better than that during the competition (compare Fig. 23(b) to Fig. 21(a)) and the loop closing performance shown in Fig. 23(a)). Two types of interactions are clearly evident: 175 loop closures were manually added (on average, one every 24 seconds), and these comprised a very small fraction of the total number of loop closures. The figure also shows abrupt *decreases* in the number of automatic closures; these represent cases in which the human operator invoked the "graph-simplify" command to increase the speed of optimization (as described in Section 4.7).

The number of interactions with the system is plotted as a function of time in Fig. 24. The overall average rate is 1.87 interactions per minute, but generally occurred in bursts. At one point, the operator did not interact with the system for 5.17 minutes.

7.2 Human assistance during task allocation

The task operator assisted the system by either delegating task allocation to the autonomous planners or assigning tasks manually. The key performance measure of the system is the cognitive effort required by the

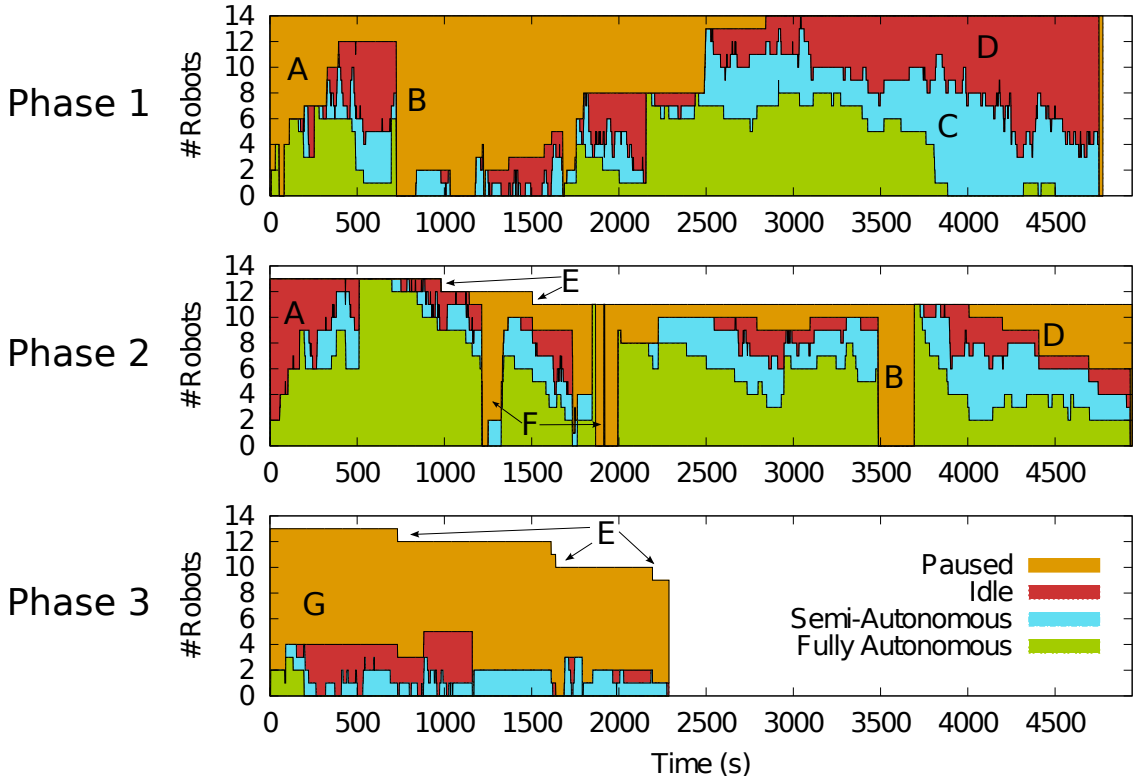


Figure 25: Robot modes of control during MAGIC with significant events annotation. At the beginning of each phase, operators strategically released robots in waves (A) to provide initial separation. Serious mapping errors occurred in both of the first two phases (B), for example, early in phase 1, a catastrophic odometry error on a single robot caused the global state to diverge, forcing the operators to pause all of the robots while they fixed the problem. The task operator slowly released robots and manually guided them to make important loop closures, bringing the map back under control. In phase 2, the mapping error was caused by an incorrect automatic loop closure, causing the warping near the top of our phase 2 map, as seen in Fig. 21. Near the end of phase 1, the autonomous planner proved unable to finish navigating a challenging maze of ramps, causing the operators to attempt to semi-autonomously navigate several robots through the maze (C). The navigational issues proved design related (our wheels, excellent for skid-steering, had difficulty gripping the smooth ramps) and the operators were unable to completely explore the maze. As phases 1 and 2 neared completion, the operators allowed robots to remain idle or paused them (D), seeing no additional tasks to carry out. (E) denotes robots being disabled by mobile OOIs. At times, operators paused the entire team (F) to prevent robots from wandering too close to these mobile OOIs. In the final phase, we quickly encountered a difficult mobile OOI neutralization (G) that blocked the safe passage to the rest of phase 3. The operators left the team paused in a safe area as they attempted to position pairs of robots for neutralization, but competition time expired before we were successfully able to neutralize the mobile OOI.

Phase	Pause	Waypoint Assignment	Neutralize Static	Neutralize Mobile	Comm. Relay Mode
1	60	405	7	0	2
2	68	267	4	9	5
3	21	142	0	3	0

Table 3: Number of tasks assigned semi-autonomously by type.

human to support planning for the fourteen robots. As with mapping, this is difficult to measure objectively.

We considered three metrics which, together, characterize the degree to which our system operated autonomously: 1) interactions per minute, 2) time spent in fully autonomous/semi-autonomous/tele-operated mode, and 3) distance traveled in fully autonomous/semi-autonomous/tele-operated mode.

A key measure of our system is that we made no use of our tele-operation capability; our system always operated in either fully-autonomous mode (where tasks are assigned and carried out autonomously) or semi-autonomous (where tasks are assigned manually but carried out autonomously).

Before presenting results for the three metrics, it is helpful to know how the competition unfolded. In Fig. 25, the mode assigned to each robot is plotted over time. The figure captures the major events in the competition, graphically depicting the number of robots that were active at each point in time, and highlighting when problems occurred. For example, the period of time in which all robots were paused in order to allow the sensor operator to correct the global map is readily apparent.

Interactions per minute. Interactions per minute directly measure the number of commands issued by the human operator to the system, and thus represent a measure of how "busy" the human operator was. It is an imperfect measure of cognitive loading, however, since it does not reflect the effort expended while watching or thinking about what the system should do. Conversely, it tends to over-count the number of salient interactions; for example, the operator may provide a series of waypoints to a robot as a single coherent action, but this would be counted as multiple interactions. Still, the measure is objective, well-suited to comparisons between teams, and reasonably representative of load.

See Fig. 26 for the number of interactions per minute for the task operator during the MAGIC competition. The task operator went up to 2 minutes without interacting with the system and averaged 4.25 interactions per minute over the entire competition. The figure shows that the task operator tended to intervene when problems occurred in prolonged bursts, rather than small, quick system adjustments.

The type of human interaction sheds light on the limitations of what the autonomous system could not

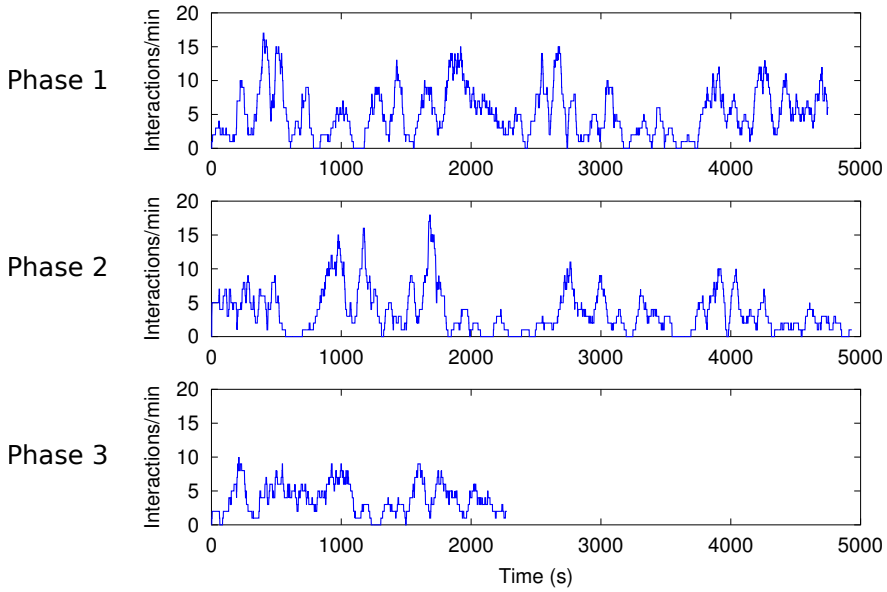


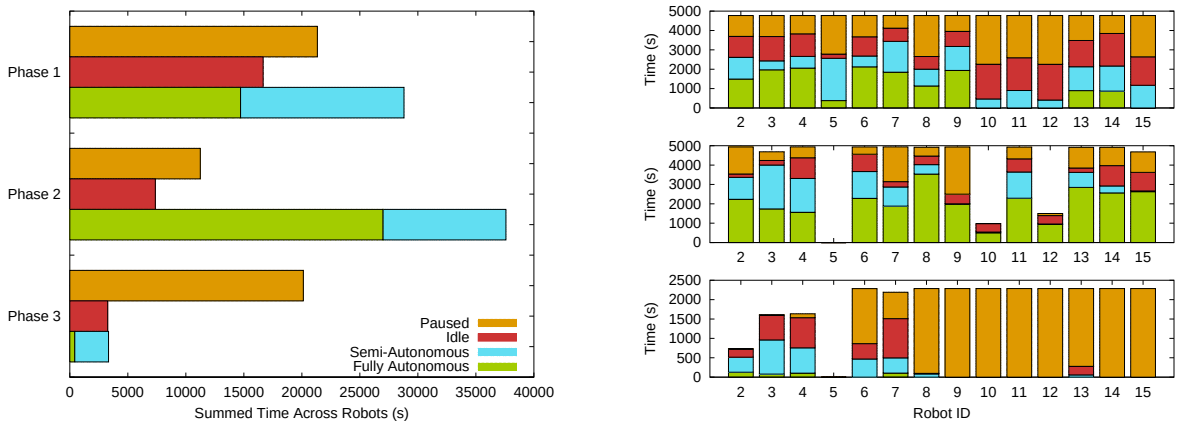
Figure 26: Task operator interactions per minute. Plots are displayed for each of the phases of MAGIC. Higher peaks in the plots indicate frequent interactions by the operator. Abrupt spikes indicate batch operation such as releasing all agents to the autonomous planners or pausing the entire team of robots.

Phase	Fully Autonomous	Semi-Autonomous	Tele-operation
1	0.220	0.211	0.0
2	0.480	0.189	0.0
3	0.015	0.109	0.0

Table 4: Ratios of operational time spent in each mode of control to total time spent in each phase of MAGIC. Though the robots also spent time paused or idle, all tasks were executed completely autonomously. Numbers do not sum to one due to additional time spent in either paused or idle states.

accomplish on its own. The sensor operator only interacted with the system to assist in the global state estimate, with the very occasional OOI confirmation, and thus offers no addition insight. However, the task operator interacted with various types of high-level commands as seen in Table 3. Since we did not design mobile OOI tracking or the final step of static OOI neutralization (i.e., turning on and off the laser pointer), we expect these tasks to require intervention. However, as Table 3 shows, the task operator intervened mostly to give robots movement commands. These human-directed movement commands were not for exploration tasks, but were used to force loop closures and position robots for OOI neutralizations. Had the autonomous planners explicitly coordinated robots to cause large-scale loop closures rather than solely exploring new terrain, a significant amount of task operator interaction would have been eliminated.

Time spent in autonomous/semi-autonomous mode. A robot that spends all of its time in fully autonomous mode requires virtually no management effort on the part of the human operator. In contrast, a robot that spends its time in semi-autonomous mode must periodically receive new tasks from the human



(a) Phase Total

(b) Per Robot

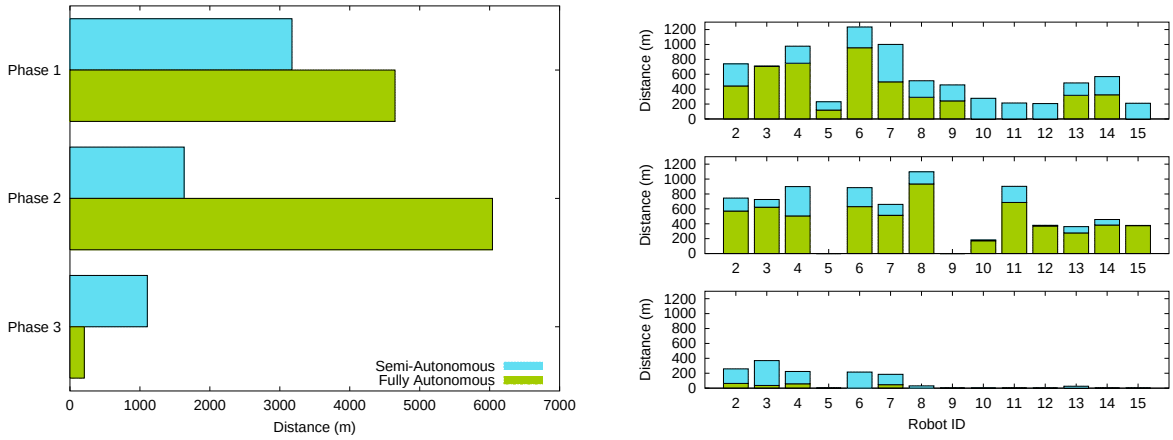
Figure 27: Time spent in each robot state during MAGIC. In (a) the combined amount of time robots spent in each mode of control is shown for each phase while (b) shows exact times for each individual robot. Shortened bars indicate robots being disabled or, in the case of robot 5, being removed from the competition due to mechanical difficulties.

operator. The ratio of time spent in these two modes thus measures both the degree-of-autonomy of the system and the load on the operator.

The amount of time spent in our various modes is shown graphically in Fig. 27 and in Table 4. These plots additionally show time spent in paused and idle, but the key information is the ratio between semi-autonomous and fully-autonomous. In phase 1, robots split their time between fully-autonomous and semi-autonomous. In phase 2, they were in fully-autonomous mode (relying on the autonomous planners for their task assignments) over 71% of the time. In phase 3, the situation was reversed; requiring the robots to move at very specific times and to strategically-chosen locations in order to avoid being destroyed by a hostile OOI. As a result, they were in fully-autonomous mode only 12% of the time.

We additionally plot the breakdown on a per-robot basis in Fig. 27, which provides additional insight into how the operator's time was distributed. What is immediately clear is that the ratios are *not* uniform between robots. Instead, some robots require more manual task assignment than others. This is to be expected: robots in challenging areas require assistance quite frequently. Meanwhile, robots in relatively straightforward environments make continuous progress with virtually no supervision.

Distance traveled in autonomous/semi-autonomous mode. This metric is similar to the previous metric, but instead of tallying the amount of time spent in each mode, it measures the distance traveled in each mode. We use "distance traveled" as a proxy for "useful work accomplished". This is reasonable



(a) Phase Total

(b) Per Robot

Figure 28: Distance traveled in each autonomous mode during MAGIC. (a) shows the combined distance traveled by all robots in each phase while (b) shows the distances for individual robots. Tele-operation was omitted from this figure as it was not employed during MAGIC.

Phase	Fully Autonomous	Semi-Autonomous	Tele-operation
1	0.594	0.406	0.0
2	0.787	0.213	0.0
3	0.156	0.844	0.0

Table 5: Ratios of distance traveled for each mode of control to total distance traveled in each phase of MAGIC.

since the bulk of the competition relates to exploring the large contest area. The ratio of distance traveled between these two modes measures the effectiveness of the autonomous task allocation and the load on the operator.

In the three phases, the distance traveled by the robots in fully-autonomous mode 59.4%, 78.7%, and 15.6% of the total distance (see Fig. 28 and Table 5). These numbers are in rough agreement with the time spent in fully-autonomous mode, providing evidence that the autonomous task allocation is as effective as task allocations by the humans. This is in agreement with our subjective impression of the system: human intervention is usually required to deal with some sort of problem (a robot stuck on an obstacle, for example) rather than to correct a gross inadequacy in the automatically-computed task.

8 Conclusion

Autonomous urban reconnaissance missions like MAGIC 2010 present interesting challenges in global state estimation, multi-agent cooperation, human-robot interfaces, and robot perception. We developed a system

of fourteen robots able to produce metrical maps of large scale urban environments without relying on prior information such as GPS or satellite imagery, and with only modest human assistance.

Our Decoupled Centralized Architecture (DCA) is a key idea underlying our system design, allowing robots to cooperate but without the communication and global state synchronization challenges typical of a fully centralized system. Crucial to enabling coordination between robots was our autonomous global state estimation system, based on a graph-SLAM approach and an automatic loop-closer with a low false-positive rate. Individual robots navigated autonomously by analyzing a 3D LIDAR point cloud to identify drivable terrain and dangerous objects. Robots autonomously planned paths through this environment to their specified goal locations.

Incorporating a human in the loop presents opportunities and challenges. A human can provide valuable guidance to planning and can assist with sensing and mapping tasks. However, it is challenging to provide a comprehensive view of the system's state without overwhelming the user. Our system addressed this by allowing humans to focus on different tasks (planning and sensing), and to support them with user interfaces that alerted them to events worthy of their attention. These interfaces provided both an accurate situational awareness of individual robots and the entire team, and the ability to intervene at multiple levels of autonomy, when required.

The MAGIC 2010 competition showcases the progress that has been made in autonomous, multi-agent robotics. However, it also highlights the shortcomings of the current state-of-the-art. First, it remains difficult to maintain a consistent coordinate frame between a large number of robots due to challenges like wheel slip and perceptual aliasing. Our system coped with these failures at the expense of greater human operator workload. Second, it remains unclear how to optimally integrate human commanders into a multi-robot system, i.e., how to design user interfaces, manage cognitive workload, and coordinate autonomous and manual tasking at multiple levels of abstraction. We demonstrated a workable approach, but a more systematic study could help identify general principles. Finally, maintaining reliable communications remains a challenge: much of our approach was constrained by the performance of our radio system. Ultimately, we feel that competitions like MAGIC 2010, motivated by real-world problems, are invaluable in identifying important open problems and in promoting solutions to them. These competitions serve as a reminder that there are few truly "solved" problems.

Acknowledgments

Special thanks to all our sponsors, including Intel and Texas Instruments, which allowed us to build a large team of robots. Our robot design greatly benefited from the design and CAD work of Javier Romero. The Oaks Plaza Pier Hotel and Glenelg Self-Storage in Adelaide, Australia graciously allowed us to operate our robots in their facilities. Many University of Michigan Undergraduate Research Opportunity Program (UROP) students helped design, build and, test our system. Our host and liaison in Adelaide, Chris Latham, was both friendly and helpful beyond any reasonable expectation. We also thank our anonymous reviewers for their helpful suggestions. Finally, our congratulations to TARDEC and DSTO for successfully organizing such a large and complex challenge.

References

Andriluka, M., Friedmann, M., Kohlbrecher, S., Meyer, J., Petersen, K., Reinl, C., Schauß, P., Schnitzspan, P., Thomas, D., Vatcheva, A., and Stryk, O. V. (2009). Robocuprescue 2009- robot league team darmstadt rescue robot team (germany).

Bahr, A. (2009). *Cooperative Localization for Autonomous Underwater Vehicles*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Balakirsky, S., Carpin, S., Kleiner, A., Lewis, M., Visser, A., Wang, J., and Ziparo, V. A. (2007). Towards heterogeneous robot teams for disaster mitigation: Results and performance metrics from robocup rescue. *Journal of Field Robotics*, 24(11-12):943–967.

Bhattacharya, P. and Gavrilova, M. L. (2007). Voronoi diagram in optimal path planning. In *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 38–47. IEEE.

Bosse, M. C. (2004). *ATLAS: A Framework for Large Scale Automated Mapping and Localization*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Burgard, W., Moors, M., Fox, D., Simmons, R., and Thrun, S. (2000). Collaborative multi-robot exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 476 –481 vol.1.

Cunningham, A., Paluri, B., and Dellaert, F. (2010). Ddf-sam: Fully distributed slam using constrained factor graphs. In *IROS*, pages 3025–3030. IEEE.

Davis, T. A., Gilbert, J. R., Larimore, S. I., and Ng, E. G. (2004). A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):353–376.

Dellaert, F. and Kaess, M. (2006). Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1203.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271. 10.1007/BF01386390.

ELROB (2006). European land robot trials.

Fischler, M. and Bolles, R. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis andd automated cartography. *Communications of the ACM*, 24(6):381–395.

Geraerts, R. and Overmars, M. H. (2007). Creating high-quality paths for motion planning. *International Journal of Robotic Research*, 26:845–863.

Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2432–2437, Barcelona.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007). A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems (RSS)*, Atlanta, GA, USA. The MIT Press.

Hähnel, D. and Burgard, W. (2002). Probabilistic matching for 3d scan registration. In *Proc. of the VDI-Conference Robotik 2002 (Robotik)*. VDI Verlag.

Himmelsbach, M., von Hundelshausen, F., Luettel, T., Manz, M., Mueller, A., Schneider, S., and Wuensche, H.-J. (2009). Team MuCAR-3 at C-ELROB 2009. In *Proceedings of 1st Workshop on Field Robotics, Civilian European Land Robot Trial 2009*, ISBN 978-951-42-9176-0, Oulu, Finland. University of Oulu.

Huang, A., Olson, E., and Moore, D. (2010). LCM: Lightweight communications and marshalling. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

IGVC (2011). Intelligent ground vehicle competition.

Jackel, L. D., Krotkov, E., Perschbacher, M., Pippine, J., and Sullivan, C. (2006). The darpa lagr program: Goals, challenges, methodology, and phase i results. *Journal of Field Robotics*, 23(11-12).

Jacoff, A. (2011). Technical adjudication of the magic2010 robot competition. Presentation to at TARDEC quarterly robotics review.

Kaess, M., Ranganathan, A., and Dellaert, F. (2007). iSAM: Fast incremental smoothing and mapping with efficient data association. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Rome; Italy.

Konolige, K., Agrawal, M., Blas, M. R., Bolles, R. C., Gerkey, B., Sol, J., and Sundaresan, A. (2009). Mapping, navigation, and learning for off-road traversal. *Journal of Field Robotics*, 26(1).

Langerwisch, M., Reimer, M., Hentschel, M., and Wagner, B. (2010). Control of a semi-autonomous ugv using lossy low-bandwidth communication. In *The Second IFAC Symposium on Telematics Applications (TA)*. IFAC.

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., Koch, O., Kuwata, Y., Moore, D., Olson, E., Peters, S., Teo, J., Truax, R., Walter, M., Barrett, D., Epstein, A., Maheloni, K., Moyer, K., Jones, T., Buckley, R., Antone, M., Galejs, R., Krishnamurthy, S., and Williams, J. (2008). A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774.

Matthies, L., Kelly, A., Litwin, T., and Tharp, G. (1996). Obstacle detection for unmanned ground vehicles: A progress report. In *Intelligent Vehicles' 95 Symposium., Proceedings of the*, pages 66–71. IEEE.

Montemerlo, M. (2003). *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597.

Moore, D. C., Huang, A. S., Walter, M., Olson, E., Fletcher, L., Leonard, J., and Teller, S. (2009). Simultaneous local and global state estimation for robotic navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Morton, R. D. and Olson, E. (2011). Positive and negative obstacle detection using the hld classifier. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Olson, E. (2008). *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Olson, E. (2009a). Real-time correlative scan matching. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan.

Olson, E. (2009b). Recognizing places using spectrally clustered local matches. *Robotics and Autonomous Systems*.

Olson, E. (2011). AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Olson, E., Leonard, J., and Teller, S. (2006). Fast iterative optimization of pose graphs with poor initial estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2262–2269.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition.

Richardson, A. and Olson, E. (2011). Iterative path optimization for practical robot planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Schafer, H., Hach, A., Proetzsch, M., and Berns, K. (2008). 3d obstacle detection and avoidance in vegetated off-road terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 923–928. IEEE.

Sibley, G., Mei, C., Reid, I., and Newman, P. (2009). Adaptive relative bundle adjustment. In *Proceedings of Robotics: Science and Systems*, Seattle, USA. The MIT Press.

Strom, J. and Olson, E. (2011). Occupancy grid rasterization in large environments for teams of robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2007). Stanley: The robot that won the darpa grand challenge. In *The 2005 DARPA Grand Challenge*, volume 36 of *Springer Tracts in Advanced Robotics*, pages 1–43. Springer Berlin / Heidelberg.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W. ., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8).

U.S. Army (2010). U.S. army unmanned aircraft systems roadmap 2010-2035.

Wein, R., Berg, J. P. V. D., and Halperin, D. (2008). Planning high-quality paths and corridors amidst obstacles. *International Journal of Robotic Research*, 27:1213–1231.

Yamauchi, B. (1998). Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, pages 47–53, New York, NY, USA. ACM.