To appear in IEEE INFOCOM, April 2006

Mobile Emulab: A Robotic Wireless and Sensor Network Testbed

David Johnson Tim Stack Russ Fish Daniel Montrallo Flickinger† Leigh Stoller Robert Ricci Jay Lepreau

University of Utah, School of Computing and Department of Mechanical Engineering†

***Abstract*— Simulation has been the dominant research methodology in wireless and sensor networking. When mobility is added, real-world experimentation is especially rare. However, it is becoming clear that simulation models do not sufficiently capture radio and sensor irregularity in a complex, real-world environment, especially indoors. Unfortunately, the high labor and equipment costs of truly mobile experimental infrastructure present high barriers to such experimentation.**

We describe our experience in creating a testbed to lower those barriers. We have extended the Emulab network testbed software to provide the first remotely-accessible mobile wireless and sensor testbed. Robots carry motes and single board computers through a fixed indoor field of sensor-equipped motes, all running the user's selected software. In real-time, interactively or driven by a script, remote users can position the robots, control all the computers and network interfaces, run arbitrary programs, and log data. Our mobile testbed provides simple path planning, a vision-based tracking system accurate to 1 cm, live maps, and webcams. Precise positioning and automation allow quick and painless evaluation of location and mobility effects on wireless protocols, location algorithms, and sensor-driven applications. The system is robust enough that it is deployed for public use.

We present the design and implementation of our mobile testbed, evaluate key aspects of its performance, and describe a few experiments demonstrating its generality and power.

I. Introduction

Experiments involving mobile wireless devices are inherently complex and typically time-consuming to set up and execute. Such experiments are also extremely difficult to repeat. People who might want to duplicate published results from another laboratory, for example, must devote substantial resources to setting up and running such a laboratory—and even then, the environmental conditions are likely to be substantially different. Duplicating one's own work is similarly difficult, due to the need to position and move mobile devices exactly as one did previously.

For these reasons, simulation has been a primary methodology for researchers in the wireless and sensor network domains. Simulations are easier to set up than physical experiments, are easy to repeat and modify, and are highly portable. It is becoming clear, however, that current simulators are unable to model many essential characteristics of the "real world" [2], [13], [20], [26], [27], [28]. The simplifying assumptions in current wireless simulators may result in differences between the behavior of the system in simulation and the behavior of the realized system in the real world.

Therefore, many mobile wireless systems should be studied and evaluated through experiments on actual mobile devices. As we also argued earlier [22], for such experiments to be commonplace, the capital costs and human effort required to perform such experiments must be substantially reduced—by an order of magnitude or more. A testbed for mobile wireless research should contain actual mobile devices, provide means for programming the devices, ensure that motion is performed accurately, and ease the collection of experimental data. To be economical, a single testbed must be shareable: in fact, it should be available online, and usable by researchers at sites far from the testbed itself. Finally, to provide diverse physical environments, the community needs several such testbeds. It is therefore important to reduce the cost of building and maintaining such a wireless testbed.

The mobile testbed we have developed has a clear role in evaluating new mobility-related network protocols, applications, and systems. Researchers may also use mobility to quickly construct and evaluate many different static network topologies. In addition, our testbed's high degree of automation combined with its accurate and precise antenna positioning provide new ways to develop and validate wireless simulation models, at all radio and network levels. Developers of such models can first test a simple model in a simple physical testbed environment, iteratively adding model sophistication until its results match the physical world. Environmental and model complexity can then be gradually added to improve the model's accuracy. The testbed's automation, as shown in Section VI-A, would make this long process practical.

In this paper we describe our experience in creating a production-quality testbed for mobile wireless and sensor network research. Our system, based on a major extension to the Emulab network testbed software [23], is designed to show that such testbeds can be built: they can provide accurate positioning and monitoring, can enable automated experiments by both on-site and off-site users, and can be built and maintained at relatively low cost using open-source software and commercial off-the-shelf (COTS) equipment. We believe that our testbed is an efficient and cost-effective solution, and is therefore an attractive (and often superior) alternative to simulation for experiments that involve mobile wireless devices.

Largely sponsored by NSF grants CNS–0335296 and EIA–0321350.
Contact: testbed@flux.utah.edu, http://www.emulab.net/.

Our initial deployment uses a small office space, robots that move at modest speed, and supports only sensor motes as wireless devices under test, typically with their radios set at low power. Thus our current testbed is limited in the physical environments that it can emulate fully accurately. However, we are quite certain that it is a valuable complement to simulation for exploring many wireless and sensor protocols and applications.

Moreover, the system is scalable and both the software and hardware are general. The software applies without change to 802.11, and the current robots could accommodate a second (attenuated) 802.11 interface on their iPaq-like computers running Linux, to act as an additional object of experimentation. In addition, we believe that both our overall design and its major software components could be ported to some non-Emulab-based wireless testbeds with only modest effort.

The contributions of this paper are as follows. First, we present our mobile testbed. To our knowledge, our additions to Emulab have created the first remotely accessible testbed for mobile wireless and sensor network research. It has been deployed for public production use since February 2005. Second, we show that our testbed model is economical, since it demonstrates that useful mobile testbeds can be built at modest cost, using COTS hardware and open source software. Third, we detail the algorithms that we developed as part of building our testbed from COTS parts. In particular, we describe how our mobile testbed ensures accurate robot positioning using medium-cost video camera equipment. Finally, we present results from initial experiments that were carried out in our mobile testbed. These results highlight the testbed's automation capabilities.

The rest of this paper is structured as follows. Following a review of related work in Section II, we present the testbed architecture in Section III. We then detail two issues that are essential for reliable mobile experimentation: accurate location of mobile devices (Section IV), and motion control (Section V), including validation and microbenchmark results. In the last parts of the paper, we describe initial examples of experimentation on our testbed (Section VI), discuss open issues and future work (Section VII), and conclude in Section VIII.

II. Related Work

One major way in which we differ from all of the related projects below is our integration with Emulab. By building on this popular and robust software platform for running network testbeds, which operates more than a dozen sites with thousands of users, we inherit many features useful to users and administrators. For example, Emulab was designed from the ground up to be a multi-user testbed environment, so that its resources can be shared by a large number of projects, and it supports a hierarchy of projects, groups, and users. Emulab also provides a rich environment for controlling experiments, including a scriptable "distributed event system," which various parts of the system can generate or react to—in Section VI, we run an experiment that makes use of this feature. Emulab supports over a dozen device types, including generic PCs, PlanetLab slivers, emulated widearea network links, real 802.11 links, and the simulated resources of *ns*. This feature is useful for experimenting with systems involving such a mixture, for example, hierarchical wireless sensor systems [12], including those incorporating nodes across the Internet, such as the "Hourglass" data collection architecture [16]. Emulab is Web-based and script or GUI-driven, but also exports an XML-RPC interface so that all interaction can be programmatic. Finally, Emulab can reliably handle experiments of very large scale, a major challenge [14].

MiNT [10] is a miniaturized 802.11 testbed. Its focus is on reducing the area required for a multihop 802.11 testbed, and on integrating *ns* simulation with emulation. It achieves mobility through the use of antennas mounted on Lego MindStorm robots, tethered to a PC running the applications. To avoid antenna tangle, each robot is limited to moving within its designated square. In contrast, in this paper our focus is on the mobility of the robots, while our example hardware environment is a wireless sensor network, although our software can also control 802.11 networks. We have untethered robots, so that mobility is not hampered by cables, and provide accurate movement and "ground truth" location of the robots, whereas MiNT does not address positioning accuracy.

The ORBIT testbed [18] provides a remotely-accessible 64-node (planned 400) indoor grid with 1 m spacing, that can emulate mobility. User code is run on PCs, which are dynamically bound to radios. By changing the binding of a PC to a radio, the PC appears to move in discrete hops. In contrast, we focus on true mobility, and target sensor networks as well as 802.11. The ORBIT project plans to add an outdoor component with constrained mobility, although perhaps not with external access. It will use 3G and 802.11 radios carried by campus buses on fixed routes.

MoteLab [21] and EmStar [11], [12] both support fixed sensor network testbeds. MoteLab is a Web-accessible software system for general-purpose sensor network experimentation, and has been installed at several sites. Like many testbeds, including Emulab, it assumes a separate "control network" over which it provides a number of useful features. As do we, these include mass programming of the motes, logging all transmitted packets, and the ability to connect directly to the motes. MoteLab uses a reservation-based scheduling system with quotas, whereas we currently use Emulab's dual model of a batch queue and interactive first-come first-served use.

The EmStar "ceiling array" testbed is used primarily for the development of the EmStar software, which focuses on the integration of "microserver" nodes, which have more processing power than a typical sensor network node, into sensor networks. A clever aspect of the EmStar software is that applications emulating the sensor nodes can be run on a single PC, yet optionally be coupled to real, distributed wireless devices for communication.

The Mirage testbed [4] is complementary to all of the other testbeds in that it uses resource allocation in a sensor net testbed as a target to explore new models of resource allocation, e.g., market-based models.

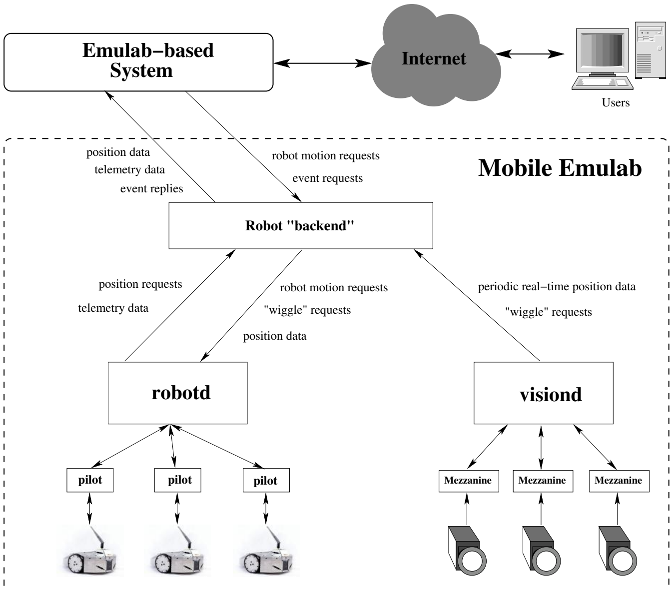


Fig. 1. Mobile Emulab software architecture.

The WHYNET [24] project eventually intends to integrate a large number of types of wireless devices, although it is not clear they will ever provide remote access. The ExScal project at Ohio State is investigating fixed sensor networks up to thousands of nodes, but remote access is also not a priority. The SCADDS project at USC/ISI has a fixed testbed of 30 PCs with 418MHz radios deployed in an office building.

III. System Overview and Design

Emulab is designed to provide unified access to a variety of experimental environments. It provides a Web-based front end, through which users create and manage experiments, a core which manages the physical resources within a testbed, and numerous back ends which interface to various hardware resources. The core consists of a database and a wide variety of programs that allocate, configure, and operate testbed equipment. Back ends include interfaces to locally-managed clusters of nodes, virtual and simulated "nodes," and a PlanetLab interface. Emulab users create "experiments," which are essentially collections of resources that are allocated to a user by the testbed management software, and act as a container for control operations by the user and system.

We extended the existing Emulab framework for our testbed, which includes both robot-based mobile wireless devices and new software for managing those devices within the testbed. The software architecture of our mobile testbed, and its relationship to Emulab, are shown in Figure 1.

A. Software Architecture

Our testbed software provides experimenters the capability to dynamically position robots and use them to conduct experiments. To this end, we have provided new user control and data interfaces. We have also written a backend which directly interfaces with both Emulab and the new components of our robot control system called *visiond*, *robotd*, and *pilot*. This allows users to track robot location changes in real-time and ensures that robots reach their destinations.

Component structure and dataflow is shown in Figure 1 and described below. When an experimenter requests that a robot be moved to a new position, the request is passed through Emulab to the backend. The backend performs bounds-checking on the requested position, and passes it down to the robot control component, called *robotd*. *robotd* queries the backend to ensure that it has the latest location data for the robot in question, then breaks up the position request into a series of primitive movements for the *pilot* daemon on the robot. After each primitive move is completed, feedback from the vision system is used to correct any positioning errors. This process is repeated until the robot reaches a position within a fixed distance from the requested position or a retry threshold has been reached.

Robot Localization. Tracking and identification of the robots is handled by a vision-based tracking system called *visiond*, using ceiling-mounted video cameras aimed directly down. As described later in Section IV, we improved an open source object tracking software package to transform the overhead camera video into x, y coordinates and orientation for detected objects. Individual camera tracks are then aggregated by *visiond* into a single set of canonical tracks that can be used by the other components. These tracks are reported at 30 frames per second to the backend since queries from *robotd* require high precision. The backend in turn reports snapshots of the data (one frame per second) to the Emulab database, for use by the user interfaces. This reduction in data rate is an engineering tradeoff to reduce the communication bandwidth with, and resulting database load on, the Emulab core.

Robot Control. The *robotd* daemon centrally plots paths so that robots reach their user-specified positions safely and efficiently. It also maneuvers robots around any dynamic obstacles encountered during motion. Individual motion commands are sent to the *pilot* daemon, which runs on each robot and provides a simple motion control interface for each robot. *pilot* currently uses the low-level motion commands and data structures present in the API provided for Acroname Garcia [1] robots. The Acroname API exposes additional robot data and functionality, such as battery levels, monitoring sensor values, adjusting sensor sensitivities and thresholds, and adjusting general robot parameters. Path planning is described in Section V.

B. User Interaction

The hardware in the mobile testbed is allocated and configured much like the ordinary PCs that are already a part of Emulab. The user creates an "experiment" through a web interface and submits *ns* code that specifies the topology. Besides operations like setting the TinyOS kernel to upload to a mote, in the *ns* file a user can schedule events that control robot movement, program start and stop, etc., using Tcl and our event constructs. For example, Figure 2 shows an excerpt of the code used in one of our experiments to walk a robot around an area in half meter increments and

```
set walker [$ns node]
set ltor 1
for {set y 0} {$y <= $HEIGHT} {set y [expr $y + $YINCR]} {
    set row($rowcount) [$ns event-sequence {}]
    for {set x 0} {$x <= $WIDTH} {set x [expr $x + $XINCR]} {
        if {$ltor} {
            set newx [expr $XSTART + $x]
            set newy [expr [$walker set Y_] + $y]
        } else {
            set newx [expr $XSTART + $WIDTH - $x]
            set newy [expr [$walker set Y_] + $y]
        }

        if {[$topo checkdest $walker $newx $newy]} {
            $row($rowcount) append \
                "$walker setdest $newx $newy 0.2"
            $row($rowcount) append \
                "$logger run -tag $newx-$newy"
        }
    }
    $rowwalk append "$row($rowcount) run"
    incr rowcount
    set ltor [expr !$ltor]
}
```

Fig. 2. Extended NS code used to walk a robot around an area and log output from the onboard mote.

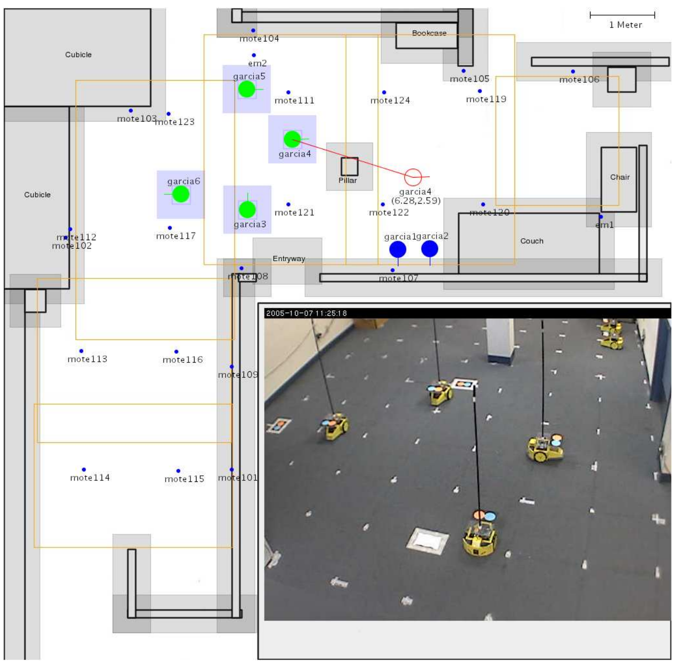


Fig. 3. The robot mapping and positioning GUI lets users track robot movement and control it with click and drag operations. Large filled circles are robots; empty circles show goal destinations; smaller filled dots are fixed mote locations; obstacles are in grey. This composite superimposes one webcam's view of part of the robot arena onto the GUI view.

log data received on the mote. For more interactive use, we have developed several new user interfaces, including Java applets to dynamically position robots via a drag-and-drop interface and view telemetry data in real-time. Live images of the robot testbed are provided via webcams. Figure 3 shows the positioning applet with a superimposed webcam image.

C. Hardware Resources

Space. As shown in Figure 3 the mobile testbed is currently deployed in an L-shaped area of 60 m^2, 2.5 m high, with six robots that can be positioned anywhere in that area.

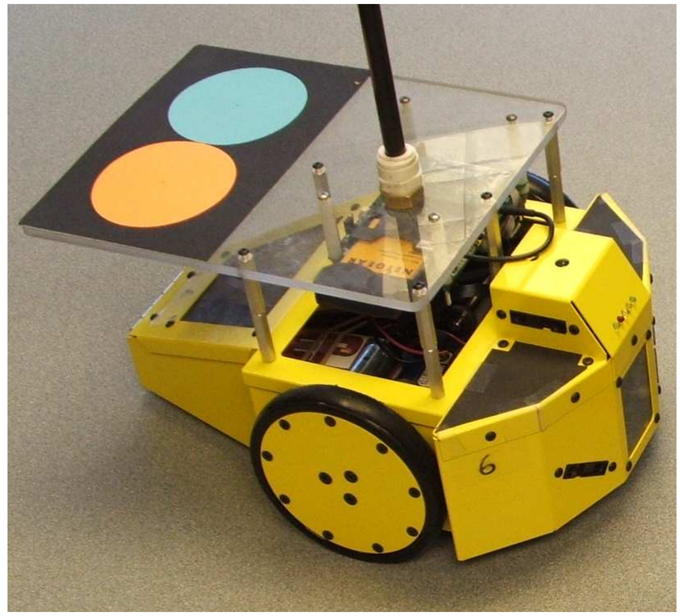


Fig. 4. Acroname Garcia robot with Stargate computer, WiFi card, Mica2 mote, extended platform with tracking fiducial, and base of antenna wand. The coaxial cable loop on the right goes to the antenna. Note that the antenna is centered over the axle, coincident with the origin of the robot coordinate system, avoiding antenna translation when pivoting.

Overlooking this area are six[1] cameras used by the robot tracking system and three webcams that provide live feedback to testbed users.

The area in which we are currently operating is a mix of "cube" and regular office space on the top floor of a four story steel-structure building. The space is carpeted, flat, and clear of obstructions, except for a single steel support beam in the middle of the room. The area is "live," with people moving near and across the area during the day. This aspect of the space adds a certain amount of unpredictability and realism to experiments. Removing this aspect of the space could be done by running the robots during off-hours. To do that, we need rechargers for the robot's batteries that need no human intervention; Acroname is building them.

Robots. We currently use Acroname Garcia robots, which we chose for their reasonable size (19x28x10 cm), cost ($1100), ease of use, and performance characteristics. The use of differentially steered two-wheeled robots simplifies the kinematics and control model requirements. Using a commercial platform avoided the overhead of addressing the many engineering issues inherent in in-house robot design and construction. Because our testbed is based on a readily available commercial robot platform, other teams can replicate the testbed with modest effort.

The robots operate completely wirelessly using 802.11b and a battery that provides at least two to three hours of use to drive the robot and power the onboard computer and mote. Motion and steering come from two drive wheels that have a rated maximum of two meters-per-second. Six infrared

[1]The number of cameras required is unrelated to the number of robots; that they are equal today is a coincidence.

proximity sensors on all sides of the robot automatically detect obstructions in its path and cause it to stop. These sensors are a key component in making it possible to run the robots in a "live" space, since the readings provide a means to detect and navigate around previously unknown obstacles.

The factory configures the Garcia robots to carry an XScale-based Stargate [9] small computer running Linux, to which we attach a 900MHz Mica2 [8] mote. To the Stargate we attach an 802.11b card that acts as a separate "control network," connecting the robot to the main testbed and the Internet. The Stargate serves as a gateway for both Emulab and the experimenter to control and interact with the mote, and for the user to run arbitrary code. Users can login to the Stargate, and will find their Emulab home directory NFS-mounted. Figure 4 shows a fully-outfitted robot. For the experiments reported in this paper the mote's antenna location was standard, directly attached to it, right on the Stargate board. We have recently raised the antennas to waist level using carbon-fiber wands and coaxial cable. The antenna location now more closely approximates human-carried nodes, and avoids RF ground effects and interference from the robot body.

We selected 900MHz radios for their multihop capability in the constrained space of our initial testbed. Testing several different radios (Mica2 900/433, Telos 802.15.4) at all power levels showed that only 900MHz was able to provide an "interesting" transmission distance of a few feet.

Fixed Motes. The stationary motes, currently numbering 25 in our modest prototype, are arranged on the ceiling in a roughly 2-meter grid and on the walls near the floor. All of the fixed motes are attached to MIB500CA serial programming boards [6] to allow for programming and communication. The 10 near-floor motes also feature an MTS310 full multi-sensor board [7] with magnetometers that can be used to detect the robot as it approaches. These motes are completely integrated with the Emulab software, making it trivial to load new kernels onto motes, remotely interact with running mote kernels via their serial interfaces, or access serial logs from experiments.

Finally, most Emulab testbeds, including ours, provide dozens or hundreds of PC nodes. Experimenters can leverage these nodes as processing stations for WSN experiments, or can use them together with mobile wireless and sensor nodes to create diverse computer networks.

IV. ROBOT LOCALIZATION: *visiond*

For accurate and repeatable experiments, our mobile testbed must guarantee that all mobile antennae and RF devices are at their specified positions and orientations. Accurate localization is also important for robot motion as described in Section V. Robot localization must scale to cover an area sufficiently large to enable interesting multi-hop wireless experiments. Finally, a localization solution must be of reasonable cost in terms of setup, maintenance, and hardware.

As is typically the case in robotic systems, our robots' on-board odometry could not localize the robots with sufficient accuracy. We therefore developed a computer vision-based localization system to track devices throughout our experimental area. Our vision algorithms process image data from video cameras mounted above the plane of robot motion. These algorithms recognize markers with specific patterns of colors and shapes, called *fiducials*, on each robot, and then extract position and orientation data.

To obtain high-precision data while limiting hardware costs, we made a number of engineering tradeoffs. First, we mount video cameras above the plane of robot movement looking down, instead of installing one on each robot. This solution is economical: not only does it remove requirements from the robots (power, CPU time, etc.), but overhead cameras can track many robots at once. Second, our video cameras are pointed straight down, perpendicular to the plane of robot movement. As described below, this simplifies the required camera geometry models and increases precision. Third, all robots are marked with the same, simple fiducial. This simplifies the object recognition algorithms and lowers processing time per image. Finally, we use relatively low-cost cameras [15] ($460) with even cheaper ultra-wide angle lenses ($60). This gives us a modest cost per camera and requires fewer of them. Surprisingly, we found that relatively simple image processing algorithms compensate for the resulting image distortion.

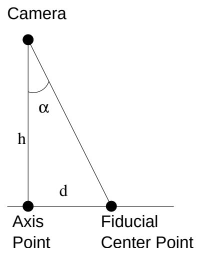
A. Localization Software

We use Mezzanine [17], an open-source system that recognizes colored fiducials on objects and extracts position and orientation data for each recognized fiducial. Each fiducial consists of two 2.7 inch circles that are widely separated in color space, placed next to each other on top of a robot. Mezzanine's key functionality includes a video image processing phase, a "dewarping" phase, and an object identification phase.

During the image processing phase, Mezzanine reads an image from the frame grabber and classifies each matching pixel into user-specified color classes. To operate in an environment with non-uniform and/or variable lighting conditions, the user must specify a wider range of colors to match a single circle on a fiducial. This obviously limits the total number of colors that can be recognized, and consequently, we cannot uniquely identify robots through different fiducials. Instead, we finesse this area's hard problem by exploiting whole-system capabilities. We obtain unique identification by commanding and detecting movement patterns for each robot (the "wiggle" algorithm), and thereafter maintain an association between a robot's identification and its current location as observed by the camera network. Mezzanine then combines adjacent pixels, all of which are in the same color class, into color blobs. Finally, each blob's centroid is computed in image coordinates for later processing (i.e., object identification).

B. Dewarping Problems

The original Mezzanine detected blobs quickly and effectively, but the supplied dewarping transform did not provide nearly enough precision to position robots as exactly as we needed. The supplied dewarping algorithm is a global function *approximation*.



where

$$\alpha = \arctan \frac{d}{h}$$

$$d_{image} = d_{world} * \cos(a * w)$$

and w is the "warp factor."

Fig. 5. Cosine dewarping.

We observed two important problems. First, the function was a poor fit for lens distortion, so it was necessary to add more control points to improve the fit. Second, the grid shape by the approximation function would tilt and bend globally when any control point was moved, so it never provided high precision anywhere, and was extremely sensitive to small movements of the control points. This produced high variability in position data returned by Mezzanine. We observed that moving a fiducial 1–2 cm in the motion plane resulted in a 10–20 cm jump in the fiducial's reported position.

C. An Improved Dewarping Algorithm

To avoid these problems, we replaced the dewarping algorithm with a geometric transformation that accounts for observed mathematical properties of wide-angle lens distortion, and included interpolative error correction to further reduce our error. We obtain an accuracy of 1 cm worst-case absolute error, and 0.34 cm RMS absolute error.

Our dewarping algorithm is based on correcting cosine radial lens distortion as a function of the angle from the optical axis of the lens. Figure 5 describes the algorithm. A single parameter (the *warp factor*) adjusts the period of the cosine, and two parameters adjust the linear scale factors in X and Y to calibrate the image-to-world coordinate transformation, based on center, edge and corner point locations surveyed to 1-2 mm accuracy. Other parameters include the height of the camera's focal point above the plane of robot motion and the position of the optical axis in the image. We use Mezzanine to accurately locate the optical axis as the pixel point where the coordinates of a fiducial remain stationary as the lens is zoomed in and out.

Cosine dewarping linearizes the geometric field (straightening out the barrel distortion into a flat grid.) We zero out the residual error left after cosine dewarping at these calibration points, and interpolate the error correction over blending triangles that span the space between the measured points by way of barycentric coordinates [5], [25].

The surprise is that even with cheap lenses, we didn't need to consider any distortion except radial. We were prepared to continue modeling and correcting asymmetries radiating from the optical axis, or moving circularly around it, as textbooks suggest. However, these $60 lenses conform closely to the simple cosine model.

Only nine calibration points per camera are used in this scheme, which is a small enough number to be handled automatically from multiple fiducials by the Mezzanine calibration program. It also leaves the remainder of the measured grid points (25 to 44 per camera in our irregular space) to measure the performance of this approach. We evaluated several triangle patterns and selected this one for its accuracy and simplicity of algorithm.

Metric	Algorithm		
	original	cosine dewarp	+ error interp
Max error	11.36 cm	2.40 cm	1.02 cm
RMS error	4.65 cm	1.03 cm	0.34 cm
Mean error	5.17 cm	0.93 cm	0.28 cm
Std dev	2.27 cm	0.44 cm	0.32 cm

TABLE I
LOCATION ERROR MEASUREMENTS

Figure 6 graphically compares location errors at grid points before and after applying the error interpolation algorithm. Figure 6(a) shows measurements of the cosine dewarped grid points and remaining error vectors across all cameras. The circles are the grid points, and the error vectors *magnified by a factor of 50* are shown as "tails." Since the half-meter grid points are 50 cm apart, a tail one grid-point distance long represents a 1 cm error vector. Points with two tails are in the overlap zones covered by two cameras.

Figure 6(b) shows the location errors after applying the error correction and interpolation algorithm. Figure 6(c) superimposes the blend triangles. Camera locations are apparent at the centers of triangle patterns. Notice the lack of tails at the triangle vertices, where the error was zeroed out.

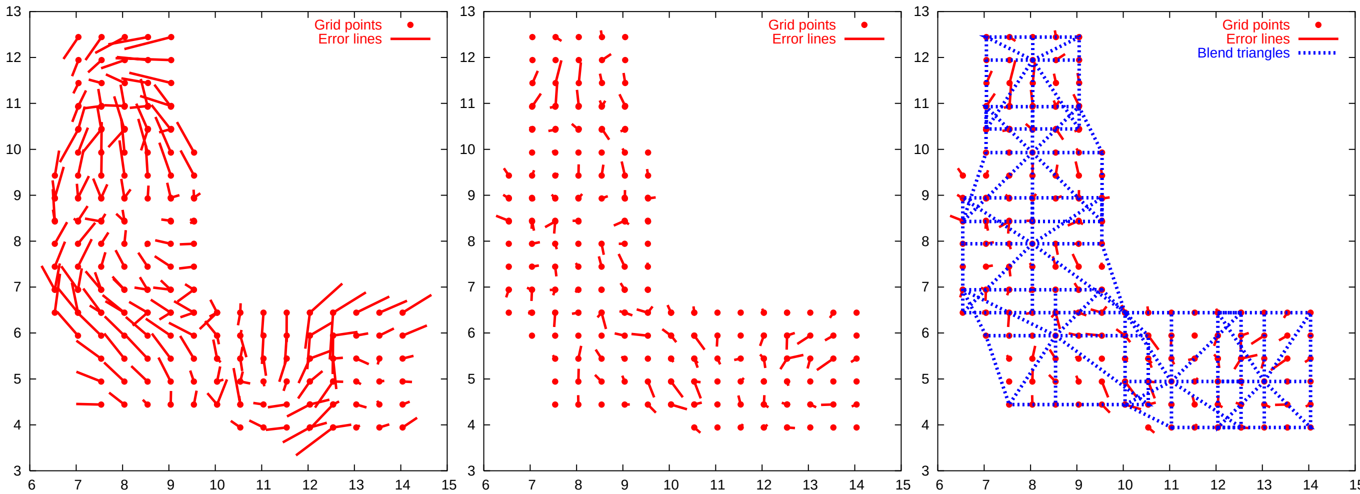
D. Validation

To obtain as much precision as possible, before modifying Mezzanine's dewarping algorithm, we measured out a half-meter grid over our experimental area. This allowed us to calibrate our new algorithm and measure its effectiveness with high precision. Using hardware-store measuring tools and surveying techniques, we set up a grid that is accurate to 2mm.

In Table I are the results of applying these algorithms to a fiducial located by a pin at each of the 211 measured grid points and comparing to the surveyed world coordinates of these points. (Points in the overlap between cameras are gathered twice.) The *original* column contains statistics from the original approximate dewarping function, gathered from only one camera. Data for the *cosine dewarping*, and *cosine dewarping + error interpolation* columns were gathered from all six cameras.

V. ROBOT CONTROL: *robotd*

robotd is responsible for directing robots to their user-specified locations. Users may dispatch robots to any attainable position and orientation within the workspace, and are not required to plan for every obstacle, waypoint, or path intersection. Once new destinations are received via the Emulab event system, the daemon creates a feasible path and guides the robots to their destinations using periodic feedback from the vision system. For simplicity, the paths created by the daemon are comprised of a series or waypoints, connected by line



(a) Cosine dewarped grid points 50 cm apart, with error vectors *magnified 50 times.*

(b) After applying error interpolation.

(c) With error interpolation blend triangles added.

Fig. 6. Comparison of dewarping before and after error interpolation, with error vectors *magnified 50 times.*

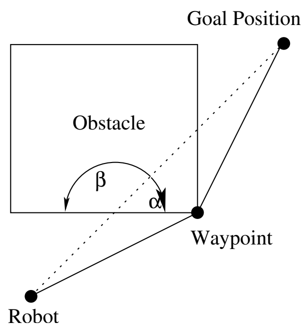


Fig. 7. Example obstructed path and waypoint selection.

segments. Approach to goal points is repeatedly refined based on "ground truth" localization from the vision system; i.e., when the robot's initial (long) move is complete, a maximum of two additional (small) refining moves are made.

A. Robot Motion

Robot motion control is handled by a daemon running on the Stargate called *pilot*. *Pilot* listens for motion commands from the central path planner and then splits them into a series of actions to be executed by the built-in microcontroller on each robot. These actions are handled by the Garcia's built-in motion commands, called primitives. These primitives require only a distance or angular displacement argument, and move the robot using dead reckoning until the goal position has been achieved, or an unexpected obstacle has been detected with the on-board proximity sensors. In either case the robot stops all motion, and alerts the *pilot* application via a callback.

B. Forward Path Planning

Since numerous obstacles exist within the robot workspace, we developed a simple path planner loosely based on established visibility graph methods [3]. Unlike common visibility graph methods, our method does not create a tree of all possible waypoint nodes in the workspace. We use an iterative forward planning algorithm, which simplifies *robotd*.

Robotd reviews all known obstacles to detect any obstacle intersections with the ideal path. Corner points are defined as the vertexes of the nearest obstacle exclusion zone that the ideal path line intersects. We select an intermediate waypoint by computing the angle at which the path intersects the side of the obstacle and choosing the corner point yielding the shallowest angle. For example, in figure 7, angle α is smaller than β. In this case, our planner would select the corner point on the α side of the ideal path.

If the ideal path to the goal is unobstructed by the current obstacle after the robot reaches an intermediate waypoint, the robot will proceed to the goal. Otherwise, we iterate: another intermediate waypoint is generated coincident with the next corner point closest to the goal point.

Robot goal positions are checked for conflicts with known obstacles and workspace boundaries. Longer paths are split into multiple segments of 1500 mm to reduce the possibility of accumulated position errors. The path planner uses an iterative reactive approach, which avoids the need for replanning if the workspace changes over time.

C. Reactive Path Planning

Our robots are capable of detecting obstructions in their path using proximity sensors. In our testbed environment, this can occur due to the temporary presence of a person, another robot, or office debris. When a path is interrupted, the affected robot calls *robotd*, which will supply a new path to negotiate around the obstacle. If the detected obstacle is not found within the list of known static obstacles, a temporary rectangular obstacle similar in size to another robot is created. The robot will then back up a short distance to ensure enough clearance for rotation and then execute the above path planning algorithm. If the robot encounters another unknown obstacle close to a previously discovered one, they are assumed to be the same

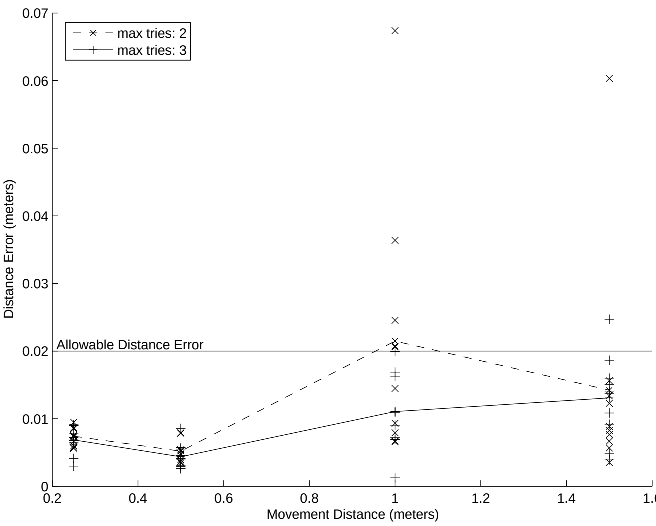


Fig. 8. Robot motion distance error.

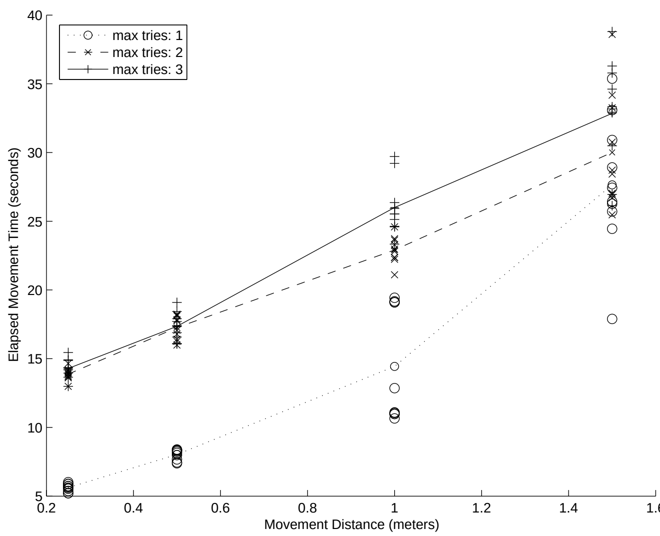


Fig. 9. Robot motion elapsed time for various length movements.

and the two estimates will be merged. This effectively allows *robotd* to determine the actual size of the obstacle and properly plot a path around it.

The combination of the simple per-robot "optimistic" planner with the simple reactive planner has worked well in our environment. However, multi-robot planning will clearly be needed to support a future continuous motion model, when timing is more critical, and a more dense robot deployment.

D. Microbenchmarks

As shown in Figure 8, the final distance error decreases as the number of refinements increases. The default value of maximum tries for each waypoint is set at three. At this setting, we consistently achieve robot positioning within the allowable distance error threshold. With only two tries allowed, robots can still attain final positions within acceptable tolerances, especially when movement lengths are less than one meter.

Figure 9 depicts the total elapsed time for movements of varying lengths. With either one or two waypoint position refinements allowed, a robot can achieve a posture within the allowable distance error, and requires minimal extra time as movement length increases. The elapsed movement time is expected to increase linearly as distance increases, and the bottom plot illustrates that this holds true. Furthermore, the slope of the plots for greater numbers of retries is less, indicating that relative overhead of position refinements decreases as movement length increases.

VI. Experiments

In this section, we describe the results of three experiments using our mobile testbed. These are examples of the testbed's usefulness and also serve as macrobenchmarks of some key metrics of the testbed's performance. The first experiment also demonstrates the network-level irregularity of real life physical environments.

A. Radio Irregularity Map

It is well known that the transmission characteristics of real radios differ substantially from simulation models [2], [13], [20], [28]. Indeed, irregularity of real-world radio transmission is one of the main motivators for our testbed. In this experiment, we generated a map of the radio irregularity in our testbed space as manifested at the network (packet) level. Such a map is useful to our experimenters, and could be used to develop and/or validate more realistic models for simulation.

In parallel, three robots traversed non-overlapping regions of our space, stopping at points on a half-meter grid. At each point, the robot stopped and oriented itself in a reference direction. The attached mote listened for packets for ten seconds. One of the wall-mounted motes, with an antenna at the same height as the robots' antennas (approximately 13cm at that time), transmitted ten packets per second using a modified version of the standard TinyOS `CntToLedsAndRfm` kernel. The receiver logged packets using a modified version of TinyOS's `TransparentBase`. The radios were tuned to 916.4 MHz, and the receiver's power was turned down to approximately -18 dBm (corresponding to a `PA_POW` setting of `0x03` on the mote's ChipCon CC1000 radio). The entire mapping took 20 minutes to complete.

Figure 10 shows a graphical representation of this data. As can be seen in the data, there is much variation in packet reception rate throughout the area. As expected, reception rate does not decrease as a simple (i.e., linear or quadratic) function of distance. However, we also see that reception rate is not a monotonic function of distance; there are areas in the map in which if one travels in a straight line, on a radial away from the sender, reception gets worse, then better, then worse again. There are islands of connectivity in otherwise dead areas, such as around x=8, y=10, and the inverse, such as around x=9.5, y=5.5. Furthermore, in some areas (such as between x=10 and x=12), reception falls off gradually, and in others (such as around y=9), it reaches a steep cliff. This surprising behavior is a fact of life for sensor network deployments. We argue that while running algorithms and protocols under

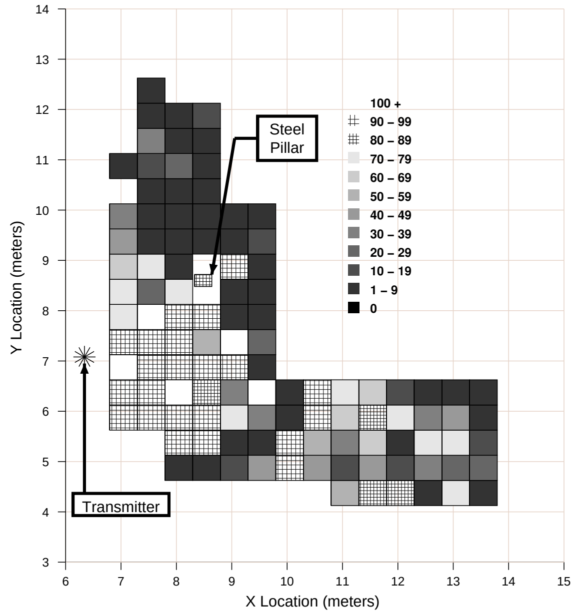


Fig. 10. Packet reception rates over our testbed area, first run.

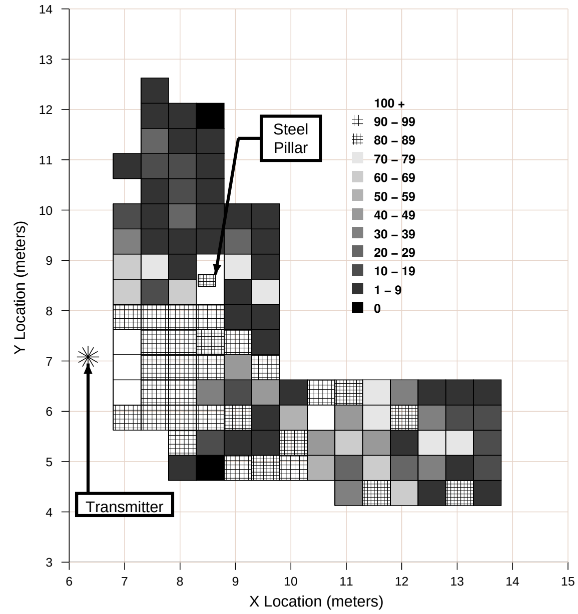


Fig. 11. Packet reception rates over our testbed area, second run.

simulation models, which makes them easy to reason about, has its place, it is necessary to run them in real environments to understand how they will perform in deployment. Indeed, Zhou et al [28] show that a specific type of radio asymmetry, radial asymmetry, can have a substantial effect on routing algorithms. It is far beyond the state of the art for a model to fully capture the effects of building construction, furniture, interferers, etc., in a complicated indoor environment.

We repeated this experiment immediately after the first run had completed, in order to assess how repeatable our findings were. As shown in Figure 11, while the results are not identical, they are close—the contours of the areas of good and poor reception are similar. The overall similarity suggests our methodology is good, while the differences reflect the fact that temporal effects matter. The second run took 18 minutes.

Figure 12 shows the received signal strength indication (RSSI) for packets received in the first run. The RSSI is measured using the CC1000's RSSI line, and is sampled every time a packet is *successfully* received. We can now relate the signal strength to the packet reception rate of earlier figures. Interestingly, we see little correlation. In the topmost area of the figure we see good RSSI (indeed, some of the highest), even though the packet reception rate is low. In contrast, near the lower right corner, we see overall low RSSI values, even though the overall packet reception rate is better.

Emulab's programmability was key to performing this experiment. Since its input language is based on *ns*, which is in turn based on Tcl, it includes familiar programming constructs. Thus, we were able to construct the set of points for each robot's data collection using loops. Emulab's event system coordinated the experiment for us—when a robot reached a data collection point, an event was generated. We used this event to start the 10-second data collection process; thus, we were able

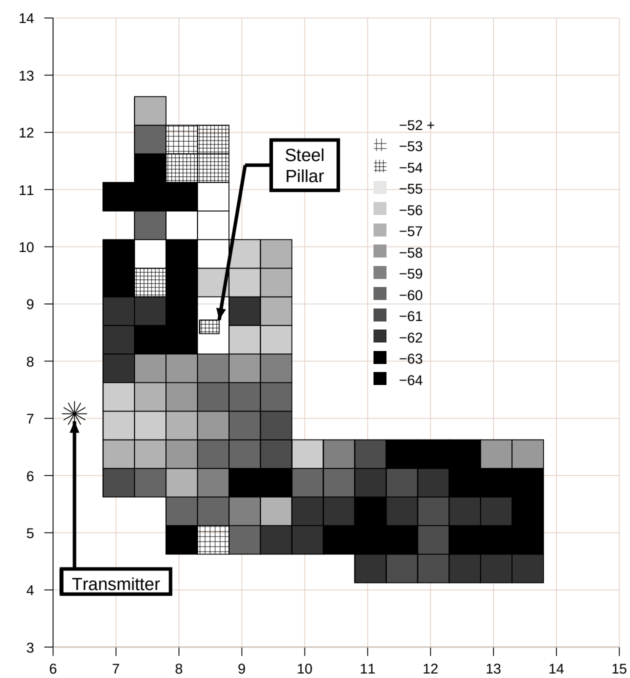


Fig. 12. Received signal strength in dBm for packets over our testbed area, first run. Higher numbers (less negative) indicate stronger signal.

to ensure that the robot was stationary during data collection. The event system allows for synchronization between robot movement, the vision system, and user programs.

One of the advantages of taking these measurements with a programmable robot is that it is easy to examine different areas at different levels of detail. At a different time than the figures made above, we mapped a smaller portion of our area,

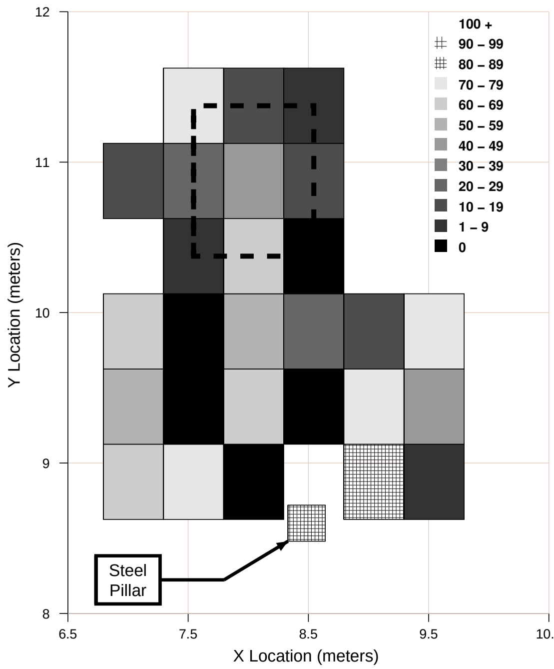


Fig. 13. Packet reception rates over a subset of our testbed area, taken at a different time than Figures 10 and 11.

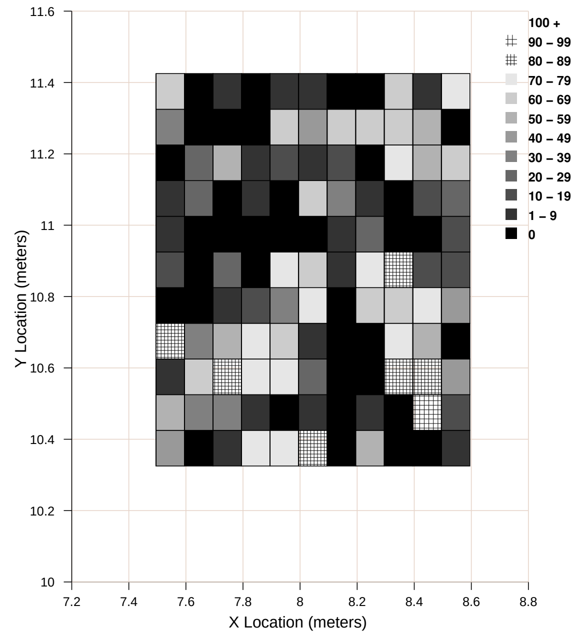


Fig. 14. Packet reception rates over a square meter of our testbed area, with a resolution of 10cm. This area corresponds to the area outlined with a dotted line in Figure 13.

again using a half-meter grid (Figure 13). Here, we see more temporal variation than in the two back-to-back runs: this map does not exactly match the corresponding areas of the previous maps. We then picked an interesting area of the small map, shown outlined with a dotted line, and ran a "zoomed in" experiment over one square meter of it, taking measurements every 10 centimeters over a period of 36 minutes (Figure 14).

We can see from this figure that even small differences in location can make large differences in radio connectivity, and that the topology is far from simple. (The patchwork reception pattern is likely due to RF "ground effect," since during these experiments the mote's antenna was only 10 cm above the floor.) From these results, we can conclude that repeatability is not achievable without precise localization of the robots; in our environment, given its real-world characteristics, clearly repeatability will suffer even with precise localization. But, even if we were to construct a space in which there were no external interferers or movable objects, so that we could work on a highly-detailed indoor radio propagation model, we would not be able to get repeatable results, let alone accurate ones, without precise localization.

Figure 15 shows the breakdown of the time it took to execute the experiment. At the base is the time taken to sample the radio for ten seconds at each grid point. The "long moves" are the half meter traversals from one point to the next. The remaining times are those needed to refine the position to within 1.5cm of the intended destination and reorient the robot. As one can see, from 50% to 60% of the motion-related time is spent in refining the position, which mainly consists of large rotations and small movements. However, position

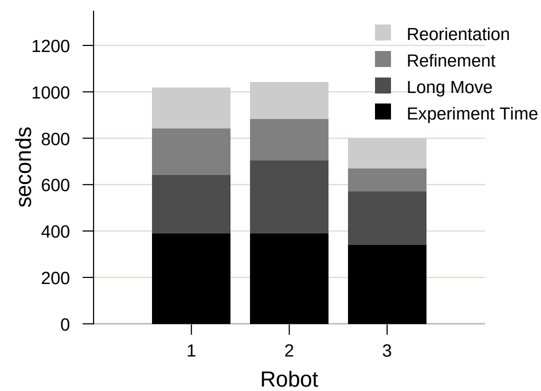


Fig. 15. Breakdown of time spent executing the walk pictured in Figure 11.

refinement only accounts for 12% to 18% of the overall time, and this additional cost is well-worth the increased positioning precision. In the future, we hope to decrease this time by using a continuous motion model that constantly refines the position, requiring fewer changes in orientation.

B. *Sensor-based Acoustic Ranging*

A mobile wireless testbed such as ours invites study of sensor-based ranging and localization. Since our testbed provides the "ground truth" positions of both the mobile and fixed motes, an experimenter can easily verify the quality of a localization system. Coupled with Emulab's automation facilities and real-world (indoor) RF and audio effects, much more complete evaluation is possible than with simulation or manual testing.

We evaluated an acoustic ranging sensor network application from Vanderbilt University [19], available in TinyOS. It uses the standard "time difference of arrival" (TDOA) technique, in which radio and audio signals are simultaneously emitted. Receiving nodes can measure the propagation time of the slow-moving audio signal simply by computing the difference in arrival times of the RF and audio signals. Vanderbilt's software is more advanced than this; complex synchronization and audio frequency filtering are employed to reduce range estimation error. Each mote is loaded with the TinyOS application. Listening motes receive radio packets and hear a succession of chirps from the single sending mote, and can then compute range. Vanderbilt's software uses the same standard Mica sensor boards [7] that we have, with a 4kHz buzzer and a microphone capable of sensing frequencies up to 18kHz. However, it is important to note that this application was specifically meant for outdoor use because of problematic audio echoes in contained indoor settings.

We ran the acoustic ranging application on our fixed mote testbed. Each mote is approximately 20 cm above the floor, and is attached to a wall (sheetrock or standard cubical walls). Since exact positions of the fixed motes are stored in a database, we can easily compare the real ranges with application-estimated ranges. Each of the nine motes with MTS310 sensorboards was loaded with the acoustic ranging application using Emulab's programming facilities. However, the ranging results from all nine motes often exhibited highly erroneous estimates. This was apparently because several of the motes were separated by an L-shaped corner, strengthening the resulting acoustic echoes. Consequently, we re-ran the application on only the 6 motes that were in an area with no obstructing walls that could affect the acoustic and RF transmissions. We ran numerous trials and found a minimum ranging error of 10.9 cm and a median error of 89.0 cm. The median error is approximately a factor of 12 greater than discovered by Vanderbilt [19]. We expect that the vastly increased error is due to indoor audio echo effects.

In summary, our automated and mobile testbed is a valuable platform on which to easily test ranging and localization applications in an indoor environment. When mobility is coupled with evaluation of a ranging or localization application, researchers can quickly test such applications under different wireless conditions by simply moving the robots to areas with different RF conditions.

C. Multihop Sensor Networks

To confirm that we can create interesting multihop networks in our space, we ran a popular TinyOS application on our testbed, `TinyDB`. `TinyDB` presents a database-like interface to sensor readings, and thus can make use of the sensor boards on our nodes. We lowered the power output on the transmitters to force the network into multiple hops. Figure 16 shows the topology created by `TinyDB` for a 16-node network. The thick lines indicate current links between nodes, and the thin dotted lines represent nodes that have re-parented themselves.

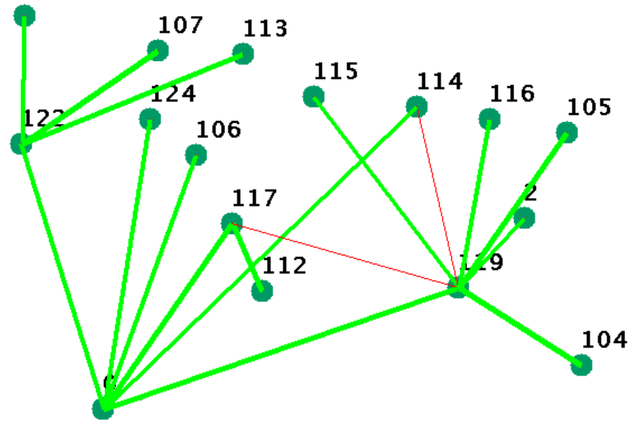


Fig. 16. Multihop network topology created by TinyDB.

`TinyDB`, `Surge`, and many other TinyOS applications require that each mote be programmed with a unique ID for routing purposes, so that data can be associated with the mote that collected it. Emulab aids in this process, automatically programming a unique ID into each mote. The user can also supply a desired ID, so that certain nodes can be designated as base stations, etc.

VII. Limitations, Open Issues, and Future Work

A. Software System and Algorithm Issues

As we discussed earlier in Section V, we expect to replace the current waypoint-based motion model with a more general continuous motion model, allowing more classes of experiments. Our overall software architecture and the localization system's precision and update rate should make this a fairly straightforward, though significant, task.

We plan to provide a way for an experimenter to transparently inject simulated sensor data into the "environment." The user will specify or select a time-varying simulated sensor data "flux field," and our testbed will inject that into the user's application via TinyOS component "shims" we will provide.

When physical testbeds are large, space sharing among multiple experimenters becomes possible and valuable. Emulab already supports space sharing, but provides little help in separating experimenters' RF transmissions or mobile robots. We will pursue an evolutionary path in adding such support.

MoteLab has several useful features we do not, such as a per-experiment MySQL database and a simple reservation system. We may include MoteLab itself into our testbed, as a separate subsystem, but we will at least adopt those features. Similarly, EmStar and Mirage have strengths complementary to ours, and probably can be included without undue disruption to either codebase.

B. Physical Infrastructure

Based on our experience, we plan or contemplate a number of improvements and changes to our testbed physical infrastructure. We will be adding a second 802.11 card on our nodes, so that they can be used for WiFi experiments. In order to provide interesting multi-hop topologies in our space, we will use attenuators. Unlike the experience of the MiNT developers, we have been able to disconnect the internal antenna on PCMCIA wireless cards, making attenuation possible via an external connection. Unfortunately, the 802.11 chipset most popular for wireless research, Atheros, is not available in the form factors provided by the Stargate: PCMCIA and CF.

We may greatly expand the area and scale of our current testbed, by using a much larger, isolated space (90' x 90') we have available, and/or by extending throughout our building's hallways. Should we do the latter, we will need to develop or adopt a different localization system, for it is not practical to install downward-looking video cameras throughout such a large and sparse area. In fact, with an appropriate localization system, our system could be deployed on an outdoor testbed.

We have designed and prototyped a low-cost ($35) power measurement circuit, installed it on a single mote in our fixed testbed, integrated it into the Emulab control and monitoring software, but have not yet fully evaluated it. We plan to add power circuits to all fixed nodes, and investigate options for putting them onto robot nodes. We also intend to look into modifications to the vision system to make it work in lower light, so that the testbed can be used for light-sensor experiments. The most promising options seem to be LEDs or black-and-white fiducials.

VIII. Conclusions

We have described the design and implementation of a robotic-based mobile wireless and sensor network testbed. Designing and building this testbed required us to solve a number of hard problems, particularly with respect to robot localization and movement. Our experience so far shows it to be a promising testbed, valuable for a range of experiments in mobile and wireless networking. Since it is remotely accessible, it can provide the mobile and wireless sensor networking community with a practical complement to simulation.

Acknowledgments

We owe great thanks to Kirk Webb for much help in engineering, management, and operations. We are grateful to Mark Minor for his contributions in a variety of areas relating to robotics. Bill Thompson and Tom Henderson gave us guidance in the computer vision area. As always, Mike Hibler helped in a variety of ways; of special note is the entire day he spent gathering validation data using a cardboard robot body. We are grateful to Eric Eide for feedback on earlier drafts and his editing help, to Dan Gebhardt and Kirk for their ongoing work on the power measurement circuit, to Grant Ayers for contributing substantially to the fixed mote installation, and to Jon Duerig and Kevin Atkinson for helping with editing and the bibliography.

References

[1] Acroname Corp. Garcia robot. http://www.acroname.com/garcia/garcia.html.

[2] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level Measurements from an 802.11b Mesh Network. In *Proceedings of SIGCOMM*, Aug. 2004.

[3] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[4] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Mirage: A Microeconomic Resource Allocation System for SensorNet Testbeds. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, Sydney, Australia, May 2005.

[5] H. S. M. Coxeter. *Introduction to Geometry*. John Wiley & Sons, Inc., 1969.

[6] Crossbow Corp. MIB510 Serial Gateway. http://www.xbow.com/Products/productsdetails.aspx?sid=79.

[7] Crossbow Corp. Mica2 multi-sensor module MTS310. http://www.xbow.com/Products/productsdetails.aspx?sid=7.

[8] Crossbow Corp. Mica2 Series mote. http://www.xbow.com/Products/productsdetails.aspx?sid=72.

[9] Crossbow Corp. Stargate Gateway. http://www.xbow.com/Products/productsdetails.aspx?sid=85.

[10] P. De, A. Raniwala, S. Sharma, and T. Chiueh. MiNT: A Miniaturized Network Testbed for Mobile Wireless Research. In *Proceedings of IEEE INFOCOM*, Mar. 2005.

[11] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. In *Proceedings of the 2004 USENIX Technical Conference*, Boston, MA, June 2004.

[12] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks. In *ACM SenSys*, Nov. 2004.

[13] J. Heidemann, N. Bulusu, and J. Elson. Effects of Detail in Wireless Network Simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, Jan. 2001.

[14] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Feedback-directed Virtualization Techniques for Scalable Network Experimentation. Flux Group Technical Note FTN–2004–02, University of Utah, May 2004. http://www.cs.utah.edu/flux/papers/virt-ftn2004-02.pdf.

[15] Hitachi KP-D20A Camera. http://www.hdal.com/Apps/hitachidenshi/content.jsp?-page=microscope_medical/1_CCD_color/details/KPD20A.html&-path=jsp/hitachidenshi/products/industrial_video_systems/.

[16] J. Ledlie, J. Shneidman, M. Welsh, M. Roussopoulos, and M. Seltzer. Open Problems in Data Collection Networks. In *Proceedings of the 11th ACM SIGOPS European Workshop*, Leuven, Belgium, 2004.

[17] Mezzanine: An Overhead Visual Object Tracker. http://playerstage.sourceforge.net/mezzanine/mezzanine.html.

[18] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, Mar. 2005.

[19] J. Sallai, G. Balogh, M. Maróti, Á. Lédeczi, and B. Kusy. Acoustic Ranging in Resource-Constrained Sensor Networks. In *International Conference on Wireless Networks*, June 2004.

[20] M. Takai, J. Martin, and R. Bagrodia. Effects of Wireless Physical Layer Modeling in Mobile Ad Hoc Networks. In *Proceedings of ACM MobiHoc*, Oct. 2001.

[21] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *ISPN/SPOTS*, Apr. 2005.

[22] B. White, J. Lepreau, and S. Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. In *Proc. HotNets-I*, Princeton, NJ, Oct. 2002.

[23] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002.

[24] WHYNET Scalable Mobile Testbed. http://chenyen.cs.ucla.edu/projects/whynet/.

[25] Wolfram Corp. Barycentric Coordinates. http://mathworld.wolfram.com/BarycentricCoordinates.html.

[26] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *SenSys*, Nov. 2003.

[27] J. Zhao and R. Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *ACM SenSys*, Nov. 2003.

[28] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of Radio Irregularity on Wireless Sensor Networks. In *SenSys*, Nov. 2004.