TrueMobile: A Mobile Robotic Wireless and Sensor Network Testbed

David Johnson *Tim Stack* *Russ Fish* *Dan Flickinger* *Rob Ricci* *Jay Lepreau*

University of Utah

University of Utah Flux Group Technical Note FTN–2005–02

April 8, 2005

Abstract

Simulation has been the dominant research methodology in wireless and sensor networking. When mobility is added, the frequency of real-world experimentation becomes tiny. 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 costs in labor, equipment, and tedium of truly mobile experimental infrastructure in real-world environments typically present an insurmountable barrier to such experimentation.

We describe our experience in creating an initial testbed to lower those barriers. Our system, called "TrueMobile," already deployed for public use, provides the first remotely-accessible mobile wireless and sensor testbed. Robots carry motes and single board computers running Linux through a fixed field of sensor-equipped motes, all running the user's selected software. In real-time, interactively or driven by pre-programmed events, remote users can position the robots, control all the computers and network interfaces, and log data. TrueMobile 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.

We present TrueMobile's design and implementation, evaluate key aspects of its performance, and describe a few experiments demonstrating its generality and power.

1 Introduction

Experiments involving mobile wireless devices are inherently complex, and in general, time-consuming to set up and carry out. Such experiments are also extremely difficult to repeat. People who might want to duplicate published results in 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." The shortcomings of modern simulation do not merely lead to minor inaccuracies in experimental results. Rather, the simplifying assumptions that are built into current wireless simulators lead to profound differences between the behavior of system in simulation and the behavior of the realized system in the real world.

It is apparent that mobile wireless systems must be studied and evaluated through experiments that utilize actual mobile devices. For such experiments to be commonplace, however, the capital costs and human effort required to perform such experiments must be substantially reduced — by an order of magnitude or more. We believe that widespread improvements to the development and evaluation of mobile wireless and sensor networks will require the creation of many testbeds to support this type of research. A testbed for mobile wireless research must contain actual mobile devices, provide means for programming the devices, ensure that motion is performed accurately, and provide ways for collecting a variety 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 environments for research, several such testbeds must eventually exist. It is therefore important to reduce the cost of building and maintaining such a wireless testbed.

TrueMobile's important new features of automation and accurate and precise positioning of RF devices enable the construction and validation of simulation models at all radio and network levels (see section 6). A real-world testbed such

as TrueMobile provides new opportunities for development of wireless simulation models. Developers of such models must first test their initial model in a clean-room environment free from outside influences of any kind. Enviromental complexity can then be added in gradually to improve the effectiveness of the simulation model [30, 28] . For instance, one could first add physical obstacles to RF propagation, and continue by expanding the effects of outside influences of many types.

In this paper we describe our experience in creating an initial testbed for mobile wireless and sensor network research. Our system, called *TrueMobile*, is a prototype 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 commercial off-the-shelf (COTS) equipment. We believe that TrueMobile 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.

The contributions of this paper are as follows. First, we present the TrueMobile system. To our knowledge, TrueMobile is the first remotely accessible testbed for mobile wireless and sensor network research. Second, we show that our testbed model is economical: TrueMobile demonstrates that useful mobile testbeds can be built at modest cost, using COTS hardware and open source software. Third, we detail the novel algorithms that we developed as part of building our testbed from COTS parts. In particular, we describe how TrueMobile ensures accurate robot positioning using medium-cost video camera equipment. Finally, we present results from initial experiments that were carried out in the TrueMobile testbed. These results highlight the automation capabilities of the testbed and the measured real-world effects that are present in the TrueMobile environment.

The current system is deployed for public use, but it represents just the first phase of our deployment. This paper describes both the current state of the testbed and our plans for future enhancements.

The rest of this paper is structured as follows. Following a review of related work in Section 2, we present the TrueMobile system architecture in Section 3. We then detail two issues that are essential for reliable mobile experimentation: accurate location of mobile devices (Section 4), and motion control (Section 5), including validation and microbenchmark results. In the last parts of the paper, we describe initial examples of experimentation on TrueMobile (Section 6), discuss open issues and future work (Section 7), and conclude in Section 8.

2 Related Work

There are a number of related testbed projects, for both fixed and mobile testbeds. Here, we give an overview of the work done by others, and how our work advances the state-of-the-art in mobile and sensor network testbeds.

One major way in which we differ with all of the related projects below is our integration with Emulab [26]; by building on this software platform for running network testbeds, 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, which includes a scriptable "distributed event system," which various parts of the system can generate or react to—in Section 6, we run an experiment which makes use of this feature. Emulab supports multiple device types, including generic PCs, emulated widearea network links, and real 802.11 links. This feature is useful for experimenting with systems involving such a mixture, for example, hierarchical wireless sensor systems [6], including those incorporating nodes across the Internet, such as the "Hourglass" data collection architecture [12]. Emulab is Web-based and script or GUI-driven, but also exports an XML-RPC interface so that all interaction can be purely programmatic. Finally, Emulab can reliably handle experiments of very large scale, which they report was a major challenge [9].

The MiNT testbed [20] at SUNY Stony Brook is a miniaturized 802.11 testbed. Their focus is on reducing the area required to run a multihop 802.11 testbed, and on integrating ns simulation with emulation. They achieve mobility through the use of antennas mounted on Lego MindStorm robots, tethered to a PC where the applications are run. 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 wires, and provide accurate movement and "ground-truth" location of the robots, whereas MiNT does not address positioning accuracy.

The ORBIT testbed [13, 16, 17] at Rutgers uses a combination of 802.11 and cellular telephone (3G) equipment. Some of this equipment is carried by campus buses on a fixed route, providing a limited mobility environment in an outdoor setting. Indoors, they use a large (currently 64; planned 400-node) grid to emulate mobility—user code is run on PCs, which are dynamically bound to radio nodes. Thus, by changing the binding of a PC to a new radio node, the PC appears to move, though only discreet hops, not true mobility, are possible. In contrast, we focus on true mobility, and target sensor networks as well as 802.11.

Harvard's MoteLab [25] and UCLA's EmStar [5, 7] both support fixed sensor network testbeds. MoteLab is a Web-accessible software system for general-purpose sensor network experimentation, and has been installed at MIT and

Berkeley as well as Harvard. Like many testbeds, including TrueMobile, 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 concentrates 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.

Intel's Mirage testbed [2] 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.

The WHYNET [27] project eventually intends to integrate a large number of types of wireless devices, but so far has not addressed remote access. The ExScal project at Ohio State is investigating static sensor networks up to thousands of nodes, but remote access is also not a priority. The SCADDS project at USC/ISI has a static testbed of 30 PCs with 418MHz radio deployed in an office building.

3 System Overview and Design

TrueMobile is based on Emulab [26], a testbed management framework from the University of Utah. Emulab is designed to provide consistent and seamless access to a variety of experimental environments. Emulab 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 scripts 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.

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 TrueMobile, and its relationship to Emulab, is shown in Figure 1.

3.1 Software Architecture

Our testbed software provides experimenters the capability to dynamically position robots and use them to conduct experiments. To this end, we have extended core Emulab functionality, providing 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. This backend functions primarily as a data broker and translator of user requests to the robot system. The robot-specific components include a robot control component, called *robotd*, which maneuvers robots to user-specified positions, thus freeing the experimenter from the necessity of specifying the details of each robot's path to a final position. A computer vision-based localization component, called *visiond*, processes data from an overhead network of video-cameras and provides robot position and orientation data to both the backend and to *robotd*. This allows users to track robot location changes in real-time and ensures robots reach their destinations.

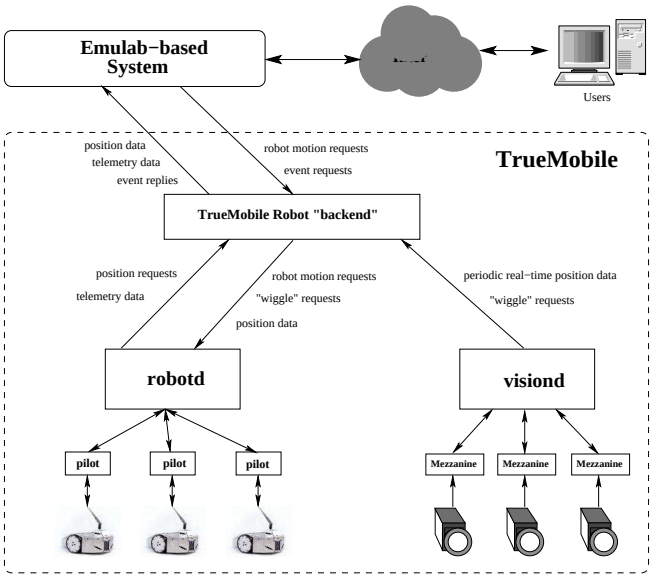


Figure 1: TrueMobile software architecture

Component structure and dataflow is shown in Figure 1, and is 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 *robotd*. *robotd* then queries the backend to ensure that it has the latest location data for the robot in question. Finally, *robotd* transforms and issues the position request to the *pilot* daemon, which runs on each robot and actually moves the robot to the new position. Since internal robot wheel odometry is a poor estimate of actual distance traveled, *robotd* re-queries the backend after the initial move is completed to discover how much correction is necessary. This process is repeated until the robot reaches a position within a fixed distance from the requested position, or until a retry threshold has been reached.

Robot Localization. Tracking and identification of the

robots is handled by a vision-based tracking system, using six ceiling-mounted video cameras aimed straight down. As described later in section 4, we improved an open source object tracking software package to transform the overhead camera video into X,Y coordinates and orientation for any 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 as 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. Plotting a path for the robots to reach their user-specified positions is performed centrally by the *robotd* and then individual commands are sent down to the *pilot* daemon. The current *pilot* daemon uses the API provided for Acroname Garcia [4] robots. It makes use of the built-in motion commands and data structures. The low-level robot control interface is well abstracted, allowing development to concentrate mainly on path planning and movement. The control aspects required to move specific distances and angles are handled by the tools provided by Acroname. Additional functions are also handled by the API, such as battery levels, monitoring sensor values, adjusting sensor sensitivities and thresholds, and adjusting robot parameters. Path planning is described in section 5.

Emulab Integration. We have extended the Emulab interface to allow experiments with mobile robot nodes. Experimenters may select several robots, perhaps in tandem with other node types to collect or process data, customize various Emulab experimental settings, and direct Emulab to instantiate an experiment. Experimenters may customize a set of robot positions, software to load onto nodes (i.e., a custom TinyOS kernel for motes, or data processing software for a master wireless sensor network node). Dynamic robot positioning is achieved using the existing Emulab event system, which has been extended with an event agent running on the backend, translating requests from the user to motion commands to *robotd*. For example, Figure 2 shows an an excerpt of the NS code used in one of our experiments to walk a robot around an area in half meter increments and log data received on the mote. We have also developed several new user interfaces, including Java applets to dynamically position robots 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.

```
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($lpc) set Y_] + $y]
        } else {
            set newx [expr $XSTART + $WIDTH - $x]
            set newy [expr [$walker($lpc) set Y_] + $y]
        }

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

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

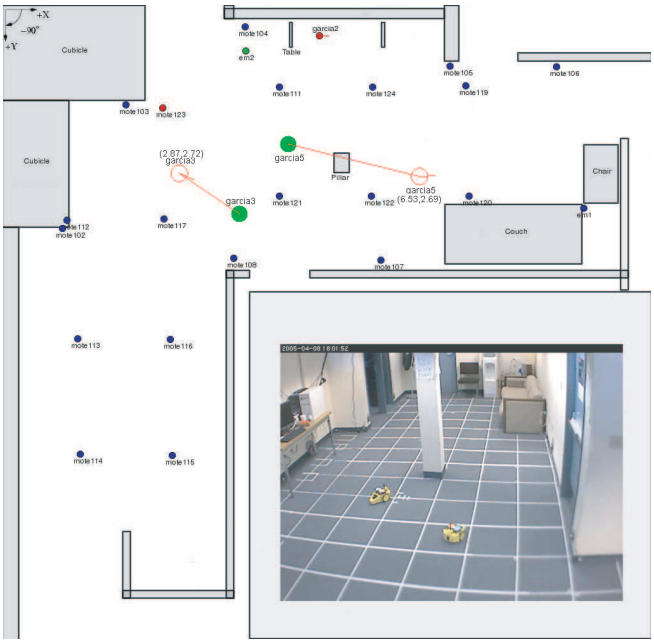


Figure 3: The TrueMobile robot positioning interface allows specifying robot movement with click and drag operations. Green dots are garcias; red vectors indicate desired motion; red circles show desired destination; blue dots are static mote locations; obstacles are in grey. A snapshot of the webcam view of the robot arena is superimposed onto the lower right corner of the interface view.

3.2 Hardware Resources

Space. As shown in Figure 3 the mobile testbed is currently deployed in an L-shaped area of 60 m^2, height about 2.5 meters, with five robots that can be positioned anywhere in that area. Overlooking this area are six cameras used by the robot

tracking system and two webcams that give live feedback to the users of the testbed.

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; however, we currently lack the infrastructure for recharging the robot's batteries without human intervention.

Robots. We currently have five operational Acroname Garcia robots, These robots were chosen as a platform for their size, cost, ease of use, and performance characteristics. The Garcia robot is a commercial robot platform, offering multiple configurations for a reasonable cost ($1100). The use of a commercial platform allows the wireless testbed to be developed without requiring the engineering considerations inherent to robot design to be solved in-house. The use of differentially steered two wheeled robots simplifies the kinematics and control model requirement. Most importantly, because our testbed is based on a readily available commercial robot platform, other teams may 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, although we found the internal robot controller may stall above 0.4m/s, so we currently use that as our maximum speed. Six infrared 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 [21] small computer system, to which we attach a 900MHz Mica2 mote. The Stargate serves as a platform for control of the robot by the testbed and interaction with the mote by the experimenter. To this we attach an 802.11b wifi card that acts as a separate "control network", connecting the robot to the main testbed and the Internet.

Static Motes. The fixed motes are arranged on the ceiling in a roughly 2-meter grid and near the floor on the walls. All of the fixed motes are attached to serial programming boards (model MIB500CA [29]) to allow for programming and communication. The near-floor motes also feature a full multi-sensor board (model MTS310) 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, since the TrueMobile software is derived from the Emulab software, other sites who install TrueMobile to manage their own mobile testbeds could choose to add cluster PC nodes. Experimenters could then leverage those PCs as processing stations for WSN experiments, or use them together with fixed or mobile wireless and sensor nodes to create highly-diverse computer networks.

4 Robot Localization: *visiond*

For accurate and repeatable experiments, TrueMobile must guarantee that all mobile antennae and RF devices are at their specified positions and orientations, within a small tolerance. Accurate localization is also important for robot motion, described in Section 5. 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.

When we started to develop TrueMobile, it became apparent almost immediately that our robots' on-board odometry was insufficient to localize the robots with sufficient accuracy. We therefore developed a computer vision-based robot 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. Our system tracks robots with an accuracy of 1 cm worst-case absolute error, and 0.34 cm RMS absolute error.

To obtain high-precision data while limiting hardware costs, we made a number of engineering trade-offs. 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 models of camera geometry we employed, and increased precision. Third, all robots are marked with the same, simple fiducial. This simplifies the object recognition algorithms and lowers processing time per image.

These simplifications also introduce several complications. First, mounting the video cameras perpendicular to the plane of robot motion increases the total number of cameras necessary to cover our area, relative to a trinocular videocamera-based localization system [15]. Second, use of the same fiducial on each robot prevents unique identification of each robot; thus, we must maintain robot identity at a different layer. Third, since our current vision software requires color fiducials, light conditions must be carefully controlled and the

vision software must be calibrated to handle light variability. Fourth, cost minimization requires the largest possible per-camera viewspace. Ceiling height limits distance from the ground plane in which we can place the video cameras, so we have installed very wide-angle zoom lenses on each camera with an over 90 degree field of view.

Our image processing algorithms compensate for the resulting image distortion.

4.1 Hardware

We use video cameras and lenses that combine to produce high-precision localization, yet are not prohibitively expensive. The most important camera features were (1) resolution and (2) control modes for light and color. However, digital cameras with resolutions higher than 640×480 pixels all exceeded our cost constraints. We evaluated standard analog security cameras, and found that the analog resolution produced is too low to extract sharp fiducial outlines. Standard security cameras also lack manual controls for light and color settings, which are needed in our environment to combat the effects of lighting variability. After extensive evaluation, we chose to use the Hitachi KP-D20A analog CCD camera [10], which provides sufficient analog resolution and good, manual control of light and color settings. The camera cost was $460.

To cover the testbed arena with as few cameras as posisble, we needed to use wide-angle lenses on our cameras. Wide-angle lenses produce barrel distortion, which can be partially accounted for in software, but which decreases our system's precision. Low-distortion wide-angle lenses can cost many thousands of dollars, so we chose to use inexpensive lenses and correct for distortion in software using better camera geometry models and interpolative error correction. We are using Computar 2.8–6.0 mm varifocal lenses set at focal lengths of 2.8 mm, each costing approximately $60.

The total per-camera cost of our current system is approximately $750. We expect to reduce that to $600–$700 in a large-scale implementation by covering more space with fewer cameras and improving our distortion correction.

4.2 Localization Software

To keep our costs low, we chose to use Mezzanine [14], 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. For space reasons, we focus on the first two of these steps below.

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 each robot by placing a different fiducial on it. We instead obtain unique identification by commanding and detecting movement patterns for each robot, and thereafter maintain an association between a robot's initial 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).

4.3 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*. Specifically, it is a single global biquadratic polynomial with two bicubic terms and eight coeffients:

$$c_1 + c_2x + c_3y + c_4x^2 + c_5xy + c_6y^2 + c_7x^2y + c_8y^2x$$

This function requires a set of known (image,world) coordinates, corresponding to visible features such as tape marks in the image identifying grid lines. While developing TrueMobile, we deployed visible marks on the floor and established the needed mappings using tools that came with Mezzanine.

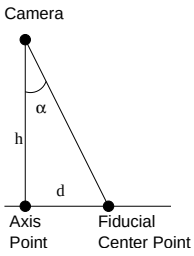
We observed two important problems. First, the function was a poor fit for lens distortion, so it was necessary to add more control points to improve the fit. Second, the grid shape would tilt and bend globally when any control point was moved, so it never got really close anywhere, and was extremely 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.

4.4 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 have obtained an accuracy of 1 cm worst-case absolute error, and 0.34 cm RMS absolute error. This section summarizes the new algortithm; details about camera setup and calibaration are omitted for space.

In development of our new algorithm, we wanted to use a simple geometric model that accurately describes barrel distortion. One suggested method of handling barrel distortion radiating symmetrically from the camera's optical center is to use a 6th-degree polynomial, with coefficients obtained from a least-squares fit on measured data points [11].

$$\kappa_1 + \kappa_2 r^2 + \kappa_3 r^4 + \kappa_4 r^6$$



where

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

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

and w is the "warp factor."

Figure 4: Cosine dewarping

This resembles the Taylor's polynomial expansion of cosine, truncated to four terms:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}$$

A true cosine is transcendental, with an infinite number of terms as a polynomial.

Computer vision experts we consulted suggested that the most significant term of radial lens distortion might be accurately modeled by a cosine function [23]. We were prepared to continue modeling and correcting assymmetries radiatiating from the optical axis, or moving circularly around it. Contrary to our expectations, these $60 lenses conform closely to the simple cosine model, for our accuracy requirement.

Our new dewarping algorithm is based on correcting cosine radial lens distortion. Figure 4 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. Other parameters include the height of the camera's focal point above the plane of robot motion (corrected for the height of the fiducial location on the robot), and the position of the optical axis.

An interesting problem is that, when dewarping, we are converting from d_{image} to d_{world}, but to get the angle a exactly we need to already know the dewarped world coordinates. In our algorithm we iterate, using each d_{world} approximation to calculate a new angle α and hence a new d_{world}, converging in fewer than eight iterations.

Cosine dewarping is still a global model, using model parameters which are a compromise across the whole image. We know the surveyed center, edge and corner point locations to 1-2 mm accuracy, acquire their pixel coordinates with Mezzanine, and use them as calibration points. We remove pixel jitter by averaging 30 frames of pixel positions.

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 which span the space between the measured points by way of barycentric coordinates [3, 24]

Only 9 calibration points are used in this scheme, which is small enough to be handled automatically from multiple fiducials by Mezzcal. 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 chose this one for its accuracy and simplicity of algorithm.

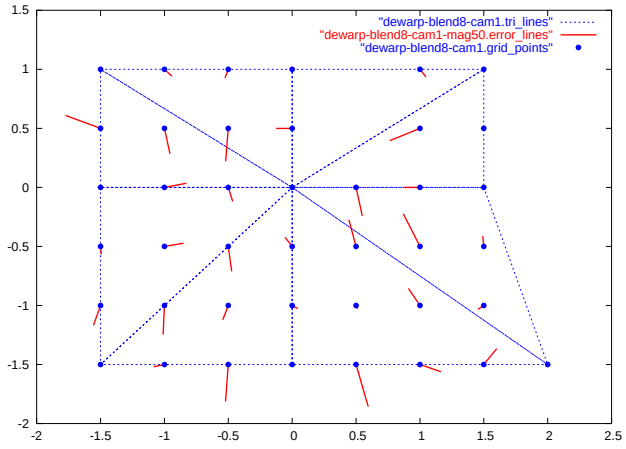


Figure 5: Dewarped grid points, error vectors *magnified by a factor of 50*, and interpolation triangles.

Figure 5 shows a measurement of the dewarped grid points and remaining error from one camera, with interpolation triangles. The circles are the grid points, and the error vector *magnified by a factor of 50* is shown as "tails". Notice the lack of tails at the triangle vertices, where the error was zero'ed out. The gap in the upper middle is the pillar, with the door area at the upper right. Table 1 gives overall error statistics.

4.5 Validation

To obtain as much precision as possible, before modifying Mezzanine's dewarping algorithm, we measured out a half-meter grid in 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 1 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.

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 1: Location error measurements

5 Robot control: *robotd*

As mentioned in Section 3, TrueMobile currently only supports a motion model consisting of a series of user-specified waypoints. Using TrueMobile's event system, experimenters can approximate continuous motion by linking a series of straight line moves. However, since the robot currently decelerates at each waypoint (due to a limitation in the manufacturer's open-source code, that we plan to fix), the motion will be "jerky," and cannot be tightly linked to absolute time.

Our next major improvement will be to evolve into a true continuous motion model. Since our localization system is so accurate and precise, and provides position information at 30Hz, this should not be hard once we have enhanced the robot's key primitives to avoid deceleration.

The Robot Control daemon is responsible for directing robots to their user-specified locations. Users may position robots at any attainable position 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 plots a course and guides the robots to their destinations using periodic feedback from the vision system. For initial simplicity, the courses plotted by the daemon are comprised of a series or waypoints, connected by line 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 more (small) refining moves are made.

5.1 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 breaks them up into a series of behaviors to be executed by the builtin microcontroller on each robot. Incoming commands are directions for a robot to move to a new position relative to its current location. Robots are ignorant of their own global location, instead relying on wheel odometry to navigate during movements.

To achieve a new posture, a robot executes two motions in succession: a pivot followed by a straight move forward or backward. We make several simple optimizations to avoid excessive rotation. Once at the goal point, another pivot is executed to achieve the requested final orientation.

These maneuvers are handled by the Garcia's builtin motion commands, called primitives. These primitives require only a distance or angular displacement argument, and move the robot until the robot has achieved its goal or detected an unexpected obstacle with its on-board sensors. In either case the robot stops all motion, and alerts the pilot application via a callback.

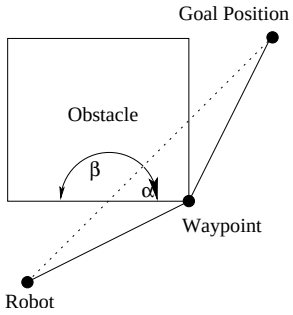


Figure 6: Example obstructed path and waypoint selection.

The robots are only capable of navigating to waypoints via pivots and line segments. This initial motion model was chosen for its simplicity and low development time. Another advantage, however, is that even the simplest robotic platforms can support this motion model.

5.2 Forward Path Planning

Since numerous obstacles such as a pillar and furniture exist within the robot workspace, we developed a simple path planner. It only handles robots individually, without regard to their potential interaction. However, TrueMobile will execute robot moves simultaneously, should the user request it.

It first reviews all known obstacles to detect any obstacle intersections with the initial path. The set of obstacles that the path intersects are merged into a single rectangle and one of the rectangle's corner points is chosen as a waypoint. We select the corner point by computing the angle at which the path intersects the side of the obstacle and choosing the corner yielding the shallowest angle. For example, in figure 6 the alpha angle is smaller than the beta, so the planner selects the corner under that side of the path. If a goal is directly across from an obstacle (e.g. an angle between 60 and 90 degrees), we choose the corner closest to the robot, to avoid excessive movement.

After the robot reaches the first corner point, if the path to the goal is unobstructed by the current obstacle, the robot will proceed to the goal. If the new path to the goal point is still obstructed by the current obstacle, another intermediate waypoint is generated coincident with the next corner point closest to the goal point. A *next* corner point is defined as either one of two corner points reachable by traveling around the perimeter of the obstacle exclusion zone.

Robot goal positions are checked for conflicts with known obstacles and workspace boundaries. Longer paths are split into multiple segments of 1.5 m to reduce the possibility of accumulated position errors. The path planner computes only

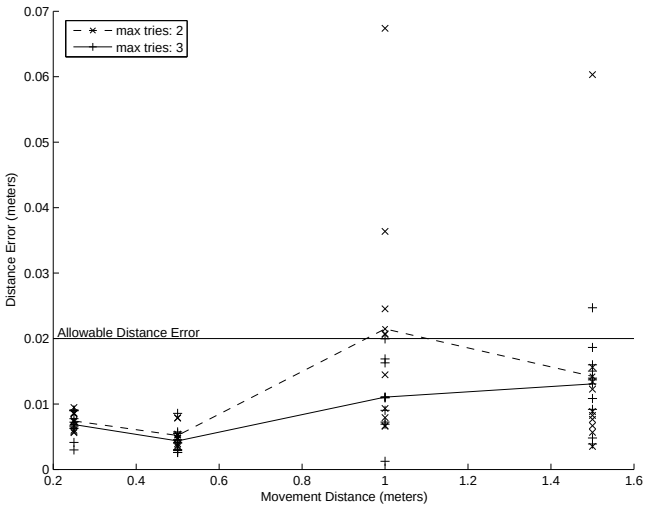


Figure 7: Robot motion distance error

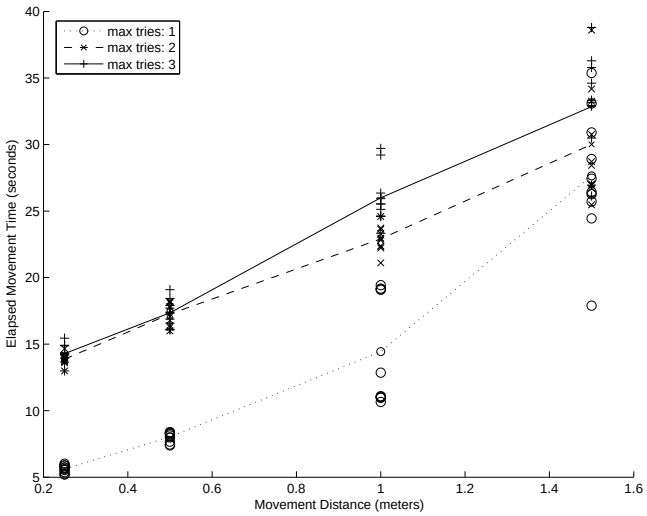


Figure 8: Robot motion elapsed time for various length movements

the next waypoint in the overall path, which avoids the need for replanning if transient obstacles arise.

5.3 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, or an error in TrueMobile's model (map) of the environment. 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. In the case that the obstacle is larger than the approximated size, the robot will continue to detect it and expand its size until the obstacle has been successfully negotiated.

The combination of the simple per-robot "optimistic" planner with the simple reactive planner has worked well in our environment, so far. 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.

5.4 Microbenchmarks

As shown in Figure 7, the final waypoint 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 considering movement lengths of less than one meter.

Figure 8 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 requiring 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.

6 Experiments

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

6.1 Radio Irregularity Map

It is well known that the transmission characteristics of real radios differ substantially from simulation models [31, 22, 8, 1]. 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 sec-

onds. One of the wall-mounted motes, with an antenna at the same height as the robots' antennas, 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 9 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 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 [31] 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 10, 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 to complete.

Figure 11 show the received signal strength (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. Thus, this figure gives us an idea of the correlation between signal strength and packet reception rate. Interestingly, we see little correlation. In the top left area of the figure, we see good RSSI (indeed, some of the highest in the figure), even though the packet reception rate is low. In contrast, near the lower right corner, we overall lower RSSI values, even though the overall packet reception rate is better. This is a phenomenon that warrants further study.

Emulab's programmability was key to our ability to perform 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 to ensure that the robot was stationary during data collection. This allows for synchronization between robot movement, the vision system, and user programs.

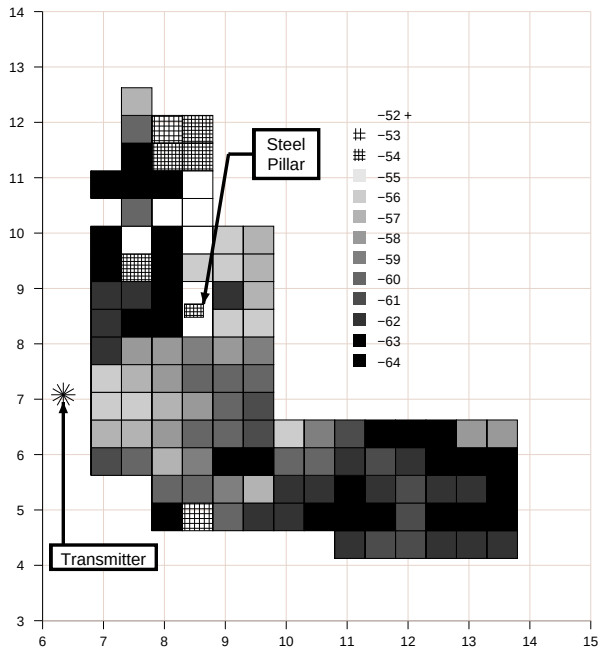


Figure 11: Received signal strength in dBm for packets over our testbed area, first run. Higher numbers (less negative) indicate stronger signal.

One of the advantages of taking these measurements with a programmable robot is that it is easy to examine different areas in different levels of detail. At a different time than the figures made above, we mapped out a smaller portion of our area, again using a half-meter grid. This is shown in Figure 12. Here, we see more temporal variation than we did with the two back-to-back runs: this map does not match exactly with 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, shown in Figure 13, over one square meter of it, taking measurements every 10 centimeters over a period of 36 minutes.

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. From this, we can con-

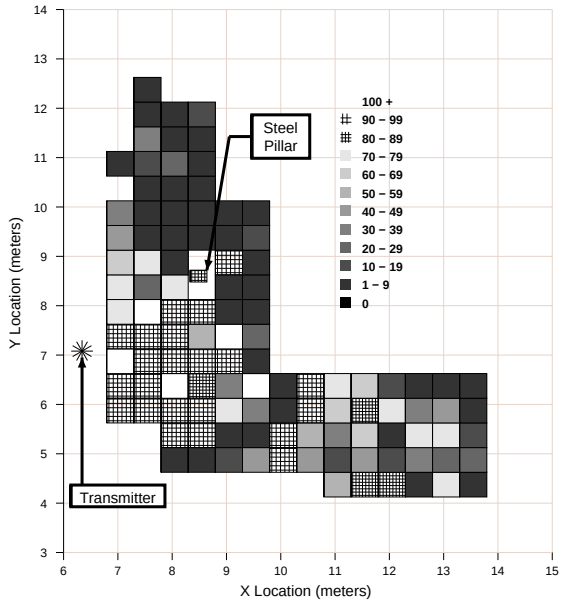


Figure 9: Packet reception rates over our testbed area, first run.

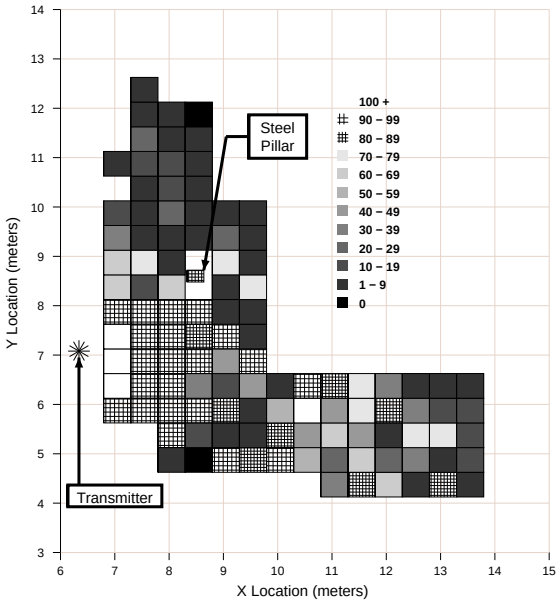


Figure 10: Packet reception rates over our testbed area, second run.

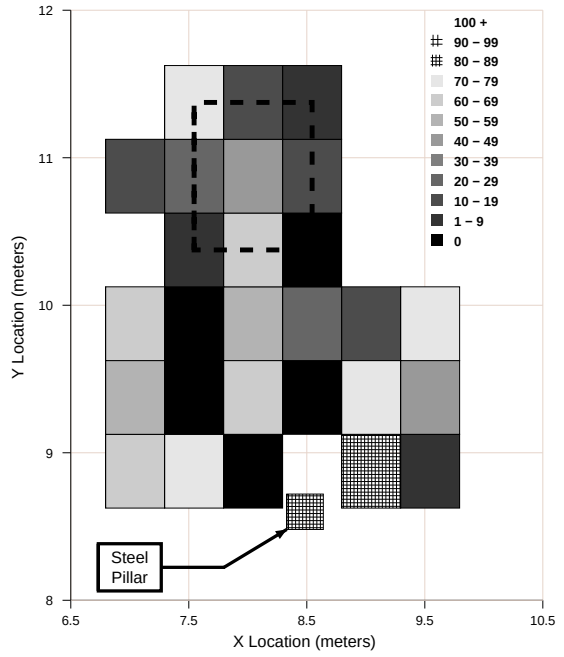


Figure 12: Packet reception rates over a subset of our testbed area, taken at a different time than Figures 9 and 10

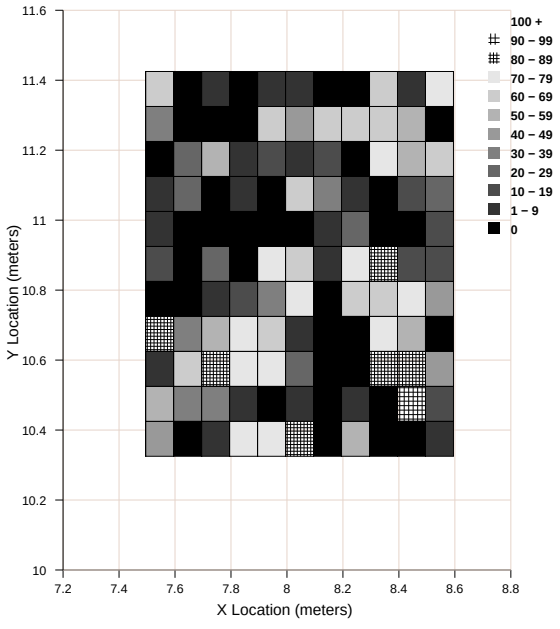


Figure 13: Packet reception rates over a square meter of our testbed area, with a resolution of 10cm. This area corresponds to the area outlined with a dotted line in Figure 12

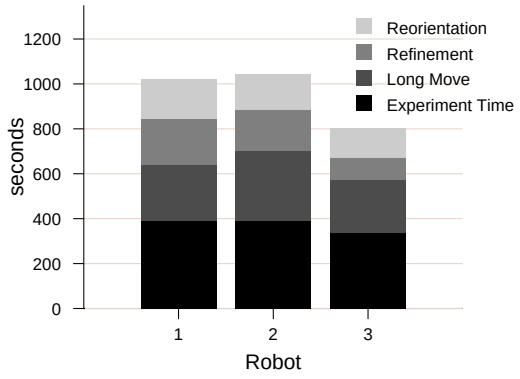


Figure 14: Breakdown of time spent executing the walk pictured in Figure 10

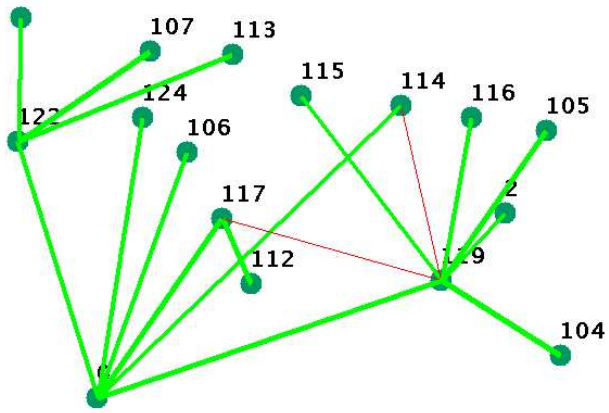


Figure 15: Multihop network topology created by TinyDB.

clude 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, say, 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 14 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 you 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 refinement only accounts for 22% to 29% 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.

6.2 Sensor-based Localization

A mobile wireless testbed such as TrueMobile invites study of sensor-based ranging and localization. Since TrueMobile provides ground truth of robot positions through *visiond*, and also the locations of all static motes, an experimenter can easily verify performance of a localization protocol. Coupled with Emulab's automation facilities, and real-world RF effects, much more complete algorithmic evaluation is possible.

We evaluated a simple acoustic ranging and localization sensor network application from Vanderbilt University [18] (this software also is available in TinyOS). The main idea of acoustic ranging is for one node to broadcast a radio packet, and as soon as the transmission finishes, generate noise from a "buzzer" sensor onboard. A listening mote can then compute difference in travel time for the packet and the generated sound. Vanderbilt's software is more advanced than this; complex synchronization and audio frequency filtering are employed in the to reduce range estimation error. Each mote for which one wishes to learn relative range is loaded with a TinyOS application. Listening motes receive radio packets and hear a succession of chirps from the single sending mote, and can then compute range. This application uses standard Mica sensor boards, with a 4KHz buzzer and microphone capable of hearing frequencies up to 18KHz [19].

This application was meant specifically for outdoor use because of problematic audio echoes resulting from contained indoor settings. After running on TrueMobile, we observed that range was overestimated by approximately 30cm. This figure is a factor of three higher than found by the Vanderbilt researchers (-8.18cm) [18]. The difference may be explained by indoor audio multipath effects and perhaps by the additioans of real-world, indoor radio propagation delay. Finally, TrueMobile is clearly a valuable platform on which to test ranging and localization applications in a real-world environment.

6.3 Multihop Sensor Networks

In order to confirm that we can run multihop interesting 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 turned the power output on the transmitters down in order to force the network into more than one hop. Figure 15 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 in the topology.

`TinyDB`, `Surge`, and many other TinyOS applications require that each mote be programmed with a unique ID for routing purposes, or 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.

7 Limitations, Open Issues, and Future Work

Software System and Algorithm Issues

As we discussed earlier in Section 5, the current waypoint-based motion model should and will be replaced with a more general continuous motion model, allowing more classes of experiments. Our overall software architecture, localization system's precision and update rate, makes this a fairly straightforward, though involved, 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 TrueMobile will inject that into the user's application through TinyOS component "shims" we will provide.

When physical testbeds are large enough, space sharing among multiple experimenters becomes possible and valuable. Emulab already supports space sharing, but provides little help to separate experimenters' RF transmissions or mobile robots. We expect to 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 hope to include MoteLab itself into TrueMobile, 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.

Physical Infrastructure

Based on our experience, we plan or contemplate a number of improvements and changes to our testbed physical infrastructure. We will raise the antennas on our robots, using dowels or wands, to waist-level, so that they more closely approximate human-carried nodes. We will put power meters on all fixed nodes, and investigate options for putting them onto robot nodes. We have low-cost ($35) circuit prototyped, but have not fully evaluated it yet. Custom hardware (cables, mounting board, etc.) will be required to mount sensors on the robot motes. We are in the process of spec'ing such hardware. We are also planning to look into potential modifications to our vision system to make it work in lower light, so that the testbed can be used for light-sensor experiments. We think the most promising options are to use black-and-white fiducials, or to replace the fiducials with LEDs. We will be adding a second 802.11 card on our nodes, so that they can be used for WiFi experiments. In order to make this work in our area, we will most likely use attenuators like the MiNT testbed, but lower left. We expect to greatly expand the area of our current testbed, either by procuring a larger, separate room, 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.

8 Conclusions

We have described our experience in creating a mobile sensor network testbed. Building this testbed required us to solve a number of hard and interesting problems, particularly with respect to robot localization and movement. Our experience so far shows it to be a promising testbed, useful for a range of sensor network experiments. By making it available to the public, we provide the sensor networking community with a more realistic alternative to simulation.

References

[1] 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.

[2] 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, 2005. IEEE.

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

[4] Garcia Robots from Acroname, Inc. http://www.acroname.com/garcia/garcia.html.

[5] 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, 2004. USENIX.

[6] L. Girod, T. Stathopoulos, N. Ramanathan, and J. Elson. A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks. In *SenSys'04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004. ACM.

[7] 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 *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004. ACM.

[8] 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.

[9] 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. Technical Report FTN–2004–02, University of Utah Flux Group Technical Note, May 2004. http://www.cs.utah.edu/flux/papers/virt-ftn2004-02.pdf.

[10] 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/.

[11] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision*. McGraw-Hill Inc, 1995.

[12] 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. ACM.

[13] J. Lei, R. Yates, L. Greenstein, and H. Liu. Wireless Link SNR Mapping Onto An Indoor Testbed. In *Proceedings of the IEEE Tridentcom*, 2005.

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

[15] J. Mulligan, V. Isler, and K. Daniilidis. Trinocular Stereo: A New Algorithm and its Evaluation. *International Journal for Computer Vision*, 2002.

[16] M. Ott, I. Seskar, R. Siracusa, and M. Singh. ORBIT Testbed Software Architecture: Supporting Experiments as a Service. In *Proceedings of IEEE Tridentcom*, 2005.

[17] 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 and Networking Conference*, 2005.

[18] J. Sallai, G. Balogh, M. Maróti, Á. Lédeczi, and B. Kusy. Acoustic Ranging in Resource-Constrained Sensor Networks. In *International Conference on Wireless Networks*, pages 467–, 2004.

[19] Mica2 Multi-Sensor Module MTS310. http://www.xbow.com/Products/productsdetails.aspx?sid=7.

[20] P. D. A. R. S. Sharma and T. cker Chiueh. MiNT: A Miniatuirized Network Testbed for Mobile Wireless Research. In *Proceedings of IEEE Infocom*, Mar. 2005.

[21] Stargate Gateways from Crossbow. http://www.xbow.com/Products/-productsdetails.aspx?sid=85.

[22] 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.

[23] B. Thompson and T. Henderson. Personal communication, Mar. 2005.

[24] Barycentric Coordinates. http://mathworld.wolfram.com/BarycentricCoordinates.html.

[25] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed.

[26] 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.

[27] Whynet Scalable Mobile Testbed. http://chenyen.cs.ucla.edu/projects/whynet/.

[28] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *SenSys'03: Proceedings of the First International Conference on Embedded Networked Sensor Systems*, pages 14–27, Los Angeles, CA, 2003. ACM.

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

[30] J. Zhao and R. Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *SenSys'03: Proceedings of the First International Conference on Embedded Networked Sensor Systems*, pages 1–13, Los Angeles, CA, 2003. ACM.

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