SAND2009-7909

Unlimited Release SANDIA REPORT

SAND2009-7909
Unlimited Release
Printed November 2009

Autonomous Intelligent Assembly Systems LDRD Final Report

Robert J. Anderson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND 2009-7909
Unlimited Release
Printed April 2013

Autonomous Intelligent Assembly Systems LDRD 105746 Final Report

Robert J. Anderson
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1125

Abstract

This report documents a three-year to develop technology that enables mobile robots to perform autonomous assembly tasks in unstructured outdoor environments. This is a multi-tier problem that requires an integration of a large number of different software technologies including: command and control, estimation and localization, distributed communications, object recognition, pose estimation, real-time scanning, and scene interpretation. Although ultimately unsuccessful in achieving a target brick stacking task autonomously, numerous important component technologies were nevertheless developed. Such technologies include: a patent-pending polygon snake algorithm for robust feature tracking, a color grid algorithm for uniquely identification and calibration, a command and control framework for abstracting robot commands, a scanning

capability that utilizes a compact robot portable scanner, and more. This report describes this project and these developed technologies.

Acknowledgements

This author wishes to thank the following Sandians for their invaluable contributions to this project over the last three years and for their input to this final report.

Fred Rothganger, who spent many long hours developing vision algorithms and implementing behavior on board the Hagar platform.

Charles Q. Little, who interfaced with the Canesta sensor, and helped identify its deficiencies.

Ralph Peters, who both interfaced with the Swiss Ranger imager, and brought the marching cubes algorithm to fruition on our system.

Dan Morrow, who mastered the intricacies of voice recognition.

Table of Contents

Table of Figures

List of Tables

1.0 Introduction

Robot systems have recently achieved remarkable success in military missions, both on ground and in the air, but there is a huge difference in the how these missions are conducted. Air missions are entirely autonomous. Robot aerial drones receive their flight plans and target coordinates, and can then perform with minimal human intervention. Ground robots, on the other hand, are entirely teleoperated. Robot operators use closed-circuit cameras to monitor, and joystick controllers to remotely control every action of a remote robot. The goal of this LDRD was to develop a level of autonomy so that ground based robots could perform manipulation tasks in a cluttered environment without constant human intervention.

Navigation for ground robots is substantially more complex than for their aerial cousins. The potential for collision for ground based vehicles is constant, whereas aerial vehicles have tens to hundreds of meters separation between nearest obstacles. Air-based navigation via either GPS or target tracking is also simpler. GPS is fully adequate for navigation for an aerial vehicle without any additional decision making processes required. Furthermore, vision based operations for aerial vehicles are essentially 2-D, i.e., occlusion doesn"t present a problem to aerial operations. Ground based vehicles, on the other hand, can see a very different view of the world just by moving a few feet.

The mission space for the ground robotic systems is also substantially different. Military ground robots are deployed for disabling IEDs (Improvised Explosive Devices) as a primary mission. They are used to open doors, deploy sensors, aim weapons, and gather debris. In the future they could be used for setting up a "Green-zone", conduct decommissioning in a contaminated facility, or build barriers for force protection. NASA tasks of interest include docking to moon-based trailers, base-station maintenance and satellite assembly. Mobile robots have the ability to reach out and manipulate objects in their environments, and this ability we loosely call "assembly".

Currently these tasks are conducted by remote teleoperation or by humans or not at all, but there are high costs associated with manual operations. In a war zone, the humans present targets for insurgents. In radiation areas, the humans are receiving unnecessary

doses. On a proposed lunar base both humans the neither operators nor teleoperator currently exist, and are extremely expensive to provide.

This LDRD seeks to prove that autonomous mobile robots can perform useful cooperative assembly operations in unstructured outdoor environments. Tasks should be defined by object primitives: not by hardcoded waypoints and/or taught motion paths. Whereas DARPA had already focused on navigation for mobile vehicles, this LDRD seeks to take it a step further: actual interaction and manipulation with the environment.

1.1 Defining a Focus Task

Autonomy for mobile robots is a broad goal, and we decided early on to define a task that would provide focus for our research efforts. The task needed to be achievable in payload, reach, and grasping capabilities for our current robot platforms, be both unique and difficult, and require us to solve relevant problems for an actual application.

We chose "brick-laying" as our sample assembly task. The plan was to use a pair of robots, one as a navigational/leader/transport robot, and one as a follower/manipulator. This concept is shown in Figure 1 below.

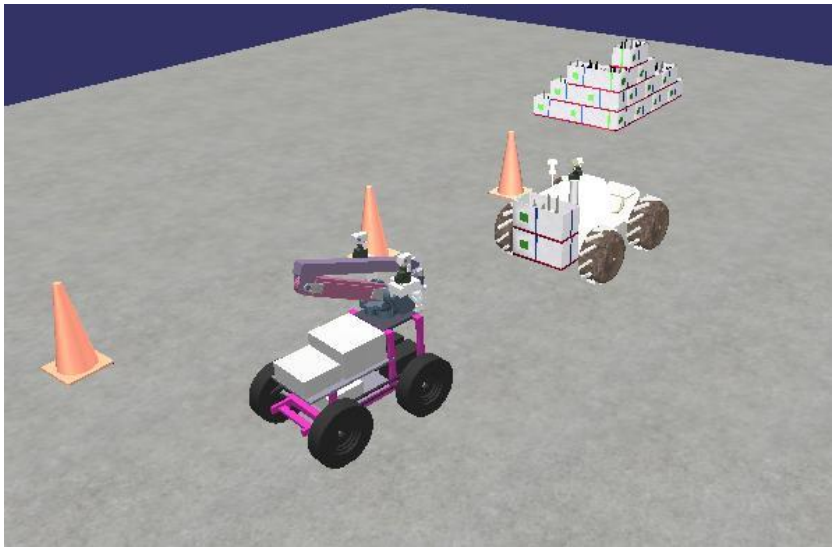


Figure 1.1: The Robot Brick-Laying Focus Task

The robots would navigate relative to a set of colored cones, identify a work-site, and then systematically construct a wall of specially made bricks at the site. We would be

free to engineer the bricks to have a simple grasp point, and to have visual recognizable features to simplify recognition and alignment.

This task would showcase precision assembly, landmark navigation, advanced scanning/segmentation/subtraction technique, pack-mule operations, follower modes, object recognition, and robust error recovery and planning. It would create a compelling demonstration of advanced assembly ---- if we could pull it off.

With this task in mind we started to focus on key elements. We would need capable platforms for implementing and demonstrating the technology. We would need a framework for command and control for these platforms. We would need to be able to identify and localize brick and cones. We would need to accurately navigate around a site. We would need to be able to communicate between robot components and maintain a common database for our models, and we would need to be able to orchestrate advanced combinations of scanners, planners, and motions.

1.2 Technology Starting Point

A number of the necessary components were already available at the outset of this project. A pair of 6-Degree-of-freedom (DOF) mobile manipulators called Turing robots had been used for substantial work in remote teleoperation (Anderson, 2008). They had no navigation system, but did utilize a pair of calibrated targeting cameras. The HAGAR robot system had been first developed for remotely monitoring Army depots, and had been used in earlier research into Autonomous Navigation. (Klarer, 1994, Eisler, 2002).

In addition, Sandia‟s Robotics group had already developed two mature software frameworks: SMART, for-real time control (Anderson, 1995) , and Umbra for high-level visualization and planning (Gottleib, 2001), and we had faith that commercial scanning technology would provide the data we needed for manipulation.

We also had an active perception group with substantial experience in interfacing with commercial sensor systems and interpreting the data streams.

2.0 Command and Control

Implementing autonomous robot behavior requires methods and techniques for converting sensory data and goals into actions for the robots. These actions cover a wide variety of different actions the involve servoing pan-tilt units, adjustments to zoom settings, driving robot bases, moving robot manipulators, setting up planners, etc. In our prior systems we had relied heavily on human operators to perform and execute these actions, usually by interacting with a graphical user interface, clicking on buttons and manipulating joysticks. To achieve autonomy, however, we needed to rethink this approach. We needed a system that would allow us to incrementally build up complex behavior using building blocks and scripting.

2.1 The SmartCmd Class Structure

The first step in developing an abstract command class was to determine the elements of a generic command – without getting caught up in the specifics of our particular robot systems. A command has a beginning and an ending. It requires time to execute and utilizes resources during execution. It can either achieve success or fail along the way. Furthermore, the operator needs to determine status for an executing command and be able interact with its execution. The operator might need to stop and abort any command. In many cases, especially in the case of robot motion, the operator needs to be able to pause, play and rewind and commands. The commands should be loaded and created from a database in a flexible fashion. Finally, the command class needed be fully embeddable. Complex command sequences should evolve from much simpler commands.

These concepts were implemented in *IncrTcl* (http://incrtcl.sourceforge.net/itcl) inside a base class that we designate a *SmartCmd*. Each *SmartCmd* object has the following key components:

- Methods for *Init*, *Play*, *Pause*, *Stop*, *Exit*, getting the current state *(GetCmdState)*, and getting the percent of the task completed (*GetPercentDone)*.

- An *Update* method which is attached to a wall clock timer during execution and periodically calls *GetCmdState*, & *GetPercentDone* methods.
- Methods for reading and writing parameters from XML files.
- The ability to recursively imbed other command objects, in parallel, or in series.
- A global "resource" list that ensures that only one command is simultaneously utilizing a named resource.

By requiring that all commands implement methods for *Init*, *Play*, etc, we were able to develop an abstract framework that extended to all of the specifics activities needed for robot control. The *SmartCmd* framework then implements a state engine for all objects of the class, as shown in Figure 2.1.

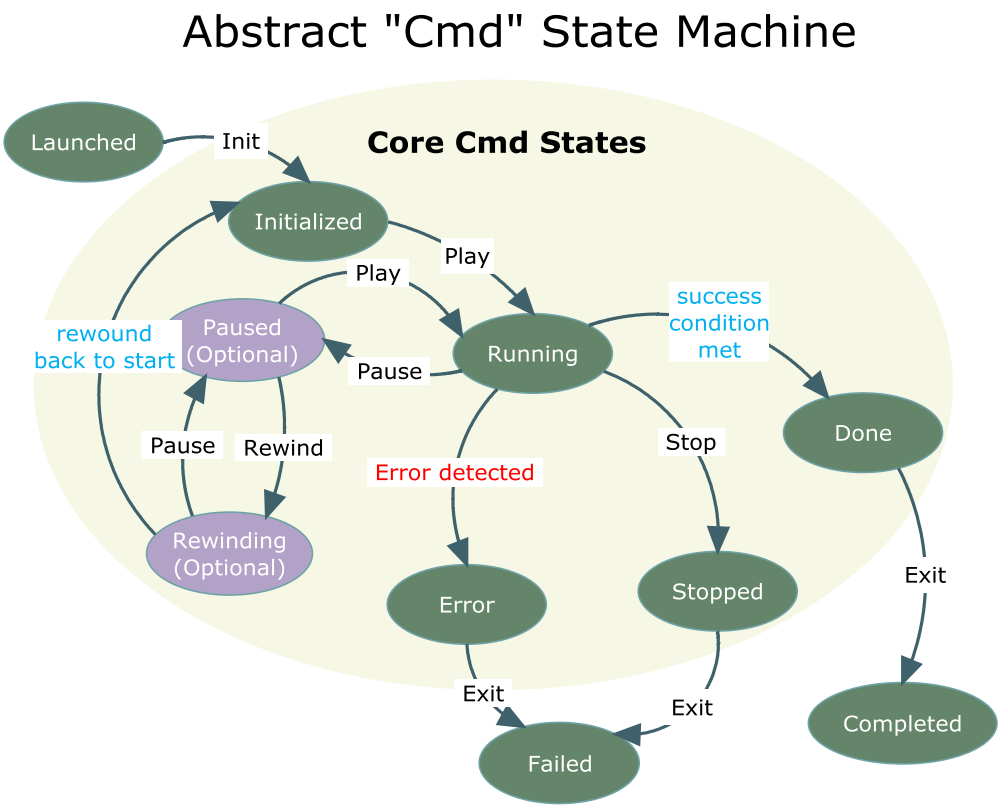


Figure 2.1: *SmartCmd* State Engine

2.2 Chaining Commands in Sequence

The *SmartCmd* approach allows advanced commands to be constructed from other simpler commands, which can in turn be combined into even more sophisticated commands. As an example of how this is done, we look at defining the commands for the *SmartMultiCmd* object. In the *SmartMultiCmd*, multiple command objects are executed in temporal sequence. The command object is initiated with a list of subcommands, each being a *SmartCmd* object itself. Initialization of the command begins with initialization of the first command in the list. Once one command in the list is finished, the next command should be executed and so forth. Upon success, each proceeds to the next command state automatically. Each command objects calls the current embedded command methods.

The *GetCommandState* method for the *SmartMultiCmd* returns the command state of the current queued command . The *Play/Pause/Stop* methods simply call the same methods for the currently running queued command. The Resource queue keeps track of named resources required for each command. If resources aren't available, the system goes into *InTransition* state. The bulk of the work in the *SmartMultiCmd* is in the update loop which must monitor the current command and initialize the next command in the queue once it completes. Figure 2.2 summarizes this behavior.

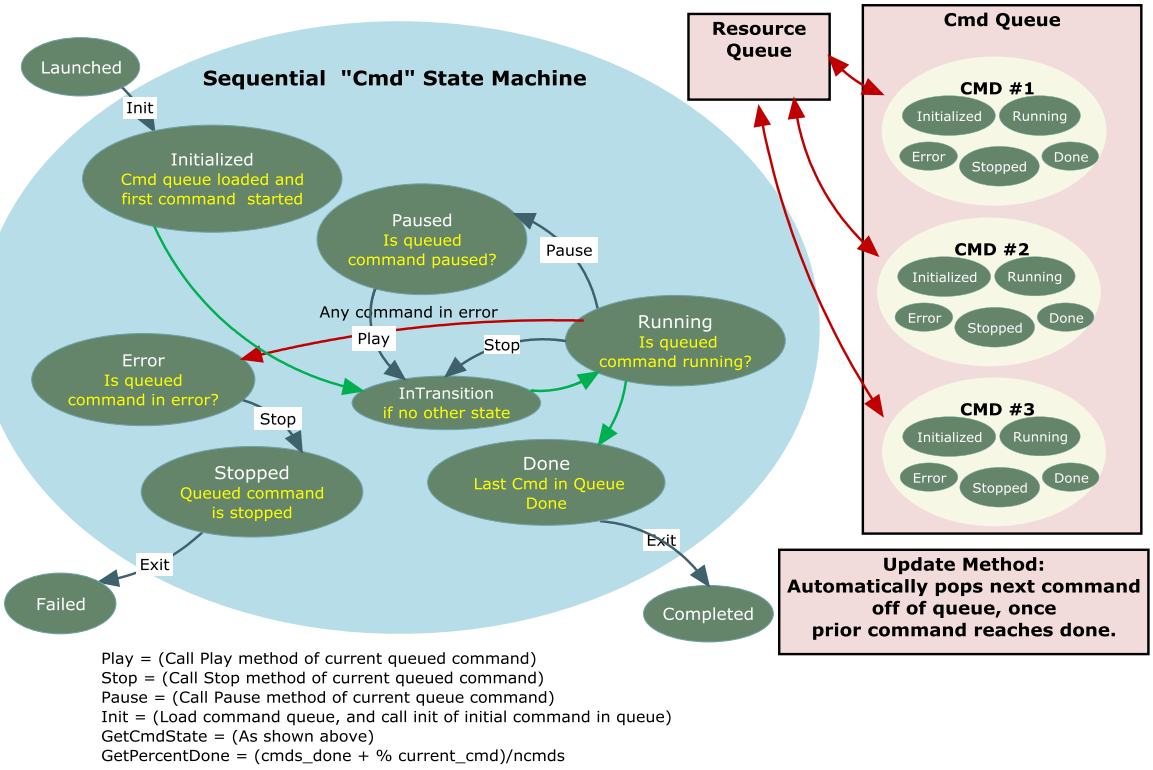


Figure 2.2: Composite Command State Diagram For MultiCmd.

2.3 Command Summary

Other composite commands implement additional features: performing operations in parallel, queuing for events, branching to an error recovery procedure if a system failed. Since every command is an instance of the base command class, methods for loading and saving via XML, for initializing, for aborting, playing, pausing etc are always available. Advanced commands such as *DoUntil*, *DoAndWait*, *AorB*, *MultiCmd* interface to individual sub-command objects, and allow increasingly sophisticated robot behaviors to be created from primitive parts. Tables 2.1, 2.2, & 2.3 summarize the different command objects that have been implemented.

Table 2.1: Robot Motion Commands.

Command Name	Description
SmartTrajCmd	Moves a robot device a long a motion trajectory

SmartMoveAlongHighwayCmd	Moves a robot to nearest point along a motion highway and continues till end of highway
SmartTagCmd	Move a robot device to a single tag point
SmartMoveRelCmd	Moves a robot device a relative distance
TrajDualCmd	Queue up a joint motion followed by a straight-line world motion.
SmartPlanTagCmd	Call planner to generate a goal destination tag.
SmartMoveAxisCmd	One degree of freedom of device is move to defined location (e.g,, just change zoom setting
SmartMoveToHighwayCmd	Move to nearest point on a motion highway.

Table 2.2: Composite and Miscellaneous Commands

Command Name	**Description**
SmartDoAndWait	Initiate first command, wait, and call final command.
SmartDoAorB	Do initial command, if it fails, try the second.
SmartDoUntilCmd	Execute sub-command A until sub-command B is done
SmartMultiCmd	Do a series of sub commands in sequence
SmartTimerCmd	Wait a given time
SmartStartWaitAndFinish	Start one command, wait for timer, call an exit command
SmartSimulcastCmd	Initiate a series of commands from a list, wait till they all complete, or one fails
SmartLoadScriptCmd	Load a defined Tcl script from file.
SmartInstantCmd	Execute a single Tcl procedure immediately

Table 2.3: Visual Tracking and Servoing Commands

Command Name	Description
SnakeGrowerCmd	Grow a polygon snake to find a target object
SnakeFindCmd	Find a desired polygon inside a visual image.
GridServoCmd	Servo PTU on found color grid
GridFindCmd	Find a match to a given color grid.
BlobServoCmd	Servo based on the centroid of a color blob.
BlobFindCmd	Find a color blob that matches a criterion.

2.4 Command Execution: Simple GUI

The figure below shows a sample GUI that is automatically created from a command set. XML files describe the various commands in system. When the XML files are parsed, the command objects are instantiated and the GUI page is automatically generated for the commands. Ideally, large sequences of commands would be chained together and executed as a single command object, but the divide and conquer approach helps to make debugging sub-commands tractable.

A VCR-line control interface provides an intuitive interface for pausing and playing and rewinding the robot behaviors.

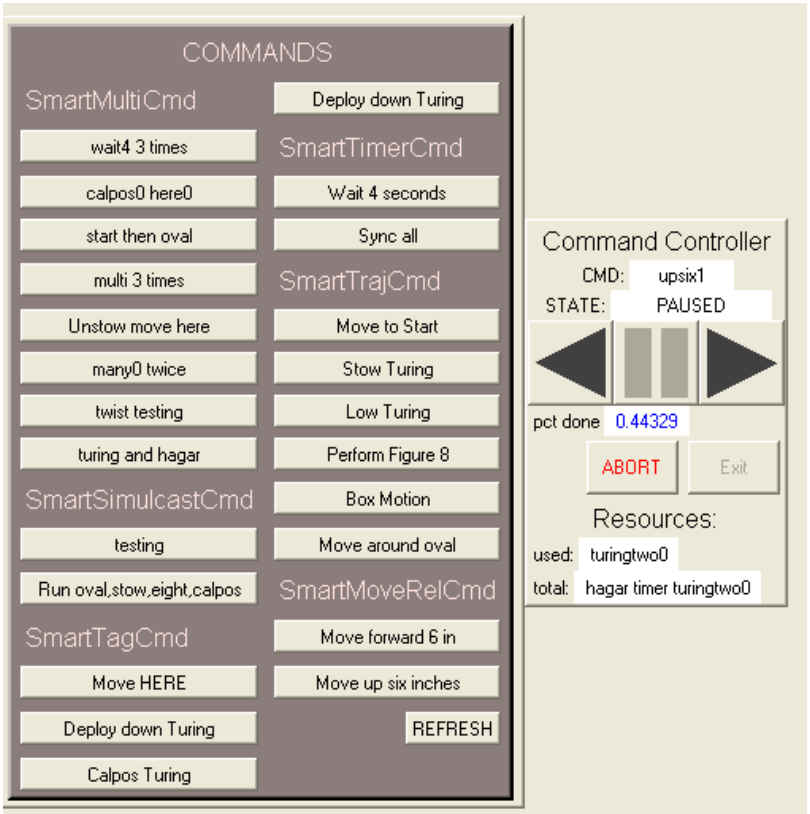


Figure 2.3: Automatically Generated User Interface for SmartCmds

3.0 Audio commands

By the final year of the LDRD it became clear that full autonomy was not going to be achievable, and that a high level of operator intervention was still required to guide and direct robot systems. Because our operations were directed to robot systems working outdoors over large areas and that likely operators would be foot soldiers, it seemed natural to free up the operator from being chained to a desktop and let them command semi-autonomous behaviors in the field by using audio commands.

To make this happen, an audio layer was implemented within the *SmartCmd* base class, called the *smartSpeechRecognizer*. Because every command had an associated label to display on the button, we decided to reuse this label as a means to queue the command. The belief was that voice recognition had advanced to a point that we could quickly and reliably recognize phrases from a command set and initiate various robot operations.

A key concept in the speech recognition is the definition of grammars. A grammar defines which phrases (as strings) that the system is able to recognize. Grammars can be defined in sophisticated ways to form numerous recognizable phrases. In addition, a callback function is invoked when a phrase is recognized. Multiple grammars can be defined and active. In this way, unique event handlers (e.g. callbacks) can be attached to each different grammar to achieve different behaviors.

The *smartSpeechRecognizer* implements two grammars: A Command grammar which is built on-the-fly based on the *SmartCmds* in the system, and a Control grammar which is fixed. The recognition logic implemented in the callback also tries to implement a valid state transition from various command states as shown in Figure 3.0. These grammars are mutually exclusive in activation – only one is active at time – which one is controlled by the state machine shown below. The Command grammar is always the initially active grammar.

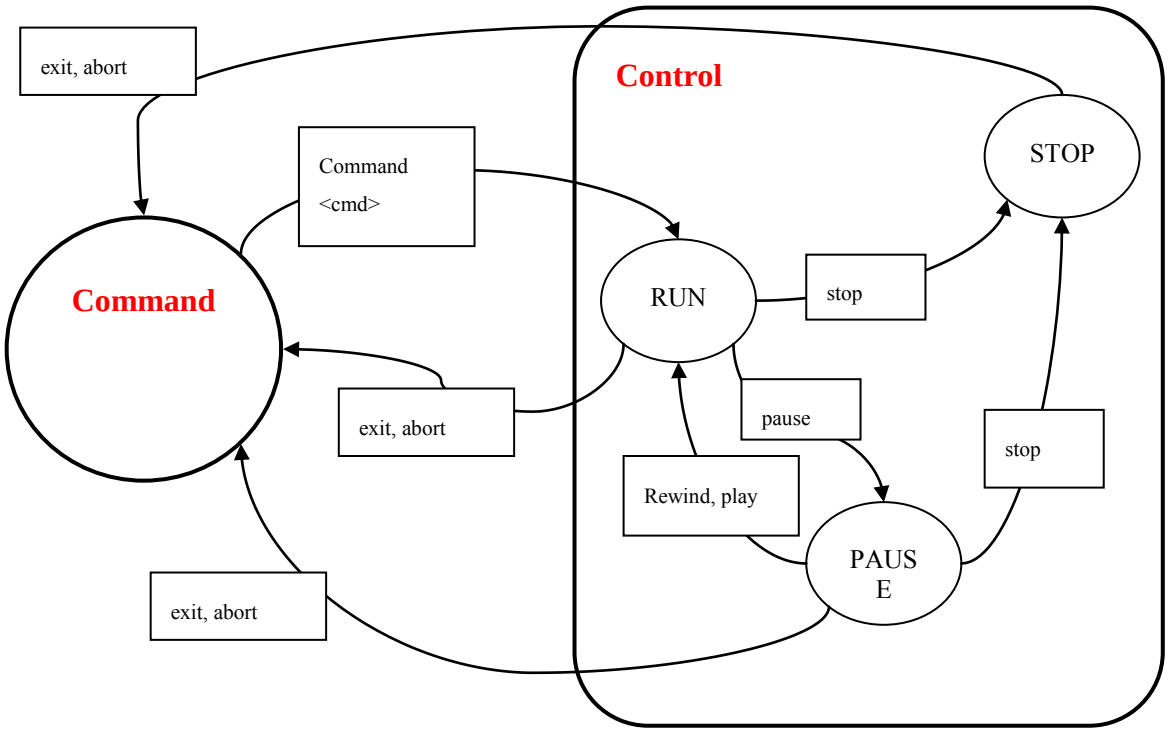


Figure 3.0: Two-Stage Grammar For Speech Recognition

The Grammar is defined by passing various string phrases to the grammar. In addition, the *smartSpeechGrammar* is designed to have an option prefix phrase. The default ones are "Command" for the Command grammar, and "Control" for the Control grammar.

Control grammar phases are based on the familiar VCR controls:

- Play
- Pause
- Rewind
- Exit
- Abort

and are intimately associated with the state engine of the SmartCmds (Figure 2.1). Exit and abort are commands to abort the command and begin listening for a new command.

3.1 Audio Example

As an example of the implemented grammar, consider the following robot system commands.

- "Move to start"
- "scan left"
- "scan right"
- "box motion"

Table 3.0 summarizes the phrases spoken by an operator, and then resulting behavior queued by the speech system based on its current state.

Table 3.0 Speech Commands and Actions

Spoken phrase	Resulting Action	Speech System state
Move to start	Nothing – prefix Command was left out	Still listening for command
Command Move to Start	Move to Start begins to execute	System now listens for Control command; in RUN state
Control Pause	Move to Start pauses	System is now in PAUSE state
Control Play	Move to Start cmd resumes	System is now in RUN state
Control Abort	Move to Start cmd aborted	System is now listening for Command
Control Pause	Nothing – no recognition	System is still listening for a Command

One reason we use the prefix is it seems to increase the accuracy by providing a longer phrase for recognition. Short, terse phrases seem to be more easily mistakenly recognized – and even totally different noises are sometimes recognized as these one-word phrases. But including the prefix has drastically cut down on these false positives.

3.2 Audio Feedback

In addition to recognition, the SpeechSDK also supports speech synthesis. Each recognized phrase is echoed to the user which gives the user confidence in what was recognized. We can specify an echo phrase or rely on a default one for Commands. For example, “Move to Box” might have a defined echo of “moving to box” or the default which prepends “executing” to the command (in this case: “executing move to box”). The recognition engine is actually turned off during this echo phase so that the echo is not in turn recognized. With a headset this is unnecessary, but with speakers it can become a problem if recognition is on while this feedback occurs.

Feedback turned out to be critical in establishing a rapport with the operator. Without audio feedback queues, the operator would be confused as to the state of the robot. Did it understand the command? Was if finished with a command?

4.0 Broadcast Communications

Another technology required for implementing cooperative autonomy among multiple robots is a robust communication framework. The communication layer must keep each robot informed of what others are doing, what sensors are reading, and the current desired and actual command state of each subtask in the system. In our architecture, all of this data is transmitted to a base station computer that orchestrates the underlying behaviors of the distributed system.

Our previous communication framework was based on point-to-point TCP and UDP sockets. A single workcell defining all of the robots and all of the data that needed to be transmitted was defined and channels were setup up between each computation pair. This worked well for a single base station and a single robot, but as more and more subsystems were added, it became more unwieldy.

We had also investigated numerous other communication frameworks: SPREAD (http://spread.org/) , Remote Procedure Calls (RPC), JAUS (http://www.openjaus.com), etc. and found them all to be lacking for one reason or another. Spread, for instance lacked efficiency due to its dual transmission of data over UDP and TCP. RPC required recompiles of each target system whenever a new communication object was required, and JAUS added significant overhead and would force us to use a limited robot vocabulary.

The requirements for communications were as follows:

- The communication architecture would need to pass data freely between any combination of SMART and Umbra subsystems.
- Data passing should be passed with minimal latency
- The overhead associated with information passing should be minimal.
- Information must transfer between processes running on the same CPU and between processes running on different CPUs.

- Rapidly changing data should be transmitted within a guaranteed minimum time step.
- Static data should still be transmitted periodically in case remote systems missed an earlier transmission.

The eventual implementation of our communication is called *smartBroadcast* and utilizes UDP Broadcasts over sockets. The UDP Broadcast allows multiple targets to receive data at once without any additional overhead. A blackboard framework is used for logging data that needs to be transmitted to a target. Only the most recent data is sent. Data is backed in binary with coded host and processor id, data type, and a two-byte cyclic-redundancy-check (CRC). To improve efficiency, name synchronization packets are used to associate multi-character data names with efficient register numbers. The data is passed at high rates with only the register numbers rather than the descriptive names.

5.0 Robot Platform Development

At the initiation of the project we had two robot platforms available: the meter sized RATLER vehicle known as HAGAR (Fig 5.1), and the mobile manipulator arm known as Turing (Fig 5.2). Hagar provided cross-country mobility, power, and suitable size for mounting sensors and scanners. The Turing system had an existing SMART based open-control architecture and a highly capable arm with installed camera PTU systems.

Unfortunately, neither platform was ready for autonomous operations. Hagar had been running a MSDOS era computing stack and a substandard GPS system. It used an Analog 2.4 GHz video radio, and a simple serial radio for command and control. The prior developers of Hagar had either retired or had left Sandia.

Turing on the other hand used a tether to hook back to the base station, and had neither GPS, nor obstacle avoidance safety sensors. The wrist/grasping closure of the Turing gripper was also not position-servo controlled, which meant we would have to accomplish wrist closure control via visual servoing.

Figure 5.1: HAGAR at Take Your Sons & Daughters to Work Day '08

Figure 5.2: Turing Robot at Take Your Sons & Daughters to Work Day '08

Under the LDRD we were able to upgrade Hagar to make a viable system for research at the robot vehicle range. New SMART modules for the compass (TCM2), Tilt sensor (XBOX_TILT), and high-precision GPS (NOVATEL_OEM4) were added to the base Hagar system (Fig 5.3). A Kalman-Filter based motion estimator module (POS_KALMAN) was developed to provide continuous inertial reference frames estimates for Hagar based off of compass data, velocity commands and GPS updates. With differential corrections, Hagar is now able to achieve positioning accuracies within a few centimeters. Furthermore, a remote "Xbox" interface was added to the Hagar platform so it could be easily moved and setup at different sites.

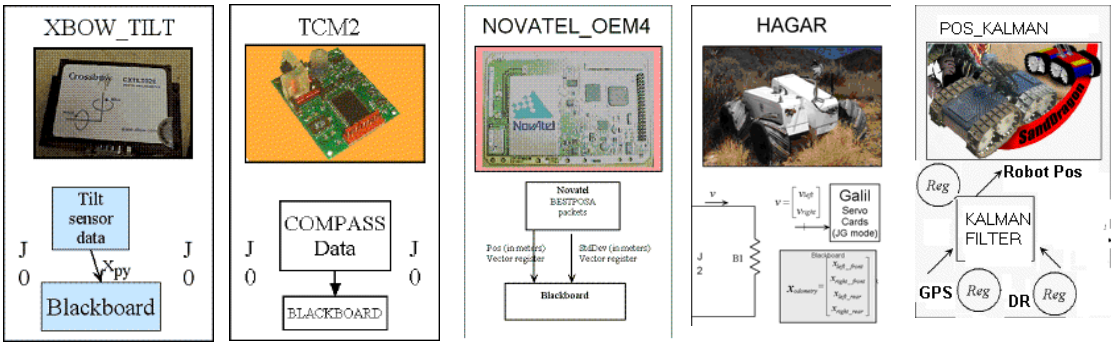


Figure 5.3: New Modules Developed For Navigation Robot

With the upgrades provided by this LDRD, Hagar has made numerous public showing and demonstrations: The X-Prize ‟07, American Nuclear Society Conference „08, Family Day „09, etc.

Updates to free Turing from its tether were planned, but were put on hold as we focused on software activities.

6.0 3D Scanning

Three-dimensional scanning had always been viewed as a critical part of autonomous assembly in unstructured environments. For manipulation, objects need to be grasped, candidate grasp points on objects need to be identified, and object poses need to be accurately measured. Even when manipulating known objects with given a priori grasp points (such as the brick targets), the objects need to be matched to existing data sets and positioned in space.

Range scanning provides two benefits over vision based systems. The scans provide a rich set of points for correlating to known models to, and it can define areas of collision free space in which a robot can move. We had hoped that commercially available portable scanning technology had evolved to a point to be useable. What we learned was that scanners each had their own idiosyncrasies, and that substantial algorithm work was still needed to make the leap from sensor scans to robot action.

6.1 The Canesta Scanner

The Canesta DP300B (Fig 6.1) was the first candidate sensor. It provided distance scans at 4 Hz over a 40 degree cone and with a 200x200 pixel array. It was affordable ($7K) and suitably compact to be deployed by a mobile robot.

We developed an Umbra module to interface to the sensor and display the data. For fine assembly we hoped to first generate background scans, and then as new objects were added or deleted from the scene planned on looking at the differences of the scan data and fit our set of known objects to the data scan. Figure 6.1 shows the scanner data of a brick object within the Umbra environment using the Canesta.

Figure 6.1 Canesta DP300B Scanner and Umbra Scan Data

6.2 Problems with scanner data.

As we worked more with the data we realized that our goal of localizing objects with 1.5 centimeters and orienting objects with 5 degrees would be difficult to achieve. There were a number of anomalies with the data. Flat surfaces, especially corners, would be distorted. Objects would have aliased appearances so an object at 6 meters of distance may appear at three meters of distance. Surfaces with poor incidence angles with respect to the scanner would give misleading data. Some of these effects are shown in Fig. 6.2.

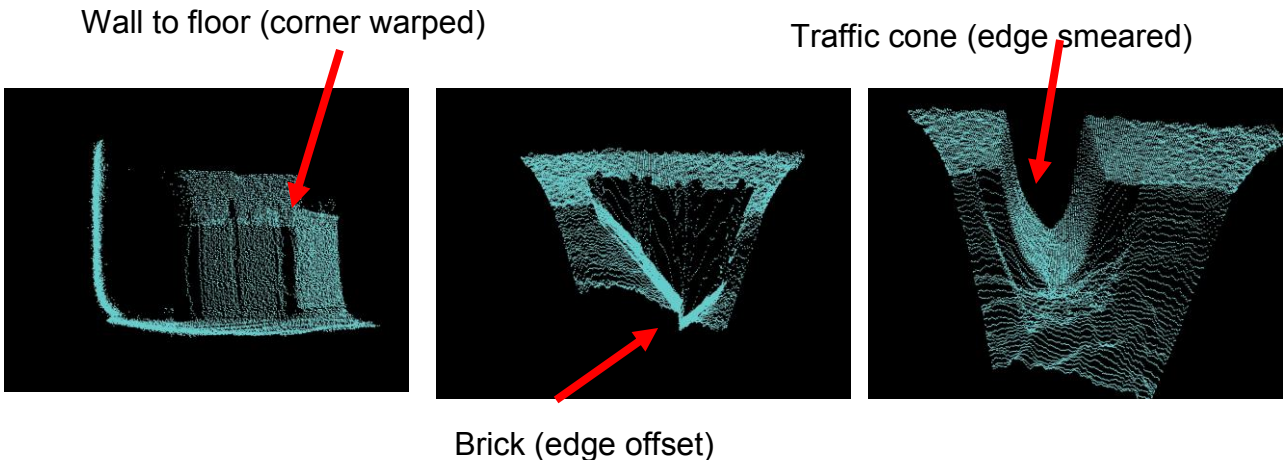


Figure 6.2: Anomalies in Canesta Scan Data.

Ultimately the number of issues in the data made it difficult for algorithms to automatically fit known objects to the scan data with a high-level of confidence. After the first year"s efforts in scanning, we decided to drop scanning as a primary activity for the LDRD and decided to rely entirely on vision based approaches for recognizing objects and placing them in the robot scene.

6.3 Scanning Revisited: Using The MESA SR4000.

Despite our early frustrations with converting scan data to accurate models for robot interaction in the first year, we considered the type of data available from an imager as a critical for successful robot operations in unstructured worlds. When we became aware of a new, more capable imager we decided to reinvestigate 3D imaging in the third year.

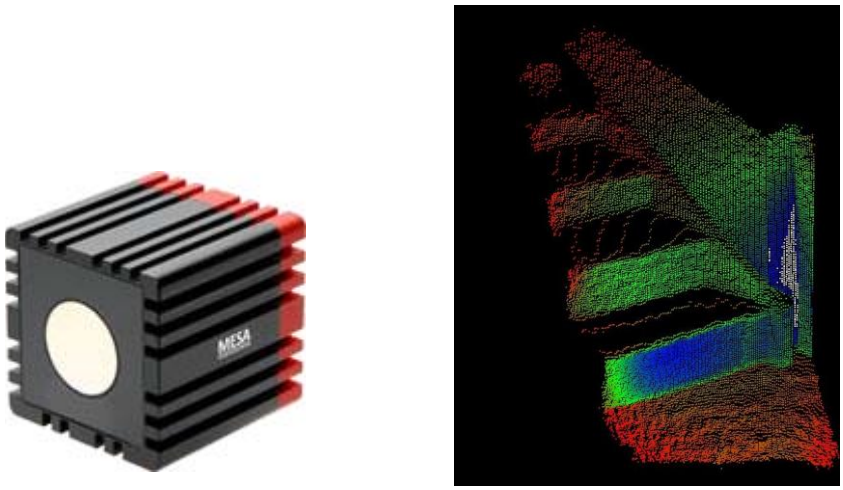


Figure 6.3 MESA SR4000 and a Stairway Scan

The MESA SR4000 imager (Fig 6.3) is a 5cm cube that meets many of our requirements. It is eye-safe, it has a high-speed USB interface with a 30 Hz update and a 176x144 array of points , it has double the range of the Canesta (7 meters vs. 3 meters), and is more reliable and has far less noise than the Canesta.

As with the Canesta, we developed an interface into our Umbra environment that would allow us to visualize and process the data clouds. A sample scan is shown in Figure 6.3.

6.4 Using Marching Cubes to Build Surface Models.

One of the limitations of our first year‟s work in scanning is that we didn‟t address how to combine multiple scans. With a scanner having only a 40 degree field of view, it is critical to be able to constantly merge new scans with old. Ideally as data is accumulated a comprehensive map of the environment is obtained. With a better scanner in hand, we decided to attack the problem of combining data into a single surface model.

A variety of methods have been suggested for merging scans. Two major classes are merging surfaces (Turk and Levoy, 94) and volumetric methods (Curless and Levoy, 96). After a review of the voluminous literature, a volumetric approach based on marching cubes (Lorenson and Cline, 87) was chosen for this work. Marching cubes software is available from a number of sources including "experimental" software that is highly parallelized and uses the large number of simple processors available on modern graphics cards. As a result of various constraints, the marching-cubes algorithm available in the Visualization Toolkit (http://www.vtk.org) was chosen because it was available, robust, and documented.

An office (Fig 6.4) was scanned using the MESA imager. The scanner was placed near the center of the room on a tripod and rotated horizontally in 0.5 radian increments for a total of 13 horizontal scans. The scanner was located at three different heights, 0.63 meters, 1.14 meters, and 1.42 meters above the floor to get better vertical coverage of the room. At each scan location, three scans were taken to provide more data which is useful in the averaging process inherent in all approaches for merging scans. A total of 117 scans were taken containing almost 3 million points (Fig. 6.5 & 6.6). Some surfaces may be invisible to the scanner because they are "dark" and reflect very little IR light. The black cases of Dell computers, and LCD monitors are almost invisible to the SR 4000 scanner.

Figure 6.4: One corner of a sample office.

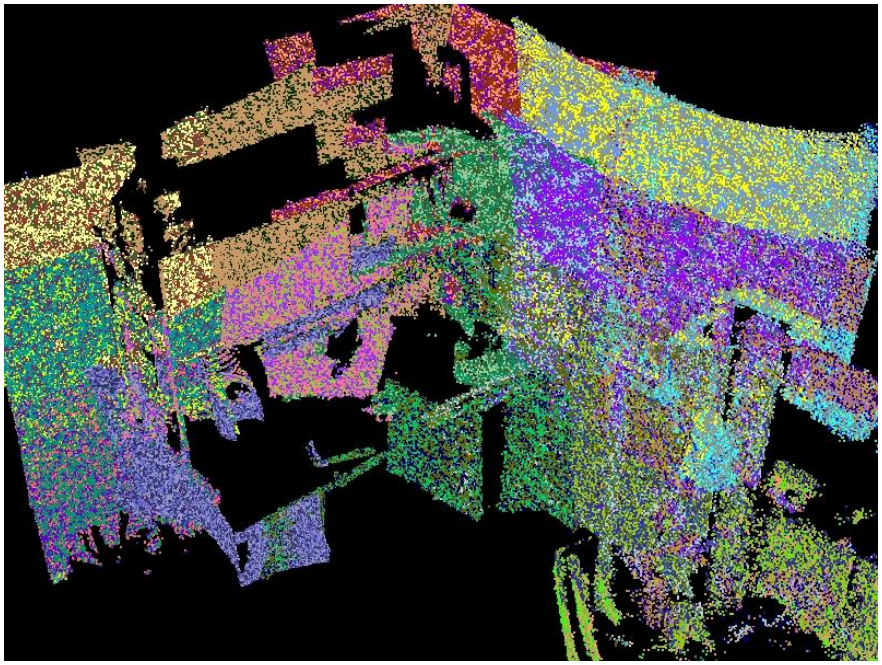


Figure 6.5: A series of scans to be merged for the office.

Figure 6.6: Overhead view of data set of complete office scan.

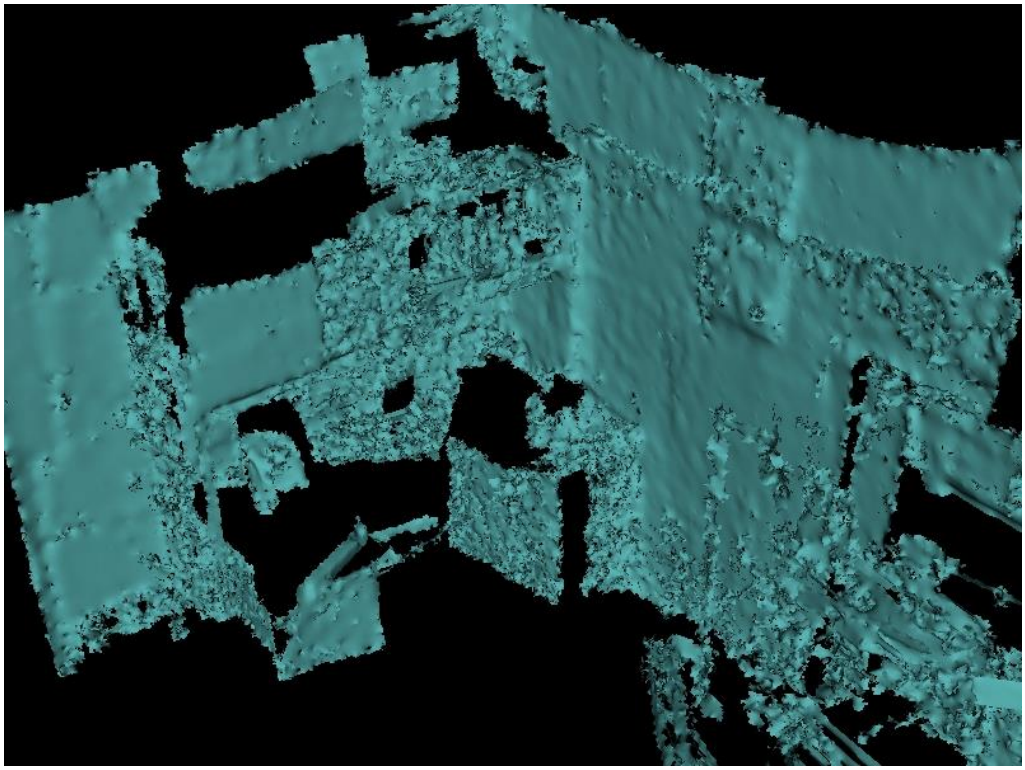


Figure 6.7: The resulting merged scan using Marching Cubes.

Figure 6.7 shows the resulting merged scan using the Marching Cubes algorithm. The segments of rough surfaces should be smooth, such as the lower vertical surfaces of the desk and the right half of the closed space-saver door, result from error in positioning the scans relative to each other.

This investigation was intended as exploratory, however, the computational time required to complete the building of the 3D model is important as it indicates future directions for further work. The scan data must be loaded, filtered for noise and other artifacts, and stored in a voxel object before the marching-cubes algorithm can be used to extract the surfaces of the 3D model. On a Dell Dimension 670 PC, the time required to sequentially load, filter, and store the data (117 scans containing 3 million points) was 370 seconds. The time used to extract the 3D surfaces, including a variety of filters to remove various artifacts like tiny, unconnected triangle patches, was 39 seconds. Little work was done in optimizing portions of the software where a lot of time is consumed; major speedups may be possible after a careful rewrite. Parallelizing this software is possible and would reduce the task time roughly in proportion to the number of CPU's available Some experimental software uses the processors on the graphics card (up to 240 processors per card!) for the marching cubes calculations and claim to have several hertz performance rates (http://nvision.sourceforge.net/).

6.5 Scanning Conclusions

Although 3D range scanning will in the future play a critical role for robot systems in unstructured environments, we barely scratched the surface in terms of generating the data needed for robot assembly. Two different sensor systems were successfully integrated into the Umbra framework. We deployed our own segmentation and difference algorithms, and integrated a conventional marching cubes algorithm for combining multiple scans. Data was successfully reduced from 3 million points to surfaces with thousands of polygons.

The scanning data was substantially simplified using Marching cubes, but not to a point where it could be used by an autonomous robot planning system. A robot system needs a world model database that can be queried, ideally consisting of known objects, and convex polygon hulls for unknown objects. The generated data set needs to be able to quickly generate answers to simple questions. What is the nearest distance to a surface? Is an object in the scanner view a known object that needs to be manipulated, or is it just clutter that needs to be avoided? In order for the data generated from the combined scans to be useful for robotic assembly operations, the data needs to be iteratively matched to expected and known objects and differenced out from the data set, and it needs to happen in real-time. In this area we grossly underestimated the level of work required to achieve success.

7.0 Feature Tracking and Object Recognition

The scanning approach described in the previous section was never intended to be the sole approach for building robot compatible models. Our teleoperated robots are all outfitted with cameras, and human operators are able to achieve incredibly dexterous operations solely from visual feedback. If an operator can make sense of a live visual camera feed, we might expect our autonomous systems to as well. For this reason, computer vision approaches have been pursued throughout this LDRD. In this section we discuss a large number of techniques and tools that were developed over the last three years.

7.1 First-Year Efforts: Canny Detectors & Hough Transforms

In the first year of the LDRD we decided to assess what conventional approaches would be able to do for us. The initial question was, could a vision system identify, track and monitor traffic cones under adverse conditions and in real-time on conventional PC hardware? These conditions included: large amounts of foreground and background clutter, occlusion of objects, partial views of cones, and highly variant lighting conditions.

Our initial approach was to use a color filter to emphasize objects matching the hue of the cones. Conventional Canny edge detectors were then used to determine line features. Finally these line features were searched to find candidate polygons. When this proved to be unsuccessful, we developed a new approach using a line-detector on a Hough transform of the gradient of the filtered image. This was first tested in the lab (Fig 7.1) and looked promising for fielding.

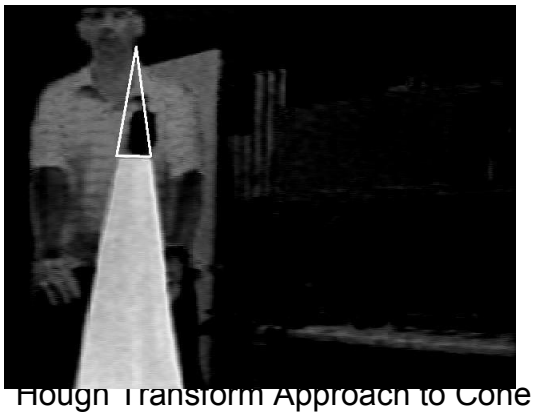


Figure. 7.1 "Hough Transform Approach to Cone Detection

Figure 7.2 Hagar automatically moving to Cones on Feb 2008.

In early February of 2008 we were to a point where we could test the cone finding algorithm on board the Hagar robot. We used the current estimate of the cone"s position relative to the robot"s base to drive the robot. The goal was for the robot to simply find cones and approach them.

The system was partially successful. Cones were recognized and the robot did approach them. But real world testing made the lack of robustness of this approach all too

apparent. This conventional computer vision approach did not maintain any state information. Every camera image it received it would reprocess, rediscover edges, and make a new estimate of a cone‟s position. There was no model evolution and there was no confidence metric for the data. With the camera placed on a jerky skid-steer vehicle the image quality was degraded by noise, and even with a ninety percent cone detection accuracy the robot would rapidly start chasing phantom estimates. Furthermore, without maintaining state, there was no means to keep track of multiple cones in the field of view.

Clearly a new approach was needed.

7.2 Polygon Snakes

Real-time vision systems in unstructured environments needs to be fast (processing 30 Hz 640x480 pixel images at frame rate), they need to handle variable lighting changes and shadows, they need to identify features that can be associated with objects (i.e., good features to track), they need to be robust to noisy images, and they need to maintain state, i.e., an object tracked in one frame needs to correspond to the same object in the next frame.

Statistical pressure snakes (Shaub & Smith, 2003) or Active Contours is a computer vision technique that achieves many of these objectives. The algorithm grows or shrinks a contour based on a computed energy function. The energy function pushes out nodes when they lie inside an image region, and the line energy pulls the contour in. It is robust to noise, it tracks targets, and gives a sense of an object by wrapping it in a closed contour. State is maintained from image frame to image frame. By emphasizing hue over grey scale variations, it is possible to obtain good performance in-spite of variations in external lighting and shadows.

The conventional snake (or active contour) algorithm still has a number of issues, however. As a target image changes in the image size, the number of nodes on the snake would change. Sharp corners, which are normally considered excellent tracking features, are rounded when using a conventional pressure snake.

We decided we could do better, and developed a new algorithm, called “Polygon Snakes” that would accurately track objects with feature corners.

Figure 7.3 illustrates the classical snake, and Figure 7.4 shows our polygon snake algorithm.

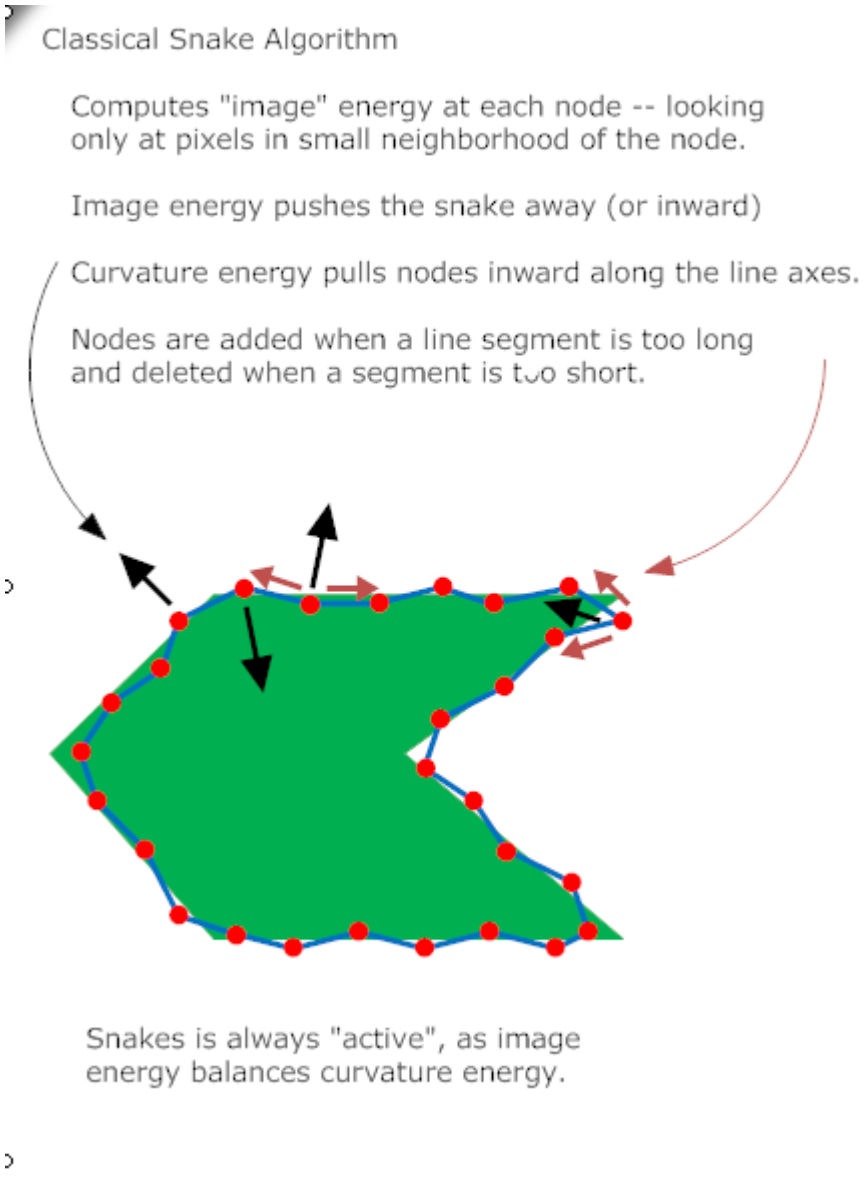


Figure 7.3. Classical Snake Algorithm.

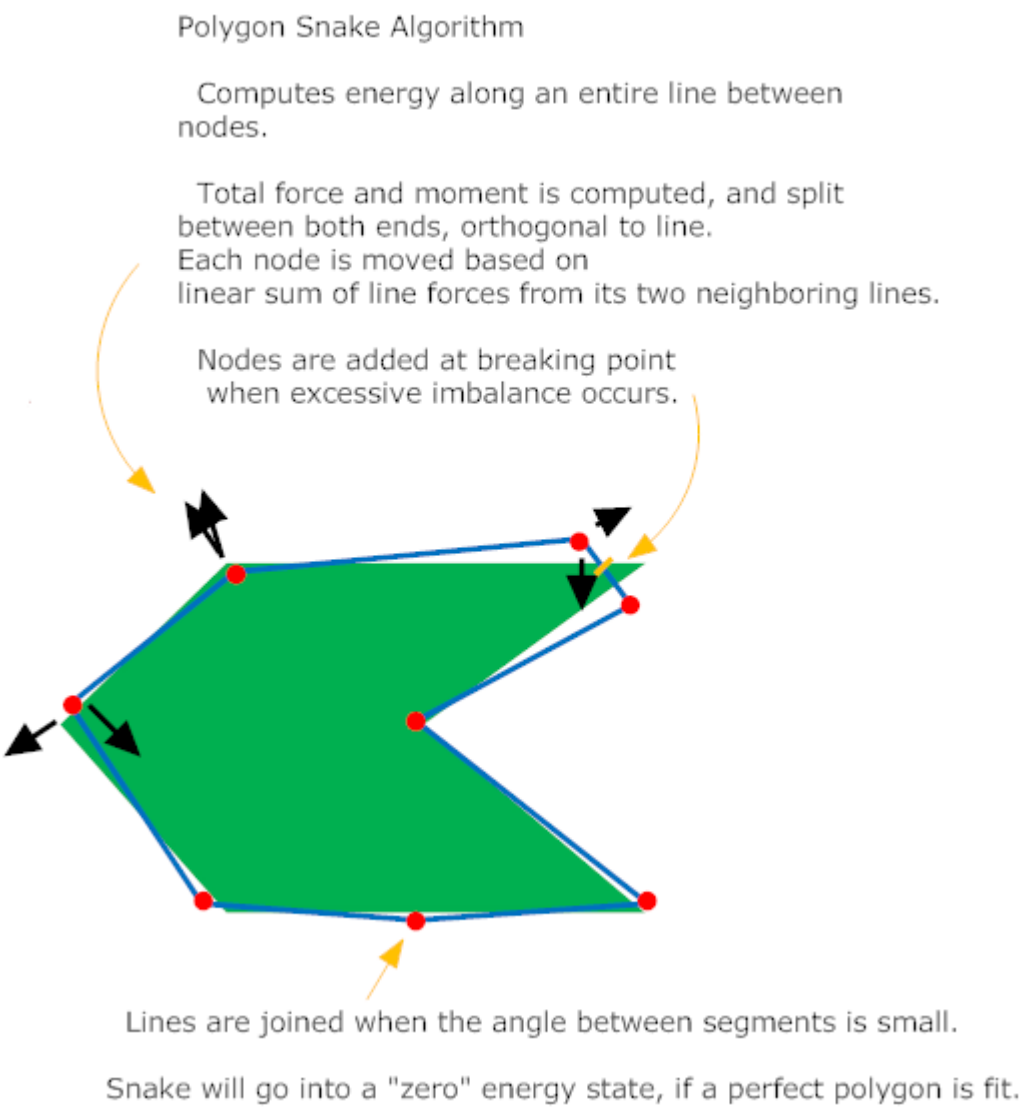


Figure 7.4: New Polygon Snake Algorithm

The process of converting video data to object models is shown below. First a target area of interest is identified from the original video (Fig 7.5).

Figure 7.5: Target Blocks

Then a Snake is seeded based on the color blob (Fig 7.6).

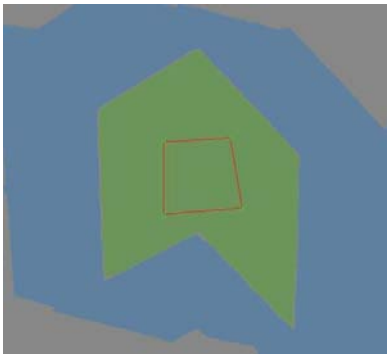


Figure 7.6: Initial Snake Seed

The snake then grows till it reaches the contour segment as shown in Figure 7.7. This happens in fractions of a second, but is broken into steps here to illustrate the snake in operation.

Figure 7.7: Evolving Snake

And finally it settles on a final contour (Fig 7.8).

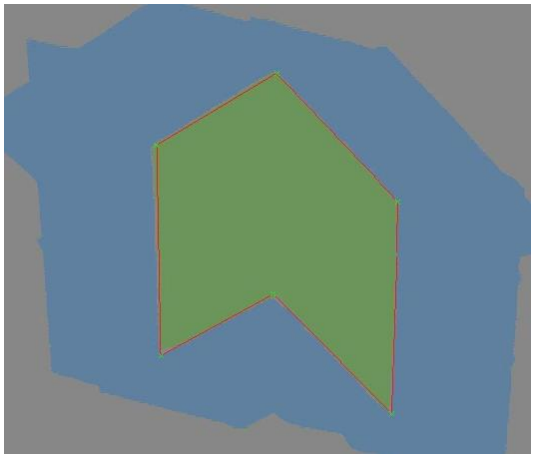


Figure 7.8: Snake Settled on Contour

The polygon snake approach overcame many of the limitations of the line detection systems. It maintains state. It tracks critical feature points. It is robust to noisy signals.

7.3 Identifying Polygons.

One of the advantages of using polygons as features of interest to recognize is that they are easy to identify and they maintain their functional shape under perspective distortion. In general, perspective distortion will cause large variations in the amount of corner angles (e.g., a rectangle may look like a diamond under a perspective transformation), but because straight lines map into straight lines under perspective transformation, it also follows that the convexity condition (i.e., is the corner angle positive or negative) is also invariant under perspective distortion.

By using polygons with unique patterns of inner and outer corners it is possible to uniquely mark and identify objects. In our code we associated target polygons with a binary encoding of their corners (i.e., a one for every corner that is convex, zero if concave), and looked for polygon snakes that mapped within the set of possible permutations. Figure 7.9 gives a sample set of polygons and the binary coding associated with the patterns.

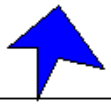
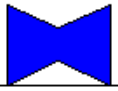
Name	Sides	Binary Pattern Encoding	Hex permutations	Shape
Triangle	3	111 111 111	7, 7, 7	
Rectangle	4	1111, 1111 1111, 1111	0xf, 0xf, 0xf,0xf	
Star Fleet	4	1110, 1101 1011, 0111	0xe, 0xd, 0xb, 0x7	
Ribbon	5	11110, 11101, 11011, 10111, 01111	0x1e, 0x1d, 0x1b, 0x17, 0xf	
Pentagram	5	11111,11111, 11111,11111, 11111	0x1f, 0x1f, 0x1f, 0x1f, 0x1f	
3star	6	111010, 110101, 101011, 010111, 101110, 011101	0x3a, 0x35, 0x2b, 0x17, 0x2e, 0x1d	
Bowtie	6	110110, 101101, 011011, 110110, 101101, 011011	0x36, 0x2e, 0x1b, 0x36, 0x2e, 0x1b	
Chevron	6	111110, 111101, 111011, 110111, 101111, 011111	0x3e, 0x3d, 0x3b, 0x37, 0x2f, 0x1f	
Arrow	7	1101110, 1011101, 0111011, 1110110, 1101101, 1011011, 0110111	0x6e, 0x5d, 0x3b, 0x76, 0x6c, 0x5b, 0x37	

Figure 7.9: Polygon Identification and Matching.

By marking objects with polygon patterns having non-repeating encoding (such as the Chevron pattern), it became trivial to identify known objects with any orientation. These patterns were applied to the sides of the target bricks to allow automated recognition.

7.4 Rapid Image Segmentation

In prior implementations of the Snake algorithm a full 24-bit color image was used and a match quality function was applied just within a small neighborhood of the contour. The image function used a complex mapping of RGB space into HSV (Hue-Saturation-Value) space in order to detect match conditions for pixels that emphasized hue over saturation and value, and required 10 floating point operations per measurement.

This approach had limitations. Defining the delta neighborhoods to get good match/no-match segmentation was a hit-or-miss process. For images with lots of snakes or long contours the excessive floating point computation could get burdensome, and most importantly, this approach did nothing to help with seeding new snakes. For a PTU system that is scanning, the ability to recognize potential regions for growing a snake was critical.

To overcome these issues we developed two new image processing modules within the Umbra framework. The first module, the *imageBlobber*, coverts a 24-bit RGB image into a palletized color image. This is done using a table lookup function that is extremely fast.

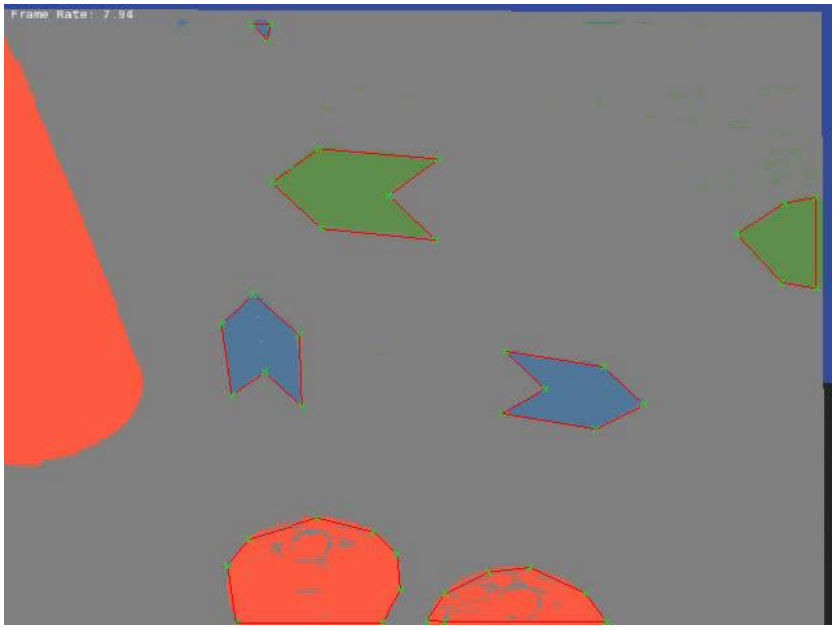


Figure 7.10: Image Output from imageBlobber

The trick is in segmenting the image color cube into palette regions that provide the appropriate matches. Color regions are defined using a 10-sided convex polyhedra in RGB space which can be generated by simple inequalities. To define a color match region, the developer aims the camera at a color feature to be monitored and takes samples under various lighting conditions by clicking on matching pixels in a sampled image. The convex hull of all of the match samples is used to define the color region. Once the color tables are defined, live video can be plugged into the module, and a much

simpler palletized color view is generated as an output using the resulting table lookups. (See Fig 7.10)

In addition to isolating regions of the desired color, the *imageBlobber* also deploys a rapid contiguous region growing algorithm, which provides as an output the centroid, size, and maximal extents of each color blob.

7.5 Autonomous Model Building

The Snake algorithm, combined with the *imageBlobber* gave excellent feature points for object tracking, and helped us achieve our objective of object recognition and alignment with centimeter accuracy. By combining these visual process techniques with the command sequences, we were able to "automate" the recognition and model building of a wall of bricks.

Figure 7.11 shows the process.

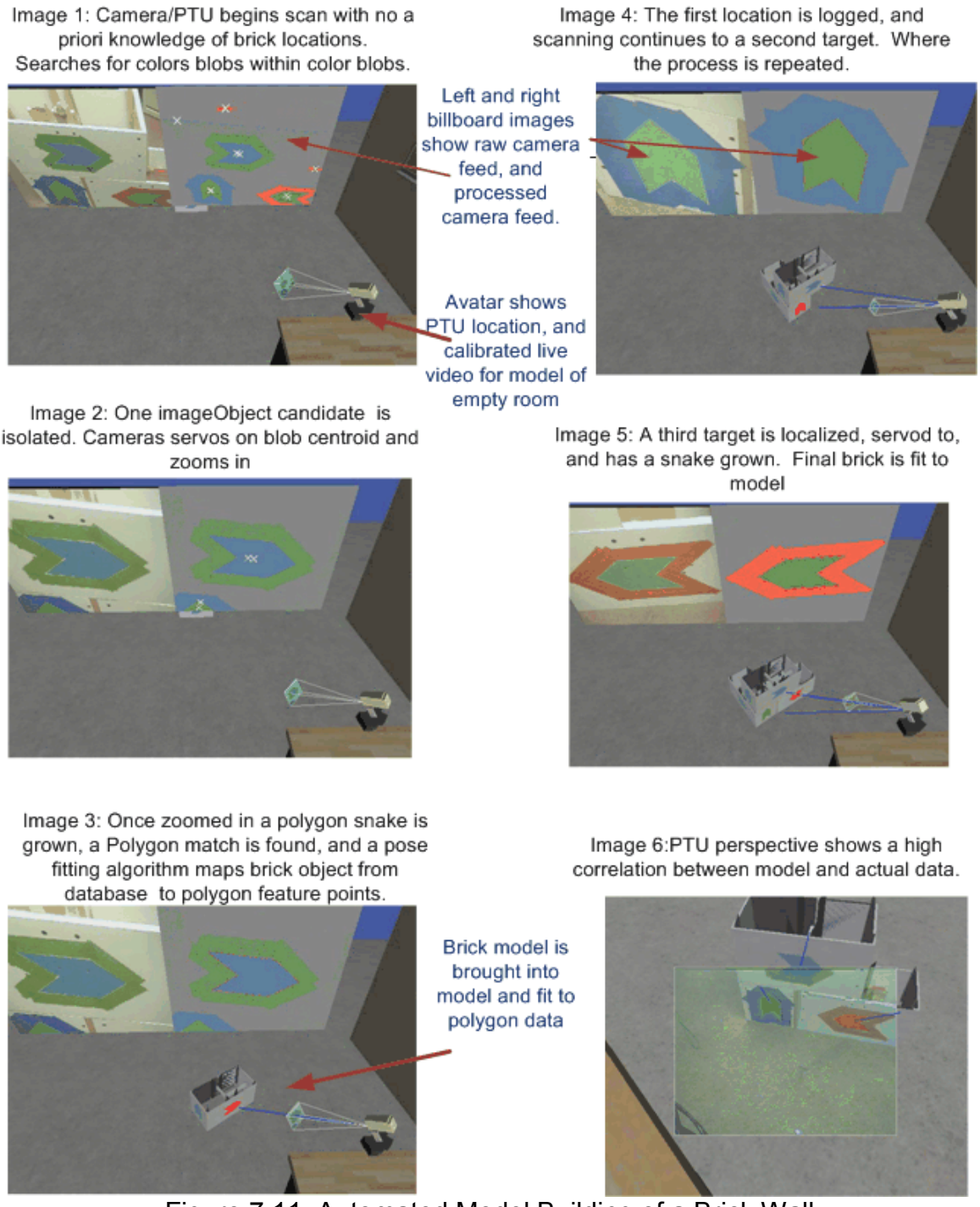


Figure 7.11 Automated Model Building of a Brick Wall

7.6 Issues with Polygon Tracking

The polygon snakes and image segmentation techniques developed within the second year were a big improvement over the first year‟s efforts. When used together, they were able to successfully scan, identify, and localize objects with known features with high precision and minimal computational overhead. They were not entirely without problems however.

A primary issue was that noise in the image and/or image distortion could cause a polygon to have more feature corners detected then actually existed. A sharp corner might be slightly rounded, or a long linear edge may split into two nearly co-linear segments. This would cause the simple matching techniques to fail. Furthermore, if the camera or vehicle was moving, then interlacing in the video feed and motion blur could greatly disrupt the input signal. Certainly the polygon representation could still be used to seed a rapid correlation match search, but this was far less efficient.

Occlusion was also a problem. If only a partial segment of a polygon was visible then the algorithm would fail to correctly identify the polygon feature. This problem could be mitigated by using zoom controls and centering on the color features before growing snakes, but this was time consuming.

Finally, color blending would affect the accuracy of the corner point measurements. Within a camera CCD, the colors on the border of a region have their perceived color altered by the colors of their neighboring pixels. This would make a color region always appear slightly smaller than its actual size, which would induce fitting errors. By zooming in to the target before growing a snake, this effect could also be minimized, but this was also time consuming.

In summary, although the approach worked for recognizing and localizing objects with know color regions, small errors were accumulating, and the resulting system robustness was less than desired.

7.7 Color Grid Pattern Matching

In the third year of the LDRD, we developed yet another approach for object marking and tracking that overcame the shortcomings of the polygon snakes. The output of the *imageBlobber* module was highly immune to the two problems plaguing the snake algorithm: image disruption due to motion, and color blending.

The *imageBlobber* directly and rapidly generates both a visual output of palletized color but also a list of each color blob, along with its centroid, size in pixels, and maximal values in UV coordinates. The centroid of a color blob is relatively immune to color blending and the disruptive effects of motion blur and vertical scan interlacing.

We decided to take advantage of this, by generating calibration targets that contained random color dot patterns arranged on a grid. A new module, the *imageColorGrid* module took the output of the *imageBlobber*, and searched for regular grid patterns in the data. These grid subsets could then be compared to stored grid patterns to find a unique match. Although the *imageBlobber* didn"t have the tracking features directly that we needed for motion servoing, the *imageColorGrid* pattern did.

The random colored grid patterns had other advantages. There were no partial occlusion issues. Only a small subset of the grid was needed for unique identification of the grid. This made the grid a great target for calibrating zoom camera systems. The same grid could be used for a large number of zoom settings.

Figure 7.12 shows the color grid matching technique being applied within an office setting.

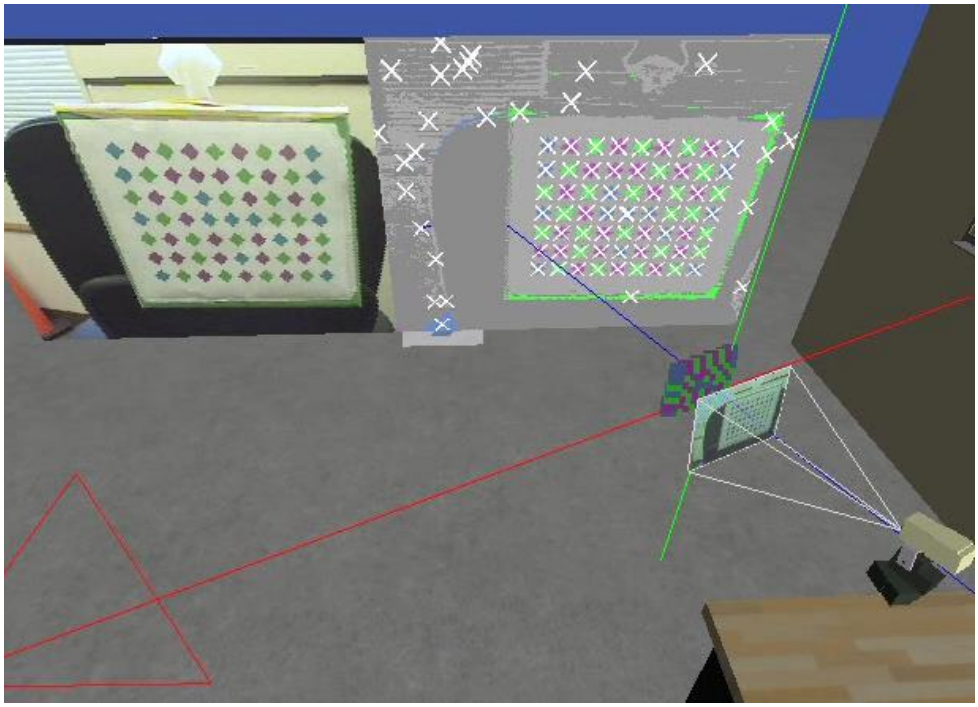


Figure 7.12: Matching and Tracking Color Grid Patterns

7.8 Image Processing Conclusions.

The polygon snakes, image segmentation techniques, and color grid approaches are all important steps in obtaining good robust recognition of known objects using vision. The polygon snake approach, itself, has deemed suitably unique to justify a patent application. (Submitted to US Patent Office, pending as of Sept. 17, 2009). The color grid approach has become a major feature of camera and robot calibration, and helps support on-going work in visual targeting.

The remaining processing algorithms have been incorporated into a copywritten Umbra add-on package called *imageApps*. The complete set of image processing modules developed by this LDRD are listed in Table 7.1 below.

Unfortunately, the image processing algorithms developed so far still fall short of the requirements needed for autonomous robot assembly. If known targets are viewed from poor angles they won"t be recognized. Large changes in lighting conditions can still disrupt the algorithms. The approach deployed searches for matches to known objects with distinctive markings, but is still unable to assess the unknown.

Table: 7.1 Umbra Modules For Image Processing.

Command Name	Description
imageAverage	Performs a moving average of last N frames
imageBlobber	Converts RGB color to pallette color to isolate colors of interest. Peforms blob region growing on the result.
imageBlobPicker	Tracks given color-in-color combination, and provides input for camera tracking.
imageColorGrid	Takes output from imageBlobber, and searches for exact matches to a color grid pattern used for camera calibration.
imageCrop	Simple Region-of-Interest reduction of an image.
imageDifference	Takes two images as an input and computes their difference
imageHalve	Simple down sample of an image
imageSegmentation	Original version of imageBlobber. It uses a less-efficient Hue-Saturation distance to determine region membership.
imageSnake	Implements the polygonSnake algorithm. Takes palette color image from imageBlobber module and grows and tracks snake objects.
imageSnakePicker	Tracks outputs of the imageSnake module and isolates snakes that match a desired goal object.
imageUVMonitor	Tracks the delta changes in U and V values for a tracked grid object.

8.0 Conclusions

The Autonomous Assembly LDRD set out to demonstrate robot assembly operations in unstructured environments, and failed to achieve its final objective. No robot ever assembled a brick wall as initially intended, nor did one get particularly close. Along this path of shattered dreams, however, there were a number of successes.

There were significant advances in visual processing for live video in unstructured environments. The polygon snake approach provided tracking, accuracy and object recognition. A patent for polygon snakes has been filed. The *imageBlobber* provides rapid image segmentation, the color dot grid tracking provides a robust means for calibrating and tracking. A new library of image processing code (*imageApps*) has been copy written and is now a package available within the Umbra framework.

A command and control structure has been developed that enables sophisticated autonomous scripting. Commands can be declared and defined simply by parsing an XML file. Higher level commands can then be chained together from simpler commands to create ever more advanced behaviors. Voice-direction has been investigated as a means to direct outdoor mobile robots, and provides another means to interact with robot systems.

A communications framework was developed that enabled complex interactions between multiple targets. This framework has greatly simplified how distributed robots communicate and how advanced systems can be built. It has already been used to support multiple projects at the robot vehicle range.

All of these software technology components have been copy written and added to our code base as registered intellectual property. These tools have already helped to advance current projects and has helped to attract new funding for visual targeting and advanced telemanipulation.

Scanning code has been developed, and two different commercial scanners have been thoroughly investigated. Limitations of these scanning technologies have been identified, and software interfaces have been developed that can process the data.

Demonstrations for X-Prize shows, Family Day, multiple technical conferences, and the Border commission were given and were well received. Finally, we now have a series of permanent demonstrations showing autonomous components.

Work continues, and will likely continue for decades to come, in the area of providing autonomy for mobile robot systems. This LDRD helped develop many of the tools that can and will be deployed in future systems.

Intentionally Left Blank

Appendix A: Glossary of Terms Used

Autonomous Navigation
Point-to-point vehicle traversals without human intervention.

Canesta
A manufacturer of 3D Range imagers using time-of-flight pulses of infrared light.

CCD
Charge-Coupled Device. A common technology used for capturing a digital image, i.e., a CCD camera.

Color Spaces
Digital representations of color, e.g., RGB or HSV.

FOV
Field-of-View. A measurement in degrees of the viewing angle of a camera.

GUI
Graphical User Interface – A user interface on a computer that an operator interacts with using a pointing device such as a mouse or touchscreen.

HAGAR
High Agility Ground Assessment Robot – A Sandia developed version of the RATLER dual –body platform.

HSV
Hue-Saturation-Value. A representation of color used for digital imagery..

IncrTcl
A object oriented extension of Tcl. It expands the standard behavior of Tcl in much the same way that C++ extends the C programming language.

IED
Improvised Explosive Device – a makeshift bomb.

JAUS
Joint Architecture for Unmanned Systems a communications framework developed by the U.S. Dept of Defense for controlling unmanned robot systems.

Inertial Measurement Unit (IMU)
a device which typically measures rates that describe changes of the six degrees of freedom (3 rotation, 3 translation) of a moving entity with

respect to a local level (inertial) system. Integration of these measurements can produce position and attitude.

Kalman-Filter
A Bayesian Estimate filter that uses a prediction-correction stage for estimating robot position and orientation from multiple noisy sensor readings.

GHz Analog Radio
Giga-Hertz radio. Typically used for transmitting NTSC video. Old analog radios suffered from multi-path and signal noise.

GPS
Global-Positioning System.

MSDOS
Microsoft Disk Operating System. An early operating system.

Novatel
A vendor of GPS units with high accuracy.

PTU
Pan-Tilt Unit. A typical two-stage control unit for moving a camera head.

RATLER™
Remote All- Terrain Lunar Explorer Robot. A multi-purpose research dual-body robot developed by Sandia's Robotics Group. (http://www.sandia.gov/isrc/sadrat.html)

RGB
Red-Green-Blue. A representation of color used for digital imagery.

RPC
Remote Procedure Call -- A interprocess communication protocal that allows a computers in a network to initiate routines on another computer.

SMART
Sandia's Modular Architecture for Robotics and Teleoperation. A modular control framework for building telerobotics control systems. (http://www.sandia.gov/isrc/SMART.html)

SmartBroadcast
A software package developed in this project that enables multiple computing elements to transfer information in real-time utilizing broadcast UDP.

SmartCmd
An abstract software entity developed in this project that encapsulates the essence of a execution element.

Spread
An open source toolkit for high-performance messaging. (http://www.spread.org)

TCP

Transmission Control Protocol. A core internet protocol of the internet.

Tcl

Tool Command Language. A general scripting language similar to python. (http://www.tcl.tk).

Teleoperation

The operation of a remotely located robot system via operator input devices such as joysticks and spaceballs.

Turing

A 6-DOF mobile research robot on a skid-steer base with a gripper and calibrated cameras.

UDP

User Datagram Protocol. A core internet protocol of the internet. Uses minimal handshaking compared to TCP..

Umbra

A Sandia developed framework for software development. (http://www.sandia.gov/isrc/UMBRA.html)

voxel

A volume element similar to a pixel, but representing a 3-dimensional element.

XML

Extensible Markup Language is a set of rules for encoding documents electronically.

References

Anderson, Robert J.“*Cooperating Robots For Improvised Explosive Device Defeat*”, American Nuclear Society Topical Meeting on “Emergency Management & Robotics for Hazardous Environments”, March, 2008.

Anderson, Robert J, "SMART: A Modular Control Architecture for Telerobotics", IEEE Robotics & Automation Magazine, Sept. 1995, pp.10-15.

Curless, Brian and Marc Levoy, “*A Volumetric Method for Building Complex Models from Range Images*”, SIGGRAPH '96, (New Orleans). In Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 303-312. August 4-9, 1996.

Gottlieb, Eric, et.al. The Umbra Simulation Framework, Sandia Internal Report, SAND2001-1533. June 2001.

Eisler, G. Richard “*Robust Planning for Autonomous Navigation of Mobile Robots in Unstructured, Dynamic Environments: An LDRD Final Report*”. SAND2002-2596, August 2002.

Klarer, P.K., “A Highly Agile Ground Assessment Robot (HAGAR) for Military Battlefield and Support Missions, SAND 94-0689C, March 1994.

Lorensen, William E. & Harvey E. Cline, “*Marching Cubes: A high resolution 3D surface construction algorithm*”, Computer Graphics, Vol. 21, Nr. 4, July 1987)

Padilla, Denise D., “*Obstacle Detection for Autonomous Navigation: An LDRD Final Report*”, SAND2004-1173, March 2004.

Schaub, H. and C. Smith, “*Color snakes for dynamic lighting conditions on mobile manipulation platforms*,” In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.

Turk, Greg and Marc Levoy, Proc. *“Zippered Polygon Meshes from Range Images”, SIGGRAPH '*94 (Orlando, Florida). In Computer Graphics Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH, pp. 311-318. , July 24-29, 1994.

Distribution:

2	MS1125	Robert J. Anderson	6472
1	MS1188	Fred Rothganger	1434
1	MS1188	Dan Morrow	1432
1	MS1188	Ralph Peters	1434
1	MS1125	Jake Deuel	6472
1	MS9018	Central Technical File	8945-1
2	MS0899	Technical Library	9616
1	MS0188	LDRD Office	1030


Sandia National Laboratories