Final Report
Autonomous Cross Country Navigation Technologies

submitted to

Defence Research Establishment, Suffield

by

Field Robotics Center
Robotics Institute
Carnegie Mellon University

Last Revision: 21 May 96
Executive Summary

This document reports on the work done on the research program titled “Autonomous Cross Country Navigation Technologies”. This program was a part of a collaborative effort between Boeing Inc, the US Military Tank Automotive Command (TACOM), Defence Research Establishment, Suffield, and Carnegie Mellon and has been named the “Transitional Unmanned Ground Vehicle” (TUGV) program.

The main motivation behind this project was to evolve sensors and algorithms for autonomous cross country operation closer to military applications. To this extent we sought sensors that were quiet, robust and had the possibility of enabling high speed navigation. Additionally, we sought to port the algorithms to Ada running on military spec computing hardware. The program was planned in three phases starting with implementation of the sensors and algorithms on a workstation mounted on the vehicle and culminating in a system completely resident on a military spec “super computer”.

We have demonstrated autonomous operation of our HMMWV in a cross country setting. The final system used stereo vision and previously developed navigation software to achieve speeds in the order of 5 miles per hour. The vehicle successfully steered around obstacles and was able to track designated paths over an extended period of time. The results are noteworthy since this program is one of a few programs in the world that have accomplished outdoor navigation using stereo vision. Also, the program is the first known to have used high resolution digital cameras to navigate autonomously. We were not successful in porting the algorithms to the military spec super computer because of delays in delivery of hardware and incompleteness of the software environment to port algorithms.

Our experience suggests an agenda for the future. Perhaps the most important issues to improve performance are to obtain a larger horizontal field of view and to improve the resolution of the depth computation. This process will require modification of algorithms to deal with wider field of view lenses and will require faster processors than we have used to date. The advantages are that we expect improved vehicle speed and an improved ability to resolve small objects in the vehicle’s path. Additionally, we suggest that the work we have done to date is a natural stepping stone to other tasks such as strategic planning where the vehicle is able to recognize dead-ends and recover from them in order to autonomously traverse from a starting point to a goal over a considerable distance.
# Table of Contents

1. **Introduction** 1  
   1.1 The Program 1  
   1.2 Project accomplishments 2  
   1.2.1 Developments relevant to Phase I 2  
   1.2.2 Developments relevant to Phase II 2  
   1.2.3 Developments relevant to Phase III 3  
   1.3 This document 3  

2. **Architecture** 5  
   2.1 Phase I 5  
   2.2 Phase II 6  
   2.3 Phase III 6  

3. **Navigation Software** 8  
   3.1 Overview of Ranger 8  
   3.1.1 Operational Modes 8  
   3.1.2 Goal-Seeking 8  
   3.1.3 World Model 8  
   3.1.4 Vehicle Model 9  
   3.1.5 Hazard Assessment 9  
   3.1.6 Arbitration 10  
   3.1.7 Sensors 10  
   3.1.8 Adaptive Perception 11  
   3.2 Software Design of Ranger 11  
   3.2.1 Position Estimator 11  
   3.2.2 Map Manager 12  
   3.2.3 Vehicle 12  
   3.2.4 Controller 13  
   3.2.5 Perception 13  
   3.3 Software Design of Ranger 14  
   3.3.1 Software architecture 14  
   3.3.2 Main 15  
   3.3.3 Vehicle Model 15  
   3.3.4 Vehicle State Queue 15  
   3.3.5 Position Estimator 16  
   3.3.6 Terrain Map 17  
   3.3.7 Map Manager 18  
   3.3.8 Vehicle Command Queue 18  
   3.3.9 Controller 19  
   3.3.10 Tradeoffs 20  

4. **Image Processing** 21  
   4.1 Stereo Vision Algorithm 21
4.1.1 Data Flow 21
4.1.2 Range Images from Stereo Vision 23
4.1.3 Horizontal vs. Vertical Baseline 24

4.2 Rectification of Stereo Images 28
4.2.1 Non-Rectified Image Pair 29
4.2.2 Rectification Matrix 29
4.2.3 The Program easy_rect and The Result 31

4.3 Camera Calibration (Correction of Lens Distortion) 32
4.3.1 Deriving Distortion Parameters 34
4.3.2 Deciding when distortion correction is required 36

4.4 Image Processing on the MCC 39
4.4.1 Computing the Image Slice Dimensions 40
4.4.2 Processing the Image Slice 44

5. A Design for a Generic Stereo Based Autonomous Vehicle 48

5.1 Perception Sensor Requirements 48
5.1.1 Response Distance 48
5.1.2 Minimum Range 49
5.1.3 Maximum Range 49
5.1.4 Response Angle 49
5.1.5 Horizontal Field of View 50
5.1.6 Vertical Field of View 50
5.1.7 Sweep Rate 51
5.1.8 Angular Resolution 52
5.1.9 Stereo Angular Resolution 53

5.2 Positioning Sensor Requirements 53
5.2.1 Path Following 53
5.2.2 Obstacle Avoidance 54

5.3 Computing Requirements 54
5.3.1 Theoretical Requirements 55
5.3.2 Empirical Requirements 55

5.4 Design 58
5.4.1 Upgrade Paths for the Image Capture Block 58
5.4.2 Electrical Design Details 60
5.4.3 Mechanical Design Details 62

6. Conclusions and Future Work 64
List of Figures

Figure 1  The TUGV vehicle  1
Figure 2  A short cross country excursion.  3
Figure 3  Phase I architecture  5
Figure 4  Phase II architecture  6
Figure 5  Phase III architecture  7
Figure 6  Terrain Map  8
Figure 7  State Space Vehicle Model  9
Figure 8  Hazard Detection and Avoidance  10
Figure 9  Adaptive Perception  11
Figure 10  System Data Flow  12
Figure 11  The principle of Stereo Vision  21
Figure 12  Data flow for Stereo Vision  22
Figure 13  Adaptive Stereo.  26
Figure 14  Vertical Baseline Stereo.  27
Figure 15  Adaptive Vertical Baseline Stereo.  28
Figure 16  Non-Rectified Image Pair (Top Camera & Bottom Camera)  30
Figure 17  easy_rect demo  32
Figure 18  Rectified Image Pair (Top & Bottom)  33
Figure 19  The Grid (distorted)  34
Figure 20  Four step transformation (as shown in Roger Tsai’s paper)  35
Figure 21  The Grid (distortion corrected)  37
Figure 22  Undistorted to Distorted coordinates mapping  37
Figure 23  Organization of the Mission Computing Cluster (MCC)  39
Figure 24  Upper and Lower Regions of Interest (ROI)  41
Figure 25  Inverse affine of ROI and new ROI bound  43
Figure 26  Image data ROI including modulo adjustment  43
Figure 27  Relative Placement of the Camera Images  45
Figure 28  Division of the Image Data among AP’s  45
Figure 29  Generic Response Distance  48
Figure 30  Response Angle  50
Figure 31  Vertical Field of View  51
Figure 32  Vertical Field of View  51
Figure 33  Sweep Rate  52
Figure 34  Vertical Pixel Spacing  53
Figure 35  Throughput for All Algorithms  56
Figure 36  The proposed design  60
1. Introduction

1.1. The Program

This is a report on a collaborative research program to develop unmanned cross country navigation for the vehicle shown in Figure 1. The program was jointly funded by Boeing Corporation and the Defence Research Establishment Suffield (DRES) during the period Dec 1994 to March 1996.

Figure 1. The TUGV vehicle. The vehicle is a retrofitted military ambulance with onboard computers, power generation, inertial and vision sensors. The steering, throttle and brakes are computer controlled.

The motivation of the Transitional Unmanned Ground Vehicle (TUGV) program has been to demonstrate unmanned navigation on military spec hardware. The eventual goal was to site navigation software (written in Ada) on a Mission Computer Cluster (MCC) developed by Boeing. The main sensing modality for navigation is intended to be stereo vision. The program was designed in three phases.

- **Phase I**: development of a low performance system using stereo vision and navigation software (written in C) running on a workstation. The speed of the vehicle is limited by the ability to process stereo images.

- **Phase II**: Porting of stereo vision to the MCC in the form of optimized VRTL code. This was intended to allow fast processing of high resolution images. This phase is intended to highlight the development of the real-time stereo system.
• **Phase III**: Porting of navigation software to the MCC (in Ada) and demonstration of combined system.

### 1.2. Project accomplishments

We have demonstrated autonomous operation of our HMMWV in a cross country setting. The final system used stereo vision and previously developed navigation software to achieve speeds in the order of 5 miles per hour. The vehicle successfully steered around obstacles and was able to track designated paths over an extended period of time. Below, we list the developments as they relate to each phase of the program.

#### 1.2.1. Developments relevant to Phase I

Our program has met the following milestones:

- retrofit and repair of the HMMWV testbed
- installation of jig for stereo vision
- incorporation of digital cameras and frame grabbers
- incorporation of stereo vision algorithms into navigation software
- development of lens calibration and software correction for misaligned optical axes.
- installation and incorporation of an Inertial Navigation System (MIAG).

Figure 2. shows range data computed from a typical run of Ranger. It also shows the topological map of the terrain that is built from the range data collected during an outdoor run.

#### 1.2.2. Developments relevant to Phase II

Many components of the MCC have been delivered very late in the program and both Phase II and Phase II have been delayed on this account. We have met the following milestones:

- developed software to run on the array processors on the MCC.
- developed timing tests for stereo vision on the MCC.
- developed software and hardware interfaces between the MCC and other devices such as the Sensor Interface Unit (SIU) and the sparc.
1.2.3. Developments relevant to Phase III

We have developed a detailed design of the navigation software and are close to implementation on a workstation. The methodology we have used is to understand the requirements, and do formal testing of the design before the code is written. In this quarter we expect to produce a detailed design for RANGER in Ada. At the point the inputs are obtained from file and written to the screen. In a future program, this code will be tested on the MCC.

1.3. This document

The rest of this document is divided into five chapters:

Chapter 2  
*Architecture:* Describes how the computers, sensors and actuators are organized for each phase.

Chapter 3  
*Navigation Software:* gives details of the software used to control the vehicle.

Chapter 4  
*Image Processing:* gives details of the software used process the images from the stereo cameras.

---

Figure 2. A short cross country excursion. (a) shows a sequence of range images from a stereo vision system mounted on the vehicle. (c) shows a sequence of reflectance images from one of the cameras. (b) and (d) show an overhead view of an elevation map that was generated.
Chapter 5  

A Design: this provides a preliminary design for a stereo vision system to be implemented on a generic cross country vehicle including requirements for sensing and computing.

Chapter 6  

Conclusions: this suggests the important lessons learned from the program and makes recommendations for the future.
2. Architecture

This chapter show the three designs for the architecture of the TUGV computing and sensing.

2.1. Phase I

In Phase I, the main computation is sited on a single workstation (Sparc 20) as shown in Figure 3. Images from the digital cameras are processed on this machine into range maps and then converted into topological (terrain) maps. These are then used by the navigation software (Ranger) to determine steering angle and speed for the vehicle. Speed and Steering angle are communicated to the vehicle controller which servos the actuators to achieve the desired commands.

Figure 3. Phase I architecture
2.2. Phase II

In the Phase II architecture, stereo processing is shifted onto the MCC while the navigation software is still resident on the sparc workstation as shown in Figure 4. Use of the MCC requires that the digital cameras be hooked up to the Sensor Interface Unit (SIU). This delivers the image data to the MCC array processors. The stereo code runs on the array processors and is written in the VRTL microcode. Range data from the array processors is merged (parts of the array processor work on parts of the incoming images) into one by the data processor on the MCC and shipped to the Sparc which then produces commands for the vehicle controller.

![Figure 4. Phase II architecture](image)

2.3. Phase III

In Phase III all processing occurs on the MCC as shown in Figure 5. The navigation software runs on the Data Processor part of the MCC and directly produces commands for the vehicle controller. In addition, debug information is sent to the Sparc where it can be displayed. This is done so that much of the debugging software that is already available on the sparc does not have to be duplicated on the MCC.
Figure 5. Phase III architecture
3. Navigation Software

3.1. Overview of Ranger

RANGER is an acronym for Real Time Autonomous Navigator with a Geometric Engine. The system is an autonomous vehicle control system which specializes in high speed driving in rugged cross country environments. It has evolved from earlier work on the same problem at CMU and from the original work on the Autonomous Land Vehicle at CMU and at Hughes Aircraft Corp a decade ago.

Ranger has navigated over distances of 15 autonomous kilometers, moving continuously, and has at times reached speeds of 15 km/hr. The system has been used successfully on a converted U.S. Army jeep and on a specialized Lunar Rover vehicle.

3.1.1. Operational Modes

The system can autonomously seek a predefined goal or it can be configured to supervise remote or in-situ human drivers. The predefined goal may be a series of points to visit, a continuous path to follow, a compass heading or a path curvature to follow.

3.1.2. Goal-Seeking

The system can follow a predefined path while avoiding any dangerous hazards along the way or it can seek a sequence of positions or a particular compass heading. In survival mode, seeking no particular goal, it will follow the natural contours of the surrounding terrain.

3.1.3. World Model

A computerized terrain map data structure is maintained which models the geometry of the environment. It is an array of elevations that represents the world as a 2-1/2 D surface where the vertical direction is aligned with the gravity vector. This representation, combined with a model of vehicle geometry, permits a robust assessment of vehicle safety.

Figure 6. Terrain Map
3.1.4. Vehicle Model

The system is based on a tightly-coupled, adaptive feed-forward control loop. It incorporates measurements of both the state of the vehicle and the state of the environment and maintains high fidelity models of both that are updated at very high rates.

At sufficiently high speeds, it becomes necessary to explicitly account for the difference between the ideal response of the vehicle to its commands and its actual response. The vehicle is modelled as a dynamic system in the sense of modern control theory. Although the system uses a nonlinear model, the linear system model expressed in the following generic block diagram provides a sense of the important signals and transformations involved.

FIFO (First-In, First-Out) queues and time tags are used to model the delays associated with physical i/o and to register contemporary events in time. The command vector \( u \) includes the steering, brake, and throttle commands. The disturbances \( u_d \) model the terrain contact constraint. The state vector \( x \) includes the 3D position and 3 axis orientation of the vehicle body as well as its linear and angular velocity. The system dynamics matrix \( A \) propagates the state of the vehicle forward in time. The output vector \( y \) is a time continuous expression of predicted hazards where each element of the vector is a different hazard.

3.1.5. Hazard Assessment

Hazards include regions of unknown terrain, hills that would cause a tip-over, holes and cliffs that would cause a fall, and small obstacles that would collide with the vehicle wheels or body.

The process of predicting hazardous conditions involves the numerical solution of the equations of motion while enforcing the constraint that the vehicle remain in contact with the terrain. This process is a feed-forward process where the current vehicle state furnishes the initial conditions for numerical integration. The feed-forward approach to hazard assessment imparts high-speed stability to both goal-seeking and hazard avoidance behaviors.
System components above the state space model in the software hierarchy translate the hazard signals $y(t)$ into a vote vector. This is accomplished by integrating out the time dimension to generate a vote for each steering direction based on a normalization of the worst case of all of the considered hazards.

In the figure, the system issues a left turn command to avoid the hill to its right. The histograms below represent the votes for each candidate trajectory, for each hazard.

Higher values indicate safer trajectories. The hazards are excessive roll, excessive pitch, collision with the undercarriage, and collision with the wheels. The tactical vote is the overall vote of hazard avoidance. It wants to turn left. The strategic vote is the goal-seeking vote. Here it votes for straight ahead.

**3.1.6. Arbitration**

At times, goal-seeking may cause collision with obstacles because, for example, the goal may be behind an obstacle. The system incorporates an arbiter which permits obstacle avoidance and goal-seeking to coexist and to simultaneously influence the behavior of the host vehicle. The arbiter can also integrate the commands of a human driver with the autonomous system.

**3.1.7. Sensors**

The navigator accommodates both laser rangefinder and stereo perception systems and it incorporates its own integrated stereo vision algorithm. In either case, the design achieves significant increases in vehicle speeds without sacrificing either safety or robustness.
3.1.8. Adaptive Perception

A moving vehicle generates images which contain much redundant information. Removal of this redundancy is the key to Ranger’s ability to drive a vehicle fast. A new range image perception algorithm has been developed that selectively extracts a very small portion of each range image in order to reduce the perceptual throughput to a bare minimum. In this way, vehicle speed is less limited by the computer speed.

The algorithm searches each image for a band of geometry that is between two range extremes, called the range window as shown in the figure. Only the data between the white lines is processed. The algorithm also accounts for vehicle speed by moving the range window out as speeds increase.

![White Lines Outline Range Window](Figure 9. Adaptive Perception)

This approach also stabilizes the sensor in software because the search for the data of interest adapts automatically to both the shape of the terrain and the attitude of the vehicle.

3.2. Software Design of Ranger

At the highest level, the system can be considered to consist of 5 modules as shown in the flow diagram: (Figure 10.)

3.2.1. Position Estimator

The **Position Estimator** is responsible for integrating diverse navigation sensor indications into a single consistent indication of vehicle state. Vehicle state information includes the positions of all actuators and some of their derivatives, and the 3D state of motion of the vehicle body. This module may be the built-in navigation Kalman filter or another system which generates the same output.
3.2.2. Map Manager

The Map Manager integrates discrete samples of terrain geometry or other properties into a consistent terrain map which can be presented to the vehicle controller as the environmental model. It maintains a current record of the terrain immediately in front of the vehicle which incorporates all images necessary, and which automatically scrolls as the vehicle moves.

The map manager forms an abstract data structure when combined with the terrain map itself. Images are transitory input flows at this level of conceptualization. Consumers read or write the map in navigation coordinates and are ignorant of the underlying transformation and scrolling operations.

3.2.3. Vehicle

The Vehicle object is both the control loop feedforward element and an abstract data structure which encapsulates the vehicle state. This module incorporates FIFO queues which store a short time history of vehicle states and vehicle commands. Old state information is required by the map manager in order to register images in space. The current vehicle state is used as the initial conditions of the feedforward simulation. Old commands are used in the feedforward simulation as well. This module also provides daemons which compute derived properties of vehicle state such as the planning range window and the maximum safe speed and curvature. Another set of daemons is used to convert curvature, steer angle, and turn radius to any of the other two based on the vehicle state and the wheelbase.
3.2.4. Controller

The **Controller** object is responsible for coordinated control of all actuators. This module includes regulators for sensor head control, an obstacle avoidance tactical controller and a path following strategic controller. It also incorporates an arbitrator to resolve disagreements between the latter two controllers.

3.2.5. Perception

The **Perception** module is responsible for understanding or interpreting input images and putting them in a form suitable for the Map Manager to process. Examples of perceptual preprocessing include stereoscopy (stereo vision) which computes range from two or more images taken from disparate viewpoints, and terrain-typing which labels each pixel in an image as rock, road, shrubbery or tree. Stereo vision is the only perception that is currently supported.
3.3. Software Design of Ranger

3.3.1. Software architecture

RANGER mainly consists of three abstract data types (ADTs). The three ADTs are the Vehicle Command Queue, the Vehicle State Queue, and the Terrain Map. Each ADT has a particular data manager which performs the functions to maintain its data. In addition, the Map Manager and Controller make use of the data included in the Vehicle Model. This is detailed in the following figure, which shows the overall software architecture for RANGER.

![Software Architecture Diagram]

The components of the architecture are connected primarily via method invocation. The ADT data managers have direct access to the ADTs. Other components access the ADT through method invocations.

The following sections provide a high-level overview of the components of the RANGER software. Where appropriate, references to the original C source code are given. Note that where types are given, they are suggested abstract types, not specific implementation type decisions.

With regard to C code (RANGER/C), there are several features which are not needed for TUGV ADA version of RANGER. These features were not implemented in the ADA version. The features are:

- Discretes: This refers to a particular method of posting fixed obstacles in the terrain map. RANGER is
not concerned with discretes, and any code or data elements of RANGER/C dealing with discretes shall be omitted.

- **Commandable Heads**: This refers to sensors which are mounted on swiveling or otherwise movable heads. RANGER is only concerned with fixed cameras.

- **Parameter files (*.p)**: RANGER/C uses a run-time parser called “Tiny-C” which is able to read in parameter files and bind values to certain variables at run-time, without recompilation of the software. There is no equivalent facility in Ada, and any parameters set by Tiny-C shall be added to the actual RANGER code and set with hard-coded values at start-up time. This will be implemented using a globals package and an initialization method invoked during program start-up.

- **X library calls**: This refers to RANGER/C interfaces with “X” graphics libraries for debugging purposes. These calls and routines shall not be ported. An input/output package will be implemented to encapsulate i/o to the environment. Currently this package is setup for ascii i/o to screen and file.

### 3.3.2. Main

The Main routine performs initialization routines and the main program loop. In the event that a catastrophic equipment or algorithm failure occurs, Main must issue an emergency stop command to the vehicle, and then exit. The exact reaction has not been implemented in the ADA version of RANGER. Hooks for this and exception handling has been included. All methods include the ability to throw exceptions and a exception handler is included.

### 3.3.3. Vehicle Model

The Vehicle Model is the set of data which represents all relevant, permanent characteristics of the vehicle. Examples of such data are the wheelbase, the turning radius, the tire radius, and the distances from an origin (in the vehicle frame-of-reference) to all important points on the vehicle. All values in the vehicle model are fixed, and should not change at run-time. The vehicle model for RANGER is nearly identical in content to the vehicle model provided in RANGER/C, vehicle.h (struct Vehicle_Model_s). Note that “speed control” is not needed in RANGER; this refers to an extra option for fixed speeds that is not needed for TUGV.

### 3.3.4. Vehicle State Queue

The Vehicle State Queue is an abstract data type which holds the vehicle states. These states are retrieved and calculated from data which the vehicle’s global positioning system (MIAG) provides. The vehicle state in RANGER is similar to that in RANGER/C, vehicle.h (struct Vehicle_State_s).
Each state is timestamped to represent when it entered the system. When given specific times, the queue must be able to retrieve old states. The queue also must be able to estimate the vehicle states when it is given times that occurred between recorded states (i.e., the timestamps are used for interpolation between states).

Note that in RANGER/C, a coordinate transform is calculated and stored with every state. This is a matrix used to convert between vehicle coordinates for that state and true world coordinates. Since not every state is used, and many states are interpolated (and therefore do not have a transform stored anyway), it is questionable whether it is actually useful to do this. The transform has been left out of the RANGER vehicle state, as it can be calculated as needed from other state information.

Major functions for the vehicle state queue are:

- New state: Accept a new state.
- Most recent state: Return the most recent state.
- State at time t: Return the state for a given timestamp, possibly interpolating between known states.
- Range window: Return the range window for a given state. See RANGER/C, vehicle.c (VEH_GET_PLAN_WINDOW).
- Transform: Return the mathematical transform associated with a state. This can be calculated directly from the vehicle pose. See kinematics.h (KIN_COMPUTE_TRANSFORMATION_FROM_POSE).

### 3.3.5. Position Estimator

The Position Estimator is the RANGER module responsible for acquiring and saving vehicle state information. This module uses the positioning sensor hardware to acquire the state information, then inserts that data into the Vehicle State Queue.

In RANGER/C, a Kalman filter performs the position estimator function. The major function of the position estimator is to retrieve data from the position sensors, convert that data into a vehicle state, and put that state into the vehicle state queue.
3.3.6. Terrain Map

The Terrain Map is an abstract data type comprising an array of data about the topology of the world surrounding the vehicle. The map is a kind of “window” looking onto the world. Each entry in the map is called a cell, and represents the average elevation of a rectangular region of space within the world. When the vehicle first begins to travel, all the cells fit together, forming one cohesive window onto the world. However, as the vehicle travels beyond the boundaries of the map, it “wraps around” to the other side (left-to-right, top-to-bottom, etc.). When this happens, the cells where the vehicle has wrapped to now represent the region the vehicle then inhabits - not their original positions. The position in the world that a cell represents changes as the vehicle travels along. In other words, the map cells wrap around with the motion of the vehicle.

In order to avoid confusion, each cell contains data to indicate what region of the world its data currently represents. Particularly, each cell stores the distance from the center of its current region back to the origin. This distance is referred to in RANGER/C as the cell’s age. This gives RANGER a very simple way of determining whether to use or discard the data currently stored in a cell: if the distance the vehicle is from the origin, plus the distance from the vehicle to a given cell, is greater than the distance stored in that cell, it means the vehicle must have already traveled past (i.e. farther than) that area; hence that cell’s data must be out of date, and that cell is ignored for the current cycle.

This affects the way that data is both stored and retrieved from the map. When storing new data in a cell, the map can either discard old data every cycle, or it can attempt to determine if new data fits in the same region that the cell is already representing (and discard the data if it does not). RANGER/C refers to this as cell overlap, and has two options: frame overlap (throw out old data every cycle), or age overlap (throw out old data if a cell’s data has aged). Only one of these options will be used on a given run of RANGER, so the option can be set at initialization time. Frame overlap is the most commonly used option.

When data is retrieved from the map, the formula for age described above is used to determine whether a given cell’s data is good for the current cycle. If so, the data is used. If not, it is ignored. However, each cell also stores the cycle when any data was last inserted into that cell. The map contains a flag that indicates the maximum number of cycles that cell data should be allowed to age before being discarded as out-of-date. This prevents map data from becoming too old and possibly causing the vehicle to ignore new obstacles.

The C version of the terrain map can be found in RANGER/C, map.h (struct Map_s). The C version of the terrain map cells can be found in RANGER/C, map.h (struct Map_s). The actual data are scattered among several variable-length arrays whose head pointers are stored in the terrain map data structure.
3.3.7. Map Manager

The Map Manager is the RANGER module that takes range data, converts it into information about elevations at different points in the world, and then creates and maintains an elevation map of the terrain surrounding the vehicle. The Map Manager’s main function is convert range data into terrain map coordinates. Range data is in the form of triplets (x, y, r) where x and y represent pixel coordinates inside a frame of vision data, and r represents the distance from the vehicle to the nearest object from that pixel location. Range data is calculated by a stereo processing component from an image (or set of images) that the cameras on the vehicle obtain.

The major functions of the Map Manager are:

- **Range window computation**: The map manager tries to determine what the maximum and minimum interesting ranges are, based on the state of the vehicle when the image was obtained and the distance travelled since that time. Data for ranges outside the desired range can be discarded. This usually results in much greater performance for the map manager, as the percentage of uninteresting data is quite high. See map_mgr.c (MAP_MGR_COMP_RANGE_WIN).

- **Image to vehicle conversion**: Data comes into the map manager in the form of ranges. This is referred to as the image coordinate system. Data must be converted from image coordinates to vehicle coordinates. Vehicle coordinates are represented as x, y, and z, where x and y are longitudinal and latitudinal coordinates relative the vehicle, and z is the altitude of the point in the surrounding world described by x and y. Once the cameras are mounted on the vehicle, this transformation is fixed; each camera has its own slightly different transform. See map_mgr.c (MAP_MGR_CONVERT_PIXEL_COORDS).

- **Vehicle to world conversion**: Data in vehicle coordinates (which are relative to the vehicle’s position at a given time) must be converted into the world’s fixed coordinate system, so that they can be inserted into the terrain map. See map_mgr.c (MAP_MGR_CONVERT_PIXEL_COORDS).

- **Map update**: Data in world coordinates are inserted into the terrain map. See map.c (MAP_ACCUM_ELEV).

3.3.8. Vehicle Command Queue

The Vehicle Command Queue is an abstract data type which holds the commands which have been sent to the vehicle controller. The RANGER vehicle command is almost identical to the vehicle command in RANGER/C, vehicle.h (struct Vehicle_Command_s). Each command is timestamped for historical purposes. No command except the most recent is ever referenced by RANGER, so theoretically this queue could be only one element long. Note that in RANGER/C, the curvature, steering angle, and turn radius are all kept. These are redundant information from discussions with Robotics (only the curvature is needed in RANGER. Also, the brake controller provided for TUGV is an “all-or-nothing” hardware; the brakes are either on all the way, or they are not on at all. Usually, the brakes are used only for emergency stop situations.
Major functions of the Vehicle Command Queue are to accept new commands and to return the most recent command.

3.3.9. Controller

The Controller is the RANGER module responsible for sending steering and acceleration commands to the vehicle. In order to generate the commands, the controller uses data from the terrain map to evaluate the surrounding terrain, then tries to choose a travel path which will take the vehicle closest to some strategic goal, while still maintaining a minimum level of safety for the vehicle. The RANGER/C controller software is described by controller.c, starting with the function CNTRL_CMD_VEHICLE. The following are the major functions performed by the controller:

- **Initial arc generation:** The controller generates an array of equally-spaced steering curvatures which the vehicle could follow from its current pose. The number of arcs generated is fixed (CNTRL_GEN_TURN_CMDS).

- **Strategic vote generation:** The controller calculates the strategic value of each arc. This is a relative value between 1 and 0, indicating how close an arc brings the vehicle to satisfying its strategic goal. In RANGER/C, there are three types of strategic goal: a geographic point, a compass heading, and a relative vehicle curvature. The strategic goal is fixed for a given run of RANGER, since there is no way to feed in a new command once the vehicle is running. First, the strategic goal is calculated and then it is changed into a point goal for this cycle (CNTRL_RUN_SC_PART1). Second, the strategic vote value is generated for each arc (CNTRL_RUN_SC_PART2). After all strategic votes are generated, the arcs are sorted so that the highest (i.e. most strategic) arc comes first. Tactical calculations on the arcs are done in the order determined by the strategic vote values.

- **Path feed-forward:** The controller calculates the pose that the vehicle would be in if each of the potential arcs were followed. This is done by starting at the vehicle’s most current position (i.e. its most recent state) and “stepping” forward along the arc incrementally (CNTRL_DO_TRACKER_FEEDFORWARD). Each step represents an increment of time, forward from the current time. The amount of time stepped forward is dependent on vehicle speed, and so is calculated each cycle. See the variables total_time and dt, in CNTRL_GEN_TURN_CMDS.

- **Tactical vote generation:** At each point along each potential arc, the potential vehicle pose is calculated by “dropping” the vehicle virtually at that point (CNTRL_SIM_VEH_RESPONSE). Then the pose is evaluated for six safety criteria (CNTRL_COMPUTE_HAZARD_MERITS). The tactical vote for a given arc is equal to the lowest (i.e. most unsafe) value of any of the six criteria at any point along that arc.

- **Steering command generation:** Based on the strategic and tactical votes for each arc, controller picks the best steering command to follow (CNTRL_ARBITRATE). The algorithm starts with the most strategic arc, and proceeds in the sorted order until an arc is found whose tactical vote exceeds the minimum safety criterion. If no such arc is found, then there is no safe path for the vehicle, and an emergency stop command is immediately issued.
• Speed command generation: The controller generates a desired speed command to be issued along with the steering command. RANGER/C has two ways to do this: either a default speed command is always used (the “nominal” speed command) or a new speed command is generated each cycle (CNTRL_GEN_SPEED_CMD).

• Command execution: The controller stores the chosen command in the vehicle command queue, then passes the command to the vehicle control unit (VCU) for the vehicle (CNTRL_OUTPUT_CMD).

The following fixed values are used internally by the controller:

- [integer] Number_of_Arcs : The number of arcs calculated each cycle.
- [integer] Minimum_Safety_Factor : The minimum acceptable tactical value for an arc.

### 3.3.10. Tradeoffs

The major tradeoff in the RANGER system is processing time versus code simplicity. The RANGER software represents a real-time system with serious CPU and bandwidth constraints. Any code optimization which can significantly reduce CPU or bus bandwidth usage should be considered. On the other hand, RANGER code is so complex that significant changes may introduce defects that do not exist in any mirrored form in the original RANGER. This would significantly increase the difficulty of debugging and testing.

Since the C and ADA version of RANGER will be maintained in parallel, major structural changes were minimized to control software maintenance costs. It was desired that a person who understood the C version would be able to comprehend the ADA version with minimal effort and vice versa. As a result the sequencing of actions, method names and variable names were retained as much as possible.

Another major decision was driven primarily by the need to be independent of which ADA compiler was used and the complexity issues. The C version of RANGER is highly dependent upon pointers. The matching ADA concept is access types. The conflict is that ADA compilers (prior to ADA95) could implement access types and memory management schemes just differently enough to severely impact performance and reliability. To deal with the memory management issues, an internal custom memory manager would need to be added to the ADA version. (There currently exists one in the C version). This would significantly increase the complexity and is not a trivial undertaking. Due to these issues it was decided not to use access types. Recognizing that performance issues may force a move to access types, all ADTs include redundant methods such as isValid() which are required for an access type implementation. In the event that access types become necessary these methods can be easily modified.
4. Image Processing

This chapter reviews the area of processing images for the purposes of navigation. We have developed an extensive collection of tools for processing stereo images such that the robot vehicle can perceive depth as it navigates. This chapter details the stereo vision algorithm and explains how two important tasks (rectification of images and lens calibration) are performed.

4.1. Stereo Vision Algorithm

Stereo vision is a method of obtaining depth information for a robot. The idea is simple—two images of the world are taken from cameras that are separated by a known distance. A feature in the world will appear in different locations on the two image planes (Figure 11.). This difference (known as disparity) can easily be translated into a range to the feature in the world. The main problem in stereo vision is the trying to find correspondence between features in the two images. This is an expensive computation and thus requires specialized computing if high resolution depth is necessary at frequent intervals.

![Figure 11. The principle of Stereo Vision](image)

4.1.1. Data Flow

The implementation of correlation-based stereo vision is a straightforward series of transformations of the input image based on the techniques used at CMU and elsewhere in the past. The data flow diagram for this component is given below.
4.1.1.1. Scale and Normalize

This module replaces each pixel with its normalized value based on the mean and standard deviation of intensity over the correlation window. This operation enhances texture and removes localized bias from the image. An efficient moving average algorithm is used to compute the convolutions quickly.

4.1.1.2. Rectify

This module warps the input images into ideal epipolar geometry. This operation ensures that a match for a given region of the first image will be found along the same row (horizontal baseline) or same column (vertical baseline) of the second image.

4.1.1.3. Correlation

This module computes the correlation between each window in the first image and each window in the second image within the disparity image. An efficient moving average algorithm is used to compute the convolutions quickly.

4.1.1.4. Disparity

This module finds the disparity value at which the correlation curve for each pixel in the first image is maximum. Optionally, a subpixel disparity estimation routine computes subpixel disparity. Pixels whose maximum correlation scores are low or whose maxima are not sufficiently unique are marked as bad.
4.1.1.5. Cleanup

This module uses a classical blob-coloring algorithm to remove those connected regions in the disparity image whose overall size is a small fraction of the total image.

4.1.1.6. Triangulation

This module uses the camera model of the first camera to convert disparity to \((x,y,z)\) coordinates in the camera frame.

4.1.2. Range Images from Stereo Vision

In a typical image, the pixels that are actually processed by the adaptive perception algorithm form a jagged-edged band across the horizontal. The width of the band decreases quickly as the vehicle speed increases and adaptive look-ahead moves the window up the image. However, the validity of the small incidence angle assumption\(^1\) guarantees that adaptive perception will generate a perfect wedge of geometry which is exactly the requirement for the current planning cycle regardless of the vehicle attitude or terrain shape.

The following figure gives a sequence of range images for a run of the RANGER simulator on very rough terrain using a simulated rangefinder where the pixels that were actually processed fall between the thin black lines. On average, even in this worst case, only 200 range pixels out of the available 10,000 (or 2\%) were processed per image. Thus, the 2\% geometric efficiency of the sensor is effectively increased to 100\% and throughput is increased by a factor of 50 times, or two orders of magnitude.

There are five range images arranged vertically on the left. These are rendered as intensity images where darker greys indicate increasing distance from the sensor. The terrain map constructed by the perception system is rendered on the right. The top figure shows the map as an image where lighter greys indicate higher elevations. In the center of the map is the vehicle at the position where the 5th image was captured. The lower right figure is the same terrain map rendered as a wireframe surface from the vantage point of the initial position.

\[^1\] The assumption that the measured range significantly exceeds the sensor height.
There are three hills in the scene whose range shadows are clearly visible in the terrain map. In the first image, the vehicle is accelerating but still travelling relatively slowly. The range window is relatively wide and positioned near the bottom of the image. The first hill is in the range window. In the second image, the second hill is in the range window and the first hill has already been processed. Indeed, none of the left side of the image is processed because the data in the range window is occluded. In the third image, the third hill is now in the range window. In the fourth image, the vehicle is driving past the first hill and is rolled to the right because of it. This rolls the image to the left and the algorithm compensates appropriately. In the fifth image, the range window has moved past the third hill to the flats beyond and a fourth hill is barely visible in the distance.

The system performs identically on real images, but simulated ones were used here to illustrate several points within a limited space.

4.1.3. Horizontal vs. Vertical Baseline

Figure 14. illustrates the operation of adaptive stereo on two horizontal baseline input images. The initial input images appear at the top. The normalized, texture-enhanced images appear below the input images. The disparity image is shown to demonstrate the spurious matches which are a by-product of the disparity window\(^1\) approach. These correspond to local maxima in the correlation curve, but there is no information available to detect this. Finally, the cleaned-up range image is presented. It incorporates an efficient filter, based on classical region-finding techniques, which removes the local maxima and provides a clean range image for processing by the rest of the navigation system.

The feasibility of vertical baseline stereo is demonstrated in the vertical stereo pair shown in Figure 15. The original angular resolution was 640 rows by 486 columns. Run-time for this particular pair was 0.7 secs on a SPARC-20 while searching 60 disparity levels and reducing resolution to 160 rows by 123 columns.

A run of the same stereo pair from the previous example is shown in Figure 15. This run is of sufficient density to completely populate a 1/2 meter resolution terrain map between the ranges of 25 meters and 30 meters.

---

1. A disparity window is the equivalent of a range window for stereo vision. Since disparity and range are inversely related through the stereo baseline (camera separation), a unique disparity value corresponds to any range.
A breakdown of both vertical baseline runs is shown in the table below (Table 1:)

Figure 13 - Adaptive Rangefinder Perception. The five simulated images were taken in order from bottom to top as the vehicle advances. In each image, the dark horizontal lines outline the range window. These data windows were merged into a single terrain map that is represented from an overhead viewpoint - as an image in the top right and as a wireframe surface in the lower right figure.

A breakdown of both vertical baseline runs is shown in the table below (Table 1:)

---

Range Image Sequence

<table>
<thead>
<tr>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Image 4</th>
<th>Image 5</th>
</tr>
</thead>
</table>

Terrain Map

- Hill 1
- Hill 2
- Hill 3
Figure 14. Adaptive Stereo. Top two images are the original pair. The next two are texture enhanced to prepare for stereo processing. Adaptive stereo searches only a narrow disparity window in order to minimize computations. A by-product of this technique is spurious matches which appear as small error regions in the disparity image in the bottom left. In conversion from disparity to the range image in the bottom right, these small error regions are removed.
Figure 15. Vertical Baseline Stereo. This figure demonstrates vertical baseline stereo. The input images to the left are converted to disparity represented in the disparity image in the top right. In this case, adaptive stereo is not used so all disparities are searched. The terrain map to the bottom left is rendered from an overhead viewpoint with intensity proportional to elevation. At the bottom right, the same map is rendered as a wireframe surface from the perspective of the cameras.
4.2. Rectification of Stereo Images

The stereo vision algorithms we use rely heavily on an epipolar geometry constraint. In other words, the algorithms assume that the optical axes of the two cameras are parallel. This ensures that a match for a given feature in the first image will be found along the same column (for vertical baseline) in the second image which greatly reduces the search space. However, it is generally not possible to mechanically align

![Figure 16. Adaptive Vertical Baseline Stereo. When adaptive stereo is used, both the vertical field of view and the disparity window searched are limited as shown above. Runtimes for both adaptive and non-adaptive variants are given in the table below.](image)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value (Non-adaptive)</th>
<th>Value (Adaptive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input / Output / Adaptive Rows</td>
<td>640 / 160</td>
<td>640 / 160 / 45</td>
</tr>
<tr>
<td>Input / Output / Adaptive Cols</td>
<td>486 / 123</td>
<td>486 / 123 / 123</td>
</tr>
<tr>
<td>Disparities</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>Setup</td>
<td>1 msecs.</td>
<td>4 msecs.</td>
</tr>
<tr>
<td>Scale &amp; Rectify</td>
<td>32 msecs.</td>
<td>11 msecs.</td>
</tr>
<tr>
<td>Normalize</td>
<td>84 msecs.</td>
<td>26 msecs.</td>
</tr>
<tr>
<td>Correlation</td>
<td>532 msecs.</td>
<td>50 msecs.</td>
</tr>
<tr>
<td>Disparity</td>
<td>14 msecs.</td>
<td>4 msecs.</td>
</tr>
<tr>
<td>Cleanup</td>
<td>22 msecs.</td>
<td>6 msecs.</td>
</tr>
<tr>
<td>Triangulate</td>
<td>8 msecs.</td>
<td>5 msecs.</td>
</tr>
<tr>
<td>Total Runtime</td>
<td>700 msecs.</td>
<td>96 msecs.</td>
</tr>
</tbody>
</table>

Table 1: comparison of adaptive and non-adaptive stereo
camera with very high precision. We have developed a program that accepts two camera images and from the users interaction it generates a rectification matrix (usually referred to as an ‘H’ matrix). This matrix when used on one of the images, rectifies it such that the epipolar constraint is justifiable.

### 4.2.1. Non-Rectified Image Pair

An image pair as is obtained directly from the stereo cameras is shown in Figure 17. Note how the pair satisfies neither the epipolar geometry constraint, nor is the disparity zero at \( \infty \) range.

### 4.2.2. Rectification Matrix

The aim of stereo rectification is to warp the second (bottom) image, so that the pair satisfies the epipolar geometry and infinity range constraints.

Image rectification can be done by:

\[
\begin{bmatrix}
  c' \\
  r' \\
  1
\end{bmatrix} = H_{3 \times 3} \begin{bmatrix}
  c \\
  r \\
  1
\end{bmatrix}
\]  

(1)

where \( \begin{bmatrix}
  c \\
  r \\
  1
\end{bmatrix}^T \) is the original image coordinate and \( \begin{bmatrix}
  c' \\
  r' \\
  1
\end{bmatrix}^T \) is the rectified image coordinate. The objective is to find \( H_{3 \times 3} \) which warps the second image to suit the epipolar geometry and the infinity range constraints.

This can be done by hand picking a number of matching features in the two images to find a least square solution.

1. To find \( H_{0j} \), pick i-matching features from the top & bottom image. We use columns \( (c_j) \) from the top image and the associated columns & rows \( (c_{2i}, r_{2i}) \) from the bottom image to solve for \( H_{00}, H_{01}, H_{02} \):

\[
\begin{bmatrix}
  c_{10} \\
  c_{11} \\
  \vdots \\
  c_{1i}
\end{bmatrix} = \begin{bmatrix}
  c_{20} & r_{20} & 1 \\
  c_{21} & r_{21} & 1 \\
  \vdots & \vdots & \vdots \\
  c_{2i} & r_{2i} & 1
\end{bmatrix} \begin{bmatrix}
  H_{00} \\
  H_{01} \\
  H_{02}
\end{bmatrix}
\]  

(2)

Which takes the form of:

\[
\bar{y} = \bar{A} \bar{x}
\]  

(3)

To solve for \( \bar{x} \) using Pseudo-Inverse;
Figure 17. Non-Rectified Image Pair (Top Camera & Bottom Camera)

The epipolar line:
(they are off by approx. 15 pixels)

At Range = Infinity
(off by approx. 9 pixels)
\[ x = \left( \left( \begin{array}{c} \alpha^T \\ \alpha \end{array} \right)^{-1} \left( \begin{array}{c} \alpha^T \\ \alpha \end{array} \right) \right)_y \]  

(4)

Which actually looks like:

\[
\begin{bmatrix}
H_{00} \\
H_{01} \\
H_{02} \\
\end{bmatrix} = \begin{bmatrix} c_{20} & r_{20} & 1 \\
    c_{21} & r_{21} & 1 \\
    \ldots & \ldots & \ldots \\
    c_{2i} & r_{2i} & 1 \\
\end{bmatrix}^T \begin{bmatrix} c_{20} & r_{20} & 1 \\
    c_{21} & r_{21} & 1 \\
    \ldots & \ldots & \ldots \\
    c_{2i} & r_{2i} & 1 \\
\end{bmatrix}^{-1} \begin{bmatrix} c_{20} & r_{20} & 1 \\
    c_{21} & r_{21} & 1 \\
    \ldots & \ldots & \ldots \\
    c_{2i} & r_{2i} & 1 \\
\end{bmatrix}^T \begin{bmatrix} c_{10} \\
    c_{11} \\
    \ldots \\
    c_{1i} \\
\end{bmatrix} \\
\]

(5)

2. Using the same method, \( H_{10}, H_{11}, H_{12} \) can be solved using rows \( (r_{1i}) \) from the top image and the associated columns & rows \( (c_{2i}, r_{2i}) \) from the bottom image that have matching features at infinite range. The equation is:

\[
\begin{bmatrix}
H_{10} \\
H_{11} \\
H_{12} \\
\end{bmatrix} = \begin{bmatrix} c_{20} & r_{20} & 1 \\
    c_{21} & r_{21} & 1 \\
    \ldots & \ldots & \ldots \\
    c_{2i} & r_{2i} & 1 \\
\end{bmatrix}^T \begin{bmatrix} c_{20} & r_{20} & 1 \\
    c_{21} & r_{21} & 1 \\
    \ldots & \ldots & \ldots \\
    c_{2i} & r_{2i} & 1 \\
\end{bmatrix}^{-1} \begin{bmatrix} c_{20} & r_{20} & 1 \\
    c_{21} & r_{21} & 1 \\
    \ldots & \ldots & \ldots \\
    c_{2i} & r_{2i} & 1 \\
\end{bmatrix}^T \begin{bmatrix} r_{10} \\
    r_{11} \\
    \ldots \\
    r_{1i} \\
\end{bmatrix} \\
\]

(6)

3. With \( H_{00}, H_{01}, H_{02}, H_{10}, H_{11} \), and \( H_{12} \), the rectification matrix is solved;

\[
H_{3 \times 3} = \begin{bmatrix} H_{00} & H_{01} & H_{02} \\
                        H_{10} & H_{11} & H_{12} \\
                        0 & 0 & 1 \end{bmatrix} \\
\]

(7)

4.2.3. The Program \textit{easy_rect} and The Result

The process of running the \( H \) matrix generation program (\textit{easy_rect}) is shown in Figure 18. This program allows the user to pick matching rows, columns, or \textit{both} in two images and generate an \( H \) matrix along with a least square error indication. Images in Figure 18. are rotated clockwise, since they are true images coming in from cameras which are mounted sideways.

The user can zoom in and out to pick the exact pixel; this is a major advantage over other programs using automated feature picking. To solve for the \( H \) matrix, the minimum number of features to be matched are 6 (3 for rows, 3 for columns), and the number can go down to 3, if features at \( \infty \) range are used to solve for both \( [H_{00} \ H_{01} \ H_{02}] \) and \( [H_{10} \ H_{11} \ H_{12}] \). However, since \( H \) matrix is solved in a least square sense, it is always better to use more matches.

A total of 12 matches (6 for row, 6 for column) in the previous images were used to generate this \( H \) matrix;
which was used to create the result in Figure 19. Remember, the \( H \) matrix is only applied to the bottom image, the top image is unchanged. As the figure shows, the ideal epipolar geometry constraint and the zero disparity at infinity range constraint hold after the rectification. TUGV’s stereo program was used to prove that the rectification matrix generated by this method makes range data more reliable.

4.3. Camera Calibration (Correction of Lens Distortion)

It would be nice to meet the epipolar geometry constraint and the zero disparity at \( \infty \) range constraint by just applying the \( H \) matrix to the second (bottom) image, but the issue of lens distortion remains.

In this section, I go over how the parameters are found, how the image is corrected, and explain why lens distortion correction does not have to be used in the current setup.
Figure 20. shows the effect of lens distortion using a 8.0 mm lens and a perfect one-inch by one-inch grid. Although the lighting is poor, it is obvious from the coordinates that the grid lines are curved. This is the effect of lens distortion, and the aim here is to:
• Determine the distortion parameters.
• Determine whether the distortion is bad enough to require correction.

4.3.1. Deriving Distortion Parameters

The lens distortion model used here is an adaptation of Roger Tsai’s work on lens calibration. The basic idea is to have a set of known non-coplanar points (can be coplanar, but numerical stability will be lost) in world coordinates and image coordinates to derive the parameters;

\( R, T \): Rotation & Translation matrix which maps \((x_w, y_w, z_w)\), the world coordinate to \((x, y, z)\), the camera coordinate.

\( f \): Focal length of the lens

\( \kappa_1, \kappa_2 \): Radial lens distortion parameter

\( s_x \): Uncertainty scale factor

---

The four steps of the transformation from a 3D world coordinate to a computer image coordinate is shown in Figure 21. on page 35. The parameters we need in order to correct distorted image \((X_f, Y_f)\) into non-distorted image \((X_u, Y_u)\) are \(\kappa_1, \kappa_2,\) and \(s_x\). Assuming the parameters have been derived via Tsai’s method, the relationship between \((X_f, Y_f)\) and \((X_u, Y_u)\) is:

\[
X_u = X_d + X_d \left( \kappa_1 r^2 + \kappa_2 r^4 \right)
\]

(9)

\[
Y_u = Y_d + Y_d \left( \kappa_1 r^2 + \kappa_2 r^4 \right)
\]

(10)

where \((X_d, Y_d)\) = distorted image coordinate and \(r = \sqrt{X_d^2 + Y_d^2}\).
where \((C_x, C_y)\) = computer image coordinate for the point in the image plane \([504, 520]\) in our case, 
\((d_x', d_y') = center to center distance between adjacent sensor elements in the X, Y direction \([(0.000009, 0.000009)\) in our case], 
\(N_{cx} = number of sensor elements in X direction [1024], N_{fy} = number of pixels in a line as sampled by the computer [1008].

Now that we have the mapping, it is possible to create the inverse mapping. The TUGV stereo vision algorithm loads a pixel array built using this mapping, where each (non-distorted) pixel contains the pixel coordinate for the associated pixel in the distorted image. Using this technique, the time it takes to correct distortion is very small compared to computing the mapping in real time. Figure 22. shows that the grid is close to being straight as a result of the distortion correction. Therefore, applying distortion correction before stereo rectification will increase the chance of outward pixels being correlated with the matching pixels. The drawback is the occurrence of aliasing which occurs when a smaller number of data points (pixels) is used to create a larger part of the image. This could be a major drawback to stereo vision, since we could actually lose a fair amount of data from aliasing.

4.3.2. Deciding when distortion correction is required

There are two issues that need to be considered;

- Effect of distortion on feature matching
- Effect of distortion on triangulation

To check these effects, a few undistorted image coordinates \((X_f, Y_f)\) are chosen to find their associated distorted image coordinates \((X_d, Y_d)\) and their differences \(\Delta(X_f, Y_f)\). The result is shown on Table 2 on page 38, and is graphically shown in Figure 23.
4.3.2.1. Effect of distortion on feature matching

Since we are using vertical baseline stereo, distortion will cause a problem in stereo matching when relative distortions in x direction are very different. The relative distortion difference (the difference of Δ) is largest, for example, in the case where the same feature appears in the right center of the top image (e.g. (1000,520)), and the upper right corner of the bottom image (e.g. (1000,0)). Their difference of Δ in x-direction is;
With this lens the difference of $\Delta$ in $x = 30 - 22 = 8$ pixels

(14)

This means that the same feature will show up 8 pixels off in the x direction when trying to match features in stereo. But, since the stereo algorithm currently begins by scaling the image down by about 16:1 in the x direction, the worst case difference of 8 pixels is now reduced to less than 1 pixel, and thus we can argue that the distortion will not significantly affect the stereo matching.

### 4.3.2.2. Effect of distortion on triangulation

This time, we focus on the effect in the Y direction. Assume that the cameras are looking at an object 15 meters away. Since triangulation is expressed as;

\[
y = \frac{bf}{d} = \frac{b}{\delta}
\]

(15)

where $y=15m$ and $b=1m$, $\delta = 1/15$ rad = $4^\circ$. With a $40^\circ$ FOV lens that covers 1040 pixels;

\[
\frac{4^\circ}{40^\circ}1040\text{pix} = 104\text{pix}
\]

(16)

So, we must consider relative distortions where some feature is separated by ~100 pixels between the top and bottom images. By looking at $\Delta (1000,0)$ and $\Delta (1000,100)$, we can see that their difference in Y-direction is $(30,-32) - (29,-24) = 8$ pixels. In other words, using the current 8:1 scaling in the Y-direction, they have conserved their relative positions, a desirable characteristic for the stereo vision algorithm.

<table>
<thead>
<tr>
<th>Undistorted to Distorted image coordinates</th>
<th>((X_f, Y_f))</th>
<th>((X_d, Y_d))</th>
<th>(\Delta (X_f, Y_f) = (X_f - X_d, Y_f - Y_d))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1000,0)</td>
<td>(970,32)</td>
<td>(30,-32)</td>
<td></td>
</tr>
<tr>
<td>(1000,10)</td>
<td>(970,41)</td>
<td>(30,-31)</td>
<td></td>
</tr>
<tr>
<td>(1000,100)</td>
<td>(971,124)</td>
<td>(29,-24)</td>
<td></td>
</tr>
<tr>
<td>(1000,300)</td>
<td>(976,311)</td>
<td>(24,-11)</td>
<td></td>
</tr>
<tr>
<td>(1000,480)</td>
<td>(978,482)</td>
<td>(22,-2)</td>
<td></td>
</tr>
<tr>
<td>(1000,500)</td>
<td>(978,501)</td>
<td>(22,-1)</td>
<td></td>
</tr>
<tr>
<td>(1000,510)</td>
<td>(978,510)</td>
<td>(22,0)</td>
<td></td>
</tr>
<tr>
<td>(1000,520)</td>
<td>(978,520)</td>
<td>(22,0)</td>
<td></td>
</tr>
</tbody>
</table>
4.4. Image Processing on the MCC

The Mission Computer Cluster (MCC) is a military multi-processor computer specialized for image processing. It contains two main parts— data processor intended for sequential computation and array processor intended for parallel computation. A project goal has been to port the stereo vision to the array processor and the navigation algorithms to the data processor.

As configured for the TUGV project, the MCC contains:

- 2 Digital Processing Modules (DPM), and
- 4 Signal Processing Controllers (SPC), each containing 16 Signal Processing Elements (SPE).

The organization of the processors is shown in Figure 24.

This is a total of 64 SPEs, a signal processor specialized to work on images. The SPEs for a SPC execute in lock step, each executing the same instruction but on a different part of the data. This approach is well suited to algorithms that process large amounts of data and can partition the data for parallel processing. This is the case for image processing. This approach also reduces the complexity of the SPEs by using a common execution unit.
When started, the SPEs are given a set of macros that they can execute. The data to process arrives via the Data Flow Network (DFN). When data arrives, the SPC issues macro execution orders and passes arguments to the SPEs. SPE processing is performed in SPE memory and the results are transferred to the SPC which continues processing the data, possibly the merged data from all the SPEs. The results of this processing are transferred to one or all DPMs.

The DPMs have access to the outside world via MIL-STD-1553 communications that give them control of the vehicle, the cameras and image frame grabbers and allow for external supervisory and monitoring functions.

The MCC performs the following operations on image data to produce a disparity map which can be translated to a range map. The image processing operations are done sequentially in stages. Each stage leaves its results in memory for the next stage to use. The stages are:

- **Unpack and Subsample** - The data arrives in a packed format (discussed below). It is unpacked into separate left and right memory areas. Simultaneously the data is subsampled in the horizontal direction to reduce the number of pixels that must be processed.
- **SPE to SPE transfer** - each SPEs data is transferred to its adjacent SPEs to give rectification a bigger sample to work from and thus allow for greater misalignment in the camera orientations.
- **Rectification** - the images are adjusted to align the images by columns so that later on, the image difference routine will be working with matching parts of the image.
- **Difference of Gaussian** - edge enhancement that highlights details of the image for stereo matching.

And, for each disparity that is checked, these operations are performed:

- **Image difference** - the images are offset by the current disparity and subtracted
- **Matching** - a matching algorithm checks each point in the difference image to see how well the images match.

### 4.4.1. Computing the Image Slice Dimensions

The image slice or Region of Interest (ROI) is a vertical range within the image that spans the width of the image. It can be moved vertically or heightened under command of Ranger in order to adapt to the terrain ahead. The ROI serves to reduce the stereo vision computation time by limiting the amount of data to be processed.

The slice specified by Ranger is expected to be the output of the stereo vision computations and the stereo vision algorithms are expected to do whatever is required to produce the slice. To do so, stereo vision must determine the ROI as it maps into the camera image data by working backward from the Ranger ROI through the stereo vision operations.
The AP receives the ROI parameters relative to the upper camera image. The equivalent region in the lower frame is computed by adding the minimum disparity to the lower bound of the upper ROI and by adding the maximum disparity to the upper bound of the upper ROI. The left and right bounds are unchanged. The bounds of the two ROIs, upper and lower, are combined to give a maximum bound that encompasses both ROIs. This bound is the image size that the disparity calculations in stereo vision require to generate the ROI that Ranger wants.

From this ROI, Stereo Vision ultimately determines the bounds of the camera image data that must be processed to get the memory image for this ROI. The two are not the same because the cameras are not aligned exactly and the data for some pixels on the edge of the ROI are lost in some of the computations. Figure 25. shows how the combined ROI for the upper and lower images is produced from the upper ROI given by Ranger. Note that the ROI is shown for one APs image half (with overlap). Also note that the lower camera’s perspective results in its ROI being higher in the image.
The combined upper and lower ROI will be the output from the Difference of Gaussian (DoG) functions, during which some border pixels will have been corrupted and trimmed off, 2 from both the top and bottom sides for the Difference of Gaussian (Columns) function. Therefore, four rows are added to the combined ROI to give the DoG ROI, the image size the DoG functions need. Two rows on the left and right are also corrupted in the Difference of Gaussian (Rows) processing, but these cannot be trimmed because of hardware characteristics - the image width must remain a multiple of 16 pixels.

An unknown number of pixels are corrupted on all sides for the image rectification routine. The degree of misalignment between the camera orientations ultimately determines this. For a particular point in the upper image, a camera displacement or rotation could result in the matching point in the lower image being out of the camera picture.

The rectification routine uses an affine transformation to correct for various image errors, therefore an inverse affine is done on the DoG ROI bounds to determine a rectangle that will contain the necessary data for the transformation. Since the AP breaks the ROI into 16 vertical slices which will eventually go to the 16 SPEs, the inverse affine transformation is actually done on the bounds of each slice to help protect against a curvature that might require more data.

Figure 26. shows how the DoG ROI is inverse transformed and how new ROI bounds are determined from it. This figure also shows how corruption happens during rectification with the rotated dashed box indicating the inverse affine of the DoG ROI. Where a corner extends beyond the camera’s view, non-image data from the SPE memory is used.

Due to limitations in the PBM macros, the first row of the slice of the image the AP can receive must begin on a multiple of 16 boundary and the height must also be a multiple of 16 pixels.

To accomplish this, a variable number of columns (0 - 15) are subtracted from the top side to achieve the alignment and then a variable number of columns are added to the bottom side to meet the height requirement. If these computations result in a height that is greater than the processing memory size, the ROI is truncated on the bottom side. If the bottom edge extends past the bottom of the camera view it is truncated to the bottom of the view. The result is the rectification ROI as shown in Figure 27.

In the last step in working backwards, the rectification ROI is converted to the PBM image ROI. To reduce the processor workload columns are removed from the image, currently at a 7:1 ratio. This does not reduce the resolution of the stereo vision computations since the range (actually disparity) output is determined along the vertical axis. The image ROI is wider than the unrectified ROI by this ratio.
Figure 26. Inverse Affine of ROI and new ROI bound

Figure 27. Image data ROI including modulo adjustment
There is a finite amount of memory in the APs to hold the image and if that was not the limit then there is a finite amount of processing time that can be dedicated to each image. Therefore a somewhat arbitrary maximum image size must be set. Currently the maximum height of that memory is 384 pixels. Plainly the smallest value for maximum width is 480 pixels, the width of half a frame (960 / 2). But this number must meet two granularity requirements; one derives from the first stage in the processing where columns are removed from the image data. The other granularity requirement is that the width must be a multiple of 16, the number of SPEs, because equal portions of the slice must be given to each AP.

In the case of 7:1 subsampling, the integer that gives a number greater than the minimum width is 5 (N * 16 * 7 > 480), which gives a width of 560 and 35 columns to each SPE which is 5 columns per SPE after the 7:1 subsampling. This is obviously going to cause some overlap of the output of the two APs that work on this frame. This overlap is handled in the Ranger program.

The position of the image slice that is to be downloaded from the PBM image memory can now be calculated. The upper bound is the edge of the modulo 16 correction, the left bound is the image left bound (for the left half AP case), the bottom bound is the edge of the modulo 16 height correction, and the right bound (for the right half AP case) is the left bound less 7 times the sum of the heights of the SPE slices. Notice that due to the choice of coordinates the lower bound is larger than the upper bound.

The right half case differs only slightly from the left half case. In the left half case the downloaded area is against the left edge; in the right half case the right edge of the image slice is against the right edge of the image.

4.4.2. Processing the Image Slice

In operation, the cameras are mounted on their sides. Rows in the camera’s frame of reference, are columns in the normal frame of reference. This results from the change to vertical baseline from horizontal baseline by rotating the camera jig that holds both cameras. The stereo vision software assumes that the left camera from horizontal baseline stereo moved to the lower position for vertical baseline. Therefore, top in the camera’s frame of reference is left in the normal frame of reference. In the following description rows and columns will still be specified in the normal frame of reference to avoid confusion.

The TUGV stereo vision process begins with 4 cameras mounted in vertical pairs, with one pair beside the other, as shown in Figure 28. Each camera has a 1024x1024 image array with 8 bit resolution.

The output of these cameras is fed to the Sensor Interface Unit (SIU) which preprocesses the data and feeds it to the MCC.
The images arrive at the MCC merged into two 1008x960 frames; left camera pair and right camera pair. Each pixel of a frame is a 16 bit word with an 8 bit pixel from each camera in each of the bytes, upper camera pixel in the most significant byte.

As shown in Figure 29, each AP is assigned to process more than half of an image. The overlap results from some granularity in the systems but it also simplifies the job of merging the two halves of the image.

Each AP will receive the data for a complete frame from the SIU. From its processor ID and a lookup table, each AP knows which half, left or right, it is to process. The AP also knows the region of interest. Before the next image arrives, this information is passed to the PBM macro which positions the correct half of the slice for that AP into the low end of the PBM during image loading.
All the data for the slice are unpacked into the SPE memories, including the data outside the Ranger ROI. Simultaneously the 7:1 sub-sampling is performed by averaging the 7 input pixels to produce the value for the output pixel.

All image processing, including rectification and DoG, is done on a SPE by SPE basis. Without other measures this would result in corruption of 2 columns on each side of the 5 column SPE slice.

This potential problem is circumvented by the SPE to SPE transfer. When the image slice is loaded into the SPE memories, each SPE loads its portion of the SPE slice into the middle of a 3 slice size memory. Then the SPE to SPE function causes each SPE to send its slice to its adjacent SPEs; first clockwise, then counter-clockwise, with the clockwise slice going into the upper portion and the counter-clockwise slice going into the lower portion. The result is that each SPE slice is now 384 columns by 15 rows and the column corruption will be outside the SPE’s own slice. This transfer is not as inefficient as it might appear because there is a special AP instruction that simultaneously transfers one data word for all 16 SPEs in 2 clocks.

The rectification algorithm is passed three source data parameters. The first is the pointer to the 5,0 location relative to the 15 row ROI image slice, i.e. the start of this SPEs 5 row slice. The second is the number of columns in the image slice. Rectification is also passed three arrays of data that give the starting offset, relative to the source data pointer for the first pixel in each row. These arrays are generated at start-up time from rectification matrices and contain the truncated integer of that offset and fractional values for the horizontal and vertical offset from the exact point. The fractional values are stored as integers multiplied by 65536. The function also requires delta-X and delta-Y values that describe the movement required in the source array for each pixel movement in the destination array. All the data in the 5 row image slice are rectified, including the data outside the ROI.

It is immediately obvious that there may be some boundary problems in this arrangement.

- The margin of 5 rows (a SPE slice) on the top of the top SPE and the bottom of the bottom SPE is not correct image data. For the top it is the data from the bottom SPE and for the bottom it is the data from the top SPE. In each half of the frame this is corrected on the side closest to center of screen by the image slice being greater than half of the frame.
- The modulo corrections to ROI bound and ROI width may increase the width of the image slice so that it must be truncated.
- If the cameras are not sufficiently aligned, the rectification matrices may result in a bounded inverse Affine area that extends outside the image slice.
- It is also possible that the ROI from Ranger will translate into an ROI for the lower frame that extends past the upper limit of the image. These cases are detectable by comparing the bound rectangle (includ-
The remaining stages of image processing are performed and the disparity map for the ROI requested by Ranger is produced. The disparity map is transferred to the SPC processor of the MCC for one last stage of processing - cleanup. For a variety of reasons there will be flaws in the disparity map. There are areas where stereo vision was unable to get a good match at any disparity but chose the best match. The cleanup procedure assumes that the disparity for adjacent pixels, both horizontal and vertical, will only change by a small amount. Changes greater than this are assumed to be flaws and are set to 0 which is equivalent to an infinite range.
5. A Design for a Generic Stereo Based Autonomous Vehicle

5.1. Perception Sensor Requirements

An autonomous vehicle sensory system must satisfy several requirements in order to be useful. It must have sufficient resolution to resolve a small obstacle, generate data at sufficient rate, and image a wide enough field of view to present alternatives when part of the way is blocked by an obstacle.

5.1.1. Response Distance

It is possible to define a quantity called the response distance which applies to both a panic stop (issuing the brakes fully) and an impulse turn (issuing a hard turn without changing speed):

\[ s_{\text{response}} = T_{\text{react}} V + \frac{V^2}{2\mu g} \]

where \( s_{\text{response}} \) is the distance required to turn or stop, \( T_{\text{react}} \) is the reaction time of the autonomous systems, \( V \) is the vehicle initial velocity, \( \mu \) is either the coefficient of friction or the maximum lateral acceleration in g’s as the case requires. The first term can be called the reaction distance and the second is the maneuver distance.

This relationship is plotted below for typical values of the coefficient of friction or lateral acceleration.

In both cases, we have implicitly assumed that actuator dynamics can be neglected or absorbed into the reaction time.
All components of the reaction time except the actuator component can be considered equal for both braking and turning. However, on conventional (Ackerman) steered vehicles, the time required to complete movement of the steering actuator can often significantly exceed that required for braking. Also, the coefficient of lateral acceleration can be lower than the coefficient of friction because it is limited by the propensity to roll over in a turn. In short, the reaction distance is larger for turning than for braking.

5.1.2. Minimum Range

The minimum range required of a sensor depends on velocity and the reaction time. The vehicle is committed to travelling a certain distance at any speed within which it cannot stop. The minimum required range can be read from the previous graph where \( V \) is the minimum speed, and \( \mu \) is the coefficient of friction.

5.1.3. Maximum Range

The maximum range required of a sensor depends on the velocity and the reaction time. Let us assume that the vehicle should be able to turn 90 degrees if it sees a large obstacle ahead - without slowing down. Then the maximum required range can be read from the above graph where \( V \) is the maximum vehicle speed, and \( \mu \) is the maximum lateral acceleration expressed in g’s.

5.1.4. Response Angle

Define a turning stop maneuver as hitting the brakes while engaged in a turn. It is possible to define, for the turning stop maneuver, a quantity called the **response angle**:

\[
\psi_{response} = \frac{s_{response}}{\rho}
\]

where \( \rho \) is the radius of curvature of the turn and \( s_{response} \) is the response distance. In the particular case of a turn at the minimum safe radius of curvature, we have:

\[
\psi_{response} = \frac{T_{react}V + \frac{V^2}{2\mu g}}{\max\left(\frac{V^2}{2g}, \rho_{kin}\right)}
\]

where \( \rho_{kin} \) is the minimum turn radius available mechanically. This relationship is plotted below for typical values of the coefficients of friction and lateral acceleration and a minimum kinematic turn radius of 7.5 meters.
Clearly, the response angle grows roughly linearly while the turn radius is limited by the mechanism. Beyond some velocity (here 6 meters/sec), the turn radius becomes limited by the lateral acceleration and the response angle decreases quadratically.

5.1.5. Horizontal Field of View

The horizontal field of view will be determined by the turning stop maneuver and hence by the response angle. The rationale for this choice is that when the vehicle is executing a turn, it will have just enough sensory look-ahead to stop if an obstacle appears. Another important matter to consider is that a sensor normally cannot change its horizontal field of view dynamically, so it is necessary to allocate horizontal field of view for a range of velocities.

\[ HFOV = \max_V \{\psi_{\text{response}}\} \]

This may mean that even though the field of view requirements reduce as speeds increase, the physical sensor cannot take advantage of it.

5.1.6. Vertical Field of View

There are several potential mechanisms that might be used to determine requirements on the vertical field of view. The major kinematic requirement which influences the vertical field of view is the pitch angle induced in the vehicle body by the most challenging, yet navigable, terrain.

Let the highest safe body pitch angle be \( \theta \). Figure 32. illustrates the two extreme cases which determine the vertical field of view required to ensure that the vehicle is able to see up an approaching hill or past a hill that it is cresting.
On this kinematic basis, the vertical field of view required is four times the maximum pitch of the body.

\[ VFOV = 4\theta \]

### 5.1.7. Sweep Rate

This section considers a dynamic basis for specifying the required vertical field of view. It will be useful to assume that the height of the sensor is always significantly less than the range measured. This will be called the **small incidence angle assumption**. Under this assumption, the vertical field of view can be expressed as follows:

\[
VFOV = \theta_{\text{max}} - \theta_{\text{min}} = \frac{h}{R_{\text{min}}} - \frac{h}{R_{\text{max}}} = h\frac{(\Delta R)}{R_{\text{max}}R_{\text{min}}}
\]
The **sweep rate** of a sensor can be defined, in image space, as the vertical field of view (VFOV) generated per unit time. Thus, it has units of angular velocity. It may represent the physical motion of the elevation mirror in a rangefinder or the product of the VFOV and the frame rate for a camera. The sweep rate required of a perception sensor is related to the velocity of the vehicle. If $\dot{\theta}$ is the sensor sweep rate, $T_{cyc}$ is the sensor frame period, $h$ is the height of the sensor, and $R$ is the average range over the field of view, the requirement on sweep rate can be approximated as in Figure 34.

$$\dot{\theta} = \frac{s}{RT_{cyc}} = \frac{\Delta R (h/R)}{RT_{cyc}} = \frac{\Delta Rh}{R^2 T_{cyc}}$$

![Figure 34. Sweep Rate](image)

The sweep rate is the quantity that relates directly to vehicle velocity. Nonetheless, if the sensor frame rate is fixed, the required vertical field of view is simply:

$$VFOV = h \frac{\Delta R}{R^2} = h \frac{VT_{cyc}}{R^2}$$

### 5.1.8. Angular Resolution

The angular resolution required of a sensor can be related to the smallest object that needs to be detectable at a given range. Straightforward kinematic approximations provide the relationship between the angular extent of a pixel and the spatial extent of its footprint on an object. Nominally, the smallest object that matters is 1/2 to 1/4 of a wheel radius, and pixels must be small enough to land 6 on that 1/4 radius in order to resolve the obstacle from one slightly smaller.

To use the graph, compute $dz$ as wheel_radius/24, then choose the response distance that corresponds to the maximum vehicle speed. Finally, determine which curve crosses the intersection just established.
5.1.9. Stereo Angular Resolution

Methods that analytically characterize the angular resolution of stereo vision do not yet exist. The characterization of the frequency response of stereo vision to a given input image spectrum is an open research question. Therefore, it is not possible to accurately quantify the angular resolution requirements of stereo vision. However, it is safe to assume that a stereo range image is a smoothed version of reality over a smoothing window whose size is equal to the stereo correlation window. Therefore, the above results need to be increased by the size of the stereo correlation window. Historically, it has never been possible to resolve small objects in a stereo range image.

5.2. Positioning Sensor Requirements

Requirements on positioning sensors can be generated from considerations of obstacle avoidance or from considerations of goal seeking. If obstacle avoidance is not robust, it is necessary to cause the vehicle to track its specified path within an error window which has been previously guaranteed to be free from obstacles.

5.2.1. Path Following

For example, when following a road based only on position feedback, an error of half a road width may be acceptable. If perceptual road following is incorporated into the system, then the combined positioning and perception accuracy should admit errors no larger than half a road width.
5.2.2. Obstacle Avoidance

In a more realistic situation, prior information about the terrain is not available and the system must accurately resolve, locate, and avoid obstacles.

5.2.2.1. Relative Position Accuracy

The location of an obstacle relative to the vehicle is determined by perception sensors, so the only requirement on positioning systems is that they reflect vehicle motion accurately enough that the vehicle can avoid an obstacle based only on positioning feedback once it has been located. Using a maximum range of 30 meters and a wheel radius of 1/6 meter, the system should accumulate error with distance travelled at a rate that does not exceed:

\[
\text{relative accuracy} = \frac{1/6}{30} = 0.5\% \ \text{DT}
\]

Requirements on heading, attitude and odometric sensor indications can be any mixture which meets the above relative accuracy requirement.

5.2.2.2. Absolute Attitude Accuracy

With regard to estimating tip-over, the system must be able to differentiate attitude to about 1 degree absolute accuracy. This will permit differentiation of 30 degrees of roll from 29 degrees of roll assuming that perception sensors are perfectly accurate and that the vehicle can tolerate 30 degrees of roll. Realistically, several degrees error in perception must be tolerated as well, so that a final implemented system will have to avoid positions where predicted roll will be 5 degrees less than the maximum roll that the vehicle should be able to tolerate without tipping.

5.3. Computing Requirements

There are two approaches that may be used to predict computing requirements for autonomous vehicles - theoretical and empirical.
5.3.1. Theoretical Requirements

It is possible to estimate the requirements on perception based solely on the interrelationship between angular resolution, field of view, and sweep rate or frame rate. Extensive analysis, too detailed to repeat here, indicates the following relationship between vehicle speed, reaction time, and theoretical computing requirements:

\[
\text{Algorithm} & \quad \text{Estimate at Minimum Acuity, 4 second Reaction Time, and 10 m/s speed} & \quad \text{Complexity} \\
\hline
\text{constant flux} & 250 \text{ Mflops} & o\left(T_{\text{react}}^4 V^4\right) \\
\text{adaptive sweep} & 0.7 \text{ Mflops} & o\left(T_{\text{react}}^3 V^3\right) \\
\text{adaptive sweep, scan} & 0.035 \text{ Mflops} & o\left(T_{\text{react}}^2 V^2\right) \\
\text{ideal} & 0.0045 \text{ Mflops} & o\left(T_{\text{react}} V^2\right) \\
\hline
\]

**Table 3. Throughput Estimates**

In the table and the following figure, the following definitions apply:

- constant flux refers to a perception algorithm that continually processes images at 2 Hz.
- adaptive sweep refers to a perception algorithm that modulates vertical field of view based on speed.
- adaptive sweep, scan refers to a perception algorithm that modulates both vertical field of view and pixel size.
- ideal refers to a perception algorithm that optimally computes the minimum amount of data.

The actual data for all 4 second reaction time curves is plotted below on a logarithmic vertical scale.

Unfortunately, these theoretical results do not reflect many other computations that are necessary. The value of the theory is that it predicts the rate of growth of requirements with speed and reaction time.

5.3.2. Empirical Requirements

The actual real-time performance requirements of the Ranger system are demonstrated in the following tables. With adaptive perception enabled, using oversampling factors of 2 both horizontally and vertically\(^1\), the overwhelming computational cost is in the obstacle avoidance part of the system.

---

\(^1\) Oversampling of 2 implies, in this case, that 4 range measurements fall into a 0.75 meters terrain map grid cell on average.
While adaptive perception resamples a range image for optimum coverage of the terrain, the specific attributes of the laser range sensor used for this run are given in the table below:

**Table 4. ERIM Range Sensor Parameters**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Rows</td>
<td>64</td>
</tr>
<tr>
<td>Image Cols</td>
<td>256</td>
</tr>
<tr>
<td>HFOV</td>
<td>80°</td>
</tr>
<tr>
<td>VFOV</td>
<td>30°</td>
</tr>
<tr>
<td>HIFOV (angular resolution)</td>
<td>0.3125°</td>
</tr>
<tr>
<td>VIFOV (angular resolution)</td>
<td>0.4688°</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>2 Hz</td>
</tr>
</tbody>
</table>

Adaptive perception and adaptive regard are affected primarily by the following parameters:

**Table 5. Configuration Parameters**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Speed</td>
<td>3 m/s</td>
</tr>
</tbody>
</table>
The run-time of the system under these configuration parameters is summarized below:

Table 5. Configuration Parameters

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imaging Density</td>
<td>1.2</td>
</tr>
<tr>
<td>Column Oversampling Factor</td>
<td>2</td>
</tr>
<tr>
<td>Row Oversampling Factor</td>
<td>2</td>
</tr>
<tr>
<td>Map Resolution - x</td>
<td>0.75 m</td>
</tr>
<tr>
<td>Map Resolution - y</td>
<td>0.75 m.</td>
</tr>
<tr>
<td># Turn Commands</td>
<td>15</td>
</tr>
</tbody>
</table>

The 64 milliseconds quoted for perception apply to range image processing only - the portion of the code enabled for a laser rangefinder. Stereo vision requires significantly more processing.

For the built in stereo vision algorithm, it has been demonstrated that a dedicated SPARC 20 processor can barely meet the needs of both stereo vision and navigation. The following stereo parameters are the ones that primarily affect speed of processing.

Table 6. Run-Time (SPARC 20)

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perception</td>
<td>0.064 secs</td>
</tr>
<tr>
<td>Strategic Planning</td>
<td>0.002 secs</td>
</tr>
<tr>
<td>Tactical - Command Generator</td>
<td>0.003 secs</td>
</tr>
<tr>
<td>Tactical - Feedforward</td>
<td>0.125 secs</td>
</tr>
<tr>
<td>Tactical - Arbitrate</td>
<td>0.014 secs</td>
</tr>
<tr>
<td>Total Runtime</td>
<td>208 msecs.</td>
</tr>
</tbody>
</table>

The 64 milliseconds quoted for perception apply to range image processing only - the portion of the code enabled for a laser rangefinder. Stereo vision requires significantly more processing.

Table 7. Stereo Configuration Parameters

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Rows</td>
<td>128</td>
</tr>
<tr>
<td>Image Columns</td>
<td>128</td>
</tr>
<tr>
<td>Adaptive Disparity</td>
<td>on</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.5 meters</td>
</tr>
</tbody>
</table>
Using these parameters, stereo vision runs in 0.4 seconds on SPARC 20. These parameters can be adjusted to improve speed, but only at a cost in reliability.

5.4. Design

Rather than implementing stereo vision based navigation for a single vehicle or a single application, DRES intends to research the use of this technology in many roles. Therefore, flexibility, cost effectiveness and expendability are key elements of the design presented here. Generally, the components specified are the low performance members of their families, and the three functional blocks: image capture, processing and Inertial Navigation System (INS) can all be upgraded with only a minimal effort.

Since DRES does not have a single vehicle or application but intends to incorporate this technology into many vehicles designed for many roles for research purposes, this design is very cost effective and is easily upgraded for higher performance. The three functional blocks: image capture, processing and INS can all be upgraded with only a minimal effort.

5.4.1. Upgrade Paths for the Image Capture Block

The usual goal of upgrades to the image capture functional block is to use a wider field of view lens to allow for better path planning or to detect smaller objects by increasing the sensor resolution and effectively putting more pixels “on the ground”:

- The lens specified in the design is the widest angle lens commonly available (6.5 mm with a 126 degree field of view). Even an expensive lens is relative cheap so the widest angle lens was chosen. Thus the upgrade possibilities are limited but there are specialty manufacturers that offer lenses with a wider field of view. One interesting product that uses field splitting to achieve a 180 degree field of view is made by John Ellis ((408) 374-0670).

- Typically the sensor will have a number of defective pixels. This should not be a problem in applications where a number of pixels are averaged to reduce the processing requirement. But as the requirement for resolution increases, the averaging decreases and the effect of bad pixels may eventually become important. In this case the image sensor can be improved to eliminate bad pixels. The sensors specified in this design are Class 3 devices which would have been adequate for the TUGV project although Class 0 sensors were used.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Range</td>
<td>8 m</td>
</tr>
<tr>
<td>Max Range</td>
<td>20 m</td>
</tr>
</tbody>
</table>

Table 7. Stereo Configuration Parameters
• Major Defective Pixel - A pixel whose signal deviates by more than 25 mV from the mean value of all active pixels under dark field condition or by more than 15% from the mean value of all pixels under uniform illumination at 80% of saturation.

• Minor Defective Pixel - A pixel whose signal deviates by more than 8 mV from the mean value of all active pixels under dark field condition.

• Point Defect - An isolated defective pixel.

• Cluster Defect - A group of 2 to 6 contiguous major defective pixels.

• Column/Row Defect - A group of more than 6 contiguous major defective pixels along a single column or row.

Notes: No Row defects are allowed and chroma column defects are allowed. The processing functional module can be upgraded by using a faster SPARC processor board or a parallel processor machine such as a DataCube, although such a change may also require a change in the frame grabber. The code would have to ported to the new platform or operating system, but this should be a straight-forward process since both Ranger and the stereo vision code are written in C. The INS module specified is very accurate but if required it can be upgraded by replacing it with a more accurate version of that model or another model. The TUGV experience is that there are more INS interface standards than there are INS manufacturers and therefore a change to another model or manufacturer may require considerable hardware changes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Point Defects</th>
<th>Cluster Defects Total</th>
<th>Column Defects Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Major</td>
<td>Minor</td>
<td></td>
</tr>
<tr>
<td>C0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td>&lt;=5</td>
<td>&lt;=50</td>
<td>0</td>
</tr>
<tr>
<td>C2</td>
<td>&lt;=10</td>
<td>&lt;=100</td>
<td>&lt;=4</td>
</tr>
<tr>
<td>C3</td>
<td>&lt;=20</td>
<td>&lt;=200</td>
<td>&lt;=8</td>
</tr>
</tbody>
</table>

Table 8:
5.4.2. Electrical Design Details

A lens is thought to be very old and well understood technology. During the course of the TUGV project we learned that there are some potentially important interactions between a mechanical iris, the type of lens elements and the image distortion introduced by the lens. Simply stated, the setting of the mechanical iris in the lens has an effect on the lens distortion. Normally this effect would not be noticed but as the resolution of the imager increases or the field of view of the lens increases this becomes a problem. A change in the brightness of the scene would cause a change in the image distortion which would introduce an error in the perceived position of objects. Additionally if the irises of the lenses of the stereo pair have different settings the difference in distortion could cause poor matches between the images.

With the resolution and lenses used for TUGV this effect was not yet a problem. But it will become a problem as resolutions become greater and field of views become wider. To completely avoid this problem a manual iris lens was selected and an electronic (rather than mechanical) iris control in the camera is used to compensate for scene brightness. The camera specified has the inputs required for this control but a custom unit would be required to accept commands from the stereo vision algorithms. This control has the added benefit of allowing the stereo vision algorithms to dynamically adjust the brightness of the scene so that the area of greatest interest has the best contrast.
However, with a manual iris lens, an additional procedure would be required. The cameras would be pointed at typical terrain and the mechanical iris would be adjusted until the exposure is in the middle of the electronic iris control range. This would likely be done just once since the adjustment range of the electronic iris is similar to that of the mechanical iris.

The components of this design have been selected to work from a 12 volt battery supply since DRES robotic vehicles are generally not large enough to carry gasoline or diesel powered generators. The ability to work with 12 volt power also includes other sources such as generators and AC power supplies. With respect to power requirements, it is important to note that the SPARC VME card will be running Unix (Solaris) and will require maintenance of power and proper shutdown protocols to avoid corruption of the storage devices and loss of data.

The TUGV program did not use the TALIN model INS specified here. Instead, the MAPS model, also made by Honeywell, and the MIAG model from Lear/Siegler, were used due to past experience and for compatibility with existing systems. Therefore, we can only say that Honeywell verified that it is possible to communicate with the TALIN via standard RS-232 protocol.

Table 9:

<table>
<thead>
<tr>
<th>Item</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lens</td>
<td>Cosmicar</td>
<td>0618-C/C20612</td>
<td>6.5mm, f/1.8, manual iris, fixed focus, 1&quot; format</td>
</tr>
<tr>
<td>Camera</td>
<td>Pulnix</td>
<td>TM-1001</td>
<td>b/w, 1024x1024, Class 3 sensor, standard functional options c/w 30DG-02 cable</td>
</tr>
<tr>
<td>SDV Board</td>
<td>EDT, Inc</td>
<td>SDV</td>
<td></td>
</tr>
<tr>
<td>Iris Ctl.</td>
<td>GFE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INS</td>
<td>Honeywell</td>
<td>TALIN 2500</td>
<td>Embedded GPS, Vehicle Motion Sensor</td>
</tr>
<tr>
<td>Ancaeus interface</td>
<td>GFE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VME SPARC Board</td>
<td>Themis Computer</td>
<td>Sparc 10MP</td>
<td>SuperSPARC or HyperSPARC, 100 MHz, 2 S-Bus slots, 1 Ethernet, 4 serial, 1 SCSI-2, supports Solaris or VxWorks</td>
</tr>
</tbody>
</table>
5.4.2.1. Supplier Addresses

Cosmicar (available from Pulnix)

EDT, Inc
Engineering Design Team, Inc
1100 N.W. Compton, Suite 306
Beaverton, Oregon 97006
(503) 690-1234
FAX: (503) 690-1243

Pulnix
PULNiX America Inc.
1330 Orleans Drive
Sunnyvale, CA 94089
(408) 747-0300
(800) 445-5444
FAX: (408) 747-0880

Honeywell
Military Avionics
Honeywell Inc.
2600 Ridgway Parkway
Minneapolis, MN 55413
(612) 951-5226
FAX: (612) 951-5110

Themis Computer
Themis Computer
3185 Laurelview Ct.
Fremont, CA 94538
(510) 252-0870
http://www.themis.com

5.4.3. Mechanical Design Details

The only critical aspect of the mechanical design is the camera mount and this is critical in two areas: rigidity and repeatability of the camera positions and rigidity of the mechanical connection of the position sensors to the cameras.

The final version of the TUGV camera mount and the vertical baseline stereo implementation is recommended for DRES use. It is a 42 inch long, 2.5 inch diameter aluminum tube with index holes cut about every 6 inches down the length of the tube. The cameras are mounted on circular clamps with pins that index into the holes in the tube. This design allows the cameras to be removed easily which is a common requirement in the TUGV project. A picture of this assembly is available on the TUGV project home page (http://www.frc.ri.cmu.edu/tugv/).
Note in that picture that the cameras are mounted on their sides. This started as a convenience while the decision between horizontal and vertical baseline stereo was still being made. In the case of DRES, but due to the Class 3 image sensor and the associated Imager Specification Note 1, it may also be better to have the camera on its side.

In the case of the TUGV vehicle, a military HMMWV ambulance, the metal shell is sufficiently inflexible that the position sensors are tightly coupled to the camera mount which is mounted to the front of the shell.

The orientation of the magnetometer and the INS are not a significant concern since the INS accepts sensor orientation information at start-up and corrects its world view from this. Obviously the magnetometer is a magnetic sensor and should not be placed near any magnetic field sources, ferrous materials or magnetic shielding materials.
6. Conclusions and Future Work

This section briefly summarizes the main conclusions of the project and points to some possibilities for future work. Initial experimentation on the TUGV has verified the ability to navigate autonomously with the developed hardware and software components and has pointed to several specific areas that have good potential for performance enhancement. These include:

- **Processor Upgrades**: The need for high performance computing has been known since the start of the program. Given the tradeoff that exists between the throughput and the resolution of computations, the engineering approach has been to sacrifice resolution in order to meet response time requirements. In the long run, both resolution and speed requirements must be met. The general strategy has been to acquire the highest performance computing available that is consistent with the overall goals of the program. In future work on autonomous navigation, it will be important to upgrade computing to track the inevitable growth in processor technology.

- **Camera Upgrades**: Digital high resolution cameras were employed on the program in order to exploit the expected throughput of the Boeing MCC processor hardware. Experience with these cameras has indicated that interface hardware and auto-iris circuitry are immature - at least for the purchased components. Longer term, it will be necessary to upgrade the camera technology to remove these shortcomings. A second aspect of camera performance with a long history in the field is the horizontal field of view. A basic requirement of obstacle avoidance is that of imaging all terrain that the system will or can traverse. Yet, no stereo or camera system has been able to achieve this requirement. Available techniques include multiple stereo pairs, panning sensor heads, and omnidirectional sensors. A custom designed camera with wide horizontal field of view promises considerable benefits and seems to be one of the few promising long term solutions.

- **Integrating Off-the Shelf Stereo and Laser Rangefinders**: Instead of continuing to pursue the development of stereo vision, it may be possible to use systems developed at other laboratories in order to concentrate on the navigation aspects of the problem. Similarly, laser rangefinders provide a range imaging capability in hardware that, today, is superior in performance to stereo in terms of range and angular resolution, throughput, and accuracy. The disadvantages of laser rangefinders include higher power consumption and active irradiation of the environment. For this reason, stereo vision continues to compete with lasers in space and military scenarios where either low power consumption or stealth are important. Given that each sensor has its relative merits, it may be useful to investigate sensor fusion algorithms that will process data from both sensor modalities simultaneously in order to exploit both.

- **Strategic Planning**: It has long been known that strategic planning capabilities are necessary in scenarios where prior terrain information is not available and beneficial in scenarios where such information is available. As a navigation controller, Ranger incorporates strategic control (path tracking) but no strategic planning element. Ranger must be given a path to track which is relatively free of complicated obstacle mazes, or it is subject to the well known inability of local planners to avoid “local planning minima”. However, strategic planning has been extensively developed and demonstrated on the ARPA UGV program in a module called Dstar which was specifically developed to address this problem. Ranger and
Dstar have been identified as key components on autonomous navigators of enhanced autonomy and the integration of these components is a natural next step in the evolution of the technology.

- **Continued port to Ada.** The goals of the program have included a partial port of the Ranger system to the Ada programming language. A natural extension of the work is to port the remaining components of the system and to bring up the Ada software on a real vehicle.

- **Identification/ Characterization:** Two areas that have received limited attention in the past, but are nonetheless important are calibration of the vehicle dynamic models and tuning of the obstacle avoidance signal processing. For the former, it is necessary for the software to have sufficiently accurate models of the pure delays and dynamic response of various actuators. In the latter, a comprehensive examination of the ability to resolve obstacles from range images from various sensors is needed. These two aspects of modelling directly affect the performance of obstacle avoidance.