Journal of Electronic Imaging 19(4), 043024 (Oct–Dec 2010)

Function-based design process for an intelligent ground vehicle vision system

Robert L. Nagel
James Madison University
School of Engineering
MSC 4113, HHS 3224
Harrisonburg, Virginia 22807
E-mail: nagelrl@jmu.edu

Kenneth L. Perry
Missouri University of Science and Technology
Department of Computer Science
300 West 12th Street
203 Centennial Hall
Rolla, Missouri 65409

Robert B. Stone
Oregon State University
School of Mechanical, Industrial and Manufacturing Engineering
406 Rogers Hall
Corvallis, Oregon 97331

Daniel A. McAdams
Texas A&M University
Mechanical Engineering Department
Engineering/Physics Building 108, MS 3123
College Station, Texas 77843

Abstract. *An engineering design framework for an autonomous ground vehicle vision system is discussed. We present both the conceptual and physical design by following the design process, development and testing of an intelligent ground vehicle vision system constructed for the 2008 Intelligent Ground Vehicle Competition. During conceptual design, the requirements for the vision system are explored via functional and process analysis considering the flows into the vehicle and the transformations of those flows. The conceptual design phase concludes with a vision system design that is modular in both hardware and software and is based on a laser range finder and camera for visual perception. During physical design, prototypes are developed and tested independently, following the modular interfaces identified during conceptual design. Prototype models, once functional, are implemented into the final design. The final vision system design uses a ray-casting algorithm to process camera and laser range finder data and identify potential paths. The ray-casting algorithm is a single thread of the robot's multithreaded application. Other threads control motion, provide feedback, and process sensory data. Once integrated, both hardware and software testing are performed on the robot. We discuss the robot's performance and the lessons learned.* © *2010 SPIE and IS&T.* [DOI: 10.1117/1.3528476]

Paper 09217PR received Nov. 10, 2009; revised manuscript received Oct. 14, 2010; accepted for publication Oct. 29, 2010; published online Dec. 28, 2010

1017-9909/2010/19(4)/043024/13/$25.00 © SPIE and IS&T.

1 Introduction

Vision is a powerful sense providing an abundance of information about one's surroundings and affording intelligent interaction without ever requiring physical contact. It is through vision that one learns of potential obstacles or pitfalls, and one is able to investigate our 3-D world.[1] Affording this same level of perception to intelligent vehicles is fundamental to achieving full autonomy. Intelligent vehicles are required to perceive and interpret unknown environments such that representations can be generated from which decisions can be made, paths can be planned, and operations can be carried out.[2] Researchers at Stanford,[3,4] Carnegie Mellon,[5,6] Cornell,[7,8] Virginia Tech,[9,10] and other Defense Advanced Research Projects Agency (DARPA) Grand[11] and Urban[12] Challenge participants have developed successful vision systems for intelligent vehicles. However, their design documents provide only limited insight into each team's actual design process.

This paper focuses on using engineering design methodologies as the foundation for intelligent vehicle design. Both conceptual and physical phases of the engineering design process are described and demonstrated by following the function-based design process used to develop the Intelligent

Ground Vehicle Competition (IGVC) robot, MAX, and more specifically, MAX's vision system. MAX was developed following the guidelines of the 2008 IGVC (Ref. 13) as a part of two graduate-level design courses at Missouri University of Science and Technology (Missouri S&T). The IGVC is an annual competition held at Oakland University in Rochester, Michigan, where college students design and construct intelligent ground vehicles to compete in up to four competitions: robot design, autonomous challenge (obstacle course), navigation challenge [global positioning system (GPS) waypoint course] and JAUS (Joint Architecture for Unmanned Systems) communication.[13] This upcoming 2011 IGVC will be the 19th annual competition.

MAX was developed to compete in the autonomous challenge where vehicles are expected to negotiate around an outdoor course within a prescribed time limit without exceeding 5 mph. Entries are expected to be fully autonomous; thus, they must navigate the course by perceiving the environment and avoiding all obstacles. Neither obstacles nor course layout is to be preprogrammed; robots are to navigate in an unknown course environment and avoid unknown obstacles. All computational power, sensing, and control equipment for the robots must be carried onboard the vehicle. Either white or yellow lines mark the course, and obstacles typically consist of construction makers, barrels and cones.

MAX is designed as a modular and scalable robot, in both hardware and software. The software developed for MAX is a multithreaded application, which controls and collects data from each sensor, provides feedback via a graphical user interface, and controls the robot's operation based on its perceived environment. The multithreaded application provides the benefit of software modularity; when new devices are added to the robot or a new robot operation is desired, the entire code base does not require modification. Instead, a new thread can be added to control the new hardware device and ready information for the existing threads to access. Alternatively, if new operations are desired, new threads can be added to access existing information and command the desired operations. Hardware modularity is achieved though common power and communication busses for all sensors, and drive and control components.

Since vision and vision processing systems are fundamental to robots' autonomy, this paper primarily focuses on following a systematic engineering design process as the foundation for the design, development, and implementation of the vision systems for our 2008 IGVC entry. Section 2 discusses the design methodology followed. Section 3 discusses conceptual design consisting of functional and process analysis. Process analysis leads to design elements dealing with user-based interactions of the robot such as setup, configuration, calibration, and emergency procedures. Functional analysis leads to specific solution strategies such as a modular architecture as well as the implementation of a vision system, which consists of a camera and a laser range finder and a path-determination system based on ray-casting. Section 4 describes the physical design phase, which includes the vision system, image processing, and path-planning algorithms. Testing performed on the robot vision system is discussed in Sec. 5, and concluding remarks are provided in Sec. 6.

2 Design Methodology

The design methodology of the vision system for MAX, our IGVC robot entry, is initially intertwined with the overall design process for the entire intelligent ground vehicle. This design process, adopted loosely from Otto and Wood's *Product Design* text[14] begins in conceptual design with the identification of customer needs. Functionality is used to translate the customer needs into the engineering domain, and product modularity is identified from the product functions following function-based, modular heuristics.[15] Concepts are then developed, evaluated, and iterated before a final concept can be chosen. Once a final concept is chosen, the design process moves into the physical design phase where first design parameters are detailed, then components are specified and prototypes are developed to test the design. The design may be iterated any number of times before the assembly and testing of the final product. These two phases,

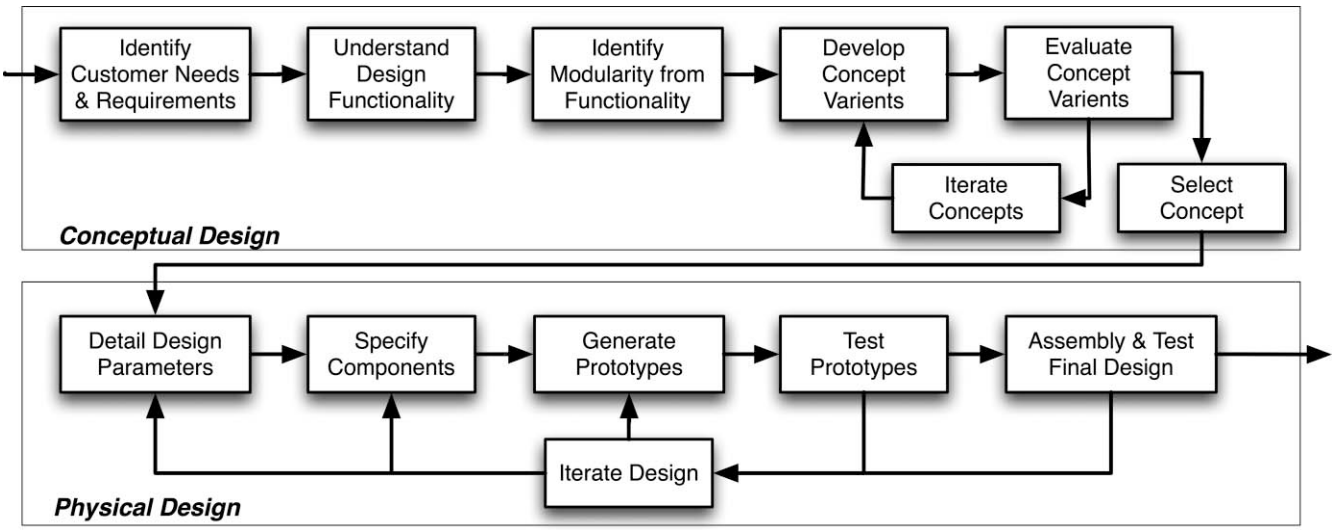


Fig. 1 Flow chart of the general design process taken to develop the robot and its vision system.

conceptual design and physical design, are illustrated in the flow chart provided as Fig. 1.

Relating the design methodology to the intelligent ground vehicle, first during conceptual design the requirements, needs and outcomes are identified from the rules and objectives of the IGVC. These needs are then translated to a functional representation of the robot, which is used as an intermediary to (1) identify both software and hardware modules in the final design and (2) aid with the identification of solution principles following Zwicky's morphological approach.[16] Solution principles are assembled into complete concepts, which are evaluated with respect to their feasibility and ability to meet the identified requirements, needs, and outcomes. The best designs are iterated, and a concept is selected. Physical design begins with the design parameters (i.e., target specifications) being identified from the IGVC guidelines. The exact components are selected to meet the chosen solution principles and the design parameters. When solutions cannot simply be chosen or purchased, prototypes (physical and/or virtual) are developed, and tests are performed to validate the design attributes. With this validation information, the design is iterated to ensure a final desired level of quality is attained. Finally, in the case of our IGVC robot, the final design is constructed—for designs that would not be a one-off, further design efforts, such as design for manufacture, are required.[17–20]

3 Conceptual Design Phase

Fundamental to the conceptual phase of the engineering design process is the utilization of functional and process modeling. Functional modeling, found in many engineering design texts,[14,17,18,21–25] provides a description of flow transformations occurring within the product being designed. Functions describe the transformation of input flows available into the output flows desired.[14] Process models, however, are similar to activity diagrams,[14] and focus outside the product on interactions with the environment and the customer to describe the specific product transformations required for operation.[26] The sum of all functions identified for a design describe what the product must do, while processes describe what will be done with the product. For example, the IGVC requires that each robot carry a payload. Functional modeling would describe how the payload is stored, secured, and released within the robot, while process modeling would describe the actions required of the operator to place the payload within the robot. More specifically, functional and process models are defined as follows.

1. Functional modeling is the overall approach to modeling what a product must do in terms of elementary operations such that the product may achieve an overall goal or purpose.[27]
2. Process modeling is the overall approach to modeling why a product is required as a series of customer-driven, product-based operations related through input and output flows, the product being designed, and time.[26]

During conceptual design, process and functional models act as intermediaries to translate the customer speak, referred to as "statements of need," into engineer speak describing what a product must do to meet these customer needs. Both functional and process models are based on what a product must do instead of how (solution principles and components) it will be achieved. Considering both process and function in the design process provides many benefits, including explicit identification of customer need, comprehensive understanding of the design problem, enhanced creativity through abstraction, innovative concept generation with a focus on answering what criteria must be met, and a structured organization that can be applied to both the design problem and the design team.[14,28] Specifically, for the design of MAX, function and process enable the robot's design to be considered from a fresh perspective instead of simply a redesign of prior IGVC entries, and while this may not immediately produce a winning robot design, it does bring new and, hopefully, innovative ideas and outlooks into an established competition.

Table 1 Raw customer needs and flow mapping for the vision system.

Raw Customer Need[13]	Flows
Robot needs to avoid hitting the vertical obstacles (e.g., construction drums) placed on the course	Solid material (vertical obstacle), status signal (vertical obstacle)
Robot needs to visually discern the course boundaries	Solid material (ground with or without boundary), status signal (boundary presence information)
Robot needs to visually discern standing obstacles within course boundaries	Solid material (vertical obstacle), status signal (vertical obstacle)
Robot needs to stay within boundaries of the course during operation	Status signal (boundary presence information), control signal (robot direction)

To generate the functional and process models, first the raw customer needs must be understood. For the IGVC robot, these raw customer needs are extracted from the competition rules.[13] For example, raw customer needs for the vision system of an IGVC robot are provided in Table 1. The raw customer needs are then mapped to specific flows required for both the functional and process representations of the robot. For example, the customer need, "Robot needs to avoid hitting the vertical obstacles (e.g., construction drums) placed in the course," maps to the flow, solid materials, representing the actual vertical obstacles inside the robot's sphere of influence and the flow, status signal, representing the information collected by the robot on the vertical obstacles.

The functional[22] and process[26] models generated from these flows take the form of diagrams describing the transformation of flows. Models contain three specific flow types: material flows represented with bold arrows, energy flows represented with thin arrows, and signal flows represented with dashed arrows. Each block in the models represents a change that must occur to each flow to achieve the desired outcomes specified by the customer's "statements of need." For consistency, the nomenclature for the terminology is taken from the functional basis lexicon,[29] the functional basis provides a standard lexicon consisting of function and flow terms for the generation of function-based models.

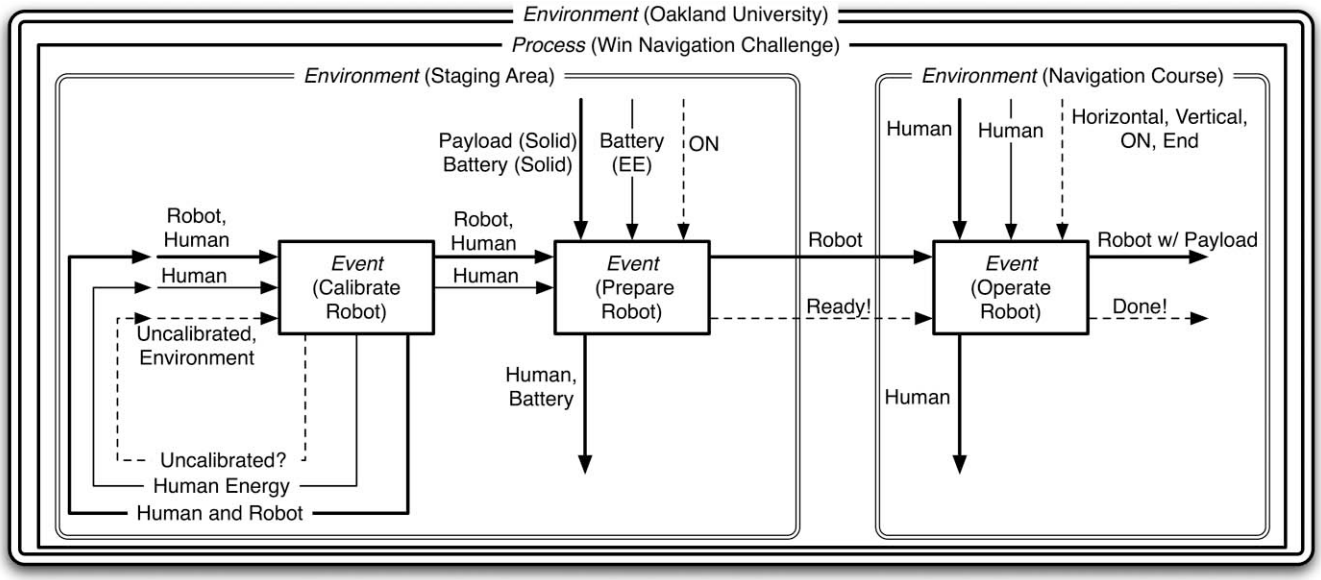


Fig. 2 Environment and high-level process model for the robot.

For MAX, a first-pass functional model detailing how the robot is expected to operate is generated, followed by the generation of a first-pass process model detailing the required interactions with the robot (e.g., setup, calibration, start/stop, and emergency procedures). The high-level model (shown in Fig. 2 for the IGVC robot) describes the environments as well as the process and subprocesses that detail where and how the robot must operate. These subprocesses are termed events.[26] The robot will be operating at Oakland University and there will be two subenvironments: (1) a staging area where the robot can be calibrated and set up and (2) a navigation course where the robot actually competes in the navigation challenge. To calibrate the robot, the operator and environmental information is required as well as a signal indicting that the robot is uncalibrated. Once calibrated, the robot may be prepared. Preparation involves charging or replacing batteries, adding a payload to the robot, and setting the robot to an *on* or *ready* position to enable the robot to be started with a single key stroke. In the navigation challenge, surface information, obstacle information, boarder information, and potentially, emergency stop information are required. Once done with all of the events, the robot with its payload leaves the win navigation challenge process.

Each of the events (subprocesses) in the process model (shown in Fig. 2) is further decomposed to consider specific changes (termed configurations) required of the robot to fulfill the desired outcomes from each event.[26] For example, the process, calibrate robot (provided as Fig. 3) describes the human operator interacting with the robot to actuate a calibration routine. Information from the environment, represented as a dashed line, is collected by the robot. Once this information is collected, the information is transmitted as a status signal back to the operator. The operator can then use the information to regulate (i.e., calibrate) the robot for

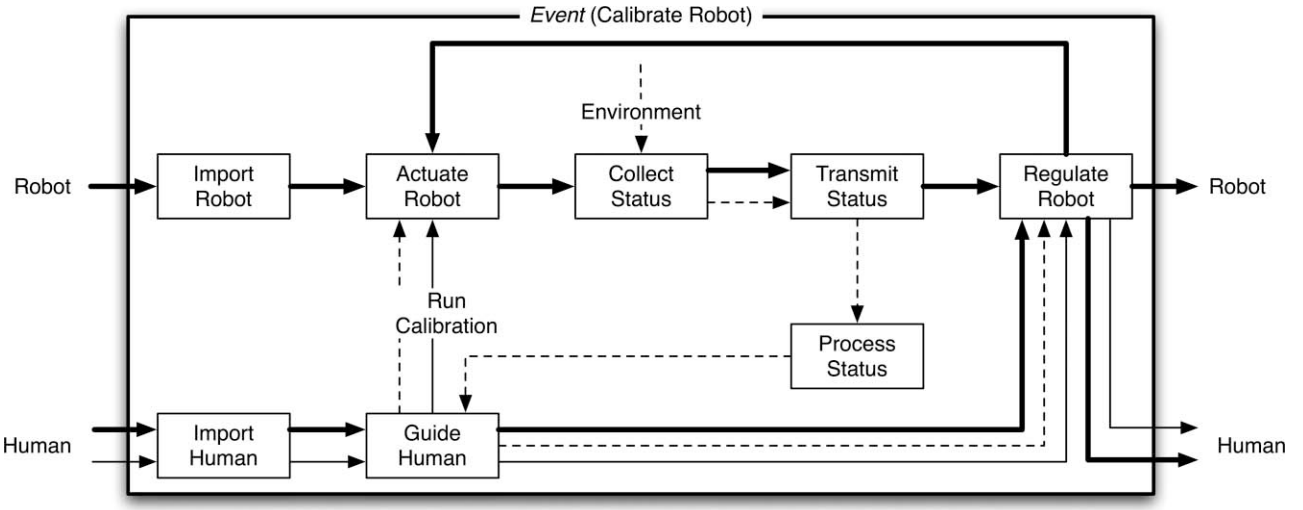


Fig. 3 Configuration model for the calibration robot event of the win navigation challenge process.

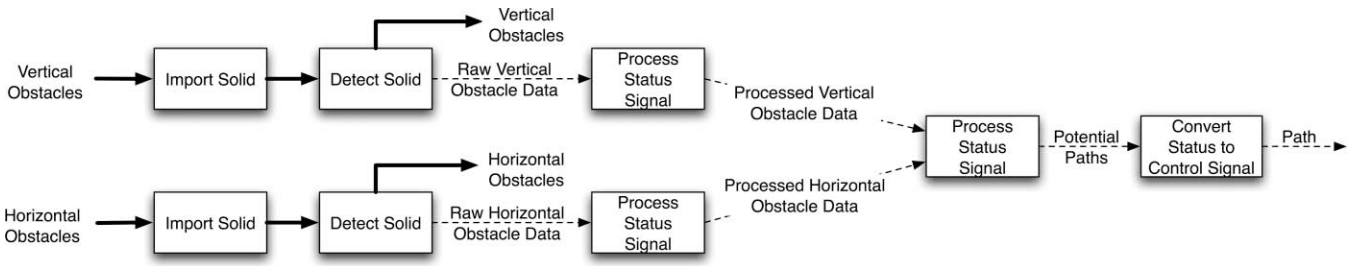


Fig. 4 Vision system and path determination portions of the functional model for the robot.

its operating environment. This process model describes the customer's interaction through an interface on the robot to understandal environmental conditions and to make adjustments to the robot's settings.

Configuration information can then used to update the functional model to include flow transformations related to user interactions with the robot. The final functional model of the IGVC robot includes flow transformations for all of the identified needs of an IGVC robot as well as functionality for the detection of solids (vertical and horizontal obstacles), processing of the status for understanding and making decisions about navigational data, and actuation of electrical energy to represent both the startup and the termination (both intended and emergency) of the robot.

Abstracting the robot functions rather than starting with the specific components used on similar robots leads to the separation of the vision tasks that must be performed by the robot to achieve autonomy. As functions were chosen to abstract the IGVC requirements into a functional model of a robot, two types of obstacles are identified. One type are those that the robot must drive around; the other type are boundaries that the robot must stay within. The functionality for the model was derived to detect these two types of obstacles; they are (1) the detection of obstacles visible in a vertical plane (e.g., cones, construction pylons, barrels) and (2) the detection of obstacles visible in a horizontal plane (e.g., white and yellow lane lines). The obstacle detection and path determination portion of the functional model is provided in Fig. 4. The vision functional model abstracts the processing of vertical obstacles with three function blocks. First, import solid, represents vertical obstacles coming into the sphere of influence of the robot. Detect solid represents the robot's collection of information. The collected vertical obstacle information takes the form of a status signal (dashed arrow). The status signal is processed before it is combined with the horizontal obstacle data—also a status signal represented by a dashed arrow. The process status signal block provides an abstraction for the robot's attempt to find potential paths from the combined vertical and horizontal data, and the convert status to control signal is the robot making a decision on which path it should traverse.

One of the major design goals of our IGVC robot is to develop an architecture that is modular both in the hardware and software. To identify potential modularity during the design process when it is more simple to implement, the functional model was divided into "chunks" via the application of modular heuristics.[15] Modular heuristics provide steps to identify modules based on potential flow paths through a functional model; these flow paths are (1) flow dominance—the flow passes through a sequence of functions with its state largely unchanged, (2) flow branching—the flows diverge or converge at a function in the model, and (3) flow conversion—the flow's type changes (e.g., liquid changes to gas). Figure 5 provides the functional model of the vision system of our robot with potential modules identified via modular heuristics.

Through the application of modular heuristics, five potential modules for the vision system are identified from the functional model. Two hardware modules are identified via the flow conversion heuristic whereby the solid obstacle flows (bold arrows) are converted to status signals (dashed arrows) to represent the flow of raw obstacle data into the robot's computer. One of these hardware modules represents the detection of vertical obstacles while the other represents the detection of the horizontal obstacles. Three additional modules are identified for the software: (1) one for the processing of raw vertical obstacle data, (2) another for the processing of raw horizontal data, and (3) a third for the analysis of the processed obstacle data. These distinct modules lead to a multithreaded software design where hardware access and

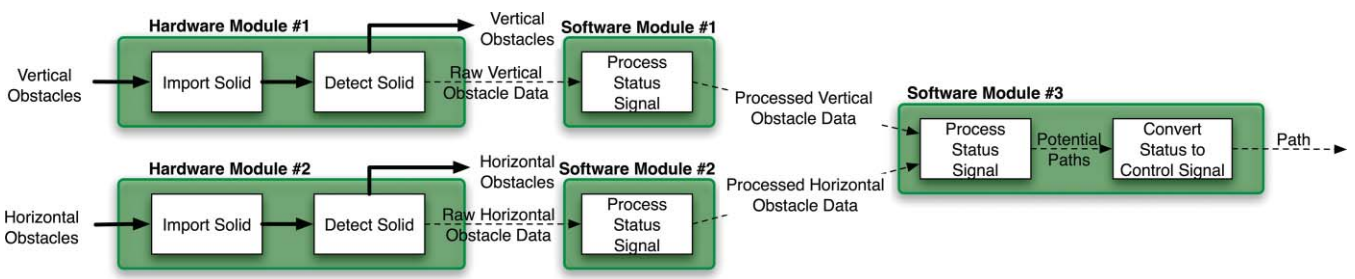


Fig. 5 Potential modules identified during functional analysis. (Color online only.)

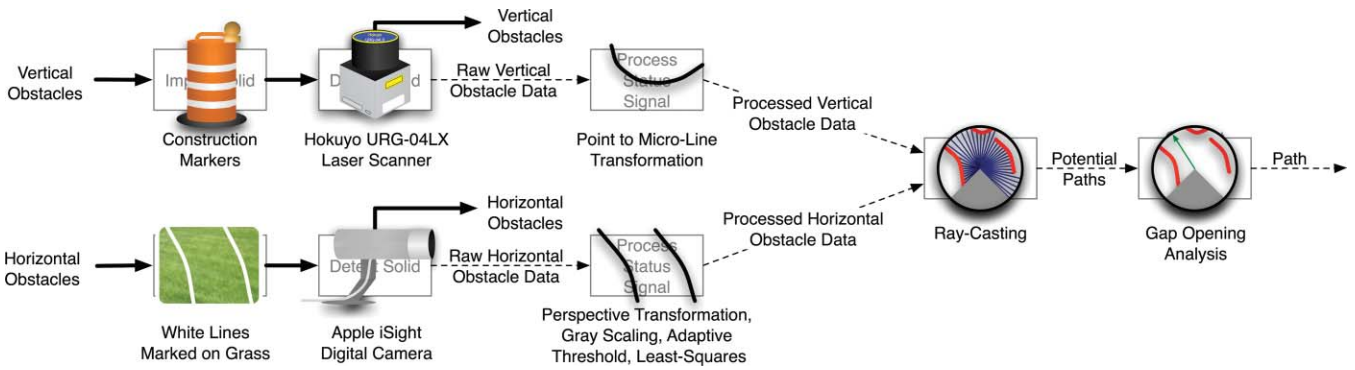


Fig. 6 Component-to-function mapping for the vision system and path determination portions of the functional model of the robot. (Color online only.)

computation requirements are each processed on separate threads. This design enables each device to work independently and concurrently.

To identify components, the functional model is used as a framework of what the robot must do. Research is performed to identify solution strategies consisting of components and algorithms that answer how to meet the identified functionality. A number of different solution strategies are identified for each function in the functional model. These solution strategies are mixed and matched via a morphological matrix[16] to develop unique concepts, which are then evaluated based on their feasibility and ability to meet the identified requirements. The concepts, which are identified as having the most potential, are refined until the best solution for the identified criteria is selected. While this ideation is performed for the entire functional model to generate a final design, for simplicity, it is shown in Fig. 6 only for the vision system and path determination portions of the functional model. Once solutions are chosen for the entire functional model, they are pieced together to develop an initial concept for the intelligent ground vehicle. It is this initial concept that now moves into the physical design phase.

4 Physical Design Phase

During the physical design phase, the conceptual design is now developed into the final product. Following the design parameters set by the guidelines of the IGVC, specific components are identified for each of the solution concepts. These components or strategies are described in text under the solution concepts in Fig. 6. A construction barrel represents the vertical obstacles that might be encountered on the obstacle course; these vertical obstacles are detected via a Hokuyo URG-04LX laser scanner.[30] White lines represent the horizontal obstacles that might be encountered and are detected by an Apple iSight digital camera.[31] The raw data from the laser scanner and the iSight will be processed independently (discussed in Sec. 4.1 and 4.2) before being analyzed for potential gaps where the robot can traverse. A ray-casting algorithm is implemented for the robot's path identification algorithm (discussed in Sec. 4.3).

Further development focuses on the assembly and testing of prototypes for each system. Prototype assembly and testing follows the hardware and software modules identified as functional modules during conceptual design. Thus, each piece of the vision and path determination systems are independently assembled and tested before their final assembly into the robot.

The design process continues similarly on the software side. For example, consider all aspects of the software linked to the vision system. The model of the software system (illustrated in Fig. 7) includes six of the robot's seven threads. (The GPS thread, which is not used with the vision system, has been omitted for simplicity.) The seven threads include: (1) main/dashboard—responsible for providing a visual interface for the user; (2) GPS—responsible for controlling the universal serial bus (USB) serial connection with the GPS device; (3) LIDAR (light detection and ranging)—responsible for collecting and processing vertical obstacle information (discussed in Sec. 4.1); (4) camera—responsible for collecting and processing horizontal obstacle information (discussed in Sec. 4.2); (5) path finding—responsible for finding identifying potential paths in the collected sensory data (discussed in Sec. 4.3); (6) motion control—responsible for providing pulse width modulation (PWM) signals to the motor controllers; and (7) artificial intelligence (AI)—responsible for making decisions based on data received from the pathfinder and passing on decisions to the motion control thread. Each of these threads in the software is independently developed and tested (discussed in Sec. 5) both in software and with its respective hardware device before being assembled into the complete robot software system. Physically, the final code is split into two parts: (1) a library for handling the device controllers, the path finding, motion control, and AI, and (2) a main application that combines all of the elements in the library with a graphical user interface (dashboard) and operator control. All software is written in C and C++ using open source libraries.

The convention followed in Fig. 7 is that rectangular boxes are actions to be performed, while diamond boxes are flow control based on a condition. Conditions include single-case-based true/false or the completion of groups of tasks. Dashed boxes represent the data of a thread and are used to share information between threads. Dashed arrows represent the generation of a new thread or forcing a no longer required thread to rejoin the main thread.

It is the integration of both hardware and software that is key to a successful vision system. The following subsections describe the following three key parts of our IGVC robot's vision system: (1) the laser-scanner-based vertical vision

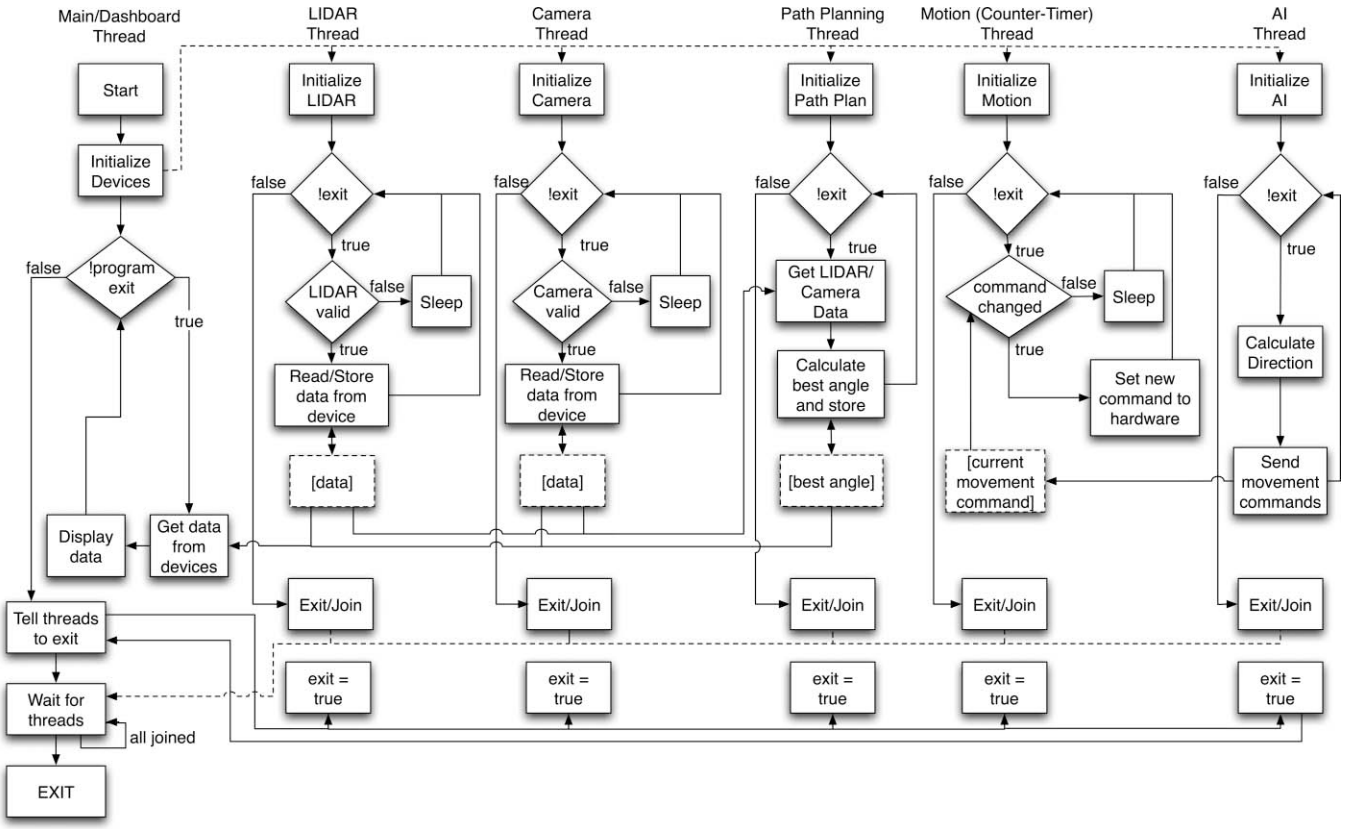


Fig. 7 Analytical model of the modular software's thread layout.

system, (2) the camera-based horizontal vision system, and (3) the ray-casting-based path determination algorithm. Details on each of these key vision system parts are discussed in detail in the following subsections.

4.1 *Vertical Obstacle Vision System*

For the vertical obstacle detection, the vision system is required to detect physical obstructions within a 180-deg field of view from the front of the robot. For this, a Hokuyo URG-04LX laser scanner or LIDAR system is chosen. The Hokuyo URG-04LX (Ref. 30) provides a 240-deg field of view with a 0.3516-deg angular resolution from 20 to 4090 mm. The LIDAR is mounted at front center of MAX beneath a shield to block direct sunlight (as shown in Fig. 8), and is connected to the robot's onboard computer, an Apple MacBook, via a Sewell SW-1301 RS232 to USB adapter.[32]

The thread that interfaces the robot's software with the LIDAR receives the distance and angle for each ray that strikes a vertical obstacle from the LIDAR's driver. This information, stored as an array, is filtered to remove erroneous hits. For each sweep of the LIDAR, erroneous hits occur in one of two ways. First, erroneous hits occur from self-visualization; the LIDAR sees itself and the front edges of the robot. Second, erroneous hits occur when the LIDAR detects obstacles occurring beyond the calibrated vision range of the robot. These erroneous hits are either very close to the robot (approximately 0.25 m) or very far (approximately 4 m). The actual LIDAR threshold range values are empirically

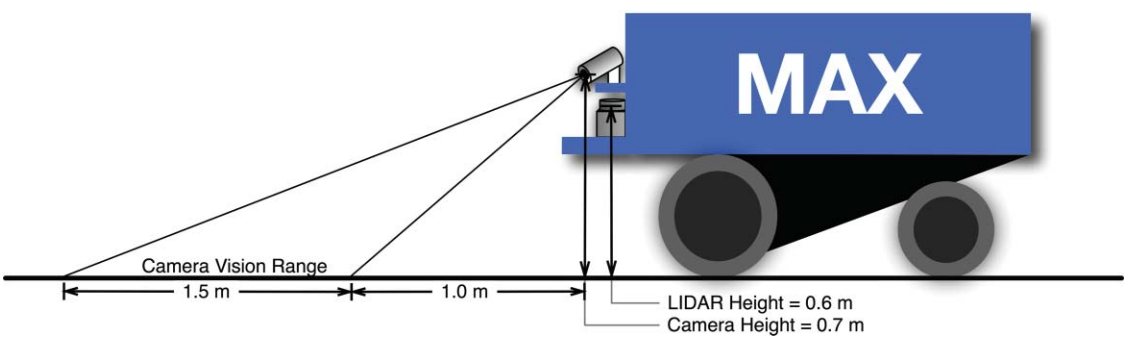


Fig. 8 Vision system layout for the robot. (Color online only.)

Fig. 9 Left: original image of the 1- × 1-ft (30.48-cm) checkerboard tiled floor with distortion; right: image of the checkerboard tiled floor with the perspective corrected. (Color online only.)

identified, and hits beyond them are removed with a bandpass filter.

Once the array of points is filtered, the remaining points are converted and scaled to fit into the local coordinate system. The local coordinate system, which places its origin at the front center of the robot, is based on the position of the robot whereby the angle of the robot's heading is always 0 deg. Since the LIDAR system is firmly affixed at the origin and moves with the robot, its coordinate system is already aligned with the robot's and requires only scaling. Each hit from the LIDAR is scaled a 1/5600 unit. This scaling factor was determined experimentally.

4.2 *Horizontal Obstacle Vision System*

Horizontal obstacles (i.e., those located on the ground in front of the robot) at the IGVC tend to consist primarily of white lines; however, there could potentially be yellow lines, sand traps, and potholes. To detect these obstacles, an Apple iSight is chosen. The Apple iSight [31] is autofocusing from 50 mm to infinity with automated shutter speed adjustment and a video capture rate of 30 frames/s. The iSight is mounted directly above the LIDAR at front center of MAX, as shown in Fig. 8, and is connected to the robot's onboard computer via a firewire.

To interface with the iSight, the Open Computer Vision (OpenCV) library, developed initially by Intel, is implemented.[33] OpenCV provides all of the tools and algorithms required to convert the raw image captured from the iSight into the line-based obstacle information required by the ray-casting algorithm for path determination. On initialization of the camera vision thread, the Apple QuickTime drivers[34] start the iSight video stream, and OpenCV begins capturing individual frames from the live video feed. Since the iSight is positioned at a relatively shallow angle with respect to the ground, each image must be corrected for distortion. To correct this, a distortion matrix fits the image within a trapezoid to correct perspective. The correction ratios are determined via an empirical study whereby images are taken of a 1-× 1-ft (30.48-cm) checkerboard tiled floor with increasing distortion ratios until the tiles appear to extend vertically toward the horizon. This distortion correction is shown in both the original and perspective corrected forms in Fig. 9.

To analyze each image for horizontal obstacles, an adaptive threshold algorithm is applied. The adaptive threshold converts a gray-scale image into a binary image by filtering out pixels with values above or below a limit calculated from each pixel's pixel neighborhood.[33] Thus, groups of pixels of similar "color", are filtered from the image (turned to black) and pixels, where the "color" changes are left in the image (turned to white). The application of an adaptive algorithm allows the robot vision system to see all changes in color, i.e., white lines, yellow lines, sand traps, and potholes from the grass. In Fig. 10 (left) the adaptive threshold algorithm has been applied to the checkerboard floor image provided in Fig. 9 (right).

4.3 *Vision Processing—Path Finding*

To determine gaps or openings around obstacles, a ray-casting algorithm is utilized. Ray-casting, which was initially pioneered by Goldstein and Nagel,[35] is a technique originally designed for simulating light in reverse for use in 3-D graphics. Ray-casting is scalable, however computationally expensive,[36] but its simplicity lends its application well for applications ranging from virtual navigation to large intelligent ground vehicles such as with the Defense Advanced Research Projects Agency (DARPA) Urban Challenge.[37] A complete discussion of all of the vision applications of ray-casting is beyond the scope of this paper; however, a few of the vision applications where ray-casting algorithms have been applied include navigation of virtual environments such as with computer games,[38] pointer techniques,[39] and medical applications where computed tomography (CT) or magnetic resonance imaging (MRI) scans provide the virtual landscape[40] to navigating actual 3-D environments such as with intelligent ground vehicles.[2,37,41,42] In robotics, ray-casting algorithms are typically used like a virtual LIDAR scanning the sensor collected information such that either a 2-D or 3-D map can be generated from which navigation decisions can be made.[2,41]

Our application is similar to this; we have applied ray-casting to scan through the data collected by the LIDAR and camera vision systems to find the openings where the robot can most easily pass through the obstacles and continue along the obstacle course. To ready the LIDAR and camera data for the ray-casting algorithm (performed in the pathfinder thread), both data sets must be converted to lists of lines. This

Fig. 10 Left: image of the checkerboard tiled floor after the adaptive threshold is applied; right: image of the checkerboard tiled floor after least–squares regression is applied.

conversion to lines increases the likelihood that an identified object (which is stored as an infinitely small point) will be struck by a cast ray. For the LIDAR, the data set consists of an array of points. Each point is converted to a horizontal line of length 1/5600 units based off point center. The line length of 1/5600 is the same as the scaling factor to provide an overlap so that side-by-side points represent one continuous object. For the camera, even with the conversion to binary, significant noise remains present in the image. To deal with the noise, the image is sectioned into 32×32 (1024) squares. Squares with white space below a set threshold are ignored, while those above the threshold have a least-squares regression applied on the white pixels in the square. The threshold for errant white noise is adjusted for the robot's environment and must be recalibrated each time the robot is moved to a new testing or competition environment. The application of least-squares regression provides a "best fit" line for the white pixels in each of the 1024 squares as shown in Fig. 10 (right). These LIDAR and camera point-to-line conversions are performed within each hardware device's specific thread before being handed off as separate line lists to the pathfinder thread.

The pathfinder thread, which contains the ray-casting algorithm, runs separately from and simultaneously with the threads that control the aforementioned LIDAR and the camera, as shown in Fig. 7. Thus, as new information is being processed by the LIDAR and camera threads, prior information is being processed by the pathfinder. The pathfinder thread begins by collecting the last sets of lines processed by the LIDAR and camera threads. These two lists of lines are compiled into a single list before rays are cast at each line from every degree within the range of -135 to 225 deg. This is shown pictorially in Fig. 11. Each ray cast starts at the origin of our robot's coordinate system with an initial intersection point of infinity. The actual ending point for each ray in X and Y coordinates is calculated based on Eqs. (1) and (2), where θ is the angle of the ray cast, and 100 units is the length of each ray. Two lists are created to keep track of the rays that are cast: one for those that do not intersect lines—the openings list—and a second for those that do—the obstacles list.

$$\text{rayend}_x = -100 \cos\theta, \tag{1}$$

$$\text{rayend}_y = 100 \sin\theta. \tag{2}$$

For each ray cast, the algorithm calculates theoretical intersections between the ray and the lines based on obstacles. These theoretical intersections are the intersection points if the ray and the lines extend infinitely in both directions. If the theoretical intersection corresponds to an actual point on both the ray and a line, then the ray-casting algorithm has found an obstacle. Since each could potentially strike multiple lines (from camera, LIDAR or both), the algorithm checks to see if each intersection point is closer than any prior known intersection points. If the intersection point is indeed closer than a prior known intersection, then the intersection is stored in the list for intersecting lines—this is the obstacle list. If, however, the ray does not intersect with lines from the LIDAR or camera or the intersection is further from origin than a prior known intersection, the ray is stored in a second list—this is the openings list.

To determine the path the robot should take to miss obstacles, a tend angle, or angle the robot should turn to move around obstacles, is calculated. The tend angle is initially straight forward (0 deg on our coordinate system), and through operation, it tends toward the most forward path available for the robot to avoid obstacles. When the ray-casting algorithm finds an obstacle in the forward most path of the robot, the openings list is searched for the nearest

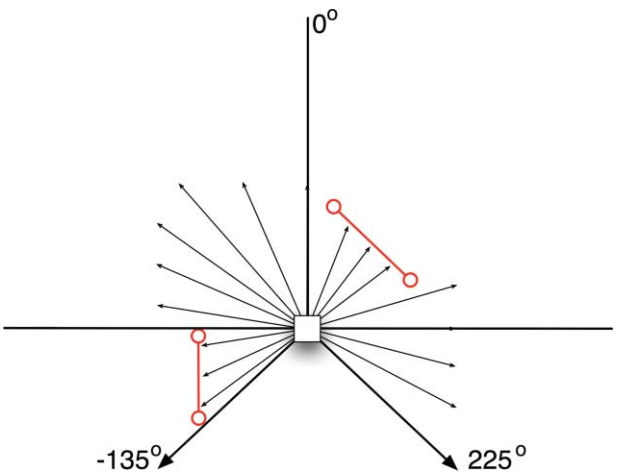


Fig. 11 Representation of the ray-casting algorithm applied to find lines visualized from obstacles detected via LIDAR and camera vision systems. (Color online only.)

opening between obstacles with enough room for the robot to fit. Once an opening is found, the horizontal component of the triangle formed at the opening by the angle of the initially cast ray is offset by 0.3 units; a new tend angle is calculated based on the initially cast ray and the offset. This offset, used to insure the robot will safely pass by obstacles, is determined to be 0.3 units from the width of the robot. The new non-0-deg tend angle is made available to the AI thread, where the direction that the robot should head is determined—either straight, hard or soft left, or hard or soft right. The direction is handed off to the motor control thread. The motor control thread controls the counter-timer hardware, and relays the direction information. On the hardware side, the counter-timer generates a PWM signal for the motor controllers, which ultimately drives the robot based on the PWM signal.

5 Testing and Results

As stated in Sec. 4, prototype assembly and testing during the physical design phase of MAX enabled each hardware and software module to be tested before implementation into the final design. Once the final implementation was assembled, testing was again performed with the entire platform to validate initial assumptions and to ensure that each module once integrated still performed as anticipated. Testing of each module and of the final design was performed similarly, using both virtual test scenarios to validate code logic as well as hardware tests to validate physical assumptions. Further details on testing and results for the vision system follow.

The final implementation of the robot image processing and path-finding systems are tested in two ways. First is software validation, whereby test case scenarios are run and results are verified. Second, the robot with its fully implemented vision system, is taken outside to a sample course. The robot is tested with a variety of vision scenarios with the results observed. This physical testing continued on the sample course at the IGVC before the actual competition. During the testing of the final implementation, there are no major modifications to the robot's systems. Instead, the only modifications were adjustments to configurations and filter ranges.

To validate the software, testing is performed at several levels. At the lowest level, unit testing is performed by using the UnitTest++ library[43] and test-driven development (TDD).[44] TDD is a development paradigm specifying software testing before the code is written. TDD specifies that tests should be run on every build to verify that code modifications do not result in broken code, where code previously functioned correctly (helping to prevent regressions). Under the TDD paradigm, a testing framework is constructed to verify the program, and tests are written that exercise the software in various ways to verify the code's "correctness". Tests are designed as short-to-test, small pieces of functionality (i.e., each test verifies just one aspect of the code). Having modular software facilitates TDD testing as it enables modules to be tested as discrete checks. Each test simulates the resultant data from real scenarios that might be encountered by the robot. For example, a test of the pathfinder would feed the pathfinder thread simulated line data and check if the tend angle returned from the pathfinder matches the expected result. Following a TDD paradigm enables each software module to be tested at startup without having to connect all of the hardware to the system. Modules can then be developed separately without having to consider the code required to interface with specific hardware devices, and once each module is incorporated into the overall system there is some assurance as to each module's functionality.

Software testing is also performed using the actual hardware. Since the software is modular and all the hardware uses standard (USB and Firewire) connections, each piece of hardware is tested remotely from the robot through direct connection to a development PC. When pieces of hardware are not connected to the computer, the software simply disables them, allowing for easier and more versatile testing. This ability is also useful when the robot is running, allowing for tests to be run with as few hardware pieces as possible. For example, the robot can be run with only the LIDAR or camera sending data to the pathfinder. These tests enable more efficient debugging of the integrated robot system.

Once implementation is complete, MAX, with completed (alpha level) software and vision hardware, is taken outside to a sample course constructed on the Missouri S&T campus. MAX is physically placed in a variety of obstacle scenarios involving vertical barriers and white lines painted on the ground. An opportunity is provided for the robot to navigate around the obstacles. The display of the robot's laptop computer provides continuous feedback on the obstacles being processed and the tend angle desired. The images provided in Fig. 12 demonstrate the vision and path finding of MAX.

To begin, MAX is placed before a white line painted on the grass (Fig. 12, top left). An image is captured from the video stream and corrected for distortion (Fig. 12, top middle) before the adaptive threshold and linear regression can be applied to generate lines representing obstacles (Fig. 12, top right). The pathfinder displays the tend angle, adjusted to miss the obstacle, to the user via a graphical user interface (GUI) on the display of MAX's onboard Apple MacBook (Fig. 12, bottom left). The tend angle (shown as a dark blue arrow extending upward and right from the origin) directs MAX to turn right. A large trash barrel is now placed before MAX to test the vertical obstacle avoidance (Fig. 12, bottom middle), and the tend angle adjusts further right to avoid the barrel and the remaining visible white lines (Fig. 12, bottom right).

During testing, several minor problems are found mainly dealing with environmental conditions. First, the camera vision system is found to be highly sensitive to shadows. Since shadowed areas are darker than areas of direct sunlight, the adaptive-threshold algorithm tends to consider the shade to direct sunlight difference as a line. This erroneous line information is then propagated to the pathfinder and the tend angle adjusts to avoid this boundary between shade and direct sunlight. This is found to be a large problem during testing on the practice course at Missouri S&T, it turns out to be a nonissue at the IGVC since there are no shadows for the robots to contend with.

A similar problem is found during midafternoon testing when the sun tends to be its brightest; the bright sun washes out some of the color difference between the white lines and the grass. With the differential reduced, the adaptive threshold does not always find the difference, and the line is not avoided. Adjusting the block size of the adaptive threshold algorithm helps to increase the algorithm's sensitivity and

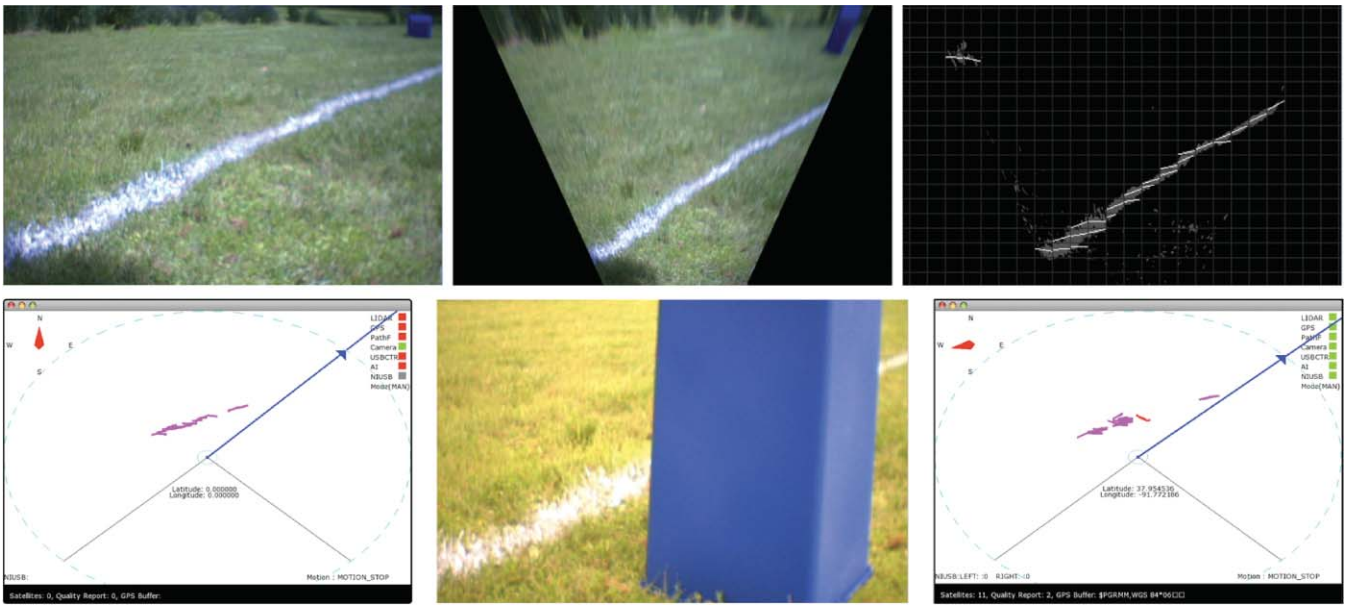


Fig. 12 Processed images and pathfinder output resulting from the placement of obstacles in the path of the robot during field testing on the Missouri S&T sample course. (Color online only.)

improves operation during periods of bright sunlight. This problem, however, cannot be completely corrected by the configuration changes and continues to cause problems for the vision system during the competition.

The final issue discovered stems from the hardware specifications. Since the LIDAR is not rated for outdoor use (its disturbance light level is 10,000 lux), it randomly stops responding during a washout.[30] To work around this, the LIDAR thread monitors the state of the LIDAR, and if the LIDAR turns off or stops responding, the robot pauses its movement for 5 s, restarts the LIDAR, waits for the LIDAR to recalibrate, and then, continues to move. Finding this problem during the testing at Missouri S&T was fortuitous as a washout occurred once during competition, and the robot was able to recover from the failure.

6 Discussion and Conclusions

An engineering design approach to intelligent ground vehicle development was presented. The function-based, engineering design process presented provided the foundation for the design of the IGVC robot, MAX. Functionality was considered through both the conceptual and physical design phases and not only provided the framework for module identification but also provided the framework for concept identification. Once in the physical design phases, the functional foundation was maintained by considering each module independently for assembly, testing, and final integration. Specific modules identified prior to component assignment led to a multithreaded software system distributing not only the vision tasks to independent threads but also all other processing and hardware access needs such as navigation, self-monitoring, way-point analysis, and motor control. This multithreaded design enabled processing tasks to run concurrently (i.e., new image information can be collected while current information is being analyzed to determine the optimal path for the robot). The modularity of the robot's hardware and software systems enabled each to be divided into subsystems that can be implemented as relatively simple, tractable pieces, and assembled into the complete robot system. It is this utilization of function and process that led to a design integrating the IGVC requirements into a cohesive yet modular system.

At the IGVC, the robot system performed well, and the vision system operated as intended. White lines and barrels generally were visualized and the ray-casting based pathfinder was able to project tend angles to avoid obstacles. In each competition run, the robot navigated the course consistently up to the switchback that is placed within the lane lines of the course. At the switchback, however, problems did arise. In each run, the robot enters the switchback at maximum velocity, and while the visual system is able to update and perceive the obstacle, the pathfinder, running on the prior data set, could not find a valid path through the switchback. This coupled with the robot operating at maximum velocity when entering the switchback and the robot's wide turning angle, resulted in a consistent error that terminated each competition run.

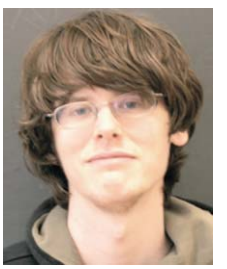
The issues identified at the switchback as well as throughout testing at the competition revealed a number of issues that should receive reconciliation in future versions of the vision system and path-planning algorithms. These reconciliations include (1) increased camera height to reduce the distortion of the image feed, (2) autodetection and status monitoring for all devices, (3) virtual course with virtual path planning such that information from the LIDAR and the camera are compiled into a virtual representation of the course so virtual paths can be analyzed ahead of the robot and remembered after the robot's departure, (4) distributed computing across multiple networked computers, (5) improved camera

filtering to remove noise, and (6) better control over the robot's speed such that it can be decreased when more time is required to determine appropriate paths, and conversely, it can be increased when less time is required.

References

1. B. K. P. Horn, *Robot Vision*, The MIT Press, Cambridge, MA.
2. H. H. González-Banos, and J.-C. Latombe, "Navigation strategies for exploring indoor environments," *Int. Journal of Robotics Research*, **21**(10–11), 829–848 (2002).
3. S. Thrun, et al. "Stanley: the robot that won the DARPA grand challenge," *J. Field Robot.* **23**(9), 661–692 (2006).
4. Stanford Racing Team, "*Stanford's robotic vehicle Junior*," Interim Report (2007); available from: http://www.darpa.mil/grandchallenge/TechPapers/Stanford.pdf.
5. C. Urmson,et al. "A robust approach to high-speed navigation for unrehearsed desert terrain," *J. Field Robot.* **23**(8), 467–508 (2006).
6. C. Urmson, et al. "Tartan racing: a multi-modal approach to the DARPA urban challenge," (2007); available from: http://www.darpa.mil/grandchallenge/TechPapers/Tartan_Racing.pdf.
7. I. Miller, S. Lupashin, N. Zych, P. Moran, B. Schimpf, A. Nathan, and E. Garcia, "Cornell University's 2005 DARPA grand challenge entry," *J. Field Robot.* **23**(8), 626–652 (2006).
8. M. Campbell, E. Garcia, D. Huttenlocher, I. Miller, P. Moran, A. Nathan, B. Schimpf, N. Zych, J. Catlin, F. Chelarescu, H. Fujishima, F.-R. Kline, S. Lupashin, M. Reitmann, A. Shapiro, and J. Wong, "Team Cornell: technical review of the DARPA urban challenge vehicle," (2007); available from: http://www.darpa.mil/grandchallenge/TechPapers/Team_Cornell.pdf.
9. B. M. Leedy, J. S. Putney, C. Bauman, S. Cacciola, J. M. Webster, and C. F. Reinholtz, "Virginia Tech's twin contenders: a comparative study of reactive and deliberative navigation," *J. Field Robot.* **23**(9), 709–727 (2006).
10. C. Reinholtz, et al. "DARPA urban challenge technical paper" (2007); available from: http://www.darpa.mil/grandchallenge/TechPapers/Victor_Tango.pdf.
11. DARPA, "Grand challenge 2005 team technical papers" (2005) [cited 2010 Oct. 8]; available from: http://www.darpa.mil/grandchallenge05/techpapers.html.
12. DARPA, "Teams: DARPA urban challenge" (2007) [cited 2010 Oct. 8]; available from: http://www.darpa.mil/grandchallenge/teamlist.asp.
13. Intelligent Ground Vehicle Competition, "The 16th Annual Intelligent Ground Vehicle Competition" (2008) [cited 2008 April]; available from: www.IGVC.org.
14. K. Otto, and K. Wood, *Product Design: Techniques in Reverse Engineering, Systematic Design, and New Product Development*, Prentice-Hall, New York. (2001).
15. R. Stone, K. Wood, and R. Crawford, "A heuristic method for identifying modules for product architectures," *Design Stud.* **21**(1), 5–31 (2000).
16. F. Zwicky, *Discovery, Invention, Research—Through the Morphological Approach*, The Macmillian Company Toronto (1969).
17. K. T. Ulrich and S. D. Eppinger, *Product Design and Development*, 3rd ed., McGraw-Hill/Irwin, Boston (2004).
18. D. G. Ullman, *The Mechanical Design Process*, 4th ed., McGraw-Hill, Boston 2010.
19. S. D. El Wakil, *Processes and Design for Manufacturing*, 2nd ed., Waveland Press, Prospect Heights, IL (2002).
20. C. Poli, *Design for Manufacturing: A Structured Approach*, Butterworth-Heinemann, Boston (2001).
21. D. Cutherell, "Product architecture," Chap. 16 in *The PDMA Handbook of New Product Development*, M. Rosenau Jr., Ed., Wiley, New York (1996).
22. G. Pahl, W. Beitz, J. Feldhusen, and K. H. Grote, *Engineering Design: A Systematic Approach*, 3rd ed., Springer Verlag, London (2007).
23. L. Miles, *Techniques of Value Analysis and Engineering*, McGraw-Hill, New York (1961).
24. C. L. Dym and P. Little, *Engineering Design: A Project-based Introduction*, Wiley, New York (2004).
25. N. Cross, *Engineering Design Methods: Strategies for Product Design*, John Wiley & Sons, Ltd, Chichester (2000).
26. R. L. Nagel, R. S. Hutcheson, R. B. Stone, and D. A. McAdams, "Process and event modelling for conceptual design" *J. Eng. Design*, (2009).
27. R. Stone, K. Wood, and R. Crawford, "Using quantitative functional models to develop product architectures," *Design Stud.* **21**(3), 239–260 (2000).
28. D. G. Ullman, *The Mechanical Design Process*, McGraw-Hill, New York (2002).
29. J. Hirtz, R. Stone, D. McAdams, S. Szykman, and K. Wood, "A functional basis for engineering design: reconciling and evolving previous efforts," *Res. Eng. Design* **13**(2), 65–82 (2002).
30. Hokuyo Automatic Co., L., *Range-Finder Type Laser Scanner URG-04LX Specifications* (2005): http://www.acroname.com/robotics/parts/R283-HOKUYO-LASER1s.pdf.
31. Apple Corporation, *iSight User's Guide* (2004): http://manuals.info.apple.com/en_US/iSightUserGuide.pdf.
32. Sewell, *Sewell SW-1301 Serial to USB Adaptor* (2007).
33. Intel Corporation, *OpenCV Reference Manual* (2000): available from: http://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf.
34. Apple Corporation, *QuickTime*, in *Developer Connection* (2009): http://developer.apple.com/quicktime/.
35. R. A. Goldstein, and R. Nagel, "3-D visual simulation," *Simulation* **16**(1), 25–31 (1971).
36. J. Hurley, "Ray tracing goes mainstream," *Intel Technol. J.*, **09**(02), 99–108 (2005).
37. P. Stone, P. Beeson, T. Mericli, and R. Madigan, "Austin robot technology — DARPA urban challenge technical report" (2007); available from: http://www.darpa.mil/grandchallenge/TechPapers/Austin_Robot_Tech.pdf.
38. A. Lamothe, J. Ratcliff, and D. Tyler, *Tricks of the Game Programming Gurus*, Sams Publishing, Indianapolis IN (1994).
39. S. Lee, J. Seo, G. J. Kim, and C.-M. Park, "Evaluation of pointing techniques for ray casting selection in virtual environments," in *Proc. 3rd Int. Conf. on Virtual Reality and Its Application in Industry*, Z. Pan and J. Shi, Eds., SPIE **4758**, 38–44 Hangzhou, China (2003).
40. M.-E. Bellemare, P. Haigron, A. Lucas, and J.-L. Coatrieux, "Depth map based scene analysis for active navigation," *SPIE Conf. on Physiology and Function from Multidimensioal images* SPIE **3660**, San Diego, 202–213 (1999).
41. P. Pfaff, R. Kümmerle, D. Joho, C. Stachniss, R. Triebel, and W. Burgard, "Navigation in combined outdoor and indoor environments using multi-level surface maps," in *Proc. Workshop on Safe Navigation in Open and Dynamic Environments at the IEEE Int. Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA (2007).
42. R. Hadsell, A. Erkan, P. Sermanet, J. Ben, K. Kavukcuoglu, U. Muller, and Y. LeCun, "A multi-range vision strategy for autonomous offroad navigation," in *Proc. IASTED Int. Conf. on Robotics and Applications*, International Association of science and Technology for Development, Würzburg, Germany (2007).
43. N. Llopis and C. Nicholson, *UnitTest++*, SourceForge http://unittest-cpp.sourceforge.net/.
44. K. Beck, *Test-Driven Development: By Example*, Addison-Wesley, Boston (2003).

Robert L. Nagel is currently an assistant professor with the School of Engineering at James Madison University. Dr. Nagel received his PhD degree from Oregon State University, his MS degree from the University of Missouri–Rolla (known now as Missouri University of Science and Technology), and his BS degree from Tri-State University (known now as Trine University). He has been a member of the Design Engineering Lab, a researcher at General Motors in the Vehicle Development Research Lab, and has worked on contract projects with both the United States Army and United States Air Force. His research interests include understanding customer needs, functional and process modeling, failure analysis, design for sustainability, design of mechatronic systems, and systems integration.



Kenneth L. Perry is currently the lead developer for Interdisciplinary Design Collaborative, LLC, in Rolla, Missouri. Perry is working toward his MS degree in the area of software compilers with the Computer Science Department at the Missouri University of Science and Technology, where he received his BS degree. Perry has been an undergraduate researcher with the Design Engineering Lab and has worked on several software development projects such as FunctionCAD (a functional modeling design tool), and the Intelligent Ground Vehicle Competition (IGVC) robot MAX.

Robert B. Stone is a professor with the School of Mechanical, Industrial and Manufacturing Engineering at Oregon State University. Dr. Stone's research interests include design theories and methodologies, specifically product architectures, functional representations, automated conceptual design techniques, and renewable energy systems design. He leads the Design Engineering Lab. He has authored chapters on product architecture and reverse engineering techniques in product design texts. He is a member of the ASME Design Theory and Methodology Committee and a past chair of its signature international conference. Prior to initiating his graduate work, Dr. Stone was a Space Shuttle flight controller for the Guidance, Navigation and Control Section with the Missions Operation Directorate of the National Aeronautics and Space Administration (NASA) Johnson Space Center. Stone received his PhD degree in mechanical engineering from the University of Texas at Austin.

Daniel A. McAdams is currently an associate professor of mechanical engineering with Texas A&M University. Dr. McAdams received his BS and PhD degrees from the University of Texas at Austin and his MS degree from the California Institute of Technology. Dr. McAdams has been an associate professor and the associate chair of graduate affairs for mechanical engineering at Missouri University of Science and Technology. His research interests are design theory and methodology with specific focus on functional modeling, innovation in concept synthesis, biomimetic design methods, model construction and selection for design, and failure avoidance as applied to product design.