Kratos: Princeton University's entry in the 2008 Intelligent Ground Vehicle Competition

Christopher A. Baldassano, Gordon H. Franken, Jonathan R. Mayer, Andrew M. Saxe, and Derrick D. Yu

Princeton University School of Engineering and Applied Science, Princeton, NJ, USA

ABSTRACT

In this paper we present Kratos, an autonomous ground robot capable of static obstacle field navigation and lane following. A sole color stereo camera provides all environmental data. We detect obstacles by generating a 3D point cloud and then searching for nearby points of differing heights, and represent the results as a cost map of the environment. For lane detection we merge the output of a custom set of filters and iterate the RANSAC algorithm to fit parabolas to lane markings. Kratos' state estimation is built on a square root central difference Kalman filter, incorporating input from wheel odometry, a digital compass, and a GPS receiver. A 2D A* search plans the straightest optimal path between Kratos' position and a target waypoint, taking vehicle geometry into account. A novel C++ wrapper for Carnegie Mellon's IPC framework provides flexible communication between all services. Testing showed that obstacle detection and path planning were highly effective at generating safe paths through complicated obstacle fields, but that Kratos tended to brush obstacles due to the proportional law control algorithm cutting turns. In addition, the lane detection algorithm made significant errors when only a short stretch of a lane line was visible or when lighting conditions changed. Kratos ultimately earned first place in the Design category of the Intelligent Ground Vehicle Competition, and third place overall.

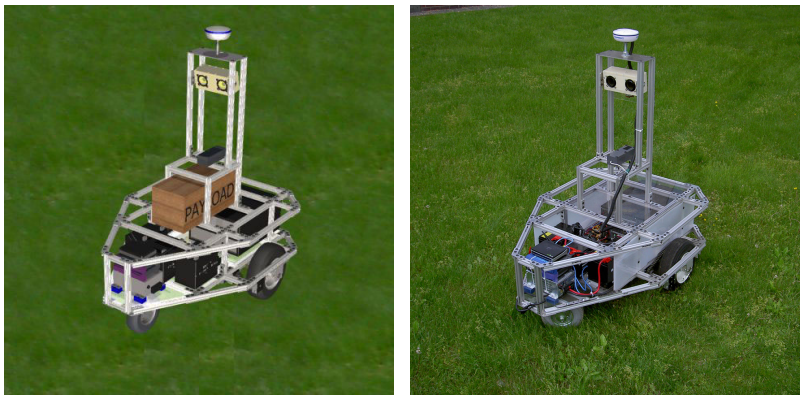
Keywords: robotics, IGVC, stereo vision, lane detection, IPC, Kalman Filter, cost map, path planning

1. TEAM OVERVIEW

The Princeton Autonomous Vehicle Engineering (PAVE) IGVC team consists of nine undergraduate students at Princeton University, all of whom were members of Princeton's all-undergraduate semi-finalist DARPA Urban Challenge team and bring hands-on experience with computer vision and autonomous navigation. Our entry into the 2008 IGVC, Kratos, builds upon the systems in our prior robots, Prospect Eleven[1] and Prospect Twelve.[2]

2. DESIGN PROCESS

The short development time for this project (February through May, 2008) necessitated an efficient design process. We analyzed the requirements and restrictions specified in the rules for the 2008 IGVC[3] and distilled a set of concrete, solvable design objectives. Before construction began the entire robot was modeled with the *Autodesk Inventor 2008* CAD software to ensure the design's feasibility (Figure 1).



(a) CAD Model **(b)** Real-World

Figure 1: Visualizations of Kratos

2.1 Mechanical Design

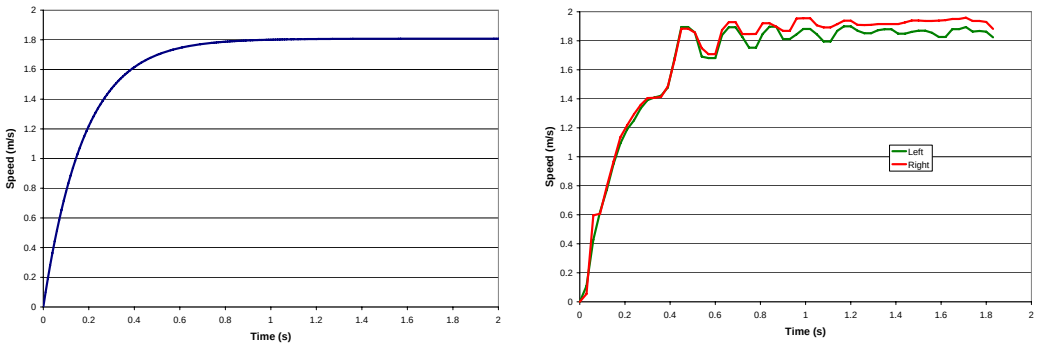
Based on competition rules, Kratos had to be capable of climbing a 15% grade at 5 mph. A comprehensive power analysis, factoring in real-world conditions, informed our drive motor selection. We assumed a 90% gearbox efficiency in each of three reduction stages. Based on experimental results we used .2 and .1 as coefficients of rolling friction over grass and wood respectively. We additionally adjusted the motor power curves for increased winding resistance. Applying our analysis to the IGVC requirements we identified three distinct cases of robot motion, as outlined in Table 1.

Case	Net Force	Power Required	Motor Power	Power per side
1: Full speed forward on grass	196.2	438.5	601.5	300.8
2: Turning in place on grass	130.8	292.3	401.0	401.1
3: Driving up a 10° incline	268.4	600.0	823.0	411.5

Table 1: Power requirements for robot motion. Speed in (m/s), force in (N), power in (W). Cases 1 and 3 assume power is split evenly between the each side of the robot, whereas case 2 assumes only one motor is operating.

The drive-train configuration consisted of two CIM motors on each side of the robot. This configuration produces a maximum of 440W per side, which is enough power to meet all of the cases outlined in Table 1, though 80A cicruit breakers slightly reduced the available power. To determine final gearbox ratios and wheel diameters, we performed a full analysis of the motor power curve. Key values for the 2-motor setup were taken from published data,[4] additional motor parameters were calculated, and a numerical simulation of the motor power curve was conducted.[5]

Kratos operated with 12.4" diameter wheels and a 40:1 overall gear ratio, consisting of a 12:1 planetary gearbox and a 10:3 chain drive. This configuration provides ample power and quick acceleration in all of the design cases, with substantial safety margins for known failure points such as torque limitations, chain strength and current limits. The only drawback of this setup is that the predicted maximum speed in all three cases was reduced from 5 mph to around 4 mph. However, we considered it better to relax our original design requirements than operate with reduced safety margins. A numerical simulation was used to predict our top speed and acceleration, graphed in Figure 2a. These results are very closely aligned with actual speed data taken from the robot, shown in Figure 2b. These graphs demonstrate the accuracy of our pre-fabrication simulation.



(a) Numerical Simulation **(b)** Data From Test Run

Figure 2: Predicted and Actual Robot Speeds

2.2 Fabrication and Assembly

Using the *Autodesk Inventor 2008* CAD software we converted individual parts either to 2D engineering drawings for manual fabrication or CNC programs for automated manufacturing. We selected the 80/20© framing system over aluminum or steel bars to construct Kratos' chassis; these custom-extruded aluminum bars are stronger than square-tube extruded aluminum of the same size and allow for rapid assembly and disassembly. The primary chassis bars are held together by custom machined aluminum plates, while 2 inch aluminum L-brackets provide corner reinforcement in areas of high stress. A platform of 1/4" polycarbonate rests on the bottom

of the chassis and supports Kratos' internal components. Polyethylene sheets line the perimeter of the robot, protecting sensitive internal equipment from external hazards.

2.3 Electrical Design

The IGVC requires at most a running time of 6 minutes. Given the value of extended testing, however, we aimed instead to have enough on-board power for one hour of continuous operation with at least three hours on standby. An analysis of the power requirements for all electrical components showed a required 983W at peak and 236W during idle; at 12V, this translates to 82A and 20A, respectively. We consequently selected three deep-cycle batteries with a rating of 33A-hr, providing a total of 99A-hr at full charge. Kratos is therefore able to last under its own power for over one hour of continuous operation and remain self-powered at idle for over 5 hours. An on-board 12V battery charger allows Kratos unlimited operation in any location with a 120V AC power source.

2.4 Electronics and Computing

A Labjack UE9 data acquisition unit allows for digital and analog input/output between our electronics and computers via Ethernet. The UE9 generates PWM signals that command Kratos' Victor884 motor controllers and performs 4x quadrature counting to track the position of the US Digital HB6M rotary incremental encoders on each axle. The UE9's digital I/O controls Kratos' warning alarm and detects the hardware motor-cutoff Emergency Stop status.

Given the computational intensity of the algorithms we employ (Section 3), Kratos is equipped with two on-board computers, one devoted to vision processing while the other is responsible for navigation and low-level control. Both computers are configured identically, with 2.0GHz Intel Core2Duo processors, 2GB of RAM, and 200GB hard drives, powered by 200W DC power supplies. The cases are mounted together and secured to the chassis with shock-isolating feet. A gigabit Ethernet switch connects the two computers and the Labjack UE9.

2.5 Sensors

The sole environmental sensor on Kratos is a Point Grey Bumblebee2 color stereo camera, avoiding the challenges posed by fusing separate obstacle and lane detection sensors. The Bumblebee2 has a horizontal FOV of 70° and an effective range of 18 meters. We accounted for the Bumblebee2's minimum detection distance of 1m by mounting it on Kratos' sensor tower (with a 12° downward pitch) such that its undetectable zone is contained entirely within the robot's physical envelope.

Kratos is equipped with a Hemisphere A100 GPS receiver, located at the top of the sensor tower. Using WAAS differential corrections it provides global position data to 0.6 m accuracy, as well as heading and velocity, at 10Hz via an RS232 serial connection. A Honeywell HMR3000 digital compass outputs magnetic bearing with 1° accuracy at 20Hz, also over RS232. Finally, two US Digital HB6M rotary incremental encoders, one mounted on each wheel axle, provide local motion data accurate to .1 mm. The GPS receiver, camera and compass are all vertically aligned within the sensor tower, which is positioned above the midpoint between the two wheels. All of Kratos' sensors therefore share the same reference location, eliminating the need to perform frame transformations in software.

3. SOFTWARE

3.1 Platform

Kratos' two computers run the Windows Server operating system for ease of use and maximal hardware compatibility. All software is written in platform-independent C++ using the Visual Studio IDE. The Newmat[6] and wykobi[7] libraries provide linear algebra and geometry functionality respectively, and the Qt library and Designer allow for rapid multi-platform GUI development. Each software component runs as an independent program connected to a central server using Carnegie Mellon's Inter-Process Communication (IPC) platform,[8] which provides an API for message publishing and subscription, serialization, and timing.

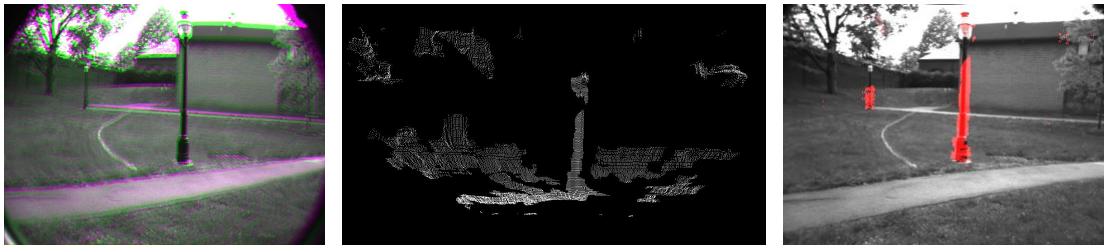
We developed a novel C++ wrapper for IPC to speed development, minimize errors, and ensure all modules share identical message definitions. Most robotics frameworks are either oriented around named messages (i.e. IPC) or named services (i.e. Microsoft Robotics Developer Studio[9]). Our wrapper, IPC++, object orients the framework around the messages themselves and localizes it to a dynamically linked library, allowing changes to message definitions without recompilation. From a programming standpoint IPC++ improves upon IPC by abstracting away message identifiers and memory management. We intend to publicly release IPC++ in 2009.

3.2 Vision

3.2.1 Obstacle Detection

Point Grey's stereo vision library initially computes a disparity map by matching similar pixels between the two simultaneously captured stereo images. Leveraging knowledge of the camera's intrinsic characteristics as well as the inverse relationship between disparity and distance, Point Grey's library then converts the disparity map into a depth map of each matched point in the image. We finally apply a simplified version of the obstacle detection algorithm proposed by Manduchi et al.,[10] which searches for pairs of points that are roughly vertical (Algorithm 1). The results of our stereo obstacle detection are shown in Figure 3.

Stereo obstacle detection's primary shortcoming is the imperfection inherent to disparity maps; pixel matching is difficult in areas with low texture, poor lighting, or uneven contrast. To improve the quality of its disparity map generation Point Grey's library imposes several validation checks on points in the disparity map, including rejecting pixels in low texture areas or that match to a large number of other pixels. Another significant problem with stereo vision is its inherent maximum and minimum sensing distances; disparity is too small to detect for distant objects, and close objects are not in the field of view of both cameras. In practice we found the 18 meter effective detection range provided by the Bumblebee2 more than adequate for the IGVC, and as discussed earlier (Section 2.5) we were able to design around the minimum detection range.



(a) Camera Image **(b)** 3-D Map **(c)** Obstacle Points

Figure 3: Stages of Obstacle Point Detection

Input: Depth Map, MaxDepthDiff = .3 meters, MinHeightDiff = .3 meters, MaxHeightDiff = .5 meters, MinAngle = 80°, FocalLength
Output: Obstacle Points
foreach *Point P in Depth Map* **do**
  SearchHeight = FocalLength × MaxHeightDiff / Depth(P);
  SearchBase = 2 × SearchHeight / tan(MinAngle);
  T = the inverted triangle of pixels above P, with height SearchHeight and base SearchBase;
  foreach *Point Q in T* **do**
    HeightDiff = |Height(Q) - Height(P)|;
    DepthDiff = |Depth(Q) - Depth(P)|;
    XDiff = |LateralPosition(Q) - LateralPosition(P)|;
    if *DepthDiff < MaxDepthDiff and MinHeightDiff < HeightDiff < MaxHeightDepth and HeightDiff / XDiff > tan(MinAngle)* **then**
      FlagAsObstacle(P);
      FlagAsObstacle(Q);
    end
  end
end

Algorithm 1: The depth map to obstacle points algorithm.

3.2.2 Lane Detection

Our lane detection algorithm, an extended version of the system used by Princeton in the DARPA Urban Challenge, functions in three phases. First, a series of filters is applied to each pixel in the image to determine

whether it might fall on a lane line. Two color filters respond to pixels that correspond to the yellow and white colors of lane markings, while a rectangular pulse width filter responds to edges of the correct width for a lane marking (adjusted by vertical location in the image). Finally, an obstacle filter raises the score of pixels that do not fall on an obstacle. The image of obstacle points is blurred and then re-thresholded to binary values to better block pieces of the obstacles from being marked as lanes. This type of filter was made possible by using the same camera for both obstacle and lane detection, giving us a common image space for processing.

The results of all three filters are fused into a "heat map" indicating the likelihood that each pixel falls on a lane marking. We then apply the RANSAC algorithm[11] to find the parabolic fit that passes near the most lane pixels. Because a parabola does not perfectly describe the geometry of a lane, a greedy pixel-by-pixel search is performed from the furthest pixel in the lane marking. Figure 4 demonstrates the stages of the lane detection algorithm.

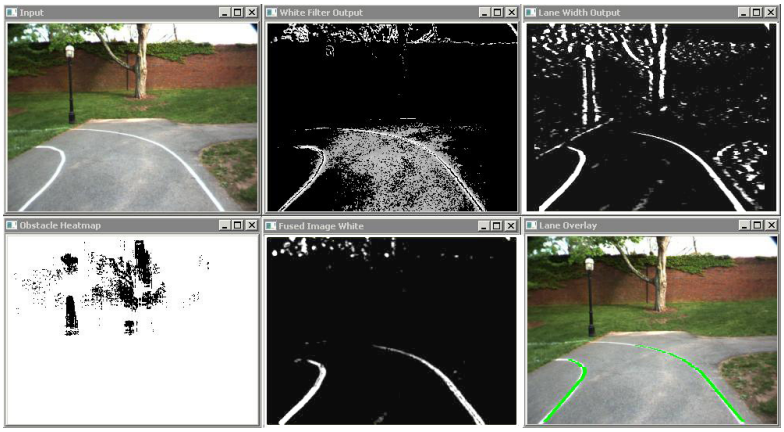


Figure 4: Stages of Lane Detection

3.2.3 Vision Results

Our stereo obstacle detection scheme is able to reliably detect obstacles up to 18 meters (60 feet) away, with the number of obstacle points detected increasing exponentially as the distance decreases (Figure 5). After testing, we determined that the depth calculations performed by Point Grey's libraries showed a slight bias at long distance. We accounted for the bias by empirically fitting a correction function, yielding

$$d_a = \frac{2}{100000}(d_m)^4 + d_m,$$

where d_m is the measured depth and d_a is the actual depth.

Our lane detection algorithm was tested extensively during the DARPA Urban Challenge. We evaluated algorithmic performance on 200 images along a highway, with one solid lane marking and several dashed lane markings. Between two and three lanes are visible in each image. In the 200 images, we find that one lane is found in 7% of images, two lanes are found in 88% of images and three are found in 5% of images. In all 200 images, only 5 false positives were observed.

3.3 State Estimation

We use a square root central difference Kalman filter (SRCDKF) to fuse GPS, wheel encoder, and compass data into a near-optimal estimate of the vehicle's position.[12] The central difference Kalman filter (CDKF) is a type of sigma point filter like the unscented Kalman filter (UKF) which uses a deterministic set of points to represent a multivariate Gaussian distribution over states and measurements. When propagated through nonlinear process and measurement models, these points represent the resulting posterior Gaussian random variable accurately to the second order Taylor series expansion of the nonlinear system. In contrast, the popular extended Kalman filter (EKF) is only accurate up to the first order.

To obtain numerical stability and efficiency, the square root formulation of the CDKF manipulates the lower triangular Cholesky factor of the error covariance matrices. In this way the represented covariance matrices are

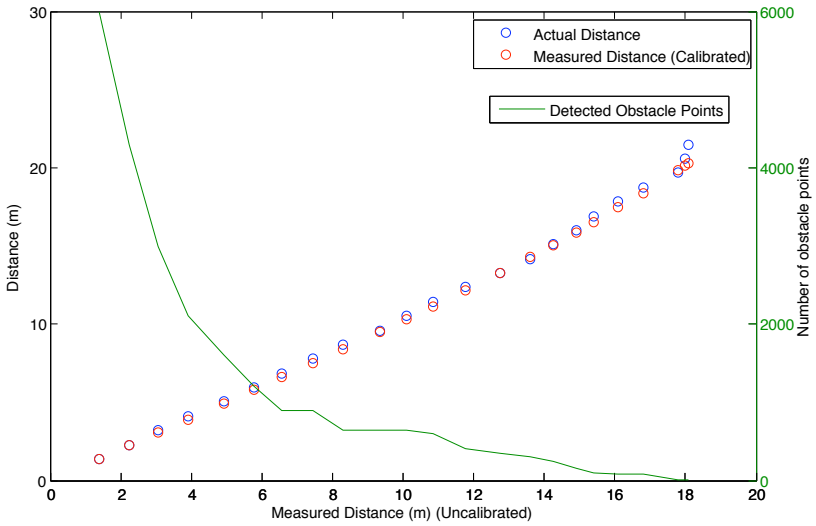


Figure 5: Stereo Vision Results

necessarily symmetric and positive definite. The computational complexity is $O(L^3)$ where L is the length of the state vector. We represent the state of the robot with a six variable state vector,

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \\ \delta \\ v_r \\ v_l \end{pmatrix}, \tag{1}$$

where x is the vehicle x coordinate in meters in a Cartesian local frame, y is the vehicle y coordinate in meters, $\theta \in [0, 2\pi)$ is heading, $\delta \in [0, 2\pi)$ is a bias between compass heading and GPS heading, v_r is right wheel tread speed in m/s, and v_l is left wheel tread speed in m/s.

3.3.1 Process Model

We use a discrete time model of a differential drive robot given by Larsen et al.[13]

$$\mathbf{x}(t+dt) = \begin{pmatrix} x(t) + \left(\frac{v_r(t)+v_l(t)}{2}\right) \sin\left(\theta(t) + \frac{v_l(t)-v_r(t)}{2b}\right) dt \\ y(t) + \left(\frac{v_r(t)+v_l(t)}{2}\right) \cos\left(\theta(t) + \frac{v_l(t)-v_r(t)}{2b}\right) dt \\ \theta(t) + \left(\frac{v_l(t)-v_r(t)}{b}\right) dt \\ \delta(t) + \eta_1 dt \\ v_r(t) + \eta_2 dt \\ v_l(t) + \eta_3 dt \end{pmatrix},$$

where b is the track width of the robot, $\eta = [\eta_1 \quad \eta_2 \quad \eta_3]^T$ is a Gaussian random variable and dt is the integration time step of the system (approximately 1/20 s in our implementation).

3.3.2 Measurement Models

The filter incorporates sensor measurements using nonlinear measurement models that predict the value of a given sensor measurement from the current estimate of the state variables. GPS position measurements are predicted to be

$$\mathbf{y}_{\text{GPS Position}}(t) = \begin{pmatrix} x(t) + \alpha_1 \\ y(t) + \alpha_2 \end{pmatrix},$$

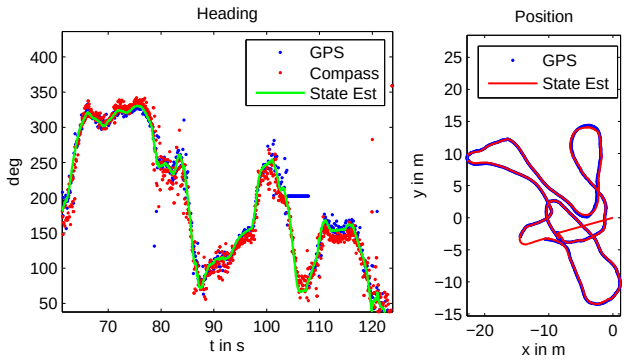


Figure 6: Sample input measurements and filter output

where $\alpha = [\alpha_1 \quad \alpha_2]^T$ is a Gaussian random variable corrupting the x and y GPS measurements. The GPS heading measurement is predicted to be

$$y_{\text{GPS Heading}}(t) = (\theta(t) - \pi u(-v_r(t) - v_l(t)) + \zeta) \,\%2\pi,$$

where $u(x)$ is the Heaviside step function, ζ is Gaussian noise corrupting the GPS heading measurement, and $\%$ is the modulo operator. When the robot is traveling in reverse, the GPS measured direction is opposite the robot heading. The $\pi u(v_r(t) + v_l(t))$ term incorporates this effect by subtracting π from the predicted heading value when the robot is traveling backwards. The GPS speed measurement model is

$$y_{\text{GPS Speed}}(t) = \left| \frac{v_r(t) + v_l(t)}{2} + \sigma \right|$$

where σ is Gaussian noise in the GPS speed measurement. The absolute value reflects the fact that the GPS speed is always a positive reading. Measurements of the vehicle's wheel speed are provided by encoders, which use the following measurement model,

$$\mathbf{y}_{\text{Wheel Encoders}}(t) = \begin{pmatrix} v_r(t) + \xi_1 \\ v_l(t) + \xi_2 \end{pmatrix},$$

where $\xi = [\xi_1 \quad \xi_2]^T$ is a Gaussian random variable. Finally, we take the compass measurement model to be

$$y_{\text{Compass Heading}}(t) = (\theta(t) + \delta(t) + \beta) \,\%2\pi,$$

where β is Gaussian noise and $\delta(t)$ is the estimated bias between the compass heading and GPS heading. This bias term allows the filter to automatically correct for magnetic variation by learning a correction factor based on GPS heading.

3.3.3 Testing and Results

The SRCDKF was implemented first in Matlab and checked against an implementation of a linear Kalman filter before being transcribed into C++. The C++ implementation was verified against the Matlab code and tested in a simulator of the process and measurement dynamics before being deployed on the robot. Figure 6 shows sample heading and position estimates from the filter.

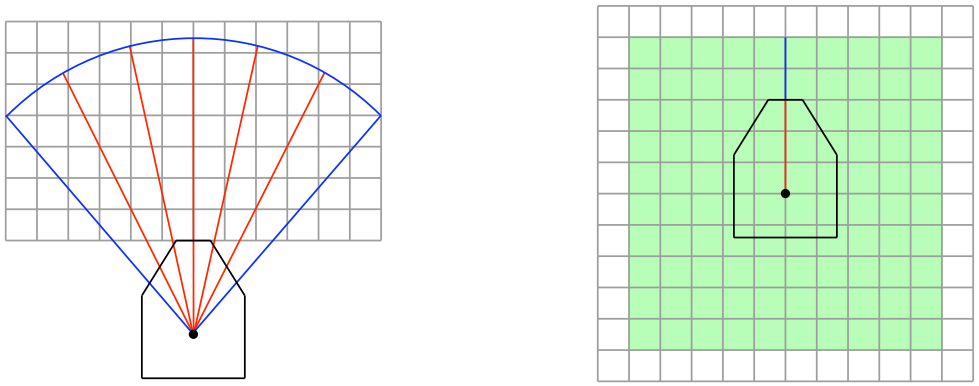
3.4 Navigation

Kratos' navigation scheme follows a cost map approach: the environment is gridded into square cells, which each have an associated numerical cost. With this environmental representation, path planning can be reduced to the well-understood graph search problem.

3.4.1 Cost Map Generation

Stereo points are stored in a dedicated cost map representing the camera's field of view. Each cell in the stereo cost map is assigned a numerical cost according to

$$c(p) = \max(k_{\text{clear}}, \min(1, n(p))k_{\text{first}} + \max(0, n(p) - 1)k_{\text{additional}}), \tag{2}$$



(a) Stereo cost map divided into segments

(b) Kratos' "footprint." The red segment indicates the maximum side length, the blue segment represents the buffer, and the shaded green area represents the entire footprint

Figure 7

where $c(p)$ is the cost associated with the cell containing the point p, $n(p)$ is the number of obstacle or lane points in the cell, k_{clear} is the cost associated with a clear cost cell, k_{first} is the cost associated with the first stereo point in a cell, and $k_{\text{additional}}$ is the added cost for each additional stereo point.

The camera's field of view is divided into s discrete angular regions (Figure 7a). Clear cells closer than the nearest non-clear cell in each region as well as all occupied cells are transformed into global coordinates according to the state estimation data associated with the stereo capture and averaged into the global cost map according to a set learning rate α. By initializing the global cost map's cells with a cost k_{starting} greater than k_{clear}, open cells' costs are lowered as they are viewed to be clear. Figure 9a shows a cost map generated from stereo vision data with our algorithm.

Because the robot's size is greater than a cost cell's, the cost of a cell does not represent the total cost of the robot being located at that cell. A second global cost map stores the sum of the cost at each cell within the robot's "footprint," a square with side length equaling the robot's longest protrusion from the reference point plus a buffer (Figure 7b).

3.4.2 Waypoint Selection

When GPS waypoints are provided, they are presorted to give the shortest overall path by enumerating all possible permutations. When lane following, the target waypoint is inferred based on the detected lanes. The detected lane lines are extended backwards to the position of the robot in order to determine which lanes are on the left and right sides. If both lanes are available, the farthest point on the shorter lane is averaged with the corresponding point on the other lane to place a waypoint in the center of the lane; if only one lane is detected the waypoint is placed a constant distance to the left or right of the farthest point detected.

3.4.3 Path Planning

Path planning is accomplished via a 2D A* graph search from the robot's current position to the destination waypoint. Search nodes correspond to cells in the "footprint" cost map, expanded by allowing transitions to the eight neighboring cells - a tradeoff of an increase in branching factor for a shallower solution. Node cost $f(n)$ is determined by

$$f(n) = g(n) + h(n) \tag{3}$$

according to the standard A* representation, where $g(n)$ is node depth and $h(n)$ is the heuristic value.[14] Node depth is assigned according to

$$g(n) = c(n) + \sqrt{(x_n - x_{n'})^2 + (y_n - y_{n'})^2} + g(n'), \tag{4}$$

(a) Two unfavorable paths **(b)** A preferable path

Figure 8: Equally optimal paths with the first heuristic (endpoints are green)

where $c(n)$ is the cost of node n in the "footprint" cost map and n' is the node expanded to yield n. The overall path, therefore, minimizes its total cost

$$\text{path cost} = \text{path length} + \sum_{n=start}^{goal} c(n). \tag{5}$$

The primary heuristic $h(n)$ is the length of the shortest cell-wise path between n and the goal assuming all cells have not been seen, developed according to:

$$\text{straight}(n) = \max(|x_n - x_{start}|, |y_n - y_{start}|) - \min(|x_n - x_{start}|, |y_n - y_{start}|), \tag{6}$$

$$\text{diagonal}(n) = \sqrt{2}\min(|x_n - x_{start}|, |y_n - y_{start}|), \text{and} \tag{7}$$

$$h(n) = (cellsize + k_{starting})(\text{straight}(n) + \text{diagonal}(n)), \tag{8}$$

where x and y are indices in the cost map and $cellsize$ is the length of a cell side. Though h is not an admissible or consistent heuristic because cell costs along the path may have decreased below $k_{starting}$ (if $k_{starting} > k_{empty}$), in practice we found using an expected cost value below $k_{starting}$ resulted in too expansive a search to be performed in real time. The resulting path is therefore not guaranteed to be optimal in preferring seen to unseen cells, but will avoid obstacles.

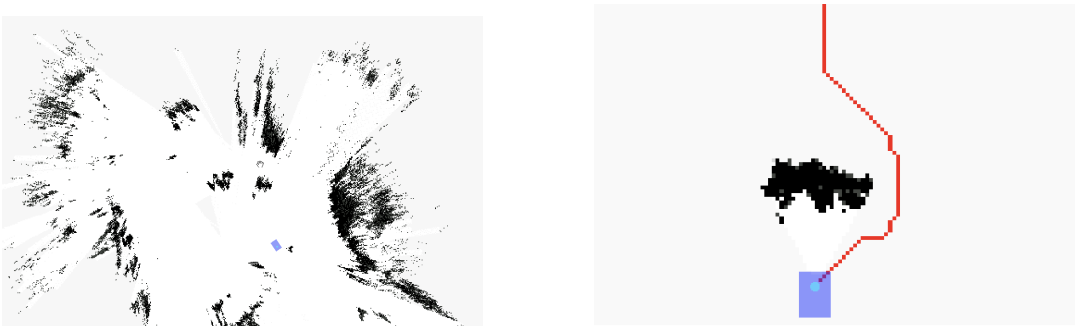
Though the heuristic above proved adept at avoiding obstacles and efficiently computing paths, it makes no preference in path structure; so long as two paths have the same total cost according to Equation 5, they are considered of equal optimality. This feature is problematic when implementing cell-wise paths in the real world, where they are naturally interpolated. Figure 8a demonstrates a trivial example, where two paths in an obstacle-free environment are viewed as equally optimal, though upon interpolation they clearly are not; Figure 8b shows the most desirable path in this situation, which interpolates roughly to a straight line. This path selection issue is more pernicious than mere detours, however; if paths alternate between various solutions (i.e. between the two paths in Figure 8a), a robot in the real world will behave erratically. In the worst case, if the paths alternate with an obstacle between them, the robot may never converge to any of the paths and collide with the obstacle.

We solved this path arbitration problem by adding a second "path straightness" heuristic that breaks node cost ties. The second heuristic $h'(n)$ measures how close the node is to the straight-line path between the start and goal cells according to

$$h'(n) = \|(\vec{n} - \vec{start}) \times (\vec{goal} - \vec{start})\|, \tag{9}$$

where two components of the vectors are x and y from above and the third is 0. Applying this second heuristic guarantees path arbitration will be settled in favor of straighter paths, like the one in Figure 8b.

In order to further minimize computation we implemented the graph search on a binary heap with stored node indices, improving the update key operation from $O(n)$ to $O(\log n)$ time. While this optimization substantially increased memory consumption, it resulted in a significant performance improvement. When implemented on its final form on Kratos, the planner operated at roughly 12Hz in an obstacle-free environment and 5Hz in a cluttered environment. Figure 9b shows a real-world trajectory generated by our algorithm.



(a) A cost map generated from stereo vision data **(b)** A path planned through a cost map

Figure 9: Cost map creation and trajectory generation results

(a) Proportional navigation law **(b)** Cross track error law

Figure 10: Path tracking controller geometries

3.5 Path Tracking

The node centroids on the path found by the path planning module are connected with line segments to form a continuous desired path. We can describe this curve by its arc length parametrization $\mathbf{r}(s) \in \Re^2$ where $s \in [0, L]$ and L is the total length of the path. The path tracking module takes this desired path $\mathbf{r}(s)$, the position of the robot $\mathbf{p}(t)$, and the velocity vector of the robot $\mathbf{v}(t)$ as input and computes desired left and right wheel speeds $v_l(t)$ and $v_r(t)$ respectively which cause the robot to track the path. We implemented and evaluated two path tracking control systems.

The first system is a proportional navigation law based on the angle between the robot's current heading and the heading to a look ahead point on the path (Fig. 10a). In this method we first find the point $\mathbf{r}(n^*)$ on the path nearest to the point $\mathbf{p}(t)$ between the robot's wheels, where

$$n^* = \text{argmin}_n \|\mathbf{r}(n) - \mathbf{p}(t)\|.$$

Then we find the point $\mathbf{r}(n^*+d(t))$ a distance $d(t) \propto ||\mathbf{v}(t)||$ ahead along the path. Scaling the look ahead distance with speed improves stability at the cost of worse tracking. Finally, we set the desired rotational velocity $\Omega(t)$ of the robot proportional to $h(t)$, the signed angle between the vector from the robot to the look ahead point, $\mathbf{r}(n^* + d(t)) - \mathbf{p}(t)$, and the robot's current velocity vector $\mathbf{v}(t)$.

The second system is a cross-track error controller proposed by Hoffman et al. for carlike robots.[15] The cross-track controller minimizes the crosstrack error $e(t)$, defined as the signed distance from the closest point on the path $\mathbf{r}(n^*)$ to the point $\mathbf{p}(t)$ between the robot's wheels,

$$|e(t)| = ||\mathbf{r}(n^*) - \mathbf{p}(t)||.$$

The crosstrack error metric in a point model of the robot evolves according to

$$\dot{e}(t) = ||\mathbf{v}(t)|| \sin(\psi(t)),$$

where $||\mathbf{v}(t)||$ is the robot speed and $\psi(t)$ is its heading relative to the trajectory, that is, the angle between $\dot{\mathbf{r}}(n^*)$ and $\mathbf{v}(t)$ (see Fig. 10b).[15] The control law is nonlinear and given by

$$\Omega(t) \propto \psi(t) + \arctan \frac{k_e e(t)}{||\mathbf{v}(t)||},$$

which will send the cross track error to zero if the proportionality constant is large.[15]

Although these two schemes may appear quite different, for straight paths the angle to the look ahead point is $\psi(t)+\arctan \frac{e(t)}{d(t)}$, so the controllers are identical if the same proportionality constant is used and $d(t) = \frac{1}{k_e}||\mathbf{v}(t)||$. For curved paths, the proportional navigation law will tend to cut corners to the inside due to the look ahead point rounding the corner first, while the cross track law will tend to cut corners to the outside. A compromise between these extremes would be a fruitful improvement for the future. For the competition we settled on the proportional navigation law approach.

The forward translational velocity $\nu(t)$ is set proportional to the local path curvature to slow the vehicle in corners. Left and right wheel speeds are determined from the desired rotational velocity $\Omega(t)$ and translational velocity $\nu(t)$ according to

$$\begin{aligned} v_l(t) &= \nu(t) + \frac{b\Omega(t)}{2} \\ v_r(t) &= \nu(t) - \frac{b\Omega(t)}{2} \end{aligned}$$

where b is the track width of the robot. These desired wheel speeds are attained using a proportional-integral controller for each wheel.

4. RESULTS

4.1 Navigation Challenge

The IGVC Navigation Challenge tasked entrants with navigating a static obstacle field, including a course-wide fence, to achieve waypoints in an arbitrary order. Kratos' first and third runs successfully navigated the fence and the majority of the waypoints. Both trials ended when Kratos brushed a plastic construction fence. Based on recorded data we attribute these failures to the proportional navigation law path tracking scheme, which was known to cut turns; our stereo vision system accurately detected the fence and the trajectory generation scheme plotted paths around it.

Kratos' second run proved the strength of its generic trajectory generation algorithm at navigating an unstructured environment. The course-wide fence inadvertently ended short of the course boundary, providing a second opening for passage. Kratos was able to successfully identify and navigate this gap without any prior knowledge of the course structure.

4.2 Autonomous Challenge

The Autonomous Challenge consisted of a figure-eight track of parallel white lines dotted with obstacles such as traffic barrels and trash cans, and included two wooden ramps. One of the most challenging sections of the run, located at the outset of the course, was a three-layer chicane that required robots to squeeze all the way to the right of the lane, all to way to the left of the lane, and then all the way back to the right of the lane to avoid colliding with traffic barrels. On its best trial Kratos successfully navigated the chicane and then continued halfway around the course before failing to see a lane line on a tight turn and exiting the course. On many of the trials Kratos failed to pass the chicane for a variety of reasons. One major issue was the maneuverability of our robot. The large wheelbase and weight of the robot made precision turning difficult, and there was very little room for error when passing through the narrow openings; on one of the runs, the robot was disqualified since it brushed against one of the obstacles on the way out of the last layer of the chicane. Another identifiable problem was our path planning strategy. If the path seemed nearly blocked (as in the chicane case) our A* search would often try to "escape" by backing up and attempting to find another way around. This is not necessarily unintelligent for a robot trying to reach a waypoint, but disregards the specific rules of the Autonomous Challenge. Finally, the camera's field of view did not allow it to see lanes close to the robot's body, and the lane detection algorithm had difficulty picking out the lanes in the middle of the chicane since only short pieces were visible. Lane detection

also required manual retuning as lighting conditions changed, causing the system to fail if not tuned frequently or if the sun was near the horizon (such that lighting conditions depended on the direction of travel).

5. CONCLUSION

In this paper we presented Kratos, a small autonomous ground robot capable of navigating a static obstacle field. Kratos was ultimately awarded 3rd place overall, Rookie of the Year, 1st place Design Competition, 4th place Navigation Challenge, and 6th place Autonomous Challenge at the 2008 Intelligent Ground Vehicle Competition. Work is currently underway on a new robot, building on Kratos' design, to be entered in the 2009 IGVC.

ACKNOWLEDGMENTS

Kratos would not have been possible without the contributions of teammates David Benjamin, Ben Chen, Will Hu, and Tom Yeung. We are deeply indebted to Professor Robert Schapire for his advice and support throughout the development of Kratos. Thanks are also due to the Princeton Autonomous Vehicle Engineering Prospect 12 team, especially Professor Alain Kornhauser and team leader Issa Ashwash, who always seemed to have replacements ready when equipment failed. We are grateful for the generous support of the Keller Center for Innovation in Engineering Education, the School of Engineering and Applied Science, Department of Computer Science, and Department of Mechanical and Aerospace Engineering.

REFERENCES

[1] Atreya, A. R., Cattle, B. C., Collins, B. M., Essenburg, B., Franken, G. H., Saxe, A. M., Schiffres, S. N., and Kornhauser, A. L., "Prospect eleven: Princeton university's entry in the 2005 darpa grand challenge," *Journal of Field Robotics* **23**(9), 745–753 (2006).

[2] Kornhauser, A. L., Ashwash, I., Baldassano, C. A., Benjamin, D., Cattle, B., Collins, B., Downey, A., Franken, G., Gorman, L., Glass, J., Glass, Z., Herbach, J., Hu, W., Mayer, J. R., Momen, S., Saxe, A. M., and Yu, D. D., "Prospect Twelve: Princeton University's Entry in the 2007 DARPA Urban Challenge," Submitted to *IEEE Transactions on Intelligent Transportation Systems* (2008).

[3] Thiesen, B., "Igvc competition rules," (2008). http://www.igvc.org/rules.html.

[4] BaneBots, I., "Two 62mm cim motor specifications," (2007). http://banebots.com/pc/P80A-nnnn-0015-M4/P80A-43-0015-M4.

[5] Baldassano, C., Benjamin, D., Chen, B., Franken, G., Hu, W., Mayer, J., Saxe, A., Yeung, T., and Yu, D., "Kratos: Princeton university's entry in the 2008 intelligent ground vehicle competition," 2008 IGVC Tech Paper (2008).

[6] Davies, R. B., "Writing a matrix package in C++," in [*OON-SKI94*], 207–213 (April 1994).

[7] Partow, A., "Wykobi computational geometry library," (2007). http://www.wykobi.com/.

[8] Simmons, R. and James, D., *Inter-Process Communication: A Reference Manual* (August 2001). http://www.cs.cmu.edu/afs/cs/project/TCA/ftp/IPC_Manual.pdf.

[9] Jackson, J., "Microsoft robotics studio: A technical introduction," *IEEE Robotics & Automation Magazine* **14**, 82–87 (December 2007).

[10] Manduchi, R., Castano, A., Talukder, A., and Matthies, L., "Obstacle detection and terrain classification for autonomous off-road navigation," *Autonomous Robots* **18**(1), 81–102 (2005).

[11] Fischler, M. A. and Bolles, R. C., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM* **24**(6), 381–395 (1981).

[12] van der Merwe, R. and Wan, E., "Sigma-point kalman filters for integrated navigation," in [*Proceedings of the 60th Annual Meeting of The Institute of Navigation (ION)*], (2004).

[13] Larsen, T. D., Hansen, K. L., Andersen, N. A., and Ravn, O., "Design of kalman filters for mobile robots; evaluation of the kinematic and odometric approach," in [*Proceedings of the 1999 IEEE International Conference on Control Applications*], **2**, 1021–1026 (1999).

[14] Russell, S. and Norvig, P., [*Artificial Intelligence: A Modern Approach*], Prentice-Hall, Englewood Cliffs, NJ, 2nd ed. (2003).

[15] Hoffmann, G. M., Tomlin, C. J., Montemerlo, M., and Thrun, S., "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in [*Proceedings of the 26th American Control Conference*], 2296–2301 (2007).