Team members

Donald Abrams (Senior, CIS)
Sherol Chen (2nd-year grad student, CIS)
Patrick Griffith (Sophomore, ECE)
Amit Hetawal (2nd-year grad student, CIS)
Ezra Kissel (2nd-year grad student, CIS)
Gayathri Mahalingam (2nd-year grad student, CIS)
Kevin McCormick (2nd-year grad student, CIS)
Adam Milecki (Senior, ECE)
Rohith MV (1st-year grad student, CIS)
Ben Nielsen (Junior, ME)
Gowri Somanath (1st-year grad student, CIS)
Donald Scott (Junior, CIS)
Steven Taylor (Senior, CIS)
Bill Ulrich (5th-year grad student, CIS)

Faculty adviser statement

I certify that the engineering design in the vehicle by the current student team has been significant and equivalent to what might be awarded credit in a senior design course.

_____

Prof. Christopher Rasmussen
Dept. Computer & Information Sciences
University of Delaware

1 Introduction

Warthog is the inaugural entry of UD Robotics in the Intelligent Ground Vehicle Competition. The inception of the team was an informational meeting held in November, 2006, and the RMP 400 robot base was ordered from Segway soon after. Nearly 30 interested students interviewed for positions, and about half were invited to join the team. All team members enrolled in a 3-credit Computer & Information Sciences graduate seminar taught by Prof. Rasmussen in the spring called "Robot Navigation and Autonomy." The robot was delivered at the end of January, and the first meeting of the course took place the following week.

The fourteen students on the team were divided into three subteams: Vehicle, Sensing, and Planning. The Vehicle subteam was responsible for mechanical and electrical design and shop work; the Sensing team was concerned with obstacle detection and map creation; and the Planning team worked on aspects of waypoint navigation, obstacle avoidance, and low-level motion control.

Hardware and software issues, milestones, goals, and resources were tracked through a password-protected MediaWiki wiki page (http://www.mediawiki.org/wiki/MediaWiki). In addition, several key project management tools were used. All code was kept in an SVN repository (http://subversion.tigris.org), Trac was used for bug tracking and as an interface for browsing the code (http://trac.edgewall.org), Doxygen was used for code documentation (http://www.stack.nl/~dimitri/doxygen), and Mailman was employed for mailing lists and mail archiving (http://www.gnu.org/software/mailman).

In a little more than four months we have put in an estimated 1000 person-hours to furnish Warthog with all of the hardware and software it needs to attain competition-ready autonomy. We have accomplished a great deal and are looking forward to IGVC 2007.

Innovations

As this is our "rookie" entry, almost everything on Warthog is an innovation for us. The RMP 400 platform (detailed below) provided a significant mechanical head start in that issues related to the motors, motor batteries and charging, gearing and drive system, chassis, and an API for low-level control and proprioceptive sensing had already been engineered by Segway. However, this made for nothing more than a very rugged remote-controlled toy. All sensing and software necessary to imbue Warthog with autonomy had to be designed, created, integrated, and tested by us over the course of the semester.

The Segway RMP 400 chassis on delivery was basically an "empty box," as seen in the picture at right. We had to fashion an internal shelving system, lay out and secure all internal devices and interconnects, and mount and calibrate all external sensors. We also had to develop an auxiliary power source to run all of the devices that we added.

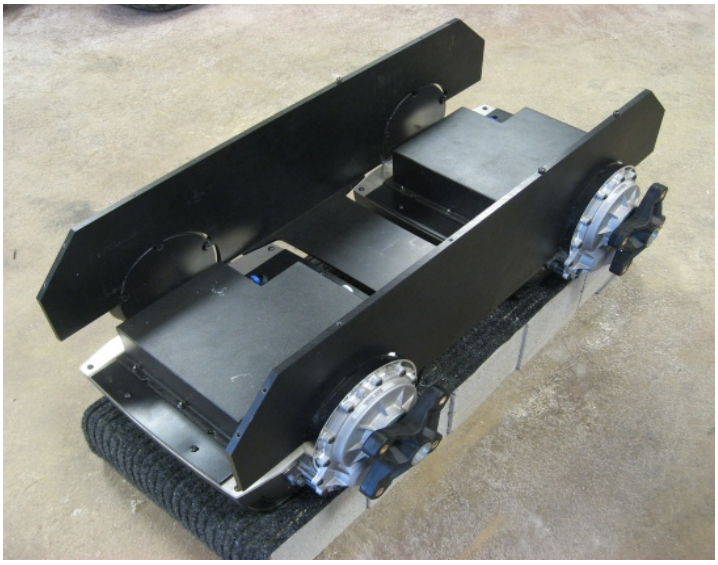


Figure 1: RMP 400 as delivered (wheels and lid removed)

Our software development process was extensive. Although it initially appeared that we had an excess of students, there were so many papers to read, algorithms to code and test, and so much integration to be done that the team felt stretched a bit thin by the end of the term. Nonetheless, we crafted a multi-layered system from off-the-shelf algorithms and custom code that exhibits several innovative aspects including:

* Fusion of ladar and computer vision for robust obstacle detection
* Downward-pitched ladar for variable-height obstacle detection
* Simultaneous Localization and Mapping (SLAM) for reliable obstacle mapping

2 Hardware

2.1 Chassis and drive system

Warthog is based on a Segway RMP 400. The RMP 400 is a 4-wheel, differential drive or "skid steer" vehicle with 21" ATV tires. Each pair of wheels (front and rear) constitutes a *powerbase*. In contrast to other Segway products which have only one powerbase, the RMP 400 is *statically* stable rather than dynamically---it does not require motion to achieve balance. An independent electric motor supplying 0.071 Nm/amp of torque at the motor shaft drives each wheel through 24:1 gearing. The motors are capable of 70 amps peak current per wheel and 24 amps continuous current. The top speed of the RMP 400 is 18 mph; this is limited in hardware for the competition to 5 mph. The robot can climb 45 degree slopes and make zero-radius turns (aka "spin in place").

Two 72V lithium ion batteries run the motors in each powerbase. Across both powerbases, these

have a total capacity of 1600 watt-hours and provide an average run time of 12 hours under good terrain and temperature conditions. A charger integrated into each powerbase recharges the batteries from empty in an average of 8 hours. Two buttons control operation of each powerbase. One supplies power to the User Interface (UI) electronics, and the other activates the motors. Both must be pressed before a powerbase can move, so four buttons must be pushed before the entire robot is ready to receive and act upon motor commands.

As delivered, the RMP 400 had a length of 44.5", width of 30.5", and height of 21". Ground clearance is about 3.5". Our physical modifications consisted of installing an internal polypropylene and aluminum shelving system to secure electronic devices, batteries, and the control computer, as well as external mounting of the sensors and some switches and buttons on the top and rear plates, respectively. These increased the height to 42" (the top of the GPS antenna) while leaving the length and width unchanged.

2.2 Safety

A large red e-stop button is mounted on the rear of the vehicle and is attached directly to the Segway-provided e-stop circuits of both powerbases . This button physically latches--it must be pulled out to disengage it. However, the Segway e-stop circuit actually shuts the motors off rather than simply pausing them. Thus, in addition to unlatching the e-stop button, the motor activation buttons must be pressed to bring the robot out of e-stop. Wireless e-stop functionality is provided by a Linear DXR-701 single-channel digital receiver with an isolated relay output. The nominal range of this device is 500 feet in open air. The remote e-stop is connected in series with the manual e-stop such that triggering either one causes an e-stop.

2.3 Computing

The sole computer controlling Warthog is a Dell XPS M1710 laptop with an Intel Core Duo T2600 2.16 GHz CPU and 2 Gb RAM, running the Debian Linux operating system. The internal lithium-ion battery provides an average run time of about 2.5 hours. All sensing and planning software runs as a single process. The computer is connected to the RMP 400 via two USB cables, one per powerbase (duplicate commands are sent to the front and rear powerbases). Low-level motion commands are sent in the form [*desired forward speed, desired turn rate*]. The UI electronics take care of PID control to achieve and maintain the commanded values.

A wireless Logitech Rumblepad joystick is used to control the robot remotely when necessary. In

open air, it offers an effective range of 50-100 feet. For higher-level remote commands and telemetry, we use a Fujitsu P1610 tablet PC running Windows XP communicating via an ad-hoc wireless connection. This computer is extremely portable, weighing about 1.0 kg, and has an 8.9" touch-sensitive screen. It makes it easy to walk along with Warthog over all kinds of terrain while monitoring the state of its software as it processes sensor readings and plans motions.

2.4 Sensors

Warthog's primary sensors are a SICK LMS-291 laser range-finder, a Unibrain Fire-i400 Firewire color camera, a Novatel Propak-V3-HP GPS, and a PNI TCM2-50 digital compass. The Segway RMP 400 base also provides extensive proprioceptive sensing regarding odometry, wheel torques, pitch and roll angle and rotational velocities, remaining battery life, and so on. A diagram of all external devices and how they are connected to the control computer is given below.

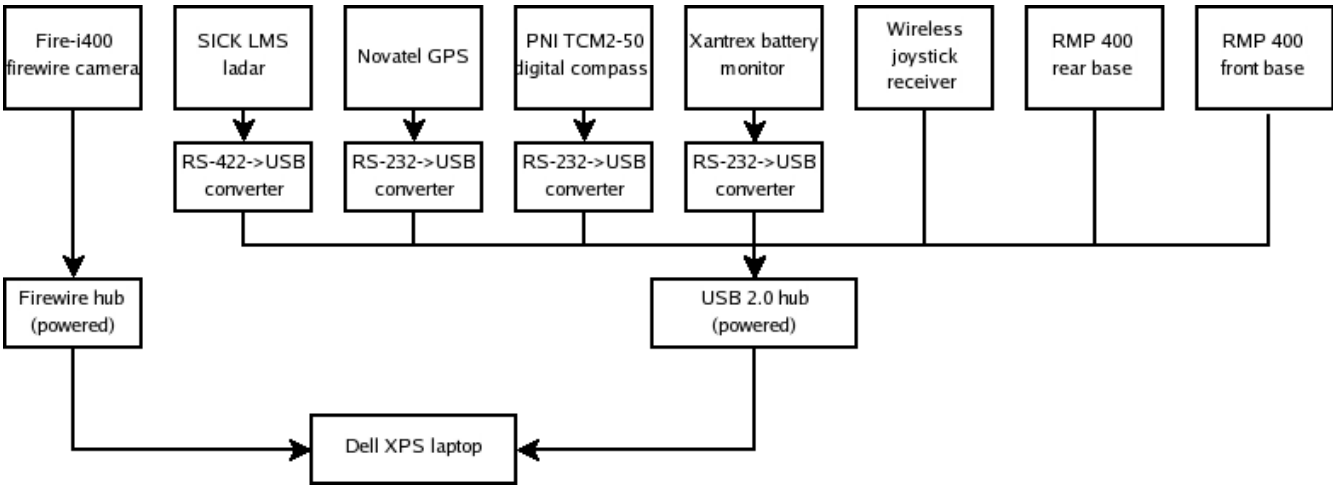


Figure 2: List of sensors and how they connect to the control computer

2.5 Auxiliary electrical system

While the RMP 400's motors are powered by Segway-provided batteries integral to each base, all other electrical devices except the computer require a separate power system. Two 12V, 32 Ah AGM deep-cycle lead-acid batteries are connected in series to create a 24V battery. These weigh 50 lbs. total. AGM batteries are excellent for the rugged conditions created by autonomous vehicles; they cannot spill even if broken and can be mounted in any orientation. An externally-mounted Xantrex

battery monitor with an LCD display shows the state of the battery. It has a serial connection to the control computer to report charge remaining for graceful shutdowns. A 24V, 8 A smart charger also rides onboard Warthog, permanently attached to the battery. Plugging in its cable to an AC outlet commences charging that does not need to be monitored.

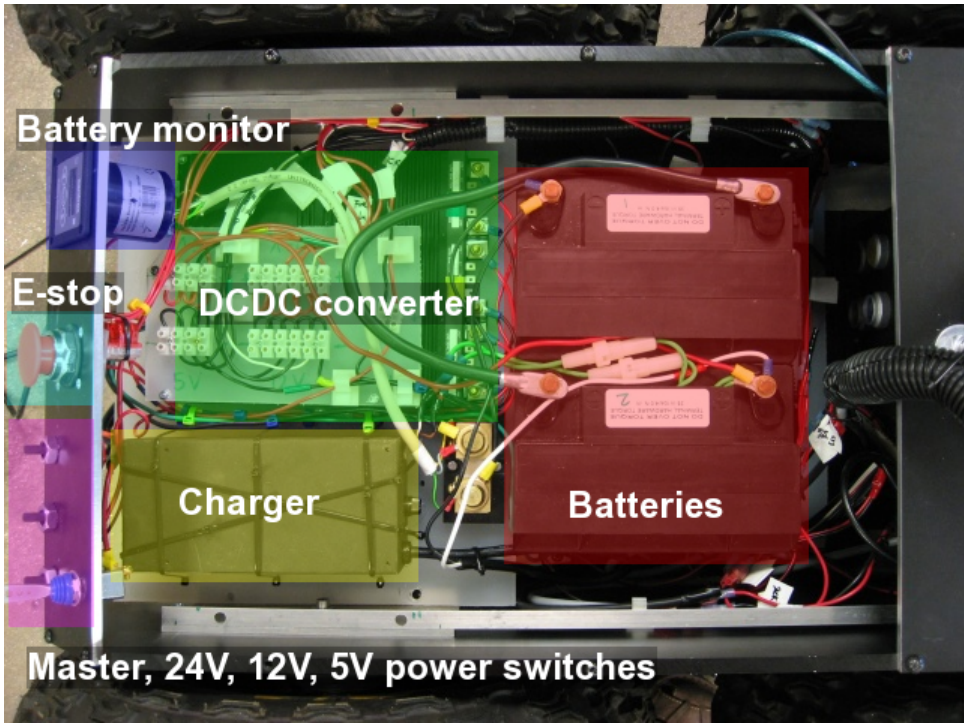


Figure 3: Auxiliary electrical system components

The batteries are connected via a fuse to a Vicor ComPAC DC-DC converter which outputs 24V, 12V, and 5V power at up to 100 watts per output. The Vicor offers EMC filtering, transient protection, and reverse polarity protection at about 85% efficiency. The arrangement of the various devices in the auxiliary electrical system is shown in the figure above.

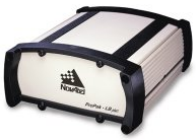
At the top of the next page is a table of the electronic devices which receive their power from the DC-DC converter (the battery monitor draws power directly from the battery). As can be seen, the watts drawn from the DC-DC converter are nowhere near its limits. The total average current draw, with a 10% safety factor added, is about 1.6 A. Given the capacity of the batteries and the efficiency of the converter, this translates into an expected run time of about 8.5 hours to go from a full charge to a 50% charge.

24V devices	Average power	Average amps
SICK ladar	20 W	0.84 A
12V devices		
Novatel GPS	2.5 W	0.21 A
Firewire hub (camera)	1 W	0.084 A
Linear wireless e-stop receiver	0.72 W	0.06 A
5V devices		
USB hub (wireless joystick)	1 W	0.20 A
TCM2-50 digital compass	0.1 W	0.02 A

Table 1: Power consumption of electronic devices

2.6 Major component list

The following table lists the major components detailed in the previous subsections that went into the construction of Warthog.

	Item	Retail cost	Our cost
	Segway RMP400	$30,000	$28,500
	SICK LMS-291	$5,000	$0
	Unibrain Fire-i400	$425	$0
	Novatel Propak-V3-HP	$5,500	$2,700
	PNI TCM2-50	$800	$0

	Item	Retail cost	Our cost
	Linear DXR-701+ remote	$83	$83
	Manual e-stop button	$80	$80
	D-Link USB hub	$27	$27
	2 x Concorde SunXtender PVX-340T battery	$188	$188
	Vicor ComPAC	$436	$436
	Soneil 2416SRF charger	$160	$160
	Xantrex battery monitor	$225	$225
	Dell XPS M1710	$4,155	$0
	Fujitsu tablet PC	$1,629	$0
	Logitech Rumblepad	$20	$0
	Total	$48,728	$32,399

Table 2: Major hardware components and their costs

3 Software

Warthog's software architecture consists of a set of *modules*, or C++ classes, which provide a particular functionality like trajectory following, mapping, waypoint homing, and so on. These are instantiated and their methods are called as needed by a single main() function, one for the Navigation Challenge (NC) and one for the Autonomous Challenge (AC). All modules and submodules used are listed below. Unless otherwise noted, each module is used for both challenges.

Module	Purpose
Camera	Image capture, internal/external calibration, low-level image processing
Ladar	Range data capture, filtering, transformation to vehicle coordinate frame
LineTracker (AC)	Detection of near-parallel painted lines in image sequences
MotionPlanner * FieldDStar * TangentBug * TrajectoryFollower	Reactive obstacle avoidance, waypoint homing, path planning given map, following planned path, etc.
ObsMap * VehicleMap * GlobalMap	Maintain obstacle map in vehicle and UTM frames, perform SLAM to mitigate noisy state
ObsTracker	Detection of barrels, barricades, and other above-ground obstacles using ladar and image data--decides what goes into map
State	Measure and filter robot position, heading, velocity, battery levels, and other variables based on various sensor readings
Trajectory	Representation of planned paths in UTM frame. Distance + angle TSP on unordered waypoints (NC), splining for smoothness

Table 3: Software modules used for both challenges

External libraries used include OpenCV for computer vision (http://opencvlibrary.sourceforge.net), the Bayes++ Bayesian Filtering library for state filtering (http://bayesclasses.sourceforge.net), and Player/Stage (P/S) for robot interfacing and simulation (http://playerstage.sourceforge.net) [GVH2003]. P/S's simulation capabilities have been especially helpful in prototyping approaches to both challenges, allowing arbitrary obstacle, waypoint, and line configurations while accurately simulating ladar returns and the robot's dynamic capabilities. A screenshot of a simulation of the Autonomous Challenge is shown at the top of the next page.

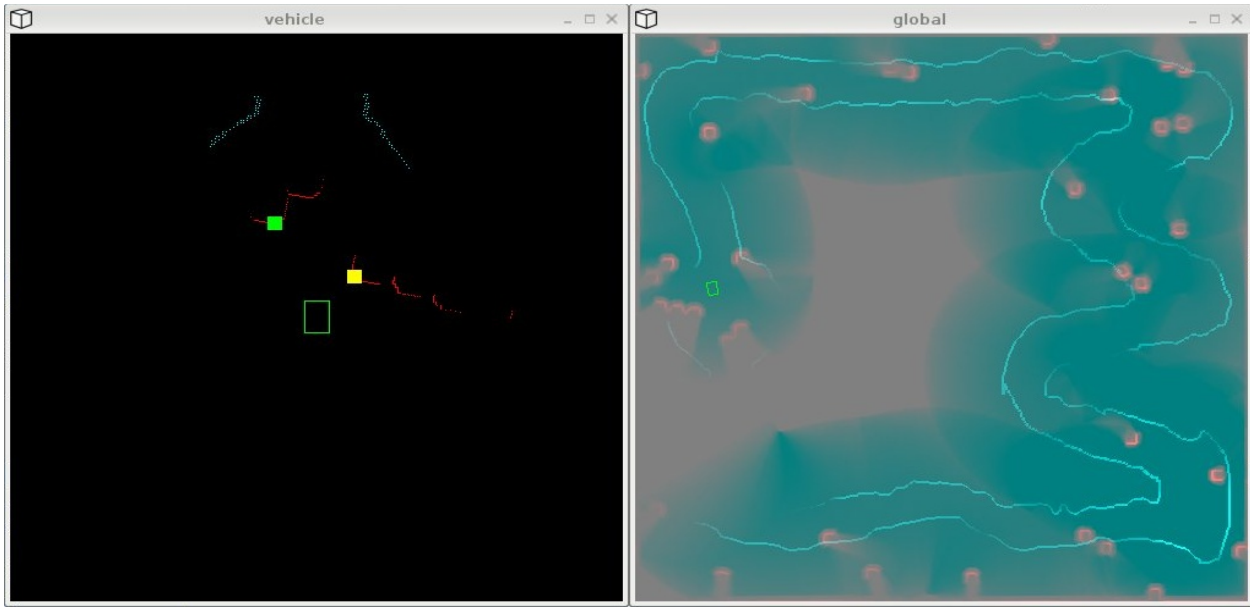


Figure 4: Autonomous Challenge simulation using Player/Stage robot library. A visualization of ladar obstacles and painted lines is shown in vehicle coordinates in the left image (Warthog is the empty green rectangle). The right image shows a global map of the same scenario combining barrels and lines seen over the course of the run.

3.1 ObsTracker (obstacle detection)

Our SICK laser range-finder is set to a maximum range of about 8 m for maximum range resolution, a 180 degree field of view, and a 30+ fps refresh rate. It is mounted about 0.5 m (20") above the ground and is pitched down so that the laser can detect 0.125 m (5")-high obstacles directly in front of Warthog about 8 m away. 5-gallon pails are the shortest positive obstacle we must detect for either IGVC challenge, and these are at least 0.25 m (10") tall. Traveling at the maximum allowed speed of 5 mph (2.235 m/s), such obstacles will be visible to the ladar for about 1.2 s, allowing for reliable detections based on multiple sightings. Barrels, cones, trees, and other obstacles taller than the ladar height are of course visible at any range less than the maximum.

Color image-based obstacle detection algorithms are used as a supplement to ladar sensing of obstacles because they offer a wider range of detection distances, if at some cost of reliability. First, obstacles can be detected in images beyond the 8 m ladar range in order to start avoidance maneuvers earlier. Second, nearby obstacles invisible to the ladar because of their height and position (e.g., the crossbar of a barricade or a very close 5-gallon pail which may be out of the ladar scan plane) are still visible to the camera. We use two basic strategies: (1) explicit detection of orange-and-white traffic barrels, the most common obstacle; and (2) detection of obstacles as "non-ground" through modeling of the ground color distribution. The latter approach is especially important

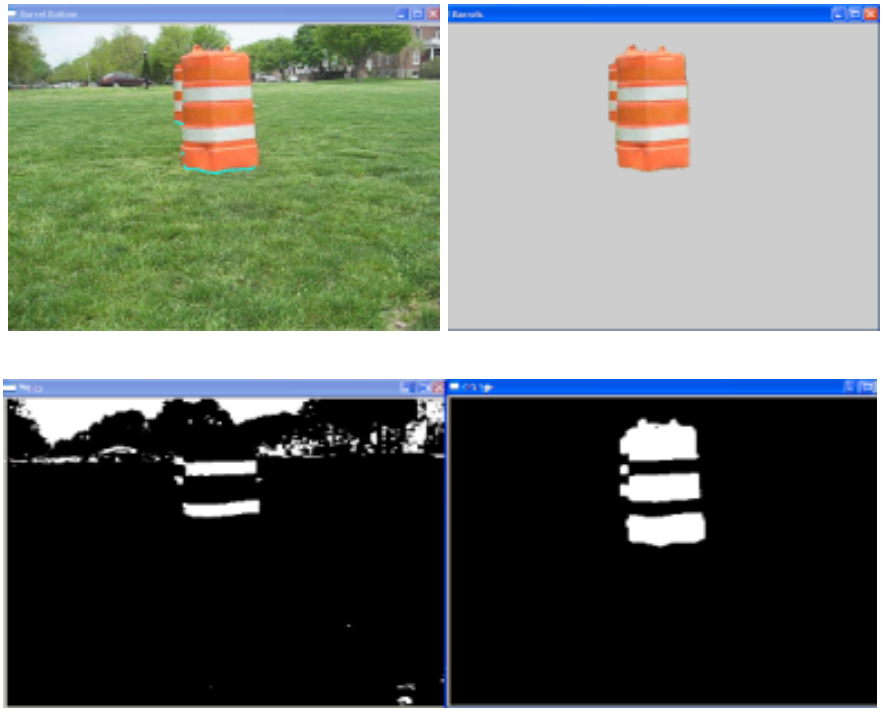


Figure 5: Color barrel detection. (Upper left) Captured image with final detected barrel bottoms in cyan; (Upper right) Segmented barrels; (Bottom left) Detected white pixels; (Bottom right) Detected orange pixels

for barricades, whose middle sections may entirely elude ladar detection. Without explicit depth information, image-detected obstacles can be transferred to the VehicleMap (a vehicle-coordinate map of obstacles) with appropriate internal and external calibration of the camera and a planar ground assumption via their bottoms (i.e., where they touch the ground).

Barrel detection

White and orange pixels are preliminarily detected in each image through image-normalized thresholds. Vertical derivatives are performed on these color masks to find horizontal stripe edges, then a state machine classifies vertical strips as belonging to a barrel based on whether they exhibit appropriate alternation and enough stripes are present. An example image containing a barrel, the masks generated for each color, and the final segmentation are shown in the figure above.

Ground color modeling

An alternative to looking for specific obstacle types is to assume that the ground, typically grass for IGVC, represents the dominant color in the image. If this color or mixture of colors can be reliably

determined as the robot moves, then pixels which do not match it may be assumed to be obstacles. In the paper [UN2000] (see References section), the ground colors are obtained from a trapezoidal reference area at the bottom of the image that is assumed free of obstacles. These colors are modeled with a histogram or mixture of Gaussians that is adapted as new images are captured. Steps of the algorithm for a sample image are shown below.

The reference area need not be assumed to be free of obstacles. In recent work [DKSTB2006], an approach to dynamically resizing the reference area to exclude ladar-detected obstacles was shown to work quite well, and we use this technique.

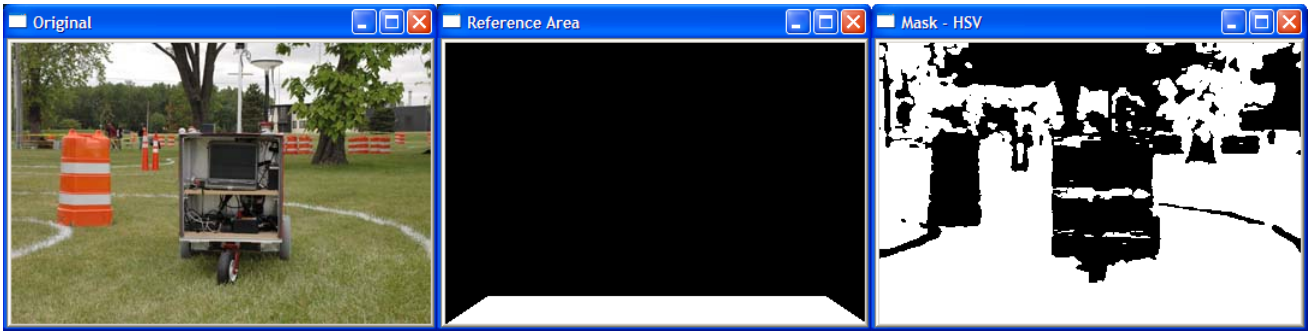


Figure 6: Non-ground segmentation. (Left) Captured image; (Center) Assumed ground reference area; (Right) Pixels matching ground color shown in white, obstacles in black

3.2 MotionPlanner

The generic MotionPlanner module handles robot movement at the lowest level. It implements reactive obstacle avoidance based on the currently-sensed ladar and vision obstacles regardless of what high-level goal is being sought. Thus, while other modules (described below) may be directing the robot toward a waypoint or down a lane, MotionPlanner modifies all steering and speed commands on their way to the powerbases to make sure that no collision occurs along the way.

TangentBug

The Bug family of motion planning algorithms navigate a 2-DOF mobile robot in a completely unknown environment with global position knowledge and local obstacle knowledge. This perfectly describes the situation at the outset of the Navigation Challenge. TangentBug [KRR1998] is an algorithm in this family which uses range data (e.g., from a SICK ladar) to compute a locally shortest path, based on a structure termed the local tangent graph (LTG). The LTG describes the extents of visible objects in polar coordinates, and the robot uses it to choose the locally optimal direction while moving toward the

target (*motion-to-goal*). Local minima in obstacle configurations which might otherwise trap the robot are explicitly detected and overcome through a *wall-following* behavior. The transition between these two modes of motion is governed by a globally convergent criterion which is based on the distance of the robot from the target.

We made several modifications to the standard TangentBug algorithm to implement it on a real robot. First, the limited field of view of our ladar (180 degrees rather than the full 360 assumed) required changes to the definition of a local minimum and to the basic characteristics of the wall-following mode. Second, the abstract algorithm treats the robot as a point, so the reactive obstacle avoidance provided by the base MotionPlanner is critical to avoid brushing against obstacles as they are passed.

Mapping

In both the Navigation Challenge and the Autonomous Challenge, as Warthog encounters more of the environment its knowledge of obstacle locations increases. Given the very accurate position information provided by our GPS device, it is straightforward to "remember" obstacle locations by placing them in the GlobalMap to facilitate longer-range motion planning. This GlobalMap is a 2-D occupancy grid map indexed by UTM coordinates which tabulates confidences that locations are free of obstacles based on accumulated evidence from multiple detections as the robot moves [TBF2005]. The robot position and heading parameters of the State are filtered from the raw GPS measurements using Bayesian filtering, but they are still somewhat uncertain, which causes GlobalMap to have somewhat "smeared" obstacles. When obstacles are close together, this could lead to the appearance of there being no passage between them when in fact there is one. In order to mitigate this problem, we use a variant of Simultaneous Localization and Mapping (SLAM) to improve the registration of newly-detected obstacles with the existing GlobalMap. This is accomplished by adapting the efficient DP-SLAM algorithm [EP2005], which can create accurate maps using just ladar data and noisy odometry.

Field D*

With GlobalMap we can plan paths through already-seen areas beyond the range of Warthog's sensors. This is desirable because TangentBug, while provably converging on the goal, rarely generates the shortest path due to its exploratory wall-following behavior. A* is a standard approach to finding the shortest path from start to goal through a set of known obstacles. However, it is expensive to run repeatedly as the start location changes due to robot motion and additional obstacles may be discovered. Several efficient variants of A* have been developed to overcome this problem.

We use one of them, Field D* [FS2006] , which runs quickly and also generates smoothly-curving paths. In the Navigation Challenge Field D* can be used to generate a path to the next waypoint after the robot has visited the first few and "painted" a large fraction of the competition area. In the Autonomous Challenge it is used more selectively in complicated barrel configurations to help Warthog "escape" to the other side more efficiently.

3.3 Trajectory and TrajectoryFollower

As waypoints in the Navigation Challenge are unordered, choosing an efficient ordering with no knowledge of obstacle locations is a Traveling Salesperson Problem (TSP). This is an NP-complete problem, but a brute-force approach is feasible for N < 10 waypoints, which is the expected number given the competition history. For higher N we fall back to an approximate algorithm. For this year's IGVC, the field is separated into northern and southern sections by a fence with one opening (location specified), making the underlying graph incompletely connected. A variant of the TSP which we also considered incorporates angular information into route costs in an effort to balance turning vs. distance traveled, given that there is a time cost associated with large rotations as well as straight-line segments.

From the TSP-generated waypoint ordering, a set of intermediate waypoints is created using splining to obtain a smooth Trajectory. In the absence of obstacles along the trajectory, this is the path that is followed. Given a set of trajectory points and the current position and heading output by the State module, TrajectoryFollower is like a virtual wall-follower which constantly updates the robot's turning rate and speed in order to "trace" the line. Our steering control law is modeled after Stanford's Stanley robot, which won the 2005 DARPA Grand Challenge [Thrun2006]. Its form is:

$$\psi(t) = \delta(t) + \arctan[\, k\, x(t) \,/\, u(t) \,]$$

where at time t $\delta(t)$ is the angular difference between the robot heading and a tangent at the nearest point on the trajectory, $x(t)$ is the lateral distance (aka *cross-track error*) to that point, and $u(t)$ is the robot's current speed. The constant k is a gain parameter. In the case where there are no lateral errors, the control law simply attempts to turn the robot parallel to the given trajectory. The second term involving the cross-track error steers the robot in a non-linear fashion towards the trajectory. The larger the cross-track error, the more severe the turning rate towards the trajectory will become. In order to follow the trajectory points as closely as possible, the robot must reduce its velocity as it

encounters turns and sharp bends that may appear in the provided trajectory. Accordingly, the robot reduces its velocity exponentially as a function of turning rate.

3.4 LineTracker

The are two major phases to how lines are handled for the Autonomous Challenge. First, lines must be detected in images and transformed to vehicle coordinates. Second is the question of how to steer the robot to continue along the course without a definite goal point given the sometimes conflicting demands of avoiding physical obstacles and not crossing the lines.

Our line detection algorithm is based on the non-ground segmentation method described above in the ObsTracker subsection. This technique outputs candidate obstacle pixels including painted lines, but also other above-ground obstacles. We want to isolate lines in order to extract directional information from them, so we perform further testing on obstacle pixels. The whiteness of line pixels is not explicitly used, since this may vary with illumination conditions and ground material. Instead, several geometric filters are first run to remove stray pixels as well as regions larger than the expected with of the lines given the camera calibration. The remaining pixels are searched iteratively with an approximate continuity criterion to find curve segments. The Hough transform is not used because extreme curvature and broken segments make it unreliable. Any curve segments over a threshold length are output explicitly. Example output is shown in the figure below.

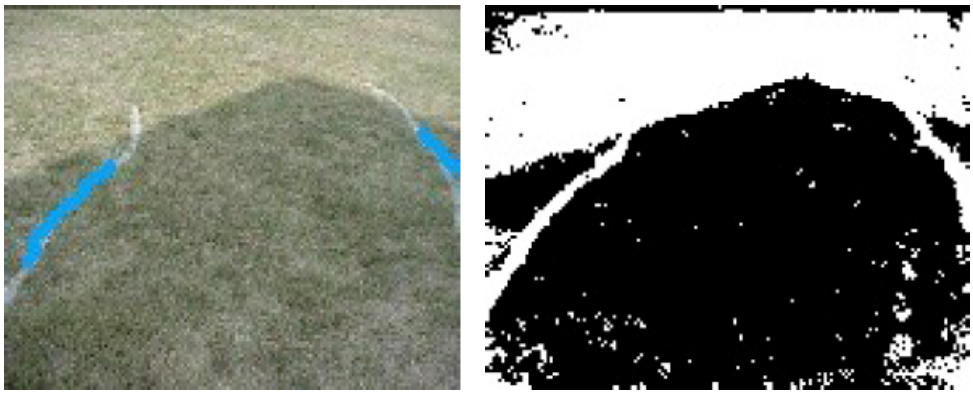


Figure 7: Line detection for Autonomous Challenge. The input image is shown on the left with the non-ground segmentation on the right.

Given the curve segments detected, we infer a vector field of directions via anisotropic diffusion from them, and the curves themselves are put into the VehicleMap as solid obstacles. The vector field gives a rough guide to which direction the robot should prefer to move in the absence of obstacles, while the lines in the VehicleMap prevent crossing them accidentally. At a higher level there is a

potential problem at hairpin turns and complex barrel switchbacks with the robot making a U-turn and heading back toward the start. This is prevented by using global information on where the robot has already been to "push" it toward novel regions.

4 References

[DKSTB2006] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. "Self-supervised monocular road detection in desert terrain." Robotics Science and Systems Conference (RSS), Philadelphia, PA, 2006.

[EP2005] A. Eliazar and R. Parr. "Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps," Neural Information Processing Systems Conference (NIPS), 2005.

[FS2006] D. Ferguson and A. Stentz. "Using Interpolation to Improve Path Planning: The Field D* Algorithm." *Journal of Field Robotics*, Vol. 23, No. 2, 2006.

[GVH2003] B. Gerkey, R. Vaughan, and A. Howard. "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems." International Conference on Advanced Robotics, Coimbra, Portugal, 2003.

[KRR1998] I. Kamon, E. Rimon, and E. Rivlin. "Tangentbug: A Range-Sensor-Based Navigation Algorithm." *International Journal of Robotics Research*, 17(9), 1998.

[TBF2005] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[Thrun2006] S. Thrun et al., "Stanley, the robot that won the DARPA Grand Challenge." *Journal of Field Robotics*, Vol. 23, No. 9, 2006.

[UN2000] I. Ulrich and I. Nourbakhsh. "Appearance-Based Obstacle Detection with Monocular Color Vision", National Conference on Artificial Intelligence (AAAI), 2000.