

Warthog

University of Delaware * IGVC 2010



Team members

Conor Gilsean (Senior, CIS)
Kevin Graney (Junior, CIS)
Kevin Schultz (Senior, ME)

Yan Lu (Grad student, CIS)
Mehmet Kocamaz (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 entering the Intelligent Ground Vehicle Competition for the fourth time, its rookie entry having been in 2007. Last year, after devoting considerable effort to upgrading both hardware and algorithmic capabilities for the Autonomous Challenge (AC), we were somewhat surprisingly rewarded with a 1st-place finish, though we did not complete the course. Our previous strength had been with the Navigation Challenge (NC), which we somewhat neglected in 2009 to focus on improving our AC performance. Despite this shift in emphasis, we finished 3rd in the NC with 7/9 waypoints reached. We analyze what happened in both challenges and our response in Innovations below.

This year, with much of the hardware drama of our first two years behind us and our core approaches to both challenges reasonably established, we have sought to greatly "robustify" and extend our algorithms. Even last year, despite our high rankings, we had our proverbial fingers crossed on every run hoping that certain situations would not arise. Beyond those few obvious cases, our understanding of what other types of course configurations might cause undesirable robot behavior was very narrow. Notably, our NC code was completely unaware (by design) of the boundaries of the course and of the existence of a prominent fence with a single open gate, leading to our disqualification on several runs for leaving the course. Our AC code had unwarranted confidence that it could always track both the left and right lines. When this assumption was violated late in the run, it confused the motion planner and the robot stopped when it could have continued.

Given the already-done work from previous years, the team this year could be fairly small. Many critical decisions were made by the entire team in weekly meetings held over the spring semester, while individual students worked on building simulator capabilities, researching hardware design changes, and developing the iPod interface. We estimate that an aggregate of 200 person-hours have been put into the project this year. As in previous years, all code was kept in an SVN repository (<http://subversion.tigris.org>).

Innovations

A key component of the effort this year to better characterize and improve the robot's behavior across diverse situations has been the development of a standalone robot **simulator** for both challenges. This simulator implements a virtual SICK ladar, GPS, Segway robot control and proprioception, and omnidirectional camera (rendering the world in simplified computer graphic form). We have used it

heavily to test our code on all kinds of hypothetical courses and situations. Our existing inter-process message-passing framework based on IPC (<http://www.cs.cmu.edu/~ipc>) has greatly simplified this endeavor. It has allowed the NC module to be run exactly as though it were really controlling the robot—it just does not know that the sensor modules and Segway module that it sends motor commands to are virtual rather than real. The AC module can be run similarly, although with a minor modification to receive virtual camera images as IPC messages rather than capturing them from the Firewire bus directly. This last feature has been aided by the use of a new OpenCV 2.1 (<http://opencv.willowgarage.com>) feature to compress and decompress image buffers in memory before sending them over IPC, allowing a large savings in network bandwidth.

For the **Navigation Challenge** we have a number of innovations this year. We finally found the time to add **awareness of the course boundaries**: based on the robot location and heading, nearby boundaries cause virtual “walls” to be inserted into the motion planner's obstacle map so that the robot turns away from them. Because of positional and heading uncertainty, these walls are inset slightly so that the robot does not get too close. Nonetheless, in the random event that the robot thinks it is outside the boundaries, the motion planner favors plans which get the robot back inside as quickly as possible. Also, in the course of simulator testing we discovered that our motion planner, which is greedy in its choice of robot maneuver which will get the robot closer to the next waypoint, can get the robot stuck in local minima arising from non-convex obstacle configurations. Although these seem to have been rare elsewhere on the course in three previous years of IGVC competition, the fence/boundary combination is one notable place where it can occur: if the robot encounters the fence, it may need to go a fair distance away from the waypoint before finding the doorway or hitting a boundary and turning around. To deal with this, we explicitly detect local minima à la the TangentBug algorithm (Kamon *et al.*, 1998) and enter a **wall-following mode** that permits the waypoint distance to increase. One possibility with Bug-like algorithms is always that the robot will wall-follow around the outer boundary of an environment. We explicitly rule that out here: the robot will not wall-follow the perimeter of the course.

On the off chance that the robot begins to wall-follow on the wrong wall or an isolated barrel, it is possible for it to loop forever. We therefore have added **loop detection** by remembering the location where wall-following began and leaving wall-following mode if it is revisited. Finally, the collision checking portion of our motion planner has been changed from a fixed to a **variable safety radius** to model the robot. A large safety radius leads the robot to keep better clearance from obstacles, but may make it think it can't fit through a smaller opening. A small safety radius makes the robot think it

can go more places, but it will come closer to brushing obstacles as it goes by them. To combine the best of both sizes, we now search for motion plans with large safety radii first and only go to smaller radii if we can't find one.

For the **Autonomous Challenge** our major change has been the addition of **several modes covering line-tracking failures** to improve robot behavior in less common situations. Chief among these is a **wall-following mode** which the robot reverts to if it loses track of one of the course lines. If, for example, the robot decides that it is not tracking the left line on the course (based on a threshold in the image processing routine), it attempts to follow the right line. The practical effect of this is that in wall-following mode we are not trying to estimate the course width, but rather to hug one side while avoiding obstacles until the opposite line is re-detected. This is important because in 2009 Warthog went far using just one mode (which you could call “both-line” tracking), but stopped when the path was wider than the maximum expected and a row of barrels blocked progress on the right side. There was space on the left but the robot had lost track of that line and thus “hallucinated” a path width that was too narrow, leading it to think that the path was completely blocked. We believe that this new mode will help Warthog have a better chance of repeated long runs. There is also a new **no line mode** which is very rarely invoked but necessary to preserve planner continuity in sections where the path is very wide or heavily occluded by obstacles.

A final motion planning innovation adopted for both the NC and AC is what we call **decisiveness**. Because the robot essentially replans at every time step, until this year that has been no built-in memory of what it decided to do “last time.” This meant that as the robot approached an isolated obstacle between it and its goal, it would sometimes oscillate between preferring a plan to the left and a plan to the right of the obstacle and not “make up its mind” until it was quite close. We now remember the last plan chosen and prefer the next plan to be “close” to it if at all possible.

A **hardware innovation** this year has been a move to two downward-looking omnidirectional cameras, allowing **stereopsis** (last year we had just one). These camera are visible in the robot picture on the title page. By default the left camera is used for AC line detection and the SICK lidar remains our primary obstacle detection sensor for both the AC and NC, but the right camera can be used as a back-up for the left and stereo depth maps can now be accessed in the event of lidar malfunction. Additionally, because the cameras are omnidirectional, we can construct stereo depth maps of the course just behind the robot that our single SICK lidar does not cover, and thus allow us to make better-informed reverse motion plans in sticky situations.

We continue to work on an **iPod Touch interface** for remotely monitoring the the robot when it is moving autonomously and for changing settings like which challenge it is about to perform, turning logging on and off, commanding "soft" stops (as opposed to the hardware e-stop), and so on.

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"), but to reduce stress on the motors we enforce a minimum radius for turning, which necessitates differential path planning.

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 alluded to above, the buttons for the front base were in a difficult to reach spot inside the chassis. Last year we solved this by removing one of the control boxes and rotating it 180 degrees to make the buttons face outward.

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 last year 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. An 8020 aluminum superstructure raises the primary camera by about 18" for a better angle for line detection, as well as adding shelves for the payload and two laptops. The sensor mast

is adjustable to a maximum height of about 55" (the top of the GPS antenna) while leaving the length and width unchanged. This year we also moved the rear bumper to make a stronger connection and to allow easier insertion/removal of the IGVC payload.

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 an aftermarket automotive keyless entry system which interfaces directly with the Segway-provided e-stop circuits. The nominal range of this device is 100 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. Warthog also sports 24"-wide front and rear bumpers from Tapeswitch wired in series with the other e-stop devices so that any crash stops the motors automatically. These bumpers require 10 lbs. of force to activate.

2.3 Computing

We have experimented with multiple networked computers controlling Warthog; all run Ubuntu 8.10 for ease of unified library updates, etc. Due to code optimization this year, though, we have reverted to a single Dell M2400 Precision workstation laptop with an Intel Core Duo T9600 at 2.8 GHz, 4 Gb RAM, and a 7200 rpm hard drive. The internal lithium-ion batteries in this machine provides an average run time of about 2.5 hours and must be swapped for continued operation. It is connected to the RMP 400 via USB (technically, two cables through a hub—one per powerbase, since 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 Segway 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 are developing a native iPod Touch app which communicates with a wireless router onboard the robot. This palmtop computer 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, as well as sending parameter changes or pause commands.

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 two Pt. Grey Flea2 cameras with a fisheye lenses for omnidirectional imagery. 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 IGVC-relevant external devices and how they are connected to the control computer is given below.

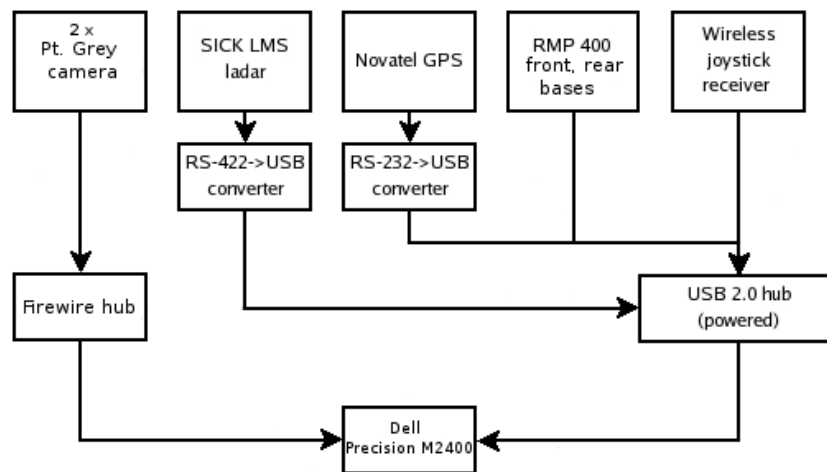


Figure 1: Primary sensors for IGVC 2010

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 workstation laptops require a separate power system. Two 12V, 32 Ah AGM deep-cycle lead-acid batteries are connected in series to create a 24V "auxiliary" 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.

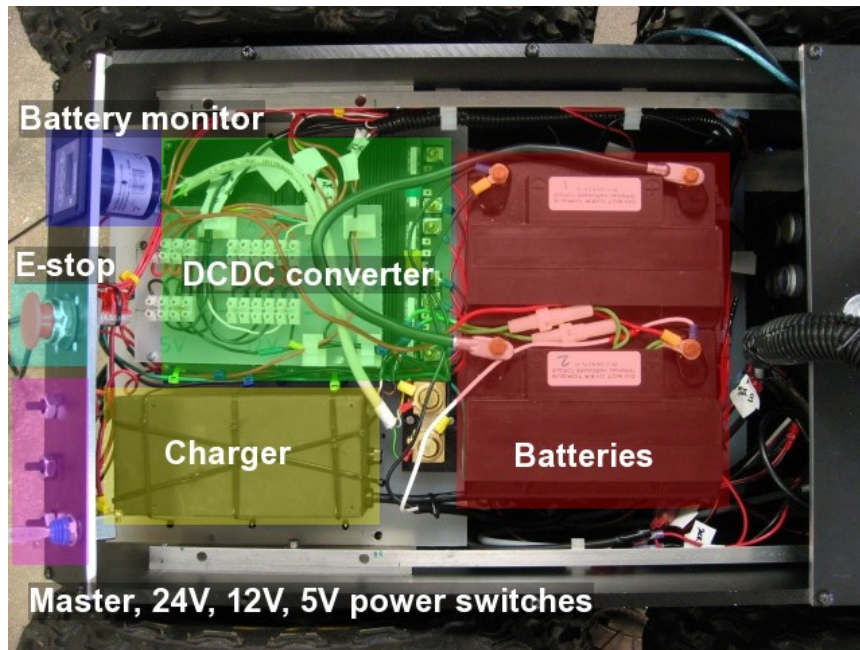


Figure 2: Auxiliary electrical system components

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.

Below 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 2.4 A. Given the capacity of the batteries and the efficiency of the converter, this translates into an expected run time of almost 6 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 (omni cameras)	1 W	0.084 A
Wireless e-stop receiver	1.2 W	0.1 A
5V devices		
USB hub (wireless joystick)	1 W	0.20 A
Wireless router	1 W	0.2 A

Table 1: Power consumption of electronic devices

2.6 Major component list

The table on the next page lists the major components detailed in the previous subsections that went into the construction of Warthog, their retail cost, and the cost to the team this year (items carried over from previous years are not counted in the latter category).

New Components	Item	Retail cost	Cost this year
	iPod Touch	\$200	\$200
	RIGHT omnidirectional camera (Pt. Grey Flea2 + Fujinon lens)	\$1,300	\$1,300
Existing components			
	Segway RMP400	\$28,500	-
	SICK LMS-291	\$5,000	-
	Novatel Propak-V3-HP	\$5,500	-
	LEFT omnidirectional camera (Pt. Grey Flea2 + Fujinon lens)	\$1,300.00	-
	Dell Precision M2400	\$900	-

	D-Link USB hub	\$27	-
	2 x Concorde SunXtender PVX-340T battery	\$188	-
	Vicor CompPAC DC-DC converter	\$436	-
	Soneil 2416SRF charger	\$160	-
	Xantrex battery monitor	\$225	-
	Logitech Rumblepad	\$20	-
	D-Link DGL-4300 wireless router	\$120	-
	Tapeswitch bumper x 2	\$350	-

Table 2: Major hardware components and their costs

3 Software

Warthog's software architecture consists of a set of *modules*, which this year are separate processes. Several of the modules are associated with device drivers—they talk directly to sensors and stream raw data, as well as providing log writing and reading functionality. One middle level module (*state*) filters the output of the sensor modules, but makes no actual decisions about what to do. Finally, at

the top level are the *pilot* modules, which are the only ones allowed to send motor commands. Only one of these is running at any given time. All modules used are listed below in alphabetical order. Unless otherwise noted, each module except *simulator* is used for both challenges.

Module	Purpose
boss	Process health monitor. Starts/stops/restarts modules as necessary using the Upstart daemon (http://upstart.ubuntu.com)
dashboard	GTK GUI interface for module management, real-time telemetry, and visualization of sensor data. Typically runs on wirelessly-connected laptop or tablet PC. A version for iPod Touch is being developed.
gps	Novatel GPS data capture, including UTM coordinates, heading, and velocity
ladar	SICK ladar range data capture and transformation to vehicle coordinate frame
pilot_auto (AC only)	Executive module for Autonomous Challenge. Performs image capture, internal/external camera calibration, low-level image processing, line detection and tracking. Integrates input from ladar to decide which direction to drive.
pilot_nav (NC only)	Executive module for Navigation Challenge; reads list of waypoints and integrates waypoint homing with obstacle avoidance.
segway	Segway RMP 400 data capture and motor control. Embedded in this module is joystick driver for manual piloting
simulator (New this year)	Simulate ladar , gps , segway modules as well as omnidirectional image capture from pilot_auto for various virtual environments
state	Measure and filter robot position, heading, velocity, and other positional variables. Used by pilot_nav to get a decent heading estimate from GPS + odometry without digital compass. Uses Bayes++ Bayesian Filtering library (http://bayesclasses.sourceforge.net)

Table 3: Software modules used for both challenges

A few selected modules are explained in more detail in the following pages due to space restrictions. **dashboard** was analyzed on last year's report.

3.1 simulator

The simulator we wrote this year encompasses simple 2-D worlds containing fixed height obstacles and lines on the ground. It replaces the functionality of the **segway** module, translating IPC motor command messages into motion and sending back telemetry regarding speed and turn rate; the **gps** module, sending absolute position and heading information artificially corrupted by noise; the **ladar** module, scanning the robot's local environment for obstacles; and a virtual omnidirectional camera

which sends images to **pilot_auto** for the line tracker to work on. Screenshots of sample NC and AC courses we have used for testing are shown below. Overlays related to **pilot_nav** and **pilot_auto** internal parameters will be discussed below, but note that obstacles are shown in black, course boundaries for the NC as blue lines, and painted lines for the AC as purple curves.

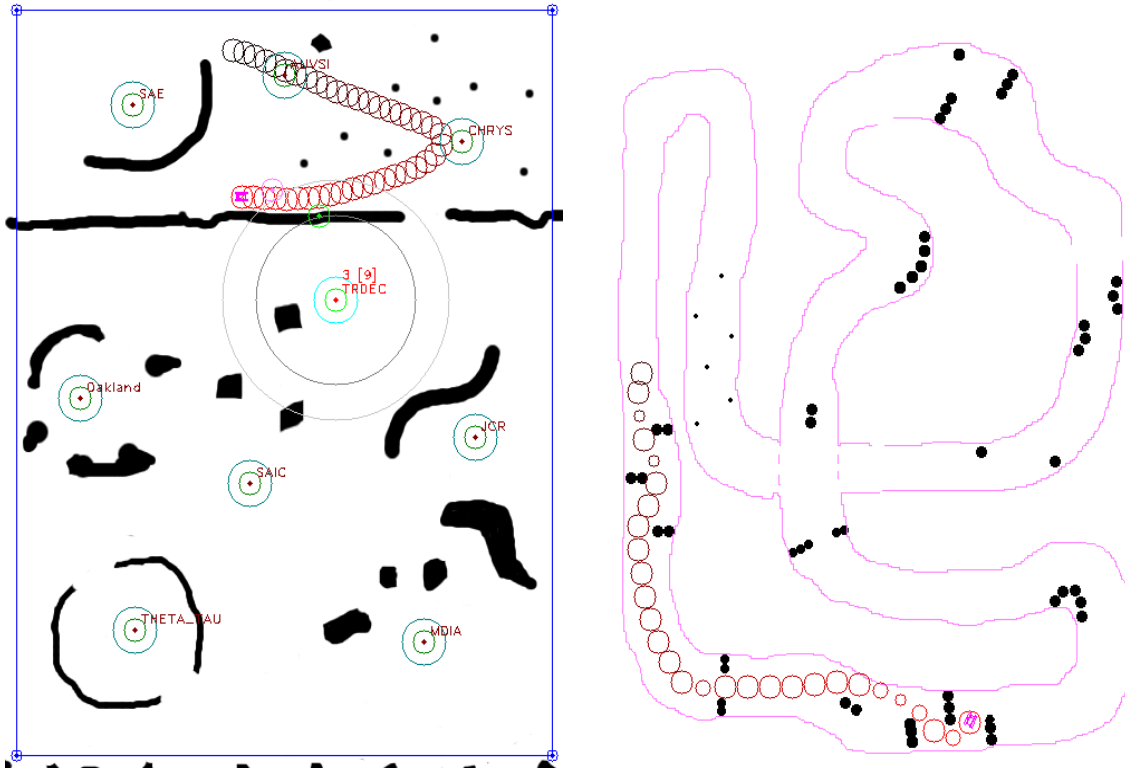


Figure 3: *simulator* module screenshots (NC course on the left, AC course on the right)

3.2 pilot_nav (Navigation Challenge)

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 with a level scan plane. This allows easy obstacle detection (vs. a downward-pitched scan plane) and longer-range motion planning, but makes shorter obstacles invisible. This year stereo is available to fill in blind spots, but we have generally not found these to be necessary at IGVC.

Our base motion planner is derived from a Dubins car model, which accounts for differential constraints on Warthog's motion in the form of a minimum turning radius. Under this model, the only maneuvers permitted are straight-line and circular arc (left or right) segments. The shortest Dubins plan from the current robot position to the next waypoint is always the first one we consider during the NC, and if this is obstacle-free the robot is in *direct mode*. If the direct plan collides with an obstacle

(based on the current safety radius), the robot enters *avoid mode* in which a set of candidate motion plans are generated and checked for collisions. The pattern of these plans tuned for each IGVC challenge. For the NC, the pattern is a foveated “spray”, or radial plans which range from 90 degrees right to 90 degrees left, with smaller angular increments closer to straight ahead. This pattern is replicated for 7 m, 5 m, and 3 m lengths. Among collision-free plans, the “best” plan to be executed is the one which gets the robot closest to a nearby goal point (for the NC, this is just the next waypoint). If no plan is collision-free, the safety radius is reduced. If all plans collide even at the smallest safety radius, the robot backs up. The spray pattern is visible in the leftmost figure below, which shows in yellow all collision-free plans, the best plan in green, and the direct plan in purple. This obstacle map and set of plans corresponds to the situation depicted in the NC simulator screenshot above, which also shows a recent history of the robot's locations as red circles.

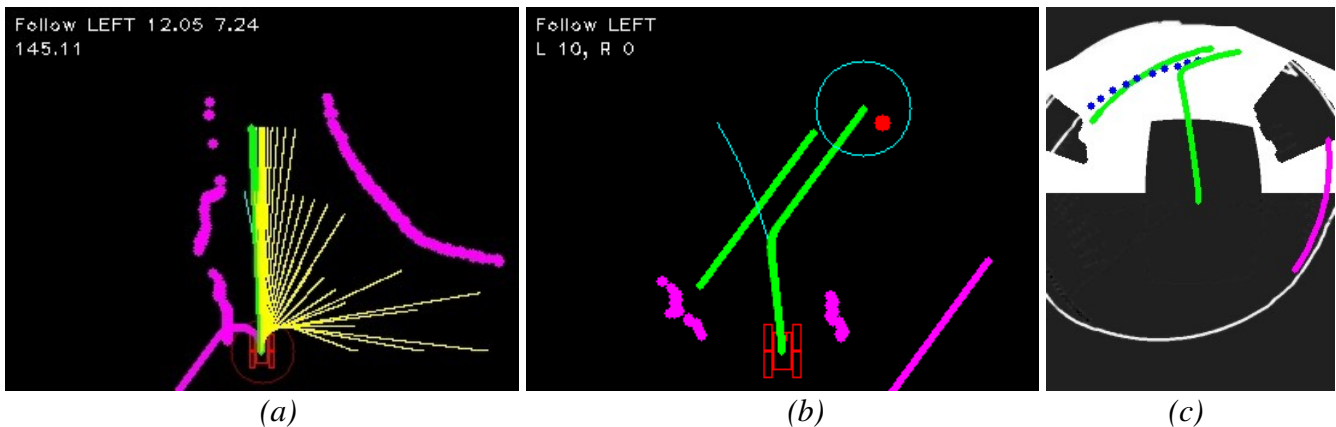


Figure 4: Motion planning. (a) *pilot_nav* obstacle map and plans from situation shown in Fig. 3 on left. Cyan curve is extrapolation of current motor command; (b) *pilot_auto* obstacle map from situation shown in Fig. 3 on right; (c) Corresponding omnidirectional image (from simulator) with masked-out regions drawn as black. Line detections are shown as blue dots.

If the robot detects that it is in a local minimum, it goes into *wall-following mode*, in which the goal point becomes the rightmost or leftmost reachable point, and only exits when the local minimum has been left, a boundary is reached, or the robot has looped around to the point where it started wall-following. In the left image of Fig. 3, the closest the robot has gotten to the next waypoint is shown as as a small green circle, and the point where it began to wall-follow as a small purple circle.

3.3 *pilot_auto* (Autonomous Challenge)

Our basic AC approach using an omnidirectional camera was described in detail in last year's report, so we only briefly review it. We model the approaching section of path very simply as two parallel lines and track it using a *particle filter* whose state has three variables, all in vehicle coordinates: (1)

angle between the path curve tangent and the vehicle heading, (2) lateral offset between the path center and the vehicle center, (3) and width of the path.

Particles consist of *state hypotheses* which are scored using a likelihood function where the image processing occurs. In this function a given hypothesis implies left and right line segments in front of the robot in vehicle coordinates. These are transformed to image coordinates and sampled at discrete intervals. At each interval along a given curve (typically there are 15-20 each for the left and right lines), pixels in a search window about 1 m wide orthogonal to the curve are examined. For each search window the ratio of the maximum to minimum intensity (aka the blue channel after median filtering) is computed. If the ratio is over a certain threshold, that window is counted as containing a line detection. The total number of line detections over all search windows across both the left and right curves is the likelihood for the whole hypothesis.

In order to not be distracted by contrasting colors on barrels or other obstacles, or the robot itself (since it appears in the omnidirectional image), a *mask* is created for each new frame based on the SICK ladar and knowledge of the camera position and robot dimensions. Any non-zero pixels in the mask are assumed to not belong to the ground, and thus are skipped in calculation of the likelihood function. Any search windows that have too many masked-out pixels are omitted from the mean score for the hypothesis because of insufficient data to be reliable.

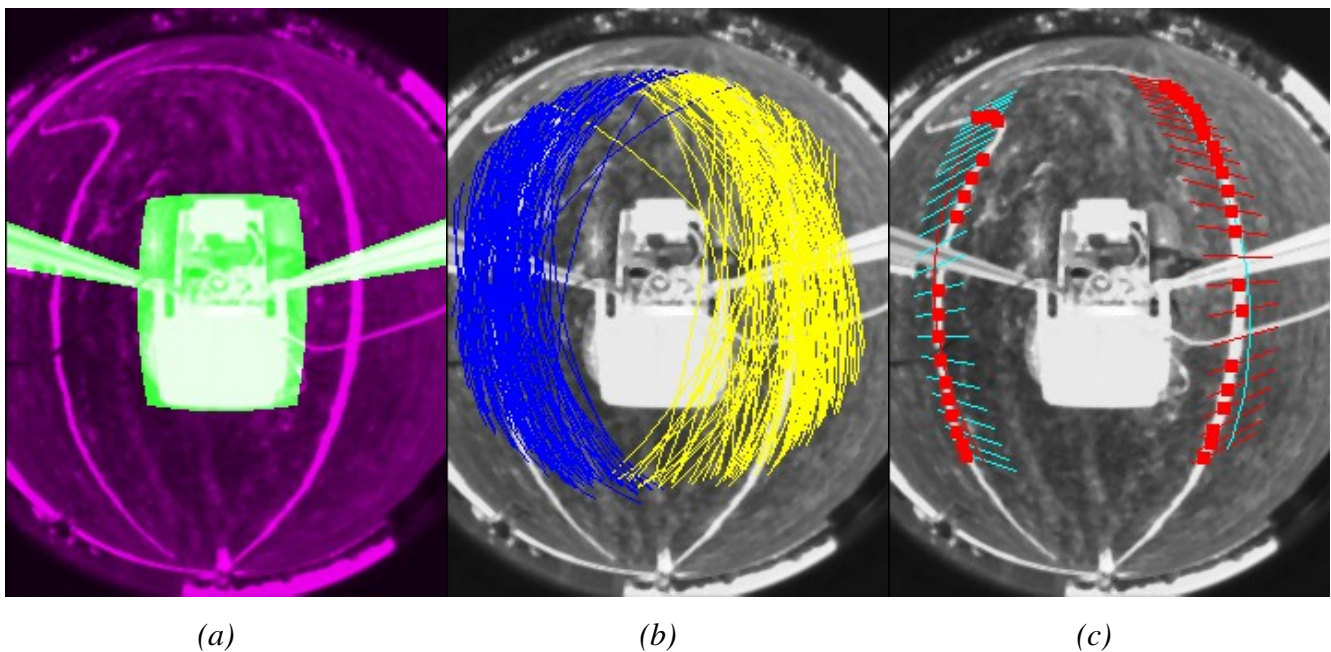


Figure 5: Steps in AC line finding. (a) Obstacle/robot mask (in green) used to exclude pixels from consideration; (b) 100 path hypotheses from particle filter; (c) State estimate from particle filter with search windows shown + line locations (red dots)

The plan pattern used by **pilot_auto** is different from that of **pilot_nav**. Rather than a fixed radial spray, an adaptive 2-D grid confined to the estimated path region is generated. There are no GPS waypoints to serve as goals, so in the default mode the nominal goal location is a “carrot” within the path some constant distance ahead (roughly 5 m) and on the centerline. In the obstacle map of Fig. 4(b) this is drawn as a red dot.

New this year is the detection of situations in which either or both of the lines is lost by the tracker. This is accomplished by counting line detections over all of the search windows of the maximum likelihood path estimate. If there are too few detections on one side, that side is counted as lost. If only one side is lost, the robot goes into *wall-following mode* where it follows the single remaining tracked line. In this mode, the plan pattern is grown outward from the tracked line to the maximum allowed path width, and the carrot goal point is set to be a fixed distance from the tracked line since there is no trusted path width estimate. If both lines are lost, the robot goes into *free mode* where the plan pattern is grown outward from an imaginary line directly in front of the robot to the maximum allowed width. This gives the robot “momentum” to effectively continue in a straight line, with obstacle avoidance, until one or both lines are reacquired.

The screenshot from Fig. 3 on the right shows the history of robot locations in a sample AC run as red circles. The differing circle radii indicate which safety radius was necessary to pass that point of the course—recall that larger radii are preferred for greater clearance, but smaller ones are chosen as necessary. Fig. 4(b) shows the robot's obstacle map and motion planner at the moment captured in Fig. 3—the robot is in left wall-following mode because the right line is almost completely blocked by a line of barrels. This is obvious in Fig. 4(c), which shows the simulated omnidirectional image for the same situation. The barrels are masked out in black and one can see that there are no line detections on the right, while there are a good number on the left.