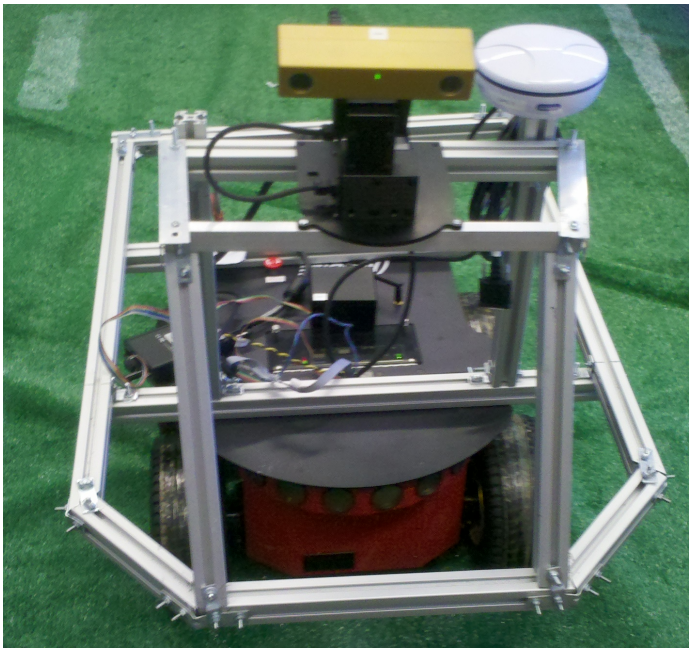
Fordham University

Intelligent Ground Vehicle Competition 2011

Mario



Stephen Fox, Ilya Naoumov, Emir Ogel, Bryan de la Rosa, Margaret Wolf, Rosa McGee, and Brendan Offer

Robotics and Computer Vision Laboratory
Fordham University
441 E. Fordham Road, Bronx, NY

Faculty Advisor Statement: I hereby certify that the engineering design of Mario was done by the current student team and has been significant and equivalent to what might be awarded in a senior design course.

Signature: _____ Date: _____

Contents

1 Team Overview

Fordham University's IGVC team is comprised of seven undergraduate students ranging from seniors to freshmen. Team members are split between two of our New York City campuses located at Lincoln Center in Manhattan and at Rose Hill in the Bronx. The Fordham Robotics and Computer Vision Laboratory (FRCV Lab) is based at the Rose Hill campus, which is approximately a forty-five minute commute from Lincoln Center. In order to allow students at both campuses to participate, we carefully organized the team in such a way that, given a sufficient grounding in a professional collaborative work flow, remote work would be both possible and efficient.

Drawing from experiences last year, we first focused on centralizing the work flow of the team in order to foster collaboration within a large group. In the fall semester, we recruited many students who volunteered to participate in the design and development of the software and hardware components of the robot. With about eighteen interested students, it was imperative that we set up a versioned code repository, ticketing system, and documentation wiki. We explored multiple possibilities, such as Sourceforge.net and other free services; we also considered hosting our own service, but ultimately we secured generous sponsorship from *Projectlocker.com* for a subversion subscription service integrated with the popular Trac ticketing and wiki system, exactly the services we had decided upon before setting out on our search for a solution. Sponsorship from Projectlocker.com cut out the overhead of setting up, maintaining and securing a central server for our collaboration.

Once the collaborative infrastructure was in place, we divided the team into several smaller groups to facilitate rapid productivity and training. The fall semester consisted primarily of training for software development techniques, the collaborative work-flow process, and software development in a Linux/UNIX environment. Our robot runs on Debian Linux, so the ability to develop software in a Linux/UNIX environment is a necessary skill for productive contribution; many of the freshmen members of the team had no prior experience working in a Linux/UNIX environment.

Furthermore, no robotics background is required to join the robotics team. In order to establish a baseline of robotics knowledge, we developed the basic framework of robotics research, motivating our contribution to the field through this competition. The team studied occupancy grids, basic mapping and localization techniques, various sensors available for robotics, and sensor fusion techniques. For the duration of the fall semester, we met twice weekly: once as a full team for training, and additionally as small groups of three for task-focused work.

The team changed significantly in the spring semester. Four students in the fall received academic credit for their workthrough the Computer and Information Science Department's Senior Projects and Internships course. These students were required to produce written reports and meet with closely with the team lead and the faculty advisor, Professor Lyons, throughout the semester. However, participation on our team is otherwise entirely voluntary, so many students who contributed in the first semester were unable to continue second semester because of graduation, paid internships and other compelling career opportunities. Our team in the spring has consisted of the seven students listed as the authors of this paper. In order to compensate for the reduction of resources in the second semester, the team lead met individually with members of the team throughout the week and held a weekly scrum meeting on Thursday afternoons. All students who participated in the second semester voluntarily produced a short written report each week, and gave presentations on current progress. At the scrum meetings, demonstrations of achievements were given to the entire team, and tasks for the following week were clearly laid out at a pace acceptable for extracurricular work. We found this model to be very productive.

2 Team Approach to Software

The skill levels of our team members is widely varied. Some are strong software engineers; others are freshmen with only intro-level programming experience. We elected not to limit participation on the team on the basis of skill, but rather ded-

ication and interest. Particularly, since this is only the second year that Fordham is particpating in the competition, we strongly believe that investing time in developing skills of novice members will help maintain continuity after senior members graduate. To account for the disparity in skill, we designed the software architecture in such a way that everyone would be able to contribute productively. We use a modular software architecture implemented in C++ using the ARIA library [**?**] as our robot meta-operating with the on-board the Debian Linux computer. A modular architecture affords us a number of advantages. We can

1. Exchange modules with the same interface to change functionality (our sensor modules follow a common interface),
2. Allocate modular tasks based on skill level,
3. Test modules individually to identify functional problems, and
4. Make development progress on individual modules or clusters of modules without being held back by incomplete ones.

We have found this architecture to be a crucial aspect of the development of our software suite, providing a framework within which novice members have been able to gradually accept more difficult tasks as their software development capabilities mature.

In addition to a modular architecture, collaboration was a key aspect of our team's development lifecycle. We utilized our subversion repository for rapid collaboration, modification, and merging of software changes. This gave our team the ability to quickly diagnose problems in each others modules. Detailed notes and log messages have also made it easy to track our progress, motivating us to change the pace of our work if at any point we fell behind.

3 Hardware Overview

3.1 Base Platform

We are using a modified Pioneer 3 - AT model robot manufactured by Adept MobileRobots (formerly known as Mobile Robots, Inc.). The platform is equipped with a 44.2368 MHz Renesas SH2 32-bit RISC microprocessor with 32K RAM and 128K FLASH memory to handle communication with many of the robot's components, including the motor controllers [1]. It is further equipped with a 1.8 GHz Pentium M processor with 512 MB of RAM, a 2.5 inch 120 GB SATA HDD, and serial ports. We found that 512 MB of RAM was insufficient, particularly for processing large sets of point cloud and image data, so we upgraded the memory to 1 GB of RAM. For remote access in the field, the machine is equipped with a wireless b/g card.

We have customized the sensors of our Pioneer 3 platform by adding a Point Grey BumbleBee 2 stereo vision camera mounted on a PTU-D46 pan-tilt unit for precise control of its movement and a broader field of view. Stereo vision provides us a sensor which returns depth, as well as color image data. We use the ARIA drivers for manipulating the pan-tilt unit, the proprietary Triclops SDK for communicating with the BumbleBee 2 camera over IEEE1394, and the OpenCV 2.1 library for manipulation of image data.

Last year, we used a SICK LMS200 laser range finder; however, the Pioneer 3 platform has a very limited supply of power (three 7.5 Ah 12 V lead-acid rechargeable batteries) relative to the energy consumed by the SICK LMS200 device. Maintaining battery life sufficient for both efficient testing, and running the course was a major difficulty last year: our battery life was less than fifteen minutes with the motors and the laser. For this reason, we have elected to use a sensor which consumes significantly less power, having fewer moving parts, providing us a consistent battery life of more than three times our previous configuration.

In order to qualify at the competition, we have had to make certain modifications to the hardware of the Pioneer 3 platform we had avaible to work with. We have made the following hardware modifications:

1. Expansion of the chassis to meet the minimum size requirements of 3 ft.x2 ft.,
2. Replacing the hard-wired kill switch with a larger, centralized kill switch for improved safety,
3. Addition of a 100 ft. wireless kill switch,
4. Tuning of the system's hardware settings to maintain the minimum average velocity of 1 MPH, and finally
5. Addition of a differential GPS unit for qualification and participation in the navigation competition.

The superstructure is made of 80/20 T-slotted aluminum [2] . We selected the 80/20 T-slotted material over cheaper generic aluminum for its stability and ease of assembly. The structure, designed by junior Bryan de la Rosa, is an essential part of our system integration plan. Primarily, the stereo camera has a very narrow field of view, and a considerable blind spot for near obstacle avoidance. In order to compensate for this, we have recessed the camera and raised it to a height suitable for safely obstacles, flags, and lane markers in a forward-facing direction. We have mounted the camera 15 cm from the front of the robot, and 75 cm from the ground plane.

We are using a Hemisphere A100 Smart Antenna GPS unit to triangulate our position and, while in motion, obtain headings toward our waypoints. Mounting the GPS unit was a problem at first: it could acquire a strong signal in a low location, but interference caused a signal lock to be unpredictable, or often lost. We determined we needed mount it in a high position, but out engineer working with our superstructure was able to attach the mount near our camera which has allowed us to get much more consistent readings. This minimizes interference, particularly from the electric motors of the chassis, and provides a clear view of the entire sky, independent of the robot's orientation. We have mounted it roughly in the center of the robot, 70 cm from the ground plane.

There is an additional space that was constructed behind the GPS unit. This space is intended to provide weatherproofing of the vehicle, storage for other electronic devices, including the wireless kill switch and the compass, and an area to mount the payload. The structure will ultimately be covered in a plastic exterior, thus weatherproofing internals of the robot and providing a professional appearance.

4 Software for Sensors

4.1 Stereo Vision

Our primary sensor for obstacle avoidance and environment modeling is the Point Grey BumbleBee 2 IEEE 1394 color camera mounted on a pan-tilt unit. The Point Grey's proprietary Triclops SDK provides a robust interface for obtaining data from the digital camera, as well as accessing and calibrating the on-board processing provided by the unit. Further image processing is done with the OpenCV 2.1 [3] library.

Using the Triclops SDK, the stereo vision pipeline first computes disparity, after which depth is computed for each pixel. Points with bad disparity are ignored, and the remaining pixels are converted to a 2.5D point cloud. Once the lane and flag detection modules have processed the data, the point cloud is converted to a PCL [4] point cloud data structure, which is then filtered using the built-in statistical removal filter, and downsampled before being input into our obstacle avoidance algorithm. We have found that downsampling greatly improves our the frequency at which we can process data.

4.1.1 Lane Following

Our first step is to mask the image for points within a 10 cm threshold of the ground plane. This allows us to run our analysis without false positives from white obstacles or the white lines on the orange construction barrels. In last year's competition, we did not use a stereo camera, but instead used a homography to map the

image plane to the ground plane; to prevent convolution of lanes with white obstacles, this required a much more complicated algorithm than this elegant integration with the point cloud.

Once we have masked the image for the ground plane (simply by nulling pixels linked to a vertical threshold above our desired one), we process the image first by filtering it through a colormask. The exact parameters of this threshold is best determined empirically on a large set of data with similar lighting conditions to those expected at the competition. For this reason, we have developed a user interface which allows us to easily fine-tune the parameters at the competition.

Next, we convert the image to grayscale, at which point we filter it through a binary threshold. The remaining pixels in the image which are not black are search with OpenCV's built-in Hough Line Detection function, parametrized to search for lines of sufficient length and to ignore noise. We have found the parametrization of this step to be particularly important, for dead grass, reeds, and other background objects on the ground can cause many false positives.

Hough Line Detection returns a compact data structure of the location of the lines in the image. At this point, we access our point cloud through the known u, v, and modify the height of the lines directly, by raising their vertical values in 3-space. Our lane following algorithm fuses the 2.5D spatial data with the standard image

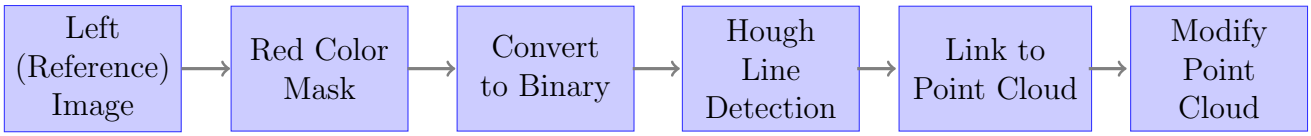
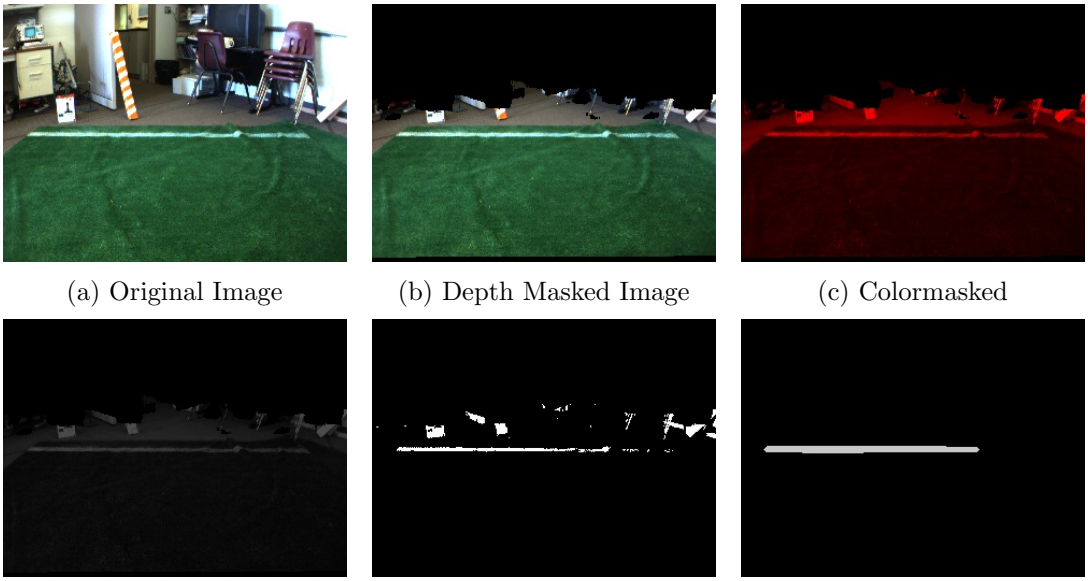


Figure 1: Lane Detection Pipeline

data. The image from the left reference camera is stored as an OpenCV IplImage* data type, allowing us to access the pixels by their u, v coordinates. In order to link this to our point cloud data type, we created an array data structure of the same dimensions as the image. In each cell, we store a pointer to our 3D points within the point cloud.



(a) Original Image (b) Depth Masked Image (c) Colormasked

(d) Grayscale Image (e) Thresholded Image (f) Hough Line-detect Image

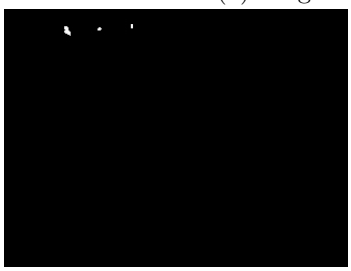
Figure 2: Lane-detection Image Pipeline

4.1.2 Flag Detection

Before the flag detection pipeline can be run to detected flags it must be calibrated to optimally filter images for the green and red flags. A vital part of our algorithm is changing the color space from RGB to YCrCb. This is a vital part of our algorithm because the YCrCb colorspace isolates the intensity values which are convoluted with the RGB values in the other colorspace. This allows for more consistent filtering in our flag detection module independent of changes in lighting. We designed a simple graphical interface using OpenCV to easily determine the range of filter of the image for Cr and Cb values. This must be done for each the green and red flags. The user interface will be used at the competiton to improve the calibration of the current module, which will be stored in the global calibration file. For the current calibration program, we used a large set of stored data from the course at the Student Ground Robotic Demonstration on the National Mall this past April [5].

For the flag detection module pipeline, the image is received from the stereo vision driver. Next, we use OpenCV to convert the color space from RGB to YCrCb. Afterwards, the image is masked appropriately for each flag by using the calibration

(a) Original Color Image



(b) Green Flag Detection

(c) Red Flag Detection

Figure 3: Flag Detection Pipeline

settings. Once the green and red flags have been located, we return pixel coordinates of rectangular bounding boxes of the flags. The Navigation module receives these pixel coordinates, which are linked to the depth map, to determine each flag's position relative to the vehicle. Once it has these relative positions, the navigation module sets virtual waypoints between sets of flags to guide its goal seeking behavior.

4.2 A100 Hemisphere GPS

The Hemisphere GPS A100 Smart Antenna was purchased through Hemisphere's Educational and Research Sponsorship Program [6]. They provided a training session that helped to familiarize us with the device. After understanding the theoretical fundamentals of GPS navigation, including differential GPS, we implemented software to integrate the GPS unit with ARIA's built-in GPS library class. However, the ARIA driver was insufficient for our needs. In order to take advantage of the on-board processing and expanded information provided by the A100 GPS unit, we designed a driver to interface with the binary messages produced by the device. The binary messages allows us to obtain more information in a single message than is

packed in the standard NMEA messages.

5 Vehicle Control Software

5.1 Vector Field Histogram

We have implemented a modified version of the Vector Field Histogram Plus [7, 8]. The original papers call for an occupancy grid; however, we have implemented a stateless, goal-seeking algorithm integrated with our 2.5D, down-sampled point cloud for fast obstacle avoidance with minimal hysteresis. Testing for our VFH module was originally done in MobileSim distributed with the ARIA SDK for the Pioneer platforms. However, the turning behavior of the actual vehicle and the simulated vehicle are quite different. We have done extensive outdoor testing to fine tune the parameters of the VFH to maintain the minimum velocity required by the competition.

5.2 Waypoint Driving

For the Navigation Challenge, we have developed a two-part strategy. We first preprocess the posted coordinates, searching for clusters of GPS waypoints. By locating areas of the field that are most populated with waypoints, the robot's software is able to concentrate on an areas where the waypoints are relatively close to one another. In the event of low power or other hardware problems, this assures that the robot is able to reach the maximum number of waypoints possible. The robot's decision to approach a particular waypoint is based on a special distance-obstacle heuristic.

The second part of our strategy focuses on the robot's performance on the field. In order to prevent the robot from frequently moving through the fence in pursuit of waypoints, we designed software that allows the robot to detect the fence in the middle of the field to create a "virtual wall" that further divides the clusters of waypoints. This will prevent the robot from crossing the wall (once detected by the

stereo vision) until all the waypoints on the fist half of the fence have been reached.

The robot's goal headings are calculated by using Haversine's formula. By taking a reading of the robot's current location and the coordinates of the waypoint, the waypoint module is able to find the heading necessary to reach the goal with respect to North. This provides an "ideal" input to our modified Vector Field Histogram local obstacle avoidance module, which determines a set of safe directions for the vehicle to move.

6 Testing, Benchmarking and Documentation

6.1 Documentation Procedures

One important feature of our software development lifecycle has been documenting the code we write. This enables modules within our software suite to be utilized for other projects with minimal overhead. Each student on the team learned our standardized Doxygen documentation practices, which produces an API reference in both LaTeXand HTML format. All internal and external functions must be documented appropriately, describing the input parameters and expected output, and each module, usually wrapped in a C++ class, is documented for expected functionality with references to algorithms used within.

6.2 Parameter Calibration Techniques

Many of our modules require unique calibration of parameters based on the characteristics of the environment. On account of frequent bad weather, we have had to test our vehicle in various environments, particularly mock courses set up indoors during the winter time. To facilitate the tuning of parameters, we have developed graphical user interfaces to quickly define new parameters for our software which can be input into a global configuration file loaded at run-time. This prevents us from having to recompile our software while testing or tuning the system, saving us valuable time and battery life while testing in the field.

Using the robot's on-board wireless card, the vehicle can broadcast an *ad-hoc* connection, or connect to a wireless access point. We establish a connection to the robot through either a remote SSH connection from a laptop, or through a TCP client-server connection.

In addition to extensive field testing, we have designed a miniature mock course for basic testing inside our laboratory. This was necessary in order to maintain the pace of development throughout the winter, where we received record amounts of snow in NYC. We purchased synthetic outdoor carpet grass from Home Depot, on which we painted broken and solid white lines, and created our own construction barrels and horses.

Furthermore, we utilized MobileSim, software distributed with the ARIA library that is built on an older version of the Stage simulator. However, while this helped to test the dynamics of our obstacle avoidance system, simulation is less useful for stereo vision than sonar and laser range finding sensors.

Special Thanks

Special thanks to our faculty advisor, Professor Damian M. Lyons, in the Computer and Information Science Department at Fordham University. He generously volunteered his time outside of class to work individually with every member of the team. We would also like to thank the Fordham University Deans for sponsoring our travel, and continually encouring our work in the Fordham Robotics and Computer Vision Laboratory.

References

[1] Mobile Robots Inc., 19 Columbia Drive, Amherst, NH 03031, *Pioneer 3 Operations Manual*, version 5 ed., July 2007.

[2] "80/20® Inc." http://www.8020.net.

[3] "Opencv (open source computer vision) library." http://opencv.willowgarage.com/wiki/.

[4] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[5] "Auvsi foundation: Student ground robotics demonstration on the national mall." http://www.auvsifoundation.org/dcdemo.

[6] "Hemisphere GPS." http://www.hemispheregps.com.

[7] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 7, pp. 278–288, August 2002.

[8] I. Ulrich and J. Borenstein, "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," pp. 1572–1577, 1998.