Project Andros:
Engineering an Autonomous System for a Commercial Explosive Ordinance Disposal Robot

Tennessee Technological University
Autonomous Robotics Club [1]
IGVC 2011

[1]I, Stephen Canfield, sign that this project consisted in significant engineering design effort:

Contents

1 Introduction

The Autnomous Robotics Club (ARC) of Tennessee Technological University, with the support of Remotec, has had the pleasure of experimenting with the concept of a bolt-on module to give an Explosive Ordinance Disposal (EOD) Robot autonomous capabilities. When it comes to Explosive Ordinance Disposal tasks, robots save lives. All around the world, robots are being used to allow humans to work safely when performing the very dangerous task of bomb investigation and disposal. These EOD robots have become very popular not only with the armed forces, but with domestic police as well. If they were autonomous, and given the precision and expendability of computers, the safety of our service men and women could be greatly increased. In previous years we competed with the Mini-Andros II platform from Remotec, but at the 2010 IGVC, it suffered from a catastrophic hardware failure. When we talked to Remotec about repairs, were were instead gifted with the more modern Andros HD1. Not only has this given us a chance to prove the bolt-on nature of our system, but it has given us a chance to refine out systems to a new level.

2 Motivation and Goals

The project focus for the 2011 TTU ARC was to reconfigure our robot sensors and control system for the Andros HD1 System. The design report details the significant changes made in the transition process to the new platform as well as new designs and systems. It will, for completeness, also include systems that were only slightly changed or left the same in order to facilitate a comprehensive description of the robot.

The majority of changes needed for this transition process were hardware related. Our software was the primary focus for the 2010 IGVC as the old platform's hardware components had matured in previous year.

Our primary focus with Project Andros continues to be the production of a small form factor based "bolt-on autonomy" for EOD robots. The popularity of these small systems such as those found netbooks and tablets, is a testament to their durability and energy efficiency.

3 Design Process and Team Organization

Since ARC exists purely as an extracurricular and volunteer-based club at Tennessee Tech, we organized our group dynamics pragmatically, focusing our time commitments on a single workday once a week and two shorter weekly organizational meetings. By addressing the unique composition of members and time constraints directly, we designed our team structure such that work-flow was minimally disrupted by member scheduling conflicts.

3.1 Design Process

Building on our past experience, when approaching a design task the group starts a brainstorming session, and the meeting moderator, typically the club president, writes the important points on a white board. In fact, the team meets once a week outside the lab, in a classroom to facilitate this type of planning and discussion. After some possible solutions are identified and discussed, one or more members of the group, based fundamentally on skills and experience, volunteers to be responsible for the task and begins to work in small steps towards its completion. From previous experience, the group identified that completing small steps, and therefore gradually increasing complexity, was a much better approach than a fully-planned, complex solution. Upon completion of a task, testing and analysis revealed areas for improvement, and the task was revisited at the white board. In this manner, the design process allowed for development of a solid foundation from which to develop our more complicated algorithms and design elements.

3.2 Team Composition and Organization

The students involved with the ARC Andros Project are detailed in Table 1.

Table 1: An interdiscplinary club of undergraduates and graduate students.

Name	Major	Level	Hours
Ben Martin	Electrical Engineering	Undergraduate	100
Gerrit Coetzee	Mechanical Engineering	Undergraduate	100
Edward Tidwell	Mechanical Engineering	Undergraduate	100
Phillip Adams	Mechanical Engineering	Undergraduate	100
Dylan Jennings	Mechanical Engineering	Undergraduate	50
Ethan Jones	Mechanical Engineering	Undergraduate	40
Marbin Pazos-Revilla	Computer Science	Graduate	30
Tristan Hill	Mechanical Engineering	Graduate	30

At a broad overview level, our members fell within four major groups:

- Hardware and Electrical — Responsible for mounting sensors to the Mini–Andros chassis, repairing failed components, and power distribution aspects.
- Sensor Interfacing — Developed drivers for the various sensors used on the robot.
- Algorithms — Wrote navigation logic for avoiding obstacles, staying in between lines, and reaching waypoints.
- Systems Integration — Responsible for the "glue" which allowed each of the individual groups to work together.

3.3 Resource Sharing and Communication

Towards the goal of making the club sustainable, we utilized an online wiki forum for sharing information. In this way, members who missed important meetings or discussions could catch up as his/her schedule allowed. The forums also provided a key factor for the club's future success: documentation of progress, successes, and failures.

In addition to the documentation, we used a Subversion content management system for code sharing. This code repository proved to be a great source for members to review the latest revisions to code and quickly identify changes.

4 Mechanical System

The chassis used by the ARC is a functioning Andros HD1 from Remotec, a subsidiary of Northrup Grumman. This chassis is still being used today and is quite robust. It was donated to us as a replacement for our previous chassis the Mini Andros II, which suffered from a critical hardware failure at the 2010 IGVC.

The Andros HD1 is a ruggedized, weather-resistant EOD robot built to survive in intense environments. Like other EOD-purposed robots, Andros is a skid-steer, open-loop system. All feedback comes from the sensor array: vision, laser ranging, compass, and GPS.

The frame of the Andros HD1 features a track-wheel hybrid skid steer system that is capable of climbing difficult inclines with its attached articulators. Positioned at the rear of the robot, an arm attached to the chassis features a manipulator which would be used for the actual disarming of the explosive device. The computer housing is attached to the provided accessory mount that comes standard with the robot, remaining in compliance with our "bolt-on" philosophy.

All of the motors and drivers are contained within the robot chassis itself. No changes have been made to the Andros HD1 in this area. It is also important to note that, as much as possible, ARC has *maintained the capability of the robot to function as an EOD robot*, placing components so that they minimally interfere with the robot's previous capabilities.

4.1 Computer Box

In the previous versions of the system we often suffered from failure due to unprotected electronics. A computer system that isn't properly mounted and protected can easily vibrate itself apart. That's why we dedicated time this year to designing an ideal electronics housing (Figure 4.1) for our purposes. The entire electronics system can be removed without having to unbolt the outer housing. This allows for easy maintenance. The box also protects from rain, and provides cooling for the computer. Although tailored slightly for the Andros HD1, this box only needs two bolts to attach to a system, making it easy to transfer electronics to another platform if need be.

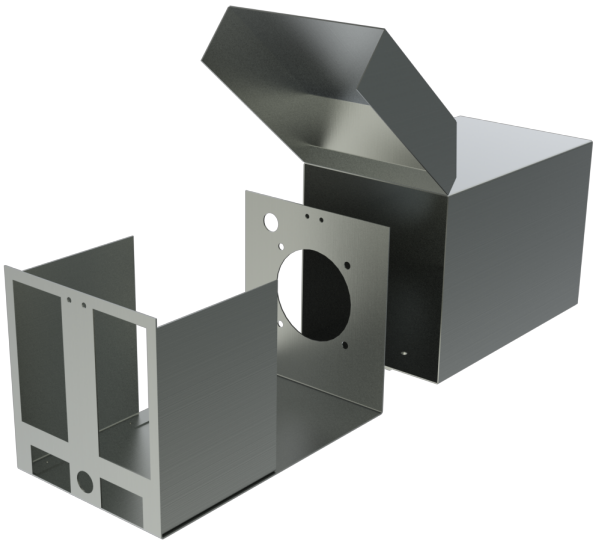


Figure 1: A 3d rendering of the computer box.

5 Sensors

Andros's sensor suite consists of a laser rangefinder, two cameras, a GPS, and an Inertia Measurement Unit (IMU). The sensors are controlled via drivers developed by the ARC, and are utilized by the navigation algorithms described in the following sections.

Many of our sensors have stayed the same from the robot's previous incarnation, but we have taken the opportunity to improve the mounting systems for them as well as improving our GPS capabilites.

5.1 Inertia Measurement Unit

We purchased an Attitude Heading Reference System (AHRS) 3DM-GX1 as our IMU. The 3DM-GX1® includes in a single package three angular rate gyros with three orthogonal DC accelerometers, three orthogonal magnetometers, multiplexer, 16 bit A/D converter, and embedded microcontroller. This unit operates over the full 360 degrees of angular motion on all three axes, providing orientation in matrix, quaternion and Euler formats at update rates of 350 Hz. Output modes and software filter parameters are user programmable. Programmed parameters and calibration data are stored in nonvolatile memory. The IMU is pictured in Figure 5.1.

The IMU was previously mounted slightly in harms way due to the limited space on the Mini-Andros II, but with a clever computer box design, this instrument is now fully protected from the elements.

Figure 2: The 3DM-GX1 IMU. Programs are stored in non-volatile flash memory and communication with the host computer is via USB

5.2 Hokuyo URG-04LX-UG01 Laser Rangefinder

The LIDAR used by Andros is the Hokuyo URG-04LX-UG01. The main function of this sensor is physical obstacle detection. The URG-04LX-UG01 pictured in Figure 3 is a scanning laser range finder with a sensing range to 5.6 meters. Measurement accuracy is within 3 percent tolerance of the current reading for most of the sensor's range. Scanning rate is 100 milliseconds across a 240 degree range. Sensor interface is simplified with a Mini-B USB connector that provides both power and sensor data communication.

At the 2010 IGVC we discovered that due to the mechanical design of the Hokuyo it is very susceptible to direct sunlight. Through our own research we discovered that the laser module sits at the top of the range-finder; when the sun shines directly on the device it causes a greenhouse effect inside forcing the laser diode into thermal safety shutdown. This has lead to the development of a shielded, adjustable mount that attaches simply with either magnets or Velcro to the body of the robot. It offers protection from the sun, impact, and rain.

Figure 3: The Hokuyo RG-04LX-UG01.

5.3 Cameras

The camera can at once function as a range-finder, color sensor, and photoelectric sensor. Coupled with a suitably powerful computer and vision software, the camera can also perform real-time object detection, line and edge detection, and motion sensing. The robot chassis comes with three built-in cameras. There is a drive camera mounted on the front level with the ground, however we decided not to use this camera in our design. The main camera is attached to the arm, which we use to

view the robot's "toes". This camera can zoom as well as change the aperture and focus, and is also the highest camera available. The cameras output a standard composite image so we used a video capture card to capture the video stream and digitize it.

5.4 Global Positioning System

Figure 4: Ag Leader GPS

Andros's GPS sensor is an Ag Leader GPS 1500 4. This GPS is a smart antenna that tracks GPS and SBAS(WAAS and EGNOS) signals allowing for sub-meter accuracy. The GPS automatically finds and connects these signals outputting as standard NMEA string at 1Hz through serial.

It's a big improvement over the previous system, which was a Trimble Survey Computer. The Trimble required excessive configuration before it would output a signal, and also required an awkward mounting situation since it was meant to be held by human hands when in use. The new GPS is designed for agricultural services, so it is easily mountable and near impervious to any normal operating conditions.

6 Electronics and Computer Systems

6.1 Emergency Stop

Considering the new requirements for IGVC 2011, our E-stop has been reworked and thesystem guarantees that the motion of the robot comes to a complete halt. We do this by means of one robust DP/DT (double pole/double throw) relay with high current capacity contacts and an emergency stop toggle button. The E-stop switch is enclosed in a red box and located in the rear part of the robot, providing easy access and visibility. The relay and wireless E-stop components are located inside the computer box. Additionally, when no commands have been sent to the microcontroller for more than 40 milliseconds, an automatic brake engages and the robot's wheels lock.

The Wireless E-stop is connected in series with the E-stop system, and it includes a wireless receiver connected to the relay. By means of a key fob transmitter, the Wireless E-stop can be activated, opening up the circuit that feeds power to the Andros motors and causing the brakes to catch.

The safety light detects whether power is supplied to the robot or not by using one of the digital input pins and a voltage divider on an Arduino microcontroller. When the pin goes low it turns off the light, when it goes high it turns it on. It begins blinking to signal autonomous mode when serial commands are sent to in in 5 second intervals. If it doesn't see a pulse after five seconds or it sees an end command from the computer it stops blinking.

6.2 Main Computer

The computer selected as the main processor for our system is the Intel D945GCLF2 Mini-ITX Motherboard, pictured in Figure 5.

Mini-ITX is a standard specification for motherboards designed for embedded applications. With the advent of the Intel Atom, the Mini-ITX market beginning to grow rapidly. This particular

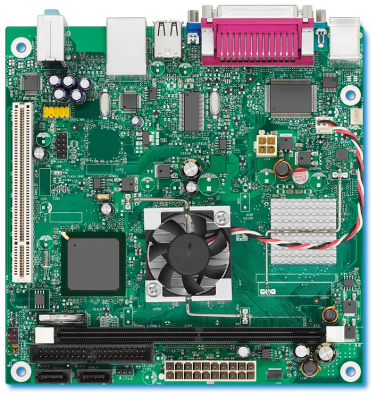


Figure 5: The Intel D945GCLF2 Mini-ITX Motherboard: 17x17cm and four logical compute cores.

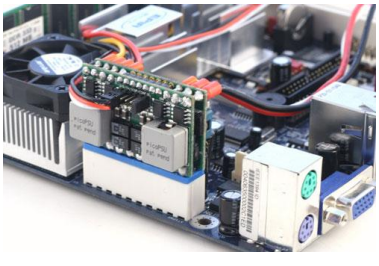
motherboard houses the Intel Atom 330 CPU, a dual-core 1.6 GHz chip with hyper-threading and 1 MB L2 cache. Dual-core with hyper-threading means that the Atom 330 has four logical cores. Multiple cores are extremely well-suited for robotics applications, since each computation in the sensory and decision chain can be decoupled into threads and run asynchronously on each core. Furthermore, since the Atom is x86-based, it is able to run the full Linux operating system to take advantage of the rich computer vision libraries available for free (openCV). The Intel Atom, in addition to its small Mini-ITX form factor (17x17 cm), is extremely low-power, with the entire system, including peripherals, typically consuming less than a 60W bulb at full utilization. The low-power consumption makes the board an ideal candidate for DC-to-DC converters designed for embedded computer applications. The D945GCLF2 is also extremely cost-effective, costing less than $90 for a motherboard with a soldered Atom CPU and on board video. This system has worked very well for us and we have yet to outgrow its capabilites.

6.3 Power and Battery Systems

The robot itself contains all the motion controls as well as camera decoders. All the external systems run off their own batteries. This way the system is more easily “bolted on" it also separates the high noise, high power draw components from the low noise low power draw components. The batteries that supply power to the Intel Atom 330 motherboard and sensors are two 12V NiMH batteries running in parallel for 10 Amp Hours of power. They were custom made for our purposes. These will run the computer and sensors for approximately two hours.

We chose to purchase a new battery system due to the aging nature of our Sealed Lead Acid batteries and the need to save space and weight.

(a) An example 12V NiMH battery.



(b) Pico-PSU

Figure 6: Battery and power supply for Atom system.

We selected a M3-ATX Pico–PSU power supply to power our main motherboard from the 12V attachable battery. The Pico–PSU attaches to the motherboard’s ATX form factor. The Pico–PSU

is designed to handle abrupt power fluctuations without damaging the vulnerable motherboard.

7 Software Strategy

7.1 General Software Philosophy

Since we know that our club operates as a revolving door for students interested in robotics at TTU, we realize that our work must be simple, robust, and useful enough to be used and assimilated by future students wishing to gain a foothold in the area of autonomous robotics. Thus, documentation and good coding practices are crucial. Additionally, since software architecture choices strongly impact the modularity of our systems, we found it wise to use a heavily modular and object-oriented approach to our software design.

To achieve modularity, we adopted two primary models: object-oriented design and distributed processing. Each sensor operates in its own thread, aggregating data into a "mailbox" without being aware of any parent process. The processes that tap into any sensor's data simply pull from the mailbox the most recent data and process it. In this way, data aggregation through sensors can be done completely asynchronously and in parallel. Furthermore, this paradigm simplifies the code since the sensor itself need not be aware of the parent retrieving the data. Refining the idea further, each sensor is made into a class, and inheritance is heavily used between similar sensor classes to reuse code and centralize changes to similar sensor interfaces. Finally, each sensor package is threaded with an event-loop that waits for an exit signal, providing a graceful exit to all processes from a central authority.

The final goal is toward an interface simple enough that a curious freshman could pick up and use in a single day to accomplish simple demonstrations of autonomy and sensor aggregation. Doing this serves a dual purpose as well: if the system is simple enough so as to be used by novices, then certainly it must be mature enough a robust and simple strap-on solution for autonomy.

7.2 Programming Environment

In deciding the software environment for our robot, we considered several factors, including ease of use, learning curve, flexibility, cost, and speed. Ultimately, we chose to develop most of the application logic in Python and the vision processing in C with OpenCV. This combination offers the simplicity of Python with the speed of C. All of this was done in the Linux operating system. There are several strong tools that come with Python. Python, through the *pyserial* package, exposes an easy-to-learn interface for driving sensors that rely on serial communication. Python also provides a matlab-like plotting interface through *matplotlib* and rich scientific libraries through *numPy* and *sciPy*. Finally, the *struct* package makes packing bitstreams for use in providing JAUS compatibility very easy.

OpenCV is a collection of open-source computer vision algorithms. Originally developed by Intel, OpenCV is now actively maintained by the open source community. We use OpenCV functions for lane and obstacle detection. Early on, OpenCV was been tested using the Atom platform and was found to be able to perform satisfactorily and in real-time, at more than 10 frames per second.

7.3 Remote Debugging

Since Andros is head–less (meaning that we do not have a monitor attached to it) we use NeatX to remotely control the robot over the network. SSH is used in cases where only textual output is desired (GPS testing) and NeatX is used in cases where visual feedback is needed (camera testing). We chose to use sockets for our inter-process communication as a deliberate design choice serving two goals: the first, to facilitate remote control and remote debugging over the network, and the second, to provide an easier route to full JAUS compatibility.

8 Systems Integration

The overarching software architecture is shown below in Figure 7. Each gray box represents a separate process. Each green arrow represents I/O. Blue boxes are Python objects, and blue boxes with white outlines are threaded separately. Black arrows represent inter-process communication, which is done by sockets. Purple arrows represent object inheritance.

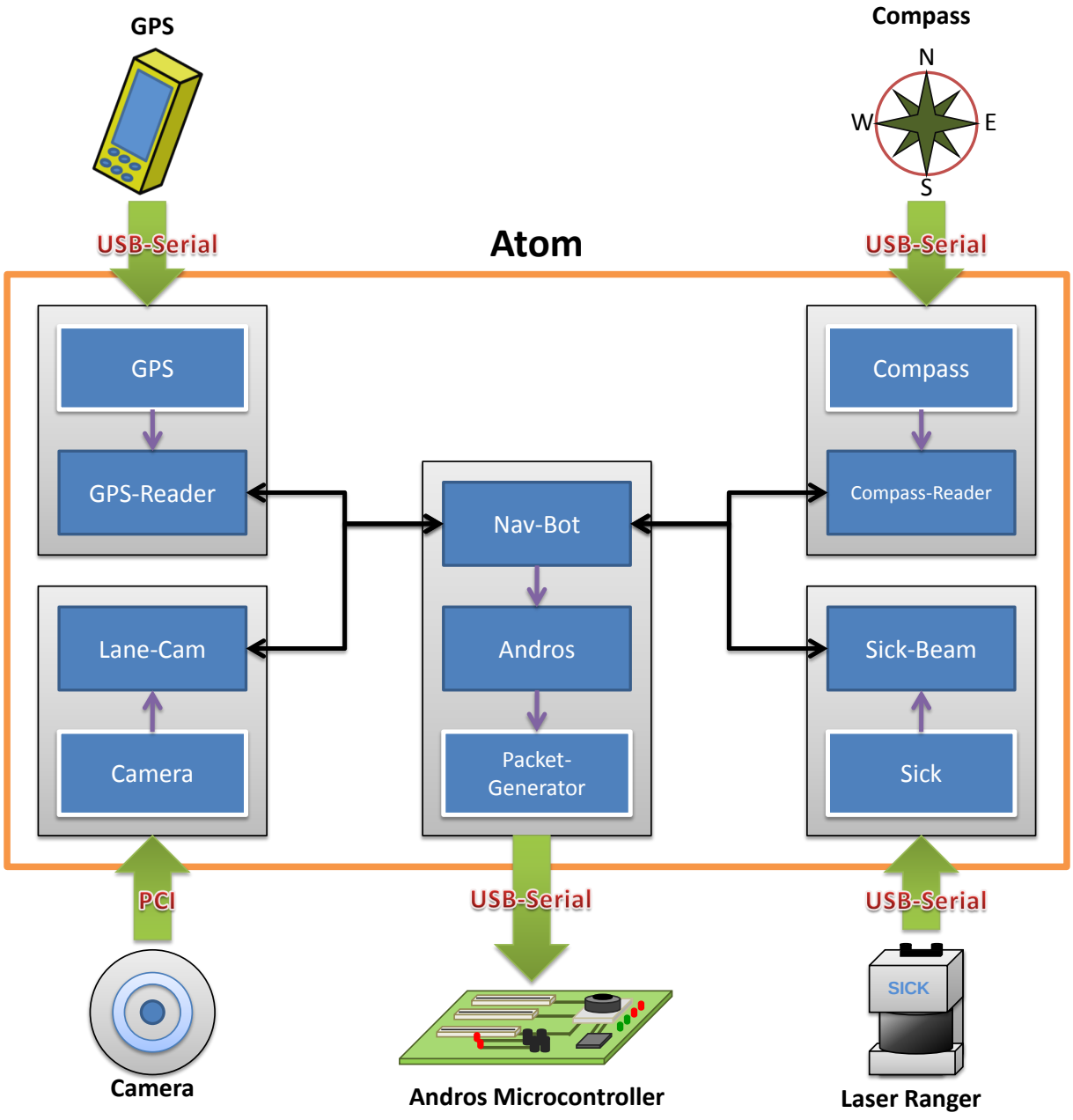


Figure 7: The software architecture: Each green arrow represents I/O. Blue boxes are Python objects, and blue boxes with white outlines are threaded separately. Black arrows represent inter-process communication, which is done by sockets. Purple arrows represent object inheritance.

As can be seen in Figure 7, there are four main sensor processes and one navigation process that execute on the Atom. All I/O is done via USB-to-Serial adapters, with the exception of the

camera system which is fed through a PCI card.

8.1 Vision

From our past experiences, vision is the trickiest sensor to utilize properly. Lighting, glare, and motion blur seek to destroy any hope of gleaning useful information that the robot can use to guide itself. Of course, it is paramount that the robot stay within lines. Thus, a given line detection scheme should be as close to 99% sensitivity as possible. Usually, maximizing sensitivity comes at the cost of specificity. That is, we can trivially "identify" all white lines by simply writing a predictor that disregards its input and always "sees" all possible white lines. Thus, the true positive rate and the false positive rate are at odds. We don't want a robot running away from phantom white lines, but we certainly can't have a robot cross a real white line. Furthermore, given that the state-of-the-art vision algorithms for robustly identifying even simple objects in arbitrary orientations, scale, and positions are still lacking, simple white line detection on an arbitrary IGVC course becomes a significantly hard problem. This is in contrast to the DARPA challenge, where white lines on roads are guaranteed to be unobscured with shallow curves. Often, in IGVC, the robot can barely see in front of itself: a switchback, for instance, almost completely obscures the robot's perceptions of what comes after it.

With these challenges, comes several important restrictions to any algorithm: it must be tolerant to lighting changes (overcast, sunny, raining), motion blur (uneven terrain), glare (reflectors, shiny colored paint), and should not depend on seeing far away (switchbacks, object occlusion)

Loosely speaking, the robot should look close to itself for oblong, moderate-sized connected regions of bright, near-white color. Thus, our proposed algorithm must find the white colors in the image (white filter), filter out glare from the tips of grass and white from pieces of hay (blob analysis), and finally detect "line-like" pieces (Hough transform).

8.1.1 White Filter

Creating a white filter is quite challenging due to the fact that "white" is not always white. This unfortunate fact comes from the internal mechanics of cameras themselves, which also don't know true white and must guess this color using a technique called (you guessed it) white balancing. Improper white balancing may result in a "blue haze" over the entire image, for instance, when photographs of the beach, ocean, and sky are taken. Thus, a white filter must account for the possibility that a "haze" might be present in an image. Our technique, which we find quite robust, is one of color segmentation using 3-dimensional k-means clustering.

K-means clustering is a simple algorithm which seeks creates random clusters of points, calculates the distances from the centroid of a cluster to all its constituent points, and then tries to minimize this distance iteratively. In our case, we treat each pixel as a point in 3-dimensional Cartesian space, with (x,y,z) equal to (R,G,B). We can think of the entire 8-bit RGB color space as a 3-dimensional cube where black (0,0,0) is in one corner and white (255,255,255) is in the opposite corner. Like colors will be "close" via Euclidean distance, so we can appropriately apply k-means clustering as a technique to find dominant colors in an image. We chose RGB space instead of HSV (Hue Saturation Value), because finding the "whitest" cluster in RGB space is as simple as finding the largest R+G+B sum. We also found that, in practice, k=5 suffices to capture white (and white only) as the lightest cluster. The lightest cluster is the cluster whose centroid is furthest in the "white corner" of the RGB cube. The figure below shows a sample image where the 5 cluster centroids are shown in the top left of the image, using RGB clustering.

The benefit of color clustering over traditional threshold-based methods should be clear. Thresholds, such as (R $>$ 200 AND B $>$ 200), for instance, are very dependent on the current lighting conditions. This threshold (200) could change significantly if the white line is in the shadow of a barrel or the amount of light from the sun. With color segmentation, we only need to guarantee that the lines are of uniform color and that they are very light compared to the other scenery. The figure below shows the output of the white filter from the picture given above.

Figure 8: The 5 color clusters from 5-means clustering are shown in the top right. Note that the white of the lines is the dominant cluster.

Figure 9: The figure from above turned into a binary image through thresholding.

8.1.2 Blob Analysis

The cluster-based white filter does a great job at capturing white under varying light conditions, but it doesn't provide a way to filter out other types of white matter that aren't part of lines. Hay, glare, and white barrels contribute to the white cluster as well and will be captured by the white filter. Thus, we need a way to distinguish from the tips of grass catching the sun and white paint on grass. We can do this by looking contouring the binary image output from the white filter, and removing any contours that produce an inner area less than a small amount. In effect, this action will remove small "blobs" of white, while keeping the large "blobs." These blobs are fully connected regions. A line segment will thus look like a large (perhaps skewed) rectangle, while glare will look like small patches. By using contours, we end up with edges, similar to a Canny transform. This effect will help us for our final step, the Hough transform.

8.1.3 Hough Transform

We employ the Hough transform to look for line segments. We thus can single out the oblong blobs because their long contours provide us with two lines each, of identical slope. As Hough is normally used in conjunction with Canny, the contouring of our image serves as a suitable replacement for this first step. Finally, to characterize our findings, we simply find the average angle of line by treating our Hough lines as vectors and using an optimized arctangent estimate. The final output

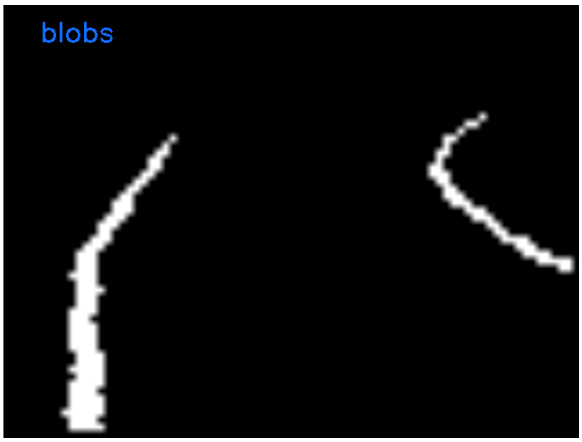


Figure 10: The figure from above with small blobs filtered out.

is then either: NULL, or <angle>, depending on whether or not lines were found and what the average angle is, respectively.

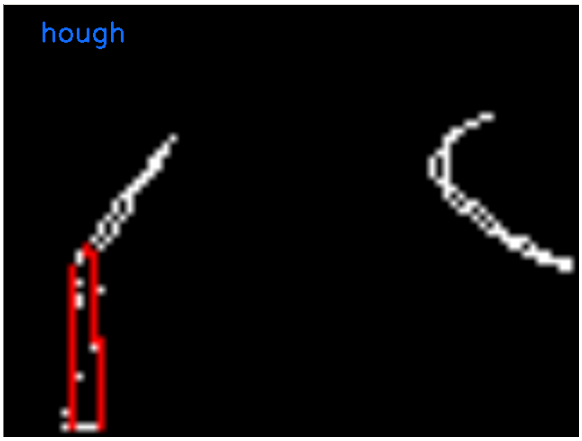


Figure 11: Applying a hough transform to detect lines.

8.2 GPS–Reader

The Ag Leader system is configured to continually output GPS information across a serial connection at an update rate of 1 Hz. The information is transmitted using NMEA Standard output messages and is decoded by the GPS–Reader Python driver. The driver makes simple functions available to the main algorithms to get the current position, convert to distances from GPS coordinates, and supply accuracy and correction data. The GPS driver also has internal functions to calibrate the conversion constants based on an experimental in-field calibration process. And finally, the system sends appropriate information to alert the algorithm if the GPS signal drops.

8.3 Navigation

Our general algorithm is deliberately the simplest possible algorithm we could devise that would consistently conquer a switchback. Our algorithm has three action states: seek, hug, and leave. We have noticed that tight spaces generally trip up robots at IGVC, so need to design our algorithm around these tight spaces. In other words, instead of writing an algorithm that seeks to get as far away from obstacles as possible, we would instead like to "hug" them, tracking their perimeter

closely. By hugging, we are able to naviggate tight spaces that a more timid algorithm might shy away from. Additionally, many robots either get stuck in corners, or turn around haphazardly and end up going the wrong way. We would like to counter this by explicitly detecting a "stuck" situation and performing the leave action. Furthermore, we have set aside steps to ensure that we never leave in the wrong direction by judiciously using GPS logs generated during the course of the run.

8.3.1 Seek

Given a waypoint, we have an extremely simple algorithm for seeking toward it. We turn in place until we are facing directly toward it, and then we head straight for it. During the seek phase, we check our camera to detect an imminent collision with a line, and we do the same with our LIDAR data for objects. If we detect imminent collision with a line, we note the slope perpendicular to this line,and construct a new line crossing through the center of the robot with the perpendicular slope. We count all past GPS points since the start of the run on either side of the line, and then turn away from the line in the direction of the least explores bisection. This simple rule ensures we never go back to the finish line. Another temporary waypoint is created in this way. If we detect imminent collision with an object, we go to the second action state, hug, via the "attach" transition. If at any point, the attach transition would result in crossing a line or the act of avoiding a line would result in colliding with an object, we perform the leave action.

8.3.2 Hug

Hugging is essentially trying to keep something an equal distance away. For the robot, we use an occupancy grid and measure distance in grid spaces. The discretization of space enables extremely simple algorithms. We simply check the neighboring cells of the robot, and turn or go straight to keep the obstacle at a constant distance to the robot's side.

our decision rule for going back to the seek state (detaching) is simply to look at the difference in robot orientation through time. We record the initial orientation using our compass and detach from the object when we have turned over 90 degrees. This step is necessary because otherwise, for example, we would track a single barrel in circles for ever.

8.3.3 Leave

The are times when the robot will make the wrong decision and end up in a bad place. This situation can happen when the robot wedges itself into a corner made from a line and a barrel. In this case, we identify when we cannot seek or hug, and we just turn 180 degrees and temporarily seek straight ahead.

8.3.4 Overall

We find that this simple algorithm has the potential to keep the robot from getting stuck, or going backwards in the course. We hope that switchback problems this IGVC will be a thing of the past.

8.3.5 GPS

For the Waypoint challenge, we also rely on the GPS and compass to give heading values as well. We apply a "stop and turn" strategy for making waypoints, turning in place and traveling in straight lines to reach waypoints. If the Hokuyo detects imminent collisions, this takes precedence and the GPS driver relinquishes control until the object is out of harm's reach.

Table 2: Estimated and Actual Costs.

Item	Estimated Cost	Cost to ARC
Remotec Andros HD1	$50,000	$0
Intel Atom Computer	$85	$85
RAM + Hard Drive + PicoPSU	$140	$90
Hokuyo	$2500	$2300
Ag Leader GPS 1500	$1000	$1000
IMU	$1500	$1500
Computer Electronics Box	$500	$500
Miscellaneous components	$250	$250
TOTAL	$55975	$5725

9 Cost Table

10 Predicted Performance

The Andros is software limited to approximately 3mph, though the hardware is capable of much faster speeds. We have climbed inclines of over 40 degrees and up stairs. The motor battery life has been shown by field testing to be about 3 hours of constant movement. The same is true of the Intel Atom powered by the 10amp hour NiMH battery system. Our waypoint accuracy is governed by the GPS accuracy, which is 1-2 meters. Our camera can detect objects up to 20 feet away, and the laser ranger can detect objects up to 5 meters away. The reaction times of the Andros are a little more complicated, being dependent on the current state of the sensor aggregation and navigation logic. In general, the cameras and Hokuyo Laser Ranger operate at 10 Hz/FPS, and the GPS at 1 Hz. Thus, depending on the sensor taking precedence, the reaction of the robot can vary from around 0.1 second to 1 second. We find this performance to be satisfactory.

11 Conclusion

We, the Autonomous Robotics Club of Tennessee Tech University are very proud of our accomplishments this year. We have designed a working prototype for a commercially producible "bolt on" EOD automation system. This system has been successfully transferred from our old Andros platform, Mini Andros II, to our new one, Andros HD1; thus proving its "bolt on" nature. The previous incarnation of this system demonstrated its capabilities as an autonomous platform in IGVC 2009, where we placed 12th in the Grand Challenge classification, and we are confident it will perform well in the 2011 competition.

The system we designed for the Andros HD1 required no direct modification of the robot, aside from the addition of an e-stop for IGVC safety purposes. Not only that, we are certain that our system could be adapted to any EOD robot of similar sensor and drive configuration.