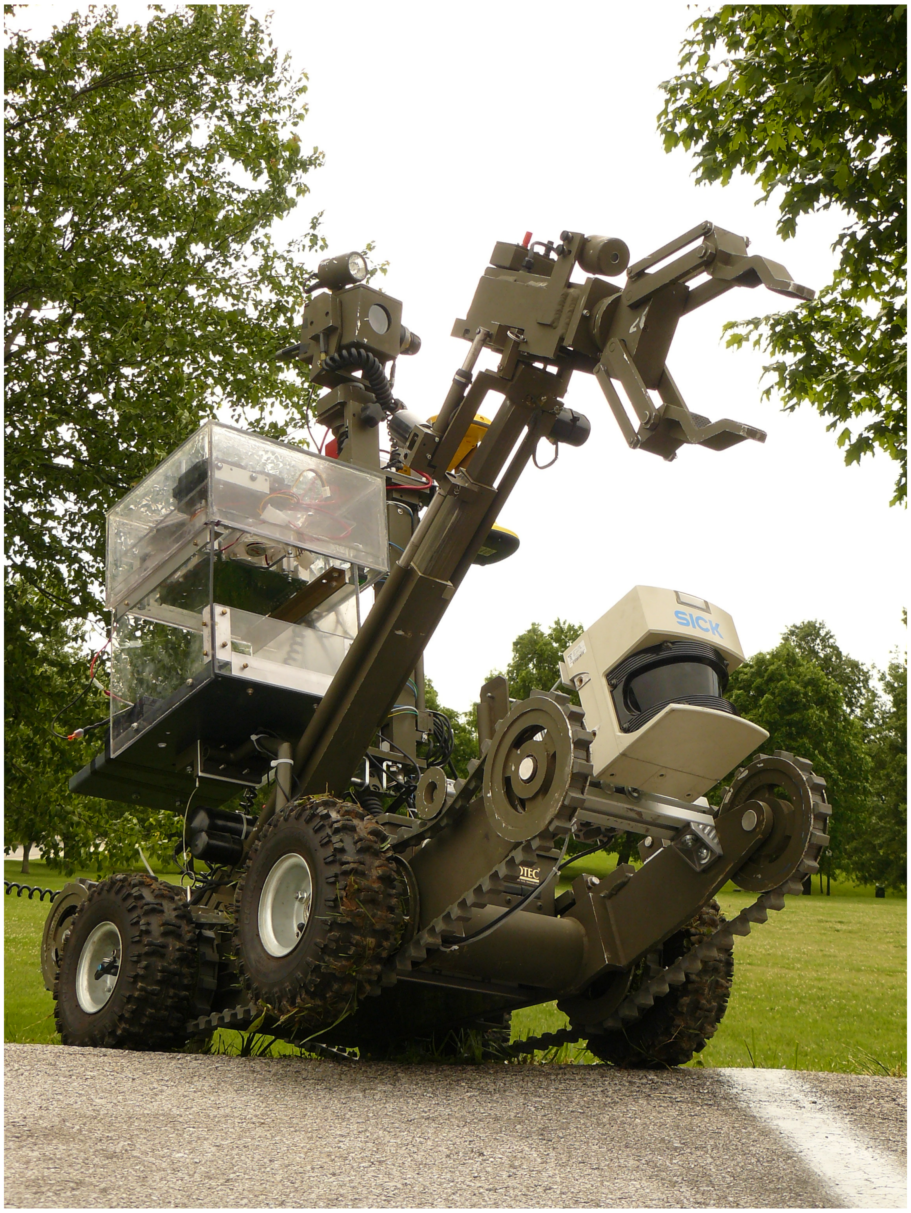
Project Andros: Engineering an Autonomous System for a Commercial Explosive Ordinance Disposal Robot

Tennessee Technological University
Autonomous Robotics Club [1]
IGVC 2009



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

Contents

1 Introduction

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. The Autonomous Robotics Club (ARC) of Tennessee Tech presents in this report a sensor and computing system designed to attach to an existing EOD robot and provide it with autonomous navigation ability. The ARC proudly presents its entry to the 2009 Intelligent Ground Vehicle Competition: Project Andros.

2 Motivation and Goals

With the rising popularity of small form–factor computer systems and "net–tops" such as the *Eee PC* and *Acer Aspire One*, there has been a surge of high–quality Original Electronics Manufacturer (OEM) parts that can be utilized for the assembly of small, cheap PC's. In addition to being smaller, these mini–systems are more energy efficient and often more economical. The combination of high energy efficiency, reasonable speed, and low price offer great benefits to the robotics world. Instead of microcontroller–only platforms which restrict software complexity or full ATX systems which restrict mobility and scalability, systems based on small form–factor PC's offer the balance of having a rich development environment, powerful computational capabilities, and relatively low power concerns. The emergence of such "computationally dense" devices helps motivate our goal to provide a small, strap–on "black–box," providing autonomy for a JAUS–compatible robot. As such, our goals are strongly aligned with those of the IGVC 2009 competition.

3 Design Process and Team Organization

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

3.1 Design Process

The design process followed by ARC this year was one of "Continual Improvement." A graphical representation is pictured in Figure 1.

When approaching a design task, the starting point was on a whiteboard, in a brainstorming session. In fact, the team met once a week, outside the lab, in a classroom to facilitate this type of planning. After some possible solutions were identified and discussed, one or more members of the group volunteered to be responsible for the task and began to work in small steps towards its

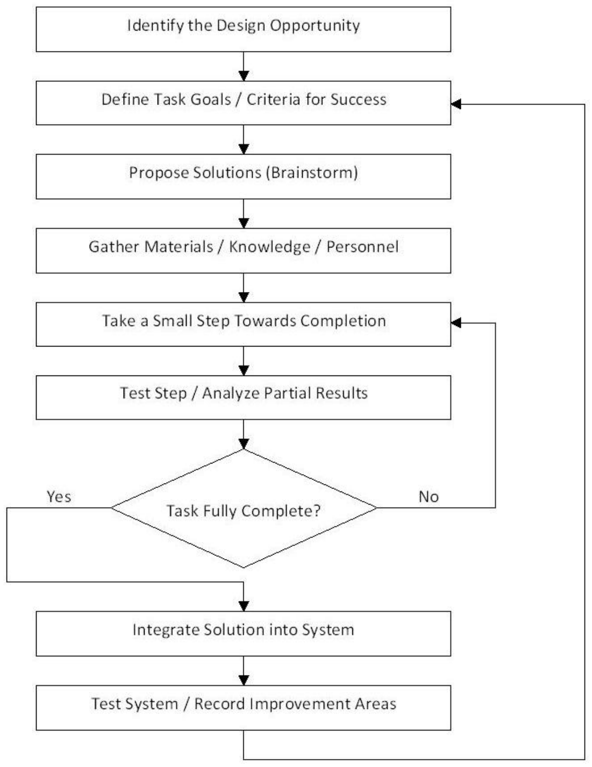

Identify the Design Opportunity
Define Task Goals / Criteria for Success
Propose Solutions (Brainstorm)
Gather Materials / Knowledge / Personnel
Take a Small Step Towards Completion
Test Step / Analyze Partial Results
Yes
Task Fully Complete?
No
Integrate Solution into System
Test System / Record Improvement Areas

Figure 1: Continual Improvement — A bottom–up approach, focusing on results at every step.

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 whiteboard. 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 interdisciplinary club of undergraduates and graduate students.

Name	Major	Level	Hours
Jeremy Langston	Electrical Engineering	Graduate	250
Hunter McClelland	Mechanical Engineering	Undergraduate	200
Marbin Pazos–Revilla	Computer Science	Graduate	150
Jason Taylor	Electrical Engineering	Undergraduate	100
Ben Eckart	Computer Engineering	Undergraduate	100
Tristan Hill	Mechanical Engineering	Undergraduate	75
Gerrit Coetzee	Mechanical Engineering	Undergraduate	20

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

- Hardware and Electrical — Responsible for mounting sensors to the Mini–Andros chassis, for repairing failed components, and for coordinating power distribution.
- 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.

It is important to note that, as pictured in the Venn diagram in Figure 2, many members worked in at least two separate design groups during the project.

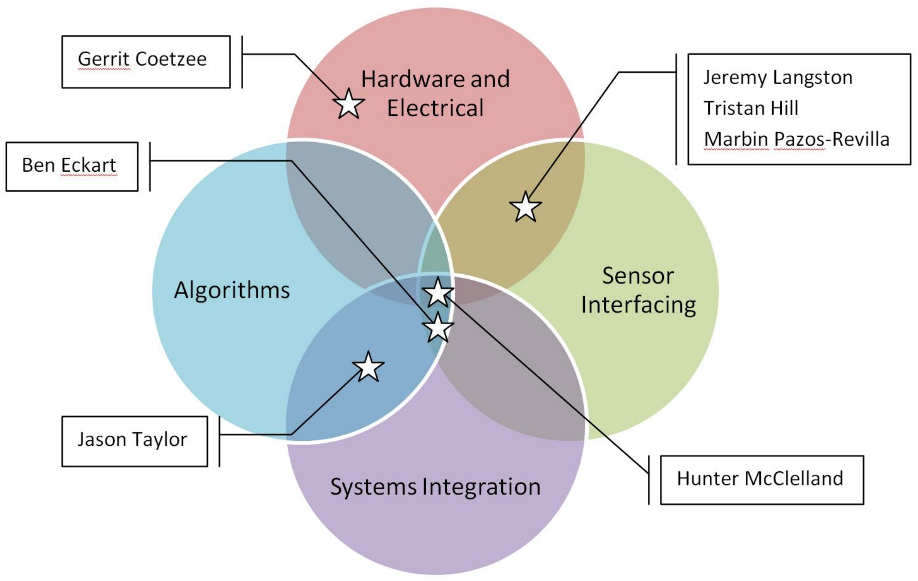
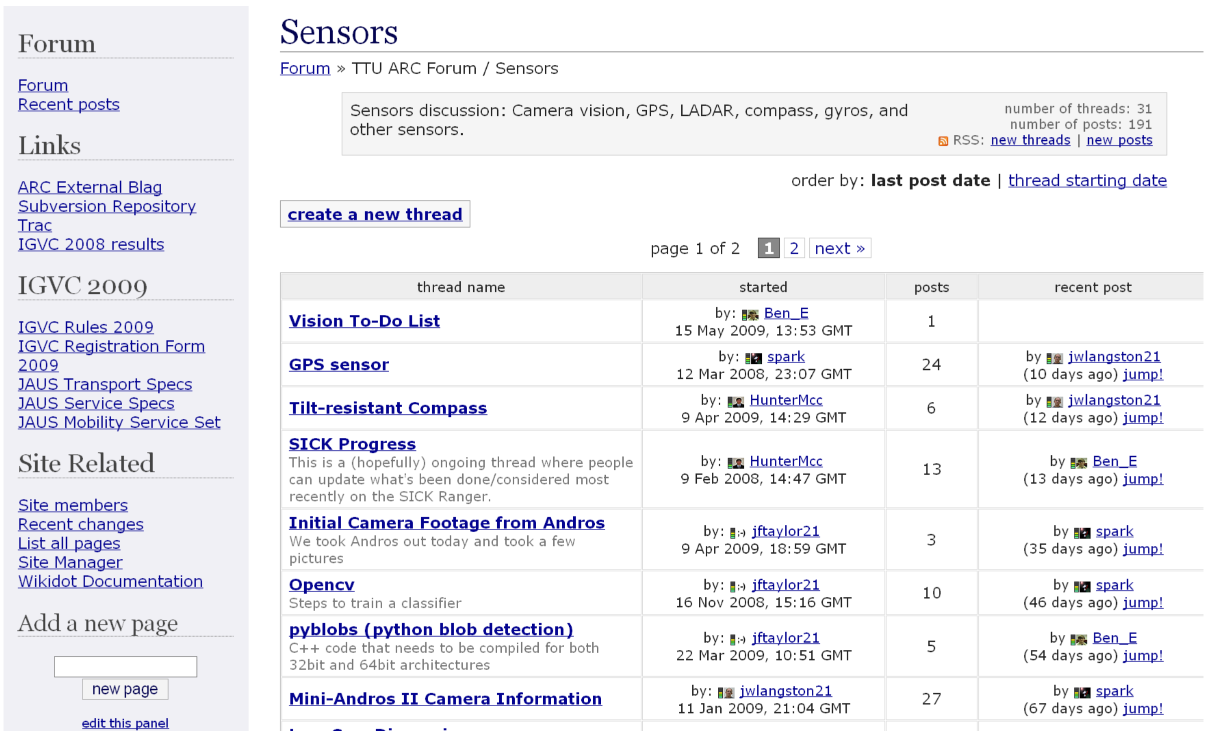


Figure 2: Venn diagram of the team arrangement.

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. A snapshot of one of the subforums is shown in Figure 3.



Forum

Forum
Recent posts

Links

ARC External Blag
Subversion Repository
Trac
IGVC 2008 results

IGVC 2009

IGVC Rules 2009
IGVC Registration Form 2009
JAUS Transport Specs
JAUS Service Specs
JAUS Mobility Service Set

Site Related

Site members
Recent changes
List all pages
Site Manager
Wikidot Documentation

Add a new page

new page

edit this panel

Sensors

Forum » TTU ARC Forum / Sensors

Sensors discussion: Camera vision, GPS, LADAR, compass, gyros, and other sensors.

number of threads: 31
number of posts: 191
RSS: new threads | new posts

order by: **last post date** | thread starting date

create a new thread

page 1 of 2 **1** 2 next »

thread name	started	posts	recent post
Vision To-Do List	by: Ben_E 15 May 2009, 13:53 GMT	1	
GPS sensor	by: spark 12 Mar 2008, 23:07 GMT	24	by jwlangston21 (10 days ago) jump!
Tilt-resistant Compass	by: HunterMcc 9 Apr 2009, 14:29 GMT	6	by jwlangston21 (12 days ago) jump!
SICK Progress This is a (hopefully) ongoing thread where people can update what's been done/considered most recently on the SICK Ranger.	by: HunterMcc 9 Feb 2008, 14:47 GMT	13	by Ben_E (13 days ago) jump!
Initial Camera Footage from Andros We took Andros out today and took a few pictures	by: jftaylor21 9 Apr 2009, 18:59 GMT	3	by spark (35 days ago) jump!
Opencv Steps to train a classifier	by: jftaylor21 16 Nov 2008, 15:16 GMT	10	by spark (46 days ago) jump!
pyblobs (python blob detection) C++ code that needs to be compiled for both 32bit and 64bit architectures	by: jftaylor21 22 Mar 2009, 10:51 GMT	5	by Ben_E (54 days ago) jump!
Mini-Andros II Camera Information	by: jwlangston21 11 Jan 2009, 21:04 GMT	27	by spark (67 days ago) jump!

Figure 3: Collaborative documentation and discussion with a wiki–style forum for all ARC members. This particular snapshot is of the Sensors subforum.

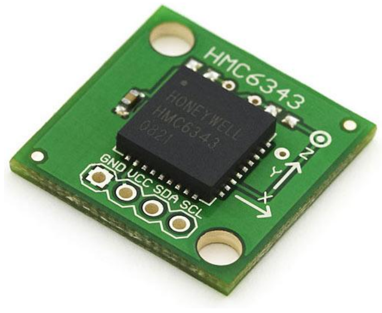
In addition to the documentation, we used a Subversion content management system for code sharing. This code repository proved to be a great source for members to review the latest revisions to code and quickly identify changes.

4 Sensors

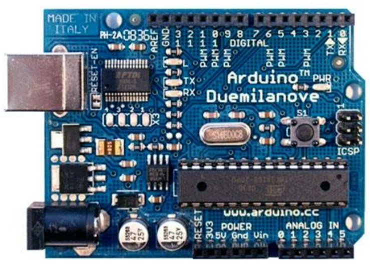
Andros's sensor suite consists of a laser range finder, two cameras, a GPS, and a compass. The sensors are controlled via drivers developed by the ARC, and are utilized by the navigation algorithms described in the following sections.

4.1 Compass and Arduino Microcontroller

We purchased a Honeywell HMC6343 tilt–resistant digital compass to supplement our GPS with heading information, both to assist in the waypoint challenge and to close the turning loop on the skid–steer design. Full three–axis readings are reported: heading (0°, 360.0°), roll (-90.0°, 90.0°), and pitch (-90.0°, 90.0°). These reports are retrieved at the rate of 5Hz, using I2C (inter–integrated circuit) communication via an Arduino microcontroller. Both compass and microcontroller are pictured in Figures 4(a) and 4(b), respectively.



(a) Honeywell HMC6343 tilt–resistant digital compass.



(b) Arduino microcontroller.

Figure 4: Programs are stored in non–volatile flash memory and communication with the host computer is via USB. The Arduino interfaces with the compass via I2C.

4.2 SICK Laser Range Finder

The LIDAR used by Andros is the SICK LMS–291 laser range finder, pictured in Figure 5. The main function of this sensor is physical obstacle detection. Andros's SICK Ranger has a maximum distance of 32 meters with a resolution of 10 millimeters. The range finder takes 181 length measurements at 1° increments per sweep. The SICK is mounted at the front of the robot between the articulators.

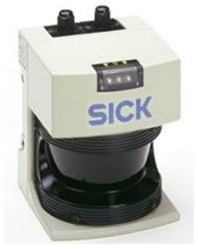


Figure 5: The SICK LMS–291 laser range finder.

4.3 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. There is a manipulator camera attached to the arm, which we use to view the robot's "toes". And finally, the main camera sits on top of the extendable boom. This camera can zoom as well as change the aperture and focus, and is also the tallest camera available. The cameras output a standard composite image so we used a video capture card to capture the video stream and digitize it.

4.4 Global Positioning System

Andros's GPS sensor is a GeoXT handheld unit from Trimble. The GPS is used in competition to locate and communicate way points to the high–level processes. Its specifications are as follows: accuracy of 1 to 3 meters, on board memory of 512 MB, and battery life of 24 hours. The refresh rate is 1 Hz and communication is accomplished via a serial connection.

In conjunction with the GeoXT, we use a Trimble GeoBeacon DGPS correction unit to receive real–time correction data, and transmit the correction to the GeoXT via Bluetooth. The combined GPS + DGPS system performs reliably with an accuracy better than 1 meter.

5 Electrical, Mechanical, and Computer Systems

5.1 Emergency Stop

Considering the requirements for IGVC 2009, our E–stop system guarantees that the motion of the robot comes to a complete halt. We do this by means of two robust DP/DT (double pole/double throw) relays with high current capacity contacts and an emergency stop toggle button. The E–stop is enclosed in a red box and located in the rear part of the robot, providing easy access and visibility. 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 a series of DP/DT relays. By means of a key fob transmitter, the W.E–stop can be activated, opening up the circuit that feeds power to the Andros motors and causing the brakes to engage.

5.2 Main Computer

The computer selected as the main processor for our system is the Intel D945GCLF2 Mini–ITX Motherboard, pictured in Figure 6.

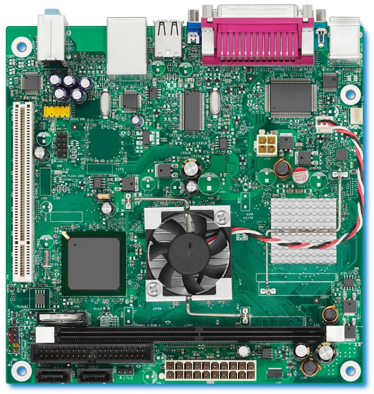


Figure 6: The Intel D945GCLF2 Mini–ITX Motherboard — 17x17 cm and four logical computing cores.

Mini–ITX is a standard specification for motherboards designed for embedded applications. With the advent of the Intel Atom, the Mini–ITX market is beginning to grow rapidly. This particular motherboard houses the new Intel Atom 330 CPU, a dual–core 1.6 GHz chip with hyperthreading and 1 MB L2 cache. Dual–core with hyperthreading 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 around $90 for a motherboard with a soldered Atom CPU and onboard video.

5.3 Mechanical

The chassis used by the ARC is a functioning Mini–Andros II from Remotec, a subsidiary company of Northrup Grumman. While a little outdated, this chassis is still being used today and is quite robust.

The Mini–Andros II is a ruggedized, weather–resistant EOD robot built to survive in intense environments. Like many 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 Mini–Andros II features a track–wheel hybrid skid steer system that is capable of climbing stairs with its attached articulators. The front articulator serves as the mount point for our laser ranging device discussed in the Section 4. The robot also has an extending boom with a camera mounted on it. Positioned at the rear axle, an arm attached to the chassis serves as the base for the payload and computer housing. Lastly, the robot features a manipulator arm which would be used for the actual disarming of the explosive device.

All of the motors and drivers are contained within the robot chassis itself. No changes have been made to the Mini–Andros II 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.

5.4 Power and Batteries

The robot itself contains all the motion controls as well as camera decoders. Except for the SICK laser ranger, all the external systems run off their own batteries. This way the sensing/computing 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 battery that supplies power to the Intel Atom 330 motherboard is a SLA 8.0 Amp–Hour battery, pictured below.

(a) A standard sealed lead acid battery for powering the Atom.

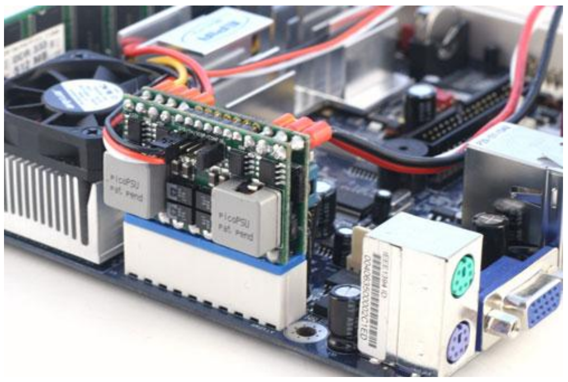

(b) Pico–PSU

Figure 7: 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.

At 12.5 Volts, the Atom board draws 2.1–2.9 amps. This means that the system draw no more than 37 Watts. Unfortunately, this calculation does not exactly translate into how long we can run on an 8 Amp–hour battery. The batteries steadily lower in voltage as they are used. The power remains constant and since

$$i = P/v$$

as the voltage drops, the current increases to compensate. As the current increases, it drains the battery more quickly, causing the voltage to drop faster. Also, the picoPSU automatically shuts down when the battery is less than 11.2 volts for more than one minute. Thus, the problem is not exactly linear, though it can be roughly approximated as such. Figure 8 qualitatively explains this phenomenon.

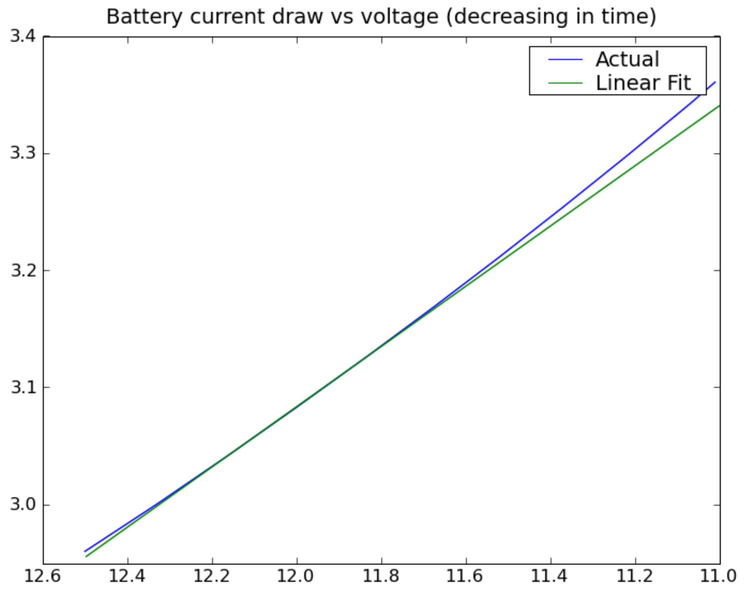


Figure 8: Power characteristics of the Atom.

This effect has a significant impact on the battery life. A linear approximation would show a battery life of about 2 hours and 48 minutes, but we found that the computer could run approximately 2 hours before power loss occurred. Heat dissipation is another concern for such computationally dense devices. We stressed the system by running system stress tests and monitoring the temperatures. The CPU temperature never rose above 45^oC, which is an acceptable value according to the Intel Atom 330 specifications.

6 Software Strategy

6.1 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 will be both robust and simple enough to serve as a strap–on solution for autonomy.

6.2 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.

6.3 Remote Debugging

Since Andros is head–less (meaning that we do not have a monitor attached to it) we use SSH and VNC to remotely control the robot over a wired or wireless network. SSH is used in cases where only textual output is desired (GPS testing) and VNC 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.

7 Systems Integration

The overarching software architecture is shown below in Figure 9. The figure is color–coded, and the scheme is detailed below the figure in its caption.

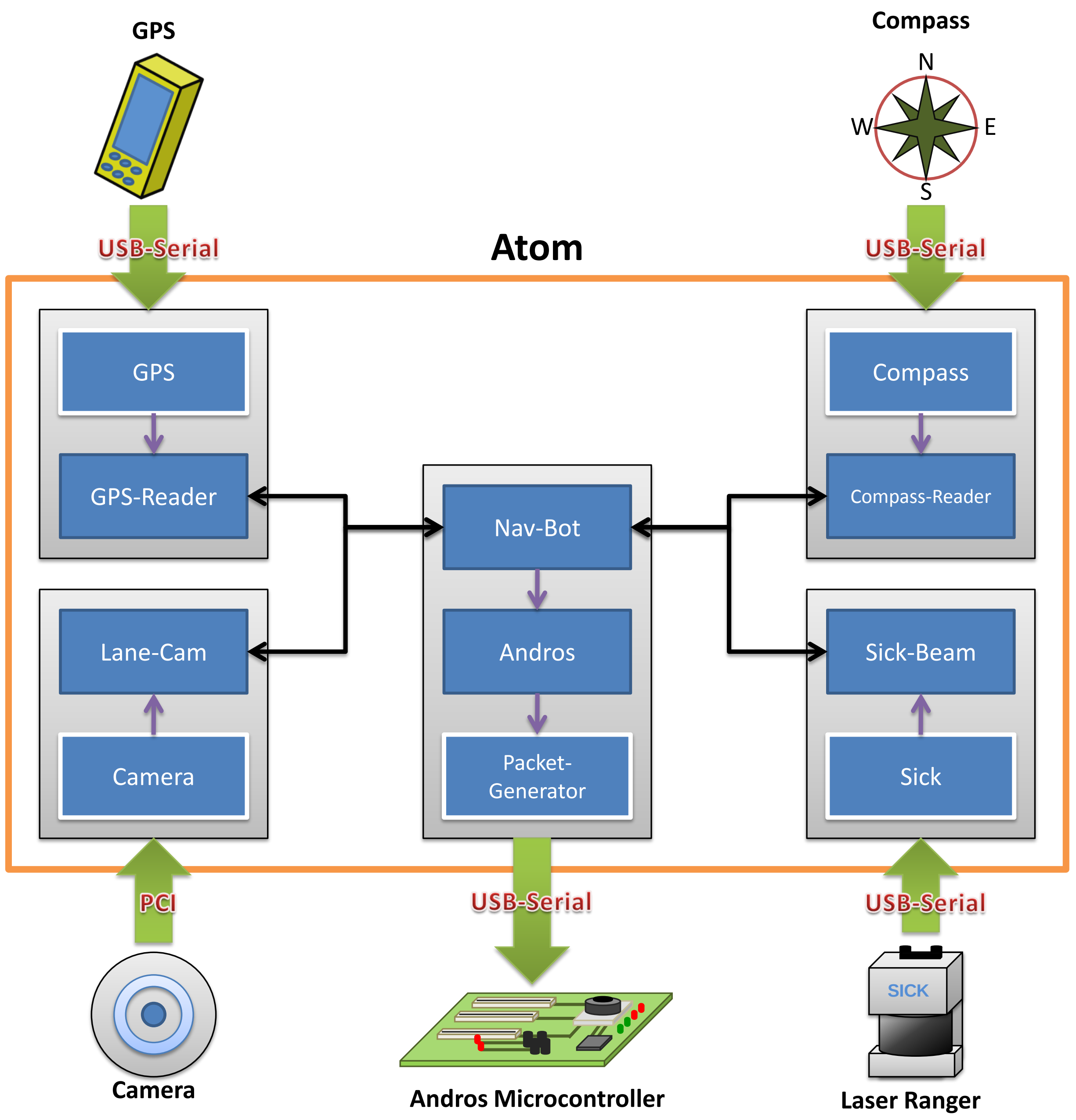


Figure 9: The software architecture: each gray box represents a separate process. Each green arrow represents I/O. Blue boxes are Python objects, and blue boxes with white outlines are threaded separately. Black arrows represent inter–process communication, which is done by sockets. Purple arrows represent object inheritance.

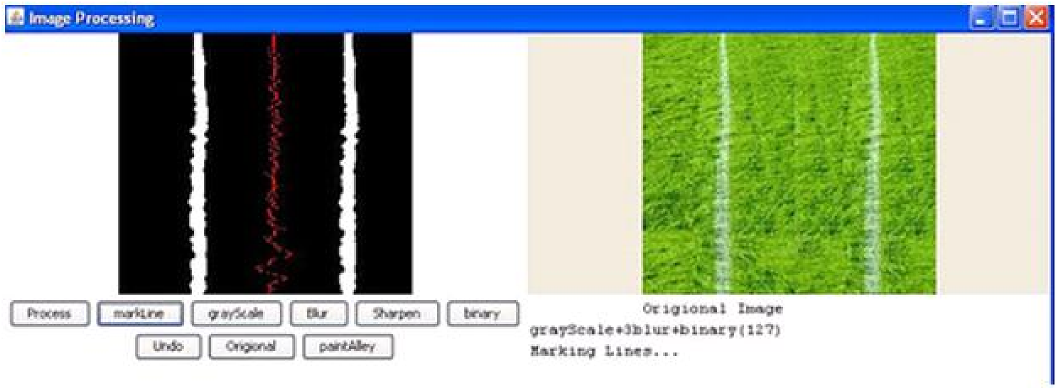
As can be seen in Figure 9, there are four main sensor processes and one navigation process that execute on the Atom. All I/O is done via USB–to–Serial adapters, with the exception of the camera system which is fed through a PCI card.

7.1 Sick–Beam

We wrote our own drivers for the SICK Ranger in order to be able to communicate in a continuous scanning mode, asynchronously, at 38.4 Kbaud. Our base class, Sick, simply gathers the most recent scan and places it in a "mailbox." The Sick–Beam class interprets this data to instruct the robot when there could be an impending collision with an object by checking a rectangular beam of influence in front of the robot. The speed at which scans are received is approximately 10 Hz. Thus, the robot's reaction times based on this sensor are approximately at a maximum of 0.1 seconds. Computation on the data is negligible compared to the I/O times. We find this to be suitable for obstacle detection and avoidance. The output of this process is simply a heading from -180° to 180°. 180° indicates a desire to turn in place counterclockwise, and -180° is a desire to turn in place clockwise. A 0° heading corresponds to a desired heading for straight ahead.

7.2 Lane–Cam

The Lane–Cam object inherits from a Camera base class, which provides basic GUI capabilities and the ability to capture output to video and stream over the network. The interfacing is done through the PCI capture card into OpenCV. Our current camera algorithm looks for two dominant lines using white blob filters and tries to find the middle point of the two lines. The algorithm is depicted visually in Figure 10.



```
Origional Image
grayScale+3blur+binary(127)
Marking Lines...
```

Figure 10: Simplistic lane following: finding the center between two detected lines.

There are several instances where this vision algorithm does not prove robust enough. For instance, shown in Figure 11, a man stands on the field in harsh lighting conditions. Due to the nature of the camera, the shadow of the man completely obscures part of the white line. This type of obstruction will clearly cause problems for our current process. It is expected that upon development of a robust solution for dealing with dashed lines, perhaps this can be extended to deal with "bad situations" of lighting and shadows.

The output of the camera is the same as the Sick process, outputting a "best–guess" heading from -180° to 180°.

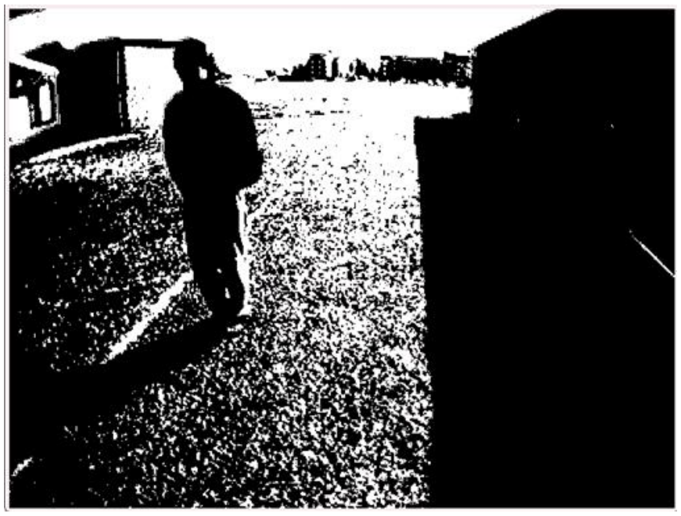


Figure 11: An example of a "bad case" for vision filtering.

7.3 Compass–Reader

In order to interact with the HMC6343 Compass, an intermediary microcontroller was required to translate I2C to serial communication standards. The compass connects to the microcontroller via a simple custom expansion shield, supplying power and communication. I2C commands and replies are sent as to achieve a sampling frequency of 5Hz. Once a complete heading is sampled, it is sent at 9600 baud to the high–level process using a USB–Serial adapter. The microcontroller of choice was the AVR–based Arduino Duemilanove due to its simplicity. All power required for both the Arduino and HMC6343 is supplied via a single USB connection.

7.4 GPS–Reader

The Trimble GeoXT 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 functionability 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.

7.5 Nav–Bot

Nav–Bot is the high–level algorithm that synthesizes the sensor data to traverse the course. Lane–Cam drives the robot until Sick–Beam interrupts its control with an impending collision alarm. At this point, Sick–Beam takes control of the robot to move closely around the object while still checking with the data from Lane–Cam to simultaneously not go out of bounds. In this way, Nav–Bot will stay in between the lines and hug obstacles, but not collide with them. The outputs from both Lane–Cam and Sick–Beam are simply heading guidelines as explained in the previous sections. A linear transformation of the heading value is used to turn the heading into wheel speed commands for the Andros. Thus, a -180° heading will cause the robot to turn in place, while a 30° heading will cause Andros to slowly drift to the right. New commands are given every 40 milliseconds.

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 Sick Ranger detected imminent collisions, this takes precedence and the GPS driver relinquishes control until the object is out of harm's reach.

Nav–Bot inherits from the Andros and Packet–Generator classes. These classes take the wheel speed commands and map them to the proprietary telegram stream format for use by the Andros microcontroller. Though we are very proud of our work on this aspect of the robot, unfortunately, the details of these classes cannot be explained in this report, as they would illuminate details of the Andros microcontroller interface, which is under a Non–Disclosure Agreement between Remotec and the ARC.

8 Cost Table

An itemized list of costs can be found in Table 2

Table 2: Estimated and Actual Costs.

Item	Estimated Cost	Cost to ARC
Remotec Mini–Andros II	$40,000	$0
Intel Atom Computer	$85	$85
RAM + Hard Drive + PicoPSU	$140	$90
SICK LMS	$6000	$0
Trimble GeoXT and GeoBeacon	$4600	$200
Arduino Microcontroller	$50	$50
Miscellaneous components	$50	$50
TOTAL	$44,925	$475

9 Lessons from Experience

Collectively as a club, we have slowly come under the realization that robustness of design is critically important for large robotics projects. When faced with high member turn–over due to graduating seniors, over–burdened juniors, and the whimsical interests of freshman and sophomores, it becomes increasingly important to design systems robustly. Thus, our design intent is to engineer such that components will not break often, and when broken, will break gracefully and in a way such that they can be fixed by someone who potentially was not present for the initial design.

Another lesson is that anything that *can break* will probably break at some point. Thus, it is a good idea to design enclosures in such a way that the innards of the robot can be opened without much effort. In all of these design goals for robustness, one of the best ways to sidestep the issue is to primarily use components that have already been professionally engineered. That is, if financially feasible, it is very sensible to try to aggregate as many off–the–shelf components as possible in an attempt to avoid re–inventing the wheel. Not only does this decrease development time, but such off–the–shelf products come with vendor support and/or strong community support.

10 Predicted Performance

The observed speed of the Andros is about 1–2 meters per second at maximum speeds. We have climbed inclines of over 40° and up stair sets. The motor battery life has been shown by field testing to be about 2 hours. The same is true of the Intel Atom powered by the SLA 8 Amp–hour battery. Our waypoint accuracy is governed by the GPS accuracy, which is about 1 meter. Our camera can detect objects up to 20 feet away, and the laser ranger can detect objects up to 105 feet away (although we don't look this far ahead). 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 SICK Ranger operate at 10 Hz/FPS, the compass at 5 Hz, 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 from Project Andros. We have designed a working prototype for a commercially produceable "bolt on" automation system and have applied it successfully to an existing EOD robot. The system presents obvious advantages over the previous tele–operation required to navigate the robot. By configuring our system to communicate via the JAUS standard, Andros is ready for demonstration to interested parties and participation in the IGVC 2009 competition.