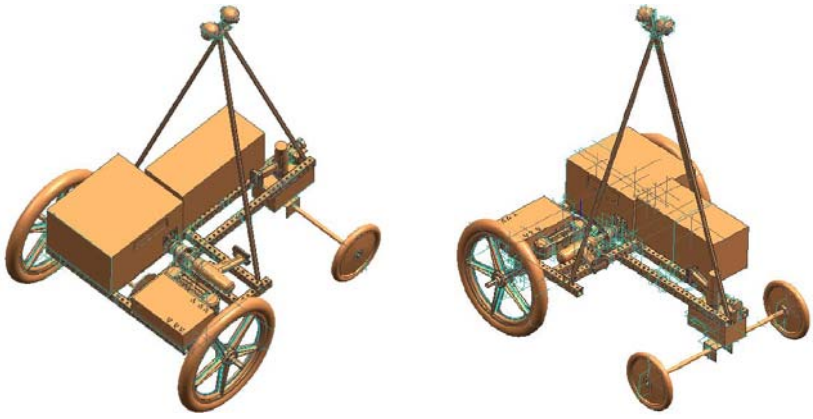
# Oliver

Vehicle Design Report

IEEE Robot Team
University of Wisconsin – Madison



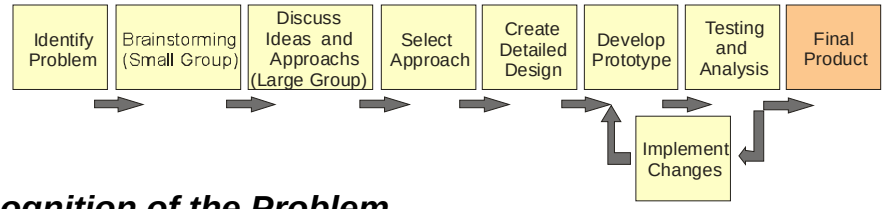
12th Annual Intelligent Ground Vehicle Competition
Oakland University – Rochester, MI
June 12-14, 2004

# I. Introduction

The IEEE Robot Team from the University of Wisconsin – Madison designed and created a robot to compete in the 12th annual Intelligent Ground Vehicle Competition (IGVC). The autonomous robot named Oliver was the seventh robot produce by a dynamic team of students over the last eight years. Oliver was the first robot created by the IEEE robot team to compete in the IGVC. The team has pooled their experience and genuine love of robotics to make this first entry to the IGVC to be a highly competitive for the veteran IGVC teams.

# II. Design Process

In the creation of the different components and subsystems on Oliver we followed the basic process seen in the diagram below. We began by identifying the problems proposed by the competition including items need to be identified and tasks need to be accomplished as well as initial strategies. We did our initial brainstorming in small groups and later discussed these ideas in a large group setting. After sufficient discussion we selected the approaches we felt would be most beneficial to the goal of the robot. From the selected approaches the team created detailed plans on how to design or setup each section. Using these plans, prototypes were developed. Through testing and analyzing the prototypes changes and retesting was done until we were satisfied with the final product.




## *2.1 Recognition of the Problem*

In order to create a proper solution, the problem proposed by the competition must be fully understood. To begin to understand the problem the team examined the competition website to try to gain a good understanding of the competition. Through studying the rules and former years design report comments our team obtained this understanding. We came to recognize the basic needs that the robot will need to perform to accomplish a task such as identifying some physical objects such as construction barrels and some visual objects such as simulated potholes. Through analyzing the specifics of the problems we began to formulate approaches to solve the problems.

## *2.2 Preliminary Ideas*

Our team's earliest meetings were spent discussing various approaches to the problems we identified. Through this process, many approaches were conceived. These different approaches dealt with varying types of chassis and drive systems, types of sensors, and methods for code. Among the styles of chassis and drive systems considered were using a tank, car, and go-cart style. Considering the type of course the team decided

upon a go-cart style in order to maximize accuracy and simplicity. The types of sensors that were considered were shaft encoder, infrared, sonar, camera, laser distance finder, and global positional system (GPS). The sensors chosen by the team for navigation and obstacle identification were shaft encoder for velocity measurements, the sonar as a backup warning system, the camera for the primary obstacle locator, and the GPS unit for an approximate location finder to be used in the waypoint navigation course. Our team also weighed the benefits and extra work involved with using distributed embedded systems, and a network linking them, and decided this was a worthy project. It was then debated whether the team should develop software in c/c++ or Java, and based on the team's overall proficiencies, it was decided to develop the main algorithms in MATLAB and eventually implement them in Java.

## *2.3 Team Organization*

In order to create a fully functioning robot more efficiently, our team was broken down into groups that focus on different areas of interest. The four main groups were financial, mechanical, electrical, and software. With in each group, projects with goals were established. Some of the projects included designing the chassis, getting sensors like the sonar to work reliably, and writing code for the artificial intelligence, control system, and network. Team members were free to choose which group and project they felt best fit their skills. Crossover and communication between the groups was encouraged to promote the integration of subsystems into a functioning robot.

To ensure progress of the team towards the team's goals the team regularly met in both small and large groups. The small group meetings allowed the concentration of a group to work on one of the projects. The large group meetings allowed for announcements and cross talk between groups.

### 2.3.1 Team Members

| Member Names | Major | Projects |
|---|---|---|
| Summeet Chandra | CmpE | Software – Network, Main App. |
| Mitch Cohen | CmpE | Software and Electrical – Control, Network |
| Jerry Ding | EE | Software – AI, Control |
| Tom Ebersole | EE | Electrical – Microcontroller |
| Ben Gartner | CmpE/CS | Software and Electrical – Main App., Peripheral Scanner |
| Jon Gullixson | EE | Electrical – Microcontroller |
| Dana Heitner | CivE | Financial and Mechanical – Chassis |
| Danny Minh Do | ME | Electrical – Power System |
| Eakkachai Pengwang | ME | Software – Vision |
| Brian Prodoehl | EE | Software and Electrical – AI, Vision, Network, Main App., Control, E-Stop |
| Chris Riley | ME | Financial and Mechanical – Chassis, Computer Model |
| Sean Scannell | CmpE | Electrical – Peripheral Scanner |
| Chris Scharnke | EE | Electrical – E-Stop |
| Jon Steffen | Bus | Electrical – Computer |
| Neil Thompson | EE | Software – Vision |

## *2.4 Development*

### 2.4.1 Methods

Proper software design and coding techniques were followed, resulting in highly modular, fully-encapsulated classes and packages. Unit testing was used where applicable, and all software was written with easy readability and maintenance in mind.

### 2.4.2 Design Tools

Many design tools were used in the development of Oliver. Software development was done with the eclipse IDE and inDart Visual Debugger IDE. Physical models were designed using Unigraphics NX.

## *2.5 Construction*

Prior to actual construction of the robot or components, the item to be constructed was designed in detail in an effort to minimize mistakes and flaws. The layout of components and structure of the chassis were modeled using Unigraphics. This allowed for critical decisions regarding basic structure to be made before actual manufacturing. The bulk of the parts were made by purchasing stock material and students machining the necessary components. With the proper parts created the pieces were constructed to form the body of the robot. Additional components such as the cameras were then added to the robot.

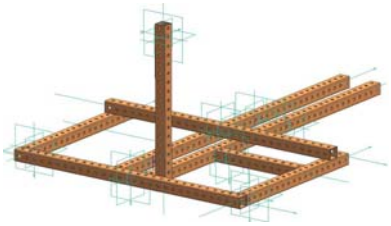
## *2.6 Testing*

Upon completion, each component was thoroughly tested to ensure safe, reliable, and durable use. These tests became a vital portion in troubleshooting and tweaking of systems and components.

# III. Mechanical Design

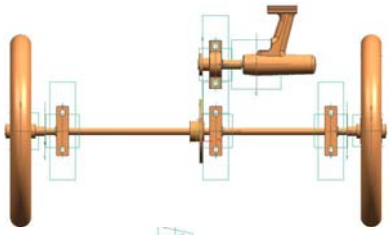
---

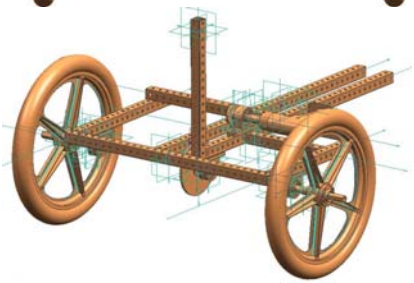
## *3.1 Chassis*



The chassis was constructed by using stock extruded square steel tubing. The tubing came with holes on each side every 1”. The tubing components were held together by bolted plate fasteners to provide a highly modular, sturdy base. The modeling of the robot allowed for verification that each system and component of the robot was adequately positioned, fastened, and supported.

## *3.2 Drive System*

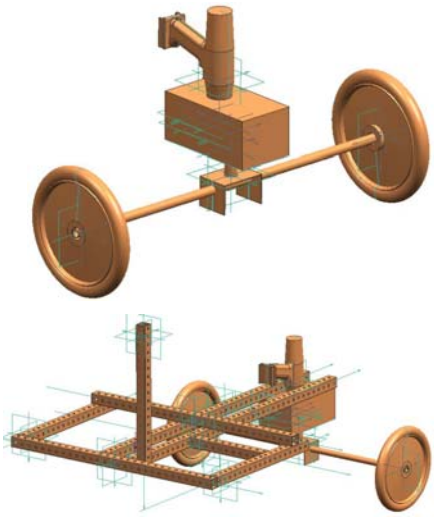




Oliver’s drive system was design to be as straight forward as possible. The drive motor is an 18 Volt cordless Milwaukee drill providing approximately 400 inch-pounds of torque. The speed of the drill is reduced by a 3:1 ratio using chain and sprockets. The sprocket is then attached to a single rear axel. The rear wheels in turn are locked in position with respect to axel. The system is able to output approximately 120 lbf. This

force is ample to provide locomotion for the robot up any incline presented in the competition.

### 3.3 Steering



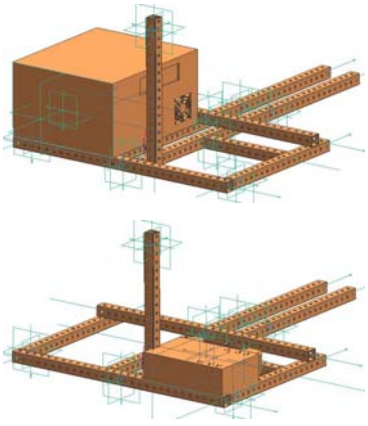
The steering is one of the most unique aspects in the body of the robot. Prior to settling on a means of steering, many methods were examined. Among these were a tank steering, with a motor on each side, go-chart steering, where the entire front axel pivots, and car steering, where the front wheels pivot but not the axel. Due to the nature of the course being winding requiring arc movement, the go-chart steering was chosen. By choosing this style of steering the model of the robot's movement was simplified.

The steering method was based on the angle of the front axel. The angle of the axel was adjusted by a 12 Volt Milwaukee cordless drill. Between the drill and the axel's vertical holder was a gear box containing a configuration of bicycle sprockets and chain. This gear box reduced the speed of the drill at approximately a 4:1 ratio. The wheels on the ends of the axel were free to rotate separately allowing angle adjustment while stationary.

### 3.4 Sensor Placement

The sensors were place in strategic locations that aided their collection of the type of information they were intended for. The sonar proximity scanner was placed low on the front end in an attempt to scan for people, buckets, and construction barrels. The sensor is mounted on a stepper motor and set to scan 270 degrees including the front and sides of the robot. The two cameras are mounted approximately five feet off the ground. The primary camera faces forward and points down at approximately 40 degrees. The secondary camera intended for backing-up faces backwards and points down approximately 70 degrees. These placements provide a high vantage point from which the cameras are able to capture important regions of the obstacle course.

### 3.5 Other Component Placement



With additional components such as a computer, battery bank, E-Stop, and payload locations on the chassis were determined based on the computer model. The computer was laid sideways, in the back left of the robot, on cushioning and secured by straps in place. The battery bank was located in the back right of the robot. Additional power strips with connectors were located in this area to

promote ease of connection, battery replacement, and safety. The E-Stop button was located 2'5" off the ground, centered on the back of the robot. The payload was placed in the front slightly shifted to the left of center to provide a better view for it's camera while not interfering with the sonar sensor.

# IV. Electronic and Software Design

## *4.1 Overview*

Applying the lessons of the past few years, this year's electronics and software were geared towards ease of design, development, debugging, and maintenance. The robot now relies on a network of embedded systems, and the software is entirely modular.

Monolithic embedded systems, in which all functions are performed by a single microcontroller, have resulted in many problems in the past. Because of the multitude of tasks required of it, the code is often difficult to read, and it can be almost impossible to add or expand a function without interfering with something else. Diagnosing and fixing problems was another difficult task. The systems were prone to crashing or hanging, and finding the culprit was nearly impossible. To avoid these problems, it was decided that a distributed solution would be developed. Each embedded task would run on a dedicated microcontroller, and these microcontrollers would be linked by a network. To accomplish this, a low-cost, feature-rich, high-performance microcontroller, the ST7FLITE0, was used. These microcontrollers have an internal 8MHz clock, five ADC's, two counters, a PWM output, a real-time clock, and a hardware SPI network interface. Coming in a DIP package, running on 3.3-5V, and costing only $0.80 each, these were a very viable solution. Furthermore, an inDart Development Kit, Visual Debugger, and Cosmic C Cross-Compiler made the development and testing of each system very easy.

## *4.2 Network*

The first embedded system developed was the network master. This was required to act as invisible middle-man, routing packets of data to and from the embedded and PC systems. The goals of the system were to operate reliably and invisibly, requiring as few I/O lines as possible, supporting 2Mbit transfer rates and having a worst-case latency of less than one millisecond without impeding the general operation of the other devices. The first task was to design and agree upon a transport protocol, defining the packet structure and each system's unique 4-bit identifier. These identifiers are shown below. With the protocol finalized, design of the general cycle of the network master was the next task. It is vital that the network not impede the operation of the various systems, but it still needs to collect and route packets. Eventually, a method of polling each system, waiting, and repeating was

| | |
|---|---|
| 0x0 | Specifies Null Packet |
| 0x1 | PC Master Control |
| 0x2 | PC Control System |
| 0x3 | PC AI |
| 0x4 | PC Vision System |
| 0x5 | Embedded Control System 1 |
| 0x6 | Embedded Control System 2 |
| 0x7 | Embedded Proximity Sensing System |
| 0x8 | Embedded E-Stop |
| 0x9-0xE | Unassigned |
| 0xF | Embedded Network Master |

agreed upon. Serial Peripheral Interface, the network interface supported by the ST7FLITE0 microcontroller, has an interesting full-duplex nature, in which bytes are sent and received simultaneously, and only simultaneously. The network takes advantage of this when communicating with a system. It transmits either a null packet or a packet intended to go to that system while receiving either a null packet or a packet intended to go to another system. After this transaction is completed, the network master ceases communication with that system and advances to the next. After it has communicated with all systems, it waits (currently for 500ms) and repeats the process. We have performed only limited tests on this system, but by all accounts it appears to operate within the timing and performance specifications agreed upon.

To take advantage of this network, a simple solution was sought. It was finally determined that a library of functions that would allow a system to send and receive packets with simple send and receive functions is the best choice. A file, RNTP.c, was developed, including a function to initialize the ST7 control registers, enabling and configuring the SPI interface and local data structures. Also included was a simple send function, to which the user passes the data byte and the ID of the destination device (per the RNTP standard), and a simple receive function that returns a pointer to a packet that has yet to be processed. Packets are transmitted and received through queue data structures, so the first packet in is also the first packet out.

It is very desirable for the PC systems to appear independent on the network, so a system was developed to provide a transparent layer between the embedded network and all the PC system processes. An SPI UART manufactured by Maxim was used to provide the link between the embedded network and the PC serial port. In software, a Java process was developed to decode incoming packets and store them, and also receive outgoing packets from each PC system, and send them to the embedded network. This also provides a simple way for PC systems to communicate with each other, although this capability is not currently used.

### *4.3 Power System*

The power system was the focus of a senior design project, and the result was a reliable, long-lasting solution devoid of all the shortcomings of the past years. To power the embedded systems, long-lasting rechargeable NiMH 12V batteries were used. The power requirements of the embedded systems are fairly low, with some devices needing 12V, and some needing 5V. The overall max current draw is between one and two amperes, so two 5V and 12V regulators in parallel provide clean 5V and 12V power sources, with a max current draw of two amperes. Powering the PC was a source of conflict, but it was finally agreed upon to use a battery-backup uninterruptible power supply, providing over 40 minutes of runtime without being plugged in. Although not long, this is more than enough than what is needed for the competition. The drive motors are standard 12V and 18V cordless drills, and the easiest way to power them is to use several of their batteries in parallel. This system was designed with quick battery-swapping and long life in mind, and has passed numerous comprehensive tests.

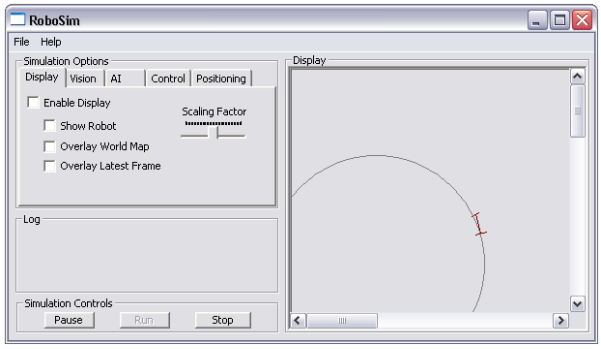
## *4.4 PC Systems*

### 4.4.1 Overview

With the network as a sort of middle ground, the remaining systems can be easily placed into two categories: PC and embedded. On the PC side of things, the vision system captures information about the area in front of the robot, the artificial intelligence pieces together information on where the robot is and what is around it to determine the best course of action, and the control system takes this course of action and makes it a reality.

### 4.4.2 Main Control Application

Linking these systems together is a main control application that allows fully scalable simulations to be run. The extent to which each system runs is configured by the user. On one end of the spectrum, the user can run a complete software simulation, in which the vision system acquires information from a series of files, to test the decision-making capabilities of the artificial intelligence and control system. On the other end of the spectrum, the user could perform a full test-run, in which the robot acquires information from the real world, and attempts to navigate a real course. Integral to this application are graphical and physical models of the robot, necessary for software simulation. These elements have proved invaluable to the development of the rather unique control system, which was unexpected. A screenshot of the main control application is shown to the left.



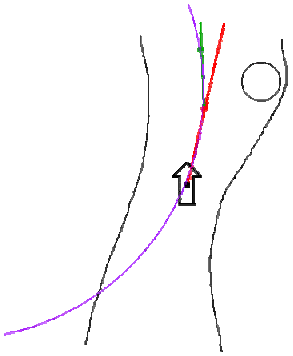

### 4.4.3 Vision System

The vision system needs to acquire images of the area in front of the robot and transform that image into a segment of a top-down map of the course, in which obstacles are white, and everything else is black. This has always been the main consumer of processor bandwidth and resources, so special care was taken in its design and implementation. To optimize speed and efficiency, a native, compiled solution was sought. Since Windows XP was selected as the operating system, several built-in technologies are available for acquiring images from a video device. These include Windows Image Acquisition (WIA) and Windows Image Acquisition Automation (WIAA). Functioning prototype systems were developed in Visual C++ utilizing WIA, but integrating them with the Java main control application was too difficult, so another solution was sought. ARToolkit, developed at the University of Washington, provided capture capabilities as well as many advanced image processing features, but lacked support for several of the cameras we used during development. In the end, the Java Media Framework (JMF) and Java Advanced Imaging API (JAI) were used. JMF provides support for video capture and frame grabbing, while JAI provides all of the image processing operations we need in a fast, compiled format (many of which take advantage of MMX). This solution can easily acquire images, turn them into black-and-white images (with white being obstacles and path edges), and transform them, fulfilling

many of the requirements. The only missing link is determining the threshold with which to clean up the image, or the way to determine whether a pixel is part of an obstacle or not. To help find the best way of thresholding the images, many images from previous years (obtained from the competition web site) were analyzed. In particular, histograms were observed. It was concluded that the red and green intensity were not very uniform between the majority of images, but the blue intensity between all images was nearly the same, with the majority of the intensities at very low levels, and a smaller-yet-still-significant group at high intensities. This group at high intensities was found to correspond to the path edges, artificial pot-holes and barrels. Handpicking a blue intensity level between the low intensity peak and high intensity peak, and thresholding the image based on that proved successful for all images, but the intensity level needed to be tweaked for each image in order to achieve the best performance. Going back to the histogram, it was determined that the optimum threshold was always right after the main hump declined. An algorithm was quickly developed to smooth the histogram and find the point beyond the maximum at which the slope is within a certain tolerance of a preset value. Applying this algorithm to find the threshold for each image yielded spectacular results. This algorithm is now an integral part of the vision system, and is capable of quickly and effectively determining a new threshold level. We have tested this vision system in many lighting conditions and have had great success.

### 4.4.4 Artificial Intelligence

The primary function of the artificial intelligence is to maintain a map of the course, in the form of a probability map, representing which areas are safe to drive on and which are not. The map segments from the vision system and the current positional data of the robot are used to merge the latest frames with the past assumptions. As the map is updated, it is also probed to determine the longest possible forward path. Since the robot drives in circular arcs, the AI needs to find the optimum circular arc for the robot to follow. This is accomplished by probing the map and finding the optimum straight line path for the robot, and probing once again from an intermediate point on that straight line path. Three points can be extracted from this and used to form an arc-shaped path. The current position of the robot is the first point, the first intermediate point is the second, and the third and final point is an intermediate point on the second straight line path. This is demonstrated in the diagram to the right.



Interested mainly in moving forward through the course, it is essential that the artificial intelligence also maintain a layer of directional data pertaining to the course. This directional field is populated with the optimal directions as determined above. When the AI is determining the optimal path, it only looks within a specified range of the directional field. This can lead to situations in which the robot becomes trapped, and these situations are handled by special cases in software.

### 4.4.5 PC Control System

The control system has many intricacies that haven't been dealt with in previous years, and has proved to be a great challenge. The main difference between this year's robot and the robots of previous years is that the paths followed this year will all be circular, as opposed to straight lines. This means that the control system will need to receive the current location and orientation of the robot, as well as two other points that represent the optimum path of the robot. Given three points, it is a trivial task to find the equation of the circle that passes through them. From this, the center and radius of the circle can be determined. The task then is to quickly align the robot with that circular path without drifting too far to either side. Software simulation has helped tremendously with the development of this system. The most error-prone stage is the initial aligning of the robot with the circular path. The body of the robot may be far from aligned, so the goal is to align the body quickly and without moving far from its current location (to minimize the chance of running off-course).

Much of the risk of driving off-course or hitting something is removed by the way in which the control system operates. With each new frame from the vision system, a new optimum course is found and the control system determines the best way to align with this new optimum course. This corrects many possible situations in which trying to align with an old optimum would drive the robot off the course.

## *4.5 Embedded Systems*

The embedded systems are fairly straightforward. There are two very different control systems, a sonar proximity sensor, and an emergency-stop system. The first control system is responsible for the forward motion of the robot, and the second control system is responsible for the turning radius. The sonar proximity sensor is capable of scanning an area and reporting back the location of any nearby obstacles.

### 4.5.1 Control Systems

The first control system controls the forward and reverse speed of the robot, safely ramping up and down to minimize wear on the motors, gears, and fittings. It receives byte commands from the PC control system, and decodes them so that the possible range 0-255 corresponds to full reverse to full forward, with 127 signifying a stopped position. As these commands are received, the actual level is incremented or decremented every ten milliseconds, and if the robot is expected to change direction (forward-to-reverse or reverse-to-forward) the motors are set to a stop position. This means that a robot traveling in full reverse (0), receiving commands to drive full forward (255), would immediately come to a stop and finally attain full speed 2.5 seconds later. A situation in which a robot moving at a slow forward speed (150) is issued commands to move at a slightly faster speed (200), that speed would be attained half a second later. For safety, the system kills the motors if a second elapses without receiving a new command signal. It then issues a command to the PC main control process stating that it stopped receiving

control signals, so that the main control process can repair the situation. Similarly, it kills the motors if it receives a halt command from the emergency stop system.

The other embedded control system receives the desired angle of the front axle, and acts like a servo to position itself at that angle. It performs a similar gradual movement as discussed above, but in a different manner. The front axle is repositioned by a single motor that rotates either clockwise or counter-clockwise at a predetermined rate. The axle is connected to a potentiometer, so that its position determines its output voltage (similar to a servo motor). This analog level is read in by the control system, and compared to the desired level. If the axle must rotate clockwise, the motor is driven clockwise, and vice versa. There is an acceptable range of variance so that the motor isn't constantly driven about some point. This system has been tested, and the speed at which the motor turns and the relative freedom with which the front axle rotates result in spectacular results.

### 4.5.2 E-Stop

The emergency stop system was the result of careful planning, with several preliminary designs and prototypes thrown out along the way. The first concept was to design and construct a simple transmitter and receiver to let the operator send a halt command to the robot. A CPCA (carrier-present, carrier-absent) scheme was adopted, and the hardware was relatively simple to construct. The transmitter unit transmitted a clean 10kHz sine wave over a very short range (5-10m), and the receiver then filtered the incoming signal and determined if there was a strong 10kHz wave present. The transmitter could then transmit data in a rudimentary fashion, having the carrier on for 1ms to represent a binary 1, and not transmitting a carrier for 1ms to represent a binary 0, with a binary 1 as a start bit. This system worked fairly well, and facilitated the sending of several commands. However, the receiver was very prone to interference, and would often receive what it believed was a stop signal. This performance is unacceptable, so a better solution was sought. Eventually an embedded system was developed to allow the robot to be driven using an RC controller. The control systems were made to switch between accepting PC commands and RC commands send by the remote control system. It was natural for the remote control system to also decode stop commands. The remote control system took on added functionality and became the emergency stop system. It should be noted that while the control systems are set to accept commands from the PC (as they will be during competition), driving commands from the RC controller are completely ignored. In addition to wireless stop signals, the emergency stop system also allows the operator to stop the robot by pressing a large button. In case an emergency stop state is entered unexpectedly, a resume button is also provided, although a special button sequence is required to issue a resume command.

## *4.6 Sensor Systems*

### 4.6.1 Camera

The cameras used during development included Logitech and Intel USB webcams, and the camera being used for competition is an iBot USB 2.0 Webcam. This camera is capable of high resolution, uncompressed video at 30 frames per second.

### 4.6.2 Sonar Proximity Scanner

An ultrasonic proximity scanner provides much needed information on the location of nearby objects, and scans the proximity of the robot on its own searching for barrels that may have eluded the vision system. Early on in the design process the possibility of hitting an object that was just out of the camera's field of view was brought up. It was determined that this needed addressing, and an ultrasonic proximity scanner would be able to reliably locate such objects.

The sensor unit was removed from an older Polaroid camera's auto-focus module, and was interfaced to a microcontroller with a custom designed circuit. The unit was then mounted to a stepper motor so that it could obtain information from all around the robot. The controller moves the stepper motor in 5 degree increments, triggers the ultrasonic unit to send a ping, and counts how long it takes for an echo to return. The accuracy of this device is more than sufficient for the task it performs.

The proximity scanner functions in two modes. In the first mode it scans back and forth, looking for very close objects. If an object is detected, a command is issued to the PC main control system, and if the object appears to be an immediate danger, a halt command is sent to the embedded control systems. In the second mode, the scanner is issued a command by another system to scan a specified quadrant. The scanner then reports back the readings for that quadrant. Commands are handled on a first-come, first-serve basis.

### 4.6.3 Optical Encoder

A standard optical encoder was used to limit the robot's maximum speed to just less than 5mph. The encoder was mounted on one of the rear tires, where slippage was minimal. This encoder is read by the control system responsible for forward motion.

### 4.6.4 GPS

A Magellan personal GPS receiver was linked to the PC over the serial port. The PC main control application then periodically retrieves the robot's position by calling a function that receives and decodes the NMEA packet sent by the GPS receiver. The initial position of the robot is subtracted from this position, resulting in an absolute position necessary for waypoint navigation. Initial tests have shown that this process is sufficient for reaching at least a couple waypoints successfully.

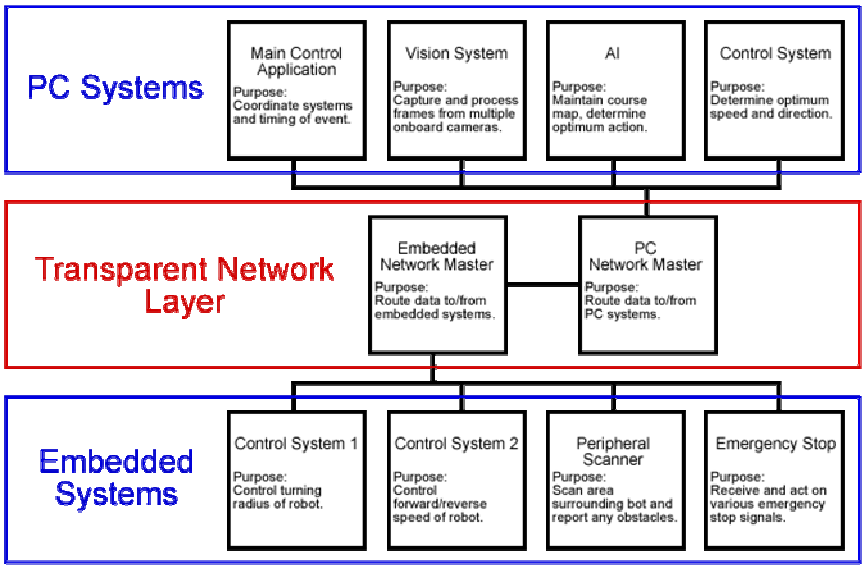
## *4.7 Computer*

The computer being used is geared for performance, which becomes a necessity when performing machine vision in real-time. The processor is an AMD XP 2600+, utilizing 512MB of DDR memory with a 333MHz Front-side Bus. It is housed in a standard mid-tower, on a standard ATX motherboard. Although this solution is not the smallest or most power efficient, it was the most cost-effective and is able to meet the processing requirements. Initially the platform was going to be a VIA EPIA single-board computer, but benchmark scores suggested that the onboard CPU would not be up to the task of real-time image processing.

Based on team proficiency, Windows XP was selected as the operating system. Lacking members with strong Linux skills, Windows XP was a natural choice.

# V. Systems Integration

## 5.1 Overview




All systems were designed with modularity in mind, so integration was a relatively simple task. The network effortlessly linked the embedded and PC-based systems together, and the PC main control application oversaw the operation of the PC systems. The PC vision, AI, and control systems were designed and developed as fully encapsulated packages. This made testing very simple, and ultimately integrating them into one application was a trivial task.

## 5.2 Sensor Integration

The sensors were designed so that each served a purpose, and functioned in a way that none of the other sensors could. The primary camera is able to look far ahead and determine the location of the boundaries and various obstacles. The sonar scanner is able to find nearby obstacles on all sides, and the GPS is able to report the robot's immediate position.

## 5.3 Path Navigation

Path navigation is achieved through the hard work of many systems. The vision system is highly capable of discerning what portions of the ground ahead of the robot are safe to drive on and which are not. This information is translated into a top-down map that the AI and control system use to determine and safely follow an optimum path.

## 5.4 Waypoint Navigation

Waypoint navigation is marginally successful with this year's robot. This is an aspect that could certainly be improved in the coming years. The error present in standard GPS signals results in an absolute positioning system that leaves much to be desired. Although capable of achieving some success on the waypoint navigation course, not much is expected in this area.

# VI. Predicted Performance

## *6.1 Preliminary Tests*

Tests were performed on all systems. The network was tested for throughput and worst-case latency using some specially developed software. The vision system was tested to ensure proper operation in numerous lighting and weather conditions, and was also tested for speed, counted in number of frames per second it could fully process. The artificial intelligence was tested by presenting the algorithms with hand-crafted test cases to ensure proper operation, and eventually with log files during trial runs. The control system was tested in the same manner. The embedded control systems were tested thoroughly using a logic analyzer and oscilloscope, and eventually with the actual speed controllers and motors to make sure that the desired speeds matched the actual speeds of the motors. The sonar proximity scanner was tested and calibrated for accurate distance measurements, and also for the speed at which it could scan a quadrant. The emergency stop system was tested for reliability under all conditions, on many different parts of campus. The entire robot still needs to undergo all-weather testing to ensure it can withstand moderate rain showers, but this is not expected to be a problem.

# VII. Other Design Considerations

## *7.1 Safety*

Safety was among the greatest concerns in designing Oliver. A portion of the safety measures were implemented through careful layout of the robot's components. An example of this is the isolation and simplification of the power system onboard the robot. In addition to having reliable systems some safety features were required through the rules. These safety measures include the dual method E-stop and the speed limit. Further safety measures are a result of the robot being developed with reliable and durable systems.

## *7.2 Reliability and Durability*

Reliability within the robot was achieved through rigorous testing of each system and component. Durability of the robot stems from the careful planning of robust systems and components. The reliable and durable systems allow for consistency in the robot's performance.

## *7.3 Maintenance and Interchangeability*

Modularity was in fact a primary focus in the design of each component. Through the modular design of the robot the linking of components was easily achieved and therefore maintained or replaced.

### *7.4 Cost*

The cost of creating a robot from scratch can be a substantial investment.

| Category | Item | Price Orig. | Price 2nd | | Category | Item | Price Orig. | Price 2nd |
|---|---|---|---|---|---|---|---|---|
| Computer | Motherboard | $30 | $90 | | Electronics | inDart Develop. Kit | $308 | $0 |
| | CPU | $105 | $105 | | | ST7 Microcontr. | $8 | $8 |
| | RAM | $105 | $105 | | | UARTs | $18 | $18 |
| | Case | $51 | $51 | | | Misc. Compon. | $50 | $50 |
| | HD | $147 | $147 | | | iBot USB 2 cam | $60 | $60 |
| Batteries | APC Back-UPS, 350VA | $39 | $39 | | | Victor 883 Spd. Contr. | $149 | $149 |
| Chassis | Steel | $50 | $50 | | Motors | 18V Mil. Drill | $0 | $229 |
| | Al. Channel | $6 | $6 | | | 12V Mil. Drill | $0 | $140 |
| | Wheels | $70 | $70 | | | Vexta Stepper | $0 | $25 |

The prices that are lower are either due to item that were donated or the need to purchase only one of the items. The net cost for our team to produce this robot was $1196. To create another similar robot the estimated cost would be $1342.

## VIII. Conclusion

The IEEE Robot Team has brought together a diverse team of students to design an autonomous robot named Oliver to compete in the Intelligent Ground Vehicle Competition. The team designed the robot to exceed all specifications while holding to our team goals. The goals we attempted to achieve included expanding student knowledge and involvement, creating of a robot that is safe, reliable, and durable, and to have fun in the process. We feel that we have in fact met these goals.