The MCP II:
Design Report for the
2007 Intelligent Ground Vehicles Competition

91.548 Robot Design
Professor Fred G. Martin
Department of Computer Science
University of Massachusetts Lowell

Prepared by the students
of the Spring 2007 Semester graduate course
(in alphabetical order):
Kenneth Dillon
Amr Elbasiony
Chris King
Joel Michel
John O'Fallon
Nathan Palmer
Haiyang Zhang

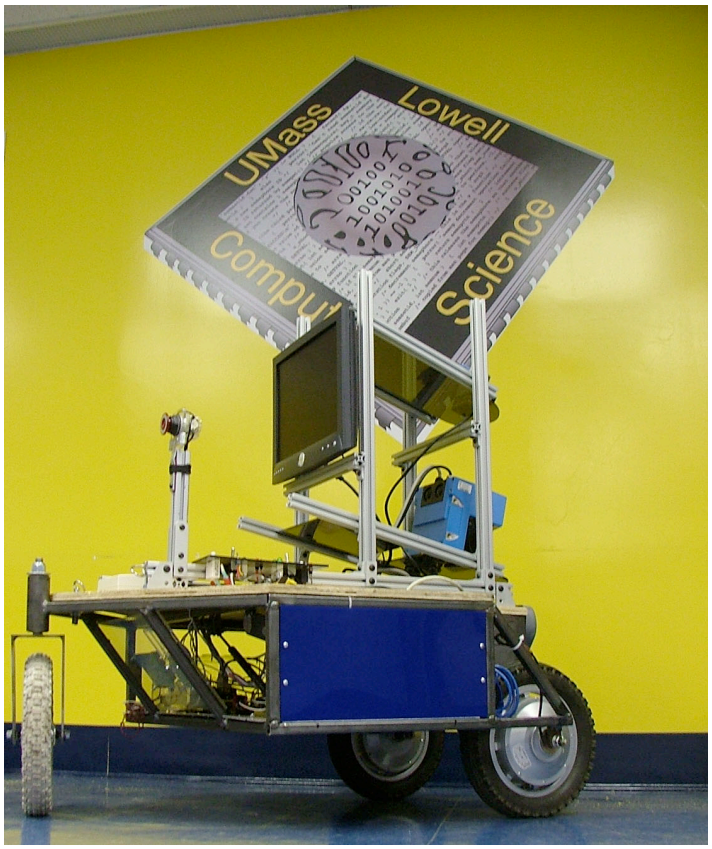
Honorary team members, alumni, and friends:
Brian Bailey
Andrew Chanler
Kyewook Lee
Yan Tran

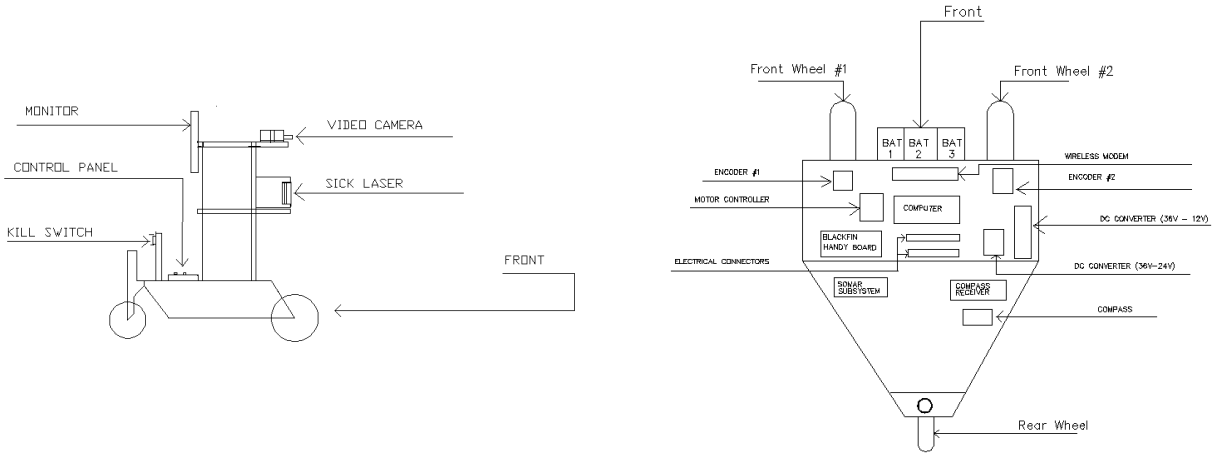
Introduction

UMass Lowell Computer Science fielded an entry in the 2005 Intelligent Ground Vehicle Competition, which was held in Traverse City, MI. Building on this work, the team won an award from the Robotics Industries Association (RIA) in September 2005. The award consisted of a 2-year loan of several critical robot design components, including a SICK laser range finder and a Foculus IEEE1394 camera.

Over the subsequent year, these components were integrated into the team's original platform, named the MCP. This robot was built into a child's ride on toy (a PowerWheels Jeep). While never stellar, the performance of this platform deteriorated as weight was progressively added, and the students decided that a new chassis was necessary. As part of developing this chassis, an entire redesign of the robot platform, including software architecture, was undertaken. This document describes the new design, the MCP II. Following are the main sections of this document:

1. **Hardware Design.** This is the largest section of the report, and describes the robot's physical design, motor subsystem, electrical subsystem, computer subsystem, and sensors.
2. **Safety, Reliability, and Durability.** This section describes the robot's safety engineering.
3. **Software Strategy.** This section describes the "Master Control Program" (which lends the robot its name), including versions for the Autonomous Challenge and Navigation Challenge, our use of simulation and virtualization, and the vision system.
4. **Design Process and Team Organization.** This section explains the team's design process, which is inspired by principles from the Agile Design community.
5. **Design Innovation.** This section highlights innovative aspects of the MCP II's design.
6. **Systems Integration.** This section describes the process the team has used to integrate the robot's subsystems.





Figures 1 and 2: Side and Top View of the MCP II

1. Hardware Design

Physical Design

The MCP II's base is oriented in a tripod formation with two twelve-inch hub motors and wheels on the front and a rear eight-inch caster wheel situated at the rear. The two hub motors are independently controlled via a single motor controller allowing the front wheel driven robot to using differential steering. Using a rear caster wheel and differential drive enables the MCP II to easily maneuver and pivot around obstacles even in tight spaces.

The central cavity of the robot contains the control and power systems which includes three 12 volts batteries placed at the front of the robot to keep the center of mass over the drive wheels. A motor controller is located directly behind the batteries. A wheel encoder is situated directly behind each hub motor to enable accurate representations of the robot's current motion. A Pentium 4 motherboard and a Blackfin Handy Board are located toward the rear of the cavity, where they are protected by acrylicate plates designed to absorb shock. These two components process the sensor information captured from the Foculus IEEE1394 camera and the SICK laser. The additional components housed in the central cavity are primarily electrical based such as DC-to-DC converters and electrical buses.

The MCP II's super structure houses the SICK laser, IEE1394 camera, and a LCD monitor. The superstructure was crafted from 8020, also known as the "Industrial Erector Set," which enabled the sensors to be angled arbitrarily until an optimal setting was discovered. Cables travel down the center of the 8020 super structure and through a small hole to the central robot's central cavity. The camera is mounted at the top of the superstructure angled downward enabling a wider viewing radius of the ground in front of the robot while the SICK laser ranger finder is mounted below the camera and at a slightly lesser angle.

Motor Subsystem

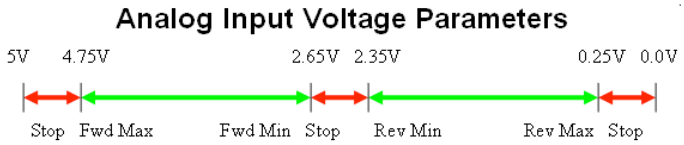
Motors

The motors chosen are XTi Hub Motors 280-1378M integrated motor and wheel assembly. This design incorporates the motor inside the hub of the wheel resulting in a compact and robust design. With the motor sealed inside the wheel, we did not need to worry about issues possible in a traditional separate motor, wheel and drive-train such as: alignment of gears, contaminates entering the drive train, lubrication, or belt slippage. These motors are designed to run from either 24 or 36 volts. We chose 36V as this resulted in a greater maximum torque and speed. The motors have a maximum no load speed of 5.7 MPH which we regulate through a dedicated CPU to 4.9MPH. At maximum power output the motors draw 10.2A and develop 28.9ft/lbs of torque at 133.7RPM which corresponds to 4.8MPH.

Controller

The motor controller used is a Courtney Electronics 600-1028M Programmable Dual 30A Motor Controller. This controller is capable of delivering 30A continuous to each motor at 36V. The controller has several modes of operation with a number of parameters, which are set via a serial connection from a separate PC. In the mode we used, "wiper mode", the output current to each motor is proportional to an analog voltage

supplied to a corresponding input pin, one for each output. This input analog voltage can range from 0V to 5V DC. We chose parameters such that from 2.35V to 2.65V the current to the corresponding motor would be zero. In this controller, a zero current actually results in a breaking action similar to shorting the output terminals of a DC motor. When the voltage is above 2.65V the current to the corresponding motor results in a forward motion. the current from a 2.66V input is the minimum forward current and the current increases proportionally with the input voltage until the maximum current at an input voltage of 4.75V. An input above this voltage is interpreted as an error and the controller reduces the current to zero. Below an input voltage of 2.35V the controller reverses the current, and the motor runs in reverse, with the lowest current at 2.34V and a maximum current at 0.25V. Below an input of 0.25V the controller indicates an error and the controller reduces the current to zero.



The controller has several programmable parameters such as acceleration, deceleration, max current surge current, surge time, and set points for the analog input voltage. These parameters can be programmed via an external PC using a program supplied by the manufacturer.

Motor Interface CPU

The motor controller accepts only analog voltages to control each motor current, while the Main CPU only communicates digitally. To interface between the two, we use the Blackfin Handy Board Robotic Controller as the Motor Interface CPU. In our application, the Handy Board accepts a serial RS232 input from the main computer and outputs a PWM signal according to the value received from the Main CPU and the present wheel speed . This PWM signal is passed through an active low pass filter which converts the signal to an analog voltage and is then sent to the motor controller.

The serial protocol between the Main CPU and the Handy Board simply consists of repeatedly sending a single byte. Each byte that is received by the Handy board is divided into the upper four bits, which correspond to the right motor speed and lower four bits which correspond to the left motor.

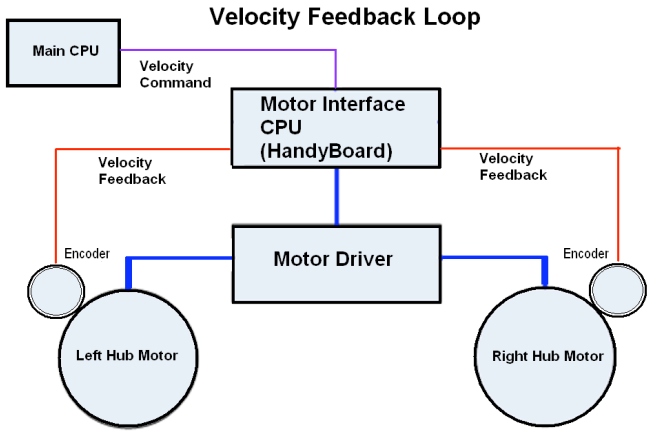
Each 4-bit value translates to the following speed values for the associated motor:

0xF = FORWARD 100%	0xB = FORWARD 48%	0x7 = FORWARD 15%	0x3 = STOP
0xE = FORWARD 85%	0xA = FORWARD 38%	0x6 = FORWARD 9%	0x2 = REVERSE 2%
0xD = FORWARD 72%	0x9 = FORWARD 29%	0x5 = FORWARD 5%	0x1 = REVERSE 5%
0xC = FORWARD 58%	0x8 = FORWARD 21%	0x4 = FORWARD 2%	0x0 = REVERSE 9%

A value of 100% corresponds to a speed of 4.9MPH, which was chosen to ensure the robot did not exceed the maximum speed of 5MPH. For example, a byte of 0xC8 would result in the left motor tuning at 2.84MPH and the right at 1.03MPH resulting in a sweeping turn to the right. While a byte of 0x25 would result in a counterclockwise pivot where each motor is turning at 0.25MPH.

Control Loop

The motor controller simply outputs a current to the motors in proportion to the voltage at the motor control input pins, with no velocity or position feedback. Through experimentation it was determined that sending at known current to the motors without some means of determining the motor speed resulted in erratic behavior. The geometry of the robot requires a large initial current to start a turn or a pivot, but once the turn is started the current needed to sustain the turn becomes much less. This resulted in oscillations when attempting to drive in a straight line, as the larger starting current would result in over shoot leading to an over correction and so on. Limiting the current to lower values resulted in a drive system that proved easier to control and possessed reduced oscillations but lacked sufficient power to drive over obstacles or transverse through soft terrain, as the current limit prevented the motors from developing enough torque. To correct this, an encoder system was added to each wheel. This gave a velocity feedback so that a velocity could be commanded by the Main CPU and then, by a feedback loop through the Handy Board, that velocity would be held steady.



The output from each encoder is read by the Handy Board, compared with the commanded velocity, and then the output voltage sent to the motor controller, which is proportional to the motor current, is adjusted as to bring the motor speed in alignment with the commanded value. The control loop uses pseudo Proportional, Derivative, and Integral (PID) feedback. Between each serial command received, the Handy Board determines the output value via the following formula:

V(next) = V(current)+P*[V(cmd)-V(enc)]-D*[V(enc)-V(enc-1)]+I*ErrorSum(n)

Where:

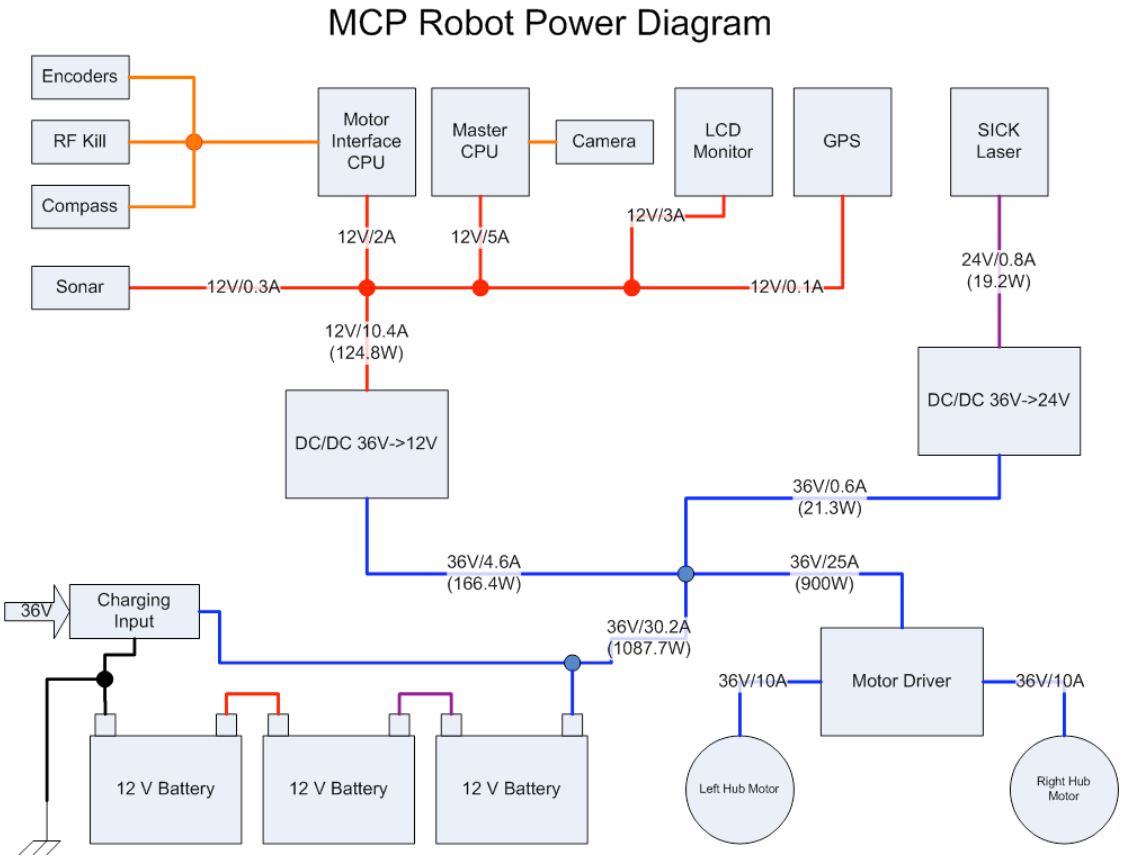
V(next) is the new voltage to be applied to the motor controller,
V(current) is the voltage currently applied to the motor controller,

P is the proportional term,
V(cmd) is the velocity commanded from the main computer,
V(enc) is the actual velocity,
D is the derivative term, and
I is the integral term,
ErrorSum is the average error(difference between commanded and actual velocities) over the last n samples multiplied by n.

Electrical Subsystem

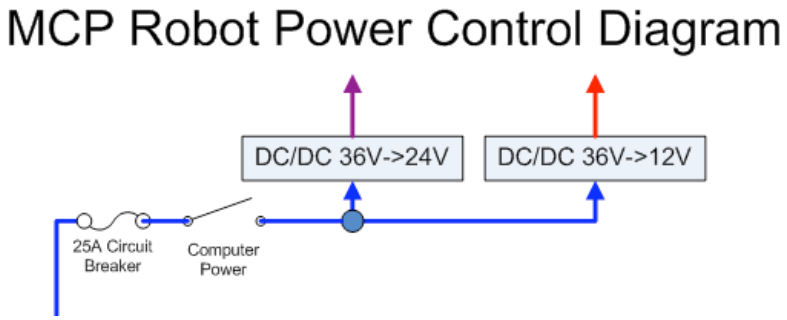
Power Distribution

The Power Diagram depicts the power distribution in the robot. Power is generated by three 12V sealed lead-acid batteries resulting in a power source of 36V, capable of supplying a continuous 99A with a total capacity of 33AH. The 36V is fed directly into the motor controller through a 25A circuit breaker, an emergency stop switch and a power switch in series for protection.The 36V is also fed, via a second 25A circuit breaker and a computer power switch to a 12V and 24V DC/DC converter. These converters supply power to the computers and sensors.



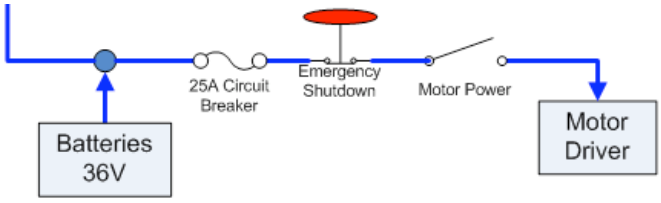
The 24 V converter is capable of supplying 2.2A at a 90% efficiency and and normally operates at 0.8A of its 2.4A maximum or 33% of capacity. It supplies current to the SICK range finding laser only.

The 12V converter is capable of supplying 12.5A at a 75% efficiency. The draw is 10.4A out of its rated 12.5A or 83% of its maximum capacity at worst case. The devices operating on 12V include: the Main CPU (ITX), the motor interface CPU



(Handy Board), the main computer monitor, the GPS, and the sonar.

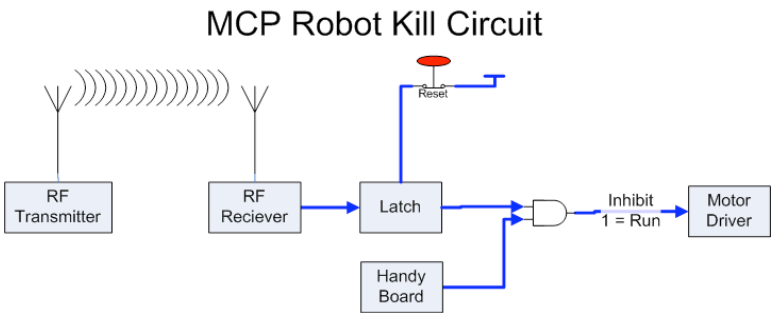
The encoders, RF Kill, and Compass are powered from an integrated 5V/5A supply on the Handy Board, while the camera is powered through a standard IEEE1394 "Firewire" interface from the Main CPU.



The total maximum draw on the batteries is just over 30A which, given the 33AH capacity, will result in at least 1 hour running time.

Emergency Stop (Kill)

Safety of bystanders, property and the robot are of up most importance. In order to prevent any injuries or damage we have implemented three ways of immediately bringing the robot to a halt: Mechanical Kill, Firmware Kill, and RF Kill. Each of these methods either cuts power to the motors or causes the motor controller to bring the motors to a stop without involving the software running on the Main CPU.



Mechanical Kill: (Above Figure) We have included a red mechanical switch on the back of the robot which, when pressed, cuts all power to the motors. This brings the robot to a quick stop.

Firmware Kill: (Left Figure) Upon power up, the inhibit signal of the motor controller is held in the inhibit state by the Handy Board controller until a valid command of 0x33, which corresponds to stop, is received over the serial port from the main computer. This prevents any spurious commands causing the robot to move or lurch during power up if the motors are inadvertently powered up before the computers are powered.

RF Kill: (Above Figure) In this method a hand held Radio transmitter sends a command to a receiver mounted on the robot which sends an "Inhibit" signal to the controller on a dedicated pin which immediately stops the robot. The inhibit signal is latched by discrete logic so that once an RF Kill signal is received, the only way to reset the controller is by a dedicated button on the robot.

Computer Subsystem

There are two computer systems running in the MCP robot. The primary system is an Intel Pentium M 1800 on a VIA mini-ITX mainboard. This design allows the computer to elegantly run off of a single 12V power line, drawing only between 2.2 - 3 amps. This system processes the input from all sensors except the encoders, and runs the MCP program.

The secondary computer system is our own Blackfin Handyboard (ref: http://www.cs.uml.edu/blackfin/index.php/Main/HomePage). This is a hand-held robot controller designed here at UML. It is based on the 600 MHz Analog Devices Blackfin® ADSP-BF537 processor, and features a wide range of sensor, motor, and low level I/O capabilities, all facilitated by a Xilinx Spartan 3e FPGA. This system receives motor control commands from the primary computer and drives the motors accordingly. While the handyboard is more than capable of processing the input from all the sensors except the Camera, we instead decided to use Andrew Chanler's Serialsense interface (ref: http://www.cs.uml.edu/~achanler/robotics/serialsense/) to convert the digital data from our sensors into a simple RS232 serial communications to the Pentium. We made this decision to simplify the serial protocol between the Handyboard and the Pentium. We wanted to avoid transmitting all sensor data over the serial line, while simultaneously receiveing motor control commands on the same line.

Sensor Subsystems

Camera

The Foculus FO124C IIDC Firewire camera is used for video acquisition through a PCI IEEE-1394 host adapter card. A Linux (edubuntu) environment is used, with the following driver setup:

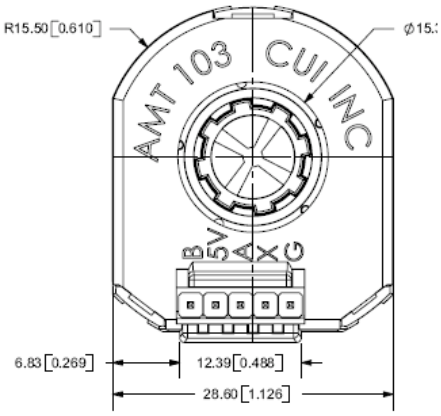
- ohci1394 - open host controller interface, driver for the IEEE-1394 host adapter.
- ieee1394 - protocol driver for IEEE-1394.
- video1394 - DMA (Direct Memory Access) to make the data from asynchronous channels on the 1394 bus available to applications. Requires: ieee1394, ohci1394.

- libdc1394 - provides high level API for controlling IEEE-1394 compliant camera. Requires video1394.
- coriander - A GUI front end for the libdc1394.

SICK Laser Rangefinder (LRF)

A SICK LMS-200 laser rangefinder (LRF) is used for obstacle avoidance in both autonomous and navigation challenge. The LRF is connected to the main computer via serial port. The LRF scans in 180 degree range, with 1 mm distance precision under our settings. The data is received by "Player" server (Ref: http://playerstage.sourceforge.net/), and sent to our Master control program via standard Player/Stage interface. Our program treats the data from LRF and Stage simulator in the same way.

Encoders



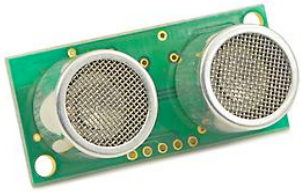
It is clear that the ability to determine the speed of the robot was relevant to controlling the robot. In order to do this, the team decided to use quadrature encoders. We chose the CUI AMT series modular encoder shown in Figure 2. The AMT VersaPak came with with on board dip switch with 16 selectable resolution options ranging from 48 to 2048 PPR. It also includes an index pulse, TTL voltage, and operates between -40 degrees Celsius and 100 degrees Celsius and the voltage consume should be between 3.6 volts to 5.5 volts direct current.The encoder is available in two mounting options: AMT102 and the AMT103. The AMT103 was used for the design since it offered a good mounting option for our design.

The critical decision however was to create a way to mount the encoders to the axle. The encoders were difficult to mount because the wheels turn on the axle and therefore the axle did not turn. It decided to mount the encoders on a smaller wheels that would be placed on the two front wheels. The smaller wheels would turn the axle that run through them. In this design the axle will turn thus enabling it to drive the encoders.

GPS

We use an OEM GPS receiver, the Garmin GPS16-HVS, to acquire the robot's geographical position (latitude/longitude) data during the navigation challenge competition. With the Wide Area Augmentation System (WAAS) enabled, the position error is less than 3 meters as specified in its document. In our testing, we measured the horizontal error to be about 1-2 meters. The GPS data is passed to the master control program via a "Player" server interface.

Sonars

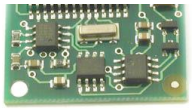


Our Devantech R93-SRF04 Ranger array operates in the Serialsense framework developed By Andrew Chanler here at UML. In its original design each Serialsense module can communicate with only one ranger, so in Spring 2005, UML alumnus Brian Corbin created a modified Serialsense multiplexer, in order to connect 8 rangers simultaneously. In order to avoid collisions between sonar signals, the multiplexer polls them one at a time. Each ranger is accurate to approximately 10 feet, and operates in approximately a 45 degree cone. The Serialsense software reads the distance signal from each ranger, and reports them back to the master control program. Each sensor is polled about once every 250ms.

The rangers are positioned around the perimeter of the robot, particularly in the back, where the SICK laser range finder is unable to see. Additionally, since the SICK laser is positioned at a downward angle, it has a blind spot directly in front of the robot. A sonar ranger is used to monitor this blind spot.

Electronic Compass

Our electronic compass is a Devantech R117-COMPASS. It uses 2 Philips KMZ51 magnetic field sensors, positioned at right angles, in order to detect the Earth's magnetic field. It has a relative resolution of .1 degrees, and is globally accurate to within 4 degrees. The compass sends a PWM signal



based on its readings, and a Serialsense board interprets that signal and sends the degree reading back to the Master Control Program. The programs for both the autonomous challenge and the navigation challenge rely heavily on this directional data.

Safety, Reliability, and Durability

When we speak of safety of the robot, we are concerned with the safety of the people and objects surrounding the robot, and specifically preventing the robot from causing any damage or injury due to a collision. In this vain, the safety features of the robot reside in three main areas: autonomous avoidance, human action, and damage minimization.

The first line of defense is the robot itself. If the robot can autonomously avoid obstacles and people, then the probability of damage or injury is greatly reduced. As is stated elsewhere in this report, the robot is equipped with a camera and a range-finding laser which, if all goes well, will prevent collisions. However, if these fail or the data is misinterpreted, the sonar system kicks in. The sonars located around the perimeter of the robot will prevent the robot from moving in any direction where an object is detected.

The second line of defense are the kill switches. These switched enable a human to stop the robot if it appears the robot is approaching an object or person. One switch is locally activated and the other remotely activated by radio signal.

Finally, areas of the robot that are likely to contact an object or person in the event of the failure of the above two safely precautions have been covered with a protective foam padding. While this will not prevent a collision, it should help mitigate any damage or injury that would occur should the robot strike an object.

The reliability and durability encompasses the ability of the robot to tolerate potential sources of damage and still function correctly and to operate correctly for a acceptable period of time without repair. In this endeavor, we have incorporated several features. The frame of the robot is box steel welded at all joints. The robot has weathered loads of two grown men (over 400 lbs.) with no damage. The two drive wheels are each capable of carrying a load of 400 lbs, and with the integrated motors are relatively impervious to contaminants. All electronics except for sensors and the RF receiver are shielded inside the robot. Heat generating components are mounted directly to the aluminum base plate while the other components are mounted on rubber standoffs for shock absorption. The sensor tower created with aluminum beams is capable of supporting many times the current load of the camera, laser and monitor.

Software Strategy

We call our main control software the Master Control Program, or MCP. The MCP uses the same fundamental decision-making process for both of the IGVC competitions (autonomous and navigation), with only a slight change in functionality differentiating the two modes, discussed below. We have developed a robust, thread-safe framework upon which each of the robot's subsystems has been built, allowing the MCP to control each one of the robot's components – from the vision system to the laser range finder to the drive controller – in an intuitive, standardized manner. Setup, fault detection and recovery, thread-safe access control, and other management functionality is abstracted away for each component, allowing the MCP thread to concern itself solely with interpretation of sensor data and decision-making. The framework we have designed further allows us to easily plug in simulated versions of each of the robot's subsystems. In order to expedite the code testing and development process, we utilize a unified simulation environment and development kit - “Player/Stage” - which was particularly useful in the early stages of development before the robot's body was completed and its hardware was operable.

Simulation and Virtualization

With "Player/Stage", we were able to start coding and testing our MCP program before the robot itself was built. Select portions of the development process would cause difficulties if more than one group members were attempting to modify the hardware or software at any given time. By developing software that can be easily integrated with a virtual environment allows for more diverse testing of the changes without interference from changes being made by other users. In the early stages of development of the MCP II, the group was faced with a decision between two simulation or virtualization environments, namely Microsoft Robotic Studio and Player/Stage.

Microsoft Robotic Studio was designed as a suite of applications which include a simple yet powerful visual programming language and a simulation engine. The visual programming language, or VPL, allows the user to program the robot “brain” in a visual manner instead of conventional programming like C or C++ without the loss of many data types and conditional statements (i.e. integers or for loops). The simulation environment allows for robust testing of the robot’s functionality without the need for hardware. Incorporated with a real-time 3-dimensional physics engine, the simulation allows for the development of realistic and functional worlds capable of simulating real world changes like light, terrain changes, different friction, and gravity (in the case of robots designed for use on other planets).

Similar to Microsoft Robotic Studio is the Player/Stage application combination which allows for robotic control and simulation on a platform independent server. Control programs are written for Player using standard C/C++ languages which make calls to drivers for each “proxy” server. The only specialized file that must be created for the Player portion of this combination is the configuration file which determines what services are to be included with the robot. In an effort to

increase the overall flexibility of Player, drivers can be fully customized for any hardware as long as it is built on top of select generic driver "classes." The Stage portion of the combination allows for the creation of a custom 2D environment where a virtual robot can be controlled to simulate and test the overall functionality of the developed software.

After reviewing each of the available options for the virtualization and simulation, the group decided to use Player/Stage for the MCP II. This was after review of the current code used in the original MCP robot, which used Player/Stage for the SICK laser range finder, while also discovering that Microsoft Robotic Studio required a version of Windows to be running on the robot. Since the robot is an embedded system, the group chose not to use a Windows platform as the primary operating system due to potential bugs, memory leaks, and other known Windows problems.

Vision System

Graph-Based Segmentation

In this vision processing method each pixel is considered a vertex in a graph. Weighted undirected edges connect these vertices. These weights are measures of dissimilarities between neighboring vertices. The image is segmented into distinct components based on color differences. A boundary between two components in a given segmentation is based on measuring the dissimilarity between elements along the boundary of the two components, relative to a measure of the dissimilarity among neighboring elements within each of the two components and is thereby adaptive with respect to the local characteristics of the data. This means that edges between two vertices in the same component should have relatively low weights, and edges between vertices in different components should have higher weights. The region comparison predicate evaluates if there is evidence for a boundary between a pair of components by checking if the difference between the components, is large relative to the internal difference within at least one of the components. A threshold function is used to control the degree to which the difference between components must be larger than minimum internal difference. The software that Amr Elbasiony developed for this purpose reports four arrays of points, indexed by their Y coordinate, representing the left track, the right track, the middle left track (if available), and the middle right track (if available).

Navigation Challenge

We use a "behavior based" algorithm to control the robot during the navigation challenge. The algorithm treats different sensor inputs separately, then combines them together based on each behavior's priority.

The Behavior categories

1) Behavior: "Go to a way point".
The robot receives its current location from GPS, and current heading from electronic compass. The target way point is selected by a greedy algorithm – The nearest unvisited way point is selected first, and after this is reached the next nearest way point is selected.

2) Behavior: "Avoid an obstacle".
The SICK laser data contains an array of distances indexed by the view angle. If any distance is less than a certain threshold, a behavior "Avoid an obstacle" is put into a behavior list. The speed is reduced if the obstacle is getting closer, and turn direction is determined by which side has a longer clear path.

3) Behavior: "Avoid going out of the field boundary"
The field boundaries are treated as infinite obstacles, and handled the same way as obstacle avoidance behavior.

Execution of Behavior

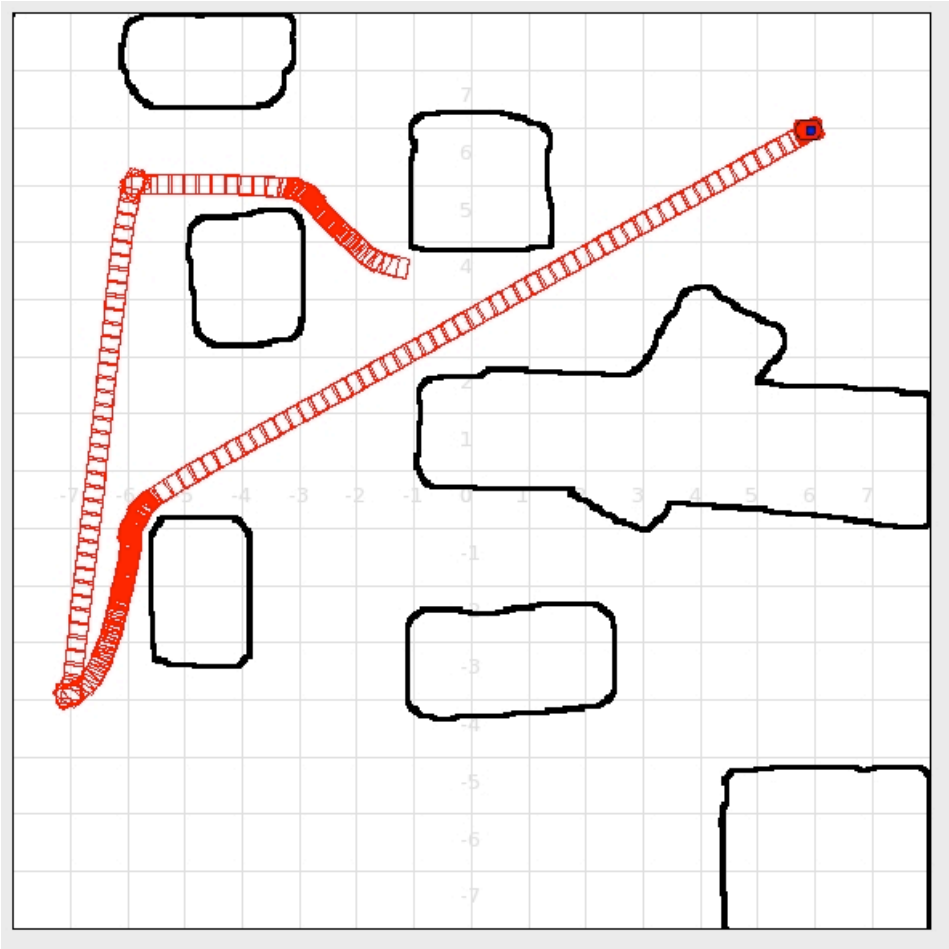
After behaviors from different sensors are generated, they are sorted by priority. The priority is based on the distance towards the way point or the obstacle, a shorter distance means a higher priority. In each control cycle, the highest priority behavior is executed. So, if an obstacle is closer than a way point ahead, the obstacle avoidance behavior is executed until the path towards the way point is clear.

Handling of GPS signal loss

Should the robot lose the GPS signal, the robot remembers the last computed bearing and distance of the next way point and attempt to head in that direction. It uses the wheel encoders together with the information from the compass to perform a very rudimentary form of dead reckoning in order to try to determine its current location. With the known location of the next way point and a best guess as to its current location, the robot is able to perform obstacle avoidance without seriously disrupting its effort to reach the next way point while no GPS signal is available.

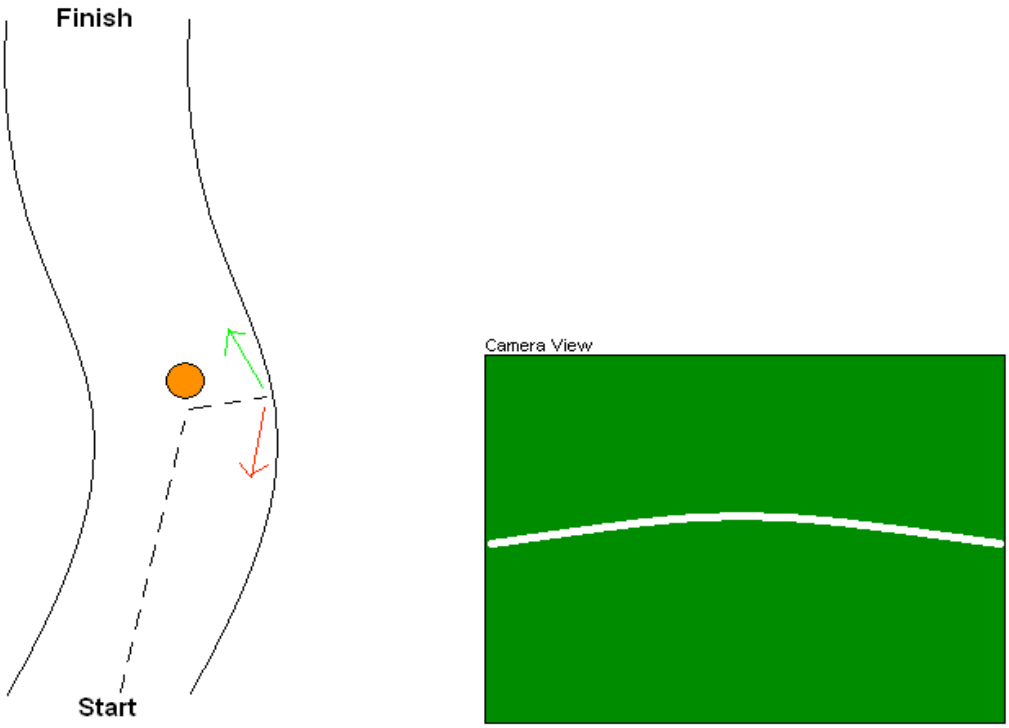
Simulation on Player/Stage

During code development, we run the program in the Player/Stage simulator. This greatly helped us to debug and optimize our software. The picture shows the trace of robot following three way points at (-6, 5), (-7, -4), (6, 6).



Autonomous Challenge

The MCP's Autonomous mode is almost identical to its Navigation mode, except that here we are not given a predetermined list of way points; instead, the camera is used to determine a continually updated "pseudo-way point." The robot's camera subsystem reports two arrays back to the MCP (among other things), representing a best guess as to the location of the left and right lines of the track for each of the 480 horizontal lines of each image reported by the camera. Given these arrays, the MCP determines and continually updates a pseudo-way point which points to the center of the track some distance up ahead. This way point always remains at the end of the MCP's list of way points. The purpose of this way point is to keep the MCP focused on completing the course, even when obstacle avoidance may result in the robot having to turn completely around. If the robot were completely reactive, without any long-term memory or goal, the robot could easily make errors like the following:



After avoiding an obstacle, when presented with what would appear to be two equally good choices (turn left or turn right), the robot could easily decide to turn right, causing it to backtrack and never complete the course. By adding a pseudo-way point, which in this case would be somewhere beyond the obstacle in the direction of the finish line, we can ensure that the MCP makes decisions that always further the goal of reaching the end of the course (in this case, causing the robot to turn left instead of right).

Obstacle Avoidance

(For Both The Autonomous And Navigation Challenges)

Because of the very different types of information each sensor subsystem reports back to the MCP, the MCP must handle the data from each subsystem in a unique manner.

Camera Data

Besides reporting left and right track line information, as explained above, the camera subsystem also reports data on objects it identifies as possible obstacles. Should the camera subsystem begin reporting information about a distant object (that is, an object above a certain horizontal cutoff in the camera's field of view), the MCP will attempt to determine if the object is an obstacle or not. If it can be identified as an obstacle with a high degree of confidence (by color, shape, etc.), then a path around that obstacle will be computed and stored in the form of additional way points (to be reached, in order, before the current way point). If it detects an object above this horizontal cutoff that it identifies as not being an obstacle (i.e. a ramp), or which cannot be definitely identified as an obstacle (i.e. a shadow or other image artifact), the MCP ignores the data for the time being. When the camera reports information about a nearby object, rather than computing new way points the robot treats the object as an "immediate consideration," explained below. Note also that the track lines nearest to the MCP are treated as "immediate considerations".

Laser Data

The laser range finder's data is first massaged into a meaningful form by considering the angle and height of the laser, and using that data to convert each of the 180 distance measurements into a (bird's-eye distance, height) pair. With this more useful data, the MCP is able to differentiate between gently sloping objects which are not obstacles, such as the ramp, and objects whose faces are nearly perpendicular to the ground and which are obstacles, such as cones and barrels. When the MCP identifies a ramp, a way point is created in the center of the ramp to ensure that the robot will safely navigate that portion of the track. When the MCP identifies a distant obstacle, way points around that obstacle may be generated, or the obstacle may be ignored for the time being, depending upon the MCP's ability to compute a path around the obstacle. Nearby obstacles are treated as "immediate considerations," explained below.

Sonar Data

Similarly to the laser range finder, the sonar data is massaged into a more meaningful form by considering the placement and angle of each sonar unit. All objects which are reported by the sonar units are treated as "immediate considerations," explained below.

"Immediate Considerations"

The primary obstacle avoidance behavior of the MCP revolves around the identification and avoidance of "immediate considerations." An immediate consideration is any nearby object, reported by any of the robot's subsystems, which has not already been cleared as a non-issue ahead of time (for example, when a ramp is identified by the laser range finder, all future reports of a potential obstacle at that location by the laser, camera, and sonar subsystems will be discarded). The robot treats immediate considerations as obstacles, and will attempt to navigate to the next way point in its list in the most direct manner possible, with consideration given to any of these nearby obstacles.

Any immediate consideration which lies in the robot's current path, whether the robot is driving straight, making a wide turn, or driving in reverse, will cause the robot to stop, turn sufficiently to ensure that it will avoid the obstacle, then attempt to navigate around the obstacle to a point where it is facing the way point, unobstructed by the obstacle. If, while attempting to navigate around such an obstacle, the MCP discovers that the robot will run into one of the track lines, the MCP will instruct the robot to turn around completely and try navigating around the obstacle in the opposite direction (counter-clockwise instead of clockwise). All such attempts at obstacle avoidance will executed at a greatly reduced speed so as to allow the camera subsystem (which updates very slowly by its nature) to remain synchronized with the situation as it develops.

Any immediate consideration which lies near to some point along the robot's current path will cause the robot to bias its movement slightly away from that obstacle so as to create a margin of safety around the robot, and will cause the robot to lower its speed slightly when nearing that obstacle.

Immediate considerations which are not projected to interfere with the robot's current path are ignored; because immediate considerations are only reacted to immediately and are never directly factored into the MCP's long-term strategy, there is no need to remember or map any of the immediate considerations which the MCP may identify. Should a previously ignored immediate consideration be determined to be in the way of a recomputed path, it will simply be reacted to as expected.

Design Process and Team Organization

The MCP II robotic design team consisted of students from the Spring 2007 Robot Design class taught by Dr. Fred Martin. Students were encouraged to be creative and choose areas of work that would build upon their interests. The design team consisted of Kenneth Dillon (undergraduate Electrical Enginee, Amr Elbasiony (graduate Computer Scientist), Chris King (undergraduate Computer Scientist), Joel Michel (graduate Computer Scientist), John O'Fallon (graduate Electrical Engineer), Nathan Palmer (graduate Computer Scientist), and Haiyang Zhang (graduate Computer Scientist).

Due to Nathan Palmer's interest in virtualization and simulation based software, Fred assigned him to research Microsoft Robotic Studios and Player-Stage in the hopes to create a virtual testing platform for the MCP II. Chris King and Joel Michel were both assigned to overall programming of the MCP II's main process due to their strong programming skills. Nathan, Chris, and Joel also helped conceive the overall structure of the robot based upon Brian Bailey's original scematic. Haiyang chose to work on the SICK laser range finder which later included the Global Positioning System (GPS) to be used in conjunction with the compass during the Navigation Challenge. Amr Elbasiony brought a vast amount of knowledge with vision based systems and chose to focus on the vision subsystem where he reviewed the the previous MCP design and designed a more efficient system capable of returning more precise data points. Kenneth Dillon and John O'fallon both chose to primarily focus on the hardware portion of the MCP II. Kenneth was responsible for developing a circuit capable of detecting the speed of the robot while assisting John with electrical systems. The duo successfully experimented with speed controllers and PIDs initially. John was also selected as the team leader where he excelled at assissting with creating an overall robotic design schedule and organizing the results of each student.

During the initial stages of the design process, the team decided that the original MCP, which was made from a Power Wheels Jeep Wrangler, was not sufficiently stable to house the contest's payload and all subsystems due to the plastic chassis, which when heavily loaded would bow causing navigation and steering to become erratic. Also the MCP used a standard car-like steering system which made it difficult to turns in tight spaces. Brian Bailey offered to design a tripod chassis made of steel to provide a strong structure that would not bow under the expected weight. The super structure had to be a light weight material and capable of pivoting individual trays for angle adjustments while also be sturdy enough to support the SICK laser range finder and LCD monitor. Initially the MCP II design was drawn on white board and paper where each member of the group took the marker to make changes or recommendations until the team selected a tripod based chassis with two drive wheels positioned in the front and a supportive rear caster wheel.

The team feels it is important to note that no formal drawings were made of the MCP II before designing and specifically we chose to avoid any used of Computer Aided Design (CAD) programs. Thus there was no specific specifications were created for the robot and the MCP II was completed knowing only the weight required to be supported and the need for flexibility and maneuverability.

The team's weekly approach to the MCP II's design is comparable to the "Agile Manifesto":

> Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the mini-increment of new functionality:Most agile methods attempt to minimize risk by developing software in short time boxes, called iterations, which typically last one to four weeks. planning, requirements analysis, design, coding,

testing, and documentation. While an iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. In many cases, software is released at the end of each iteration. This is particularly true when the software is web-based and can be released easily. Regardless, at the end of each iteration, the team reevaluates project priorities (see http://agilemanifesto.org/).

The development of the MCP II encompassed more than the software project discussed in the "Agile Manifesto" but the design and fabrication of the robot used a similar process where students were assign tasks to be completed during a given week. Each task was incremental and built upon the results of previous tasks while culminating in the MCP II. This approach was deemed the most reliable to minimize the amount of design errors while also allowing the team to save time by reducing the need for redesigning and rebuilding.

The major milestones in robot development included:

- Exploration of simulation environments including Player/Stage and Microsoft Robot Studio (February 2007)
- Exploration of PID motor control with Gamoto motor controller (February 2007)
- Acquisition of Hub motors and testing on bench (early March 2007)
- Chassis built, motors installed, batteries installed, and could be driven manually (March 24, 2007)
- PWM filter and Handy Board control (early April 2007)
- Sonar subsystem and compass from MCP I revived (April 2007)
- Superstructure built with 80/20; camera, monitor mounted (late April 2007)
- DC-DC converters acquired and mounted; computer mounted in robot (early May 2007)
- Use of a Wiki to coordinate group actions, and docs.google.com to write this design report by multi-people concurrently (May 2007)

Throughout the development of the MCP II, all of the robotics code was stored on a Subversion (SVN) server which enabled multiple individuals to modify portions of the code, say the motor drivers, without interfering with other group members developing other portions, say image processing. Upon the creation of the SVN server an initial "check-in" is required to build the current 'trunk' of code and when a user wishes to modify a part of the code they can either "check-out" all or a select part of the 'trunk'. Once the code has been modified and tested successfully by a group member that member can "commit" changes to the trunk which will either merge or replace the current file with the user modified file. However if a problem was later discovered in the modified code SVN servers allow changes to be "rolled back" to a previous state before that error entered into the trunk. Additionally if any file is modified by two users simultaneously and the files cannot be merged together a flag is noted about the conflict which will require the user to manually review each conflict and determine the course of action.

Design Innovation

Compared to our previous vehicle (MCP I), this version (MCP II) of the robot has been completely redesigned from the chassis and motors to the control program. Our innovations include:

1. The Blackfin Handyboard is a general purpose robot controller board developed by members the UML Engaging Computing lab. It can accept multiple analog and digital inputs and generate multiple PWM motor and servo control signals. It was designed as a replacement to the original Handyboard (www.handyboard.com), which was designed by Professor Martin, and became a popular robotics education platform. The Blackfin Handyboard has the potential to revolutionize the way robotics is taught. Collaborating with Analog Devices and National Instruments, we developed a Labview package for rapid development. Additionally, it is capable of running uClinux (www.uclinux.org), so development with the gcc tool chain is possible. In our robot, the HandyBoard is running a motor control feedback loop, using the motor encoder data to maintain a specific speed. It receives commands from the Pentium over an RS232 serial line, and send signals to the motor controler. The HandyBoard allows us to integrate the motor control, remote kill switch, and encoder data processing together, without multiple different circuit boards.
2. The Serialsense boards (www.cs.uml.edu/~achanler/robotics/serialsense) were also designed here at the UML Engaging Computing Lab. They provide a very simple interface for data aquisition from sensors that have complex digital signals. We are also using a modified Serialsense multiplexer to receive data from up to 8 sonar rangers simultaneously.
3. We used a variety of tools and materials to facilitate rapid prototyping. We are the first IGVC team who use a Trotec Laser Cutter to make plastic parts. The laser cutter can quickly and precisely cut acrylic into complex shapes, which would be impossible with a mechanical saw. Also, we used 80/20 aluminum framing (industrialerectorset.com) to build the superstructure of the robot. The frames are bolted together, so the superstructure can be easily reconfigured.
4. The MCP's chassis is made of steel frame and welded together into one piece. It provides a solid support for other components. Additionally, the top cover of the robot is suspended by a pneumatic piston capable of supporting the weight of the entire superstructure. This allows the cover to function like a car hood, making access to the internals of the robot extremely easy.
5. We used XTi Hub Motors 280-1378M (www.hubmotors.com) integrated motor and wheel assembly, in order to save space that is traditionally occupied by a bulky cylindrical electric motor's profile. These motors have exceeded our expectations in every way.
6. The robot remote kill switch is adapted from a wireless door bell. The wireless signal contains a 7-bit code, which only

activates the receiver on our robot without interfering with others.
7. Our graph based image segmentation algorithm is designed to identify the lines of the track and obstacles in the autonomous challenge. Because its specifically tailored for our needs, it will perform reliably on the IGVC course.

Systems Integration

During the early stages of the MCP II development, the group determined that the primary focus should be the fabrication of the chassis and testing of the motors. After the chassis had been created by Brian Bailey, the motor controller was mounted to the internal compartment and was connected to a standard dual axis PC game controller. Using the PC controller, the group was able to perform multiple stress tests on the chassis and motors to determine the overall stability, optimal driving direction, and motor configuration.

After the initial motor tests were completed the next stage of development for the drive system required the addition of a Pulse Width Modulation (PWM) to voltage circuit which would be controlled via the Handy Board. The PWM-to-voltage circuit was designed to convert output from the Handy Board's function calls into a voltage that could be interpreted by the motor controller. In parallel to the creation and testing of the PWM-to-voltage circuit, the Handy Board needed to be programmed to receive serial commands which would then be passed along to the motor controller. This part of the motor development was able to be tested and verified using a terminal emulator running on a separate PC.

While the motor systems were being developed the remainder of the group was developing the software drivers for each of the MCP II's subsystems. The primary CPU was situated on a lab bench with connections running to each subsystem. Once the chassis was fully tested and the MCP II's power system fully stabilized, the main CPU was mounted into the internal compartment with the Handy Board and motor controller.

With all the primary components mounted onto the robot, the group decided it was time to give the MCP II a manually driven test run of the subsystems. The MCP II was driven to a nearby field and tennis court where the vision subsystem was able to decipher between the grass and other objects to detect primary white lines that would serve as the guiding lines during the Autonomous Challenge. The test was performed toward the end of the day when light was fading to determine the overall functionality of the vision processing in low light situations.

After testing the vision processing subsystem the group's overall focus shifted toward the SICK laser range finder. Most of the SICK functionality is based upon the code used by the original MCP with modifications being made to how the data would be transferred to the control program. With the SICK laser properly mounted on the MCP II's chassis it was time for another test run on the field, but this run was designed to test the SICK laser and determine at what distance the MCP II should begin its avoidance procedures.

Now it was time for the group to work on pulling all the pieces together in the Master Control Program. This program was designed to receive data from the vision processing subsystem and the SICK laser subsystem while issuing commands to the Handy Board and motor subsystem via a serial connection. Initially, the group decided that the control program should focus on driving toward the midpoint of the line segments returned by the vision subsystem and this focus should only be changed when the SICK subsystem registered an object within the expected field of travel. Once this base level of the control program had been achieved more advanced reasoning and process threading was added that allows the MCP II to better follow the lines of the course while staying safely away from any obstacles.

Appendix: List of Components including Cost Estimate

Component	**Total Value (USD)** **(* = donated; # = scavenged)**
SICK LMS-200 laser ranger	6000 *
FOculus FO124C Firewire camera	1000 *
Mini-ITX mainboard & Pentium M processor	500 *
Dell LCD monitor	100 #
Garmin GPS16-HVS	100
36v hub motors (two)	500
12v lead acid batteries (three)	200
Courtney "Dual 30" PWM motor controller	100
36v battery charger & cable	250
Blackfin Handy Board robot controller	500 *
80/20 building system and steel tubing	400

doorbell for wireless E-stop	14
36v-12v DC-DC converter	100
36v-24v DC-DC converter	100
miscellaneous electronic components	300