Junior Design Team Odyssey
Final Design Report

Matthew Godman, Matthew Landavazo, Vinny Ravindran, Nicole Sims

Electrical Engineering Department,
New Mexico Institute of Mining and Technology

9 May, 2011

Contents

List of Figures

Abstract

There is often a need to scour an area without human aid. The reasons for this range from a simple convenience to a complete lack of feasibility (e.g. hazardous environments for human beings.) A remote-controlled robotic solution to such a problem may be common, but a more robust solution would require a robot to think and reason on its own, independently of a human operator. The paper details the design and testing of one such autonomous mobile robot for the purpose of navigating a predetermined outdoor course and identifying objects at specific waypoints in the course. (refer to figure 1) The heart of this particular robotic system is the MC9S12 microcontroller, which relies on a vast sensor suite in order to consistently obtain feedback regarding its surroundings and make ideal judgments on how to proceed, navigate the course, and accomplish its objectives. To describe the design process, the essential requirements for a self-sufficient robot are first described, followed by rigorous detail and analysis of the devices used to accomplish the specific project objectives. Design testing revealed a robot which could impressively avoid obstacles and reach specific destinations with few shortcomings. A more robust object detection system has yet to be implemented, but with these results it seems that robotic autonomy is well within humanitys scientific grasp.

1 Introduction

There are often reasons to develop robotic systems to remotely search areas for objects without requiring a human operator. For example, a search and rescue robot could make its way through dangerous environments in order to find the location of a survivor that needs help, and feed back location data to a rescue team that can actually help the endangered person. A more efficient system can no doubt be realized if the robotic side of the system was capable of functioning autonomously. This project explores a more primitive incarnation of such a robotic application. Although the following design criteria might seem much more trivial, it is important to note that it could well be the foundation for a more intricate search and rescue system like the one briefly described above.

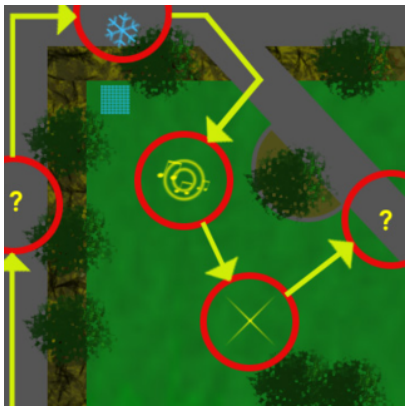

Figure 1: Obstacle Course Map

The task at hand in this case was to develop an autonomous robotic system which was capable of navigating a predetermined outdoor course. Five different areas (waypoints) on the course were highlighted by red spray-painted circles of a 3 meter radius. A variety of objects were prescribed, each with different identifying characteristics: temperature, high color contrast, ultrasonic emission, etc. It was specified that one object would be inside of each waypoint. As such it the robot was required to have the capability to make its way to each of the course waypoints, systematically seek out an object in each waypoint, and provide some kind of easily recognizable indication whenever an object was found. There are a variety of design challenges involved with the solution to such a problem. This paper details our teams solution of the design problem and the ways in which those challenges were tackled.

To put things simply, robotic autonomy requires that system can function on its own, independent of a human operator. Two primary elements are required for such a system, from an electrical engineering perspective. One is a control unit, or microcontroller, to serve as the brains of the robot. A suite of sensors are also required in order for the robot to obtain feedback data regarding its surroundings. By feeding this data back to the microcontroller and making use of programming algorithms, it is possible to program a degree of intelligence into a robot. Consideration also needs to be given to the operating environment of the robot. Since this particular project involved an outdoor area, strong consideration had to be given about the requirements of the mechanical chassis which housed the aforementioned electronics.

The rest of the paper fleshes out each of these basic elements a bit more deeply, starting with the reasons behind the choice of the microcontroller and continuing on into the individual elements of the sensor suite which make autonomous robotic navigation possible.

2 Hardware Design

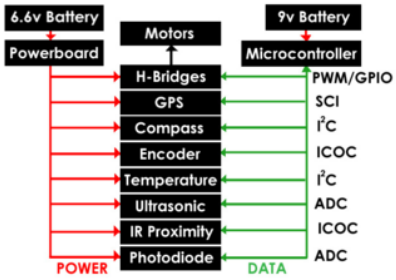

Figure 2: Block Diagram of System Design

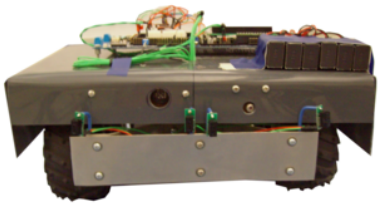

Figure 3: Odysseus

2.1 Microcontroller

Wytecs Motorola / Freescale MC9S12 has proven to be a very powerful development tool in the past microcontroller class (EE 308) at New Mexico Tech. Since this microcontroller meets the needs of the project requirements and the project team is intimately familiar with the processor and the subsystems that come with the development kit, we felt that this was an obvious pick for our control unit.

The MC9S12 has a broad range of interfacing options that are extremely valuable for our project. When devices were being considered at the beginning of the project, we rarely encountered a device that was incompatible with the MC9S12s multiple interfacing options. Even 3.3v devices can be connected because of the convenient threshold voltages on the microcontrollers logic inputs (2.5v minimum for a logical high).

Programming can be done in C, assembly, or both. The C language is a common and easy to use language. Example code is easy to find for C, so if we found any code that has already been written for any of the devices used, it would help reduce the amount of work needed in terms of interfacing.

Debugging is very simple and easy to do on this microcontroller. There is a DBug12 monitor that communicates to the terminal for terminal debugging. For example, reading and writing to memory locations, reading the condition code registers, stack pointer, and accumulator registers, setting break points, printing messages and so on. There are also some useful display options on the development board. There is a logic probe, LCD display, led array, and a seven-segment led display that proved to be more helpful for certain debugging situations.

2.2 H-Bridges/Motors/Battery

At the beginning of the the design coarse, each team was given a basic platform, from the instructors to start their robot foundation from. This includes an aluminum chassis with four DC motors, two H-bridges, and a battery. Each component is briefly summarised.

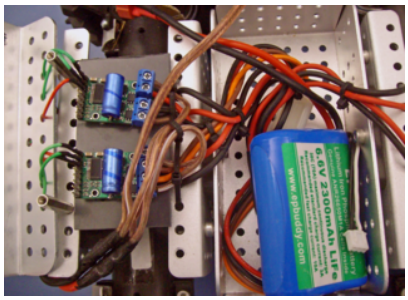

Figure 4: H-Bridge and Motor Connected to Battery

The Dagu Wild Thumper 4WD all-terrain chassis was ordered from Pololu and came with the motors, wheels, and some mounting hardware. [1]

The discrete MOSFET H-bridge (18v15) is a high-power motor driver that was also ordered from Pololu and came with a filter capacitor and headers. This device enables us to control the current flow to the motors via a PWM channel on the microcontroller, and the direction of the the motor spin with general purpose I/O lines. [2]

The power for the robot comes from the lithium iron phosphate battery purchased from EP BUDDY. This battery operates around 6.6v and has a 2300mAh life. This will give the robot the ability to run between 30 minuets and 1.5 hours, depending on how many of the current subsystems are implemented. [3]

2.3 GPS

For an outdoor robot application, our team decided that a GPS would make our overall system more diverse and robust. In our case, the GPS would mainly be used for acquiring an absolute position fix, correcting the desired heading, and correcting the dead reckoning measurements.

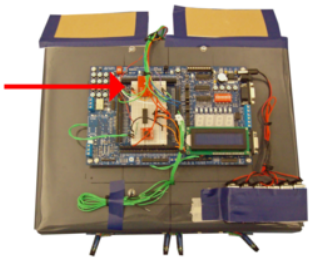

Figure 5: GPS Connected to Microcontroller Breadboard

The particular GPS that we chose (the VENUS634FLPx) is a 65 channel, low power, small form factor device. [4]

Since this module supports active and passive antennas, we were able to connect a 30dB active antenna to boost the signal strength. Initially we were concerned about not being able to acquire enough satellite locks on the course, but after testing, we found that we could easily maintain a minimum of seven satellite locks anywhere on the course. Also, after the GPS had time to warm up, the accuracy of the position readings were within a few meters (3 - 5 meters, depending on the number of satellite locks, and how long the GPS had been running). There is some internal averaging that the GPS does that improves position accuracy.

Unfortunately, we did not have enough time to integrate the GPS system into the final design. Given more time this system could easily be integrated to greatly improve the diversity and robustness of the navigation system.

2.4 Compass

We chose to use the Honeywell HMC6352 compass as our go to sensor for obtaining headings and obtaining them quickly. This sensor proved to be an invaluable asset to our dead reckoning system. This unit has many desirable characteristics. To start, its operational voltages are very well suited to our application and in place power systems, 2.7 to 5.2V. It uses a 100KHz I2C bus as a slave. Includes commands for calibration, internal averaging, and can acquire headings on demand or by polling. Another nice advertised feature of this compass is that it can be used in strong magnetic field environments.

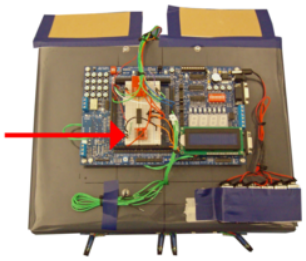

Figure 6: Digital Compass Connected to Microcontroller Breadboard

One feature that is lacking from this compass is tilt compensation. Tilt compensated compasses were out of the budget of this project and this compass was easily the most fully featured compass at our disposal to get the job done.

This compass provides up to a tenth degree resolution, outputted in a 12-bit format over two bytes of I2C. We initially set the compass to average 8 headings internally before sending us our data. In software we then took the 8 most significant bits, which virtually eliminated all and any jitter associated with the measurements.

2.5 Infrared Thermometer

The thermometer was intended for measuring the temperatures of objects at the waypoints. At waypoint two there is a very cold object, and our team

thought it would be nice to determine the temperature of that object as a form verification. The thermometer that we found and tested was an IR temperature sensor, the MLX90614, that is capable of measuring ambient and object temperature. [5]

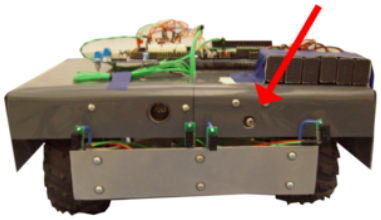

Figure 7: Mounted Infrared Thermometer

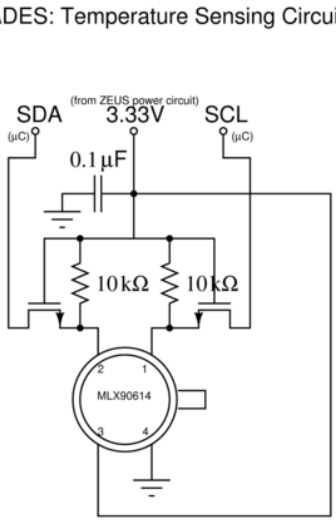

Figure 8: Infrared Thermometer Circuit Diagram

This device has a very low power consumption and interfaces to the microcontroller via the I2C bus. The data acquisition very much like the compass and can send relevant data on demand.

Testing showed that the thermometer had a range of a couple of feet indoors and gave reliable readings, but once we moved outside, the data became unusable. We are sure that the cause of the loss of accuracy is due to the sun emitting so much IR interference that thermometer becomes unusable.

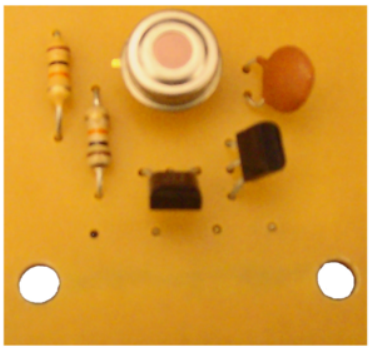

Figure 9: Infrared Thermometer on PCB

This device was not integrated into the final design for obvious reasons (it was unusable for our case).

2.6 Photodiodes

Each of the waypoints was marked by a red 5 meter diameter circle that the robot needed to enter in order to find the object or location. The initial idea for the photodiode was to use it to measure the visible light waves being reflected off of the sidewalk and grass through a voltage output. Since the grass and sidewalk are reflecting different visible light waves compared to the red line, the plan was to use the difference in the voltage output to determine when the red line was crossed over.

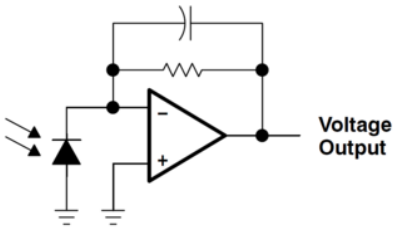

Figure 10: TSL257 Photodiode Circuit Diagram

The main source of error for both the TSL257 High-Sensitivity Light-to-Voltage Converter [8] was that it was too sensitive to ambient light and immediately saturated when outdoors making the output voltage unusable for data acquisition. The circuitry required to change the gain of the TSL257 was built within the component (see figure 10) making it difficult to change the sensitivity of the component to function reliably in ambient light.

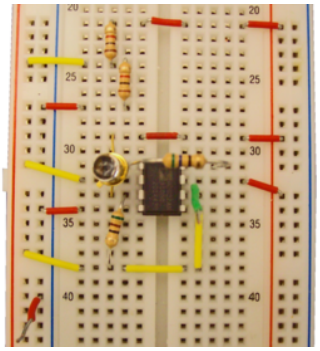

Figure 11: Infrared Sensitive Photodiode on Breadboard

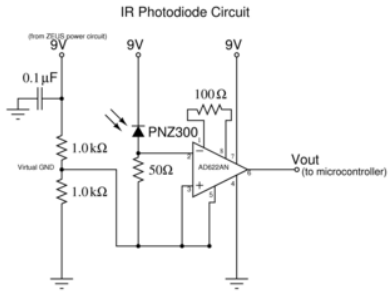

Figure 12: Infrared Sensitive Photodiode Circuit Diagram

For the PNZ300 Silicon PIN Photodiode [9] viewed the infrared spectrum of light making it more reliable in detecting that red line when crossed because red visible light is infrared on the light spectrum. (circuit diagram given in figure 12) The two problems that arose for the PNZ300 was that it relied on some amounts of light to detect the waves being reflected. To create some light for the photodiode, a SSF-LXH103UWC [10] was added, but was unable to emit the amount white light required for the photodiode. The other problem was that the PNZ300 needed to be closer to the ground for accurate readings which meant it would be in a hazard zone for when the robot entered uneven terrain. These two problems made the PNZ300 unusable because it could not accurately given data with the variation of light and it could not handle the outdoor terrain.

2.7 Ultrasonic Receiver

One of the waypoints contained an ultrasonic transmitter which the robot was intended to detect. The transmitting device, provided by the course instructors, consisted of an array of ultrasonic transducers which radiated a 40kHz ultrasonic signal into a reflector. The reflector had the effect of radiating a strong, uniform 40kHz signal in all directions in the plane of the robots height.

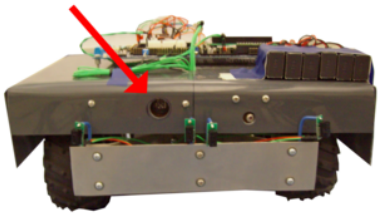

Figure 13: Mounted Ultrasonic Receiver

In order for the robot to effectively react to the ultrasonic source point, ultrasonic receiver circuitry was built in house. This not only allowed for precise control over the range at which the 40kHz signal could be detected, but served as a means to cut cost as well. The resulting circuit design is given below:

The receiver circuit makes use of a 40kHz transducer, the MaxSonar MB1100. The AD622 instrumentation amplifiers are an obvious choice in order to increase the small transduced signal into something which is easy to work with, electronically. This

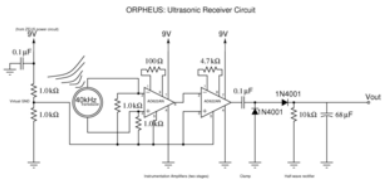

Figure 14: Ultrasonic Receiver Circuit Diagram

particular instrumentation amplifier was chosen for convenience and its low cost.

Figure 15: Ultrasonic Receiver Block Diagram

The circuit (see figure 14, or the block diagram given in figure 15) relies on a virtual ground configuration in order to emulate negative voltages, since dual rails need to be supplied for this particular chip. The gain of the first amplification stage is is actually not enough amplification for the rest of the system to reliably detect an ultrasonic signature. However, this is a decent amount of gain which the amplifier can provide reliably. Although the AD622AN is capable of supplying a gain of 1000, attempting such a high gain while operating at ultrasonic frequencies results in slewed output which is useless for the robots purposes. To circumvent this, the second gain stage is added with the second AD622. Ultimately then, the gain of the entire circuit is dependent on the resistor between pins 1 & 8 of the second AD622. Using a value of 4.7k ohms as given in the schematic, the amplifiers add a total gain of 38 dB to the incoming signal. The amplifiers are followed with a clamp and half-wave rectifier for generating a DC level from the magnitude of the AC ultrasonic signal.

This allows for a maximum ultrasonic signal detection (i.e. a DC level of roughly 4.5V) at most about 3 feet away from the transmitter. If the receiver was placed further away, a weaker signal would be received, but it would be possible to align the robot in a direction of maximum signal strength in order to systematically hone in on the ultrasonic transmitter.

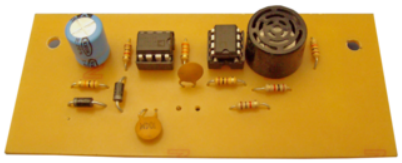

Figure 16: Ultrasonic Receiver on PCB

An algorithm has not yet been programmed into the robot, but to do so would pose little challenge.

2.8 Wheel Encoders

A wheel encoder is one of the devices we used to give feedback to our controller in order to have a true motor control system. Since it was unnecessary to have a fully functioning motor controller we used the encoder to measure the relative distance traveled.

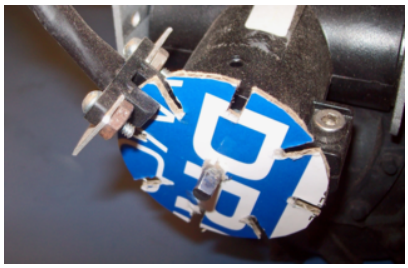

Figure 17: Encoder

In order to simplify wire management, the circuitry for the IR encoder was shrink-wrapped into the base of the device since it was simple enough. This allowed us to reduce the wiring to power and data. Two encoders were installed on the rear motors in order to improve redundancy and easy of expandability. The encoder wheels were simple cardboard cutouts that were glued to the axle of the motors, as shown in figure 17. [6]

There was some experimentation with a fuzzy logic motor control system early in the semester, but due to time constraints and the unexpected complexity of integration, we were forced to simplify our navigation problems buy operating around fixed

speeds.

2.9 Infrared Proximity Detection

Outdoor environments are unpredictable and pose many challenges that need to be considered and overcome when designing an autonomous robot. One of the major challenges is being able to detect possible walls or cliffs in order to avoid them and find a safe path to travel. We felt it was necessary that the object detection and avoidance system would be a high priority.

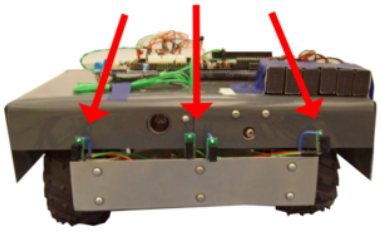

Figure 18: Mounted Proximity Sensors

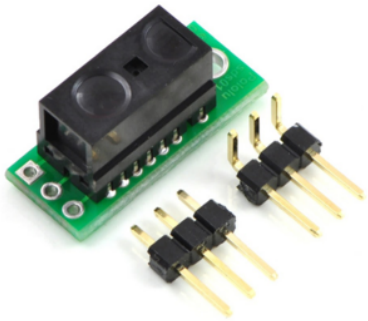

Figure 19: Digital Infrared Proximity Detector

The proximity sensors that we thought would be best suited for this system were IR proximity sensors. [7] These particular ones have a vary narrow beam width, a range of 10cm, and are resilient to ambient light. This gave us the ability to have an array of sensors that could tell the controller where an object was encountered; and as a result, the controller would be able to determine how to avoid the encountered object.

The proximity detection and avoidance algorithms were successfully integrated into our final product. In fact, the collision avoidance algorithm that our team developed, on our own, was very simple and worked surprisingly well. During field testing, we tried to get our robot stuck in all sorts of situations. The only problem that could not be overcome in a timely manner was getting stuck in a corner, but this was not a problem since there would be no corner encounters.

2.10 Powerboard

The powerboard was created to supply power to all subsystems of the robot ranging from the sensors to the MC9S12. It was designed to take the input of the EP Buddy 6.6v battery through a LM3578 switching regulator [11] that would output around 10v to power the MC9S12 (block diagram given in figure 20). A few difficulties arose when designing the powerboard causing 6 revisions of the board and consuming valuable time.

Figure 20: Powerboard Block Diagram

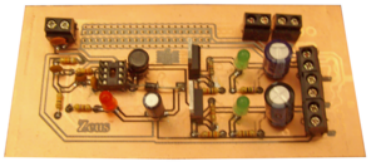

Figure 21: Powerboard on PCB

One problem that arose was the board functioned correctly when tested separately, but caused a cur-

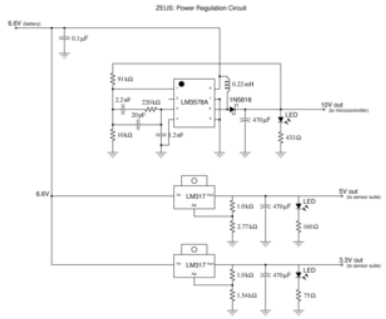

Figure 22: Powerboard Circuit Diagram

rent overload on the power supply box when combined and tested together. This problem was overcome with the realization that the powerboard was pulling more current than the power supply box could supply causing a brief current overload. When the powerboard was used to power the sensors, it functioned perfectly outputting the correct voltages to power subsystems.

Another problem was the etching process to create a printed circuit board was unreliable for large complex circuits because the traces had a tendency to not attach to the copper board. In order to etch the board, traces had to be drawn in which turned out to be a tedious task. Drilling also was a problem because it was difficult to drill the holes perfectly from components. To overcome these problems, the process of milling a circuit board and having a machine drill the holes and create the traces was used and proved to be extremely effective in saving time.

A problem towards the end of the project the inductor in the switching regulator could not handle the current when all the subsystems were connected and would burn out almost immediately after power was connected. To overcome the problem, the switching regulator was not used to power the MC9S12 and instead, 9v batteries in parallel were used to power the HC9S12 while the linear regulators from the powerboard supplied power to the other subsystems without the help of the switching regulator.

The powerboard was the largest set back within the project taking up 4 weeks of time that could have been spent elsewhere developing the other sensor subsystems. If given the option, the 9v batteries in parallel would have been used to power the MC9S12 a lot sooner allowing time for other sensor subsystems to be tested and implemented in the final design, shown in figure 22 (photo of the board given in figure 21).

2.11 Power Budget

A table is given below (in figure 23) which shows the current draw for every single component which has thus far been implemented on the robot.

Subsystems	Total Current (Amps)
Temperature Module:	0.0015
Ultrasonic Module:	0.018
Power Module:	0.016
Navigation Module:	0.15
Microcontroller:	0.2
Drive Module:	26.4
Total Current:	26.78

Figure 23: Table of Power Cost

As can be seen, the primary source of power dissipation lies in the motors. The sensor suite together only pulls about one amp altogether, while each motor on the robot is capable of pulling 6.6A when moving forward at full speed. Such figures presented here are a worst-case scenario, as the algorithms implemented thus far on the robot do not actually ever call on the motors to drive at such strenuous speeds. It is worth noting however that the battery can sustain a 70A current draw for a very short period of time, which is far more than the projected worst case described here. During typical operation it was observed that testing of the robot could go uninterrupted for about 2 hours without needing to recharge the battery. Since the robot typically runs the outdoor course in 5-6 minutes, battery life is far from an issue.

2.12 Chassis Construction

Our robot chassis served the function of a support structure for the mounting of all subsystem components, as well as providing a protective barrier

against collisions and sunlight. Due to the low budget and time constraints, we had to find a cheap and quick solution to this problem. Luckily, there was a lot of support from Norton Euart from the Workman shop. A lot of aluminum and plastic parts fabricated for the robot chassis were salvaged from the scrap material in the shop.

The brackets used to mount the wheel encoders were made from scrap aluminum sheets that were cut drilled and bent to the desired dimensions. This bracket was secured to the motor enclosure bolts near the axle on both rear wheels.

A front bumper, constructed from an aluminum alloy sheet, was used to help protect the robot from head on collisions as well as providing mounting from the IR proximity detectors. The bumper was attached to the front end of the original robot chassis.

An outer shell was constructed from a plastic sheet that was cut to the desired dimension and applied a heat gun to fold the edges. This shell was used to mount the microcontroller, ultrasonic receiver, and IR thermometer. The shell worked out great because it kept all the wires and components well isolated from encountered obstacles, for example, bushes and rollovers. The microcontroller was not mounted underneath the shell during testing, but when the robot is complete the microcontroller could be moved underneath using the same mounting holes.

3 Software Design

Approaching the software for the HCS12 microcontroller was an exercise that closely mirrored the skills we learned in our 308 microcontrollers class. We used the Freescale Codewarrior Suite and its included libraries, as well as other functions provided to us in that class for our particular microcontroller. All of our programming was done in C, and a lot of our programming was made to be debugging friendly and intuitive.

3.1 Interfacing and Integration

The first thing we did was interface each sensor individually and make simple programs to test their functionality. When we had all the sensors figured out we worked toward integration. In order to make it easier for our data to be pulled in simultaneously and for our functions to be able to use all the data and use it as soon as it is available we relied heavily on interrupts. An interrupt is an external event that causes the CPU to save what it is doing, set it aside, and tend to something else. In this manner a function can work off the most current set of data at all times. An interrupt service routine may tend to a sensor, retrieve up to date data and when the CPU returns to what it was doing it has the latest data ready for action.

3.2 The Real Time Interrupt

Our software relies heavily on what is known as the realtime interrupt. The realtime interrupt is an interrupt that occurs on regular intervals, for example, every 5ms. The easiest example of how we used the realtime interrupt is with our compass. Everytime we have a realtime interrupt occur we poll our compass for new heading data. The new heading data is then stored into a register (PORTB) so that all of our other functions can always reference PORTB for the most up to date heading information.

Another example of how we used the realtime interrupt is to set up time delays. There are times when you would like for the program to stall for a certain amount of time before moving on. For example, you can your wheels run for set duration. We did this by creating a global variable that holds a count of how many realtime interrupts you would like to pass before moving on. Then in the realtime interrupt you just decrement the count, you move on once the count reaches zero.

In motor control, you do not want your motors to just jolt on and off instantly, this will cause intense acceleration and put heavy amounts of stress on the chassis, wheels and any components on the robot. We simply created global variables that are the requested duty cycles. In the realtime interrupt, there is a routine that checks if the actual duty cycles are

at the requested cycles. If not, it makes a linear transition to that duty cycle by incrementing or decrementing the registers that control PWM duty cycle.

3.3 Timer Input Capture Output Compare (TICOC)

Input capture is another way of looking for data. To put it simply, input capture looks for a control signal to go high or low, particularly the transition of a falling edge or a rising edge. We use the input capture and its interrupt to work with our wheel encoders and our forward mounted proximity sensors.

Our proximity sensors have very simple behavior, if an object comes within 10cm of the sensor, it changes logic levels. We used input capture in order to use this change in logic level to immediately cater to this event. It is fairly obvious the event we were looking for was when our robot was about to run into something, in which case, the first thing you want to do is stop. The microcontroller looks for both rising and falling edges, in other words, looking for when an object comes into view and then goes out of view. We set global variables we called flags, and they would be set true if something was obstructing a path or set false if there were no obstructions. With this information set as global variables, now any function can monitor them and make decisions accordingly.

Our wheel encoders work in a similar manner, the logic level changes as the IR receiver is able to see the transmitter on the other side of our encoder wheel. In this sense we can use input capture to keep a count of how many times the sensor comes in and out of view. By counting how much the wheel goes by we can effectively count our distance. In our software we created a scheme so that all functions could track how far the desired distance to move is. We did this with global variables that show how far our robot still has to go. As the robot travels the interrupts generated by the wheel encoders decrement this count.

3.4 The Big Picture

We wanted to make all of our movement functions tied to the same parameters that would be stored globally, that way, any function, such as those run in interrupts could change them, allowing the way our robot reacts to new data to be very intricate. The things we store globally are:

Wait time - How long a program is waiting for, decrements in realtime interrupt.

Requested duty cycles, left and right - Sets the speed of the motors, adjusts in realtime interrupt.

Requested Distance - How far to travel, this decrements as the wheels spin, uses input capture interrupt.

Proximity Flags - Flags when objects have or have not triggered proximity sensors, input capture interrupt.

Compass Heading - The compass gets polled and the robots current heading is always stored and kept up to date, this is done in the realtime interrupt.

To make coding for our robot simple we created many simple functions to control movement, speed, and direction. We used a bottom up approach to building movement commands, creating simple ones at first and them putting them together to create more intricate maneuvers.

For example,

DIR(left, right)

This function simply changes the direction of the tires, a 1 for forward and a 0 for reverse.

DUTY(left,right)

This function changes the duty cycle for the wheels, this function looks for values between 0 and 255, 255 being maximum speed

WAIT(intervals)

This function waits for the desired number of realtime interrupts to pass

Early in our development, while we were still waiting for sensor integration to take place, we used time based motion. You can see how easy it would be create some movement commands like this, for example, move forward at half speed for 100 realtime interrupts.

FORWARD_DURATION(speed, time)
DIR(1,1)
set wheels forward
DUTY(speed,speed)
set the requested speed
WAIT(time)

wait for the time to elapse
DUTY(0,0)
stop

This same function can also be created so that the robot travels a certain distance before stopping instead of travelling for a certain amount of time.

We created many different movement functions in this manner. As we got more of our sensors integrated, we could make the functions more robust. For example, in order to keep a straight heading, before moving forward the function can look at PORTB to see what way the robot is facing. As the robot moves forward, it can keep looking at PORTB, which always has the most up to date heading. If the function detects that the robot is veering left our or right from the original heading, it can adjust the duty cycles to compensate and get the robot back on course.

If the forward proximity sensors detect an object while the robot is moving forward, they can execute an instant stop in an interrupt. The interrupt service routine can buffer all the global variables, set the duty cycles to zero to instantaneously stop, and then execute its own movement commands to try to move out of the way of the object or to move around it. When it is done it can simply take the values from the buffer, put them back in to the global variables and return. When the program returns the system is back in the state it was before the collision avoidance was triggered. Even though the robot is now stopped, the realtime interrupt will return the duty cycles back to their old requested values.

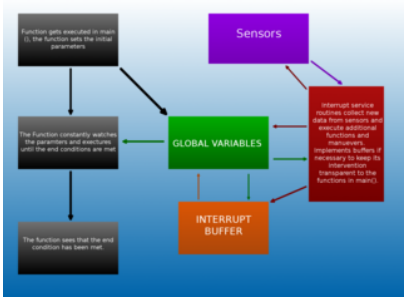

Figure 24: Flow Diagram of Code

Here is a diagram that emphasizes how the functions are only dependent on the global variables which are initially set. The interrupt service routines then manipulate these variables as a response to sensor data, making all the fine details transparent to the programmer. The programmer simply has to know what the global variables mean, and the rest is handled for him/her.

4 Budget

Hardware	Cost
Given Hardware:	$535.00
Purchased Hardware:	$333.92
Total Spent:	$868.92
Project Limit:	$750.00
Amount Over Budget:	$118.92

Figure 25: R&D Budget

Subsystems	Cost
Temperature Module:	$31.92
Ultrasonic Module:	$19.85
Power Module:	$46.87
Navigation Module:	$117.80
Microcontroller:	$170.00
Drive Module:	$329.98
Total Spent:	$716.42
Robot Limit:	$750.00

Figure 26: Final Product Cost

As is evident from the above tables (figures 25 & 26), the R&D costs of the robot exceed the allotted budget by a significant margin. This is somewhat expected due to the fact that most components did not simply work out of the box. In some cases more components were purchased than required, in order to brace for possible flaws & component damage in the analog circuit design process. Indeed, such analog circuit design issues did arise multiple times, particularly during design/testing of the power board. In addition to this, the initial inexperience with the chassis hardware led to some device damage early in the semester and replacement H-bridges were obtained. This replacement cost also makes up a substantial portion of the budget overflow in the R&D section of the table.

On the other hand, all the components currently implemented on the robot add up to a total of $742.49, which is underneath the allocated budget

limit by $7.51. Although this is currently under the spending limit, it is anticipated that a robust color-sensing system for accurate waypoint detection will send the overall robot cost over budget by a very small margin.

5 Conclusion

Through the design of an autonomous navigation robot, lessons were experienced and learned. Lessons in money and time management and group cooperation and communication were learned and skills acquired from previous courses were used to effectively design a robot. Our robot was able to reach all five waypoints relying on dead-reckoning alone. It had robust collision avoidance which granted it the ability to work its way out of difficult situations. Our robots body was designed for outdoor terrain and could handle rocks and wood chips with ease.

Although we are behind schedule, with more work our robot will have the capability of navigating outdoors with the use of both a GPS and compass making it closer to being considered a truly autonomous robot. In the future, we will implement a fully functional powerboard that can power the microcontroller and other subsystems without causing the inductor in the switching regulator to burn out. We will implement a searching algorithm to scan the waypoint for objects and research more on line-sensing to determine when the waypoint has been entered. We will integrate the remaining sensors into the final design and have a fully functioning autonomous navigating robot capable of reaching each waypoint and detecting the objects within.

References

[1] Pololu Robotics & Electronics, Dagu Wild Thumper 4WD All-Terrain Chassis, www.Pololu.com, 2011. [Online].
Available: http://epbuddy.com/index.php?main_page=product_info&products_id=49,
[Accessed: May. 8, 2011]

[2] Pololu Robotics & Electronics, Pololu High-Power Motor Driver 18v15, www.Pololu.com, 2011. [Online].
Available: http://www.pololu.com/catalog/product/755.
[Accessed: May. 8, 2011]

[3] EP BUDDY, 2S1P Battery Pack, epbuddy.com, 2011. [Online].
Available: http://epbuddy.com/index.php?main_page=proc
[Accessed: May. 8, 2011]

[4] KYTRAQ, 65 channel Low Power GPS Receiver — Flash, VENUS634FLPx datasheet, Nov. 2008 [Revised Jan. 2009]

[5] Melexis, Single and Dual Zone Infra Red Thermometer in TO-39, MLX90614 family datasheet, Sept. 2010 [Revision 006]

[6] Fairchild Semiconductor, Optologic Optical Interrupter Switch, H21LTB datasheet, Nov. 2004

[7] SHARP, Distance Measuring Sensor Unit, Digital output (100 mm) type, GP2Y0D810Z0F datasheet, Dec. 2006

[8] TAOS,High-Sensitivity Light-to-Voltage Converter, TSL257 datasheet, Sept. 2007.

[9] Panasonic, Silicon PIN Photodiodes, PNZ300, PNZ300F datasheet, Mar. 2001.

[10] Lumex, Ultra White LED, Water Clear LENS, SSF-LXH103UWC datasheet, Dec. 2001.

[11] National Semiconductor, Switching Regulator, LM3578A datasheet, Feb. 2005.