Robocup 2003

A systems engineering project

A Design Project Report presented to the Engineering Division of the Graduate School of Cornell University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering (Electrical and Computer Engineering)

By

Allen Hou
Elliot Cheung
Gregory Peng
Wajih Effendi
Shantini Supramaniam
Aaron Nathan
Emmy Lai
Carlo Soracco
Tolu Odumosu
Shing Yan

Acknowledgements

We would like to acknowledge a number of people who have made surviving this year possible. First of all we want to thank Prof. Raffaello D'Andrea for his guidance, and encouragement. We also want to thank him for trusting us enough let us make our own decisions. Not all leaders are capable of this quality of leadership. We would like to thank Dr. Jin-Woo Lee for his unflinching involvement. It cannot be easy to be continuously assaulted by requests from us all. We are extremely grateful to David Li our first semester team leader who introduced us all to the project and made us feel welcome to the Robocup family. Finally to Carlo who in addition to being a one man army, lead the team in the spring and helped us get to the finish line.

Abstract

Master of Engineering Program
Cornell University
Design Project Report

Project Title: Robocup Systems Engineering Project 2003

Author(s): Allen Hou, Elliot Cheung, Gregory Peng, Wajih Effendi, Shantini Supramaniam, Aaron Nathan, Emmy Lai, Carlo Soracco, Tolu Odumosu, Shing Yan,

Abstract:
The Robocup goal is to one day field a team of fully autonomous robots that can play soccer against a human team. This goal encompasses the fields of artificial intelligence, feedback control, robotics, and computer vision. In addition to these, there are numerous electrical and mechanical subsystems that are fundamental to the concept of playing soccer.

The Electrical design team of 2003 faced the difficult challenge of improving on a winning system. We set out to specifically improve motion control, drive sub-systems, kicking sub-system and wireless communication. In addition, we also revamped most of the previous circuits.

The team was charged with the responsibilities of designing, testing, and implementi ng the 2003 system, as well as integrating this new system with the new mechanical framework. We were also required to work within the boundaries of the competition rules, as they evolved. This year efficiency and robustness were given top-priority. We also carried out extensive research on alternative platforms for our bots (the PC-104 computer on chip).

The result of all of this effort is a highly integrated, efficient electrical system, which should outperform not only the 2002 versions, but any other team in the world.

Report Approved by

Project Advisor: _____ **Date:** _____

EXECUTIVE SUMMARY

The overarching goal of the International Robocup league is to develop a team of fully autonomous robots which would be capable of playing against a human team by 2005. In the process of achieving this goal, yearly competitions are held, and each year the stakes are raised higher and higher as we aspire to this loft challenge.

The Cornell Big Red team has performed spectacularly at the international scene. With 1st place finishes in 1999, 2000, and 2002, we are currently ranked number one in the world. This strong performance can be linked directly to the position of the team leadership in tackling the all problems from a system perspective.

Due to this system viewpoint, we have successfully pioneered unique innovations that are now becoming standard in the Robocup world. The dribbler, omni-directional drive and controlled passing are all technologies that were developed and implemented by the Big Red team.

This year, the current team set out to improve upon the previous design. There were a number of core areas that we directed our focus towards. These are

1. Motion Control
2. Wireless
3. Drive System
4. Kicking sub-system

These areas were identified as the main points of weakness, and our goal was to bring about significant improvement in the aforementioned areas.

We integrated a rate-gyro and accelerometer in an effort to improve motion control. Bluetooth technology has been integrated into the wireless sub-system. The kicking system has been completely revamped, and we now support an in-line, high voltage kicker. Most of the in-efficiencies have been eliminated from the drive system with the new H-bridge implementation.

Attention was paid to failure modes noticed from previous years. Our design for X methodology has resulted in a much simpler robot. The result of all of this effort is a highly integrated, efficient electrical system, that accomplishes the requirement specification and should outperform not only the 2002 versions, but any other team in the world.

Table of Contents

INDIVIDUAL CONTRIBUTIONS

Allen Hou

Allen was the EE in charge of implementing the angular rate gyro and accelerometer, the 2 new additions to the onboard sensing subsystem. This involved locating and evaluating several different rate gyros, designing appropriate analog filters to remove unwanted noise, and creating a test platform on the 2002 robots to field test the final gyro. In addition, he performed trade-off analysis on the configurable accelerometer parameters. He coordinated his efforts with Oliver Purwin, the Ph.D. candidate responsible for the control algorithms.

Elliott Cheung

Elliott took the lead as the analog circuit engineer, designing circuit boards for the PC-104 prototype, compiling schematics, and managing the batteries for the 2001-2003 teams. He evaluated several voltage regulators, worked on board layout for the 2003 robots, and lent his expertise to team members whenever needed. He calculated the power consumption of the different subsystems to ensure that there would be sufficient power available at all times, even in the worst-case scenario.

Gregory Peng

Greg handled the design of a completely new microcontroller subsystem. With the microcontroller as the central control component of the electrical system, Greg communicated with the other electrical team subgroups to define an interface between the microcontrollers and other electrical components. Greg worked with each subgroup to make sure all electrical components worked properly with the microcontrollers on a breadboarded prototype. He actively took part in the manufacturing, testing, and revising of all prototypes and the final production robot.

Wajih Effendi

Wajih rejoined the EE team in the spring after spending a semester doing a co-op with Slumberger. He was primarily responsible for implementing the FPGA code and developing the h-bridges used to control the motors. His FPGA work involved selecting a new FPGA, porting over the code from 2002, and adding new features while shrinking the code size. He created a high-performance h-bridge design using individual MOSFET's in order to increase the power provided to the motors. He assisted both the IR and wireless teams in their efforts and provided much needed assistance during board layout.

Shantini Supramaniam

Shantini was responsible for improving the IR transmitter/receiver circuits as well as transitioning the horizontal dribbler to an h-bridge drive system. She reduced the part count for the IR circuitry and significantly improved its performance by raising its operating frequency. She investigated and later abandoned using a current detection device to measure the rotational speed of the horizontal dribbler. She then ported and modified the 2002 drive motor code so as to control the dribbling circuit using feedback from an encoder.

Aaron Nathan

Aaron worked on both the optical sensor and the wireless system used in the 2003 robots. He designed a circuit that could communicate with the optical mouse sensor chip found in typical computer mice so the robots could more accurately determine their position. He conducted velocity tests and determined the chip could not support the increased speed of the 2003 robots. He also reworked the 2002 wireless system and added support for 4 new modules, all running at unique frequencies. The new system allowed for higher bandwidth and lower latency, which significantly increased the control on the new robots.

Emmy Lai

Emmy worked on the wireless system used in the 2002 robots (RPC) and in the 2003 robots (tx/rx 2/3 modules). She worked along with Aaron Nathan in running various tests to get the tx/rx 2/3 modules working. Also, she had been placed responsible for preparing the robots for demos, test runs, and competitions. Emmy keeps an updated list of all the working RPC modules. She also reprograms the non-working RPCs reported by other members of the team.

Carlo Soracco

Carlo was responsible for Bluetooth development in addition to leading the team in the spring semester. He analyzed several different wireless technologies before choosing Bluetooth, spent considerable time learning the Bluetooth standard, laid out the 2003 wireless boards, and selected components for the boards. He also laid out the final revisions of the 2003 boards and lent his assistance to the h-bridge design effort. He served as the primary link between the CS and Mech. E. teams and ensured successful operation of the system as a whole.

Shing Yan

Shing helped in the research for a new microcontroller during the first semester. Later, he was responsible for the research of the PC104, a single-board computer that is capable of processing large amounts of data. He researched different PC104 boards from different vendors. He also looked into the various required peripherals for the PC104. He assisted Dr. Jin-Woo Lee in preparing the prototype boards for the tests and evaluation of the PC104

Tolu Odumosu

Tolu was responsible for the kicker development. He did all the initial research into the original kicker circuit, and derived the analysis that led to the current design. With Carlo's help, he implemented the conclusions of the analysis into the current kicking sub-system. He was also responsible for the extensive research on the return path. He oversaw the entire documentation process and served as a systems engineer for the team.

SYSTEM OVERVIEW SECTION 1

The Robocup electrical system comprises of all the electronics on the robots and the wireless communication system. The electrical team is responsible for designing, building, testing and implementing all electrical circuits.

The diagram beneath (taken from last year's document), which is a simplified representation of the entire system, shows the sequential relationships between the various elements of the Robocup system. It also shows the parts of the system which are the responsibility of the electrical team.

Our goal this year was to build a reliable, robust robot, which would improve upon the previous design. To accomplish this, we approached the problem with a vastly different perspective. As we tackled each layer of the electrical design, we always asked the question *"why?."* This enabled us to revisit the original assumptions behind each circuit. In some cases, we were able to improve upon the design tremendously. For example the kicking sub-system was completely revamped. In addition, asking "why" enabled us to move from four micro-controllers, to a two micro-controller design.

The result is that the electrical design this year is slimmer, robust and more efficient. There are still areas where that it may be possible to improve even more, but by and large, we are very proud of the brevity and simplicity of our design. All schematics and diagrams are included in the accompanying CD.

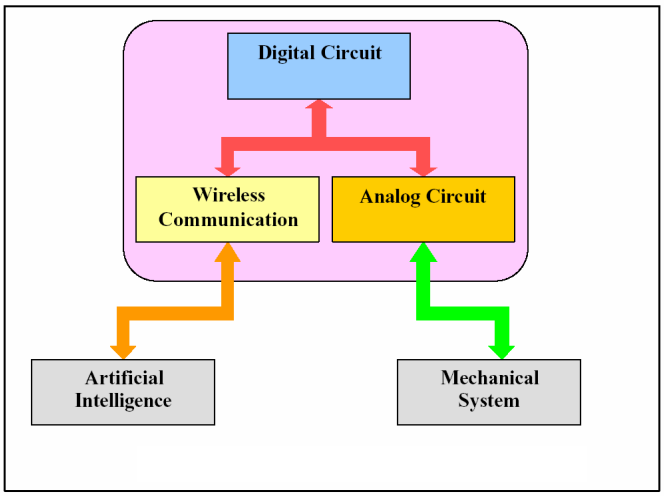


Figure 1.1 Electrical aspects of Robocup System

System Engineering Management Process Section 2

2.1 Overview

We approached Robocup 2003 with a systems engineering approach. Systems engineering is an interdisciplinary approach to create successful systems by providing tradeoffs and integration between system elements. The Robocup team consists of three system groups, the electrical team, mechanical team, and software engineering team. The goal of systems engineering is to integrate these groups in the design process and to ensure that interfaces and dependencies between subsystems are well-defined. This requires interaction between groups as well as effective management of each group's subgroups.

The first step in the systems engineering management process involves defining goals, objectives and requirements of the project. After this, a conceptual analysis is performed by establishing the overall architecture of the system, including functional groups, project scheduling, risk analysis, and preliminary cost estimates. The design phase can now begin as each functional group or subsystem begins preliminary designs and testing of ideas. In this phase, tradeoff analysis between different ideas can be made and certain ideas can be pursued or abandoned. As testing of a design continues, prototypes are developed and tested. Solutions are then solidified.

The next phase is manufacturing and system integration. Prototypes are manufactured and interfaces between subgroup's systems are field-tested. Each component as well as the system is tested thoroughly. Once this is completed, the product is deployed and used in a real situation. While all the design and testing is being carried out, it is important for the team leaders to make sure that designs and tests are being completed on schedule and within budget. Systems engineering is an iterative process where later phases in the project life cycle are repeatedly compared with the planning phases earlier in the project. For example, requirements and definitions made at the beginning of the project must be verified and met. At the same time, plans must be made in case the design process goes off schedule and lead times for parts must be taken into account in the project schedule. These are all part of the systems engineering approach to projects.

Robocup is a project that needs the principles of systems engineering in order to effectively cover all the bases of the project and to utilize human resources effectively. Maintaining technical excellence and innovation while at the same time, keeping within the boundaries of the project is the driving force of Robocup. Because we were a very new EE team this year, it was imperative that everyone on the team learned the old system and became familiar with it.

By using a systems approach, the design of the robot in 2002 was learned by each team member through a methodical understanding of the concepts and requirements of the project. We also understood the

techniques that were needed to fulfill the project specifications. Once the definitions of the requirements became familiar to everyone, each team member was given a specific technical responsibility. These "functional groups" helped to solidify team members' roles on Robocup and gave each member ownership of the project.

Other aspects of the project such as staying on schedule, staying within budget constraints, progress of each functional group, keeping track of improvements of the 2002 robots, current designs of the 2003 robots, and interactions between team members were addressed using different tools of systems engineering. Since many of the members of the EE team this year are in the M.Eng systems engineering option, we've decided to utilize some of these tools in creating a more efficient and comprehensive approach to Robocup. Some tools were more effective than others and thus were used more consistently. Outlines of such tools are further explained below.

2.2 Meetings and Minutes

Over the course of the year, EE group meetings were held at least once a week. The purpose of these meetings is to allow team members to raise issues that are related to the progress of the project. It is also a forum to allow team members time to report the progress they are having in their responsibilities. This allows other team members to offer feedback or input to what is being reported. The format of each meeting involved an agenda being laid out by the system leader, first addressing any administrative issues such as upcoming deadlines and action items for the team. After this, each team member went into what they had accomplished in the past week.

In the second semester, we began meeting twice a week. This was to facilitate awareness of the progress of each functional group more frequently. Having an additional meeting per week, created an air of urgency and accountability that was needed in order to accomplish the many tasks that were approaching in the spring semester.

A different member recorded meeting minutes each week. This helped keep track of what was talked about at each meeting so that if a member was unable to attend the meeting, he/she would know what he/she missed. Minutes helped outline the progress of each team member over the course of the year. Action items for the group were highlighted in the minutes as well. Recording the minutes also creates a history of the work progress and thought process of our team, which might prove useful to future groups. The minutes were also posted on the Robocup intranet so that they could be accessed by anyone who was interested in the progress of the EE team.

2.3 Lab Notebooks

Each team member was encouraged to keep a laboratory notebook of what they accomplished each time they worked on something in the lab. Keeping a good record of progress is important because it can help in writing documentation for future teams and was useful in helping the team members organize their ideas. Test results and observations were recorded in the lab notebooks. It was helpful for this year's team to have copies of previous team members' lab notebooks to assist them in understanding the 2002 system. In addition, this ensured that current members did not repeat the mistakes of the 2002 team.

The lab notebooks were also useful for writing the final documentation. This year, it seemed that any question about the previous Robocup system was directed to last year's documentation. Because it was well written and complete, our team was able to understand the reasoning behind the design without any guesswork. Also, the well-written documentation made it easy for the current team to develop new requirements and definitions.

2.4 E-mail/AIM

Communication is essential for any group to make progress. The entire team used e-mail constantly to inform people of announcements and to stay up to date about events going on in the team during the week. Since many people are busy outside of the lab, the best way to reach someone was through e-mail. Though minutes were posted on the intranet, they were also sent out to the team via e-mail. This seemed more effective since people could just open the attached file and read the minutes, instead of logging onto the intranet and having to remember another password. E-mail list serves were also more useful than posting anything on the intranet.

Another effective way to reach team members when needed was to use AOL Instant Messenger. Most people subscribe to the free service and use it whenever they're at a computer. Therefore, it was useful to have each others' screen names so that we could ask each other questions or see if people were in the lab or not.

2.5 Centralized Computer

There were many electronic documents and files that people needed to access over the course of the project. These include:

- Datasheets
- Code for Microcontrollers

- Parts Lists

These files were all kept on MIA, the EE computer. The computers in the Robocup lab were also networked to each other so that team members could access information from all the computers when needed. In this way, everyone had access to all the files and could find things that other members were working on.

2.6 Code

For example, the microcontroller code is used by all parts of the system. Therefore, each individual responsible for parts dealing with the micro code would be able to see previous versions of code and update what was necessary. Unfortunately, many versions of the code began to appear on the computer since people were saving different versions each time they edited code. The use of CVS (Code Versioning System) was attempted but was unsuccessful.

2.7 Datasheets

Datasheets of different parts being used in each member's circuit were kept on the computer so that they were available for quick reference when different team members collaborated to debug a problem.

2.8 Parts Lists

Parts lists were also kept in a central area so that if extra parts needed to be ordered, they could be found easily in the file. These parts were organized in a spreadsheet and contained the following information for each part:

- Part Description
- Description
- Value
- Manufacturer
- Manufacturer Part Number
- Digikey (Distributor) Part Number
- Use (specific circuit in which part is used)
- Quantity per Board
- Comments
- Total in Stock

2.9 Battery Logging

A method of logging how many times batteries were recharged was adopted. This helped us keep track of battery performance over its lifetime. If a set of batteries was recharged too many times, its capacity decreased and this log would inform us of when to replace a set of batteries with new ones.

2.10 Extracurricular Group Activity

Team building activities were an important part of developing a cohesive EE unit on Robocup. Group members went on outings a few times a semester to get to know each another better outside of the lab experience. This helped to foster unity amongst group members and allowed people to become more comfortable with each other so that working together would become more pleasant. Team unity is important to the project because it keeps people motivated to work hard and to help one another with any difficulties they experience throughout the year.

2.11 Some thoughts

Many of the tools of systems engineering mentioned above seem to be tedious and trite. However, in a complex project like Robocup, it is easy to lose sight of goals, organization of ideas and progress. By keeping records, errors that might occur at later dates are more easily traced and solved. A methodical approach to designing a system such as Robocup is the only way all bases can be covered and future mistakes can be averted and minimized. Developing a cohesive team is also important because the systems engineering process is ultimately about working with people. Only in a group that works well together can one keep track of progress and stay true to the original requirements definitions, costs, schedules and concepts.

2.12 System Engineering Product Cycle

The system engineering product cycle that the team follows is illustrated in the diagram below. At the beginning of the phase, the team members read the documentation of the previous year and study the system. We learn about the previous design, the approaches that were undertaken in the design process, the reasons for the approaches that were taken and the decisions that were made. Then we think of improvements to the previous years' system. We will then meet with the project supervisors or customers to discuss new requirements and improvements. The requirements are the basis of our design.

We then carried out our preliminary design, which mostly involves implementing circuits on breadboards. After the preliminary design, we have preliminary testing. If the results are not satisfactory, we will revise

the design and repeat the testing. After verifying that the design has met all the requirements, we will send the design out for manufacture. We will then carry out the final testing and integration of the robots. After that, the robots are in the support and maintenance stage. After the competition, the support and maintenance is passed on to the future team.

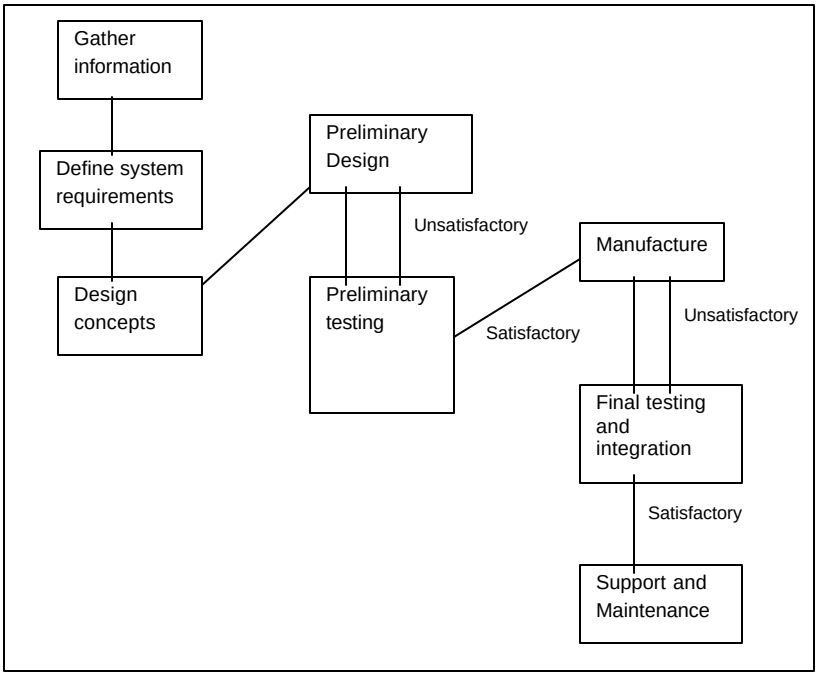


Figure 2.1 Systems Process

2.13 Customer Needs and Requirements Gathering

We, the Robocup team, are the customers ourselves as the goal of the project is to win the RoboCup competition. The customer needs are gathered from previous year's documentation, the project advisors, ideas from the current team members, and the experience from prior matches in the competition. The information gathered was compiled into the new requirements. We focused on the requirements and allocated our time and budget accordingly.

2.14 Design Issues

This year, we applied systems engineering concepts to the project. Our Electrical Engineering representative attended Software Engineering and Mechanical Engineering meetings regularly to facilitate information flow. Within the Electrical Engineering team, we made a conscious effort to ensure that all subgroups shared the same vision of the fully-integrated system and that they were making design decisions, keeping the integration of the component with the whole system in mind. Some special EE and interdisciplinary task forces were set up to resolve special systems design issues.

The concepts can be broken down into several categories:

- **Design for Testability**
- **Design for Usability**
- **Design for Maintainability**
- **Design for Robustness**
- **Design for Manufacture**

2.14.1 Design for Testability

- There are 5V, power and ground test points on analog board.
- Indicator LED's for the kicker IR sensor, wireless reception and transmission, battery meter LED's to show battery voltage.
- Test modes for the dribblers (low speed and high speed), robot drive motors, and the kicker.
- Serial port headers available for debugging the microcontrollers in real-time.
- Jumpers to disable each microcontroller to isolate problems and for running specialized tests.

2.14.2 Design for Usability

- Added a 7-segment display showing which robot number/mode a robot is configured as.
- We have a much more accurate battery meter this year, so that we don't need to use a multi-meter to measure the battery voltage.
- Several step-by-step procedure documents were written to teach new users how to set up programming/compiler software and program the microcontrollers.
- Micro controllers can be programmed using any computer with a serial port and a terminal program. A menu-driven boot loader is used for programming the microcontrollers.

2.14.3 Design for Maintainability

- We used two individual battery connectors this year for the two battery packs, so we don't have to worry about plugging the two batteries together before connecting them to the analog board.
- Microcontroller code is very modular, and it is easy to search through the code to find specific actions. The code is formatted very strictly to be very uniform. All code is heavily commented to aid the future team in learning the system.

2.14.4 Design for Robustness

- We have battery traces upwards of 120 mils (3mm), and the traces for the motors are a minimum of 60 mils for high current. The usual signal traces are only 8 mils.
- We added protection diodes in various places to prevent damage to the boards.
- Sturdy headers and receptacles were chosen for programming the microcontrollers.
- Dongles were made to be very durable. This was a problem in 2002 when the serial programming dongle often broke when it was not handled properly.
- A microcontroller crystal was selected to withstand vibrations.
- A 5V reference chip was selected for high precision and tolerance so that noise from the power supply was minimized.
- Any unreliable part in the 2002 robots was avoided.

2.14.5 Design for Manufacture

- We used larger-size surface mounts parts (>0805), so that it was easier to solder them by hand. In addition, the population house dislikes having parts that are too small.

MICROCONTROLLER SECTION 3

3.1 Overview

If one was to compare a robot to a human, the electrical system would be the robot equivalent of a nervous system. In humans, actions are commanded through the nervous system, and sensed information is returned through the same system. This is no different in robots, save for the wires and conductive metals replacing biological nerves. Sending commands and receiving sensed information is the responsibility of a critical human organ, the brain. The RoboCup 2003 robot equivalent of a brain is the microcontroller system.

Acting as a metaphorical brain, the microcontroller system must process received information and generate the appropriate response. Granted the off-board artificial intelligence (AI) computer does most of the required brainwork to make the robots play a recognizable game of soccer, but the onboard brain translates the AI's decisions into robotic actions and does the required thought-processing needed to maintain these actions. Encoded commands are received from the AI computer via a wireless module. From decoding these commands, the microcontroller system determines whether to kick, dribbler, or move. Onboard sensor feedback indicates if the robot should carry out a kick command, and feedback aids in control of the robot's movements and dribbling. Adequate microcontrollers are necessary for quick and reliable processing of these inputs and outputs.

3.2 Introduction

Microcontrollers are microprocessors with a variety of features and functionality built into one chip, allowing for their use as a single solution for control applications. The operation of a microcontroller revolves around the core central processing unit (CPU), which runs programs from internal memory to carry out a task. Such a task may be as simple as performing mathematical calculations, as is done by an ordinary CPU of a personal computer. On the other hand, the task may be more complex, involving one or many of the microcontroller's hardware features including: communications ports, input/output (I/O) ports, analog-to-digital converters (A/D), timers/counters, and specialized pulse width modulation (PWM) outputs. With access to hardware ports, the CPU can interface with external devices to control actuators, gather input from sensors, or even communicate with other devices that have their own internal processors. The control of these ports, handled by the program running on the CPU, allows for a great deal of flexibility. Inputs and outputs can be timed to occur in specific sequences, or even based on the occurrence of another input or output. A major drawback to microcontrollers is that only so much processing power can be provided due to the need to fit the CPU and all of the hardware features onto the same chip, which is usually constrained to a reasonable size for integration in compact systems.

3.3 Microcontroller Selection

In the RoboCup 2002 robots, the microcontroller system is a modular design that is fully capable of performing its required tasks. The design is mainly constrained by two related factors

1. Required features needed to control the robots' electrical subsystems; and
2. Limitations of the microcontrollers chosen, in terms of processing power and features available.

With the planned improvements and designs for the 2003 electrical system, the former microcontroller system was no longer sufficient, thus the need to select and design a new system. This process of designing the onboard brain for the 2003 robots involved many steps. It was important to understand the workings of the previous system in order to make educated decisions about improving upon the old design. In addition, it was important to be in contact with the other team members who were concurrently designing and developing electrical components that directly interact with the microcontrollers. This was necessary so as to define the requirements of the interface between those components and the microcontrollers. This information was the basis upon which we selected the candidate for the 2003 microcontroller selection and also the succeeding evaluation process.

The 2002 microcontrollers are the Microchip PIC16F8 77's. Four are used in a modular design for a variety of functions. One microcontroller is used for each of the following:

1) Handling data packets received from the wireless modules
2) Parsing this data, handling control and sensor data
3-4) Motion control and compensation

Consideration for other microcontrollers was needed to allow for greater flexibility to accommodate and allow for the new advancements of the 2003 design. To accomplish this, a faster and more feature-filled microcontroller was needed for onboard data processing and advanced I/O support. A faster microcontroller should be capable of handling all control, feedback, and sensor data from the new additions in the 2003 design. The decision to use a more feature-filled microcontroller was also aimed at supporting the additions with more I/O pins, A/D inputs, timers, and hardware serial ports.

Having more capable microcontrollers allowed for the simplification of the digital design by combining the duties and functions of microcontrollers. The goal was to shoot for integrating the wireless and main microcontrollers into one microcontroller and integrating the two motion microcontrollers into one central motion controller. This still retains the modular design by keeping main/wireless functions separate from

the complex motion control calculations and feedback loop. This simplification of parts offered such possible benefits as digital board size reduction, lower power consumption, and greater maintainability.

Several microcontrollers were taken into consideration from Microchip, Intel, Atmel, and Motorola. The final contenders were chosen due to a combination of features, ease of development, and development support, and they were then compared to the 2002 microcontroller. Feature requirements were defined by examining those of the 2002 microcontroller system and the interfaces needed to implement the planned 2003 enhancements. Basic features that were initially desired consisted of the following:

1) 40+ MHz (if CPU is 8-bits) – roughly calculated since the goal was to combine two microcontrollers of the 2002 system (each 20 MHz) into one microcontroller for 2003.
2) 4+ PWM outputs – each wheel motor requires a PWM, so the central motion microcontroller would need at least four. The main microcontroller needs PWM outputs for the horizontal dribbler, side dribblers, and extras for other possible additions in the 2003 design.
3) 5+ external interrupts – the main microcontroller for 2002 needs external interrupts for the kicker's IR sensor and detection of wireless data sent from the wireless microcontroller (which would be the wireless module in the 2003 design). In addition to these, the implementation of a ball possession detecting sensor, rate gyro, and optical motion sensor might have each required an interrupt.
4) 40+ I/O pins – the 2002 motion microcontrollers each use 18 pins, the main microcontroller uses 25 pins, and the wireless microcontroller uses a number of pins for communicating with the wireless transceiver module and the main microcontroller. Combining two microcontrollers into one would save a few pins needed for communication and programming, but it would be a good idea to have healthy amount of pins for expandability with new robot components.
5) 5+ timers – current motion microcontrollers each require three timers: one for reading the encoder/FPGAs of each wheel, and one for setting the rate of the control loop. Combining the two motion microcontrollers together would yield five timers, since only one would be needed for the control loop. However, five was a bare minimum, and more might have been needed for expandability/flexibility.
6) 2+ hardware serial ports – the 2002 main microcontroller only has one hardware serial port, which is used to receive data from the wireless microcontroller. To communicate with each of the two motion microcontroller, the main microcontroller must emulate serial ports on I/O pins through software, taking up CPU time. The plan for two microcontrollers would require the main microcontroller to have two hardware serial ports to take the load off the busy CPU, one for communicating directly with the wireless module and one for communicating with the one consolidated motion microcontroller.

IMPORTANT FEATURES	MICROCHIP PIC16F877	MICROCHIP PIC18F8520	ATMEL AT91M55800A	MOTOROLA HCS12 FAMILY
Maximum Frequency (MHz)	20	40	33	25
Internal Datapath	8-bit	8-bit	32-bit	16-bit
FLASH Memory	8k	16k	none	32-256k
Timers	3 (1 16-bit, 2 8-bit)	5 (3 16-bit, 2 8-bit)	6 (16-bit)	8 (16-bit)
PWM Outputs	2	5	12	8
Hardware Serial Ports	1 USART, 1 SPI/SCI	2 USART, 1 SPI/SCI	3 USART, 1 SPI	2 SPI / 2 SCI
A/D Channels	8 (10-bit)	12 (10-bit)	8 (10-bit)	16 (10-bit)
External Interrupts	4	4	7	12
I/O Pins	34	68	147	91
Package	44-pin	80-pin	176-pin	112-pin

Table 3.1 Feature Comparison of 2003 Microcontroller Candidates

All of the microcontrollers listed had considerable improvements over the Microchip PIC16F877's feature set and looked, on paper, to be suitable upgrades.

The Atmel AT91M55800A was an impressive looking microcontroller on paper with a 32-bit architecture at 33 MHz and a plethora of I/O pins, but it did not have on-board FLASH memory, which allows for quick and easy programming of code onto the microcontrollers. An external FLASH chip would have to be used, adding the requirement of another component and the complexity of implementing and interfacing the chip to the microcontroller. The Atmel was considered for the 2002 robots, but problems with the compiler, simulator, and an inadequate debugger caused the reconsideration of a simpler chip which met reduced requirements, hence the PIC16F877. 147 I/O pins offers a ridiculous amount of expandability, overkill for the robots.

An attempt to evaluate the Atmel, to see if the programming issues were resolved in compiler software updates, had failed. It was quickly realized how complex developing for the Atmel was going to be, since the microcontroller is based on the ARM Thumb architecture and required linking several different drivers and libraries for programming. The slightest mismatch between drivers/libraries and development software would cause an error during compilation. The developing environment for the Atmel was too finicky and convoluted due to the inclusion of unneeded features. Since other microcontrollers with comparable features existed with less complicated programming requirements, it was decided not to pursue the use of the Atmel.

The Microchip PIC18F8520 looked, to some people, like the obvious upgrade path for to the PIC16F877. The PIC microcontrollers were proven to work in the past, and the compiler software needed to program the PIC18 series was already on hand. There was already familiarity with the compiler due to studying the 2002 microcontroller code and having experience making modifications and additions to the 2002 robots for testing purposes. In addition, the programming would have been almost the same for the PIC18 as the PIC16, so past members could provide assistance.

However, the PIC18 was not so much of an overall improvement over the PIC16 microcontrollers. The PIC18 was still using 8-bit datapaths, but should have been faster than the PIC16 due to a higher clock speed. Having a 16 or 32-bit datapath like the Atmel and Motorola microcontrollers would allow for the transfer of internal data greater than 8-bits in less clock cycles, translating to greater efficiency and calculation speed. Having only five timers and four external interrupts on the main microcontroller might have caused restrictions on the number of extra features that could be implemented.

The Motorola HCS12 microcontroller family is the new high-speed version of the HC12, which were well-established in industry. All of the features met the outlined requirements with comfortable room for expanding:

1) The HCS12's might not have been 32-bit like the Atmel, but it ran at a 25 MHz bus internally at 16-bits, which was still a great improvement over the PIC16's.
2) There was plenty of onboard FLASH for program code, between 32k and 256k (though only 128-256k models are available now) – the 2002 code only used about 2k of the PIC16F877's 8k FLASH.
3) There were a good number of I/O pins, PWM's, external interrupts, timers, and A/D channels. All features were sufficient to support the planned 2003 design
4) There were models with at least two hardware serial ports for communication with the wireless module and the other microcontroller without requiring software emulation.
5) Since the HCS12 was backward compatible with HC12 code, which was well-established, a great number of resources existed for development and support. Simple, non-convoluted software development packages existed for purchase at low prices, and a free GNU compiler package existed.

The Motorola HCS12's seemed to fit the needs of the 2003 design well, on paper. An evaluation was required to see if hands-on developing was straightforward enough for the implementation of these microcontrollers into the 2003 design and to determine if the microcontroller indeed had the processing power to accomplish the design goals.

3.4 Testing and Evaluation

The process of evaluating a new microcontroller involved several crucial steps: obtaining cost-effective development and evaluation tools, becoming familiar with the developing environment, and exploring all useful features to understand performance and behavior. Only upon completing a thorough evaluation could one properly take full advantage of all that the microcontroller has to offer, with respect to integration requirements of the other electrical subsystems.

Evaluation of the Motorola HCS12 microcontrollers began with the purchase of an Adapt9S12DP256 development board and a MicroBDM12SX pod from Technological Arts (www.technologicalarts.com) to facilitate testing of all important features. The development board was reasonably priced ($99 for the board plus $56 for two expansion headers) compared to Motorola's OEM development board (~$500). Expansion header options allowed for easy and direct access to the many ports/pins of the microcontroller. Below is a technical drawing labeling the development board's features:

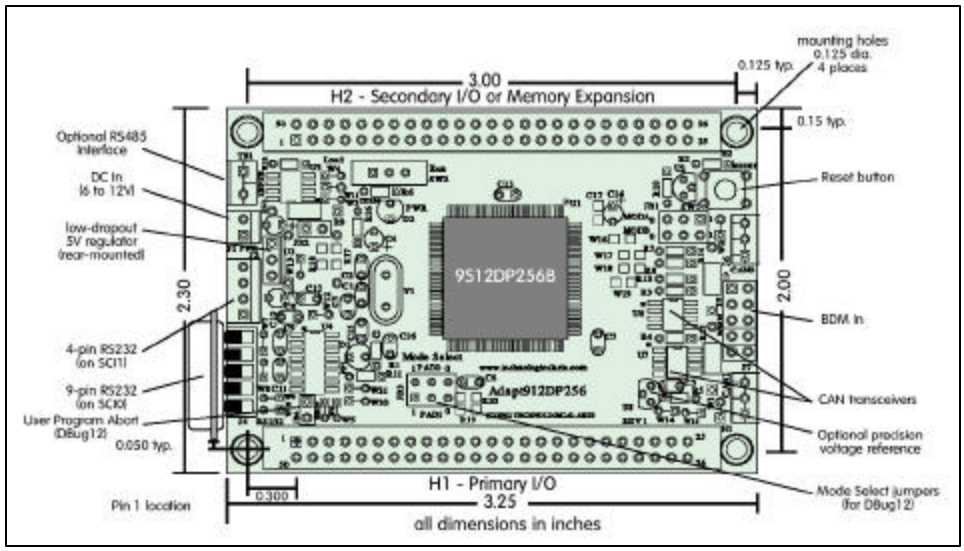


Figure 3.1 Illustration of Adapt9S12DP256 Development Board

The development software chosen for compiling C-based code for programming the microcontrollers was ICC12 from ImageCraft (www.imagecraft.com). This was also a low cost solution ($199) with a simple and straightforward user interface, compared to other solutions that were too bloated and expensive for

our purposes. For the microcontroller evaluation, a free fully-functional trial version of the ICC12 was used.

The first task done after receiving the development kit and pod module was to use the pod module to program the proper bootloader into the microcontroller's FLASH memory. A bootloader allows for the easy loading of programs into the microcontroller via a serial port connection to a computer, rather than requiring the pod module every time. The only serial bootloader readily available for use with the 16 MHz crystal oscillator on the development board did not enable the microcontroller's phase-lock loop (PLL), only allowing the microcontroller's bus to run at only half the oscillator frequency, 8 MHz. Since the bus speed of the Motorola chip is specified to run at a maximum of 25 MHz, the bootloader assembly code had to be modified to enable the PLL to generate the internal clock speed of 25 MHz. For reasons of uneven division of clocks in relation to the serial port BAUD rate on the microcontroller, a bus speed of 24 MHz was chosen in order for the serial BAUD rate to be set to the maximum of 115,200 kbps. (Users in online discussion groups have mentioned that the Motorola MC9S12DP256B can overclock to a 30 MHz bus stably and sometimes stably to a 32 MHz bus. Both of these would also allow for the 115,200 kbps BAUD.) A 24 MHz bus speed was used for the duration of the evaluation in order to eliminate instability as the cause of any problems encountered. The assembly code for the serial bootloader was available from the Motorola homepage of the MC9S12DP256B (http://e-www.motorola.com/webapp/sps/site/prod_summary.jsp?code=MC9S12DP256B Application Note AN2153SW – Software files for app note AN2153).

One of the questions asked of the Motorola HCS12 microcontroller family was how it compared to the 2002 robots' 20 MHz Microchip PIC16 microcontrollers with regard to floating point calculations. As a quick test, a series of intensive floating point calculations were looped through both microcontrollers. Each time the loop completed, an output pin would be changed from low-to-high, then high-to-low, and so on. The calculations were based on most of the floating point equations in the 2002 PIC microcontroller code for the feedback-control loop. The code of the test function with floating point definitions was identical in both compilers of the Motorola HCS12 and Microchip PIC microcontrollers. The code is listed at the end of this section.

The programs were run on both microcontrollers, and the output pins were probed with an oscilloscope to measure the time required to complete each calculation loop. The cursor feature of the oscilloscope was used to take a measurement of the time between changes on the output pin, which is accurate to about +/- 1 ms when eyeballing the cursors onto the output signal. The resulting calculation times:

- Microchip PIC16 (20 MHz): 141 ms
- Motorola HCS12 (24 MHz): 16 ms

The Motorola microcontroller was about 8.8 times faster than the Microchip PIC microcontroller when performing these floating point calculations. Just to make sure there was not some kind of caching scheme occurring on the Motorola chip that would unfairly skew the results when repetitive data is used (this shouldn't be the case since these microcontrollers do not have cache), the number of control_loop() function calls was doubled and the output measured. Doubling the number of calculations done per output change indeed doubled the time required to execute the entire loop. Though the floating point performance of the new microcontroller was quite impressive, it must be remembered that the feedback-control loop is based on inputs, outputs, and basic operations in addition to floating point calculations. Being only 4 MHz faster than the 2002 Microchip PIC, the Motorola HCS12 will offer most of its performance gains in motion control when dealing with complex calculations.

3.5 Major Microcontroller Features

Once the microcontroller had been set up with the optimal bus frequency and bootloader, the testing of features and functionality proceeded. Important features identified by interface and operating requirements of the electrical subsystem consisted of hardware communication ports, I/O ports, external interrupts, A/D ports, PWM ports, and timer/counter ports.

It was decided that the microcontrollers would be communicating with the wireless module via a serial communications interface (SCI) port. The Motorola MC9S12DP256B microcontroller contains two SCI ports, so the additional port would be used for inter-microcontroller communications. An SCI port uses asynchronous transmission and reception to transfer binary data. The asynchronous nature allows for the transmitter and receiver of the port to be used almost independently with only one connection each for the transmitter and receiver. This allows two different devices to share an SCI port - one device can connect to the receiver and another device can connect to the transmitter. Data transfer occurs at fixed speeds, or BAUD rates. For example, some standard BAUD rates supported by personal computer serial ports are 38,400, 57,600, and 115,200 kilobits per second (kbps). However, the Motorola microcontroller is not restricted to these standard speeds; it is limited by the condition of whether or not the bus frequency can be divided, using a special equation and user-selected integer whole number, to be close enough to the desired BAUD rate that data is correctly encoded for transmission and interpreted for reception. In addition, the SCI port supports various interrupt conditions, the most important being an interrupt on receive. When data is detected on the SCI receiver, the microcontroller's normal program operation is temporarily interrupted and a short segment of code can be executed to receive and handle the incoming data. Once the code related to the interrupt is complete, the microcontroller resumes the normal non-interrupt program operation.

Input/output (I/O) ports are used for a variety of basic input and output controls. Numerous I/O ports are available on the Motorola microcontroller. Some ports are dedicated for I/O, while others share connections with other microcontroller features, and it is up to the user to select what function the port will serve. Operating on 5V digital logic, signals that are about 5V represent a binary 1 or high, and signals that are about 0V represent a 0 or low. In basic usage, these binary 1's and 0's can either be output to control other devices, or read as inputs from circuits. More advanced I/O port features supported by the Motorola microcontroller are reduce current drive capability, pull-up/pull-down inputs, and external interrupts on input. The reduced current drive mode is useful for reducing power consumption for some outputs, such as turning on very efficient light emitting diodes (LEDs). Pull-ups/pull-downs can set inputs to a default logic level in the case where a device can only assert one logic level or assert nothing at all. For example, a switch that can only open or close a circuit can be used to assert a high logic level when the switch is closed. However, when the switch is open no logic level is being driven, and in this case a pull-down device on the microcontroller's I/O port can be used to default to a low input signal. External interrupts on I/O ports can signify whether a time-critical external event has occurred and allow for the microcontroller to handle the event appropriately, without delay. Detecting whether the external interrupt occurred when a signal changed from high to low, or low to high provides even more selectivity.

Some devices will have many levels of output rather than just a logic high and low. Such devices can be sensors with high sensitivity to environmental factors. The analog signal passed to the microcontroller from these sensors must be read with the analog-to-digital (A/D) ports. These ports convert the sensed analog voltage into a digital representation, usable by the microcontroller. The Motorola microcontrollers are capable of reading up to 32 different analog signals ranging from 0V to 5V, and translating the voltage into a binary number of 8-bit or 10-bit resolution. A higher resolution allows for more sensitive reading of the signal for greater precision, which is important in certain control applications.

The pulse width modulation (PWM) port is capable of eight PWMs with 8-bit resolution or four PWMs with 16-bit resolution. A PWM signal is exactly what the name suggests, a signal of square wave pulses where the width of the pulse (high time or low time) can be modulated. One can adjust the period of a PWM signal as well as the 8-bit or 16-bit duty cycle (a fraction of a pulse's period where the signal is high) for varying PWM outputs. These pulsed outputs can be used to simulate an adjustable voltage source. A larger the duty cycle gives a longer high signal time per pulse period, thus a higher average voltage. This voltage can be used to drive actuators to varying degrees.

The Motorola microcontroller's timer/counter port offers a slew of useful features. One of the applications is as an output compare port that triggers an event when the 16-bit main timer clock reaches a certain

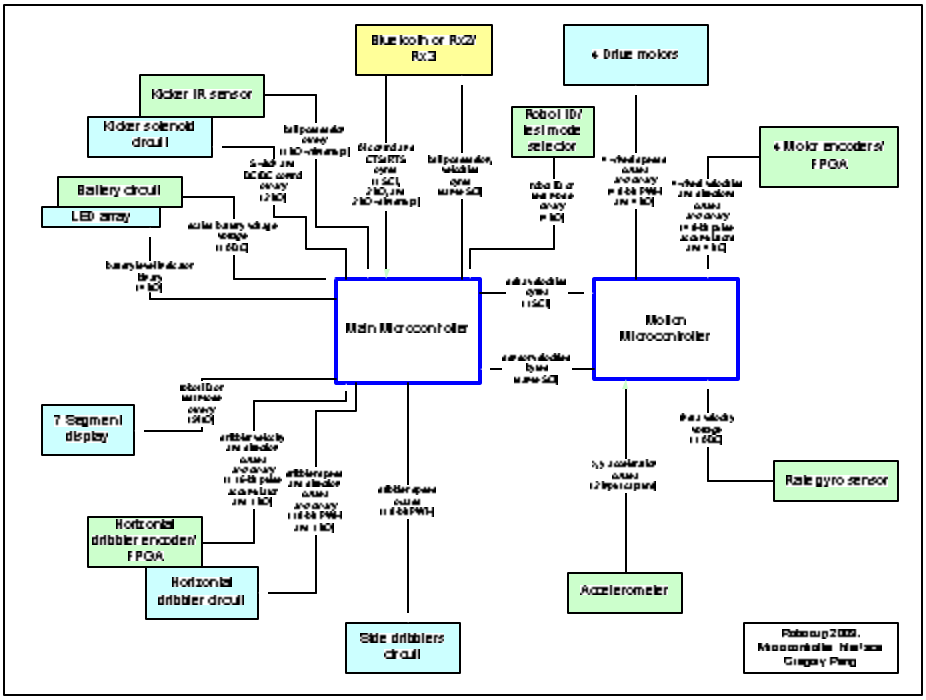


Figure 3.2 Microcontroller Interface Diagram

Yellow boxes indicate modules with both inputs and outputs to a microcontroller. Green boxes indicate modules with input(s) to a microcontroller. Blue boxes indicate a module using output(s) from a microcontroller. The connections between subsystems are listed below corresponding to the boxes in the diagram in counter-clockwise order starting at the top:

1) Bluetooth or Rx2/Rx3 wireless module:
   - The artificial intelligence computer sends kick and dribble commands and velocity data to a robot's wireless module, which then transfers this data to the main microcontroller in the form of bytes over a SCI port.
   - If Bluetooth is used, CTS/RTS inputs and outputs can be used for hardware flow control of data to prevent overflowing buffers or overwhelming a serial port receiver. The return path offered by Bluetooth allows useful ball possession and velocity data to be sent back to the artificial intelligence computer by the main microcontroller via the same serial connection to the bidirectional Bluetooth module.
   - A 2-bit input is needed to read the input from a 2-position DIP switch that indicates to the main microcontroller what wireless module the microcontroller should be configured to support.

2) Kicker infrared (IR) sensor and solenoid circuit:
   - When the infrared (IR) beam of the kicker's sensor circuit is broken, a 1-bit binary signal is sent to the main microcontroller. A high signal indicates that the beam is blocked, that a ball (or accidentally some other object) is within range for kicking. A low signal indicates that no ball is available for kicking.
   - The kicker solenoid circuit relies on a 1-bit signal to control the kicking. A high output from the main microcontroller controls the closing of a switch in the solenoid circuit, which causes the kick, and a low output causes the said switch to open, ending the kick.
   - A separate 1-bit signal is used to enable/disable the DC/DC converter for more efficient power consumption. Setting the connected microcontroller pin as an input (i.e. high impedance mode) enables the DC/DC converter to charge the kicker circuit's capacitor. The microcontroller set as an output and driving a low signal disables the DC/DC converter after the short time needed to charge the capacitor.

3) Battery voltage meter:
   - The battery voltage, fed into a resistor-based voltage divider to traverse a 0V to 5V scale, is read by an A/D input to allow the microcontroller to sense remaining battery life.

- A 4-bit reduced current drive output lights a combination of four LEDs as a visual indicator of remaining battery life. The LEDs are extremely efficient and do not require much current in order to be visible.

4) 7 segment display:
   - A 9-bit reduced current drive output controls this display. 7-bits of the signal are used to display the robot's mode in hexadecimal format. 1-bit is used as an indicator of the kicker IR status, as seen by the main microcontroller. 1-bit is used to provide the reduced power to the display. The reduced drive mode is used to improve power consumption. The display is still very visible when driven with the lower current mode.

5) Horizontal dribbler encoder with FPGA and motor control:
   - The encoder on the horizontal dribbler motor outputs two signals in quadrature format representing rotational velocity, which is handled by the FPGA for increased resolution. The FPGA then sends the main microcontroller the velocity data as a single pulse-based signal with the increased resolution and another binary signal to indicate direction. One 16-bit pulse accumulator and a 1-bit input pin are used.
   - The horizontal dribbler's motor circuit needs an 8-bit PWM and a 1-bit output signal from the main microcontroller to control the rotational speed and direction of the motor.

6) Side dribblers:
   - Both of the two side dribbler motor circuits will share an 8-bit PWM output from the main microcontroller to control the spin speed of the motor.

7) Accelerometer:
   - Having two axes for x and y directions, the accelerometer outputs two sets of pulses that represent acceleration in each direction. The timings associated with the pulses that indicate acceleration are read by two input capture pins on the motion microcontroller.

8) Rate gyro sensor:
   - The circuit implementing the rate gyro sensor will output a rotational velocity measurement in the form of a voltage, which requires an A/D input on the motion microcontroller.

9) Four drive motor encoders with FPGA and four drive motors:
   - Each encoder on the drive motors outputs two signals in quadrature format representing velocity data, which is handled by the FPGA for increased resolution. The FPGA then

sends the motion microcontroller the velocity data as a single pulse-based signal with the increased resolution and another binary signal to indicate direction. One 8-bit pulse accumulator and a 1-bit input pin are required per motor in order to capture the velocity signal and direction. Multiply this requirement by four for the four wheel motors.

- Each of the three/four drive motor circuits requires a 16-bit or 8-bit PWM and a 1-bit output signal from the motion microcontroller to control the rotational speed and direction of the motor.

10) Inter-microcontroller communication:
    - Communications between the main microcontroller and motion microcontroller may be established via serial connection, using an SCI port. The main microcontroller sends the velocity data to the motion microcontroller via this serial link. Velocity data sensed by the onboard sensors may be transferred as bytes from the motion microcontroller to the main microcontroller for wireless transmission back to the artificial intelligence computer if a capable wireless module is used.

11) Robot ID or test mode selector:
    - A 4-bit input signal from a 4-position DIP switch indicates to the main microcontroller what the robot's ID should be or which test mode to run.

Based on the finalization of the interface, both the hardware and software designs of the microcontroller subsystem could be completed. Pins on the main and motion microcontrollers had to be allocated appropriately to support the functionality needed by each subsystem. The microcontrollers' base circuits, which allow for proper power supply filtering and crystal oscillator input for clock generation, were designed to be the same as the development board used for evaluation, which itself was based on a Motorola reference. *See the schematics in the appendix for the support and operation circuitry.*

The first objective of firmware development was to find the optimal bus frequency for the microcontroller. All tests in the evaluation process had been performed with a 24 MHz bus frequency, however, this frequency limited the maximum standard SCI port BAUD rate to 115,200 kbps. Since the Bluetooth module supported much higher BAUD rate, various frequencies were tested in the 24-30 MHz range. It was found that 26 MHz allowed for stable and reliable communications at 230,400 kbps. 26 MHz was only 1 MHz over the specification. Specifications given by a manufacturer are often very conservative, and there had been no signs of instability running at this frequency for almost two months. The bus frequency change was hard-coded into the bootloader by modifying the bootloader's assembly code.

One other change in the bootloader involved using one microcontroller input pin to set the operating mode of the microcontroller rather than two input pins as on the development board.

The next task was to develop the code for each microcontroller to handle all operations associated with the hardware interfaces. This development was facilitated by dealing with each component of the electrical system in separate functions, creating a modular code structure. A modular code structure has greater maintainability and flexibility for changes to the code.

3.7 Operations of the main microcontroller

The initialize() function of the main microcontroller sets various ports to their desired configurations and initializes variables. First, the I/O ports are configured for input or output mode, reduce current drive, pull-up/pull-down devices, and interrupt conditions. The properties of conversion for the A/D ports are then configured, and the appropriate A/D pins are chosen for continuous conversion. Next, the timer port is configured for the main timer, interrupts, pulse accumulator, and modulus down counter. The PWM port is then set up. SCI ports and the wireless decoding method are selected based on input from the 2-position DIP switch. The robot's ID or test mode is determined by calling the function getRobotMode(). Variables associated with timing, kicking, dribbling, and wireless are initialized. The starting states of controls are set at the end of this function.

The main() function calls the initialize() function to initialize all settings of the microcontroller and enables all interrupts afterwards. A check is done on the SCI port associated with reception from the wireless module to see if any data was send while the microcontroller was initializing. Any data that was received is throws out to allow for new data to be received. An initial check on the battery voltage is performed by calling the checkBattery() function. Another check is carried out to call the testModes() function for the robot to perform a test mode if its DIP switches were set accordingly. The rest of the main() function is an infinite loops that polls for the setting of certain flags. These flags indicate if the microcontroller should execute the h_drib() function for the horizontal dribbler's feedback loop or execute the checkBattery() function to check and display battery voltage.

Reading the robot's ID or test mode is done through the getRobotMode() function. In this function, the 4-position DIP switch input is read and the robot ID or test mode is extracted and displayed on the 7 segment display. This function is only called upon startup during initialization of the microcontroller.

Once every 16 or so seconds, the checkBattery() function is executed. Execution occurs when the modulus down counter interrupt has occurred 1000 times at a frequency of 60 Hz and a flag is set,

indicating to the infinite loop in main() to execute the checkBattery() function. This function reads the voltage from the battery, scaled down to about 5V maximum through a resistor divider, and outputs to the 4-LED battery meter based on a range of voltages.

The robot performs a kick action only if the AI computer has sent the command to do so. If AI has told the robot to kick and the robot has the ball (sensed by input from the kicker IR circuit), the main microcontroller will execute the kick() function and kick with one of seven selected kick speeds. Kick speeds are based on the amount of time the kicker circuit's switch is held open. Before the kick occurs, the DC/DC converter is enabled to provide power to the kicker's circuit and capacitor. A stop control is sent to the horizontal dribbler to quickly stop the spin caused by dribbling the ball, and the kicker switch is activated to perform the kick. Once a kick has occurred, an output compare timer prevents further kicking for a short delay to allow the kicker's plunger to retract. Another output compare timer allows the DC/DC converter to charge the kicker circuit capacitor for a few seconds before disabling the DC/DC converter for power conservation. In the case where AI has commanded a kick but the robot does not have possession of the ball, a kick flag will be set so that the robot will kick as soon as the IR beam is tripped and the IR sensor signals the microcontroller.

Side dribblers are simply enabled and disabled by the v_drib() function call whenever AI sends a command. The horizontal dribbler is a completely different beast. The modulus down counter, interrupting at 60 Hz, also sets the flag for the main() infinite polling loop to execute the h_drib() function at 60 Hz. The feedback-control algorithm in h_drib() uses the rotational speed collected by a pulse accumulator and direction input to maintain a fairly constant dribbler rotation velocity by varying the PWM and direction outputs to the horizontal dribbler motor. One of the four possible dribbling speeds is chosen by AI. The pulse accumulator obtains the actual dribbling speed by counting the number of motor encoder pulses, quadratured. The h_drib() control algorithm attempts to correct for differences between the actual dribbling speed and the desired dribbling speed.

Wireless reception is handled by an SCI port that interrupts every time a new data byte is received from the wireless module. Using Manchester decoding, the bytes received by the SCI port are checked for detection of a valid data packet. If the correct sequence of detection bytes is received, the reception of valid data begins. Any corruption of the Manchester-encoded bytes received in the middle of a packet invalidates the entire packet. A valid packet is deemed complete when 25 bytes have been successfully received after valid packet detection. The processPacket() function is called to interpret the data received in a valid packet. processPacket() parses the data packet for AI commands. A dive play command, which was rarely to never used in 2002, is first checked. If no dive play is commanded, motion velocities

and sensor calibration data from AI are sent via the other SCI port to the motion microcontroller. Afterwards, AI commands for kicking, side dribblers, and horizontal dribblers are processed and executed.

In the event where wireless reception is interrupted, the lack of wireless data for a specified time prompts the robot to stop all actuators with the shutdown() function. This is a safety precaution in case the robot is in motion when the loss of wireless reception occurs.

Various robot test modes allow for quick inspection of a robot's functionality. Executed by main() if the appropriate robot mode is selected with the 4-position DIP switch, the testModes() function performs basic maneuvers and tests. Two tests enable both the side dribblers and horizontal dribbler, one with the horizontal dribbler spinning at its slowest speed and the other with the horizontal dribbler at full speed. A third test is solely to check the drive motors. The robot is commanded to move in a box pattern and then rotate back and forth. This demonstrates motion in both positive and negative directions on the x and y axes as well as for rotation. The last test mode commands the kicker to kick at full speed when the IR sensor is triggered, showing proper operation of everything related to kicking.

3.8 Operations of the motion microcontroller

The initialize() function of the motion microcontroller is much like that of the main microcontroller. The SCI port is configured to match the BAUD rate of the transmitting SCI port on the main microcontroller. I/O ports are configured for appropriate input or output modes, reduce current drive, pull-up/pull-down devices, and interrupt conditions. A/D ports are then configured. The timer port is configured for the interrupts, input captures, pulse accumulators, and modulus down counter. The PWM port is setup for four PWM outputs. Variables associated with timing and sensor calibration are initialized.

The main() function is also similar to that of the main microcontroller. It calls the initialize() function to initialize all settings of the microcontroller and enables all interrupts afterwards. Calibration of the onboard sensors is then done. The rest of the main() function is an infinite loops that polls for the setting of certain flags. These flags indicate if the microcontroller should execute the processPacket() function for new data from the main microcontroller or execute the control_loop() function for feedback-control operation.

When the main microcontroller sends new velocity data to the motion microcontroller, the SCI port causes an interrupt to receive this data. Upon reception of all velocities, a flag is set. This flag calls for the execution of processPacket() in the main() function's infinite loop. processPacket() extracts velocity representations for x, y, and theta as a scaled magnitude and direction. The actual velocities are

calculated by undoing the scaling of the magnitudes and combining the result with the proper sign of the velocity direction in relation to the robot. Additional information regarding gyro and accelerometer sensor calibration is also calculated from received data.

Occurring at a rate of 300 Hz, the control_loop() function contains all of the calculations, inputs, and outputs of the robot's feedback-control system. A modulus down counter runs at 3000 Hz, ten times the control rate. This is to prevent overloading the 8-bit pulse accumulators by grabbing the drive motor's encoder/FPGA counts detected by the accumulators before the motor could possibly have rotated enough to reach 8-bit limit of the accumulator. The counts recorded at each of the ten interrupts of the modulus down counter are then totaled. After the ten interrupts, a flag is set to cause the execution of the control_loop() function. The beginning of control_loop() gathers all available information regarding the current motion of the robot and desired motion. Current motion is given by the counts from the pulse accumulators for each motor as previously described. The voltage read from the gyro circuit by the A/D of the microcontroller is translated into actual rotational velocity. The accelerometer's x and y axes outputs are read by two independent input capture pins that interrupt each time a high or low signal from the accelerometer's output is detected. A time reference is taken each time the interrupt occurs, and the time the x and y outputs are high and low are calculated. A ratio involving these high and low times is used to calculate the sensed acceleration. The end of the control_loop() function consists of setting the newly determined wheel motor directions and PWM signals needed to achieve the velocities desired by AI.

3.9 Results

Initially, everything except the wireless reception algorithm responded positively in some way. The microcontrollers' features performed as expected, due to the extensive process of evaluating each feature thoroughly before implementation. Tweaking of values and adjustments to various algorithms was required to get desired results for the actuator controls. Some issues were related to conversion between float and integer data types and between signed and unsigned data types. Another issue involved overloading the 8-bit pulse accumulators for the wheel motors on the motion microcontroller. This problem was solved by increasing the number of times the pulse accumulators are read per control loop cycle. Any other problem, including the wireless reception algorithm, was related to faulty logic in the algorithm or slight error in the code, all a result of human error. Since the microcontrollers' base circuits were designed to be the same as the development board used for evaluation, it was not a surprise that no problems were experienced regarding basic operation of the microcontroller. Therefore, all faults were correctable by changes to the firmware run on the microcontrollers.

Performance of the Motorola microcontrollers was verified to be enough to support the intense calculations needed for the new motion control algorithm at a control loop sampling rate of 300 Hz, which was the same rate used in the 2002 motion microcontrollers. The motion control loop, as tested, included complete support of the rate gyro and its calculations, but did not the accelerometer's calculations. However, the tested control loop had a maximum capable frequency of 380 Hz, which gives plenty of room for the accelerometer-based calculations to occur before slowing the control loop down to near 300 Hz.

3.10 Conclusion

It would have been interesting to get a powerful 32-bit processor for the sake of having a ridiculous amount of room for expandability, though the Motorola HCS12 microcontrollers offer adequate features and performance for the 2003 electrical system, which accomplishes one of the original design goals of the microcontroller subsystem. One option that could have been explored was to use a 32-bit Motorola M*Core microcontroller for the most intensive calculations and algorithms and a separate 16-bit Motorola HCS12 microcontroller for special I/O features. However, it would have required significantly more time to learn how to develop for and evaluate two microcontrollers of different families. As it turns out, the plan for the 2004 robots is to move to using a PC-104 single-board computer design based on a 32-bit Pentium processor, which would eliminate the need for a more powerful microcontroller for data processing and calculations. However, it still may be useful to have one or two microcontrollers onboard to handle all of the hardware I/O interfaces with the rest of the electrical system and communicate commands and sensor information with the PC-104 processor via a communications port.

Future work that will be carried out this summer consists of optimizing the microcontroller code for maximum performance when executing all of the tasks required of the electrical system. Features need to be added as contingencies. For example, AI should be capable of disabling rate gyro and accelerometer data in the feedback-control loop in the unfortunate case where one or both of these sensors fails during a game (a failure could be detected by erratic robot behavior seen by the vision system). Other work to be done to the microcontroller system is to ensure that our robots are in their best shape to compete, with regards to response times, data feedback (if possible), and reliability and robustness of the overall system.

Code

Motorola MC9S12DP256B vs. Microchip PIC16F877 Floating Point Test Code

```
void control_loop() {

        float Xk1 = 83234.23423;
        float Xk2 = 23498.53453;
        float Ts = 3980.2534;
        float dif1 = 3895.2534;
        float dif2 = 9844.294;
        float INTEGRAL_STATE = 9845.22;
        float cnt1des = 90823.15;
        float cnt2des = 42342.309;
        float V_MAX = 3.3523;
        float Xk3, Xk4, Xk5, Xk6, temp1, temp2;
        float Ki = 238234.234;

        Xk1 = Xk1 + Ts*dif1;
        Xk2 = Xk2 + Ts*dif2;

        Xk3 = INTEGRAL_STATE*(cnt1des/V_MAX)*(cnt1des/V_MAX);
        Xk4 = -INTEGRAL_STATE*(cnt1des/V_MAX)*(cnt1des/V_MAX);
        Xk5 = INTEGRAL_STATE*(cnt2des/V_MAX)*(cnt2des/V_MAX);
        Xk6 = -INTEGRAL_STATE*(cnt2des/V_MAX)*(cnt2des/V_MAX);
        temp1 = Xk1*Ki + Ki*(dif1);
        temp2 = Xk2*Ki + Ki*(dif2);

        Xk3 = INTEGRAL_STATE*(cnt1des/V_MAX)*(cnt1des/V_MAX);
        Xk4 = -INTEGRAL_STATE*(cnt1des/V_MAX)*(cnt1des/V_MAX);
        Xk5 = INTEGRAL_STATE*(cnt2des/V_MAX)*(cnt2des/V_MAX);
        Xk6 = -INTEGRAL_STATE*(cnt2des/V_MAX)*(cnt2des/V_MAX);
        temp1 = Xk1*Ki + Ki*(dif1);
        temp2 = Xk2*Ki + Ki*(dif2);
}
```

This function was called several times in the main “while” loop that runs infinitely on the microcontrollers. The number of times the function could be called was limited by amount of memory available on the PIC microcontroller. The main function running the infinite loop was essentially the same for both microcontrollers’ compilers, except for the different commands for setting an output pin high and low. Below is the listing of the main function used by the Motorola microcontroller’s compiler:

```
void main(){
        int temp = 0;
        PORTA = 0x00;                   // output_low(PIN_B7) for PIC

        while(1) {
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();
```

```
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();
                control_loop();

                if (temp == 0) {
                        PORTA = 0xFF;// output_high(PIN_B7) for PIC
                        temp = 1;
                }
                else if (temp == 1) {
                        PORTA = 0x00; // output_low(PIN_B7) for PIC
                        temp = 0;
                }
        }
}
```

ON-BOARD SENSOR SUBSYSTEM SECTION 4

4.1 Introduction

The use of a 4-wheel omni directional drive in 2002 brought a significant increase in acceleration and top speed for the robots. At the same time a severe problem emerged: the issue of controllability. While the robots were able to move very fast it happened that they deviated significantly from the commanded trajectory and ended up facing a completely wrong direction. This heavily impacted precise passing and shots on the goal.

The major problem was slipping wheels. The amount of friction between the wheels and the ground is limited. Whenever very high accelerations were commanded, the wheels skidded and the robot could go out of control. The local control loop could not combat this problem, since according to the encoder readings, all wheels were moving just as they were supposed to. The robot as a whole just could not follow.

Due to the system inherent latency (about 10 frames ~167 ms) Intelligence & Control was not able to correct for deviations quickly enough. Prediction was assuming the robot was still on track. When the new vision data with the skidding robot arrived it was already too late to correct for this. The faster the robot moved, the more severe this problem, since in the same amount of time the robot could deviate more from its commanded trajectory.

Another reason why the robots are harder to control are the dynamics of the 4-wheel arrangement itself. The mapping of wheel velocities to robot velocity is not unambiguous. For more information about this, refer to Tama's sensitivity analysis.

In order to tackle these issues we decided to put some sensing onto the robots. If the robots are aware of their current velocity they are able to correct for any deviations and follow a given trajectory more closely. This chapter explains the reasoning behind the on-board sensor design, why we picked which sensors and how we implemented them.

4.2 Preliminary Analysis

The first step was problem analysis. All we knew was that the wheels were skidding and the robot was out of place. It was important to know whether the main problem was the deviation in rotation (orientation of the robot) or translation. A small analysis gave ballpark figures of the problem:

Assume that a robot is skidding out and the velocity difference (= difference between commanded and actual velocity) for each wheel is:

$$v_{diff,i} = v_{comm,i} - v_{act,i} = 0.1 \text{ m/s} \qquad \text{... Eq 4.1}$$

In fact we don't know this number but it should be good enough for a very rough guess. In fact, the exact number doesn't really matter for the following analysis. It should be noted that this velocity is not the revolutions per second of each wheel but the velocity of the whole drive module (see figure 3.1).

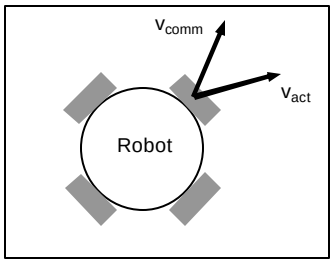


Figure 4.1 Sketch of drive module velocities

Consider 2 extreme cases:

a) The velocity deviations add up to a pure translation (i.e. all $v_{diff,i}$ point in the same direction)

   Traveled distance in 10 frames: $s_{diff} = 0.1 \text{ m/s} * 0.167 \text{ s} = 0.017 \text{ m}$

   This deviation is a bit larger than our vision error but still fairly small.

b) The velocity deviation adds up to a pure rotation (i.e. all $v_{diff,i}$ point in a circle)

   Distance wheels-center of robot: $l_{wheel} \approx 0.07 \text{ m}$

   Rotated angle in 10 frames: $\alpha_{diff} = 0.1 \text{ m/s} * 0.167 \text{ s} / 0.07 \text{ m} = 0.24 \text{ rad} \approx 14 °$

   When we kick the ball in this situation (orientation off by 14 °), the deviation between the desired ball position and the actual ball position is a function of the distance the ball has to travel. Assuming 1 m distance, the shot is off by:

   $$1 \text{ m} * \sin(0.24 \text{ rad}) = 0.23 \text{ m}$$

The deviation in case b) is one order of magnitude larger than in case a). In reality, the robot motion will be somewhere in between these two cases but the calculation shows that the error in orientation is much more severe than the translational error. We had this fact in mind when doing our sensor calibration and implementing the controls concept.

4.3 Proposed Solutions

This year, we determined that we wanted to solve the problem of robot control by using local sensing of the robot. Since controllability is mainly an issue at large speeds, the rate sensors need to be very accurate at high rotational velocities. This way, we will have an accurate assessment of the robot's true motion and be able to correct for the error. Given the current state of technology, there are different types of sensors available that can give us information about the motion of the robot. There were several sensor types we could choose from. The sensors were either based on inertial effects, such as an accelerometer or rate gyroscope, or optical, such as in an optical mouse sensor. These sensors could either measure a translational motion or a rotation but not both. In order to monitor all degrees of freedom of the robot we had to use several different sensors at the same time. The different sensor setup options are listed below.

- Use a rate gyroscope to measure rotational velocity and one dual axis accelerometer to measure two degrees of translational acceleration.
- Instead of using a rate gyroscope, use two single axis accelerometers and compute the rotational acceleration by taking the difference in the translational readings, and an additional dual axis accelerometer to measure translational motion.
- Use two optical sensors to measure the translational motion as well as rotational motion. These sensors output position data so velocity of the robot could be found by find the change in position of the robot.

One very important point during the sensor selection was the physical quantity that the sensor measures. Rate gyros measure velocity which was exactly what we wanted. On the other hand, accelerometers measure acceleration, while optical sensors measure relative position. The accelerometer's readings can't be used directly by the control algorithm since our wheels are velocity controlled. We would have to integrate the measurements to yield velocity. In addition, by doing this over a longer period of time we would also integrate the (unavoidable) sensor errors and drift. Depending on the size of these errors the calculated velocity would be off by a certain amount, which could render the data useless. The optical

One way to tackle the problem of integrating sensor error and drift is to use vision data in regular intervals. We could send the latest velocity data to the robots and use both the vision data and the on-board sensor data to get a better estimate of how the robot moves. The downside of this is that it requires a larger wireless bandwidth and a lot more computation on board.

The optical sensors provide position information without any calibration. They are easy to maintain, are not susceptible to electro-magnetic interference and provide precise relative X-Y positions. Research was done on these sensors to find out whether they can handle vibrations and collisions of robots, and if they could measure positions at high speeds. Agilent's latest powerchip that powers the Logitech MX700 mouse came closest to fulfilling the standard requirement specifications but its maximum velocity specification was still three times less than our requirement. Agilent confirmed that the software configuration on this chip can be set to increase the speed by up to four times. However, the robot speed has also increased this year and the requirements will be even higher. The datasheet and samples of this chip are not available until May 2003.

Other considerations of these sensors were how they would be mounted and the size of each sensor. Since there is limited space on our robot for additional devices, the sensors needed to be small and mounted in a way that would allow it the ability to take accurate measurements. For example, the accelerometers are very susceptible to tilt and vibrations, so it was important to consider a steady horizontal mount for them that would keep vibrations to a minimum. The optical sensors needed to be mounted as close to the ground as possible. Size is another consideration. Most gyroscopes used in industrial applications are large since they are used in helicopters and such. It was important that we find a small gyroscope that could still measure at the speeds of our robot.

After being researched, it was determined that the option of one rate gyroscope and a dual axis accelerometer would be implemented. The optical sensors that were available at the time were not designed to track at high speeds very accurately; they're only for optical mice, which operate at relatively low speeds compared to our robot. Location and mounting considerations for the optical sensor was also an issue since it needed to be mounted about 5 mm or closer to the ground. Accelerometers on the other hand, is a more developed technology and devices that match our specifications for high speed and small size could be found. Though difficult to find, gyroscopes used in robotic applications are available and are accurate enough to measure rotational speeds of up to 575 degrees/sec. As a result, we have incorporated a rate gyroscope and dual axis accelerometer to track all ranges of motion of the robot.

 Tj -re h W n BTn24 0 258443.25 760.5 TD 1

Model	Manufacturer	Max Angular Rate (º/sec)	Bandwidth (Hz)	Scale Factor (mV/º/sec)	Temp. Drift (%/ºC)
CG-16D	Tokin	± 90	100	1.1 ± 20%	± 15
MG100	Gyration	± 150	10	1.11 ± 10%	± 0.2
CRS03-11	Silicon Sensing	± 573	50	3.49	± 5

Table 4.1 Gyros that were researched

Since our robots can rotate at speeds up to 7 and 8 radians/sec, the Tokin and MG100 didn't meet our requirements in measuring rates at high speeds. However, the CRS03-11 can detect motion up to 10 radians/sec, which is sufficient for any maneuvers our robots can make.

At rest, each gyro outputs a reference voltage (V_{ref}) and the output voltage swings above and below this V_{ref} depending on the direction of rotation. In order to find angular rate in radians/sec, the following equation was used to find the rate of rotation.

$$rate = [V_{out} - (\frac{V_{dd}}{2})] \Big/ (SF \times \frac{V_{dd}}{5}) \qquad \text{... Eq 4.2}$$

where V_{out} = output voltage, V_{dd} = supply voltage, and SF = scale factor of the gyro.

Temperature drift and noise were other issues that needed to be addressed when testing the gyros. By examining the analog signal that came out of the sensor and processing it to find rate and position, we were able to see that though the signal was strong, there was still a level of high frequency noise that would need to be filtered via a low pass filter. Also, drift would need to be compensated for. By using Vision to continually recalibrate the sensor offset of the gyro, we were able to minimize the effect of drift on the measurements. The gyro is also going to be encased on the robot to prevent it from heating up since the device is sensitive to high temperatures.

4.4.2 Initial Testing of Gyros

Since this was the first time I had any experience with this technology, much experimentation would be needed to understand fully how the gyro actually worked. A sample circuit was found[1] for another gyro made by Murata but the concepts of filtering and amplifying the output signal of the gyro would be the same for all gyros. This circuit (figure 2) was used to create a test bed for the Tokin gyro.

[1] *www.murata.com/catalog/s42e2.pdf*

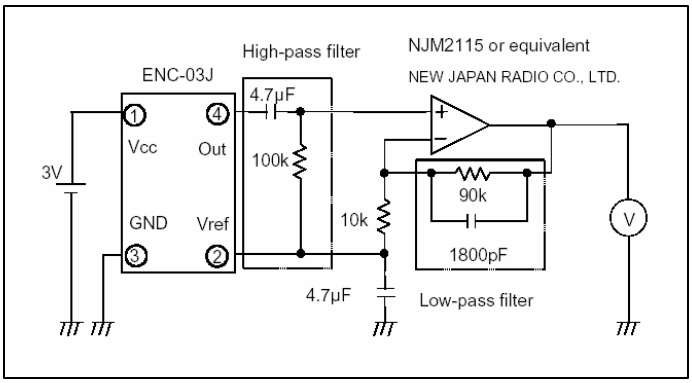


Figure 4.3 Sample Circuit Used as Test Bed

First, the gyro output was examined using the oscilloscope. As the gyro was rotated in positive and negative directions, the waveform on the oscilloscope also moved positive and negative according to the rotation of the gyro. A maximum voltage swing was observed when rotating the gyro at high rates. After constructing the sample circuit in figure 2, the output was again viewed on the oscilloscope. This time, the maximum voltage swing increased. But there were problems. The amplifier in figure 2 has a gain of 9 but the voltage swing did not look like 9. Initially, without the amplifier, the voltage swing of the gyro would be ± .5 V. With the amplifier, the voltage swing increased to ± 2 V. This did not make sense for an amplifier with a gain of 9. It was later noted that the output of the amplifier should be in the form of the following equation:

$$out_{amp} = V_{ref} + G(V_{out} - V_{ref}) \qquad \text{... Eq 4.3}$$

where G *is the gain of the amplifier,* V_{ref} *is the reference voltage of the gyro,* V_{out} *is the output of the gyro and* out_{amp} *is the output of the amplifier.* V_{out} *-* V_{ref} *is equal to the voltage swing for the gyro which was .5 Volts. The reference voltage of the gyro was 2.5 Volts. The output swing of the amplifier was 4.5 volts. If the gain was 9, this would give a voltage swing of about .22 Volts. Since it is .5, the true gain for this circuit was 4.*

To obtain the signal from the gyro while the robot was rotating, the output of the gyro was plugged into connection JP12 on the motherboard. This was a power source as well as ADC to the microcontroller. At this point, the microcontroller was configured only to output raw data from the gyro circuit. A specific rotation was then applied to the robot via the gamepad.

Initially, I used the LCD to display the output of the ADC by programming the main microcontroller to display whatever was on the ADC port at one instant after a certain length of time. The output of the ADC was labeled as G. I also programmed the main micro to display whatever wireless rotational velocity it was receiving as a voltage. This was labeled as T. Thus I could compare what the gyro was sensing to what the robot was receiving and perhaps obtain a lookup table of values for the gyro and rotational velocity packet the robot was getting from the gamepad. Since the PIC micro ADC is 8 bit, we could detect voltage at a resolution of 255, meaning that whatever voltage was detected would display a number between 0 and 255. Results of this test are listed below.

Clockwise @ 5 rad/s	T	127
	G	40
Counterclockwise @ 5 rad/s	T	127
	G	-36

Table 4.2 Results of LCD

This seemed to make sense since our positive rotation displayed G as positive and negative rotation displayed G as negative. But the magnitudes of the ADC were in no way close to what was being sent to the robot. The voltage can be found from the numbers in the table with the following equation:

$$counts/255 \times 5volts = voltage_{robot} \qquad \text{...Eq 4.4}$$

This means that from the gamepad, the robot was receiving a velocity command that equated to 2.5 volts but the gyro was only outputting .78 volts. The counterclockwise calculations should have been the negative of the previous calculation but it wasn't for this test.

A higher resolution of 10 bits was tried to give a voltage count range of 0 to 1023. Unfortunately no good mapping of values between the gyro and the gamepad value was obtained.

Many adjustments were made to the microcontroller code to see whether it was the format of the data that was causing inconsistencies. After rechecking the circuit and verifying that it was indeed correct, I went back to taking data in hopes of finding some sort of algorithm between the values I was getting on the LCD. Taking many different trial runs at different rotational rates from the gamepad in both positive and negative directions, no proportional relationship existed between T and G. Also, this method only took one sample of data. If that sample was noisy, which is very likely, then the data would be unusable and no trend could be found. A table of results is listed below.

Vtheta (gamepad)	Trial	Positive Direction (counts)	Negative Direction (counts)
1 rad/s; T = 25	1	129	126
	2	128	120
	3	129	123
	4	120	126
	5	128	117
2 rad/s; T = 51	1	128	126
	2	121	132
	3	121	122
	4	125	124
	5	122	132

Table 4.3 Results of LCD over many trials

These counts in each trial should have been as close to the T values once again but obviously, there was no trend in the values that were obtained. These tests did not tell me anything about what we were receiving from the gyro.

It was noted that the robot did not turn at the same rate at all times. This may have been because the wheels were very bumpy or the motors were not rotating smoothly. Also, it was noted that the LCD is taking instantaneous samples. Thus if there was any error in the gyro, the LCD could be displaying the error at any moment in time. My thoughts were to be able to average over more samples and see if the value of the average was good to use. This required recording data from the ADC into a memory and then downloading the memory and processing it. This was achieved by writing the ADC output to the EEPROM on the micro.

The EEPROM on the PIC could only hold 256 bytes of data which means that we could only hold 256 samples because each wireless packet is 1 byte. Once this was achieved, the data was downloaded to a PC via Hyperterminal's serial connection. I placed this data into a spreadsheet and plotted out curves over 256 data points which is approximately $256 \times .016666 = 4.26667$ seconds. From the graphs, we could see that the output of the amplifier saturates at the rates in which I took data because the Tokin can only sense 90 deg/sec or 1.5 rad/sec. At these high rates, the data is very noisy and unusable. Getting data from the EEPROM was an improvement but still not accurate enough. I was informed that the Feedback Controls Lab had computers with ADC's that could be used if the robot could be hooked up to it. This would allow for more data storage and post-processing using MATLAB.

4.4.3 Quanser Data Acquisition Board

The Quanser system is located in the Feedback Controls Lab. It consists of data acquisition boards for analog to digital conversion (ADC) and digital to analog conversion (DAC). This was found to be useful

for acquiring data from the gyro circuit rather than using the microcontroller's ADC. In this way, we could store data onto a PC and process it using MATLAB. Simulink was used for its ADC capabilities and oscilloscope function. This allowed me to display the voltage coming in from the gyro through the ADC onto the PC. I could then use MATLAB to plot the values that were obtained because the ADC stored values in an array. A special mount was needed to allow freedom of motion in the connection between the robot and computer. Because the robot would be spinning, we had to use a slip ring in order to allow the robot to spin without tangling any cables that were hooked into the computer. The circuit and gyro also had to be mounted securely to the robot so that it would spin with the robot with the least vibrations and extra motion. Once these goals were accomplished we could obtain analog outputs from the gyro circuit and process them through MATLAB. Some of the graphs that were obtained from processing the data are shown below. They show the raw voltages output from the rate gyro at different rotational speeds. In MATLAB, raw voltage could then be converted to rotational rate. Each gyro has different equations to convert voltage to rate so please refer to each gyro's datasheet before doing these calculations. The following equation was used to find the rate for the Tokin and MG100 gyros:

$$V_{offset} = V_{out@rest} - V_{ref} \qquad \text{...Eq 4.5}$$
$$Rate = (V_{out} - V_{ref} - V_{off}) / scalefactor$$

V_{offset} is the offset of the gyro when it is at rest. V_{ref} is the reference voltage of the gyro, usually 2.5 volts. V_{out} is the output voltage of the gyro and $V_{out@rest}$ is the voltage output of the gyro at rest. The scale factor is the number of mV/deg/sec of the gyro. The following graphs show the output of the gyros sampled at 60 Hz. Depending on how fast the ADC of the microcontroller is, this sampling rate may have an upper limit. The more samples of data, the more accurate the readings may be.

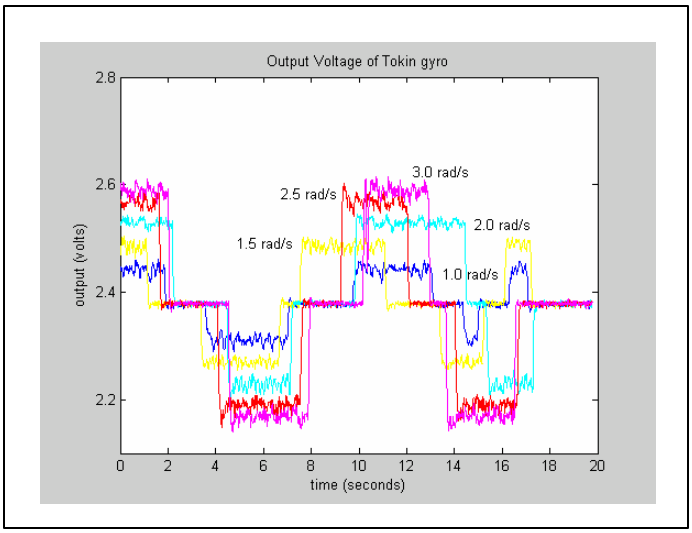


Figure 4.4

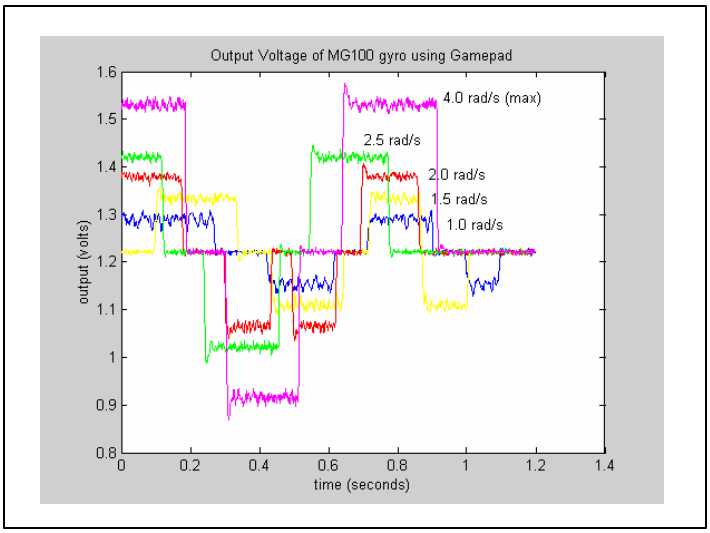


Figure 4.5

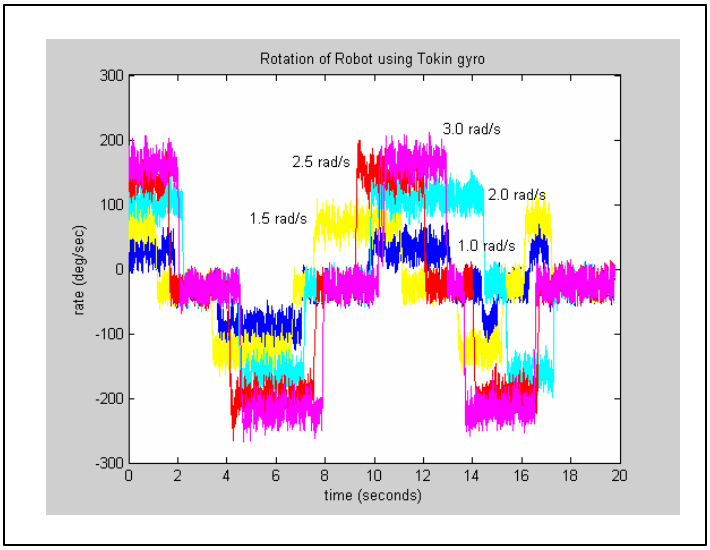


Figure 4.6

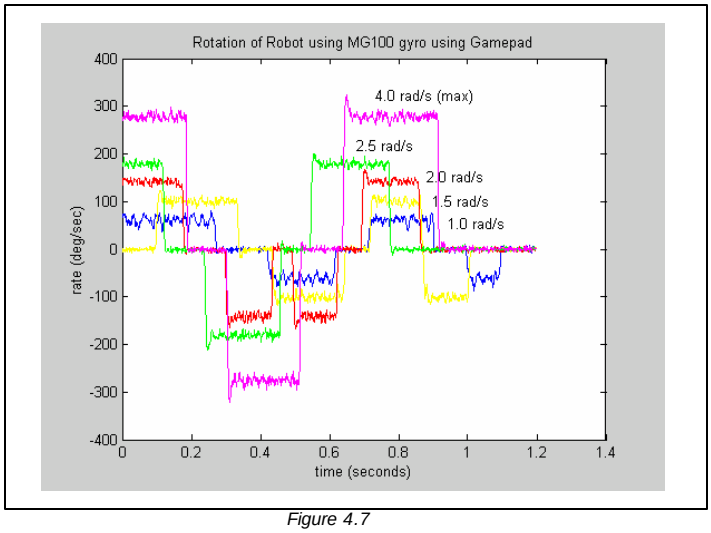


Figure 4.7

As soon as we received the CRS03-11 rate gyro we tested it using the inverted pendulum from the feedback controls lab. The Quanser system could measure the rate of rotation of a pendulum as it swung as well as accept the analog data from the gyro circuit as it was taped to the swinging pendulum (see figure 4.8).

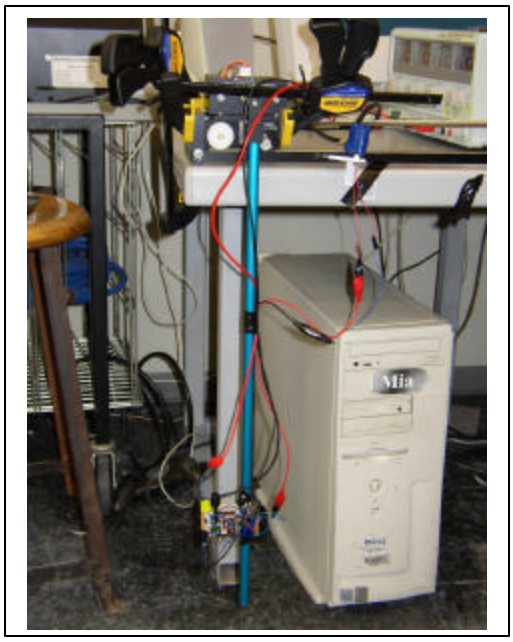


Figure 4.8 Rate gyro attached to pendulum

Hence we could compare these two sets of data and see how good the measurements of our rate gyro were. Figure 4.9 shows a comparison of the rate gyro data and the "true" velocity as perceived by the built-in encoder of the pendulum.

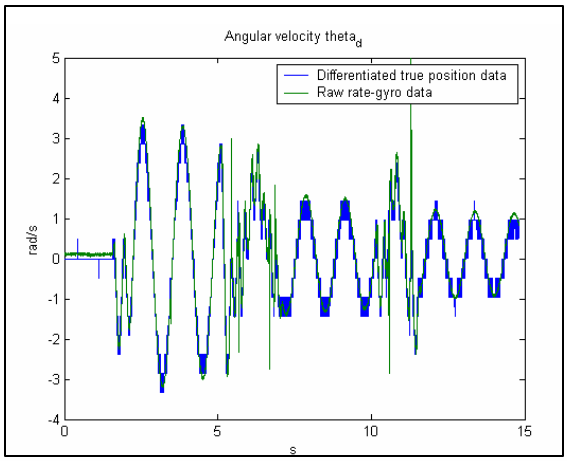


Figure 4.9 Comparison of rate gyro and pendulum data

The data provided by the rate gyro is actually very good. The readings are close to the true values. The true data is not a smooth curve but rather "lumps together". This is due to the relatively low sampling frequency of 60 Hz. We wanted this experiment to be as close to the final system as possible (60 Hz vision system). The difference between true and measured velocity is at max 0.17 rad/s. The largest deviations occur when the rate gyro data "spikes". This happened only when we were moving the pendulum very fast by hand. The spikes might stem from some nonlinear effects, such as the rate gyro board hitting the pendulum, saturation or excessive linear acceleration.

Looking at this data we decided that we would go with the CRS03-11 rate gyro in the final version of the 2003 robots.

4.4.4 Filter Design

Both accelerometers and gyros output an analog signal that needs to be converted into a digital signal that can be used by the microcontrollers. However, the signals that are output from the sensors tend to be noisy and require a filter before being converted into digital data.

There was a chance that the output of the gyro did not even need to be filtered. The output of the gyro was tested using the Quanser DAQ system to examine how noisy the signal from the CRS03-11 really was. First, the output of the gyro was sampled at 1 kHz at rest over 20 seconds. The mean of the output was then subtracted from the first 10 seconds of the signal so it would only leave the noise of the output. Remember that this is the noise of the output that would affect the readings for rotational velocity. This noise was then divided by the scale factor to get how many degrees/sec the noise added to the output

signal. Then this was integrated to see how much the measured position changed due to noise. It was seen that over the last 10 seconds, the position changed by about .4 degrees which is not bad. At first, it was believed that a filter would not be needed. However, after a Fourier transform analysis of the position drift, it was shown that there was noise at high frequencies of the signal. This coincided with the advice from Silicon Sensing that a low pass filter would be needed to filter significant high frequency noise of the gyro signal.

From figure 4.3, we see that there is a high pass filter and a first order low pass filter attached to the output of the gyro. The high pass filter is there to eliminate any DC bias as well as eliminate the long term average of the gyro. This in effect, cancels any long term drift there is in the gyro. However, since we can use Vision to recalibrate the gyro, there is no need for the high pass filter. The drift is compensated for in the recalibration process. A low pass filter would be needed to filter any high frequency noise that existed in the analog signal. The final circuit used to filter and amplify the gyro signal is shown in figure 4.10.

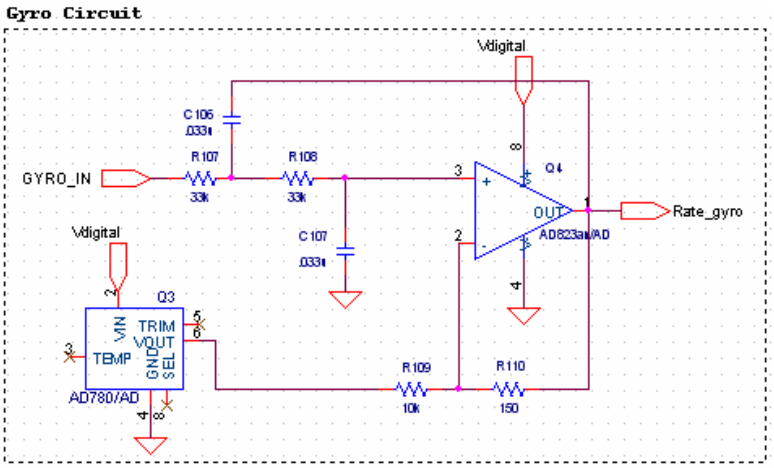


Figure 4.10 Gyro Circuit

At this point in time, all initial tests with the Tokin and MG100 gyros were finished. The CRS03-11 gyros were received and all testing was done on them. The filter in figure 3 is a second order Butterworth filter. The characteristics of this filter are a maximally flat amplitude response governed by the following equations:

$$\frac{V_{out}}{V_{in}} = \frac{1}{[1+(f/f_c)^{2n}]^{1/2}} \qquad \text{...Eq 4.6}$$

$$RC = \frac{1}{2\pi f_c} \qquad \text{...Eq 4.7}$$

where n is the order of the filter.

According to the second equation, any RC combination that results in a cutoff frequency (f_c) of 100 Hz could be suitable. Most resistors are on the order of kilo-ohms while capacitors are on the order or micro- or nano- farads for these filters. However, f_c had to be experimentally determined to be 100 Hz. This is because when tested, the filter did not exhibit 70%, or –3dB, voltage at 100 Hz. Originally, values of 33K and .047uF were used for the resistor and capacitor of the filter to get a cutoff of 102 degrees. When a 5V sine wave at 100 Hz was passed through this filter, 4.125V was output. This is 82.5% of the original input voltage when it should be 70%. The capacitor was then changed to .033uF and using the same method, the voltage of a 5V, 100Hz sine wave was filtered to be 3.625V that is 70% of the input voltage. These values of the resistor and capacitor were then used for the filter.

This filter was decided on because the bandwidth of the gyro output is 50 Hz. The 100 Hz cutoff frequency would ensure that all data within 50 Hz would be passed and all noise above 100 Hz would not be passed as part of our signal. The active part of the filter represents an amplifier with unity gain so that the original signal from the gyro is passed with no attenuation. The AD823 op-amp was chosen for its excellent rail-to-rail voltage characteristics. We want this type of op-amp to get the maximum voltage swing of the gyro. The AD780 is a 2.5 rea 13.5 TD tage of a

failing, there was no output for a little while. This will be solved by insulating the gyro from the parts of the robot that could become hot.

4.5 Technical Overview of Accelerometer

Accelerometers detect acceleration and output an analog signal that is proportional to the acceleration of the device to which it is attached. The acceleration is measured in *g* for the accelerometer chosen for RoboCup. Unfortunately, there was not as much research of different types of accelerometers as gyros due to lack of time and resources. However, one device was found to have characteristics that were suitable for use on our robots. This was Analog Devices' ADXL210E. The ADXL210E is a dual axis accelerometer that can output analog or digital data. A circuit diagram is shown below.

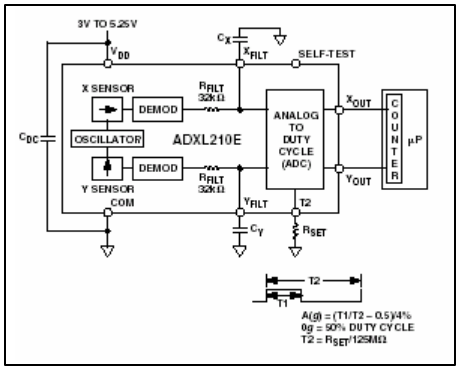


Figure 4.11 ADXL210E Diagram

When examining this accelerometer, the characteristic that was most important was noise level. At 60 Hz bandwidth, the noise floor was 1.96 m*g*. Another issue is the measurement error due to gravity. Whenever the sensor is tilted it measures the component of the gravitational force that points in its measuring direction. That means if the field was tilted or the accelerometer was not mounted correctly, the accelerometer readings would not be accurate. Other issues that make accelerometers difficult to work with are how vibrations or collisions will affect the readings of the accelerometer. These issues were not fully explored this year due to time constraints but the accelerometer circuit has been designed and tested on its own (not attached to a moving robot) and operates properly.

In order to use the acceleration data taken from the accelerometer in the same way as the gyro, we would need to integrate the data from the sensor in order to achieve velocity data. If there were any offsets due to improper mounting, they would remain the same and can be found using vision data, which would have

to be sent to the robots. Thus, if there is any bias or offset in the accelerometer, we can use Vision and compare "true data" to accelerometer data in order to recalibrate the accelerometer. The calibration algorithm to address problems of drift and bias for the accelerometer and gyro were done by Oliver Purwin (op24@cornell.edu).

As with the gyro, the data from the accelerometer must be filtered before is can be converted into digital data. As seen in figure 2, there is a low pass filter after the sensors to filter the analog signal before it is converted to a digital signal. So the ADXL210E already has a built in low pass filter. The output of the accelerometer is duty cycle modulated (DCM). At rest, the output is a 50% duty cycle. Higher and lower accelerations are described by the equation

$$A(g) = (T1/T2 - .5)/4\% \qquad \text{...Eq 4.8}$$

where A(g) is the acceleration in *g*, T1/T2 is the duty cycle, and 4% is the percent change in duty cycle per g^2.

4.5.1 Accelerometer Selection

The MEMs accelerometers that were looked at for Robocup this year were from Analog Devices. The options were narrowed down to dual axis accelerometers in order to obtain X and Y motion and to reduce the number of elements that would be on the circuit boards. Duty cycle outputs were also an advantage of certain accelerometers over others since no ADC on the microcontroller would be needed. The ADXL210E was finally chosen because it contains all these elements as well as having built in filtering capabilities which could be adjustable depending on the application used. It could also measure up to 10*g* which is more than enough for any accelerations the robots may make.

4.5.2 Setup of Accelerometer

In order to obtain data from the accelerometer, the outputs need to be interfaced with a microcontroller to take the data and calculate the duty cycle by using the microcontrollers interrupt ports. The resolution of measurements depends on the sampling rate of the accelerometer and how fast the microcontroller can accept these samples. Since the accelerometer output is DCM, a percentage of a period represents the acceleration.

[2] *This can all be found in the data sheet for the ADXL210E*

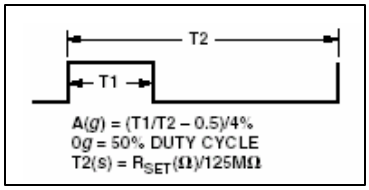


Figure 4.12 Duty Cycle Modulated Signal

T2 is the period of the duty cycle while T1/T2 is the actual measurement of acceleration. The accelerometer was first connected to a PIC microcontroller which was programmed to count the length of time of T1 and the length of time T2-T1. Adding these together gave the length of time of T2 and acceleration could then be found. The microcontroller's clock, used to count how long T1 and T2 were, was customized in order to allow enough processing time between interrupts, calculations, and printing to a serial port. The micro clock was set at 1.25 Mhz. The sampling rate, 1/T2, could be adjusted on the accelerometer by using a resistor, R_{set}. The sampling rate was set at 600 Hz by using a 200K resistor for R_{set}. 600 Hz was chosen because out control loop runs at 300 Hz and it would be easy to synchronize data with a 600 Hz signal being output from the gyro. According to the datasheet, the analog signal must be one-tenth the frequency of the DCM signal to minimize DCM errors. Since we've set our DCM frequency to 600 Hz, the filter must be set to <60 Hz. The filter was set to filter the analog signal at 50 Hz by connecting a .1uF capacitor to C_y and C_x. These settings correspond to a resolution of .012 *g*/count of time, which is sufficient for our application. This is found via the following equation:

$$resolution(g/count) = [\frac{clockrate}{samplerate}(.04)]^{-1} \qquad \text{...Eq 4.9}$$

Inputting a square wave from the function generator tested the code on the microcontroller used to detect the counts of the duty cycle. This simulated T1 and T2. The number of high and low counts was correctly proportional to the duty cycle.

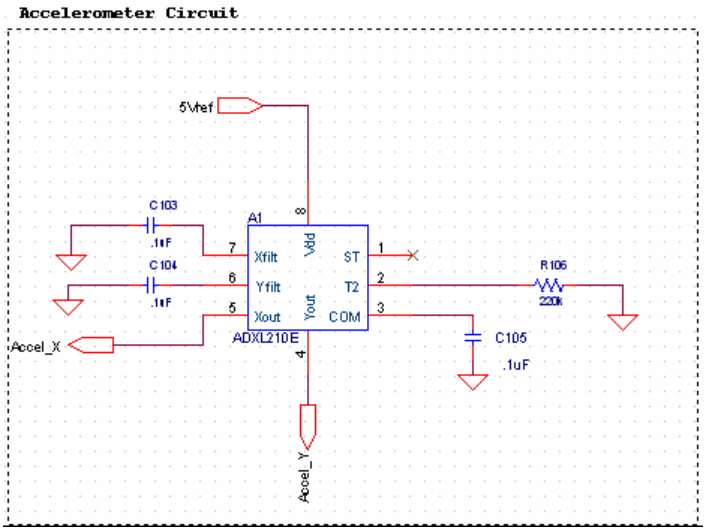


Figure 4.13 ADXL210E with 5V reference

4.5.3 Quanser Cart

Once again, we were able to use the Feedback Control Systems lab to test the functionality of the accelerometer. A breadboard was made to connect the components of the accelerometer and i ts circuit to a PIC microcontroller that would take in the DCM signal and output the counts of T1 and T2 to the serial port. A program that took serial data and put it into a text file was used to record the output data for processing in MATLAB. The ability of the accelerometer to measure acceleration was measured by taping the breadboard to a cart that is used in one of the labs in the Feedback Controls class (see figure 4.14 below). This position of the cart as we moved it was then used as a truth model to compare the accelerometer output.

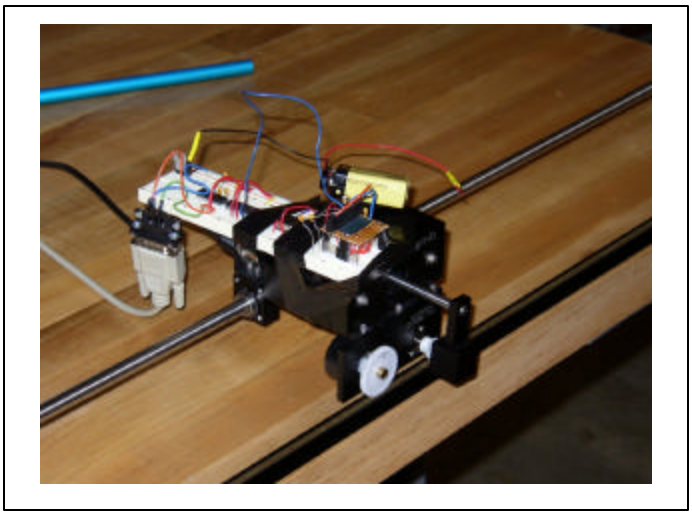


Figure 4.14 Accelerometer board attached to cart

We compared the two sets of data and checked the quality of the accelerometer readings. Figure 4.15 shows a comparison of the accelerometer data and the "true" acceleration as perceived by the built-in encoder of the pendulum.

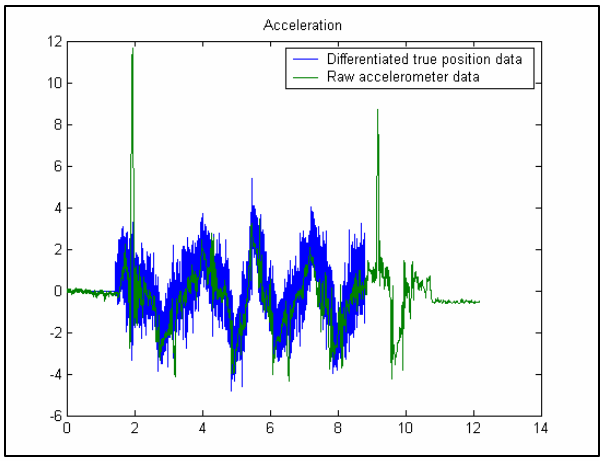


Figure 4.15 Comparison of accelerometer and pendulum data

The post processing of this experiment was more difficult than for the rate gyro since the two data streams were not synchronized. We had to use the serial port of the PC (not the Quanser system) in order to log the accelerometer data. The synchronization had to be done by hand which introduced errors of at least 0.05 s. Having badly synchronized signals makes the velocity estimation a lot harder and increases the amount of error.

On top of that, the accelerometer signal is much noisier than the rate gyro signal. In order to compensate we will have to implement some digital filtering. The most severe point is the fact that we can't directly use the readings from the accelerometer. We will have to integrate them to yield robot velocities. This integration process requires additional data from the vision system. Because of the reasons above, we have to test the system first on a final robot before we equip the whole team with accelerometers. The lead time for these components is short and the necessary circuits are already in place so they can easily be retrofitted.

4.5.4 Issues

Some characteristics of the accelerometer that were mentioned in the datasheet did not prove true when tested.

- The noise floor at certain bandwidths was not the same as specified in the datasheet. The noise was about 5 times bigger than it should be.
- Because the low pass filter was configurable by only adding a capacitor, it is difficult to measure the cutoff of the filter to test the bandwidth of the filter.
- The input voltage of the accelerometer must be as constant as possible or else it will induce errors in measurements. (The reference voltage in figure 5 was used to accomplish this)
- A digital filter may be needed on the microcontroller side to attain even cleaner data from the accelerometer.
- Higher clock rates on the PIC for higher resolution were not utilized because the microcontroller would freeze when processing time for interrupting, calculating acceleration, and outputting to a serial port were greater than the rate in which data was flowing in.
- A chip socket (LCC-8-1.27-01) made by Enplas was used to mount the accelerometer to our boards. This is to keep the accelerometer horizontal and makes it easy to change if it ever broke.

4.6 Sensor Calibration with Vision Data

To combat drift of the local sensors (rate gyro, accelerometers) due to temperature changes, we plan to use the vision signal to calibrate these sensors in regular intervals. We will compare the real position of the robots, as perceived by our vision system, with the measured position (by local sensors) and calculate the sensor deviation. This allows for relatively high precision of our local sensing and good control.

For this algorithm it will be necessary to send additional information to the robots every frame. Therefore we will have to modify the packet structure of the wireless system. In order to get good sensor calibration we want the position information sent to the robots to be as precise as possible. Hence the amount of additional data is a function of the quality of our vision system. In 2002 our vision system could detect the orientation of the robots within 2.5°. If we want to normalize this number without adding errors we will have to use 8 bits to send the orientation to the robots:

$$360° / 255 = 1.41°$$

In case our bandwidth is severely limited, using only 7 bits is definitely an option. The rounding errors are small. We could also go to a differential system and only send the difference between the last perceived orientation and the new orientation. This way we could transfer the information using 4 bits.

ROBOT VELOCITY CONTROL SECTION 5

5.1 Overview

The previous chapter explains how information about the actual robot motion is gathered, which sensors were implemented etc. This section talks about how the sensor measurements were used to augment the robot control. The goal was to follow the commanded trajectory as closely as possible.
It should be noted that the implementation of the controls system is not finished at this point since we don't have the final version of the robots yet. The presented strategy has been tested on a 2002 robot equipped with a rate gyro. It achieved a significant improvement in stability and precision regarding the rotational degree of freedom. There will be more testing and tuning of this control loop over the summer.

5.2 Schematics

The robot motion control is a cascaded control system with 2 loops (see figure 5.1).

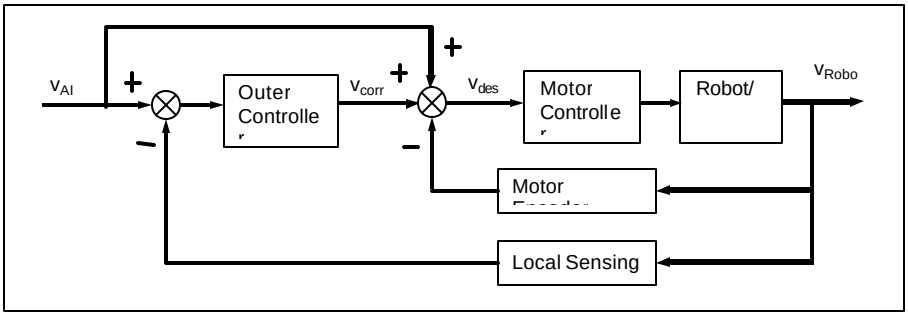


Figure 5.1 Control Loop

The inner loop (containing the motor control and the sensor readouts from the encoders) is basically the same as the one from 2002. During the previous years we sent our velocity commands from AI directly to this loop. The motor control is responsible for spinning the wheels at the commanded speeds and making the robot move.

The outer loop is an innovation from 2003. It takes the commands from AI as inputs and compares them with the measured robot velocities. The outer controller then makes corrections based upon the difference between commanded and measured robot velocities. This means that the robots can adjust their wheel velocities in order to stay on the commanded trajectory even if one or more wheels are slipping.

The outer loop uses a PI controller with a feed-forward path. The advantage of feed-forward is that a change in the desired velocity immediately affects the motor control. In the case that the robot follows the trajectory perfectly there are no adjustments made by the outer controller. A “standard control loop” would have to wait for an error to occur before the controller output is unequal to zero and the wheels would be commanded to spin.

The proportional term of the controller makes immediate adjustments to the desired wheel velocities while the integral term is able to correct for errors happening in the past.
The input to the motor loop is:

$$v_{des} = v_{AI} + v_{corr}$$
$$= v_{AI} + k_p\left(v_{AI} - v_{robot}\right) + k_i\int\left(v_{AI} - v_{robot}\right)dt \qquad \text{... Eq 5.1}$$

with v_{AI} being the vector of original commands from AI,

v_{robot} the actual robot velocities and;

v_{corr} the corrections made by the controller.

In the motion microcontroller this controls equation is implemented in its discrete time form but the idea behind is the same.

The following example should give an idea of how the controller is working:
The robot is commanded to accelerate quickly in one direction and turn at the same time. Assume that the acceleration is so fast that the wheels are operating very closely to their friction limit. Commanding the robot to turn has the effect that the necessary force to execute this maneuver exceeds the maximum friction force between wheels and ground. The wheels slip and the robot motion is out of control. Assume that the translation is roughly as desired while the rotation is off by a large amount. The controller receives the deviation between desired and actual orientation. It will bring the robot back to the desired trajectory by correcting the desired wheel speeds, i.e. speeding up some of the wheels and slowing down the other wheels.

5.3 Additional Features

In some situations we might not want this kind of velocity correction. Since there is always a certain amount of sensor noise, we implemented a dead zone for the corrected velocity v_{corr}. If v_{corr} is smaller than a certain threshold, we neglect it. This prevents jittery movement at low velocities.

Tuning of this cascaded control system works from the inside out. First, the gains (proportional, integral, antiwindup, etc.) of the motor loop are tuned to get a fast response without becoming unstable. Then the gains of the outer loop are adjusted. The response of the inner loop should be 10 times faster than that of the outer loop in order to prevent undesired instability effects.

Lookout

Over the summer we should have plenty of time to look into several other controls algorithms and tune them. One possible idea is to use some kind of "asymmetric" control, meaning that the wheels are sped up to a larger degree than they are slowed down. This way we could maintain a higher total robot velocity.

5.4 Motor Control

At the core of the robot motion control is the tight motor loop. This loop is very similar to that of the 2002 robots but there are some modifications. This chapter deals mainly with the changes made in 2003, for additional information see the 2002 EE documentation (chapter 5.3) or a feedback control textbook.
First of all, we're only using one microcontroller for motion control this year. The new micros have much higher processing power, so we could get rid of the second chip. On top of that, all of the "motion related" processing is done on the motion micro.

The function of the main micro is just to receive the wireless data and pass it on to the motion micro. In the previous year the wireless commands were parsed in the main micro, transformed into physical velocities and renormalized before they were sent to the motion micro. This procedure created more computational overhead and introduced additional errors due to the discretization when normalizing the velocities to a 7 bit number.

The feedback loop runs at 300 Hz. This is much faster than the mechanical response of the robot (around 10 Hz) while we're still receiving enough encoder counts (14 counts/sample at 2.5 m/s) at low velocities to ensure good control.

Figure 5.2 shows the drive geometry and naming conventions.

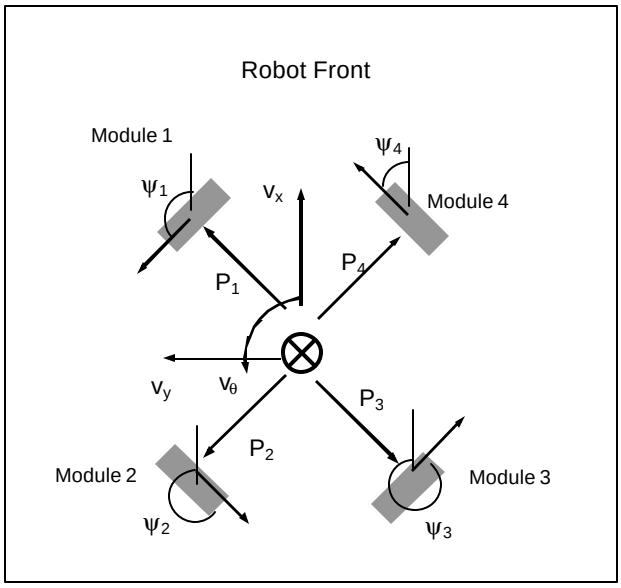


Figure 5.2 Drive geometry

Where v_x, v_y, $v_?$ are denoting robot velocities (in the robot frame of reference), P_i are the vectors from the geometrical center of the robot to the contact points between wheels and ground, $?_i$ are the directions in which the wheels are driven.

The following sections give a brief explanation of how the motor control loop works.

Receive new commands

The loop receives new velocity commands. These robot velocities v_x, v_y, $v_?$ are given in m/s and rad/s. The commands can stem from the outer (sensor) loop or directly from AI. The motor loop is indifferent to the source of the data.

Transform to wheel velocities

Using a transformation matrix these velocities are translated into wheel velocities. The matrix is a function of the drive geometry:

$$\begin{bmatrix} \tilde{w}_1 \\ \tilde{w}_2 \\ \tilde{w}_3 \\ \tilde{w}_4 \end{bmatrix} = \begin{bmatrix} \cos(y_1) & \sin(y_1) & l_1 \\ \cos(y_2) & \sin(y_2) & l_2 \\ \cos(y_3) & \sin(y_3) & l_3 \\ \cos(y_4) & \sin(y_4) & l_4 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_q \end{bmatrix}$$

where the variables l_i are the magnitudes of the P_i vectors. The wheel velocities $\tilde{w}_i$ are still in m/s. The readings from the encoders are in counts/sample, so we have to compute the wheel velocities in counts/sample as well in order to compare the two. This transformation is a function of the following properties:

- wheel diameter Δ_{wheel}: 0.051 [m]
- gear ratio R_{gear}: 13.5 []
- encoder counts per rev C_{enc}: 512 [counts/rev]
- multiplication factor (by FPGA) C_{mult}: 4
- frequency of control loop f_{cl}: 300 [sample/s]

$$\begin{aligned} w_i &= \frac{1}{f_{cl}} C_{enc} C_{mult} R_{gear} \frac{1}{D_{wheel} p} \tilde{w}_i \\ &= 575.2 \tilde{w}_i \end{aligned} \qquad \text{... Eq 5.2}$$

With w_i being the wheel velocity in counts/sample and $\tilde{w}_i$ the wheel velocity in m/s. The factor 575.2 is simply multiplied to the result of the transformation matrix above to yield the required wheel velocity.

Motor controller

The commanded wheel velocities are now compared with the observed wheel velocities (as seen by the encoders). The difference between the two is calculated and fed into the motion controller. The controller takes this error signal and computes a PWM signal which determines the voltage applied to the drive motors (and therefore the actuator effort).

The motor controller this year is very similar to the one used last year. It is basically a PI-controller with a "full-duty mode". The P term of the controller increases the speed of response of the wheels while the I term reduces tracking errors.

The “full-duty mode” is activated when the error exceeds a certain threshold. In that case the PWM is set to maximum, regardless of the actual values of the P and I terms. As soon as the error falls below another threshold we go back to PI control. The purpose of this is to reach the desired velocity as quickly as possible (with maximum control effort) without having to wait for the P and I terms to grow first.

Motor

The applied voltage makes the motors spin and so the robot moves. The revolutions of the drive motors are monitored by the encoders mentioned above. The control loop is closed.
The motors we use this year are specified for 6 V maximum applied voltage. We usually overdrive these motors, i.e. apply much more than the maximum “allowed” voltage to them. This gives a large increase in performance, since torque (and therefore acceleration) is proportional to the applied voltage. Our battery packs this year peak out at about 16 V. Even including losses over the electrical circuit we have to test whether the motors can withstand these conditions without damage. In case that we destroy the drive motors it is possible to limit the maximum PWM output and therefore the applied voltage. Care should also be exercised when tuning the feedback loop, since this determines the actuator effort and ultimately the life span of the motors.

INFRA-RED BALL DETECTOR SECTION 6

6.1 Overview

An IR beam in the 2003 robots detects the ball. When the beam is broken by the presence of a ball, a signal is triggered which informs the robot that the ball is in its possession. The vision system double-checks this status based on the images received by camera. The IR system is thus the 'eyes' of the robot with respect to possession of the ball.

The IR system contains a transmitter and a receiver circuit. The transmitter circuit generates a 5 KHz pulsed signal through a diode transmitter. This signal is then received by the receiver diode on the receiver circuit. Both of the diodes are placed and aligned in a straight line on the robot. The signal is processed and the output is compared with 0.5 Volts. If the result is 1, i.e., the signal is in the neighborhood of 0.5 Volts, then it indicates that the beam has been broken and the robot has the ball. The LED lights up and a "high" signal of 1 is sent to the micro controller. If, on the other hand, the output is 0, indicating that the beam has not been broken, the signal is 5 Volts and a "low" signal of 0 is sent to the micro controller.

6.2 Introduction

The conceptual issue of ball-detection deals with detecting the ball (once the robot is in possession of it) in a reliable and robust manner with an excellent response time. Various interferences like ambient light in the room and collisions among robots need to be taken into account. The integrity of the system i.e., the assurance of the robot that it does indeed have possession of the ball, is of prime concern. The response time involved in detecting the ball and subsequently informing the main micro controller is also of great interest as the system may be programmed to use that information to trigger other activities locally or through AI.

In 2002, the IR system was modified to filter out ambient infrared light, which prevented the system from detecting a broken beam. The 2002 design incorporated sending a pulsed signal to the receiver. Any DC signals were filtered out from the received signal through a high-pass filter before further processing. This filtration removes the ambient infrared light, leaving only the pulsed signal to detect possession of the ball.

This year in 2003, we built on the strengths and advantages of the 2002 design by improving reliability, robustness, and response time of the circuit. We achieved this primarily by using a higher frequency signal. We used a different operational amplifier (AD823) that performed better at high frequencies, compared to the amplifier (OPA 1013) used in 2002. This allowed us to increase the frequency of the transmitted pulsed signal from 240 Hz to 5KHz. We also reduced the number of operational amplifiers in the circuit from three to two by implementing a single comparator instead of a dual comparator. Furthermore, the higher frequency allowed us to reduce the size of the capacitors in the circuit. The smaller circuit and filtering elements increased the response time of IR circuit and allowed us to save more board space.

Other alternatives were considered such as capacitive proximity sensors but this was rejected because testing in 2001 had revealed that the metallic parts of the robot interfered with the performance of these sensors once they were placed on the robot. This technology has however been improved and the newest sensors released by Qprox in late 2002, may be tested in the future.

The IR system is divided into two major parts: IR transmitter and IR receiver. The technical details are described as follows:

6.3 IR Transmitter

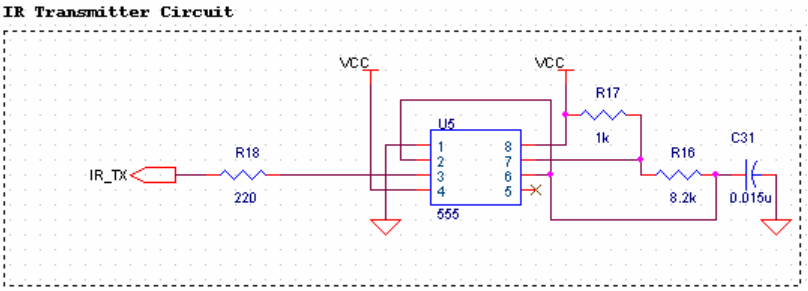


Figure 6.1 IR Transmitter Circuit

The transmitter is a 555 timer chip running an infrared LED at 5 KHz. The 555 is a standard timer/oscillator whose oscillating frequency can be set by a resistor-capacitor

pair. As we need a regular square wave at 5KHz, for each wavelength, 50% of the wave should be on and 50% should be off i.e., the duty cycle should be set to 50%. If the duty cycle was set to 100%, we would get a positive DC signal and no wave at all. Hence, we selected the resistor values such that the duty cycle of the best and worst case scenarios, given the tolerance range of the components in the circuit, hovered around 51.6% – 51.8%.

Since we increased the frequency to 5 KHz, we ensured that the receiver's received power and the amplification response did not drop off at high frequencies by using new AD823 operational amplifiers (op-amps) with higher frequency operational specifications. To adjust the strength of the transmitted signal, there is a current limiting resistor on the output portion of the circuit.

We used the same infrared LED as was used in 2002 because it operated well at the 5KHz frequency. It is brighter and shines in a wider angular field compared to the LED used in 2001 and hence it is less likely to malfunction due to misalignment.
The total current consumption for the IR transmitter is around 100mA, which is a non -trivial current requirement for the digital board to support, especially since the IR transmitter is continuously on.

6.4 IR Receiver

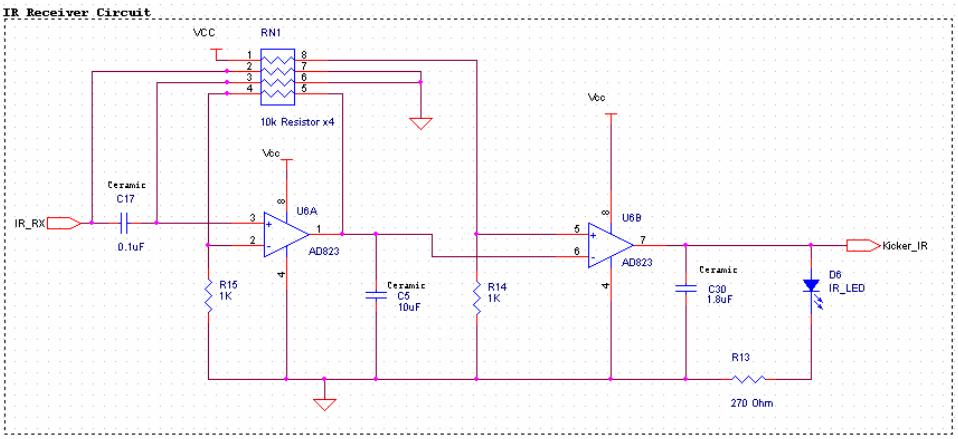


Figure 6.2 IR Receiver Circuit

The improved design this year receives a pulsed IR signal at a 5 KHz frequency. When the receiver receives the beam, the DC bias is removed (this subtracts out ambient light levels) and the pulsed AC component is further processed. The rest of the receiver circuit does simple signal processing to detect the strength of the AC component and generates a digital high if the beam is broken. Firstly, we amplify the signal by a factor of eleven, which improves the resolution of the signal. Secondly, we use a passive filter to average out the amplified signal so that it is approximately flat and can be compared to a level of 0.5V in a comparator. If the signal

al
al

WIRELESS TRANSMISSION SECTION 7

7.1 Overview

The 2003 Cornell Robocup team has taken a systems engineering approach to designing and building highly advanced autonomous robots. Every robot relies on the commands of an off-board Artificial Intelligence Computer that decides which robot should move where. The link between this computer and the robots is one of the most critical components of the system. Without it, the robots would not know what to do.

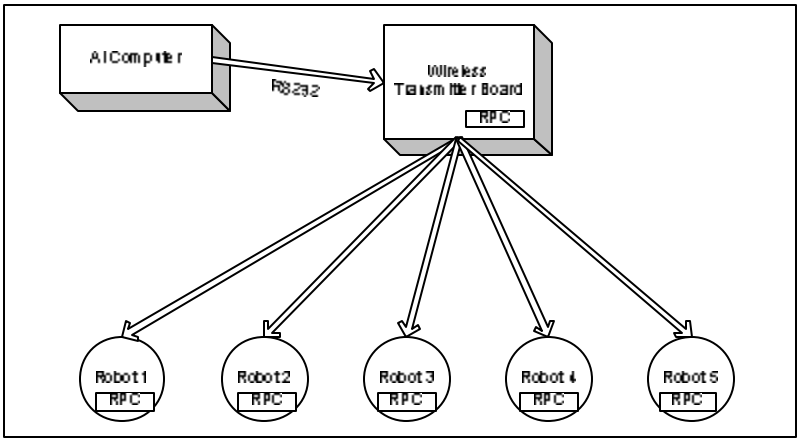
With our new system, the robots are sent commands at extremely fast rates, on the order of 100 instructions per second (per robot). Data sent to these robots works much like a pipe carrying water. Say for instance a very big pipe can handle 100 gallons per minute of water and another much smaller pipe can handle 1 gallon per minute of water. If we are looking to pump as much water as possible in a set amount of time, the obvious choice would be the bigger pipe.. When choosing the wireless modules for the 2003 system, we noted that a larger "pipe" would give us the ability to pump more data in less time. The faster the data can be pumped, the less delay there is in the system, and the more precise the AI computers commands can be.

In 2002, the AI used a somewhat elaborate scheme to "predict" where the robots would be by the time the actual data got to the robots. However, no matter how perfect an algorithm, it is still impossible to predict the future perfectly, and more often than not, this prediction algorithm would end up causing many problems with involved maneuvers such as passing and shooting. The obvious solution to this problem was to reduce the delay between the AI and the robots such that the AI does not have to do so much prediction. With less prediction, the robots get more "real-time" data and consequently perform much more accurately. This was one of the wireless team's primary goals in 2003.

Aside from speed and delay improvements, it was also decided that a bi-directional system would be very advantageous for game play. The 2002 system works in one direction; i.e. ONLY the AI can send commands to the robots. However, the 2003 system will allow a return channel so that the robots can talk back to the AI or Vision computers. We are still conducting tests with certain modules and algorithms, discussed later in this document.

7.2 A Brief Overview of the 2002 System

In 2002, the Cornell Robocup team implemented a very simple wireless solution.



After many tests with different algorithms on the Wireless Transmitter Board, it became clear to us that if we were going to reduce the delay in the system we would need to find an alternate solution to the RPC modules.

7.3 Selection of New Wireless Modules for 2003

After much brainstorming and research, we figured out the most important attributes of our new Wireless Module would be:

- High Speed
    - Speed is like the diameter of our water pipe. The wider the pipe, the higher the bandwidth. The ideal bandwidth we were looking for was 115200 bps or greater. The higher the speed, the less delay time between AI and the robots and the more flexibility in wireless packets.
- Low Setup Time
    - The wireless receiver module needs time to "warm up" before it can produce useable data. The lower this time, the faster the system can recover if a data line were to go bad.
- Simple Serial Interface
    - Advantageous over RPC parallel interface. Serial uses much fewer pins and offers a much easier implementation with microcontroller code.
    - A 5+ V serial line would be ideal, as that is the interface on the Microcontroller side.
- Multiple Frequencies
    - As per Robocup rules, each team must have at least two operating frequencies to compete. If another team uses a frequency we are using, we must be able to change our frequency to avoid interference.

Technologies we considered all are included in the RF band. The modules range from 418Mhz all the way up to 2.4Ghz. The candidates for our modules were the Radiometrix RPC, Radiometrix TX2/RX2 modules, Radiometrix TX3/RX3 modules, and Bluetooth. The distinct features of every module were analyzed and we have summarized those features in the chart below:

	RPC	**TX2/RX2**	**TX3/RX3**	**BLUETOOTH**
Max Speed	64 .Kbps	115.2 Kbps	60. Kbps	1.2 Mbps
Max Speed (a)	~40.000 Kbps	56.000 Kbps	38.400 Kbps	680 Kbps
Latency	17 ms	<10ms	<12ms	< 2 ms
Frequencies	433/418 MHz	433/418 MHz	869/914 MHz	2.4Ghz FHSS
Setup Time	~5ms	~1ms	~1ms	None
Interface	Parallel	Serial	Serial	Serial (3.3V)

Figure 7.2 2003 New Module Quick Comparison Chart

(a) Indicates actual in-lab test results

As noted, every wireless module had its unique advantages. We used the samples from 2002 that were left in the Robocup lab to generate a test bed of wireless platforms and analyze their performance. We spent many hours communicating with team members from 2002 who had worked previously with the Radiometrix modules, but there were many things that were left unexplained about the code implemented in 2002.

We decided to start from scratch and redesign the entire circuit and algorithm for using each module. After many long days and nights, we came up with working prototypes for the RX2/TX2 pair and the RX3/TX3 pair. We tested for link quality (how accurate the data was) and link speed (how fast it could be pumped) and found excellent results. The latencies displayed in Figure X.2 are the actual results of those tests.

7.4 Technical Information on the TX2/RX2 modules

The TX2/RX2 modules were first introduced to us by last year's EE team. These modules come in pairs with a separate transmitter and receiver. They operate in the 433 MHz Band and just recently, a 418 MHz version has been introduced. These modules offer us many distinct advantages including:

- Higher Bandwidth (160 kbps)
- Small Physical Footprint
- Low Latency (less than 4ms)
- Easy to operate with Serial RS232 interface
- 5V logic device
- Useable range 300m

The actual implementation of these modules is not as simple as the RPC's used in 2002. We designed a very simple circuit to implement these modules, all of which is rather straightforward and extracted from the Radiometrix datasheets.

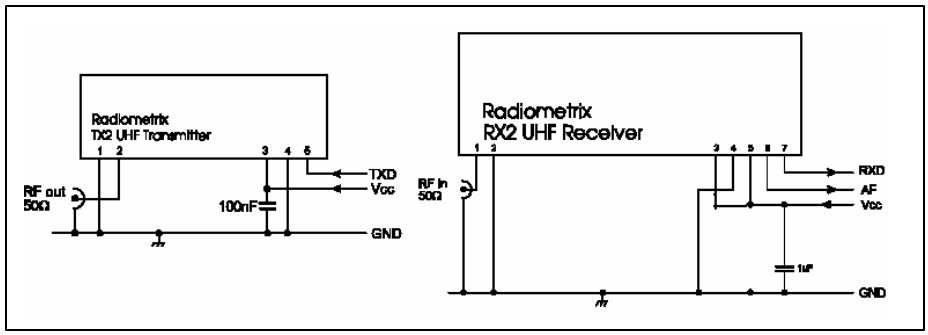


Figure 7.3 TX2 RX2 Circuit

As shown in the circuit diagrams, the transmitter (right) can be driven almost directly off any 5V logic source, indicated as TXD. The only external component is a small 100nFcapacitor used to filter the power line. The RF out connects to a 50 ohm whip antenna. The receiver (left) also has a small power filtering capacitor, and receives data on RXD. AF (pin 6) and CD (pin 3) are not used by our system.

7.5 Technical Information on the TX3/RX3 modules

The TX3/RX3 modules were first introduced to us by last year's EE team. These modules are almost identical to the TX2/RX2 modules, but operate at 869 MHz and 914 MHz. These modules will be used a backup to the TX2/RX2 modules, as they do not offer quite all the advantages of the other pair.

These modules feature:

- Medium Bandwidth (64 kbps)
- Small Physical Footprint
- Low Latency (less than 4ms)
- Easy to operate with Serial RS232 interface
- 5V logic device
- Useable range 120m
- Almost identical footprint to TX2 RX2 modules

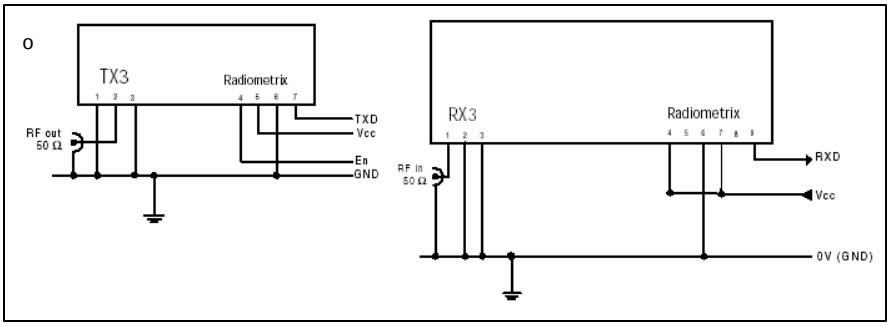


Figure 7.4: TX3 RX3 Circuit Diagrams

As shown in the circuit diagrams, the transmitter (right) can be driven directly off any 5V logic source, indicated as TXD. The RF Out connects to a 50 ohm small whip antenna. No other external components are required. The receiver (left) dos not require a power filtering capacitor, but one was added on the final design to keep the implementation of RX2 and RX3 the same. The RF In connects to a 50 ohm small whip antenna, and the module outputs received data on RXD. AF (pin 6), RSSI (pin5) and EN (pin 3) are not used by our system.

7.6 Technical Information on the Bluetooth modules

Bluetooth is unlike any of the other wireless technologies that we explored. Instead of an "anything goes" radio link where we are free to do as we please, there is a lengthy set of specifications that dictate how the wireless module should behave. That is because all Bluetooth modules are supposed to be able to talk to other modules even if one module is connected to a PC while the other might be a VCR. Bluetooth allows for the on-the-fly creation of groups of modules called "piconets." To handle this complex protocol, the Bluetooth modules we purchased run their own basic operating system and have an onboard microcontroller and flash memory. Upon initialization of the module, various settings must be configured in order to make the module run in the fashion we desire. Once everything is setup, both in flash memory plus working RAM, transmitting data is relatively straight forward.

The structure of the Bluetooth protocol is such that all communications occur at precise time intervals. When all the modules in a given area are participating in a piconet, there is never any issue of two modules trying to transmit at the same time. In the most common organization, one module is the master and the remaining modules are the slaves. In our case, the module connected to the PC is the master,

and the slaves are on the robots. In order to prevent simultaneous transmissions, the protocol only allows a slave to transmit back to the master immediately after receiving a packet destined solely for the slave. However, we want to broadcast data to all the slaves at ones which would make it impossible to have any of them return data. The solution is to broadcast data and then send empty packets to each slave so they can send back data.

Type	Header (bytes)	Payload (bytes)	FEC	CRC	Max. Rate (kb/s)	Forward	Reverse
DM1	1	0-17	2/3	yes	108.8	108.8	108.8
DH1	1	0-27	no	yes	172.8	172.8	172.8
DM3	2	0-121	2/3	yes	258.1	387.2	54.4
DH3	2	0-183	no	yes	390.4	585.6	86.4
DM5	2	0-224	2/3	yes	286.7	477.8	36.3
DH5	2	0-339	no	yes	433.9	723.2	57.6
AUX1	1	0-29	no	no	185.6	185.6	185.6

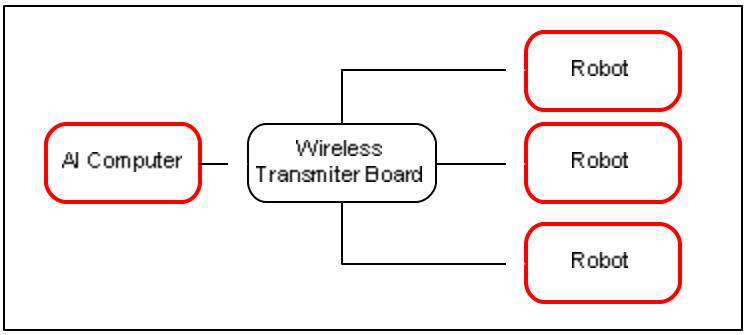
Figure 7.5 Bluetooth packet types

Above is a table showing the various packet types that are supported by the Bluetooth specification. Given that broadcast (as implemented by the manufacturer of our modules) only supports DM1 packets, that is the packet format we use for all of our transmissions. Due to the overhead inherent in any packet, this severely limits the maximum data throughput of our system. However, this is unimportant because we are interested in latency and robustness above anything else. The amount of data that we are sending is only a fraction of even the reduced bandwidth of DM1 packets.

One of the reasons we chose this particular Bluetooth module was due to its EMI shielding and its high transmit power. Our module is capable of outputting 100mW of RF power, which a considerable amount is given that most transmitters transmit at roughly 10mW.

7.7 Implementation of the Modules

When migrating from an older, more basic system to a newer and more complex system, many things can go wrong. And the implementation of the 2003 Wireless Modules was no exception. We decided to abandon the RPC units and move to a system consisting strictly of TX2/RX2 and TX3/RX3 modules. A very rudimentary implementation of the TX2/RX2 and TX3/RX3 modules was left for us by the 2002 EE team, but their code was not very robust and would work intermittently or under “special” conditions only.



The 2003 wireless modules are designed to have a constant stream of data. If this stream is interrupted for a long enough amount of time, the modules will need to set up (resynchronize) all over again. All the data sent before the resynchronization is complete will be lost. This process of synchronization is known as "phase lock". When the data stream is interrupted for long enough, phase lock is lost. After conducting extensive tests, we determined our maximum "dead time" before a stream is considered interrupted and phase lock lost at about 2ms. See the figure below for a more graphical depiction of the problem.

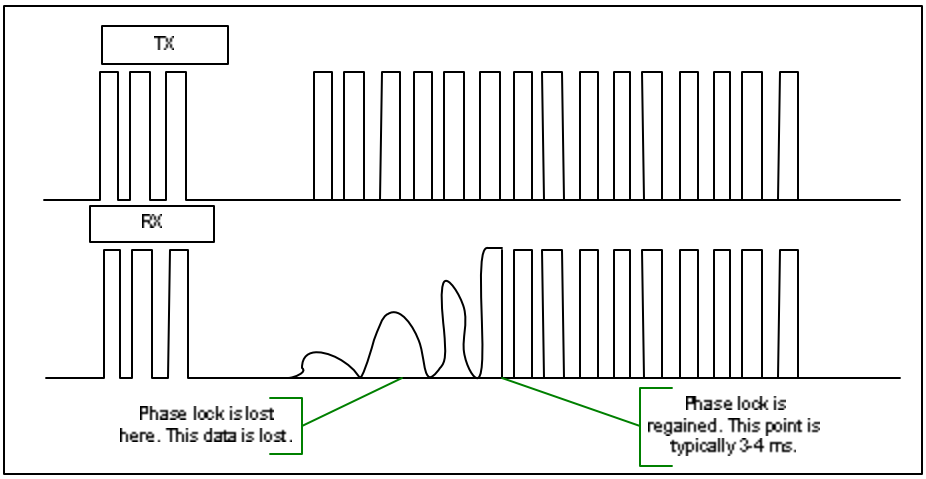


Figure 7.7 Phase Lock Issue

As shown, after a wireless packet was sent, the data in the beginning of the next packet was unusable (the squiggly lines before the data on the RX scope)

We tried using multiple microcontrollers such that one would transmit data and the other would transmit idle packets while no data was being sent, and the process worked very well. We achieved very high speeds on all the modules, and the system appeared to be very robust. We then decided that the function of the transmitter board could be entirely taken care of by the AI computer. There were concerns this may affect the AI performance and that any hiccup in the AI computer could result in a lost phase lock, but tests showed these side effects were negligible.

The new CS based transmitter can process the Manchester encoding extremely fast and eliminates the RS232 link between the old Wireless Transmitter Board and AI computer. The process of sending a packet is outlined in the diagram below:

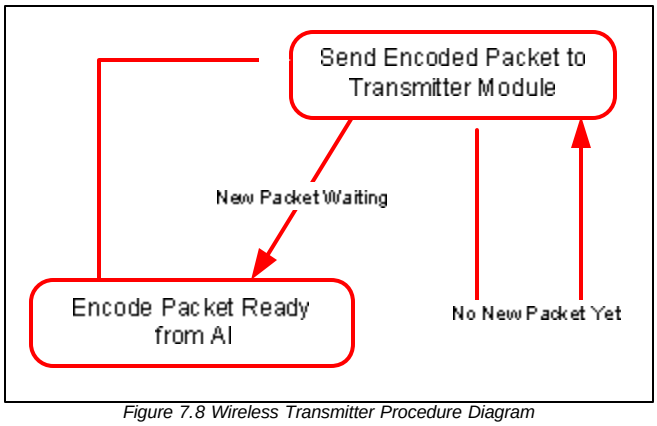


Figure 7.8 Wireless Transmitter Procedure Diagram

7.9 The Receiver

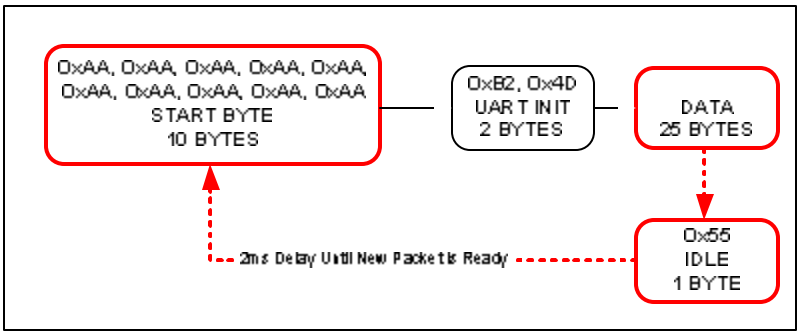
Every robot is equipped with a wireless receiver circuit. It is a very straightforward implementation of the RX3 and RX2 modules. The modules are directly tied to the main microcontroller, using a tri state buffer to prevent the modules from interfering with microcontroller programming. The serial port that is used to program the microcontroller is shared, so during programming mode nothing (wireless module) but the programming dongle should transmit.

The receiver code pretty much mirrors the transmitter code. It is triggered when the micro detects data on the serial input line, and begins to decode each packet using a Manchester lookup table. This information is then forwarded to the appropriate module on the robot.

This year the connection between the wireless module and the antenna is no longer a screw-on connection to the PCB. Now the antennas can be mounted securely on the frame, at any convenient location, and there is a shielded 50 Ohm coaxial cable that connects to the module. This was done to allow for proper placement of the antenna (FULLY outside of the robot's shell) and to make the design less prone to accidents such as knocking the antenna and damaging the PCB. Also, by grounding the chassis and mounting the antenna on the chassis, we are able to create a large ground cage which helps prevent interference issues.

7.10 The Packet Structure

When data is transferred between the transmitter and receiver modules, the data itself must be packaged into a "packet". Raw data is produced from the AI computer which includes trajectories, dribbler speeds, gyro info, etc. This data is then processed by the Wireless Transmitter Module so that it can be sent over the air.



KICKER SECTION 8

8.1 Overview

The kicking subsystem imposes a large acceleration force on the ball that simulates "kicking". It is an extremely vital component of the electrical subsystem as without the ability to kick, we would be entirely unable to score goals! The kicking subsystem is very simple. It receives a signal from the micro-controller that determines how "hard" the kick should be. The kicker is really an electromechanical system, that falls under the purview of both the Electrical Engineering (EE) team and the Mechanical Engineering (MECHE) team. From the EE standpoint, we receive a signal from the micro, and then send a pulse through an actuator, that causes the actuator to be acted upon with by a large force, the mechanical frame around the kicker harnesses this force and delivers it to the ball, causing it to move with a high velocity in the forward direction.

8.2 Introduction

8.2.1 Design Problem

The challenge is to design a system that would transfer a large force to the ball in the most efficient manner and cause it to move with the highest possible velocity.

In terms of technical requirements, the kicking system of 2002 was rated at 1.3m/s top velocity. This year our requirement is a minimum velocity of 3m/s.

8.2.2 Conceptual Overview

There are many different ways to tackle this problem. The ideas that the MECHEs toyed with include

i, Compressed gas (CO_2) based kicking
ii, Gunpowder based kicking

These ideas had they been implemented, would have changed the electrical part of this system dramatically. They were however cancelled (see the MECHE documentation for more detail). The process of determining what technology to use in the kicking system only reinforced the concept that the kicker is an electromechanical system, and both MECHEs and EEs should participate in the design phase. It was finally decided that for safety reasons we should continue to use the solenoid based system, but focus our efforts on improving on it.

The excerpt below explains how solenoids work. We recommend that whoever will work on the kicking circuit next year visit the link and read the entire page.

> *"A solenoid is an electro-mechanical component that converts electrical energy into mechanical power. Electrical current is supplied to a tight coil and the resulting magnetic field is increased by surrounding the coil with a highly permeable iron frame. The magnetic field then acts upon a plunger, drawing it from its unpowered, extended position to a seated position against a backstop or pole piece. The linear force on the plunger from the magnetic field is extremely nonlinear with position, i.e. the force is relatively high immediately adjacent to the seated position and falls off rapidly with increased distance from the seated position.*
>
> *The electromotive force is supplied by the current applied to the coil and is limited by the heat dissipation capacity of the coil. Duty cycle, or percentage of time that the solenoid is powered, is therefore a crucial factor in solenoid selection; the less time a solenoid needs to be powered, the more time it has to cool and thus can be used with higher current, providing more force. One model solenoid can have widely varying force ratings associated with different duty cycles. This search form covers solenoids rated for continuous duty, or 100% duty cycle. In general, this duty cycle will have the lowest force ratings.*
>
> *Solenoids are widely used as actuators for industrial systems and in consumer markets such as vending machines, laundry equipment, and locking and latching systems."*
>
> *http://mechanical-components.globalspec.com/ProductGlossary/Mechanical_Components/Solenoids*

8.3 Solution

8.3.1 Analysis of 2002 system

We decided to start by investigating the 2002 design to verify its efficiency. The 2002 Kicker design is extremely simple. It consists of a solenoid which is activated by a simple MOSFET switch.

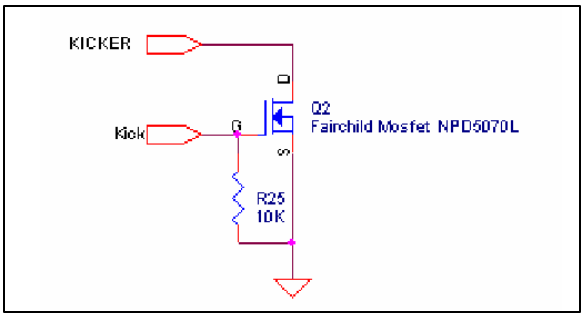


Figure 8.1 2002 Kicker Circuit Design

8.3.2 Circuit Operation

When the MOSFET (which acts as a simple switch), receives a control signal from the microcontroller, it allows current to flow through the circuit, and this causes the solenoid to be activated.

One of our first tasks was to verify that the kicker was working the way it was designed to. The first thing we did was to measure the voltages across the solenoid. These values were a lot smaller than the measured values across the batteries.

BATTERY VOLTAGE (VOLTS)	KICKER VOLTAGE (VOLTS)
29.3	12.0

Table 8.1 Measured Kicker Voltages (Voltmeter RMS)

This result was considered inappropriate by everyone who was familiar with the former system, as it should have been equal to the battery voltages, so it we tried to discover what caused the discrepancy.

We measured the current flowing in the circuit with an ammeter, and discovered that the current values hovered somewhere between (4 - 6 amps) instead of being at the maximum value of (8 amps), even with the voltage at 29.3Volts.

The manufacturer's rating was 8amps at 12Vdc, we were applying 29.3 volts. We should have observed a much higher current as we were applying a much higher voltage, instead however, we only observed between 4 and 6 amps.

Further analysis showed that since the circuit's open window was effectively 40milliseconds, there was insufficient time for the voltage to rise to its nominal value. An oscillator could detect the instantaneous value of (29.3volts), but the voltmeter registered the *rms* values which tended to be much lower.

Once we understood this, we proceeded to increase the turn on time to determine if that would have any impact on the strength of the kick. Surprisingly enough, it did not. When the switch was left on for >= 100milliseconds, the measured voltage was 29.3Volts and the current through the solenoid was of the right magnitude (8 amps), nonetheless, there was no improvement in the performance of the kick.

This made us aware of a number of things

1. **The kicker performance is constrained by a time factor, which is a function of how long it takes for the plunger to attain its maximum deflection.**

2. **As a result of (1) above, the only way to improve the power in the kick is to increase the input power (Voltage and Current) into the solenoid *DURING THE TIME WINDOW!!!***

This meant that we needed to increase the *instantaneous* power to the solenoid in the very first few milliseconds that the switch closing.

As a result of these initial experiments, we decided to try the kicker at various higher voltages. As the current "super kick" voltage of 29.3Volts was giving us a kick performance measure of 1.2m/s, we decided to measure input voltage versus kick velocity as a basis for comparison.

8.3.3 Experiments

We started at 29.3Volts, and slowly increased the voltage until we could no longer do so because the connectors were burning up. "This voltage coupled with the long turn on time (we were using a manual switch for our tests) led to a high current flow." The 80V input voltage gave us a kick of 3m/s. This is more than double the previous performance standard.

The conclusion of these tests is that ***the kicker voltage should be increased to as high a value as we can practically achieve, with a complimentary decrease in the turn on window (i.e. <= 40milliseconds)***

This places the 2002 design in an entirely different light.

8.3.4 Critical analysis of 2002 kicking system

In 2002, they tried to increase the kick strength by increasing the input voltage. This led to adding more batteries. What was really required though was to increase the INPUT POWER during the critical 10 – 40 millisecond window before the solenoid becomes fully extended. It worked, but the tradeoff was an increase in weight since more batteries had to be added to increase the voltage.

Our conclusion was that in 2002, we were not *POWER* limited, (as the batteries are capable of high power output), we were just limited by the amount of power we could pump into the solenoid during our critical 10 – 40 millisecond window. This can only be increased by increasing the instantaneous voltage, hence the need for "super kicker" batteries.

Like the team before us, we realized that we could not make the impedance of the solenoid smaller than it already is, however we *can* raise the instantaneous input voltage during the critical early milliseconds just as the solenoid is switched on.

Our requirements were therefore,

- Very high input voltages for a very short period of time.
- Minimize the number of batteries required to attain those voltages as more batteries implies more weight.

The requirements led us to the circuit shown below.

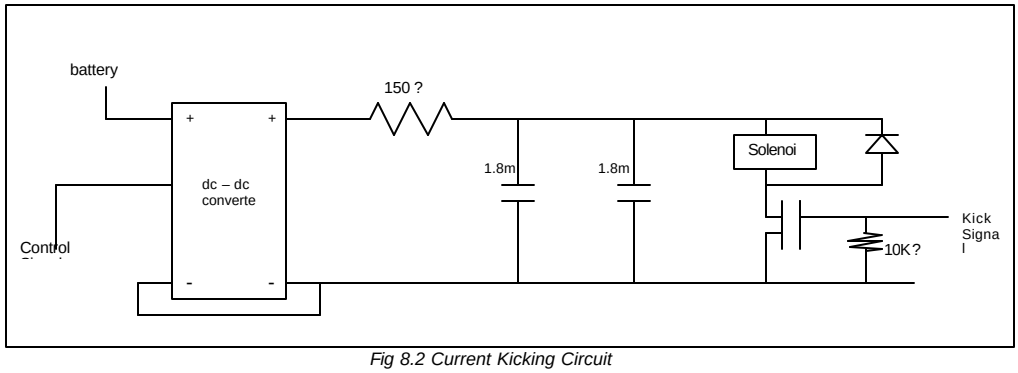


Fig 8.2 Current Kicking Circuit

The DC – DC converter is responsible for raising the input voltage from about 14Volts to close to 120Volts. This voltage charges the 1.8mF capacitors. The charging rate is determined by the value of the series resistance which is currently150? .

When the MOSFET receives a signal from the micro-controller, it closes the circuit, which then places the charged capacitors across the solenoid, and applies the entire 120Volts to the actuator. This accomplishes our design objectives by raising the input voltage, without adding more batteries. Though it is true that the DC – DC converter is heavy, this weight has to be compared with the equivalent weight of the batteries we would need to attain an input of 120Volts.

With this circuit, we were able to realize speeds of close to 4m/s. This is better than the initial specifications.

The specifications of the DC – DC converter can be found in the appendix.

8.4 Conclusion

The circuit outperforms our goals. The DC – DC converter seems to drift slightly i n it's output voltage, but this is not a major issue, and so far has not negatively impacted performance. We would have liked to have more DC converters available for comparison, but they are rare, and this was the only one that met our specifications.

The kicking circuit is doing so well, that the MECHE's may have to redesign the mechanical portions to handle the larger force.

We believe that the performance of the kick can be improved even further if some analysis is done on the behavior of the current flowing in the circuit when the switch is closed. This model could then be used to optimize our current design. We recommend that the 2004 team models the dynamic behavior of the circuit so as to fully ascertain the optimal conditions for a strong kick.

BALL POSSESSION FLAG / RETURN PATH SECTION 9

9.1 Overview

The efficiency of the RoboCup system is greatly affected by two major factors. These are the accuracy of the information which is fed to the system and also the speed at which this information is made available.

One of the measurable bottlenecks of the current system is the amount of time it takes for the AI and Vision to verify that our players actually have the ball. This time could be cut drastically by implementing a communications link between the robots and AI that would enable the robots inform AI *immediately they have the ball*.

The Ball possession flag is a parallel information path for the AI system. Sometimes Vision finds it difficult to place the ball, i.e. the ball becomes occluded. The return path for ball possession attempts to combat this by providing an alternate path for sending information on the current location of the ball back to the AI system. This should have the effect of reducing latency. In a sense, this sub-system can be compared to a 'reverse' communication system which sends information from the robots back to the AI system.

The 2002 design tried to implement this link utilizing Ultrasonic transducers. This ultrasound circuit is currently implemented in the 2002 Robots. It was never used during play.

The system requirements have evolved over the past one year. Currently, the system is required to accept inputs from the micro-controller. These inputs will indicate the following

- The IR beam is broken indicating that the ball is present on the dribble
- The Robot ID

These two pieces of information are then sent to the AI computer.

9.2 Introduction

The real problem is the communications infrastructure between the robots and the AI. It is a simple matter to get the required inputs from the micro-controller; the challenge is in collating this material and sending it to the AI system. In 2002, they proposed a simple system based on ultra-sound. The ultrasound system consisted of individual transmitters on each of the robots, and on main receiver. The receiver picked up the signal whenever any of the robots got the ball and then transmitted this to the AI system via a simple circuit.

The major problem with this elegant solution is that it does not allow for *simultaneous multiple transmissions*.

Upon serious contemplation of all the possibilities inherent in the game, one quickly comes to the conclusion that without the ability to handle situations where more than one robot has its IR beam broken, any system developed will be inadequate. Therefore it all boils down to designing a communication system that can handle multiple simultaneous transmissions.

The system should also be capable of handling a minimum number of bits so as to be capable of handling the entire information stream. It should also be capable of operating at a speed higher than the total Vision system latency. At the absolute minimum, it should be capable of operating at the same speed as the Vision system.

Thus we can rewrite the original specifications as follows;

- Must be capable of multiple simultaneous transmissions
- Must operate faster than the total system latency of the Vision / Network subsystems
- Must be capable of handling at least 4 bits of data
- It should not interfere with any of the current sub-systems

With hindsight, this seems almost obvious, but this conclusion is the result of close to an entire semester of concentrated work and frustration.

9.3 Solution

At the beginning of the second semester when we actually started work on this aspect of the project, we were told to get the ultra-sound system working. Upon reflection, this was a wrong approach, as there was no prior problem analysis and definition. This would have been a much better place to start, as it would have provided a much more prudent way of tackling the problem, and would have saved a lot of wasted time. It would be prudent to begin this discussion with an analysis of the 2002 system.

9.4 Analysis of 2002 system

As was stated earlier, the 2002 system utilizes ultrasound as its communication medium. In addition, to address the issue of multiple transmissions, it only allows for one robot to transmit during a single time slot. The problem with this is that it does not take the situation where two robots may have their IR sensors broken at the same time. In addition, since it requires some processing on the vision data to

determine which robot should be speaking, it is subject to the Vision latency, and may not be able to respond to dynamic situations where vision loses the ball, and the possession of the ball is constantly varying.

Another limitation of the 2002 system is it's reliance upon ultrasound. During our in-house tests in the lab, the range of the ultrasound transmitter is very limited.

We discovered that the effective range of the transmitters was limited to 3 – 4 meters. It is also highly directional and depends on Line-Of-Sight for it to work. We suspected this from the start as acoustic signals tend to be extremely difficult to control at very high frequencies, and also exhibit high directionality.

In addition, ultrasound is highly directional, requiring the receiver to be located somewhere above the field. It would be necessary to use multiple receivers as one would be insufficient to cover the area of the entire field. With the use of multiple receivers the problem of synchronizing and integrating these receivers arises. Multiple receivers also increase the gain of the acoustic system, and make it more susceptible to random interference. Cabling and impracticality of placing and positioning multiple receivers make this a non-affair.

We tried to increase the power of the transmitter in a bid to increase sensitivity of the system, but this meant using multiple transmitters (transducers) on the robots. Phase cancellation is a very real problem with a multiple transmitter system.

For the two reasons listed above, we decided to drop ultrasound, and to look for a comprehensive solution that would be sufficiently flexible to handle any possible situation and be impervious to interference.

It did take too long for a consensus to be reached about medium of transmission. It became very obvious that the best way to implement a system of this kind was through wireless transmission.

That being decided we tried out various wireless systems

9.5 "TDMA"

As has been mentioned earlier the whole process was a major learning process, and we made quite a number of mistakes earlier in the process. Perhaps the first was trying to utilize the "Ming" modules. These operate at 310 MHz, and are very simple analogue transmitters. They are amplitude modulated,

and have a top transmission rate of about 9 kbps. They are very rudimentary transmitters. Our original intention was to use multiple receivers and one transmitter in a TDMA.

In trying to set this up we learnt some bitter lessons, about setup time, and that it takes a finite amount of time for a wireless receiver to “lock” onto the transmitted signal. Because this time tends to vary inconsistently, the TDMA setup just could not work. From unit to unit, the set-up times varied.

After countless hours struggling with the transmitters, we finally determined that the modules were unusable in this fashion.

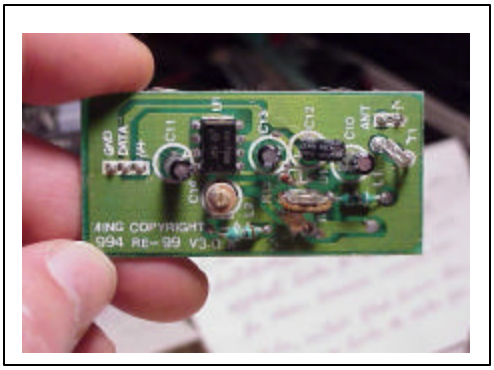


Figure 9.1 Ming Transmitter module

The data sheets are available on the accompanying CD.
After the TDMA fiasco, we were informed that the 2002 team had also realized the same limitations in the ultrasound circuit and had developed a wireless solution that was capable of multi -frequency operation. This module called the SE200 was supposed to be readily available in the lab.

We finally located the module and spent the last few weeks trying to get it working.

9.6 SE200 & Multi-channel Wireless transmission

The SE200 modules are multi-channel transceiver modules. As such each module is capable of being both a transmitter and a receiver. Theses modules are made in Germany, and one of the first problems we had was finding an English translation of the datasheet. After a lot of effort, we were forced to translate the pdf files to html and use Google's translation feature to translate the files.

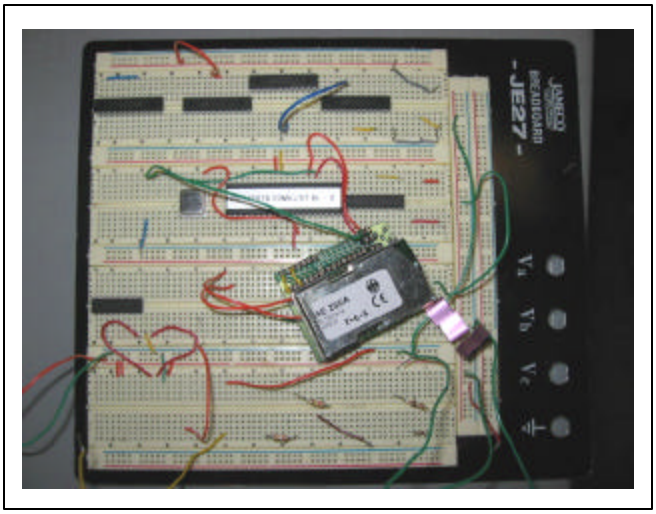


Figure 9.2 SE200 Module in Circuit

We discovered that the SE200 modules require 3V operation, which makes it difficult to operate on the 5V design that all our digital components are based on. Fortunately, there is a daughter board called the SE200 – A1 that allows for 5V operation. This daughter board contains a micro-controller, that handles all the housekeeping functions of the SE200, allowing the user to focus on the actual transmission.

The SE200 can operate in 2 modes. The first mode is the parallel mode, where it accepts 8 inputs in parallel, and then transmits them. In this mode, the micro-controller handles all the encoding and decoding functions. Unfortunately, this mode is way to slow for our purposes.

The other mode is the serial mode. In this mode, the SE200 microcontroller merely replicates the signal it sees on the specified pin. This mode requires the user to handle all the encoding and decoding functions.

9.6.1 Encoding and Decoding

It is our advice that any one working on ANY wireless system spends some time learning about different encoding and decoding schemes. Wireless data tends to be very erratic, and is easily subject to interference, especially at the power bandwidths we work at. To counter this it is necessary to encode the data in such a way as to enable accurate reception of the received signal. Transmitters also perform optimally if the number of negative signals is equal to the number of positive signals. So encoding addresses these two issues. Note however, that an encoded signal usually carries a much larger bandwidth requirement than the raw data.

Unfortunately, we were unable to implement a satisfactory encoding decoding scheme for the SE200s. The major problem seems to be that the bandwidth is limited to 10Kbps. With the encoding algorithm, it is impossible to send all the necessary data in the allotted time.

We investigated specialized encoding / decoding chips in an effort to improve efficiency but none of them proved satisfactory, and in the end we decided to write our own algorithms using a micro-controller to implement them.

Alas, we had insufficient time to implement a scheme, before the deadline for design elapsed, and so it was not possible to include the return path on the 2003.Robots.

9.7 Conclusion

Throughout the semester it became glaringly obvious that the best way to tackle this problem is through a multi-channel wireless system. The SE200 is capable of operating at 16 different frequencies but they are all in the 430 MHz range.

We investigated other transceiver systems that we would recommend the 2004 group evaluate. The most promising of these is the Nordic nRF903. This chip has a very high transmission rate and is capable of operating at over 200 different frequencies.

All the data sheets and specifications for this chip are available on the accompanying CD.

As has been pointed out earlier on in this document, it would also be necessary for the 2004 team to familiarize themselves with encoding decoding schemes, and programming micro-controllers.

BALL HANDLING (Main & Side Dribblers) SECTION 10

10.1 HORIZONTAL DRIBBLING SYSTEM

The horizontal dribbler is an essential component of the 2003 electrical system. The rules of the Robocup competition prohibit robots from grabbing and holding onto the ball. Therefore, a horizontal dribbler that spins backwards is placed at the front of the robot. On contact with the dribbler, the ball starts spinning and the dribbler allows the robot to move with the ball. In this sense, the horizontal dribbler functions as the "feet" of the robot.

In 2003, we designed and implemented code in the main micro controller and added hardware (an encoder to the dribbler motor) to the robot that allowed us to control the speed of the dribbler. In the summer, this code will be extended to vary the speed of the dribbler based on the motion vectors, (V_x, V_y, V_θ) of the robot. Given the velocity vector of the robot, the dribbler should speed up or slow down so that better ball control is achieved. For example, just before the robot turns, the dribbler should slow down so that the ball "moves" with the robot and the robot does not lose the ball. In addition, this code also allows us to stop the dribbler instantaneously so that the robot can kick the ball.

The ability to control the speed of the dribbler will also allow us to spin the ball faster in the event of a "dribbling battle." A dribbling battle occurs when two opposing robots "fight" over control of the ball. In such a case, it will be desirable to operate the horizontal dribbler at its maximum velocity so that we are able to obtain possession of the ball.

10.1.1 Analysis of 2002 Horizontal Dribbler Circuit

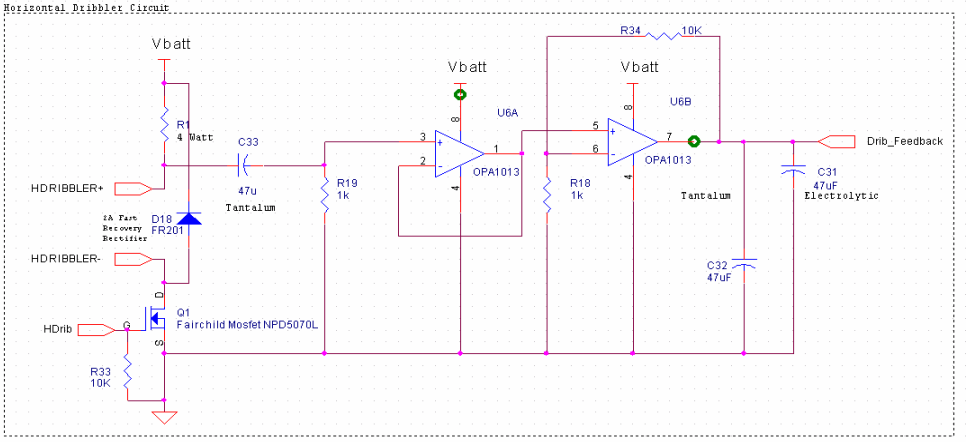


Figure 10.1 2002 Horizontal Dribbler Circuit.

In the 2002 dribbler circuit shown above, the dribbler motor is controlled by a pulse width modulated signal (PWM). The PWM has a duty cycle, which specifies the fraction of time that the voltage signal is high/on. This in turn regulates the voltage across the motor.

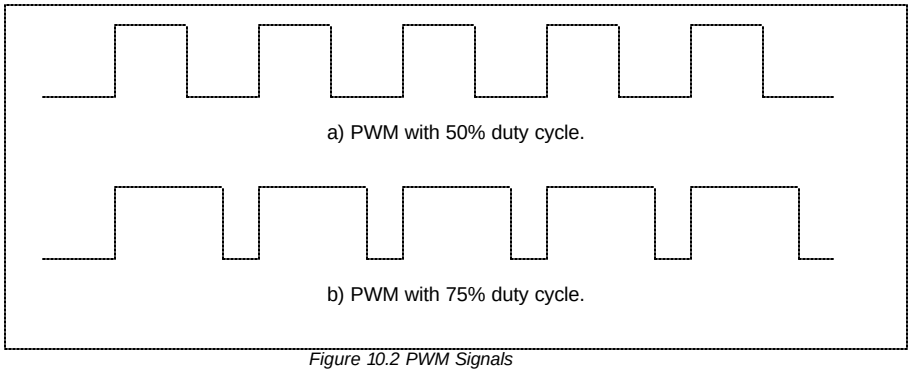


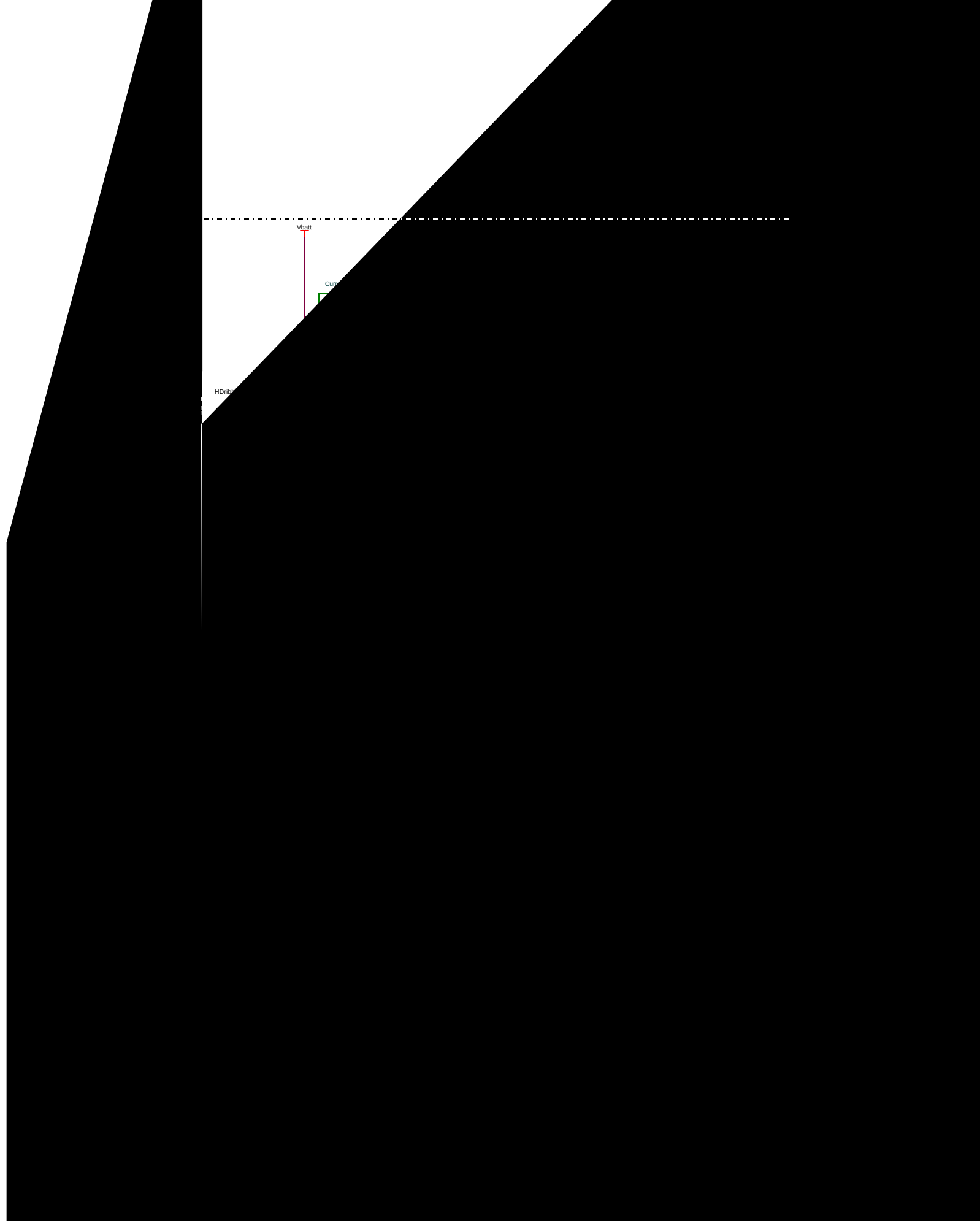
Figure 10.2 PWM Signals

For example, in figure 2a, the PWM has a duty cycle of 50%. Therefore,

$$V_{motor} = 50\% \times V_{batt} = 50\% \times 12V = 6V \qquad \text{...Eq 10.1}$$

When the PWM is on, the MOSFET switch makes contact and current flows through the circuit. The circuit measures the current flowing to the dribbler by measuring the voltage drop across the resistor R1. By Ohm's law, the voltage drop across the resistor is directly proportional to the current flowing through it so measuring the voltage drop is, in essence, measuring the current flowing through the circuit. The measured voltage is then high pass filtered to remove the DC offset and then amplified by eleven before the voltage signal is input to the main micro controller. In short, the circuit measures the torque of the dribbler motor by measuring the current flowing to the motor. The value of the current is then input to the main micro controller, which then sends instructions to slow down or speed up the dribbler, based on the value of the current.

There were a few problems with the 2002 horizontal dribbler circuit. Firstly, the value of the voltage-measuring resistor was rather high (1 Ohm) and this caused some power to be wasted. Secondly, the measured voltage had a lot of noise present due to the inductive nature of the dribbler motor. Therefore, the voltage signal to the main micro controller was pretty unstable, making it almost impossible to control the torque of the dribbler.


Vbatt
HDribb

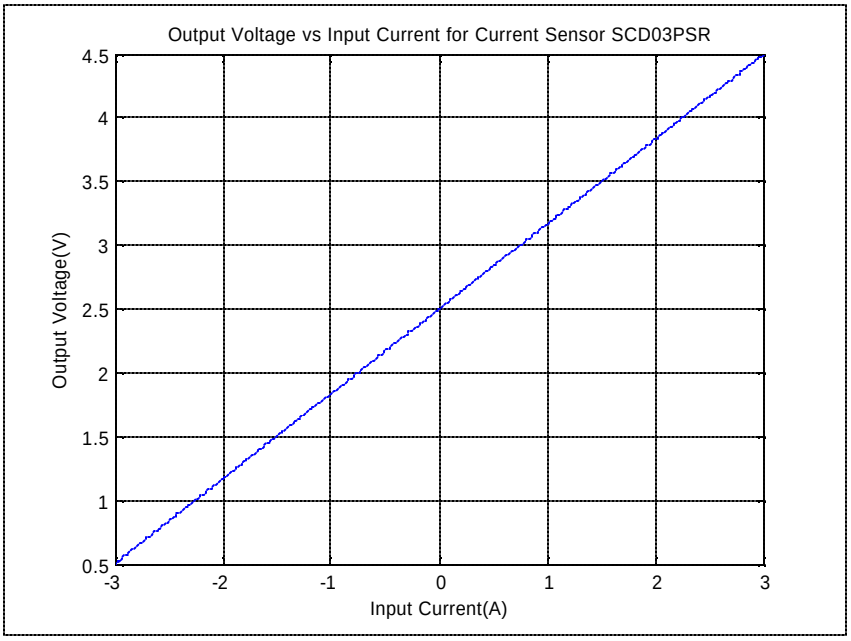


Figure 10.4 Performance graph of the current sensor

However, the current sensors were not as accurate as we hoped. According to the performance graph, when there is no current flowing, the output from the sensor should be 2.5V. However, the output measured was 2.6V. The output also fluctuated a lot when the sensor was first powered up and it only stabilized when the sensor had been switched on for some time. When different values of current were input to the sensor, the corresponding voltage outputs were not very accurate either. They were usually inaccurate by a factor of roughly +0.1V.

There were also problems with the range of output voltage values from the sensor. The range of output voltage values corresponding to the currents flowing to the dribbler is 2.5V to 3.83V. For effective torque control, this range needs to be amplified (from 0V to 5V) so that better resolution can be achieved. One

control, thii se -0.6921 Tc 0.198 TD dirffeconalhe(0) Tjie1, Tj -90nin0.0098 Tc 020853 Tw current sensat onl (measusly) Tj

of the circuit is then input to the main micro controller, which then sends instructions to slow down or speed up the dribbler, based on the value of the current.

We conducted preliminary testing of the circuit with the difference amplifier. However, we realized that torque control using a current sensor might not be very viable. The current flowing to the dribbler motor depends on mechanical aspects of the construction of the dribbler. For example, if the belt leading from the motor to the dribbler on one robot is tighter than that on another robot, this will cause more current to flow to the dribbler motor of the first robot even when both the dribblers are spinning at the same speed. Therefore, it is difficult to predict the value of the current flowing to the motor for specific situations because it will vary across the robots. However, this problem was eliminated due to the mechanical design of the 2003 robots, where belts were replaced with gears.

More importantly though, the current sensor is not very accurate in measuring the current and there might be variations in performance for each current sensor, making it difficult to calibrate the output voltage values of the sensors.

This factor combined with the cost and size of the current sensor caused us to reject this approach to measuring the speed of the dribbler.

10.2.2 Final Design Using Hall Effect Sensors

We then decided to try using Hall Effect sensors to measure the speed of the dribbler. The dribbler motor was driven using a H-bridge and an encoder is attached to the end of the motor. The encoder outputs two pulses that provide data about the speed and direction of the motor.

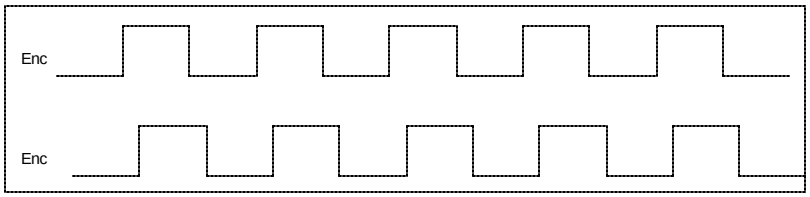


Figure 10.5 Encoder Output Pulses

The encoder we chose has 32 counts per turn. This means that for each revolution of the motor, there will be 32 rising edges on each pulse. If pulse B trails pulse A (as shown in the figure), then the motor is turning in a clockwise direction and vice versa.

The output pulses are input to the FPGA, which quadruples the signal and hence the number of counts. (Please refer to the FPGA documentation for details of this process.) We then wrote code in the main

micro controller, which reads the number of counts from the FPGA and converts the number of counts into the speed of the horizontal dribbler.

Feedback control was then implemented so that the dribbler was able to spin at a commanded velocity. A desired dribbler velocity (in terms of desired number of counts per frame) is input to the main micro controller from the AI system, based on the velocity vector of the robot. The actual dribbler velocity is detected from the number of counts output from the encoder. The difference between the desired and actual velocities (the error) is then minimized using proportional integral control. This feedback control system is shown below.

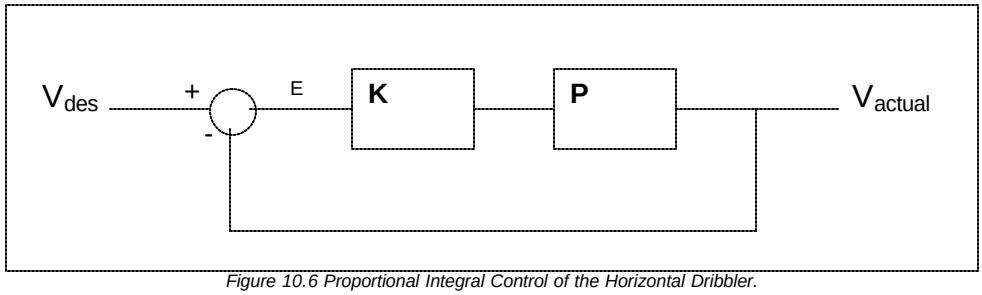


Figure 10.6 Proportional Integral Control of the Horizontal Dribbler.

K is the proportional integral controller and P is the plant.

For a proportional integral controller,

$$K = K_p + \frac{K_i}{s} \quad , \quad P = \frac{1}{Ts+1} \qquad \text{... Eq 10.2}$$

The values of the gains, K_p, K_i and the maximum anti -windup (saturation level of the integral) were adjusted so that the dribbler speed could be varied when desired. The details of the code can be viewed in the appendix.

Using the encoder to measure and control the speed of the dribbler worked very well and was implemented in the 2003 dribbling system.

10.3 Side Dribbling System

Two side dribblers are also placed at 45^o angles (to the vertical axis) on both sides of the horizontal dribbler to “catch” the ball when it wanders to the side of the horizontal dribbler. The side dribblers are controlled by a pulse width modulated signal (PWM). When the PWM (command V_Drib from the micro controller) is high/on, the MOSFET switches on and current flows in the circuit.

In 2003, we replaced the Fairchild NDP5070L MOSFET with the IRF7822 MOSFET. The new MOSFET has a higher current capacity and when it is switched on, it has a lower resistance between source and drain. Therefore, it is more efficient than the previously used MOSFET.

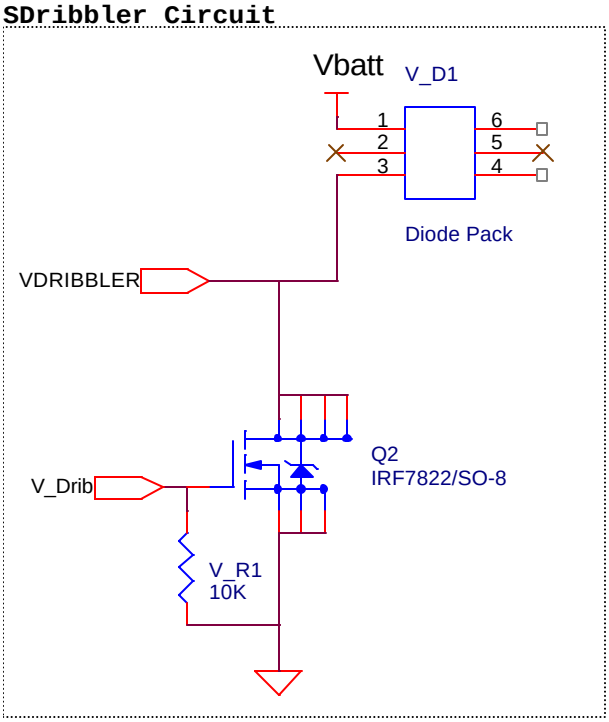


Figure 10.7 2003 Side Dribbler Circuit

ELECTRICAL DRIVE SYSTEM SECTION 11

11.1 Overview

The purpose of the electrical drive system is to provide a control and feedback interface between the microcontroller and the drive motors. If one were to look at a robot as a human, the electrical drive system would be the nerves that allow the brain to control the muscles and also provide the brain with feedback from the muscles.

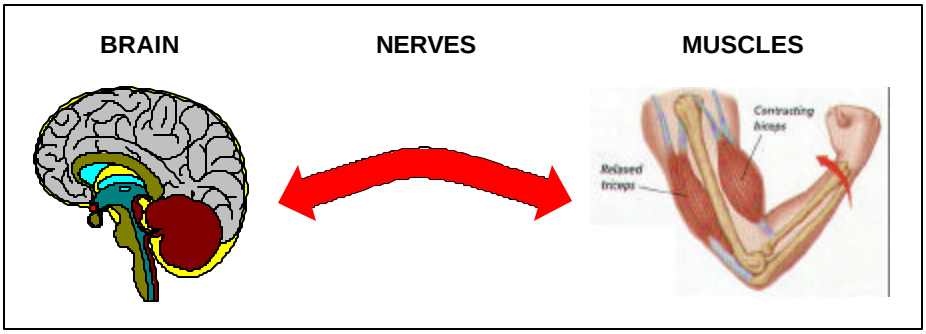


Figure 11.1 Conceptual Overview of the Electrical Drive System.

Essentially, the electrical drive system provides the following interface between the microcontroller and the motors. The microcontroller sends two bits of data to the electrical drive system (a PWM and direction). These two bits get transformed to voltages applied to the motor terminals. In the reverse direction, the motor encoder sends two bits of data to the electrical drive system (channel A and B). These two bits of data are transformed into a single pulse that is a quadrupled version of the original pulses; this quadrupled pulse gives the microcontroller four times the resolution of a original pulses. This quadrupled pulse is sent to the microcontroller.

11.2 Introduction

Last year's electrical drive system utilized full H-Bridges and an FPGA. A block diagram of last year's electrical drive system is shown below.

While this design proved to be robust, there were performance related issues. The H-bridge has an on-resistance (RDS_on) of 0.3 ohms. This is quite high especially when the motors are using a lot of current. For example, let's say the motor is pulling 3 amps of current and the battery voltage is 12 volts. In this case the voltage drop across the H-bridge is 0.9 volts; therefore, the voltage across the motor drops to

11.1 volts instead of the nominal voltage of 12 volts. This decrease in voltage across the motor results in decreased acceleration of the robot.

There was also concern about the H-bridges burning up. They are only rated for a maximum current of 4 amps; the stall current of the drive motors is around 3 amps. While the stall current is under the maximum current limitation of H-bridges, the H-bridges do get quite warm during a game.

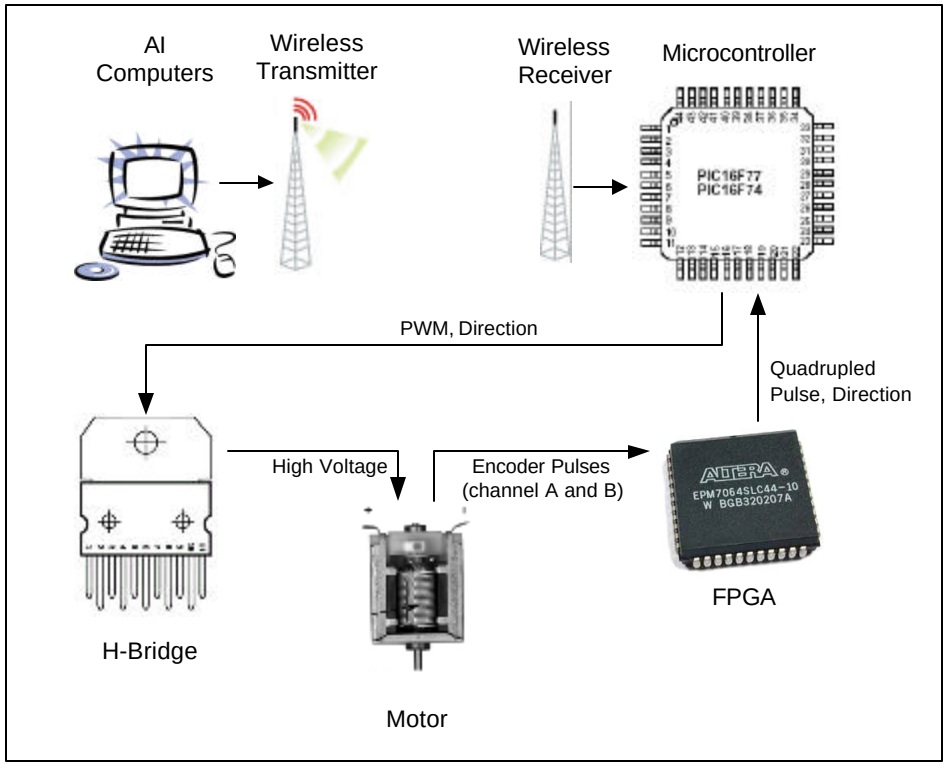


Figure 11.2 Overview of Electrical System's relation to Drive Control

11.2.1 Design Problem

The primary goal for this year's electrical drive system was to design a system that minimized the voltage drop across the analog circuitry that switched the voltage on and off at the motor terminals. This would allow for more of the battery voltage to be across the motor and therefore result in higher acceleration for the robot.

11.2.2 Solution

To achieve this goal it was necessary to replace the H-bridge of the old design. First, we set out to find either another H-bridge or implement the same function using an alternative method. This new H-bridge or alternative method must allow high current (up to 4 amps) and fast switching, since the microcontroller will apply the PWM to the motor.

Our survey for new H-bridges turned up no H-bridges that met our specifications. For some reason, there are not many commercially available, high current H-bridges available in the market. This may be either because commercial systems require unidirectional motion that only needs half H-bridges or because commercial systems are starting to use a larger percentage of brushless motors.

We quickly looked to trying to replicate a full H-bridge using two half H-bridges. We found two primary options: an STmicro modified half H-bridge and a Fairchild half H-bridge.

The STmicro modified half H-bridge (VND670SP) consisted of a device which acted like the top half of a full H-Bridge. It contained the two pMOS transistors needed for the top half of the bridge along with voltage boost converters that allowed the chip to directly interface with 0-to-5V digital logic. The pMOS transistors of this device had an RDS_ON of 0.03 Ohms. To make a full H-bridge, this chip would need to be combined with two nMOS transistors. There are single nMOS transistors available, which have an RDS_ON of 0.016 Ohms; therefore, an implemented H-bridge using this STmicro modified half H-bridge would result in an H-bridge with an RDS_ON of 0.05 Ohms, which is a significant improvement. This solution seemed like the ideal solution. Unfortunately, this idea was dropped because this STmicro chip is still in development and not commercially available.

The other half H-bridge that we found was manufactured by Fairchild (FDS4501H). This half H-bridge was a traditional half H-bridge, meaning that one chip contained an internal nMOS and pMOS transistor. Therefore, to create a full H-bridge, two of these chips would have to be used side-by-side. The problem with this implementation is that the chips had no built-in voltage boost converters, so we would have to provide the pMOS transistors with 12 volts. Also, availability was a big issue because these chips were brand new and not available to us.

The alternative to using the two half-bridges is using individual transistors to implement the H-bridge. Using individual transistors has the advantage of even lower RDS_ON than when using two half H-Bridges because there are individual transistors that have smaller RDS_ON than the transistors in the half H-Bridges. We decided to use the individual transistors instead of the two half H-brides so we could minimize our RDS_ON.

The challenge of providing 12 volts to turn off the pMOS still remained. We decided to use an nMOS transistor to switch the voltage of the pMOS gate. Therefore each H-bridge consists of 6 transistors in total (see following diagram):

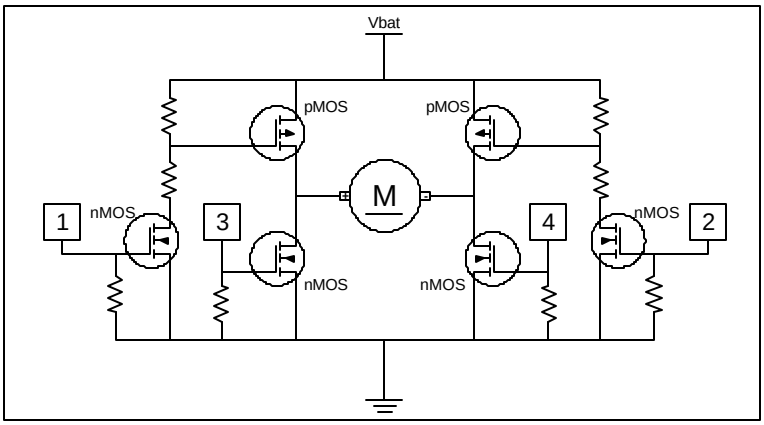


Figure 11.3 H-Bridges

In the diagram above, the inputs to the H-Bridge are shown as 1, 2, 3 and 4. The outputs are show directly connected the to motor terminals. Inputs 1 and 2 indirectly control the top half of the bridge, the pMOS transistors. When either of the inputs is high (5 volts), the corresponding pMOS is on, meaning that current is flowing through the branch. Inputs 3 and 4 directly control the bottom half of the bridge, the nMOS transistors. When either of the inputs is high (5 volts), the corresponding nMOS is on, meaning that current is flowing through that branch. Here is a chart of the motor state and the corresponding inputs:

MOTOR STATE	INPUT 1	INPUT 2	INPUT 3	INPUT 4
Forward	1	0	0	1
Reverse	0	1	1	0
Coast off	0	0	0	0
Brake	0	0	1	1

Table 11.1 Motor State table

The pMOS transistor used in the H-bridge is the IRF7424. It comes in an SOIC-8 package that contains one transistor per package. Its RDS_ON is 0.0135 Ohms. It maximum current rating is 11 Amps.

 Tj - Tj -re h W n BTn24 0 258443.25 760.5 TD 1

The electrical motor drive system works as follows:

- The microcontroller receives velocity and direction data from the artificial intelligence system.

- The microcontroller using a PI motor control algorithm converts this data into a pulse width modulation (PWM) signal and a direction bit. The PWM signal is a single bit signal; when it is '1', voltage should be applied to the motor, when it is '0', voltage should not be applied to the motor. The direction bit indicates which direction the motor should rotate.

- The two bits of data (the PWM and direction) are sent to the FPGA. The FPGA decodes this information to form a four-bit output.

- This four-bit output controls the individual transistors in the H-bridge. The H-bridge in-turn controls the voltage on the motor terminals. The voltage on the motor terminals is either 0 V, 12 V, or high impedance.

- To provide feedback, the motor encoder sends two pulses to the FPGA. These two signals are square pulses and are 90 degrees out of phase from each other.

- The FPGA uses both encoder pulses to extract a single quadrupled pulse and the direction of the motor rotation. The quadrupled pulse has four times the resolution of either of the original pulses. Both the quadrupled pulse and the direction are sent to the microcontroller.

The electrical drive system can be broken into the following components: the PI motor control system, the FPGA, and the H-bridge. The PI motor control system is discussed in its own section of the document. The H-bridge was discussed in detail above in the 'Solution' section of this document. The FPGA will be discussed in detail below.

11.3 FPGA

11.3.1 Overview

The FPGA provides two functions for the robot. First, it converts the PWM signal and direction bit from the microcontroller to the four-bit H-bridge command. Secondly, it uses the A and B encoder pulses to extract a quadrupled encoder pulse and the motor direction.

11.3.2 Selection of the FPGA

The following paragraphs from the 2002 Robocup Documentation apply for this year as well:

> *"Altera has been a generous sponsor from the past, and this year was no exception. They donated development software, design laboratory kits, and the FPGAs used in final production. We compared a number of FPGAs in the MAX 7000 family to choose one that satisfied our requirements.*
>
> *One major factor we considered was in-system programmability since the FPGAs are to be programmed on board. MAX 7000S devices are in-system programmable via an industry-standard 4-pin Joint Test Action Group (JTAG) interface. The MAX 7000S architecture internally generates the high programming voltage required to program, making it programmable with a 0-5V input."*

This year's FPGA design is very large, meaning that it does not fit on many of the FPGAs that are in the MAX 7000 family. The final board design uses the EPM7160STC100-7 FPGA because it is the only small, surface-mount (TQFP) chip that is big enough to implement the design. More than 91% of the gates in the FPGA are used. Next year's team should consider moving to a different family or alternatively using Actel or Xilinx FPGA.

11.3.3 Detailed Implementation

Last year's FPGA design was done in schematic mode, meaning that the FPGA was programmed using pre-coded digital modules that replicate digital logic gates and chips. These modules are wired together in a CAD-like program. This method of coding an FPGA wastes space and is less robust.

This year's FPGA design was coded in VHDL. The advantages of VHDL include a more optimized design. Also since the code is similar to C, it is easier to maintain. Most significantly, the code is portable between different FPGA families and even between FPGAs manufactured by different companies. In effect, this code can be compiled and programmed on to any FPGA.

11.3.4 H-Bridge Commands

The first function of the FPGA is to "decode" the PWM and direction commands from the CPU into the four-bit H-bridge command.

The FPGA switches the pMOS transistors independently of the nMOS transistors. The pMOS transistors are controlled based only on the direction bit. If the direction bit is '0', the first pMOS (input 1 to the H-bridge) is on (set to '1') the second pMOS (input 2 to the H-bridge) is off (set to '0'). Therefore, one pMOS transistor remains on at all times.

The nMOS transistors are switched according to both PWM and direction. If the PWM is '1' and the direction is '1' then the first nMOS (input 3) is off, and the second nMOS (input 4) is on. If the PWM in '1' and the direction is '0' then the first nMOS (input 3) is on, and second nMOS (input 4) is off. If PWM is '0' then both nMOS transistors are off.

Inputs from Micro		Outputs to H-Bridge			
PWM	Direction	Input 1	Input 2	Input 3	Input 4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	1	0	0	1
1	1	0	1	1	0

Table 11.2 Input and Output table

If you look at the table above, the motor is not moving in the first two cases. In the third case, the motor is rotating in one direction, and in the fourth case the motor is rotating in the opposite direction.

The nMOS and pMOS are switched independently because we did not want both the nMOS and pMOS transistors to turn on and off with the PWM. The pMOS transistors have significantly longer turn on and off time than the nMOS transistors. Therefore, it seems more optimum to just switch the nMOS transistors on and off with the PWM and leave the pMOS transistors on or off coupled to just the direction.

There is one failure mode for the H-Bridge control. Failure mode is when both the pMOS and the nMOS transistor of one side of the bridge are on. This failure mode occurs when direction is changed and the PWM is '1'. When direction changes, one pMOS is slowly turning off and the other is slowly turning on. If this happens when the PWM is '1', the nMOS turns on before the corresponding pMOS transistor turns off. Therefore there is short from power to ground directly through the pMOS and nMOS transistors; this short occurs for approximately 100ns until the pMOS transistor turns off.

To make sure that this failure mode does not occur, the microcontroller code was altered so that direction is only changed when the PWM is at '0' and the PWM is held at '0' for at least 150ns.

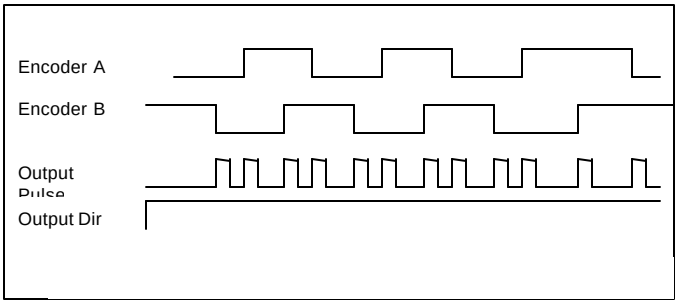


Figure 11.5 Sample Input from Motor and Output from FPGA

11.3.5 Quadruple Encoder Pulse and Direction

The second function of the FPGA is to quadruple the encoder pulse and determine the motor direction.

The encoder pulse is quadrupled to give the microcontroller four times the resolution of the location of the motor rotor. Using the encoder pulse the microcontroller is able to determine the location and the velocity of the motor.

To quadruple the encoder pulse, the FPGA is programmed to output a small pulse whenever either of the FPGA channels shows a rising or falling edge. These pulses are approximately 20ns in length.

Since the A and B channels of the encoder are offset by 90 degrees, the direction of the motor is simply a D flip-flop (DFF) where the clock is Encoder A and the input value is Encoder B. The output of the flip-flop is the direction that the motor is traveling.

11.3.6 FPGA: Horizontal Dribbler Differences

The FPGA and electrical drive system is used to control the 4 drive motors. As an addition, the horizontal dribbler will also be controlled via this system. This means that we will be able to control the speed and direction of the horizontal dribbler. This will allow very precise control of the dribbler.

In the dribbler drive system there is one extra bit of data that is sent from the CPU to the FPGA; it is the STOP_NOW bit. The STOP_NOW bit ties both motor control terminals to ground, thus the horizontal dribbler stops immediately. This was added to allow the CPU to stop the dribbler before kicking; this allows for a stronger kick, as the ball does not have backspin on it.

This STOP_NOW bit is implemented as active high and takes precedent over the PWM and direction bits. The FPGA VHDL code can be found in the appendix.

POWER AND BATTERIES SECTION 12

12.1 BATTERIES

12.1.1 Overview

Our robots are powered by rechargeable batteries. A correct selection of our batteries is very crucial because for example, if the robots are out of battery in the middle of the game, no matter how good our design of the robot is, the robot will just stay at the same place without doing anything.

There are four main concerns when we did our battery selection this year:

- Maximum current drain and internal resistance
- Capacity
- Weight and size
- Number of batteries to be placed in the robot.

12.1.2 Introduction

Maximum Current Drain and Internal Resistance

Both the maximum current drain and internal resistance contributes to a high power output. Our robots are a high-power-drain mechanism, since the various motors in our robots demand high power. Since our batteries will be providing a constant voltage, and

$$Power = Voltage \times Current \qquad \text{...Eq 12.1}$$

we have to provide a high current supply in order to provide high power with a constant voltage.

Internal resistance is minimized as much as possible. Let R_i be the internal resistance and R_L be the load resistance that we are going to drive.

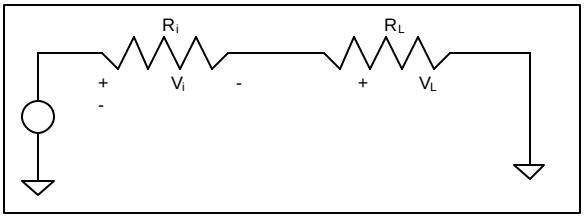


Figure 12.1 Voltage Divider Circuit

By the principle of Voltage Division,

$$V_L = V_i \frac{R_L}{R_L + R_i} \qquad \text{.... Eq 12.2}$$

At constant current, refer to equation14.1, Power is directly proportional to Voltage, so

$$P_L = P_i \frac{R_L}{R_L + R_i} \qquad \text{...Eq 12.3}$$

Let's assume we have a constant power source, in order to provide a high power output, P_L, we would like to have the fraction in equation 14.3 equal to unity. To do that, we will need R_i to be as small as possible.

As Li-Ion has high internal resistance, the main focus would be on NiMH and NiCd batteries, which has the better current drain curves and tiny internal resistance. Next, we ruled out NiCd because of their memory effect and their low power density. Thus, for the time being, we should still be going with NiMH, unless we can get some way to utilize the idea of Li-Ion.

Capacity

Capacity is another factor because the robots cannot run out of batteries in the middle of the game. We should choose the batteries that have the most capacity. For reasons that we will discuss below, we want to have the least weight. Since capacity is proportional to the weight; less weight means less capacity. Therefore, we have to choose the capacity of the batteries according to our needs.

Weight and Size

Size is always an issue because the robot's size is limited and so we would like to have the batteries as small as possible, so we can put more mechanical or electrical components into the robot for more powerful applications.

Weight is also an issue because the lighter the overall weight is, the faster the robot can move. On the other hand, the robot will also be draining less energy from the batteries because it does not require as much power to move the robot from one place to another. Therefore, less weight will give us the benefit of using less capacity batteries, which in turn means batteries that have less weight.

Selection of Number of Batteries

In this year's selection, we tried to eliminate the kicker and the dribbler batteries because they take up a significant amount of space and weight in the robot. The addition of the extra kicker and dribbler batteries in 2002 was for supplying a higher voltage and current to the solenoid and the dribbling motors. By connecting batteries in series, the resultant voltage is the sum of all the batteries' voltage. Therefore, in

order to supply the desired voltage of16V in the dribbling motor, we added another 4V of battery for the dribbler. Similarly, 7.2V extra was added to the kicker.

A phenomenon that we noticed was if we also increase the voltage supplied to the driving motors, the driving motors could be overdriven to achieve a higher acceleration and speed. In that case, there is no point of supplying only 12V to the Analog Board, since all the devices on the analog board will be drawing more than 12V. We therefore decided to increase the number of batteries in the main battery pack and take out the extra supporting batteries.

12.2 Comparisons on Different Cells

12.2.1 Maximum Current Drain and Internal Resistance

Our first approach was to look at the batteries that have the same size as the 2002's selection. Below are the comparisons that were found on the high current drain SubC cells.

Cell	Type	Weight	Impedance	mAh	mWh	Avg. V
Sanyo CP-2400SCR	NiCd	58g	5.3	2272	2535	1.118
Sanyo RC3000	NiMH	59g	5.4	2572	2952	1.148
Sanyo RC2400	NiCd	59g	4.4	2355	2684	1.140
Panasonic HHR300SCP	NiMH	57g	4.9	2894	3270	1.130
MT2500	NiMH	60g	5.5	2544	2785	1.105
GP2500	NiMH	55g	8.1	2416	2556	1.058
Saft 3000	NiMH	58g	6.6	2588	2805	1.090
SR 2000MAX	NiCd	58g	5.2	1972	2155	1.093
Sanyo N-1900SCR	NiCd	58g	5.4	2077	2293	1.104
Sanyo N-3000SCR	NiCd	80g	4.8	2972	3322	1.118

Table 12.1 SubC Cell Comparison Chart

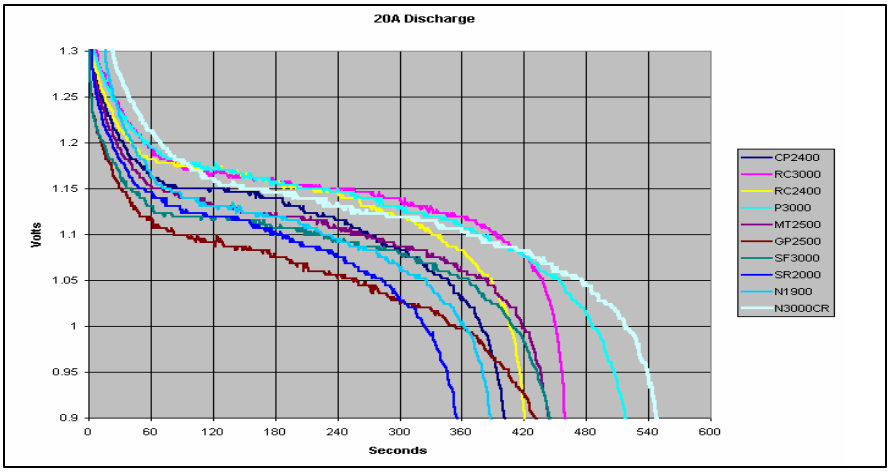


Figure 12.2 Discharge Graph of SubC cells at 20A

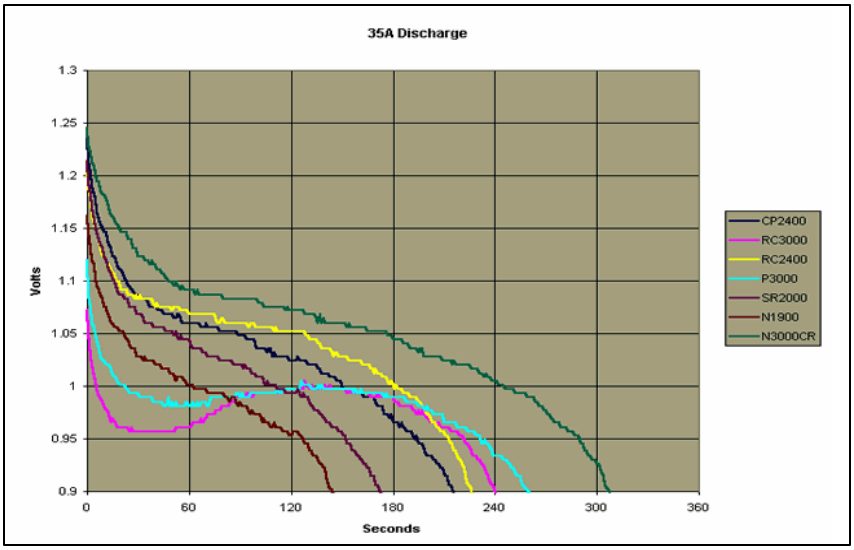


Figure 12.3 Discharge Graph of SubC cells at 35A

The P3000 curve is the 2002 battery discharge curve. It is shown that even at 35A discharge, the battery still performs pretty well. The best curve is from Sanyo's N-3000SCR cell. However, because this is a NiCd cell, as we discussed at the end of section 1.1.2.1, this cell is ruled out.

The next step we tried was looking at the slightly smaller cells, the 4/5 SubC Cells. The 4/5 here basically means these cells are four-fifths of the height of the normal SubC cells.

Cell	Type	Weight	Impedance	mAh	mWh	Avg. V
Sanyo N-1250SCR	NiCd	40g	6.9	1272	1363	1.072
Panasonic HHR200SCP	NiMH	40g	7.3	1705	1793	1.052
Sanyo CP-1700SCR	NiCd	40g	4.7	1655	1825	1.103

Table 12.2 SubC Cell Comparison Chart 2

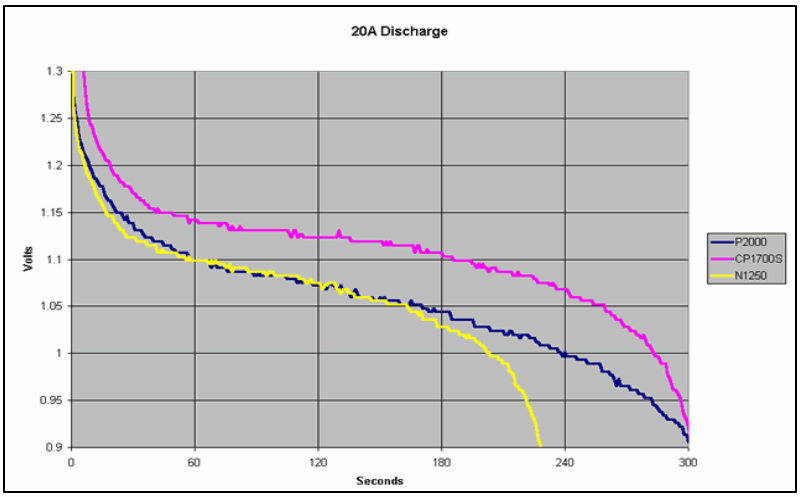


Figure 12.4 Discharge Graph of 4/5 SubC cells at 20A

We see that Panasonic's HHR200SCP performs pretty well at a 20A discharge level.

Device	Current Drain Worst case (A)
Motor (x4)	2
Horizontal Dribbler	2
Total	10

Table 12.3 Current Drain Assumption

Calculations show that the worst case of current drain for the motors is 10A. Including some other negligible devices, we are confident that 15A current drain rating is good enough for the 2003 robots. Therefore, this HHR200SCP cell is a strong candidate of replacement of the 2002 batteries.

12.2.2 Capacity

We ran some tests on the 2002 batteries. The objective was to see how long the batteries lasted. The test consisted of turning on a robot and then controlling it with the game pad.

The following parameters were used during the test:

- Robot was running all the time
- Kicker was turned on 5% of the time
- Dribbler was turned on 10% of the time
- Velocity was kept at 2 m/s
- Acceleration was kept at 4.5 m/s^2

As a result, we noticed the following:

- Speed slows down (still at a good speed) at around 18 minutes
- Slow (not good enough to be in a game), but still moving over 30 minutes

According to the results, ***the capacity of our current battery selection exceeds our requirements***. In a game, we can switch batteries at half time. Each game period is 10 minutes, so all we need is 10 minutes of running time. Thirty minutes is more than enough. In addition, the test that we ran also drains more from the battery than the robot will drain in an actual game. This is because in an actual game, the running time of each robot is much less than the whole 10 minutes. Since we have five robots on the field, not all of them are running all the time. In conclusion, we can look for less capacity batteries to save weight and space for our robot.

12.2.3 Weight and Size vs. Capacity

Having the above observation, we look at the tradeoff between weight and capacity. Take Panasonic's batteries as an example, where we compare HHR-200SCP with HHR-30SCP,

Battery Part No.	Capacity (mAh)	Dimension (mm)	Weight (grams)
HHR-200SCP	2000	23.0 x 34.0	42
HHR-30SCP	3000	23.0 x 43.3	55

Table 12.4 Weight vs. Capacity Comparison Chart

We see that when the weight increased about 25% from HHR-200SCP, we see an increase in capacity of 50%. In addition, when we have 9 cells in our 2002 robot, the overall weight will decrease from 495g to 378g if we changed from HHR-300SCP to HHR-200SCP. It is only a matter of 117g. This is pretty negligible to the overall weight of the robot. This was the reason that the 2002 robot was using HHR-30SCP instead of HHR-200SCP. We can therefore conclude that **weight is a less important factor than capacity**.

The use of lithium-ion (Li-Ion) batteries was raised; because of its high power density feature. However, it would not work because Li-Ion batteries lack the high current drain requirement. An idea was to connect the batteries in parallel in order to provide a high current. But there are still some other problems since not all cells are identical, mismatches in parallel connected cells could be a problem. Therefore, the usage of Li-Ion cells was not approved.

12.2.4 Selection of Number of Batteries

Tests were run by the ME people to determine the maximum voltage that the driving motors will take to give the best performance. The result of their tests shows 15V is the ideal voltage required. Our target was to shoot for 15 batteries.

Batteries are placed on the bottom of the robot because of the heaviness of the batteries. If we place the batteries on the top of the robot, the robot will tip over easily. Since, the driving motors and the wheels have to be placed at the bottom of the robot as well, there is not much space left for the batteries. Therefore, during the battery search, we had to look for some "slimmer" batteries, so we could fit them all in. The SubC cells that we used to use are too "fat" and take up too much space. Size A is the next to consider, as there are none in size AA and AAA cells that are high current drain. After careful measurements, it was determined that the most we can fit onto the robot is 12 size A cells.

12. 3 Conclusion

Our Final Selection

With the previous four attributes in mind, we found that Sanyo's HR-4/3AUP and HR-4/3FAUP are ideal candidates for the 2003 robots. We set the requirements as follows:

i. *current drain*: 15A
ii. *internal resistance*: 6Ω
iii. *capacity*: 2000mAh
iv. *size*: A or below

Battery	HR-4/3AUP	HR-4/3FAUP	HHR-300SCP
Brand	Sanyo	Sanyo	Panasonic
Capacity (mAh)	2900	3200	3000
Max. Current Drain	Spec: 15A Other sources: N/A	Spec: none Other sources: N/A	Spec: 20A Other sources: 35A

Discharge Curves:

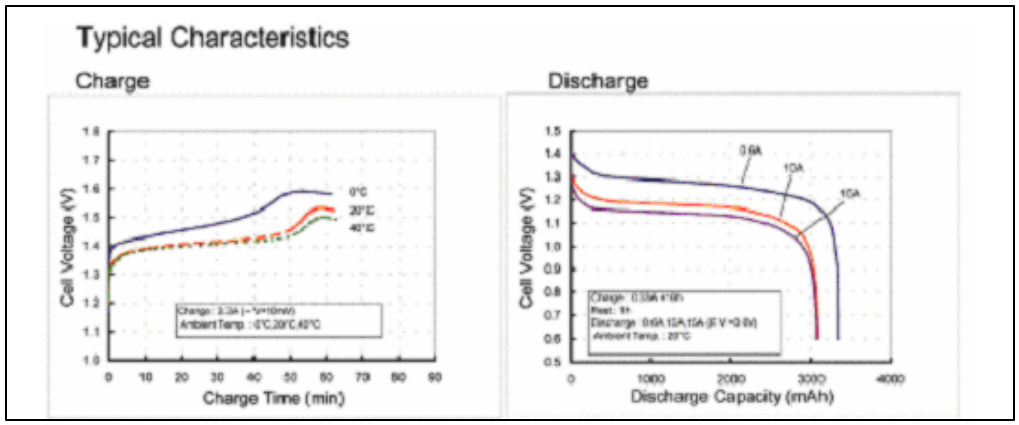


Figure 12.6 – Charge and Discharge Curves for HR-4/3FAUP

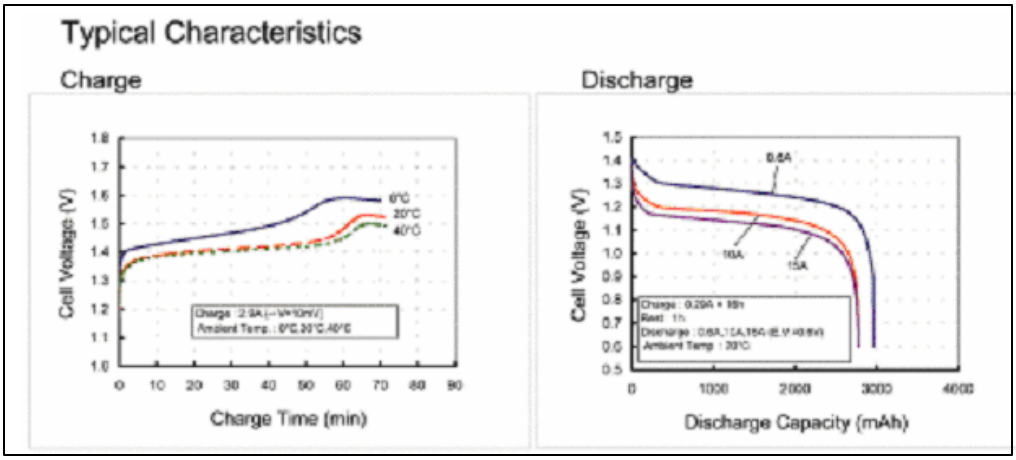


Figure 12.7 – Charge and Discharge Curves for HR-4/3AUP

12.4 Zapping

Zapping is usually done to batteries that have been used for a while. Batteries that have been used for a while can lose some of their ability to retain voltage due to chemical breakdown and crystallization within the cells. While no "miracle cure" exists to bring these cells back to their original glory and splendor, they can often regain much of what they've lost through "zapping."

Zapping is done by putting a very high current through a cell. Usually a bank of capacitors, about 70,000uF, is charged to about 70V. The energy is discharged through the cell resulting in a few thousand Amps for a very short time. In cells that have been used for a while, zapping can cause the crystallization to break down, lowering the cell's internal resistance and reducing "wasted" energy inside the cell. In new cells, it welds the cell's internal connections. The welded connections reduce the cell's internal resistance. This results in a higher voltage under high loads. As far as we know, all serious electric flight competitors used zapped cells. At the higher currents used, the voltage increase is noticeable.

Zapping only works on some cells. It appears that only high current cells such as Sanyo SCR series and Panasonic SCP series respond to zapping. This is probably because these cells use metallic electrodes. Many high capacity cells use a foam electrode and foam can't be welded. Both NiCd and NiMH cells can respond to Zapping. Properly done zapping does not seem to have any negative effects on cells.

Although zapping was not used on our batteries, we believe next year's team should look into this method more closely.

CELL	TYPE	WEIGHT	AMPS	IMPEDANCE	MAH	MWH	AVE. V
Sanyo HR-4/5AUP	NiMH	32g	20	7.6	1565	1580	1.01
HR-4/5AUP Zapped	NiMH	32g	20	6.2	1565	1610	1.08
Sanyo HR-4/5AUP	NiMH	32g	15	7.6	1565	1660	1.06

Table 12.6 Comparison between HR-4/3AUP, HR-4/3FAUP and HHR-300SCP

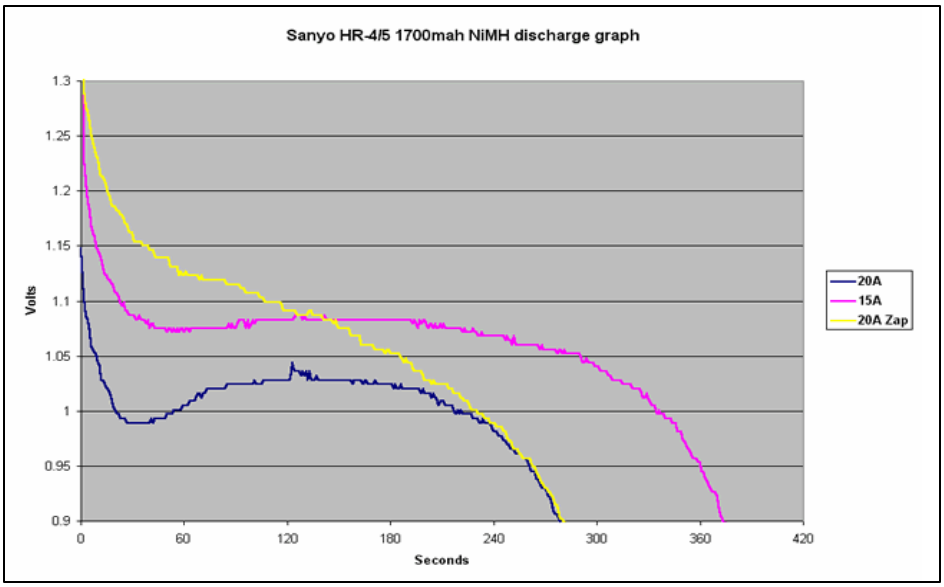


Figure 12.8 – Discharge Graph for HR-4/5 (Before and After Zapping)

From the graph above, we see that the difference of Zapping is very obvious. The voltage of the battery in high current discharge is steadier.

12.5 Further Research

We are currently looking at some Lithium Polymer Cells, E-Tec's 1200mAh cells. As we discussed earlier in the 1.1.3.3 section, Lithium cells are beneficial because of their light weight. However, the drawback was the current drain in them is poor. These E-Tec cells are proven to be the leader in powering RC electric model aircraft. They are claimed to support up to 7.2A of current drain. We believe this type of battery have a high potential of replacing NiMH.

12.6 VOLTAGE REGULATOR

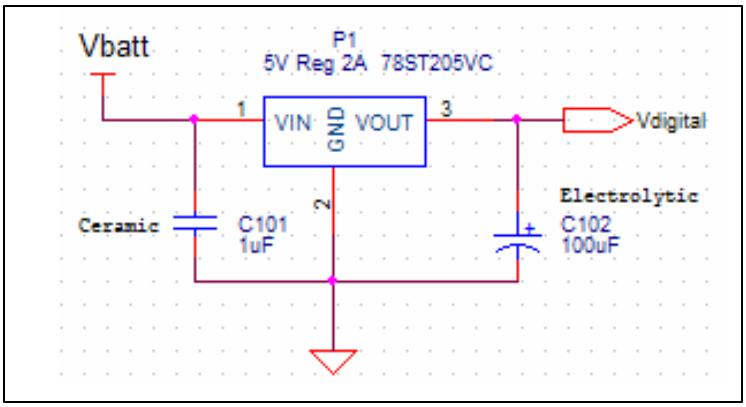


Figure 12.9 – Voltage Regulator Circuit

12.6.1 Overview

Our robot has two main circuit boards: Analog Board and Digital Board. On the Analog Board, we place high current-drain devices like the motors and power circuits in it. On the other hand, on the Digital Board, we place mostly signal devices. All the devices on the Digital Board are powered by a 5V source. Since we cannot drive the devices on the Digital Board with our 14V battery voltage, we need a robust device to convert a 14V source to a 5V source for our Digital Board devices. This device is the Voltage Regulator.

12.6.2 Background information

In 2002, a self-contained, COTS (commercial off the shelf) regulator is chosen because of the regulation issues on the 2001 board. The regulator that they selected 78ST205VC, 2A, 5V integrated switching regulator (ISR). It provided a solid performance and we decided to stay with the same regulator this year.

However, there was a problem in this 3-pin device. The 3 pins in this voltage regulator are very slim and weak that they could break very easily. To resolve this issue, the 3 pins were connected to 3 different wires securely. The next step was to wrap the voltage regulator in a shrink wrap. Then, the wires are connected to the appropriate places. Lastly, the shrunk wrapped voltage regulator is stuck to some other device on the board firmly. Because of this manufacturability issue that existed last year, we have to look for a new voltage regulator.

After some research, another version of the 2002 regulator was found. The regulator that was used in 2002 is the vertical mount version. This year, we wanted to use the horizontal mount version because

this version has two other mounting pins on the chip to securely mount the chip. The part number of the horizontal mount voltage regulator is 78ST205HC.

12.6.3 Conclusion

Our final selection was to stay with 2002's 78ST205VC. There are two main reasons we stayed with the vertical mount chip instead of the horizontal mount chip.

The main reason was budget. We did not decide to use the vertical mount version because it is cheaper than the horizontal mount version. In fact, they cost the same. The reason behind this decision was because a large amount of the vertical mount chip was found in our inventory; from some unknown reason, the 2002 team bought a lot of them. Each chip costs around $20. We would rather suffer in manufacturing than in hurting our budget; therefore, we stayed with the same chip.

A second reason was that it's a drawback to use the horizontal mounting chip. Horizontal mounting will take up more space on the board than a vertical mounting chip. Although we still have space on the Analog Board, layout routing issues made the vertical mounting chip better for our purposes.

12.7 BATTERY METER

12.7.1 Overview

During a game, it is very troublesome to pick up a multi-meter, unplug the batteries connector from the Analog Board, and then measure the voltage of each robot to see if they are running low in battery. It is very useful to build a circuit that will let us know instantaneously what the voltage of the batteries is when we need it. Therefore, this "Battery Meter" circuit is built for that purpose.

12.7.2 Description

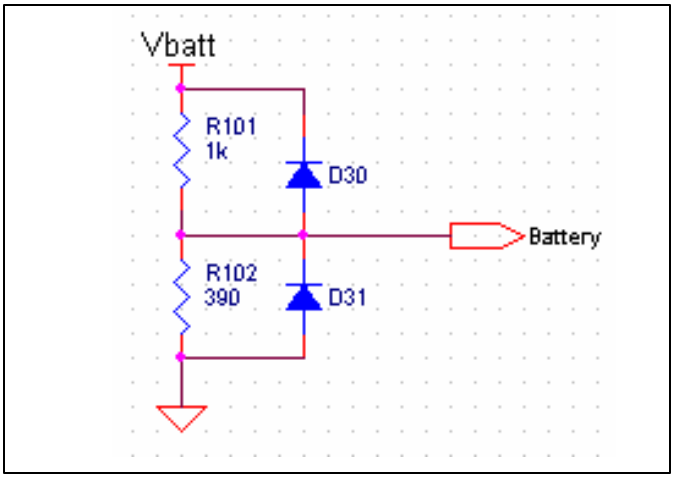


Figure 12.10 – Battery Meter Circuit on Analog Board

The battery meter circuit is basically a voltage divider circuit. We need to scale down the battery voltage to a range of 0V to 5V, so this voltage can enter the microcontroller through an ADC pin. Since we are using 12 batteries this year, and each battery may have a voltage of 1.4V, the maximum of the range is 16.8V. Therefore, 1k and 390 ohms are chosen for the circuit.

$$V_L = V_i \frac{R_L}{R_L + R_i} \qquad \text{...Eq 12.4}$$

$$V_L = 16.8 \frac{390}{1000 + 390}$$

$$V_L = 4.71$$

The resistors here are chosen with 0.1% tolerance. We need the accuracy because we intended to use this battery meter in place of a multi meter. The diodes here are just for safety purpose. They are to prevent the circuit from shorting.

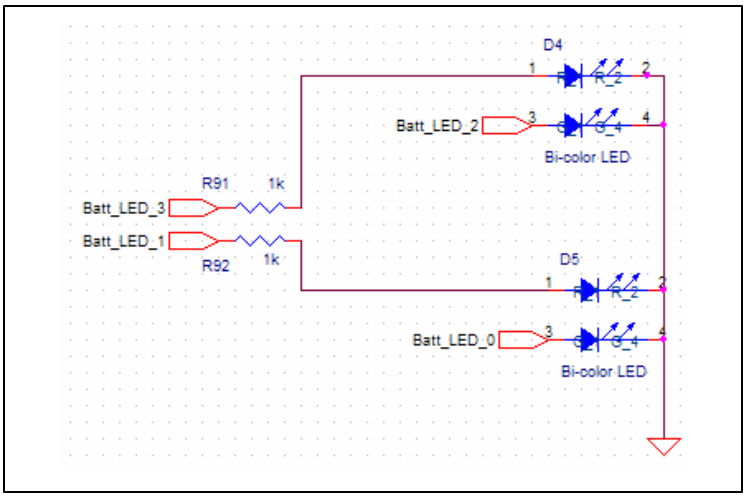


Figure 12.11 – Battery Meter Circuit on Digital Board

Two Bi-color LEDs (Red and Green) are used for the meter indicator. When the battery voltage is at the perfect level, both LEDs should be Green.

Below is a table that shows a simple reading scheme of what the battery voltage the LED colors refers to.

Battery Voltage (V)	Input voltage to Micro (V)	First LED	Second LED
>14.17	>3.977	Green	Green
13.69 – 14.17	3.84 – 3.977	Off	Green
13.32 – 13.69	3.737 – 3.84	Orange (Green + Red)	Orange (Green + Red)
12.95 – 13.32	3.634 – 3.737	Off	Orange (Green + Red)
12.58 – 12.95	3.531 – 3.634	Red	Red
12.22 – 12.58	3.429 – 3.531	Off	Red
<12.22	<3.429	Off	Off

Table 12.7 Battery Voltage Mapping

LAYOUT & SYSTEM INTEGRATION SECTION 13

13.1 Overview

In order to put our circuits together, we need to integrate them onto a circuit board. As in most industrial electronics, Print Circuit Board (PCB) is used as the "container" for our circuits. Bread boards are impractical because components can fall out easily, and space is wasted. Prototype boards which are capable of holding components securely after careful soldering, also have the problem of wasted space.

PCBs are desirable because they hold components securely and also save space because they have many layers integrated into one board. Most importantly, this process allows us to trace our errors, ff there are any. Tracing is crucial in system design because in a complex system like ours, when an error occurs, it would be very difficult to debug if we do not have a well-organized circuit laid out.

13.2 Introduction

This year, we decided on a two board design i.e. an 'Analog Board' and a 'Digital Board'. As their names suggest, the analog board contains primarily analog components and the digital board, primarily digital components.

We decided against the 'Motherboard / Daughterboard' design as was used in the previous year, because the connectors between the motherboard and the daughterboard are huge and take up large amounts of space.

This year, the mechanical design imposed a constraint on the amount of space available vertically, with the end result that there is more space available horizontally. We therefore concluded that the extra space that the 2002 style connectors required be eliminated. Hence our two board design.

A main concern in our design this year was to minimize cost, as there was a huge expense on board fabrication in 2002. To tackle this problem, we set the following objectives for this year's design:

- less layers
- less boards
- have revisions on layout before sending them out for fabrication

13.3 Schematics

The first step toward a successful layout is to have the schematic as complete as possible before layout commences. A useful analogy is that the schematic is a recipe and layout is the act of cooking. At Cornell, we use the software OrCAD Capture to draw our schematics, which will be imported into OrCAD Layout when we do the layout.

Schematics are drawn by each sub group. After each sub-group is done, we put them together into two separate master schematics depending on which board they should be placed: Analog and Digital. When putting the master schematic together we verified that each of the inputs or outputs was labeled accordingly. For example, if the input signal of the IR circuit was IN, we changed it to IR_IN because all the other sub-group circuits also have IN signals!

After that, we labeled the footprint(size) of each part on the schematic. It was not necessary to do this step; however, it made our lives much easier.

Finally, the schematics were exported to OrCAD layout. There is a communication path between the OrCAD's Capture and Layout. This is the netlist. Netlisting translates the Capture data into a "language" that Layout will recognize. After netlisting, we proceeded to Layout.

13.4 Footprints

Footprints can be thought of as ingredients in the cooking scenario. We *specify how much space we allocate to each individual part in footprint*. Usually, for common components like surface mount resistors, capacitors, etc. the footprints are already available in the OrCAD library.

However, for some unique parts (e.g. the accelerometer socket, the LED, the Micro) we had to create footprints for them because the OrCAD library does not recognize them. To create a footprint, we checked the datasheet of the component. On the datasheets, the dimensions of the chip, the pitch between the holes and the holes' diameters are specified. Gathering all these data, we drew the footprint of the component accordingly.

We made sure our footprints were perfect. For example, we labeled pin 1 with number 1. This is because even though the first pin is denoted by a square as default in OrCAD, while the other pins are denoted by a circle, if you put a component like a header on it, one will not be able to see the square shape of the pin anymore.

13.5 Schematics II

Layout is the translation of a paper schematic to a printed circuit board. Layout involves defining the physical shape of the board, the locations of the components, and the interconnections between. Usually the schematic drives the layout, but it is not necessarily a one-way relationship. Sometimes, constraints such as space (or in the analogy, lack of certain cooking utensils) make it impossible to make the recipe as first envisioned, and the schematic/recipe has to be altered in some manner.

First, we imported the netlist into layout; all the components that are specified in the schematics automatically appeared on the layout screen, with all the interconnections shown as thin yellow lines. We then created the boundaries of the boards and then placed the components within the boundaries.

We placed components together that were connected together in the schematic, as that reduced the number and length of traces which increased the probability that the auto route function would actually work.

We first tried to route the boards with auto-route once we were done with component placement. However, we never achieved a 100% routing: some connections could be routed. Attempts like replacing components, spacing out the components that have small pitches were made, but the board could not be auto-routed fully. Finally, we recognized the problem. It was the wide traces.

We then hand-routed the wide traces before running the auto-route operation. Auto-route does not do the best job in routing. We did not notice it as much with small traces that run in weird ways, but the large traces in the analog board taking circuitous routes are visible and crowd the board terribly.

After the designing phase of the board layouts were done, we advanced to the verification stage. We held ”layout parties” to check the connections to the plane layers and connections between different components. In these parties we verified the layout. The importance of this was proved, as in the very first layout party we found traces that lead to nowhere, power shorted to ground, etc. In addition we matched the trace widths to the estimated current flowing through it by checking on the web for charts of current ratings for trace widths.

13.6 Parts selection and board populating

We tried to use as many surface mounts components as possible. There are two main reasons for this. First, they are smaller than through hole parts and so will save more space on the board. Secondly,

through-holes take up the layers' space in terms of routing. This is because through-holes penetrate the whole board, from the top to the bottom, and render the underlying space unusable for routing. Use through holes only when a robust design is needed. e.g. the board to board connectors.

For surface mount components, we used nothing smaller than 0805. It is a pain, though definitely possible but very time consuming, to hand solder anything smaller than 0805.
In addition, we contracted out the population of our boards to BSU. Populating which means putting the components on the board. This company does not like doing anything smaller than 0805's. With the two reasons above, every board layouts, therefore, were done with nothing smaller than 0805's.

They also dislike working on boards with components on the backside; therefore, we had all our components placed on the top layer.

Before sending any boards out to be populated, we first hand-soldered a test board. This was to prevent $800 worth of expensive components being put on bad boards if there was a problem with the layout.

13.7 Final Comments

We tried to use as few boards as possible. The main cost of a board is the setup. It costs roughly $300 for the first board and $10 for each additional board.

We also tried to route the boards with a 4 layer design. This did not only reduce the cost, but also helped in traceability. In other words, debugging the boards is much easier.

In addition, OrCAD does not do a good job in routing board that has more than 4 layers. Weird and faulty connections could result from 6 layers.

RESEARCH SECTION SECTION 14

14.1 PROXIMITY SENSORS

Proximity sensors were researched in 2001 in order to solve the problem that the ball could not be detected because the overhead camera couldn't see the ball due to occlusion by the hat of the robots. The capacitive sensors from Q-Prox's were tested to detect the ball in this case. In the capacitive sensor, the two plates, housed in the sensing head, are placed in a position such that they operate like an open capacitor. They use the air as an insulator: at rest there is little capacitance between the two plates. These plates are linked to an oscillator, a Schmitt Trigger and an output amplifier. As a target enters the sensing range, the capacitance of the two plates increases, resulting in a change in the amplitude of the oscillator, which in turn changes the Schmitt Trigger state, thus, creating an output signal.

Tests in 2001 showed that these sensors were not robust enough to detect the ball for two reasons. Firstly, the ball is non-metallic and non-conducting, making it hard for the sensor to detect it. Secondly, even though the sensor showed some promise when tested in isolation, once it was housed in the robot, it lost most of its capability due to interference from other metallic parts of the robot. Ideally, the sensor is capable of 'adapting' to a constant background e-field, but during play, collisions will almost surely occur and robot motion is bound to be jerky. These collisions and jerky motions set off false detections.

"Like inductive sensors, capacitive sensors employ a limited sensing range, in most cases 3 to 60 mm. Their traditionally rugged design allows them to be mounted very close to the monitored process. Due to their ability to detect most types of materials mounting considerations such as proximity to detectable materials other than the intended target, must be considered in order to avoid false triggering. For this reason, if the intended target contains a ferrous material, an inductive sensor is a more reliable option." http://www.clickautomation.com/products/index.php?func=list&cid=245,

Hence, in order to detect the ball, several issues need to be resolved. Firstly, a solution is needed to isolate the sensors from the internal environment of the robot. Secondly, the sensor's sensitivity needs to be increased. Thirdly, when the sensitivity is amplified, each object has its own 'signature' on the sensor's analog output and some higher level code needs to be written to distinguish the object from the sensor output. For next year, our recommendation is to visit www.qprox.com to investigate the capabilities of new products. Their latest proximity sensors - QT310 and QT300, released in late 2002 may solve the problem, but detailed testing remains to be done.

14.2 PC/104 AND PC/104 SINGLE BOARD COMPUTERS

14.2.1 Overview

The controlling unit of the robot can be viewed as the brain of the robotic system. It controls the entire robot and every single component has some sort of relation with the controlling unit. If we want to enhance the performance of the robots in all possible aspects, the controlling unit should definitely be improved.

In designing the new system for the 2003 robots, it was proposed that if we could move some of the computations to the robots, the system would perform better. The reason is instead of having the main computer handle all the computations and send the commands to the robots via the wireless system, the computations are done on the robots themselves. This way, the results are obtained faster and the robots will have more "intelligence". In addition, more computing power is required if we want to add more control algorithms and functionalities to the robots. In order for the robots to handle all the extra computations, a powerful CPU is needed. As a result, we decided to conduct research on the high-performance single-board computers, PC/104 and PC/104-plus. The single board computer will be an alternative to another design option using Motorola HCS12.

14.2.2 Introduction

PC/104 and PC/104-plus are single board computers that have high performance and are compact in size. (Please refer to Fig 15.1) They are stackable with the PCI and ISA extension bus (Fig 15.3) so they can expand easily without backplanes or card cages. It has low-power consumption and low-heat generation which is ideal for embedded systems such as our robots. It is fully PC compatible so applications are easy to develop. Detailed specifications can be found at *www.pc104.org*.

The PC/104-plus computer comes with a faster processor and has the ISA and PCI extension bus, whereas the regular PC/104 computer only has the ISA extension bus. We made the decision to use the PC/104-plus computer because we needed the stronger computing power. A PC/104-plus computer usually utilizes a Pentium 233MHz to 300MHz chip. Most regular PC/104 computers are not within the Pentium class and only have 100Mhz processor speeds. Since we would prefer more computations to be performed on the robot (such as the trajectory generation), we needed a Pentium class processor. The regular PC/104 computers would not have performed very much better than the current microcontroller.

Fig 14.1 Top view of PC/104-plus board

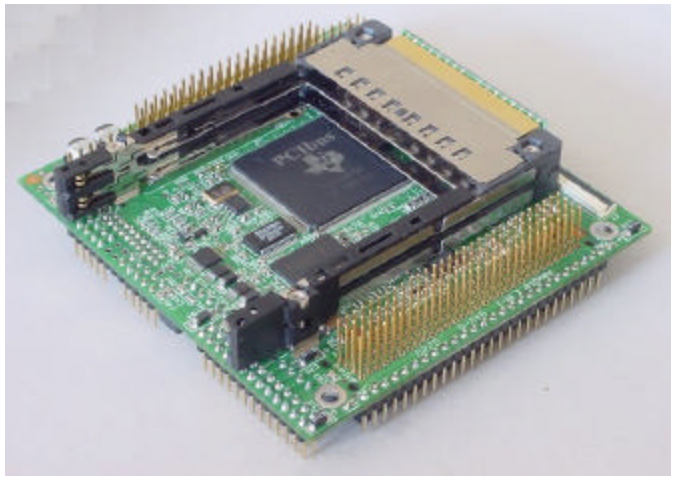


Fig 14.2 Bottom view of PC/104-plus board

14.2.3 Design goal

The design with the PC/104-plus computer causes most of the computations to be carried out on the robots themselves. All the trajectory computations will be moved to the PC/104-plus computer. The structure of the program modules will be different. There is only one single CPU as opposed to several microcontrollers in other design options. There is also no additional communication between the microcontrollers.

Using the PC/104-plus computer was an alternative design option for the 2003 robots. At the moment, we are still conducting research and developing the design. As such, it won't be used for the 2003 design. It will however, serve as a starting point for the 2004 robot designs.

14.2.4 Comparison

After we made the decision to use the PC/104-plus computer instead of the PC/104 computer, we made comparisons between several PC/104-plus single board computers on the market. The following are three boards that were considered.

Core Module P5E from Ampro Computers

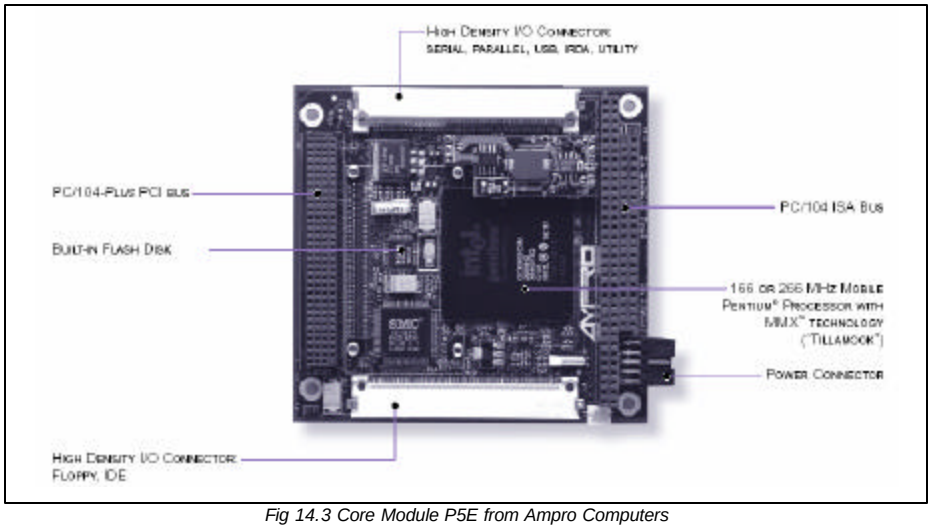


Fig 14.3 Core Module P5E from Ampro Computers

Relevant features

- 266MHz Intel Mobile Pentium processor
- Up to 256MB DRAM memory
- 3 programmable counter/timers
- 2 RS232C serial ports
- 8MB storage
- On-board Ethernet
- Windows CE compatible

Careful consideration was made to see which of the above three boards would be the best choice. PPM-TX from WinSystems is not compatible with Windows CE so it was rejected. (We plan to test and evaluate the PC/104-plus computer on the Windows CE platform so compatibility with Windows CE is required.) CoreModule P5E and CoolSpace Runner II both have all the features that we need and are good choices. However, CoolSpace Runner II is nearly twice as expensive as CoreModule P5E. Therefore, we decided to use CoreModule P5E from Ampro Computers.

PPM-TX from WinSystems

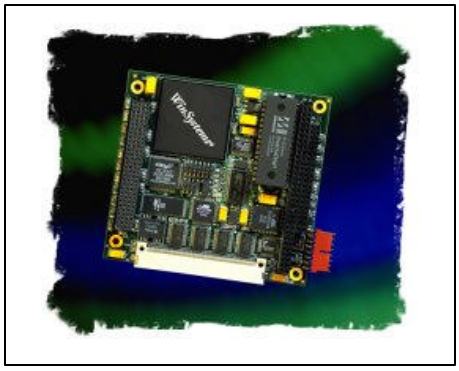


Fig 14.4 PPM-TX

Relevant Features

- 266MHz Intel Pentium CPU
- 32 to 128MB SDRAM
- 4 RS232 serial ports
- Ethernet
- NOT Windows CE compatible

CoolSpace Runner II from Lippert Inc.

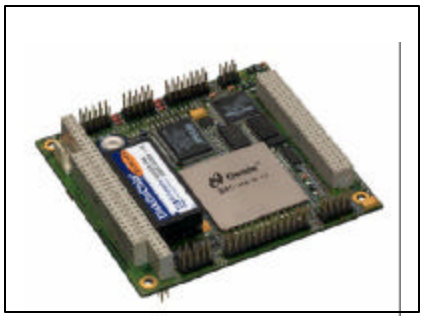


Fig 14.5 CoolSpace Runner II

Relevant Features:

- 300MHz processor
- 128M SDRAM memory
- Flash memory
- Ethernet
- Windows CE compatible

14.2.5 Interface Design

This is our current interface between the PC104-plus computer and the other components on the robot. The design of the interface for the PC104-plus computer is very different from the one used for the micro controllers. The PC104-plus computer is an off-the-shelf product; we do not have to make any modifications to it. To connect the components to the PC104-plus computer, the components are placed on the analog or digital board, and the boards communicate with the PC104-plus computer via its ISA interface.

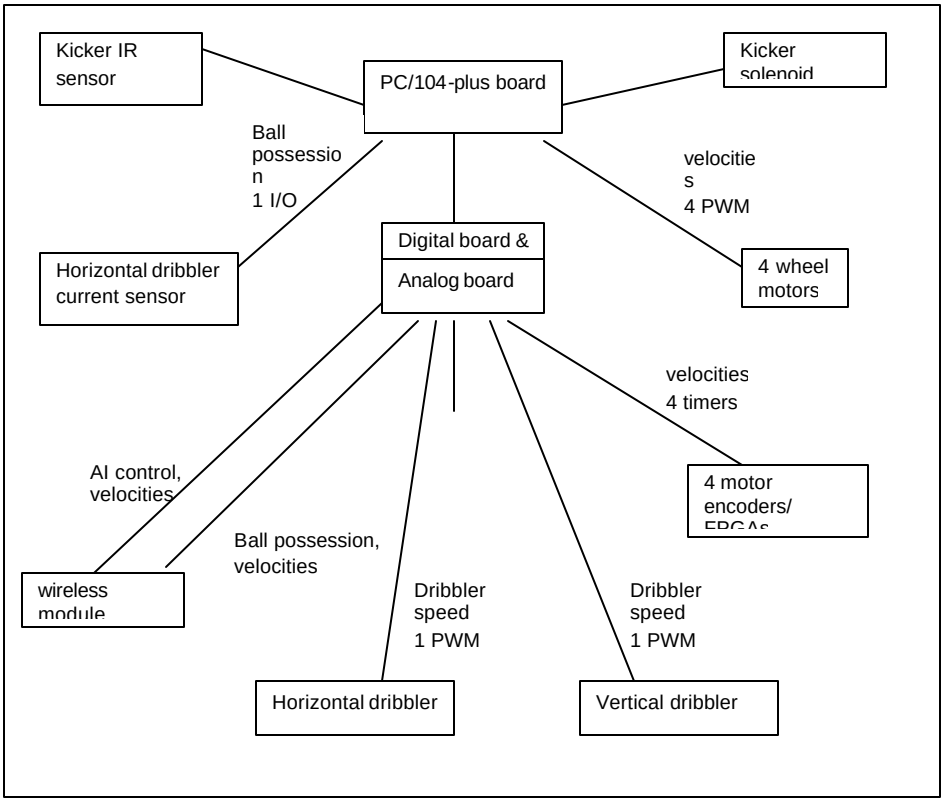


Fig 14.6 Interface Diagram

14.2.6 Digital & Analog Boards

For the PC104-plus computer to interact with all of the components on the robot, we need to use the analog and digital boards. The PC104-plus computer connects to the digital board, which is in turn connected to the analog board. The components are on the analog and digital board. The analog board connects to the wireless module.

The digital board is mainly the connection for the PC104-plus and the FPGA. The FPGA generates PWMs and processes the feedback from the motor encoders. The digital board has the following functions:

- 16 bit digital output

- 16 bit digital input
- 4 channel PWM
- 4 channel of 16 bit encoder up-down counter

The analog board holds components such as the H-bridges and motor connectors. It is very similar to the analog board for the microcontrollers. The analog board is connected to the digital board and the wireless module.

Here is the simple description of the operation: The CPU of the PC104-plus sends data to the FPGA on the digital board via the ISA interface. The FPGA then generates PWMs and sends them to the H-bridges of the analog board. The H-bridges drive the motors. The feedback from the motor encoders is sent to the FPGA and it determines the speed the robot is moving.

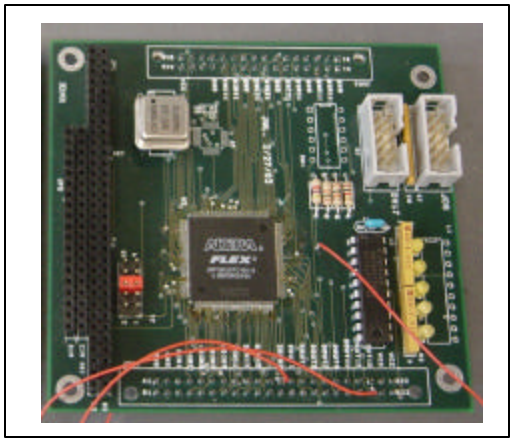


Fig 14.7 Digital board

14.3 Other components

14.3.1 Digital I/O board and Timer

Since the PC104-plus does not handle digital input/outputs and generate PWM signals like a microcontroller, we need a digital I/O board. At the beginning of the research, we searched for a digital I/O board that has timers and PWM outputs. The Quartz-MM Digital I/O and Timer Module is suitable. It has 16 digital I/O and the capability to output PWM signals.

Later we found that the digital I/O board will occupy too much space in the robot. Therefore, we decided to design a custom digital board and use a FPGA to handle digital input/outputs and generate PWM signals.

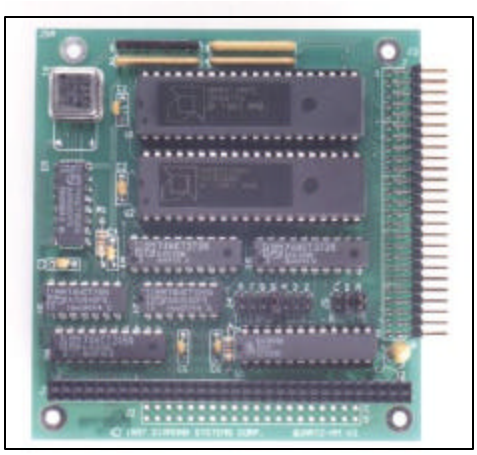


Fig 14.8 Quartz-MM Digital I/O Board

14.3.2 Connectors

To connect the PC/104-plus to the digital board, connectors are needed. PC/104-plus connectors are special configuration of 120 pins and 104 pins.

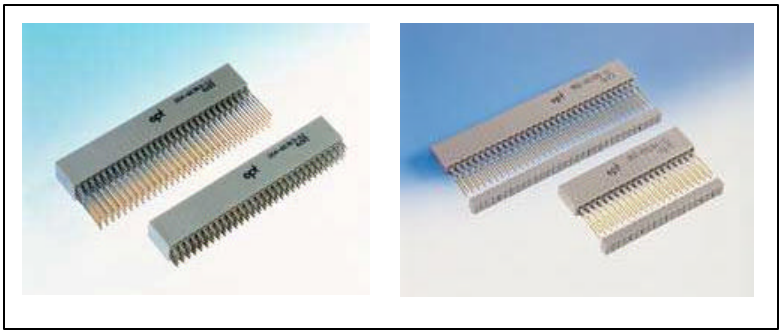


Fig 14.9 PC 104 Plus Connectors