Micromouse Handbook

Tak Auyeung, Ph.D.

June 10, 2005

Contents

Change log:

- TA 20030911 0838: add change log

0.1 Copyright Notice

Part I

Before Getting Started

Chapter 1

Who is likely to succeed?

The Micromouse design project is not suitable for everyone. From past experience, there are certain types of students who are more likely to succeed in a Micromouse project. This is, however, not to say students who are not of such types will not succeed.

1.1 Drive to "make it work"

The most important factor is the drive an individual to get something to work. This factor has been demonstrated, again and again, to be more important than intelligence and experience.

Note that there are different forces that drive students, and the drive to make something work is *different* from other drives. For example, the drive to get a good grade or "succeed" do not necessarily translate to success in this type of project.

Without a drive to make something work, a student is likely to get frustrated, procrestinate, give up or just leach on other team members as soon as any difficulty is encountered. I can guarantee that there will be many obstacles to get a Micromouse to work.

Do you have a drive to make a project work? You can ask yourself the following questions:

- –

1.2 Possession of an inquisitive Mind

Curiosity may kill a cat, but it is an important factor to get a Micromouse design to work. Although there are "traditional" designs that a team can follow, there will still be "mysterious" problems developed sooner or later. An inquisitive mind not only considers such problems as challenging puzzles to solve, but is also usually equipped with the necessary analytical and logical skills to tackle problems.

1.3 Ability to stay cool when things seem hopeless

From time to time, especially nearing Picnic Day, things may seem hopeless. You can avoid hopeless situations by not procrastinating.

Nonetheless, bad things happen. A robot may be dropped and it may break into pieces. A new "enhancement" turns out to break a lot of existing code. An upgrade of the compiler no longer compiles your program. Although you can avoid most of these situations by being careful and preparing for the worst, you can still, though rarely, get into hopeless situations.

For micromouse projects, it is important not to panic and stay cool. Panicing is not going to help anything, so you might as well try to remain calm and approach the problem methodologically. There will be pressure, but you need to focus on the work, and not the pressure.

1.4 Self Discipline

Because of the long span of time of a micromouse project, it is rather critical that as a collective, that a team has enough self discipline to work on the project as much as possible and as early as possible. The amount of work doesn't change. If you push the work to a later time, you simply end up with less time to do it. It is much better to get started early and work on the project diligently from the start.

This way, you may end up getting everything done weeks ahead of schedule. Guess what, now you can relax and enjoy your Spring Break while all the other groups try to rush their projects. Trust me, Having the option to sit back and relax *knowing* that you will have a good grade is a good feeling.

Chapter 2

Timeline

This chapter deals with the timeline of your project. I act more or less like a project manager in this class. I can remind you that you should be at a certain stage of the project, or that you should have reached a milestone. However, I do not crack the whip to make you meet the target timeline milestones. It is *your* grade that is on the line here, not mine.

2.1 Backward Timeline

Let's think about the project timeline from the target date of completion: Picnic Day. Then we work our way back to figure out at which point of time you should have what completed.

approx. date	done by then	what to work on next
Picnic Day (about two weeks into the Spring quarter)	This is show time! Your mouse has to solve the maze on Picnic Day according to IEEE region 6 rules to get an A in this class. You should have a fully functional micromouse with spare batteries, spare wheels and spare tires.	Work on the final report next. Your grade is still pending on a "satisfactory" final report for the project.
End of Spring Break	You have two weeks to Picnic Day. At this time, motion control sensor logic, exploration logic and the floodfill algorithm should be fully operational. You may need to tweak the software for better performance or improved reliability (so the mouse doesn't crash all by itself).	Work on incremental improvements from now to Picnic Day. Be sure to use version control so you can revert to a working version when a modification doesn't work (as well as you expect). Ideas you can explore include integrated turns, optimized speed runs and improved path-planning algorithms (using the floodfill result as a basis).

approx. date	done by then	what to work on next
Beginning of Spring Break	You have one week of Spring Break. There isn't much time left. At this point, you should have fully functional electronics, mechanical components and software components. The robot should be able to follow walls, explore/map the maze and solve the maze slowly (using the stop-for-every-cell approach).	Try to improve the performance by overlapping floodfill computation with motion so the robot does not need to stop for every cell, even during the exploration phase.
Mid Winter Quarter	You should have individual components working by now. The robot should be able to go straight without maze sensor feedback. The robot should be able to follow walls on either side. The robot should be able to map walls as it travels. The floodfill algorithm should work in simulation. The exploration algorithm should work in simulation.	Work on integration for the second half of Winter Quarter. In other words, make the exploration software communicate with the low level software to drive the robot. Integrate wall mapping logic into the motion control logic.
Beginning of Winter Quarter	You should have a chassis with motors and batteries mounted. The controller should be functional with preliminary control software to spin the wheels. Sensors, as components, should be specified. All interface circuits (for sensors, motors and etc.) should be specified and designed.	Implement and experiment with the sensors interface circuits. Redesign and modify as necessary. You may need to change or redesign the chassis. Construct sensor banks fixtures. Write low-level software to read sensors. Write motion control software for the motors. Write floodfill and exploration software with simulation.
Beginning of Winter Break	You should have motors, wheels and gearboxes specified. You should also have batteries specified. The controller board should be specified so its size is known. The basic design of chassis should be available at this point. The floodfill and exploration algorithm should be understood by now. Basic motion control theories should be understood by now.	Implement the first chassis based on the chassis design. Acquire mechanical parts, batteries and controller board. Begin to narrow down sensor choices (by small quantity orders and experiments).

approx. date	done by then	what to work on next
Mid Fall Quarter	You should have the overall design requirements available. Tasks should be assign to group members. The preliminary budget should be determined. The overall schedule of each group member all the way through Spring semester should be made available (for scheduling purposes).	Start to research drive components (motors, gearboxes, wheels and etc.) and the corresponding batteries. Research available and suitable controllers *and software development tools.* Study the floodfill and exploration algorithms. Write basic map storing and retrieval code. Write maze simulator. Study basic motion control logic. Research and acquire necessary tools.
Beginning of Fall Quarter	N/A	Form groups. Determine strategy and design philosophy. Research existing literature (mostly from the web) for mechanical design. Get familiarized with basic physics (force, mass, speed, acceleration, energy, power, etc.). Get familiarized with circuits like H-bridges, constant current control and etc. Study, study, study!

Part II

Robot Components

Chapter 3

Drive System

Needless to say, the drive system is important to a mobile robot. Afterall, a robot is mobile *because* it has a drive system!

A drive system consists of many components. The following is a logical list of such components, starting with the one that contacts ground:

- tire
- wheel
- axle, bearing
- gear (optional)
- motor
- motor drive circuit
- motor and motion control software

Before we explain each component, this chapter first explains some of the physical laws in driving a robot. Then, each component is discussed.

3.1 The Physics

How fast do you want the robot to go? How quickly do you want your robot to accelerate to its top speed? How much torque do you need?

Let us make our lives easy and assume linear acceleration.

In physics, $v(t) = v_0 + a \times t$, in which $v(t)$ is the velocity at time t, v_0 is the initial velocity in units of meters per second and a is acceleration in units of meters per second squared. The amount of force to accelerate your robot is $F = m \times a$, in which F is force in units of Newton (N) and m is mass in units of gram (g).

For example, if your robot has a mass of 1kg, you want the top speed to be 0.2ms^{-1}, and you want it accelerate to top speed in 0.5s, the required force is $F = 1\text{kg} \times \frac{0.2\text{ms}^{-1}}{0.5\text{S}} = 0.4\text{ms}^{-2} = 0.4\text{N}$. How much is a Newton (N)? A “pound” is the force of a 0.4545kg at one G. One G is 9.8ms^{-2}. Therefore, a “pound” is approximately 4.45N. In other words, 1N is about 0.22 pounds. That’s not much.

How can a robot’s wheels exert 0.4N? Assuming the robot is driven by two drive wheels, each wheel only needs to push at 0.2N at the axle. Intuitively, the large the wheel, the more torque it needs to deliver the same amount of force at the axle. Torque is $\tau = Fl$. If we use a wheel has a radius of 30mm, the required torque becomes $\tau = 0.2\text{N} \times 0.03\text{m} = 0.006\text{Nm}$.

Most stepper motors are specified by its detent and holding torque. The holding torque is easy to explain: this is the amount of torque to force the axle to turn one step when the motor is energized. The detent torque is the amount of torque that a motor has to turn the axle. *This* is what we need to know. Most stepper motors have both the holding torque and detent torque specified. Surplus motors, however, seldom has any specifications. You will need to purchase one and find out by testing.

In the case that torque is specified, it is often specified in oz-in (ounce-inch). How does an ounce-inch relate to newton-meter? $1\text{ozin} = \frac{1}{16}\text{lb}0.0254\text{m}$. 1 pound is 4.45N, so 1 oz-in is $\frac{4.45}{16}\text{N}0.0254\text{m}$, which computes to 0.00706Nm. In our example, we need 0.006Nm, which translates to less than 1 oz-in.

Some other times, torque is very incorrectly specified in g-cm (gram centimeter). The unit should have been g-G-cm (gram at one G centimeter). How does one 1 g-G-cm relate to N-m? It is $0.001\text{kg}9.8\text{ms}^{-2}0.01\text{m}$, which computes to 9.8×10^{-5}Nm. An oz-in is, therefore, 72.04 g-G-cm.

Note that our computation does not include any friction and angular momentum of the wheels. Furthermore, it assumes that the torque remains the same from stationary to full speed. In reality, a design needs a lot of margins. In this case, it is not unreasonable at all to specify stepper motors with at least 3 to 4 oz-in of torque.

3.2 Error Consideration

The drive system is prone to errors. Even if you use stepper motors, errors can still be introduced due to an imperfect maze floor (bumps and dust). Furthermore, the competition rules specify that the maze may be constructed with up to 5% of tolerance.

How much error is too much error? It all depends on the limits at which errors can still be corrected and your tolerance of risks. We will discuss errors and error correction in depth later. For now, having a *linear* resolution of 1mm to 2mm is a reasonable assumption.

3.3 Tire

For robots that travel on flat and finished surface, the best type of tires to use is the flat type. Very often, rubber bands used to hold bunches of brocolli work exceptionally well.

You can also try painted on anti-slip elastic material for inexpensive tires. O-rings work, too, but they are not flat and hence do not provide as much friction as flat surfaces.

Inner tubes of mountain bike tires also make excellent tires. If you have a “holey” tire around waiting to be dumped, cut it up into small sections about the size of the wheel. Compared to rubber bands, the rubber used in inner tires is considerably more durable.

If you insist to buy ‘professional’ tires, I suggest the type used for model airplane landing gears. These are generally thinner so your robot does not become too wide.

3.4 Wheel

You can easily make wheels from a bench drill and a hole cutter. Simply select a piece of wood of the right thickness (1/4” to 3/8”), then use a 2” to 2.25” hole saw to cut out a piece. It is important to use a bench drill if possible to make sure the cut edge is perpendicular to the surface.

Note that the same trick works on plastic as well. You need to select a hole saw with fine teeth to work on plastic.

Once you have a piece of circular material, you need to cut a groove so the tire does not slip out. If you have access to a lathe, great! Otherwise, you will need to construct a tool to do this.

Measure the diameter of the hole of the wheel. This should be the same as that of the drill bit on the hole saw. Go to a hardware store and find a bolt, washers and a matching nut. With washers on both sides of the wheel, insert the bolt and use the nut to tighten everything up.

Now insert the other end of the bolt into the drill press and tighten up the chuck. Now you have a simple lathe. Turn on the drill press and use a tool to cut the groove. A sharp chisel that is narrower than the groove works.

Of course, if you want to save yourself all this trouble (and fun!), you can use plastic or aluminum wheels for model airplanes.

3.5 Ball Transfer

If you have a differentially steered robot, it is useful to use a "ball transfer" as the third support point. A ball transfer is basically a ball that can rotate in any direction at any time. This is better than a wheel that swivels, especially for in-place turns. A wheel that swivels will skid and introduce errors.

You can purchase ball transfers from industrial places that make/sell ball transfers for transfer table applications. Such ball transfers are made from steel and can be quite large and heavy. They work well with a heavy robot, but lighter robots cannot even exert enough force to get the ball to rotate.

Another option is to buy some inexpensive plastic balls (from a local plastic supplier, such as TAPS in Sacramento, CA), and somehow hold the ball in with plastic material. One student of mine was very creative and purchased a pair of measuring spoons with half-sphere scoops that match the size of the plastic ball. In this case, make a hole to expose the plastic ball, use rubber band to hold the two spoons together and you have an inexpensive and light ball transfer!

3.6 Bearing, shaft, etc.

Without knowing your exact design, it is difficult to discuss bearing and shaft. However, there are generally two approaches to attach the wheel to the chassis.

The first approach involves a stationary shaft. This works well for slowly rotating wheels. This approach is easier to implement because the shaft can be a bolt fixed on the chassis by nuts and washers.

The second approach involves a rotating shaft that is mounted onto the chassis via sleeves or ball bearing. This approach is more common with faster rotating wheels. It is more difficult to implement.

3.7 Gearing

Unless you are using a motor with a gear head or a very strong and high resolution stepper motor, you probably need to gear the motor down to get the right speed and torque.

Gearing beyond 5:1 requires multiple gears. Although this *can* be done, it is generally difficult to be done right. As a result, you should select the motor properly so that you do not need to gear it down too much.

For a low gearing ratio, you can pretty much use any type of gear. Plastic gears tend to have less grinding and are generally less expensive.

One option is not to use gears at all. Instead, have the drive axle of the motor directly driving the wheel via friction. See the following figure.

In order for this approach to work, you need the tire to have good grip, and you need some force pushing the motor axle against the wheel.

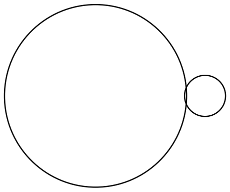


Figure 3.1: Direction Friction Drive

3.8 Motors

For smaller robots such as Micromice, there are mostly two types of motors: stepper motors and DC motors. This section explains how these two types of motors work and compare them.

3.8.1 Types of Stepper Motors

A general schematic of stepper motors is shown in figure 3.2.

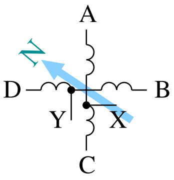


Figure 3.2: Stepper Motor Schematic

In this picture, the light blue "magnet" rotates at the center of the four coils. A, B, C, D, X and Y are electrical nodes that are connected to wires. There are different classes of stepper motors, depending on the configuration of point X and point Y in the figure.

- X and Y are both available and independent: most unipolar stepper motors are like this. This is the most general configuration as you can use bipolar or unipolar drive, with or without half stepping.
- X and Y are available but they are connected as one node: less common unipolar stepper motors are like this. This is much less general than the previous one. You can still use bipolar drive, but you cannot half step when the motor is bipolarly driven.
- X and Y are not available: this is the configuration for motors designed for bipolar drive.

By switching current through the coils, the freely rotating magnet follows the changing magnetic field and rotates. The magnet is connected to the drive axle of a stepper motor to deliver torque.

In unipolar full-step drive, only one coil is active an any time. In order to turn the magnet clockwise, the following current sequence can be repeated:

1. A→X
2. B→Y
3. C→X

4. D→Y

The beauty of unipolar drive is that the *direction* of current through the coils do not change. A controller merely needs to switch current on and off. Switching is easy when current direction does not change.

In bipolar full-step drive, points X and Y are not used. Instead, *two* coils in series are used at the same time. The current sequence to turn the magnet clockwise is as follows.

1. A→B
2. B→D
3. B→A
4. D→B

In this case, the controller needs to switch on and off as well as the *direction* of current. We will explain how to do this later. Why would anyone want to switch current direction when there is an easier solution (from the perspective of circuit design)?

It turns out bipolar drive ideally saves 50% energy while delivering the same amount of work. The strength of magnetic field is proportional to both the current and the number of winding of a coil. Let us use the following assumptions:

1. the resistance of each of the four coils is 30Ω
2. the number of winding of each of the four coils is 50
3. the supply voltage is 12V

In unipolar drive, the current is $\frac{12}{30} = 0.4$A, and there is a total of 50 winding for the active coil. In bipolar drive, the current is $\frac{12}{30+30} = 0.2$A, and there is a total of 50+50=100 winding. You can see that in bipolar drive, even though the current is dropped by 50% compared to unipolar drive, the number of winding is increased by 100%. The developed magnetic field is equally strong in both cases.

From the perspective of power consumption, unipolar drive consumes $12 \times \frac{12}{30} = 4.8$W, whereas bipolar drive consumes $12 \times \frac{12}{30+30} = 2.4$W. Hey, bipolar drive saves 50% energy!

3.8.2 How to Choose a Stepper Motor?

Stepper motors come in all sizes and resolutions. Even though our conceptual schematic of a stepper motor performs a full rotation in four steps, most steppers motors have higher resolution. Some stepper motors rotate 7.5°per step, others rotate as little as 0.9°per step. Some stepper motors are strong enough to control X-Y tables, others only need to move the head of a CD-player.

In general, stepper motors of smaller sizes (1 inch diameter and about 0.75 inch thickness) have larger step sizes and less torque, while larger ones (1.5 inch cubed or so) have finer step sizes and more torque. While it is not an absolute guideline, motors that consume about 1W to 3W are "about right" in terms of torque and power.

The decision between a larger, fine step and high torque motor and a smaller, large step and low torque one is up to you. A smaller and lighter motor makes the overall size smaller and the overall mass lighter. However, due to the lack of torque and resolution, you need to gear down the drive axle of a smaller motor.

A larger, high resolution and high torque stepper motor can be directly attached to a drive wheel. This approach simplifies the mechanical design, but it adds more mass and increases the overall size. Most other school use these high torque and high resolution stepper motors.

The bottom line can be summarized as follows:

- Do you have enough linear resolution?
- Do you have enough torque for acceleration and deceleration?
- Is it realistic to carry enough batteries to last 15 minutes?

Chapter 4

Energy System

4.1 Chemistries

If you go to an electronic store, there are plenty of batteries to choose from. The following is a table describing the different types of batteries and their characteristics.

4.2 Voltage Regulation

Two main components of a robot requires electrical power: the motors and the brain. Although motors are rated at certain voltages, such recommendations are often lose. This means using 13.2V on a stepper motor rated for 12V is unlikely to be any problem. The brain (logic) component, however, is not as flexible. If a logic component is rated at 5V, you cannot feed it 6V or 4V.

4.2.1 Logic Voltage Regulators

There are two popular voltage regulation approaches: linear voltage regulation and switching voltage regulation. Linear regulators are inexpensive and easy to set up. However, the efficiency of linear regulators is a function of the input voltage. For example, if the input voltage is 12V and the output voltage is 5V, the efficiency is only $\frac{5}{12}$. On the other hand, if the input voltage is 5.5V and the output voltage is 5V, the efficiency becomes $\frac{5}{5.5}$.

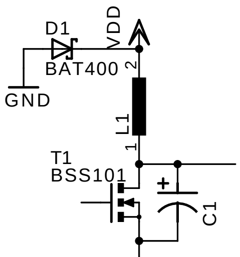
One common problem with linear regulators in the case of high input voltage is that the regulator itself has to dissipate a lot of power and gets hot. There is really no easy way to overcome this problem. One solution is to use switching regulators instead. Another solution is to use one battery for logic and another battery for motor.

4.2.2 Stepper Booster

In the case of stepper motors, it is *possible* to enhance the energy delivery to the motors so that a battery of a lower voltage can be used. The basic principle is the same as a boost switching regulator. However, because a motor does not require an exact voltage, much of the feedback and control of a boost regulator can be omitted.

Furthermore, the boost circuit can deliver more power when a step just occur, then taper off as time progresses to save energy. See the following schematic for this circuit.

This circuit uses L1 to store energy when T1 is turned on. When T1 is abruptly shut off, the positive node of C1 experiences as much potential as necessary to keep the current going as the magnetic field of L1 collapses. The output of this circuit connects to a coil of a stepper motor. Because the instantaneous current to a coil (another inductor) is zero, we need C1 to buffer the energy released by L1.


D1
VDD
BAT400
2
GND
L1
1
T1
BSS101
+
C1

Note that this circuit assumes all electrical charge of the capacitor will be discharged through the stepper motor coil before T1 is enabled. If this cannot be assumed, then a diode should be placed between the drain of T1 and the anode of C1, with the anode of the diode connected to the drain of T1.

By modulating the duty cycle as well as frequency of switching T1, the power delivered to the stepper motor can be software controlled. Note that an H-bridge is still needed for a bipolar driven stepper motor.

4.3 Energy Calculation

How big of a battery do you need? There are a few factors that determine the minimum size of battery that you will need.

Motors on a Micromouse consumes most of the power. For most micromice, each stepper typically consumes 1.5W to 3W on the average. For differential steering, two stepper motors consume up to 6W. By comparison, the logic components typically draw a total of about 80mA. Even at a supply voltage of 12V, the logic component only consumes 1W.

If we assume the whole robot consume 7W, it requires 6300J of energy to last 15 minutes. If we use a 12V battery, it must have a capacity of 6300J/12V=525Asec or 145.8mAH. This may imply that getting a battery of 150mAH is sufficient. Unfortunately, the discharge curve of a 150mAH will not sustain the required voltage for 15 minutes.

It is not an overkill to *double* or *triple* the capacity estimate. In addition to just making sure there is enough *energy* to last 15 minutes, a battery of high capacity can also *deliver* power better due to a lower internal resistance. Most NiMH or NiCd batteries are specified to discharge healthily at 1C. This means a battery with a capacitor of 150mAH can healthily discharge at 150mA. You must take this into consideration when choosing the capacity of your battery. In our current example, it is actually very reasonable to use a battery with a capacity of 500mAH (at 12V).

4.4 Charging and Discharging

Care should be taken when charging and discharging batteries to maintain safety as well as longevity of the batteries. Read the specifications of batteries to understand how they are supposed to be used.

As general rules, NiCd batteries can be trickle charged at C/30. “C” is the capacity specified in AH (amp-hour). This means a battery of 3AH capacity can be trickle charged at 100mA. A completely drained battery should be charged at C/30 for 40 to 45 hours before they are fully charged. There is no harm in trickle charging beyond 45 hours.

Some batteries, both NiCd and NiMH, are designed for “rapid charging”. Most newer NiMH batteries can be charged at 1C. This means a battery rated at 3AH can be charged at 3A. In rapid charging, charge duration should be used only as a safety measure. When charged at 1C, a timer circuit should turn off the charger in 1.2 hours. *In addition* to the charge period, one can also use the battery voltage as a hint as to when a battery is fully charged. Each *cell* peaks at about 1.55V and then *drops* to a lower voltage as it is being charged.

When a NiCd or NiMH battery is discharged, be sure not to overdischarge. When a cell in a battery is 1.1V or less (at average drain rate), the battery should be considered drained.

Chemistry	Strength	Weakness
Alkaline	Alkaline batteries are popular for small electronic devices and toys. These batteries have a high energy density and they are usually quite inexpensive when you guy "store brands".	Alkaline batteries have high internal resistance. This means even though they have a high energy-density, it is not possible to release energy at a fast rate. Furthermore, alkaline batteries discharge at a linear slope. This means the output voltage continues to drop linearly as the battery discharges. This characteristics affect the torque or motors severely. Most alkaline batteries are not designed to be rechargeable, making the long-run rather expensive.
NiCd	Nickel Cadium batteries represent the majority of rechargeable batteries. The internal resistance of NiCd batteries is small, making these batteries *dangerously* efficient at fast discharging. NiCd batteries can be trickle charged, which means a recharger can be inexpensively made. Being rechargeable, NiCd batteries are economical in the long run.	NiCd batteries have memory effect. This means when a battery is recharged without first completely discharged, the capacity of the battery will be limited. Furthermore, NiCd batteries have a relatively low energy-density. This means it takes *more* mass to store the same aount of energy compared to high energy-density batteries.
NiMH	Nickel Metal Hydride batteries share many advantages with NiCd batteries. In addition, NiMH batteries have twice the energy-density of NiCd batteries. For the same size and same mass, a NiMH battery stores twice the amount of energy. NiMH batteries also have low internal resistance and are rechargeable. NiMH batteries do not have memory effect, which means they do not need to be empty before being recharged.	*Unlike* NiCd batteries, NiMH batteries cannot be trickle charged. This is a minor issue because it is still easy to correctly charge NiMH batteries with a constant current source and a voltage monitor.
Li-ion	Lithium-ion batteries are used in many high-end cordless and rechargeable devices. The energy-density of Lithium-ion batteries is higher than that of NiMH batteries.	Although Li-ion batteries seem very suitable for robotics, they have some limitations. Li-ion batteries are not designed for continuous high drain rate. This is why cordless power tools do not use Li-ion batteries. It is also quite difficult to charge Li-ion batteries with home made circuits. Li-ion batteries require constant current charging for the first stage, then constant voltage should take over for the second stage.

Table 4.1: Battery Chemistry Comparison

Chapter 5

Chassis

The drive system and battery are the most bulky components of a robot. Once these two components are tested and determined, a chassis should be designed and produced.

5.1 Materials

There are many useable materials for a Micromouse chassis. When you are considering what material to use, you need to keep several factors in mind.

First of all, is this material strong enough? A Micromouse can collide with a wall under its own propulsion. You can assume a wall in a maze is solid. The next question is how fast is your robot traveling when it collides with a wall. In addition, you need to make sure the material you use is not excessively flexible because flexible materials promote vibration.

Secondly, is this material easy to work with? Although the Micromouse class has "traditionally" arranged with the Mechanical Engineering Department's machine shop, the general consensus is that it is quite difficult even to attend the mandatory orientation classes. It is better if you can work on the chassis at your own time and at a place that you choose.

Last but not least, is this material easy to acquire? Be warned that you may need to make several chasses before you are satisfied with one. Expensive or difficult-to-get material makes it difficult to prototype and refine the chassis.

5.1.1 Metal

A chassis made from sheet metal or machined metal is generally very strong. In fact, it is usually much stronger than necessary. Steel and aluminum are both very popular choices.

A metalic chassis often require quite specialized tools. Unless you have access to a milling machine otherwise, you will need access to the machine shop. If the chassis is not milled from a solid block of metal, you also need methods to assemble the various pieces.

5.1.2 Plastic

Plastic (acrylic) is gaining popularity in Micromouse design at U.C. Davis in the past few years. Although acrylic plastic is not as strong as sheet metal, it is strong enough for slower (that is, *most*) Micromice. The key to strength in a chassis made from sheet plastic is how the pieces are assembled.

Plastic is easy to work with. Most stores selling sheet plastic (such as Taps Plastic) also optionally cut sheets into specified dimensions. With a fine-tooth bandsaw, saw table or circular saw, you can also cut sheet plastic easily. You can also score and then "break" sheet plastic into predetermined sizes. Drilling plastic does require a little practice because plastic (especially acrylic) is brittle.

To put pieces of a plastic chassis together, you can use glue. You can use special cement for glueing acrylic, epoxy or even hot glue. For pieces with very clean cut edges, acrylic cement works best and looks best. For unevenly cut edges, epoxy and hot glue work better because they fill in the gaps.

One strange problem with plastic chasses is that the chasses tend to be too *light*. Yes, people need to add extra weight just to make sure the robot does not topple (to the front)!

5.1.3 Wood

Although wood may seem very old-fashioned and clumsy, it has characteristics that make it suitable for small robots such as Micromice.

First of all, wood has a natural self-dampening property. This is quite useful for designs with long sensor arms because bouncing and the consequent vibration can interfere with reflective sensor readings.

Wood is also easy to work with. For a Micromouse sized robot, even cutting and chiseling a chassis from a solid block is not infeasible! It is easy to assemble a wooden chassis from pieces because you can use glue or screws to hold various parts together.

Micromice utilizing ultrasound sensors should consider using wood for the chassis because of the dampening property. Otherwise, the ultrasound transducer will vibrate the entire metalic/plastic chassis and the sensor will only hear this self-induced vibration.

As for availability, you can find wood practically everywhere. I suggest the use of pressed and treated wood panels/blocks because of the increased density and strength.

5.2 Physics

Besides the size and shape of the components, another factor that determines the design of a chassis is physics. The following are almost always desireable:

- small overall size: this means less chance of getting caught and colliding with objects.
- smaller moment of inertia: two designs can have the same overall mass but one has a significantly smaller moment of inertia. A smaller moment of inertia means the robot can turn quicker and more accurately.
- lower center of mass: two designs can have the same overall mass, but one has a lower center of mass. A lower center of mass means more stability and less tendency to topple or roll-over.
- wheel size and placement: wheel size and placement can significantly

Chapter 6

Controller

A controller is also called an embedded controller, a microcontroller, an embedded computer, or a single-board computer. Regardless of the name, a controller (in this context) is simply a computer that satisfies the following requirements:

- has a processor that can execute programs
- has non-volatile memory to store programs
- has RAM to store data, RAM needs not be non-volatile
- has I/O peripherals to connect to robot related components and circuits
- the size is compatible with the robot being designed
- the power consumption is compatible with the robot being designed

This chapter discusses the selection of a controller for a robot.

6.1 Processing Requirements

Processing requirements are typically divided into processing power (measured in MIPS), program size (in kB) and data size (in kB). It is not always the case that all requirements increase at the same time. There are applications, for example, that requires a lot of processing power, but relatively little program size and data size.

For Micromouse applications, the requirements depend on the type of approach. A Micromouse using the floodfill algorithm and coded carefully to conserve data memory typically is about 20kB in program size and requires about 2kB of data. About 12MIPS (8-bit instructions) is more than sufficient. That is not very much! On the other hand, if more advanced optimization algorithms are used (such as Dijkstra's algorithm, C* or A*), much more processing power is needed. A* also has a high demand on data memory.

6.1.1 Processing Power

Processing power is typicaly measured in MIPS (million instructions per second) given the width of the ALU. The AVR Atmega128, for example, averages about 12MIPS with an 8-bit ALU.

How much processing power do you need? That's not an easy question to answer. One way to find out is to write the most processing intensive code first and run it in a simulator. You can, then, time the execution time and use that as a basis. The floodfill algorithm written in C completes in 20ms using the Atmega128 running at 16MHz. This is, indeed, fast enough for most practical purposes.

Don't forget that ISRs (interrupt service routines) also demand processing resources. If you use a high resolution encoder on a fast wheel, just the decoding logic may consume most of the available processing power. Because an ISR is typically not very complex, you can use a simulator to benchmark how much time it takes (on the average) for each invocation, then multiply that by the interrupt frequency to find out how much time the processor needs to process ISRs per second.

6.1.2 Program Size

Most controllers store programs in flash memory. Flash memory is relatively inexpensive, so a controller usually has enough program memory anyway. A typical Micromouse program using the floodfill algorithm only requires about 20kB of memory. The Atmel Atmega128 MCU provides 128kB of program memory, much more than sufficient.

Note that some controllers require programs be loaded into RAM before execution. Some others execute programs much faster once loaded into RAM. This is because most flash memory chips are slower to access compared to SRAM. In these cases, just having enough flash program space is not suffiient, you also need to make sure the program fits in SRAM along with data.

6.1.3 Data Size

Data size depends a lot on the application and algorithm. For a Micromouse using the floodfill algorithm, only about 2kB is required to store data. However, other algorithms can potentially require a lot more.

Data requirement is typically divided into three main categories. the first category are static variables. These variables (including arrays and large structures) are global and they occupy a static portion of RAM. The next category is the stack. Stack memory requirement depends mostly on the level of function call nesting and the use of large local variables. Recursive algorithms typically have somewhat unpredictable stack requirements, therefore they should be avoided. (All recursive algorithms can be translated to iterative ones.) The last category is the heap for dynamic memory allocation. While it is common for desktop applicatoin and system programs to rely on dynamically allocated memory, it is not a good idea with embedded applications. This is mostly because the memory requirement cannot be determined (in most cases).

As mentioned before, some controllers either require or prefer to load program into SRAM before execution. This means the total SRAM requirement depends on the data size as well as program size.

6.2 Size

The physical size of a controller can play an important role in smaller robots. For example, Mini-Sumo robots must fit in 10cm by 10cm, which makes the use of physically large controllers impossible. Micromouse robots do not have as much of a size limitation.

Note that a larger controller PCB means it flexes more readily and it weighs more. Both are not advantages. When given the choice, it is always better to choose a smaller controller.

6.3 Power Consumption

Power consumption depends on the processor as well as peripherals. With the Atmega128, power consumption is typically less than 30mA at 5V when the processor is running at 16MHz and all I/O devices are enabled. However, an AMD 188 running at 40MHz (with external flash and SRAM) easily requires more than 200mA at 5V.

Power consumption is important for two main reasons. First, a higher power consumption means more battery energy is required. Most robots require far more energy for the drive subsystem anyway. Second, a higher power consumption means the use of linear regulators may not be practical. This is because the power dissipation of a linear regulator is proportional to the voltage drop at the regulator and the current. Dropping from 12V to 5V and having a consumption of 200mA means a power dissipation of 1.4W, which requires heat sinking even with a TO-220 package.

6.4 I/O

6.5 Development Tools

6.6 Support and Repair

Part III

Common Robotics Theories

Chapter 7

Differential Steering

Differential steering is also called skid steering. This type of steering is used on tanks and other vehicles with treads. Wheelchairs are also steered by differential steering.

When a vehicle uses differential steering, it commands two wheels on opposing sides to turn at different speeds. For example, if the right wheel is faster than the left wheel, the vehicle steers to the left.

In this section, we'll examine some of the motion control functions that can be accomplished with differential steering.

7.1 Displacements

A differentially steered vehicle has two displacements. One is linear, and the other one is angular. The linear displacement controls how far the vehicle goes, while the angular displacement controls how much the vehicle turns. In this text, we'll use D_L for the linear displacement and D_A for the angular displacement.

D_L is the total number of steps that both motors need to perform. This is not difficult to understand. D_A, on the other hand, is a little more complicated. D_A is the total difference of steps between the two motors. In other words, if forward is positive for both wheels, D_A is the difference between the displacement of the right wheel and the displacement of the left wheel.

To keep track of D_L, every time either the right or the left wheel has a step forward, we add one to a counter. If a wheel has a step backward, we subtract one from the counter. To keep track of D_A, we assign positive to forward for one motor (we'll use left-forward), and assign negative to forward for the other motor. This means for each step the left wheel goes forward, it adds one to the counter. For each step the right wheel goes forward, it *subtracts* one from the counter.

7.2 Velocity

Just as there are two displacements, there are also two velocities. The linear velocity is v_L, whereas the angular velocity is v_A. For stepper motor designs, it is important to derive the velocity of each wheel. This is quite simple. The left motor velocity, using our convention (that left-forward is positive for angular), is $v_L + v_A$. This means the velocity of the right wheel is $v_L - v_A$.

7.3 Acceleration

You probably know by now that there are two accelerations, a_L and a_A. Note that for a robot, the maximum a_L is probably different from the maximum a_A. This is because the mass of a robot with respect to linear force available is often different from the angular momentum of a robot with respect to turning torque. As a quick example, increasing the length between the two wheels increases turning torque without changing forward force.

Once again, the acceleration for the left motor is $a_L + a_A$, whereas the acceleration for the right motor is $a_L - a_A$.

Chapter 8

Stepper Motor Control

One advantage of stepper motor based designs is that open loop control may be used. It is not uncommon that stepper based designs assume linear acceleration and deceleration. This simplifies the control logic significantly.

8.1 Displacement

Displacement (remaining distance to travel or remaining angle to turn) determines acceleration and velocity. Assuming the design has a maximum a_L and maximum a_A, we want to make use of such quantities whenever it is possible.

One great thing about using a stepper motor is that the speed profile can be precisely controled. This translates to precise displacement control. Assuming a linear acceleration and deceleration profile, the displacement is the area of the trapezoid bound by the speed profile in a speed versus time plot.

Let us assume the acceleration is a_{acc} and the deceleration is merely $-a_{acc}$. Let us also assume we have a displacement D to cover.

The first thing we need to determine is whether we need to accelerate to full speed or not. Let us assume the full speed is v_{max}. If the robot is to start from stationary and has to stop at D displacement, then the speed profile is either a triangle or a trapezoid.

The time needed to accelerate to full speed is $t = \frac{v_{max}}{a_{acc}}$. The displacement is then $D = \frac{t \times v_{max}}{2}$, which can be simplified to $D = \frac{v_{max}^2}{2a_{acc}}$. Assuming a symmetric deceleration, then $D = \frac{v_{max}^2}{a_{acc}}$.

If the desired displacement is greater than $\frac{v_{max}^2}{a_{acc}}$, then the profile needs a constant speed portion. Otherwise, if the displacement is smaller, then we may not need even the maximum speed.

If $D < \frac{v_{max}^2}{a_{acc}}$, then we can determine a top speed such at $v_{top} = \sqrt{\frac{D}{a_{acc}}}$.

In reality, however, we seldom need to predetermine the top speed. This is because the displacement can be adjusted so that the top speed needs to be recomputed anyway. If D is the remaining displacement to travel, we can use the following logic:

```
if $D > \frac{v^2}{2 \times a_{acc}}$ then
  if $v < v_{max}$ then
    if $a < a_{acc}$ then
      set $a = a_{acc}$
    end if
  else
    set $a = 0$
  end if
else
```

```
    if $a > -a_{acc}$ then
        set $a = -a_{acc}$
    end if
end if
```

8.2 Steering

Given D_L as the linear distance to travel and D_a as the angular displacement to travel, we can apply the previous algorithm independently to determine the acceleration. With the independent accelerations, v_L and v_a can be computed.

We then combine v_L and v_a to compute the actual velocity for the right motor and the left motor. In other words, each motor does not really have its own acceleration.

Note that for the angular displacement, we also need to take the sign into consideration. The modified code to handle signed quantities is as follows:

```
set $r = sign(v) \times \frac{v^2}{2 \times a_{max}}$
if $sign(r) \neq sign(d)$ then
    if $sign(d) \times v < v_{max}$ then
        set $a = a_{max} \times sign(d)$
    else
        set $a = 0$
    end if
else
    if $sign(v) \times r \leq sign(v) \times d$ then
        set $a = -sign(v) \times a_{max}$
    else
        if $sign(d) \times v < v_{max}$ then
            set $a = a_{max} \times sign(d)$
        else
            set $a = 0$
        end if
    end if
end if
$c_a = c_a + a$
if $c_a \geq f_{base}$ then
    $c_a = c_a - f_{base}$
    $v = v + sign(a)$
else if $c_a \leq -f_{base}$ then
    $c_a = c_a + f_{base}$
    $v = v + sign(a)$
end if
```

The previous code should be replicated so we end up with one for linear distance and one for angular distance. Then we combine the two and control the stepper motors. The following is the code for the left motor. Code for the right motor is almost the same, except the change $v_l + v_a$ to $v_l - v_a$.

```
$c_{ml} = c_{ml} + v_l + v_a$
if $c_{ml} \geq f_{base}$ then
    set $c_{ml} = c_{ml} - f_{base}$
    one step forward
else if $c_{ml} \leq f_{base}$ then
    set $c_{ml} = c_{ml} + f_{base}$
    one step backward
```

```
end if
```

8.3 Efficient Implementation

If you try to implement the program following the pseudocode, you will end up with a program that works, but not necessarily efficiently.

It is easy to determine whether two numbers have the same sign. The hint is to use the MSB (most significant bit) as the sign bit, then use bit-wise operations to help you determine whether two numbers have the same sign bit.

Multiplying a number by a sign is an expensive method to change the sign of a number. For example, recall the following code:

```
if $sign(d) \times v < v_{max}$ then
  set $a = a_{max} \times sign(d)$
else
  set $a = 0$
end if
```

This code can be implemented by

```
if $d < 0$ and $v > -v_{max}$ then
  set $a = -a_{max}$
else if $d \geq 0$ and $v < v_{max}$ then
  set $a = a_{max}$
else
  set $a = 0$
end if
```

The new implementation is a little less general and therefore longer. However, there is no multiplication involved. Negating constants such as v_{max} and a_{max} is easy and is done at compile time. Determining whether a number is negative or not is also easy in hardware (sign bit).

8.4 Measuring units

Measuring in units of steps is intuitive, but this is not the most effective method. When distance (displacement) is measured in steps, it is impossible to move with high precision. This is because the length of one cell is unlikely to be a whole number of steps.

As a result, it is better to represent distance as a fraction of a step. For example, let us assume a step is 2mm linearly. If we represent distance as a sixteenth of a step, then each unit is $\frac{1}{8}$mm. When a step is performed, the remaining displacement is decreased by 16.

Using this measuring unit, however, the length of a cell is no longer in whole units of 2mm. Instead, the length of a cell is measured in 0.125mm.

Note that the advantage can only be taken when we track the current location of the mouse in 0.125mm units. As a result, it may be better to change the algorithm so that we use $d_t - d_c$ as the remaining distance, where d_t is the target displacement, and d_c is the current displacement. This way, to trigger movement, we set d_t to the location of the next cell.

In other words, $d_t = n \times d_1$, in which n is an integer, and d_1 is the length of one cell measured in 0.125mm.

To avoid the stepper logic to "oscilate" when the remaining displacement is smaller than that of a step, the logic should consider it is on target when $|d_t - d_c| < 16$.

Chapter 9

Noise Related Problems and Solutions

Robots are inherently electrically noisy. Although Micromice are small robots, they are still noisy. Fortunately, most micromice designs do not require much analog circuits, which significantly limit the impact of noise to the designs.

Nonetheless, it is still important to know where noise comes from and how to limit it.

9.1 Sources of Electrical Noise

9.1.1 Basic Theories

Resistance of Wires/Traces

Because of resistance, there is a voltage drop across any trace or wire. This voltage drop is proportional to the amount of current. When there is a lot of current flowing from point A to point B, all the points in between will observe voltage differences when compared to both point A and point B.

Capacitance of Inputs

All high impedence inputs have some capacitance. Due to this capacitance, signals cannot change exactly instantly. In addition, due to capacitance, there is more current flowing between components as signals are switched.

Inductance

DC current flowing through any inductor has a tendency to maintain itself. Inductance basically determines how quickly current can change. When the resistance along a path with an inductor suddenly increases, inductance makes it impossible for the current to drop as suddenly. As a result, the current drops as a function of many factors, including inductance. Often, current drops slower than the increase of resistance. As a result, a *large* electrical potential is developed across the points where there is a sharp increase of resistance.

9.1.2 Digital Circuits

Digital circuits can cause noise on the VCC and GND signals. This is because digital circuits switch very fast. As a result, the resistance, capacitance and inductance of a trace/wire become significant enough to cause observable noise.

9.1.3 High Current Circuits

High current circuits causes voltage drop due to wire/trace resistance. In addition, fast switching high current circuits also have a lot of electromagnetic feedback due to inductance. This is *especially* the case for motors because motors are high inductance devices.

Even non-inductive high current circuits can cause problems. This is especially the case when the following conditions are met:

- The current is switched periodically. For example, a pulse width modulated heater element that draws a lot of current.
- The current is switched on and off. Again, like a pulse width modulated heater element.
- The current draw is enough to change the supply voltage.

When all of these conditions are met, the supply will see sharp changes of voltages (like a digital square wave). This is a problem for the voltage regulator because a voltage regulator cannot regulate the output voltage when the input voltage has a sharp change.

9.2 Methods to Reduce Noise

9.2.1 Basic Theories

Isolating the Logic Section

If the logic section (including the processor) resets because of a sudden drop of supply voltage when the motors are turned on, consider putting a diode between the supply and the input of the linear regulator. The diode acts as a switch so that if the input capacitor of the linear regulator has a higher voltage than the supply, the capacitor does not get drained to “help” the motors.

You should always take the diode forward voltage drop into consideration. Schottky Barrier diodes typically have a lower forward voltage drop at about 0.5V (100mA current). Take into consideration of the minimum input voltage of the linear regulator and you’ll know the minimum battery voltage.

Instead of only using a .1µF input capacitor, you can consider adding another 2µF capacitor (in parallel) to improve the isolation from the motor voltage.

The following figure shows how to use a diode for logic “stroke” protection.

Reducing Resistance

For high current paths, inceasing wire size or trace width will help reduce the resistance. Shortening the length of a trace or wire also helps. In extreme situations, you can add additional jumper wires to create more paths between nodes.

Decouping Capacitors

The sudden current demand of logic parts switching can be somewhat localized. Placing fast and low impendence capacitors across VCC and GND near logic components help provide some suspension. Typical values for ceramic decoupling capacitors are 0.01µF.

Fly-back Diodes

You can use fly-back diodes to create an alternate path for current to flow in a fast switching high-current circuit. However, you must use a *fast* diode that is rated for the amount of instantaneous current. Typical response time of *fast* diodes is between 4ns and 40ns. Unfortunately, response time is often reversely propertional to the amount of current a diode can handle.

For an H-bridge, you need four fly-back diodes per H-bridge.

Zener/Transient Voltage Suppressor Diodes

Fly back diodes are helpful for high inductance devices, but they do not solve all problems. This is mostly because the power supply cables and traces have inductance as well, which makes the resulting EMF global to everything connected to Vin and GND. Zener and TVS diodes become shunts to ground when a voltage higher than the rated standoff voltage is presented. Zener diodes are often sufficient for "clamping" excess voltage (mostly due to inductor related EMF). If response time is critical, TVS diodes are specially designed to handle fast and big spikes.

The following diagram demonstrates how to set up a zener or TVS diode for shunting excess voltage. Note that you must match the zener or TVS diode according to your supply voltage.

Inductor and Pi/Gamma Filter

When a switched high current device causes sharp changes to the input voltage to a regulator, you need to make this change smoother.

A pi filter is a circuit with two capacitors and an inductor. The inductor is inline in the supply voltage (between the input voltage and the input of the regulator. One capacitor is between ground and the input voltage, while the other one is between the input of the regulator and ground.

The first capacitor (between ground and input voltage) is for filtering noise going back to the supply. In our scenario, the noise is coming from the supply, so this capacitor is not needed.

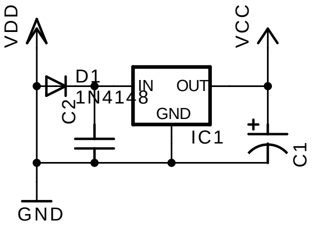
We just removed one "leg" of the symbol pi, making it look like gamma.

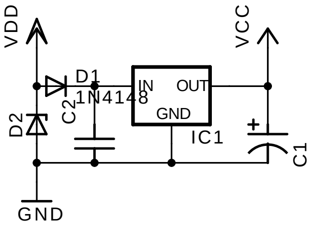
The purpose of the inductor is to stop any sudden increase or decrease of current. Note that an inductor is more effective at this than a capacitor. The capacitor between the input of the regulator and ground acts as an additional bumper to soak up additional shock energy of the sudden change.

As an analogy, the inductor is the size and mass of a tire, whereas the capacitor is the spring of a suspension.

Of course, the next logical question is, "how do we calculate the values of the capacitor and inductor?" The answer to this question depends on many factors, including the average current consumed by the regulator (load current), the magnitude of the input voltage step, the tolerance of the regulator and etc.

Instead of purchasing different inductors for testing, you can use the equation at http://home.earthlink.net/ jimlux/hv/wheeler.htm or http://www.ee.surrey.ac.uk/Workshop/advice/coils/air_coils.html. Air core inductors are nice because they are easy to make from insulated solid-core copper wires. Choose the thickness that suits your current requirements.


VDD
VCC
D1
1N4148
C2
IN
OUT
GND
IC1
+
C1
GND


VDD
D1
1N4148
C2
IN
OUT
GND
IC1
VCC
D2
C1
GND

Chapter 10

PID Loop

10.1 The Theory

The drive system of a robot is control system. In order for this system to behave predictably, it should be a *closed-loop* control system. This means some feedback should be used to adjust the output to the motors. This section discusses the theory of a well-known method of closed loop control: PID (proportional integral and differential).

For any control, there is a set point. The set point is what is the feedback *should* read. Let us use $s(t)$ to represent the set point (reference) at time t. The actual feedback, $f(t)$, however, if very unlikely to be at $s(t)$. As a result, there is an error term, which is simply the difference of the two, $e(t) = s(t) - f(t)$.

Based on the error term, a control system adjusts the output to the system.

In continuous terms, the output of a control system, $k(t)$ is expressed as follows:

$$k(t) = K_p e(t) + K_i \int_0^t e(x)dx + K_d \frac{de(t)}{dt} \tag{10.1}$$

10.1.1 The Proportional Term

The proportional term is merely a scaled error $e(t)$. This term makes perfect sense. When $e(t)$ is large, it means our set point is far away from the feedback. As a result, the output needs to be "very high" in order to bring the feedback closer to the set point. As our feedback gets closer, $e(t)$ becomes smaller, and it makes sense to reduce the output so we don't overshoot.

In a lossless system, just relying on $K_p e(t)$ should get us to the set point, eventually. However, because all practical systems are not lossless, $K_p e(t)$ never gets us to the set point. This is because at some point, the term $K_p e(t)$ is merely enough to counter the loss, and the system enters a steady state in which the input (from the heater or motor) cancels the output (heat loss or friction).

Despite the fact that $K_p e(t)$ is not sufficient by itself, it is the main component during the first phase of approaching the set point. Physically, $K_p e(t)$ is the bulk of the response to changes.

10.1.2 The Integration Term

The integration term, $K_i \int_0^t e(x)dx$ is the area between the set point and the feedback in a plot against time. As the proportional term $K_p e(t)$ "runs out of gas" as the feedback gets closer to the set point, the integration term $K_i \int_0^t e(x)dx$ continues to increase. As a result, the summation of $K_p e(t) + K_i \int_0^t e(x)dx$ should eventually become large enough so that any loss of the system is balanced by the contribution of $K_i \int_0^t e(x)dx$.

Note that that the integration term is not entirely independent from the proportional term. As the integration term helps to bring feedback closer to the set point, the proportional term gets even smaller. Eventually, in a tuned system, as the feedback becomes the same as the set point, the entire output comes from the integration term.

Physically, $K_i \int_0^t e(x)dx$ is the amount of output necessary to balance loss of the system. This term is very dependent on how the other two terms are set up, although it will, eventually, get the feedback to match the set point.

10.1.3 The Derivative Term

The derivative term, $K_d \frac{de(t)}{dt}$ only contributes when there is an abrupt change of $e(t)$. This change can be due to a change of set point, or it can be due to a change of feedback. A sudden change of feedback can be due to external factors such as a bump on the road.

This term is also called the dampening term because it dampens the effect of the other two terms. As $e(t)$ gets smaller, $K_d \frac{de(t)}{dt}$ is negative. In fact, for motor control, $K_d \frac{de(t)}{dt}$ is very negative to begin with because the response of a motor is stronger at lower speeds.

So why do we need the derivative term?

One use of the derivative term is so that we can crank up K_p. With a dampening term, we can adjust the other two terms so that the output does not "fizzle" out as the feedback gets closer to the set point. Note that $K_p e(t)$ fizzles out by itself anyway, but $K_i \int_0^t e(x)dx$ contributes more in time. Also, the response of a motor decreases as speed increases.

10.2 From Continuous to Discrete

In a discrete system, we can transform the previous equation to a discrete form (t is a natural number):

$$k(t) = K_p e(t) + K_d(e(t) - e(t-n)) + K_i \sum e(t) \tag{10.2}$$

Note that the division of n for the differential term is absorbed by K_d.

Why choose n instead of 1? To get a differential, we just need to compare the error of time t to the error at time $t-1$. This issue will be addressed later.

It should be noted that the sum term should be reset every time the reference is changed. This is because the integration of errors should be particular to a particular reference.

10.3 Practical Concerns

The main equation is simple, now comes the gritty details.

10.3.1 Resolution of Output

First, there is the resolution of the output. Since most robots are controlled by pulse-width-modulation (PWM), the output is an adjustment of the duty cycle of the PWM signal. The real question is, then, the resolution of PWM.

Obviously, a resolution of 2 (1 bit) is not sufficient. That would be the same as on-off control. On the other hand, a resolution of 1024 (10 bits) is too much because most mechanical systems have errors that exceed 0.1%. For a mechanical system that has an inherent error of 1%, 128 divisions for the PWM is enough. We will use r_k to represent the number of bits to represent the output.

10.3.2 The Resolution of $e(t)$ and K_p

The proportional term is the main component of the output of a PID loop. As a result, K_p is an important coefficient, as is the error term itself.

We need to determine the resolution of the error term, $e(t)$. Let us consider a system where the speed can be up to $x$ticks/s. This means the error term can be up to $2x$ because the motor can be spinning full speed backward and ask to spin full speed forward.

The next factor is how often to run the PID loop logic. The frequency of the loop f helps to determine the resolution of the error term. Although the actual error can be up to $2x$ticks/s, between each PID invocation, the difference only be up to $\frac{2x}{f}$. For example, if the top speed of a reversible system is 1200 ticks/s, and the PID loop is invoked at 50Hz, $e(t)$ can only range from -48 to 48, which can be represented by a 7-bit signed integer.

The number of bit required for $e(t)$ is, then,

$$r_e = \frac{\log \frac{4x}{f}}{\log 2} \tag{10.3}$$

When r_e is close to r_k, the magnitude of K_p is small. This means you need to rely on fractions in K_p to fine tune the terms. Instead of using floating point numbers, many small MCUs benefit from the use of more efficient integer/fixed number computations.

For example, even if we know that K_p does not exceed 4, we can use 8 bits to represent it. The least significant 6 bits become fractional terms. Given an integral bit pattern of 10110101_2, the actual represented value is $\frac{101101012}{2^6}$. The division by 2^6 is inexpensively performed by right shift operations.

10.3.3 The Resolution of the Integration Term

The integration term is important because it allows a system to reach the reference. Without the integration term, a PID loop can never achieve the reference signal.

The integral term relies on the summation of errors. This sum can become much larger than the error term itself. It is advisable to put limits on this term (caps). The range of the integral term should be about 20 times that of the range of the error term. Due to the magnitude of the error term, there is no problem with resolution in the integration term.

Because the integration is a large number to start with, the coefficient K_i must be a small fraction. This is easily done by assuming the binary point to the to the left of the most significant bit of the binary integer representing K_i. The trouble, however, is that we now have a large number multiplied by another large number. This can take a bit of time for an 8-bit system. As a result, some control systems choose to ignore some of the less significant digits to speed up computation.

10.3.4 The Resolution of the Differential Term

The differential term is the *change* of error. While this term can have the same magnitude as the error term itself, most of the time it is very small (one tenth the magnitude of the error term itself or smaller). In a continuous system, this is not a problem because there is *infinite* precision. However, in a digital system, this poses the problem of digitizing the contribution from the differential term.

Instead of just relying on the number of ticks between PID invocations, we can also rely in the period between ticks. Although period is proportional to the inverse of speed, it is still more accurante compared to the number of ticks at slow speeds.

Consider a realistic case where a system expects about 1000 ticks per second. This translated to one tick of millisecond. However, as the system slows down to rely on friction to come to a complete stop, the speed can be down to a small number (such as one) of ticks per second. It becomes impossible to know the actual speed. Using a timer, however, we can track the period between ticks and use the period to deduce the actual speed.

If we consider the worst case of a 16 count timer for a 50Hz PID invocation frequency, the lowest frequency is 50Hz, while the next lowest frequency (due to digitization) is $\frac{1}{15/16*20\text{ms}}$, which translates to 53.3Hz. This resolution is better than 7%, even when the tick count per time method fails. While 7% may not sound like a lot, it makes a lot of difference when there is nothing lese to rely on.

10.4 Improving Feedback Resolution

10.4.1 The Problem

The feedback and set point are both in units of "ticks per PID period" up to this point. This is okay for most cases, assuming the encoding has sufficient resolution. However, for robots with low resolution encoders, and also for low-speed situations, just counting ticks between PID invocations produces coarse and quantized results.

Let us assume the PID control loop is invoked at 20Hz (20 times per second). Let us also assume the encoder disk has 1000 transitions per revolution. At a very low speed, such as 0.1 revolutions per second, we only get 100 ticks per second. This is divided by 20 because the sampling frequency is 20Hz. As a result, we only have, on the average, 5 ticks per period.

The error term is going to be even more quantized because it is the difference between the set point and the feedback. As a result, the control signal has very poor resolution because it depends on the error term.

10.4.2 A Solution

A solution is to compute the period between ticks, and use its reciprocal as the speed. This method is not suitable when there is sufficient encoding resolution because reciprocal is difficult to compute (compared to counting ticks), and the resolution of the period-oriented approach drops as speed increases. However, at lower speeds, this period-oriented method yields much better resolution than tick counting.

In our previous example, using the tick count method, 0.1r/s cannot be distinguished from 0.09r/s because both yields about 5 ticks per PID invocation. However, using the period measurement, 0.1r/s results in a period of 10ms, whereas 0.09r/s results in a period of 11.1ms. Even with a timing resolution of 1ms, we can still distinguish 0.09r/s from 0.1r/s.

Note that timing resolution can be improved by hardware peripherals. The ATMega128, for example, has an "input capture" device associated with timer 1. It can record the duration of a pulse at resolutions down to single clock periods.

10.4.3 Practical Considerations

Although we can improve the resolution of low speed encoding, the benefit may not be useful because of other problems. For example, due to the nature of DC motors, they are not smooth at lower speeds. As a result, it may not be practically useful to use period measurement to improve the encoding resolution at lower speeds.

Chapter 11

Logic Interface

Because the stepper motor logic is mostly interrupt driven, but the AI is continuous, we need an interface design to let interrupt driven logic communicate with "background" logic. This chapter discusses and compares some common methods.

11.1 General Interrupt Service Routine Philosophy

The general idea is to keep the execution of ISRs as short as possible. This is because we want to minimize the duration in which interrupt is disabled. This approach ensure the latency of any interrupt remains short.

Note that some architectures allow interrupt be reenabled in an ISR. This is not recommended because stacked ISR invocation can be very tricky, and it leads to unnecessary problems (such as reentrancy issues, stack space issues and etc.)

The "propoer" solution is to use a simple run-time kernel. Threads can wait for a signal to continue, and ISRs can generate the signals to let the threads continue execution. This type of design is most suitable to relatively infrequent interrupts.

Without a run-time kernel, it may be necessary to prolong the execution of an ISR because decoupling is more difficult. In this case, it may necessary to reenable interrupt within an ISR. To reduce latency, interrupt should be reenabled as soon as the cause of the interrupt is handled. All subsequent processing can execute with interrupt enabled.

11.2 Asynchronous and Continuous Code Interface

Asynchronous code can be the ISRs themselves, or threads triggered by ISRs. Regardless, the execution of such code is hardware triggered. Continuous code, on the other hand, executes in the background. The interface between these two types of code can be difficult.

One of the easiest way is to let continuous code check bits that are set by asynchronous code. This method is wasteful because of the busy wait nature.

A more elegant solution is to use threading.

Part IV

Micromouse

Chapter 12

Project Management

12.1 The Necessity

Project management, or there lack of, can make or break a project. This is the case regardless of the expertise level of participants. It is, therefore, extremely important to understand project management.

Without proper project management, resources are not fully utilized, conflicts arise among team members and nothing gets done.

12.2 Resource Requirements

A team must first understand the resource requirements of a project. In Micromouse projects, there are several types of resources.

12.2.1 Finance

Micromouse projects *can* be expensive (US$1000 and more), although they can also be quite inexpensive (between US$200 to US$250). The actual expense depends on your design philosophy and expertise. More expertise tend to lower the cost because materials and tools are better utilized.

A typical project will cost about $200 to $500 if tools and materials are not readily available. A team must have all members agreeing to a budget before starting on a project. Otherwise, if a team runs out of budget in the middle of a project and have members disagree on how to proceed, it can potentially kill the project and turn members against each other.

12.2.2 Tools and Materials

If members of a team already have access to the necessary tools and materials, the actual expense can be lowered considerably. Some of the tools/materials include the following:

- temperature controlled soldering station
- a PCB holder for soldering
- soldering supplies (solder, flux, cleaning pads,...)
- general tools: electric drill, electric saw, vise, files
- inspection tools: magnifying glass, loupe
- test equipment: DMM, DMM probes, osciloscope probes, jumper wires

- programming tools: notebook computers (low-end ones are okay), inexpensive desktops (to be left in the lab)
- misc. items: binders for manuals, etc.

12.2.3 Time

This is the *most* important resource of all. Even given the best team members and the best tools, a team still need time to complete a project. Although EEC194 is listed as a 5-unit, 3-quarter class, the actual requirement exceeds the units.

1 unit of lab translates to 3 hours of actual lab time. 1 unit of lecture translates to 1 hour of lecture and 2 hours of preparation and homework. In other words, you should spend something like 6 hours on the Micromouse projects every week for the Fall and Spring quarters.

I understand you are taking other classes as well. However, if the team is to get the project done, each member *must* allocate sufficient time for the Micromouse project.

In order to aide the allocation of time, a team should have a fixed weekly meeting time or several fixed sessions. All members of a team should attend the meetings as well as the lectures and labs. Since the lecture/lab will take about 2 hours per week, each team should allocate somewhere between 2 to 4 hours for team meetings.

12.3 Personnel Issues

Every year, at least one or even two teams break down completely due to personnel issues. This is bad for everyone since no work is done and I cannot have an excuse to assign a reasonable grade. Although it is not possible (or nearly so) to change personalities, it is possible to use certain means to reduce the frequency and intensity of personnel issues.

12.3.1 Regular Meetings

Besides the class meeting time, set up another regular meeting time for the team. Make it mandatory for all members to show up at all class meeting and team meeting times. Yes, there will be times when certain members do not have assignment. However, these members can still contribute to the project as a whole. For example, the person responsible for mechanical design can spot and diagnose a software bug in the wall mapping software.

Experience has taught me the value of a role sheet for each the meetings. In other words, state who attend each meeting. You can even go to the extent of stating the arrival time and departure time of each member. This role sheet can serve important purposes in the future if there is a dispute of who did what and who did not do what.

Also, have one member keep track of the discussions in the meetings. This is important because it is easy to forget the action items and conclusions in a meeting. A tape recording is fine, but reviewing/replaying a tape can be more troublesome than just reviewing points in a plain text file.

12.3.2 Focus on the Tasks

A lot of times, focusing on the tasks instead of attitudes helps to resolve some personnel issues. Focus on what needs to be done, and how to get it done. Of course, it only takes one member in a team to have an attitude problem to drag everyone else into a fight.

If members of a team cannot focus on the tasks, please inform me as early as possible.

12.4 Task Management

A project can usually be divided into smaller tasks. It is a science to manage tasks, but the principles are very simple. First of all, figure out the dependency of tasks. Next, schedule tasks so that they follow the dependencies. Last, parallelize tasks if possible and make the most efficient use of available resources.

The following is a list of general tasks and dependencies.

1. Understand the competition and rules of the competition.

2. Understand the overall design of micromouse robots, no details are needed. This depends on 1.

3. Search for sources of components and materials. Do not get fixated on any particular models. Just get a general feel and get catalogs of available components. Do this after 2.

4. Determine/Specify the requirements of a micromouse robot. You can do this as soon as 1 is completed.

5. Design multiple alternative designs based on the information collected in 3. Rule out certain designs based on the requirements determined in 4

6. Among the alternatives in 5, select the *best* design.

7. After `taskSelectBest`, acquire the necessary components.

8. Test each component acquired in 7. This is particularly important for components that are not well specified (such as surplus items).

9. You may need to repeat 5, 6, 7 and 8 if a component does not meet the requirements.

10. Assemble the micromouse using the tested components from 8.

11. Test the assembled unit from 10. You probably need to refine the design a little to make it work better.

It is not obvious from this list that multiple threads can proceed at the same time. For example, even before motors are acquired and tested, the processor board can be assembled and tested. This assumes, of course, the type of motor is already determined (DC versus stepper) because the type of motor does have an impact on processing resource requirements.

12.5 Task Forces

I suggest each group be divided into smaller task forces. I know, we are already talking about groups of 5 or fewer people. Nonetheless, task forces are still useful for such small groups.

Task forces can be permanent or temporary. For example, when the drive system and chassis are being designed, a temporary task force for the overall mechanical design is useful. Another example is a temporary task force that researches what embedded controllers are available and evaluate different options.

At a later stage, a tuning task force can help the fine tuning of the robot. This task force may or may not include the programming members because the programmers are probably busy enough already.

12.6 Communication

Project management requires communication. It is not a question of hours, it is a question of effectiveness. Keep meeting minutes if possible, or record each meeting so important points are not easily forgotten. Have your own mailing list to make sending email to all members easy.

The bottom line is that everyone should be on the same page. Everyone should know what the current task(s) is/are and when each task should be done. Everyone should understand what others are doing in case changes become necessary.

Now, you don't have to tell everyone about every line that is changed in software. However, if your change affects other people, you have to notify people who will be affected. Version control is complex all by itself. However, you can (properly) use some tools to help make that less complex. I suggest the use of CVS for version control. This software can be run as a server so multiple developers can access the repository concurrently. It also allows non-conflicting changes to the same file. Best of all, it is robust and free!

Save your email messages so you can refer to them later. Make a folder so that Micromouse email does not mess up your main email folder.

12.7 Timeline

While progress is usually not predictable and is subject to availability of time from other classes, home assignments and exams, one can use a basic time line to estimate what should be done by what time. The following is just an estimated scheduled based on successful teams in the past. Q4 is the Fall quarter, Q1 is the Winter quarter of the following year, and Q2 is the Spring quarter of the following year.

- **First four weeks of Q4.** Form teams, distribute tasks and duties among members. Submit overall specs. in terms of target mouse characteristics. Focus on lectures and reading materials.

- **Second four weeks of Q4.** Analyze design requirements, evaluate alternative parts, select and specify parts. Define interfaces among components. Submit design analysis and justification. Get high level AI software working, specify controller board. Acquire parts.

- **Third four weeks of Q4.** Assemble mechanical components, including chassis, motors, gears, wheels and etc. A working chassis should be available during the Winter break with all components mounted (maybe except for the sensors).

- **First four weeks of Q1.** Have all first generation components completed. Test each component individually and redesign/reimplement if the design does not meet the initial specifications. Get ready for integration.

- **Second four weeks of Q1.** Motion control should be done, robot should be able to perform hard coded blind motions involving linear and angular combinations. All electronic components should be designed and produced. Sensors should be tested. Controller board should be complete.

- **Third four weeks of Q1.** Integrate components. High level AI software, mid-level motion control and sensor software should be fully integrated with the low level interrupt based engine. Designs based on DC motors should have PID loop tuned. Robot should now be tested in a real maze (maybe shrunk to 4x4) to see if it solves the maze.

- **First week of Q2.** Yellow alert state. Iron out all the known problems, robots should solve mazes relatively reliably.

- **Second week of Q2.** Red alert state. Last week before Picnic Day to fine tune and optimize the robots. Make sure spare parts are available. Calibrate and test robots with the actual maze board the night before Picnic Day. Sleep well and look sharp on Picnic Day!

- **Second two weeks of Q2.** If a robot does not solve the maze on Picnic Day, use these two weeks to make it work to get a B. Work on final report. Working robots may compete in Region 6 competition.

- **Second four weeks of Q2.** If a robot does not work during the first four weeks of Q2, get it working by the 8th week to get a C.

- **Third four weeks of Q2.** Finish a draft of the final report and submit it to me for review. Submit the final report by the last day of finals.

Chapter 13

The Competition

This chapter describes the Micromouse competition and interpretation of the rules. You should always double check with an officer of the local student chapter of IEEE in case of doubt.

For an original unabridged version, please refer to http://www.ece.ucdavis.edu/ ieee/umouse/rules.shtml. This section only emphasizes on less obvious and confusing items of the original rules.

13.1 Specification for the Maze

Note the sentences

> The coating on the top and sides of the walls should be selected to reflect infrared light and the coating on the floor shall absorb it.

All material reflect *some* infrared light. So this sentence, when interpreted in the most general sense, does not say anything. What should have written is the minimum reflectance (as a ratio) of the top and side or walls and the maximum reflectance of the floor.

Given this sentence and the common interpretation, it means the walls (top and side) reflect significantly more infrared light compared to the floor. Although no frequency is specified, you can assume it is near infrared, which goes from 880nm to 980nm.

Also, note the following sentence,

> The dimensions of the maze shall be accurate to within 5% or 2cm, whichever is less.

This is very *important*! This means your mouse has to handle errors of the maze itself. The statement does not say whether it is 5% to each side of the specified dimension or 5% as a total. I am assuming it is a total.

This still means the height of walls can range from 4.875cm to 5.125cm. You have to assume the variance can exist in the same maze.

You should design your robot so that dimension sensitive parts are easily adjustable in the field, or use some software technique to compensate for the error factor.

13.2 Specifications for the Micro Mouse

This section is relatively simple and clear. The only thing worth mentioning is that "the geometry during a run shall never be greater than 25cm by 25cm." This statement does not preclude a robot from transforming itself during a run as long as the overall geometry always fits within a 25cm by 25cm box.

13.3 Rules for the Contest

This section is the most interesting one. The HTML version of the rule is somewhat "garbled" in equations. The handicapped time should have been $R + \frac{S}{30} - B$, in which R is the time taken for the run, S is the start time of the run, and B is 10 seconds if the robot has remained untouched up to the end of the run.

This equation favors autonomous robots that do not require *any* intervention. Also, note that a run time is "the time it takes for a Micromouse to travel from the start square to the destination square." This means manually moving the robot from the destination back to the start square is worth it if it will take 10×30 or 5 minutes for the robot to autonomously travel back to the first square.

The time limit is 15 minutes. Because of the start time component in the handicapped time, later runs are generally less likely to take less handicapped time even if the paths are more optimized.

The following statement is a little value:

> The starting procedure of the Micromouse shall not offer a choice of strategies to the handler.

Obviously, you cannot *flip* switches when you are string the robot for a competition. However, this does not mean you cannot have switches and alternative strategies. You just have to choose the strategies *before* the maze is unveiled.

In addition, the robot can autonomously choose the next strategy based on what it has learned.

The rules explicitly say that the illumination, temperature and humidity of the competition site can be changed "at the discretion of the contest officials". This means don't count on it. Make you robot so it can handle a tough environment (it'll still be in door and somewhat comfortable to people).

Note that if you replace parts of the robot, you may be requested to erase memory of the maze. This means don't let your robot run out of battery in the middle of a competition.

There is *no* rule regarding memory of the maze when a robot is manually placed to the start square because it "appears to be malfunctioning". "Malfunctioning" typically includes crashing into a wall. This means you can let the robot retain most of the map (except for the last cell or two right before it crashed).

In order to observe the rule that says no strategy offering at start up, you need to design the logic of the mouse such that it clears the maze once when it is powered up (which is not a choice). Ever since this powering up, the robot retains the map even when it crashes. You use one single button to instruct the robot to begin from the start square again after it is picked up. In fact, the robot should stop when the button is pushed, and only restart after the button is released. This gives the handler time to place it in the maze.

If you decide to retain the map after a crash, you have to be careful about the last few walls entered into the map. They can be wrong! If in doubt, erase more walls. All useful algorithms are greedy, which means even if the robot assumes fewer walls then there really are, it will still be able to explore and eventually figure out the shortest path.

Chapter 14

Errors

Nothing is perfect, you can count on that. Even if you try your best to make your robot as precise as possible, there will be errors that you cannot control. For example, the floor of the maze can have pot-holes and spots where traction is low (due to dust). The maze can have up to 5% tolerance according to the rules.

Fortunately, if you prepare the robot for errors, most errors can be corrected before they are not recoverable. This chapter discusses various sources of errors and how such errors can be detected and corrected.

14.1 Sources of Errors

If you use DC motors, motion control can be tricky and introduces errors by itself. Even if you use stepper motors, you can still have various types of errors due to the environment.

14.1.1 Wheel Errors

Depending on the manufacturing quality of your wheels, you can get errors from the wheels.

If a wheel is not perfectly round, the linear speed of the robot varies even as the wheel turns at a perfectly constant rate. If a robot uses differential steering, uneven speed translates to turning (slightly) and therefore changing direction (slowly). The robot will travel in an 'S' shaped line.

Even if a wheel is perfectly round, you can still experience uneven linear speed as the wheel turns at a constant angular speed. This is because the axle may not be at the center of the wheel. This also causes the robot to travel in an 'S' shaped line instead of a straight line.

If the two wheels of a differentially steered robot are different in size, the robot will always skew to one side (the side with the smaller wheel).

14.1.2 Stepper Motor Errors

Stepper motors can skip steps when they operate close to the maximum operation speed under load. A small bump on the maze floor can easily shock a motor so that the resulting torque causes the motor to skip at least one step.

14.1.3 Traction Related Errors

When the tire of a wheel travels over the maze floor, it can encounter spots where traction is low. Stepper motors inherently move the robot in "steps". Therefore, the loss of traction can easily make the tire skid on slipery spots.

When the robot is turning, wide tires result in errors because the traction surface is large. For turning, the ideal contact surface should be a single point. This is because a wide tire contacts the surface in a line, and different points of this line (as concentric circles around the turning pivot) want to travel at different speeds. However, because the wheel is driven by a single axle, all points of the tire surface must travel at the same speed. This means *some* parts of the contact surface have to skid, and skidding always introduce errors.

Note that with traction related errors, a robot may travel more than what it intends, or it may travel less than what it intends. This is because you get errors during acceleration (which shortens the actual distance), and you also get errors during deceleration (which lengthens the actual distance).

14.1.4 Maze Tolerance

A maze can have up to 5% error. This means over the length of a 12-by-12 maze, there can be up to about 5cm of error. 5cm may not sound like a whole lot, but if a robot stops 5cm short of the destination and it makes a turn, the resulting path will *start* with a 5cm offset error, which is a lot of error to correct.

14.2 Reducing Errors

Instead of just correcting errors as they occur, you should also try to minimize errors to start with. Here are some suggestions that help reduce the amount of errors to start with so that any remaining errors can be corrected more easily.

14.2.1 Tire Width

For racing, wider is better. However, for precision purposes, narrow tires perform better. This is especially the case when a robot is turning, and Micromice turn *a lot*.

On the other hand, the width tires should still be sufficient to provide enough traction. The shape of tires also has impact on precision. O-ring type of tires has the least contact surface, therefore it provides the most precision. Flat tires (rubber bands) have more contact surface and are therefore less precise.

If your tires have contact widths of more than 5mm, they are probably too wide.

14.2.2 Wheel Roundness

If you purchase your wheels, there is little that you an do.

If you manufacture your own wheels, there is an easy way to make your wheels round. Use a screw that matches the wheel's hole size, two large washers and a nut to attach the wheel to a power drill or drill table. Then fix the power drill onto a clamp or some other attachment surface. As the drill turns, use a chisel to work on the wheel. Only use a corner of the chisel to begin the process, and never apply too much force. As long as the process is gradual, the resulting wheel is always round because of the rotation of the drill.

14.2.3 Wheel Center

Again, if you buy your wheels, there is little you can do to fix off centered axle holes.

If you make your own wheels, use the techniques of the previous section to ensure the axle hole is centered.

14.2.4 Traction

Any type of rubbery material has lots of traction. However, some are more durable than others. Rubber bands are cheap (even free!), but they do not last very long. Inner tubes of bicyle tires, on the other hand, are made from durable rubber. If you have a flat inner tube (of a mountain bike), cut it in small sections and fit the sections over the wheels.

An alternative is to use paint-on grippy materials. You can purchase these materials in cans from most home improvement stores (usually in the paint section). If you want to go with this approach, be patient and paint in layers. Trying to paint one single thick layer usually ends up with uneven coating and affects the roundness of the finished product.

Silicone and latex sealants (also called caulks) also have lots of traction. Unfortunately, these materials tend to be very thick. It is difficult to apply these materials onto the wheels evenly. Again, one trick is to use an attached power drill as a lathe to help evenly apply sealant. Unless you need sealant for other purposes, you can purchase the more expensive pressurized cans for easier application. You can also purchase the toothpaste-like tubes so you don't need use a caulk gun. Try to look for sealants that "remains permanently flexible" so you got both traction and suspension.

14.3 Detecting Errors

The previous section discusses method to passively reduce errors. This section introduces methods to actively detect errors.

14.3.1 Length to Sensors

The more distance between the sensors and the center of rotation, the more sensitive the sensors are to angular errors. This is nothing more than geometry. For example, if individual sensors are 6mm apart, and the length between the sensor banks and the center of rotation is 200mm, then the sensitivity of angular errors is $\arctan\frac{90}{200} - \arctan\frac{96}{200}$, which is $24.23 - 25.64 = -1.41$ degrees. Assuming the same basic design but a shorter distance of 100mm between the sensor banks and the center of rotation, the sensitivity becomes $\arctan\frac{90}{100} - \arctan\frac{96}{100}$ which evaluates to $41.99 - 43.83 = -1.84$ degrees.

14.3.2 Sensor Placement (Lookdown Sensors)

Instead of placing the "middle" sensor right in the middle of where the top of a wall should be, you can use two sensors right on the edge of the top of a wall. This design allows more sensitivity to offset and skew errors.

14.3.3 Distance Sensors

Instead of using discrete sensors to sense the distance to a wall, you can consider using an analog distance sensor like the Sharp GP2D12. An obvious advantage is that the distance measurement is no longer discrete, the resolution is limited only by electrical noise and the ADC (10-bit ADCs are built-in to the ATMega128).

Another advantage that has nothing to do with error detection is that you don't need sensor banks hanging in front of the robot if you use these distance sensors.

A disadvantage is that you cannot see beyond a wall. With look-down sensors hanging from wings that look over walls, a robot can map immediate walls on the other side of walls surrounding the robot.

14.4 Error Toloerance and Correction

The error tolerance has to do with how much error can a robot make before it cannot recover. Error correction is the procedure to utilize detected errors and fix them.

14.4.1 Tolerance

It is difficult to describe error tolerance in general. However, some tolerances are easy to compute.

Skew tolerance

How much angular error can a robot make before it cannot recover or crash into a wall?

Obviously, when the sensors can no longer detect the top of a wall (when there is one), there is too much skew error. However, the actual skew tolerance has more to do with how much space a robot needs to correct skew errors. When the robot is stationary, space (distance) is not a problem because differentially steered robots require no distance to rotate.

However, when a robot is traveling at a certain speed, it will take some distance before the error is corrected. Unlike offset errors, skew errors *continue* to accumulate offset until it is corrected.

The real limiting factor is how fast the motors can change speeds to steer a robot. The steering needs to factor in maximum linear acceleration/deceleration as well as angular acceleration.

There may be a closed mathematical form for this tolerance computation. However, I am not a mathematician. Instead, you can use a simulator to help you figure out exactly how much skew error your robot can suffer.

Parallel Offset

Parallel offset errors refer to the distances from where a robot is supposed to stop and make an in-place turn. Note that parallel offset errors can combine with skew errors at blind turns.

Perpendicular Offset

Perpendicular offset is the error distance of the center of the robot to the center of its path.

14.4.2 Correction

Error correction relies on hints provided by sensors. Although a robot may be equipped with simple look-down sensors, it can still sense a lot of clues to help correct errors before it is too late.

Position of Top of Wall

If you use look-down sensors, the sensors can indicate the position of the top of the wall relative to the sensor banks. If the sensor bank is placed ahead of the robot, both skew and perpendicular offset errors can yield similar positions.

Fortunately, the correction is the same regardless. With sensor banks in front of the robot, skewing to the left has similar sensor readings as perpendicularly offsetting to the left. The correction is steering right for both cases.

When a robot turns (especially in an in-place turn), the top of wall position can be used to help the robot determine when to decelerate and even "patch" angular errors. As a robot turns, some sensor will first detect a wall. The robot should slow down the turn because it should know the number of steps to perform.

Top of Wall Transitions

Parallel walls an appear and disappear as a robot travels. The transitions can be used for a robot to resynchronize its location in the maze. Even a path with just pegs will provide sufficient clues to resynchronize. The worst configuration is a straight "corridor" with just contiguous walls and no perpendicular walls even on the other side of the walls.

Chapter 15

Sensors

Sensors are extremely important for any robot. We can talk about sensors in general, but in an application like Micromouse, we can customize the discussion so the sensors are optimized for this application.

15.1 Purposes

The sensors on a micromouse serve several purposes. The first purpose, regardless of the exploration algorithm and approach, is to make sure the robot can detect errors. No robot is perfect, no maze is 100% made to specification. Errors are a way of life. Sensitive sensors that can detect slight errors can help the robot correct errors before errors accumulate to a level that can no longer be corrected (and the robot crashes into a wall).

The second purpose of sensors on a micromouse is to map the maze. Of course, this purpose only makes sense for smart mice. Wall huggers do not need to really map and remember the walls. For this second purpose, sensors that can sense further is better. This makes it possible for a robot to find walls that are not immediately around it. Sensors that can sense far walls minimize robot movement during the exploration phase and save time.

15.2 Sensor Types

There are many types of sensors that you can use. I am dividing the sensors based on *approach* instead of *implementation*.

15.2.1 Range Finding

Range finding sensors are very useful because they can typically sense walls that are not immediately around the robot. As discussed earlier, this reduces robot movement during the exploration phase. On the other hand, range finding sensors, at one point or another, must rely on the accuracy of some analog circuit. This can introduce noise and uncertainly to the error detection mapping logic.

IR Reflective Intensity

This is the simplest of all range finding sensors. An IrED shines IR to a potential wall, and a phototransistor picks up the intensity of the reflected radiation. In theory, the intensity is the inverse of distance squared.

In practice, this type of approach is only useable for very close range applications. First of all, different types of “white” paint may not reflect the same amount of IR! If a maze is painted with different types of white paint, calibration becomes impossible.

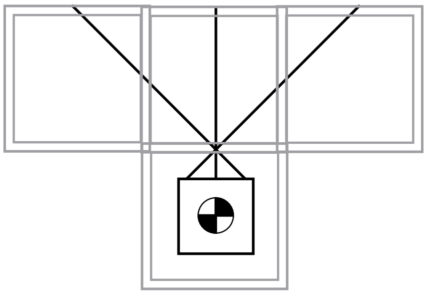


Figure 15.1: Configuration of IR rangefinding sensors

In addition, micromice are suppposed to work in a normal environment with ambient lighting. If hot lamps are used, they also produce IR. How can a robot distinguish IR from background lighting versus its own IrEDs? Well, you can use modulation to screen out some of the ambient light. However, this does get somewhat ugly in terms of signal to noise ratio.

IR Triangulation

In this approach, an IrED is also used to shine a *focused* beam of modulated IR. Unlike the previous approach, however, the sensor is not a single phototransistor. Typically, the sensor is a linear array of CMOS or charged-couple devices. Due to the triangulation angular difference of reflection at different distances, different individual sensor cells sees the illuminated spot. This is then converted either to a digial or an analog signal as output.

This approach is not prone to varying reflectance of surfaces because it is not intensity-based. Furthermore, this type of sensor is self-contained with the lens and sensors. You do need to provide either a digital or an analog interface to read the distance, however. Furthermore, most of these sensors are consistent but not calibrated. This means even though a particular sensors provides the same reading at the same distance all the time, it may not be the same reading as another sensor.

Interpreting the results of IR triangulation is not straightforward. The distance is not linear to the output of the sensor. A translation table is usually needed if you need to know the distance. Even when a distance is available, you still need to know *which* wall reflects the IR based on the distance. Be prepared to spend some time on sensor interpretation logic if you are using IR triangulation sensors.

You *can* use this type of sensor for both error correction and mapping. For error correction, you should consider arranging the sensors so that they are not at 0, 90 and 270 degrees from the direction of motion. Instead, you need two sensors that sense at an angle so they can sense the side of walls that are ahead of the robot. Refer to figure 15.1 for illustration.

The typical range of this type of sensor is from 100mm to 10000mm. This is a very useful range for micromouse applications. In addition, the precision (based on triangulation) is much better at shorter distances, which means such sensors are useful for both error correction *and* mapping.

This type of sensor is not cheap. An analog sensor typically costs $8 to $15. However, one sensor can replace many individual sensors in other types of designs.

Ultrasound Time-of-Flight

Polaroid has used ultrasound sensors for auto-focus applications for many years. You *can* use the same type of sensors in micromice. Micromouse mazes are made from materials that reflect enough ultrasound for range finding purposes.

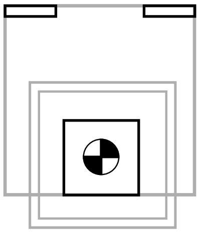


Figure 15.2: Configuration of IR reflective sensors

Ultrasound does have a few difficulties compared to IR. First of all, it takes a lot energy to produce enough ultrasound for ranging purposes. Your battery system must be capable of supplying bursts of high current. Ultrasound production is difficult to focuse, which makes the out going cone kind of large. This means it is difficult to tell *what* reflected the detected ultrasound.

Ultrasound sensors are not cheap, either. They are typically more expensive than IR triangulation type sensors.

15.2.2 Existence

Existence sensors are binary. Each can tell whether a wall exists or not. You have to use many existence sensors for error correction purposes because you need a sensor to track each potential location that the wall may appear. On the other hand, existence sensors are cheap and easy to interface with. They are also very easy to interpret in software.

Typically, existence sensors can only detect walls that are immediately around the robot. You can use the entire allow dimension of a micromouse to maximize the mapping capability.

IR Reflective

In an IR reflective existence sensor, an IrED shines IR to a location where a wall may exist, then a phototransistor picks up any reflected IR (if a wall does exist). IR reflective existence sensors are also called look-down sensors because they are typically placed on “wings” that glide over the top of walls.

This type of sensors is the oldest type and also the most commonly used due to they simplicity. They are prone to problems such as vibration (of the wing) because reflective sensors are optimized for detecting reflection at a certain distance. If the sensors are too far or too close to a surface, they may not register reflectance.

Reflective sensors *can* be used to detect walls that are not immediately around the robot. By maximizing the width of the sensor wings, reflective sensors can sense walls on the other side that are perpendicular to the direction of motion. Refer to 15.2 for a typical configuration of IR reflective existence sensors.

Reflective sensors are also very helpful in rotation error detection because they can be used to align the direction of the robot during a turn.

It is probably a good idea to at least prepare a robot to use this type of sensors as a plan B. Even if you have ideas of using much better sensors, it is still safer to make sure the robot can switch to use this type of sensors in case things do not work out as expected.

IR Interrupt

In an IR interrupt existence sensor, the photo transistor is placed on wings that glide over the top of walls, while the IrED is typically placed at the lower part of the main robot chassis. This way, if a wall exists, it blocks the light path and the robot knows a wall exists.

The best feature of an interrupt existence sensor is the robustness. It is immune to vibration of the sensor banks. Even better, an interrupt sensor has much more time to catch a wall compared to a reflective sensor. This is because a reflective sensor passes the top of a wall (perpendicular to direction of motion) quickly and only has a few samples to it. An interrupt sensor, however, remains interrupted as long as the wall exists.

Interrupt sensors are not with their problems, however. Reflective sensors are easy to interpret because they sense the existence of a wall immediately under them. Interrupt sensors, however, sense the existence of a wall that simple blocks the light path. It takes some more logic to translate the sensor readings to the location of walls.

Interrupt sensors are best suited for detecting a front wall because they are robust for walls that are perpendicular to the direction of motion. Interrupt sensors are typically useless for detection walls that are not immediately around the robot.

Chapter 16

Sensing

Different sensors require different sensing strategies. However, there are certain issues that apply to all types of sensors. This chapter discusses methods to read and interpret common types of sensors.

16.1 Common Issues

Regardless of the type of sensor, there are common questions to answer. This section deals with these questions. The next section answers some of these questions for commonly used sensors.

16.1.1 Failure Rate

Failure rate is often measured as mean-time-between-failure. A failure can be one that can be recovered, or one that is permanent. Note that in this context, failure is conditional to the fact that "components are operated in an environment that is well within the acceptable operating parameters." In other words, we are not subjecting components to extreme heat and other harmful environments.

Most electronic components either work, or they have some kind of manufacturing defects that cause the component to fail. Most manufacturers have in-house test procedures that ensure the functionality of components before such components are shipped.

Sometimes, it is necessary to operate a component outside the "safe parameters". For example, an IrED may be rated for 20mA of continuous forward current. The same IrED can probably be pulsed at 50μs using a 50ms pulse period using a 1A pulse current. However, most IrED manufacturers do not test IrEDs using the 1A pulse current method. As a result, IrEDs that pass the manufacturer's test (20mA continuous) may not work when pulsed with 1A, even at a low duty cycle.

If you are operating a component outside the "safe parameters", you may need to perform your own tests. For example, you may find that only 75% of all IrEDs can survive the pulsed current test. How you design your own "burn in" test depends on the type of component as well as how you plan to use it.

16.1.2 Repeatability

Repeatability has nothing to do with accuracy. In other words, a sensor that has superb repeatability may have very poor accuracy. Since we are not in the business of precision scientific apparatus, repeatability is often sufficient.

Repeatability is the ability of a component to repeat producing the same result under the same conditions. With an object placed at 30cm, does a distance sensor *always* produce a voltage of 1.62V? If you collect samples of the voltage, you can extract statistics from the samples.

A high repeatable sensor produces samples that have a small variance. This is a very useful feature because it means you can take very few reading, or just one, in the application. A sensor with poor repeatability needs many samples, or you have to operate with a lot of uncertainty.

How about "digital" sensors, such as look-down sensors? Even though the end result is boolean, these sensors are inherently analog. In other words, you can still measure the output voltage of a divider bridge using a phototransistor using a fixed source of IR illumination. Fortunately, most phototransistors are very repeatable.

16.1.3 Accuracy

Accuracy only applies to sensors that are supposed to be calibrated. For calibrated sensors of a particular make/model, the output is supposed to relate to some physical quantities *regardless* of the individual sensor. For example, if a type of temperature sensor is supposed to generate pulses that are proportional to temperature (say 1ms per ^oK), all samples of this type of sensor should output a pulse of 273ms in water that is about to freeze.

Unless you are using a calibrated sensor (such as one of those expensive laser-based time-of-flight distance sensors), there is no issue of accuracy.

For example, the GP2D12 sensors are uncalibrated distance sensors. Each individual one is almost guaranteed to output a different voltage at the same distance from an object. This is fine because the manufacturer never claims any consistent relationship between the output voltage and distance across samples.

Uncalibrated sensors must be calibrated before they can be used to measure any real physical quantity.

16.1.4 Digital versus Analog

Some sensors, such as look down sensors, are typically used as binary (digital) sensors. Others, such as the GP2D12 sensors, are analog sensors. A binary digital sensor is one that, from the perspective of software, only produces results of 0 and 1. An analog sensor, on the other hand, produces values that has a range (wider than 0 to 1).

Note that even though phototransistors are analog sensors that has a different $\mathrm{I_{CE}}$ depending on the amount of IR energy irradiating the die. However, the use of a comparator or schmidt trigger effectively converts the voltage measured from a voltage divider into just 0 and 1.

Binary sensors are easy to interpret, but they do not provide a lot of information, either. On the other hand, analog sensors provide more information, but it requires techniques to fully utilize such extra information. For example, traditionally, lookdown sensors are considered binary. This means either a sensor senses a wall underneath, or it does not sense a wall. Treating a lookdown sensor as an analog sensor (using an ADC), on the other hand, can differentiate a sensor that is entirely outside a wall from one that is right on the boundary. This extra information translates to higher resolution for the sensor bank location, which in return translates to better error correction behavior.

16.1.5 Sample Time

Sample time is the entire period of time from the instance a sensor captures (transduces) a physical quantity to the time that software is made known aware of it. Many factors affect the sample time of a sensor. The sensor itself, conditioning circuits and the processor each plays a part in the determination of sample time.

Short sample time is always desireable. This is because a robot is likely to be in motion during sample time. This means by the time software reads the value, the robot is no longer at the same location, or the physical quantity to measure has already changed.

16.2 Using Common Sensor Types

16.2.1 Phototransistor-IrED Pairs

Phototransistor-IrED pairs are very commonly used, yet they are often under utilized.

First, most application use a continuous current to energize the IrED, although the sampling of the phototransistor is fairly infrequent. Although the phototransistor-IrED pair does have a lag time due to physical characteristics, the lag time is often in low tens of microseconds. This means the IrED only needs to be switched on about 50μs before the phototransistor is sampled.

Let us consider a situation in which a sensor is read 100 times per second. If the IrED is energized for only 50μs for each sample, the power consumption is reduced to $\frac{50\mu s}{10ms=0.5\%}$ of that of the continuously energized design.

Another trick commonly done to phototransistor-IrED pairs is to connect IrEDs in series. Because each IrED only has a forward voltage drop of 1.7V (at 20mA), two can be connected in series when powered by a source of 5V. Obviously, the current limiting resistor value must be adjusted. Given a voltage source of 12V, up to 6 IrEDs can be connected in series.

Be careful not to connect too many IrEDs in series. Although *some* current will flow at a forward voltage of less than 1.7V, the amount is too small to emit any infrared.

The last, but most important, trick is to use a high pulse current to energize the IrEDs. Because the pulsing technique reduces the duty cycle of the on-time, we can now use a larger current and stay within the average power dissipation parameters. This turns a dimly lit light into a strobe. It helps to improve the signal-to-noise ratio (most noise comes from background lighting), and reduce the value of the pull-up resistor (to improve recovery time). The one drawback of this approach is the loss of repeatability. The pulsing circuit introduces uncertainty because it is difficult to guarantee a constant current during a pulse.

16.2.2 Reflective Triangulated Ranging Sensors

The Sharp GP2D12 sensor is great for measuring distances between 10cm and 90cm. It outputs a voltage that relates to the distance. However, the application of such sensors requires attention to a few details.

First, the GP2D12 sensor internally uses modulated (pulsed) IrED. It is important to use a low ESR (equivalent series resistance) 10μF resistor to decouple the power supply. Without such a decoupling resistor placed close to the sensor, the main regulated supply can see enough fluctuation to trigger a reset.

The GP2D12 sensor is not calibrated. In other words, different GP2D12 sensors produce slightly different voltages for objects at the same distance. An application must be provision to calibrate the sensors.

One of the main drawback of the GP2D12 sensor is the long sampling time. Although most inexpensive ADCs need a little bit of time to perform successive approximation, it is the sensor itself that contributes most to the sampling time. The GP2D12 sensor only updates the output every 20ms or so. This means for 20ms, the output voltage remains the same. Worse yet, there is no external line to indicate when the output voltage is updated.

This means in software, a robot never knows whether it is reading a freshly updated output, or an output that is just about to be updated. This means the value acquired by the robot can be up to about 40ms old! A moderately fast robot at 300mm/s moves about 12mm in 40ms. This lag also introduces problems for error correction because the information used to determine correction action can be up to 40ms old.

There is little you can do about this. You can, at your own risk, try to hack the sensor so you can tell at least when it starts to sample. This helps to reduce the lag to 20ms. You can also adjust your error correction method to perform smaller corrections, or perform corrections at intervals longer than 100ms. For linear uncertainly, you can reduce the speed when you know you are close to a cell in which you need to stop.

Spike Filtering

The GP2D12 sensor is notorious for its spike noise. This noise is due to the sudden current consumption and release (presumably for the IrED). Low pass filtering helps, but the spikes can still alter the average value sufficiently.

Consequently, a neighbor comparison technique can be used. If a sample is more than a threshold from its two neighbors, then it is replaced by an average of the two neighbors. This threshold should be based on osciloscope observation *after* the sensor is connected to the ADC circuit.

Clustering

The GP2D12 sensor only updates its output periodically. This is good and bad. It is good because we can use a low-pass filter during an output period to acquire the most accurate reading. This is bad because it means the response is delayed (by up to 40ms).

For accuracy, clustering should be used. But this means that we first have to find clusters, and secondly to use the average of a cluster after the cluster is identified.

If two clusters are different, they are easy to identify. If three samples are significantly different from the previous three samples (threshold is based on noise level), we are already three samples into the new cluster.

However, if two clusters are similar, it is not as easy to identify a cluster. The only thing we can do is to use the number of samples to mark the boundary between clusters. Nonetheless, if two clusters are not too different, using a fixed-window average across the two clusters may be acceptable.

Note that the use of clustering means we only trust data after at least one cluster's time. In other words, what we measure is at least 40ms stale. This is an inherent problem with the sensor. Clustering does not make the problem worse, but it does make the point more apparent.

The basic algorithm of clustering (with spike filtering) is as follows:

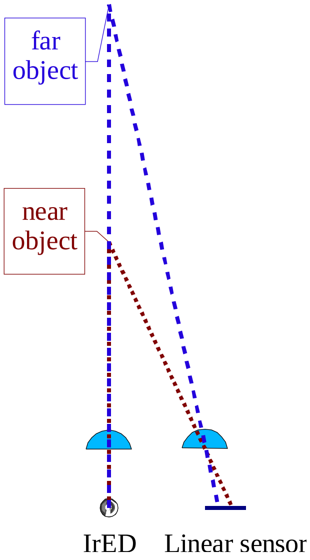
```
**if** $(|s_t - s_{t-1}| > t) \wedge (|s_{t-1} - s_{t-2}| > t)$ **then**
  $s_{t-1} \leftarrow \frac{(s_t + s_{t-2})}{2}$
**end if**
**if** $s_{t..t-2}$ is different from $s_{t-3..t-5}$ **then**
  add $s_{t-3..t-5}$ to cluster sum
  add 3 to number of samples
  compute value of previous cluster $c$
  reset number of samples to 3
  add $s_{t..t-2}$ to cluster sum
  trigger action relying on cluster transition
**else if** number of samples exceeds cluster limit **then**
  add $s_{t-5}$ to cluster circular queue
  compute running cluster value $c$
  trigger action relying on moving average
**end if**
```

Calibration

GP2D12 sensors do not come pre-calibrated. This means even though a sensor remains self consistent, two difference sensors may produce different output at the same distance from an object.

The sensor itself is based on triangulation. The emitter and the sensor are spaced at a fixed distance. When an object is at infinity, the reflection lands at a spot that is closest to the center of the sensor. This is the 0V point.

When an object is closer, the reflection lands at a spot that is further out from the center of the sensor.



Let y be the distance, and let V be the voltage. The relationship is approximated as $\frac{1}{y+a} = m \times V + b$. m is determined by the focal length of the sensor and the width of the sensor strip. a is a calibration constant based on the distance between the emitter and sensor. b is a calibration constant that accounts for the placement of the sensor strip.

Computation

Although it is possible to work with ADC values directly, it may be necessary to compute the range directly. $y = \frac{m}{V+b} - a$ is the general formula. The division is computation intensive. This is why, even if done in integral math, it should still be avoided.

When scaled properly, there is no need to perform floating point computation. m can be a large integer. y can be measured in millimeters. This way, there is no need to use floating point numbers.

Interpolation

Instead of using a function to compute the value, it is also possible to "brute force" the conversion. ADC sample points can be collected at fixed distances for each sensor. The distance of an intermediate ADC value can, then, be approximated by a linear line between the two closest sampled points. Based on the curve of the response of the sensor, this approach may not be too bad.

In terms of computation, this approach still requires one division in the interpolation. Given coordinates (x_1, y_1) and (x_2, y_2), and a new x_3, $y_3 = y_1 + (y_2 - y_1)\frac{x_3 - x_1}{x_2 - x_1}$. The true strength of this approach is that there is no need to model the response function at all.

Note that the division *can* be reduced to shifting. This technique requires the use of powers of 2 between steps of the x values (ADC readings). Taking samples based on fixed steps of ADC values is more difficult, but the gain of speed may be worthwhile.

The collection of fixed ADC steps should be done with the actual ADC device to be deployed. A special interactive calibration program should be written. The minimal interaction requires the following:

- one button to advance to the next ADC step (debounced or set a dead time of no interpretation)
- one blink pattern for under value
- one blink pattern for over value

- one blink pattern for dead-on

Due to the noisy nature of the sensor (and the ADC circuit), a fixed window of average (after filtering) should be utilized. In addition, it may be advisable to allow a one-count tolerance, especially for ADC values corresponding to further distances.

Chapter 17

High Level ”AI” Algorithm

17.1 What is it?

The high level ”AI” code of a Micromouse guides the robot through a maze. This level of code is not concerned about skew and offset errors, nor the resynchronization of location based on edge transitions. This code tries to figure out what to do next given the known walls.

17.2 The Ideal Algorithm and the Practical Algorithm

The ideal algorithm should take into account the time used for acceleration, deceleration, turning and probability of running into a wall (due to accumulation of errors). Such algorithms do exist. Indeed, to solve a maze given a set of known walls, we simply have to study solutions to solve a state-space search problem. Dijktra's algorithm, A* (pronounced A-star), C* all try to solve general state space search problems and find the optimal solution.

Unfortunately, all of the previous mentioned algorithms require quite a bit of memory and processor resources. By giving up a little bit of optimality, we can restrict the problem enough to use a much more efficient algorithm called “floodfill”.

17.3 The Floodfill Algorithm

Note that the floodfill algorithm only tries to assign a number to each cell (that indicates the distance from a destination). It does *not* tell the robot what to do next. Once we figure out the distance of each cell from a destination, it is trivial to figure out what to do next.

17.3.1 Basic Theory

The basic theory behind the floodfill algorithm is simple. The distance of a cell from another cell is simply the horizontal distance plus the vertical distance bewteen the cells. Of course, this distance does not directly correspond to the amount of time a robot requires to travel between the cells (due to time used for turning, acceleration and deceleration). However, it is close enough to give a mouse a good start.

Given this basic assumption, each destination cell has a value of zero because each is at zero distance from a destination. All neighboring cells (not counting diagonal ones) of a destination have values of 1 because they are one cell away from a destination. Similarly, all reachable neighboring cells of cells of value 1 have values of 2 because they are two cells away from a destination.

But what if we don't know the entire configuration of the maze? This is not a problem. If you have not explored a region, we simply *assume* there is no wall in that region. This means the algorithm is "greedy", it always make the most optimistic guess when information is insufficient.

While this approach may sound silly at first, it actually is a wise one. This is because the robot will look for the shortest path using this method. The proof is fairly long and complicated, but the basic idea is as follows:

> If there exists a shorter path than the taken path, the robot would have explored the shorter path to begin with. Therefore, it is not possible for the robot to first find a longer path.

This claim, however, does not guarantee that the robot will find the shortest path in the first pass. This is because the robot may be tricked to take "short paths", while each new mapped wall actually reveals the taken path is less than optimal. This claim only says that if there is a shorter path, the algorithm will find it.

17.3.2 The Algorithm

```
$l = 0$
$s = d$
$t = \emptyset$
while $s \neq \emptyset$ do
  while $s \neq \emptyset$ do
    for $c \in s$ do
      $s = s - c$
      for $m \in n(c)$ do
        if $v(c)$ is undefined then
          $v(m) = l$
          $t = t + m$
        end if
      end for
    end for
  end while
  $l = l + 1$
  swap $s$ with $t$
end while
```

In this algorithm, s is a set of cells being processed, while t is a set of cells to be processed in the next iteration. l is the "level" of cells, or the distance from a destination. d is the initial set of destination cells. $n(c)$ is a set of reachable neighbors of cell c. $v(c)$ is the value assigned to cell c, $v(c)$ is undefined for all cells initially.

17.3.3 Implementation

Central to this algorithm is the implementation of sets and cells. A region 6 maze is 16 by 16 cells, which gives a total of 256 cells. The most efficient method to implement a set is 8 bytes, and use one bit to represent one cell.

This requires the linearization of the two dimensional cells. Given that a cell is on row r and column c, the easiest linearization is $16r + c$. This uniquely assigns a number for each cell in a maze.

From the linear bit number, we can compute which byte in a set to retrieve and which bit of it represents the cell. The byte number is simply $\lfloor \frac{16r+c}{8} \rfloor$. But this is easy to compute because it really is just `2*r+c/8` in any programming language. Note that the division can be implemented by right shifts.

The bit number is also easy to compute, it is simply `c % 8` in C. Smart compilers automatically convert this to use bitwise and as in `c & 7`.

It is useful to define a structure in C to represent a set. This can be a definition like the following:

```
struct Set {
  char bytes[NUM_ROW * NUM_COL / 8];
};
```

Note that now you can pass a `struct Set` by value if you want to. Otherwise, it is impossible to pass an array by value in C.

Once a set is defined, you need to define ADT (abstrct data type) methods. The following are useful methods:

- `count` counts the number of elements in a set
- `add` adds an element to a set
- `remove` removes an element from a set
- `find` finds the first available element from a set

It is best to use the linearized number as parameter or returned value for these methods. The remaining task is to delinearize a linear number used to represent a cell. Assuming `l` is the linearized number, the row number is `l/16`, whereas the column number is `l%16`. Again, these operations can be replaced by right shifts and bitwise and.

17.3.4 Map Representation

For maximum RAM efficiency, it is best to represent each partition with a bit (1 means a partition exists, 0 means a partition doesn't exist). The program needs two arrays of bits, one to represent horizontal partitions and one to represent vertical partitions.

The boundary can be implied. This means there are only 15 columns of vertical walls and 15 rows of horizontal walls. Each column or row has 16 partitions. The total number of bytes required is $\frac{2\times15\times16}{2}$, or 60 bytes.

Once again, you can linearize the walls. However, the linearization now relies on which direction you want to look from a cell. For example, the east wall of a cell at row r and column c has a linear index of $16c+r$ in the vertical partition array. The west wall of the same cell has a linear index of $16(c-1)+r$ in the vertical partition array. The north wall of the same cell has a linear index of $16r+c$ in the horizontal partition array. The south wall of the same cell has a linear index of $16(r-1)+c$ in the horizontal partition array.

For efficiency, you can directly compute the byte number as `2*c + r/8` for the east partition of a cell, and use `r % 8` to compute the bit number in that byte. Recall that most compilers are smart enough to use shifts (instead of multiply and divide) and bitwise and (instead of modulus).

The boundary partitions are special cases because they are always there. For example, all cells on column 0 have west partitions. If you use the previous equation to computer the linearized bit number of the west wall of a column zero cell, it yields a negative number.

It is best to write some subroutines to abstract the handling of partitions. In particular, you can define two subroutines:

- `int existWall(int row, int col, int dir)` checks if a partition exists at the `dir` direction of a cell on row `row` and column `col`. `dir` enumerates the four directions (0 for north, 1 for east and etc.)
- `void setWall(int row, int col, int dir, int setClear)` sets or clears a wall. If `setClear` is non-zero, the subroutine sets a wall. If `setClear` is zero, the subroutine clears a wall.

Once you have these two functions, you can implement the logic to find reachable neighbors of a cell by checking which direction(s) does/do not have a parition blocking.

Chapter 18

High-Level and Low-Level Software Interface

The design of the interface between high-level code (floodfill, mapping, etc.) and low-level code (motion control, etc.) depends greatly on whether a run-time (or real-time) kernel is used. A multithreading kernel enables elegant designs.

18.1 The Necessity

The stepper logic is triggered by the timer. This means it is asynchronous to the rest of the system. We need to keep the execution of the ISRs short, so it is unwise to lump code for mapping, error correction and even floodfill in the timer ISR. So the question is, how do we trigger the code to map walls, perform error correction, and determine the next step?

18.2 Basic Stepper Logic

Portions of the stepper logic execute at a high frequency. The frequency division code for velocity and acceleration control executes for each invocation of the timer interrupt. There is little to gain to separate such code from the ISR using a thread interface. On the other hand, the code to move a step (and then to recompute remaining stopping distance and etc.) executes at a much lower frequency. Consequently, such code is a good candidate to be separated from the ISR.

Note that there are many sources of frequency division counter overflow. Linear velocity, linear acceleration, angular velocity, angular acceleration, left motor velocity and right motor velocity are all frequency division driven. As a result, a queue should be used to track the reason(s) to trigger frequency division counter overflow. Also, one overflow can potentially lead to the overflow of another, the handler thread must loop until the queue is empty.

Such a design also requires the queue entries maintain a “context” of the overflow. This is not difficult, because the structures for motion control are self-contained entities that provide sufficient context.

18.3 Sensor Reading Logic

At the lowest level, we have interrupt-driven ADC code to collect raw data. Such code trigger a thread that filters raw ADC reading, and convert processed ADC reading into distance (with range of uncertainty).

Once distance is known, more processing can be done. This information can be used by wall mapping, error correction and etc. Wall mapping logic is triggered based on the current location, so it cannot be triggered by the logic that computes distance.

However, the logic that tracks distance changes is triggered by the availability of new distance. This can be based on a combination of time as well as significant change of filtered raw ADC code.

The distance tracking code can trigger other threads based on patterns. A sharp transition can be used to correct parallel offset error. A slow change of side distance can be used to trigger perpendicular error correction.

18.4 High Level Logic

Compared to stepper, certain high level logic are triggered at even lower frequencies with lower priorities.

18.4.1 Mapping

The mapping logic should execute only when the sensors are passing a reliable area to read a wall. The triggering of mapping logic is performed when a linear step is performed.

18.4.2 Floodfill

The floodfill logic is triggered whenever a new wall is added, and floodfill values may be changed. As such, it is triggered by the mapping logic.

18.4.3 Navigation

The navigation logic is triggered based on the current position of the robot, and whether a change is needed. It is triggered when the "current course" needs to be changed because of the change of floodfill values. Consequently, navigation logic is triggered by the completion of floodfill computation.

Note that navigation logic is also triggered when a motion profile is completed. This means the navigation logic may be triggered directly from the stepping logic.

18.4.4 Error Correction

Error correction logic is triggered by many reasons. When a parallel wall is available, skew and perpendicular offset error correction should be done continuously.

Chapter 19

Grading Policy

This chapter discusses the grading policy for Micromouse teams at U.C. Davis. Obviously, this chapter has no relevance except to students at U.C. Davis.

19.1 Evaluation Method

Your grade depends on your proof of academic performance. This section lists all the proofs that I need for assigning your grade at the end of the spring quarter. Read this carefully so that you understand what will be used for your grade.

19.1.1 Quizzes

(This is new as of 06/23/2004, effective for the academic year immediately following.)

Unannounced quizzes will be given throughout the quarters. About 15 minutes are given for each quiz. Sometimes I give quizzes at the beginning of a period, other times I give quizzes at the end of a period. Each quiz will only cover materials already covered in the class, or prerequisite material for the course.

Because students in such a class come from different background, each quiz will have a variety of question covering different fields. Such fields include computer science concepts, programming, electronic circuits, mechanical design and etc. Note that you don't need to know everything to get a full score on a quiz, but you do need to have some expertise in at least one area.

Quizzes are individual. In other words, each student must complete his or her quiz. Failure to complete a quiz individually is considered cheating, and the quiz is voided as a result.

19.1.2 Monthly Reports

In the first week of each month, a group report should be turned in for grading purposes. The main purpose of a report is to let me know of the progress of the project, or there lack of.

The following is a list of items that should be in the weekly report.

- List of participants: this is a list of the names of the participants. I know this sounds silly, this list should not change over time anyway. Please only list the members who participated in the project in the reported month. In the past there has been cases of members totally not participating for periods up to an entire semester!

- Principle editor/author: this is a list of the names of members who participated in the writing of the report.

- Executive summary: this is a short paragraph that outlines what happened, pressing issues, short term plans (for the following month) and other topics that you think I should know. Keep this short, you have other opportunities to elaborate.
- What happened: this section lists, in detail, what happened in the previous month. Of course, only include items that relate to the project. It is best to include a small section for each component of the robot. Also, indicate who did what.
- Pressing issues: this section lists pressing issues of the group. Such issues can range from technical to personal. For example, if a member refuses to participate, it is a pressing issue. Other examples of pressing issues include technical problems that cannot be solved.
- Short term plan: this section lists, in detail, what is planned for "this month". Include a small section for each component of the robot. Be sure to indicate who is assigned to handle which task.
- Appendix: meeting logs (normally between 4 to 8 of these). For *each* class meeting and out-of-class group meeting, keep a log of the following information:
  - Date/Time: this is obvious, but people often miss this field.
  - Participants: list the members who participated.
  - Summary: at least one sentence to describe what happened in this meeting. You can expand as much as you want.

19.2 Project Completion Date

(This section is added as of 06/23/2004.)

Groups that finish the project early deserves a better grade. Historically, the best groups tend to complete by Picnic Day. 100% of this component is awarded to a group that completes the project by Picnic Day. 50% of this component is awarded to a group that completes the project before the last day of instruction of spring quarter.

Of course, there are always special considerations. The following subsections list such considerations.

19.2.1 Group Resources

Each group can have up to five members. Obviously, a group with five members can complete the project faster than a group with only two members. Or so it seems.

With more members, work can be done in parallel. However, at the same time, more communication is necessary. As a result, the actual time factor is not exactly directly proportional to the number of members. I reserve the right to make necessary adjustments on a group-by-group basis.

19.2.2 Design Complexity

The most basic traditional design requires significantly less resources compared to a design that attempts to use innovative methods and components. As a result, I reserve the rights to make necessary adjustments when a design is more complex than the traditional design.

Note that this factor has to be determined in the first quarter. I will let you know whether your design is considered complex compared to a traditional design by the end of the first quarter. It is difficult to estimate and quantify complexity, but I will try to give you some form of numerical guestimate.

19.2.3 Completeness

If a project is not completed by the end of the spring quarter, I reserve the right to assign points based on the completeness of the robot.

19.2.4 "Fit and Finish"

In the past, many groups that cannot get a good grade attempted to use fit and finish to bump up the grade. This does *not* work. This is an engineering design class, and not automobile manufacturing plant.

19.3 Grading by Group or Individually

By default, all members of a group receive the same grade. Ideally, all members of a team should contribute about evenly to the implementation of the product, and therefore all members should receive the same grade.

Unfortunately, as illustrated over and over again, grading by group is not always possible. Sometimes, work done in a group is unevenly distributed and the contributing members request that the non-contributing members be graded differently and not to inherit the default grade.

Other times, team members cannot agree on how the project should progress and split up. This also requires individual grading because there is no single end product for grading purposes.

19.4 Group Grading

Group grading is quite simple. If a micromouse solves the official maze on Picnic Day and submits a reasonable report using my guideline, each member of the team earn an A. A C is rewarded to members of teams that produce micromice that solve a maze of my design before the end of Spring quarter. A B is rewarded to members of teams that produce micromice that solve a maze of my design between Picnic Day and the end of Spring quarter. Minor adjustments (A-, B+, B- and C+) are applied based on date of completion.

The instructor, I, reserves the right to make adjustments based on factors *other than* the date of completion (when the micromouse solves a maze) and the quality of the final report.

In the event that a micromouse cannot solve the maze at the end of Spring quarter, the instructor reads the final report to assign a grade. Other factors, such as difficulty of the design and factors out of the control of team members, are considered in the determination of the final grade.

The following subsections list all the components needed for group grading.

19.5 Individual Grading

Upon request of at least one member of a team, the instructor grades each member individually. This is *not* the preferred method because a functional micromouse requires *more than* functional individual parts. The integration of parts is an essential but challenging process that can determine the success or failure of a micromouse project.

Keep in mind that a C means "satisfactory", or that a student fulfills the minimum expectations of the class. A B means "above average" and an A means "excellent/exceptional work". As a result, functional individual components that are sufficiently documented and demonstrated can only warrant a C.

In order to get a grade better than a C in individual grading, a member must produce, document and demonstrate more than just a functional component.

Chapter 20

Complexity Index

This chapter explains the complexity index of a robot (or a component of a robot). This is used as an adjustment factor for a project. This chapter is divided into sections by component.

20.1 Generally Speaking

Note that complexity is only use for positive grading adjustment when the complexity is innovative and it has the potential to improve some aspect of the functionality of a Micromouse. Useless and unnecessary complexity does not influence grading at all.

Furthermore, if you are using a non-baseline feature that is provided by a pre-designed manufactured component, the adjustment is lessened. This makes sense because the “design” part is already done! On the other hand, some adjustment will still be effective for the *application* of a non-baseline design.

20.2 Drive System

A traditional and simplistic Micromouse use differentially steered stepper motors directly attached to the wheels. This forms the baseline of the complexity of the drive system.

20.3 Stepper Drive Circuit

The baseline design uses low side switches or H-bridges for unipolar stepper motors and bipolar stepper motors, respectively.

20.4 Wall Sensors

The baseline design uses look down reflective sensors.

20.5 Voltage Regulation

20.6 Custom Circuit

20.7 Custom PCB

Part V

Assembly

Chapter 21

PCB Related Assembly Techniques

21.1 Tools

In order to assemble (populate or stuff) a printed circuit board, you may need many or few tools. It all depends on the design of the circuit board and the quality you expect from the assembly. This section discusses various tools for PCB assembly.

21.1.1 Lighting

Even before we talk about various soldering irons and soldering stations, let's talk about lighting. Lighting is important for most soldering work because bad solder joints cause a lot of problems down the line. It is worth the extra time and effort to set up proper lighting for your soldering or desoldering work in the first place.

Contrary to common belief, you *don't* have to spend much money to get good lighting. Most desk lamps will do a nice job. I do recommend the use of florescent light (especially the compact type) over incandescent or halogen type. This is because the heat produced for the latter two can lead to discomfort when you need to work close to the PCB.

One trick to get a nice bright and cool light source is to use the compact fluorescent bulbs with included reflectors. These "bulbs" are intended to replace hot flood lamps and security lights. With these included reflectors, you can disassemble the reflector of your light, making the light assembly lighter and easier to handle.

21.1.2 Magnification

When are we going to talk about soldering tools? Well, a little later.

For through-hole components, bare eyes are sufficient for most people. However, since most newer PCBs use surface mount parts to increase density and decrease cost, you will probably encounter surface mount soldering soon anyway. Surface mount components can be very tiny with even smaller leads. Placement and soldering with bare eyes can be impossible.

There are two types of magnification that you should consider. First, you may need *constant* magnification when you solder or desolder. For this purpose, you need a magnifier that you can comfortably wear for hours. While you can spend more than US$50 for a high quality hood-type magnifier, I find that the reading glasses and reading clip-on glasses are sufficient. Be sure to get the highest amplification factor. Most drug stores carry reading glasses and clip-on attachments of up to +3.0 diopters.

The second type of magnification is for examination and diagnosis. For this purpose, you need a high power magnifying glass or loupe. I recommend an 8x or 10x jeweler's loupe. A good quality loupe has less aberration, which is important because a solder bridge between two pins of a high density component can look like aberration!

Lit loupes or magnifying glasses is nice, but they are not necessary. If you set up your lighting properly in the first place, you usually don't need to worry much. Lit loupes are a bit more expensive, and you usually don't get to choose good optics in the first place.

21.1.3 PCB Cleaning

Why are we not discussing soldering yet? Well, there are more important topics to cover first.

PCB cleaning is critical to the quality of PCB assembly. This is because grease and other impurities contaminate the contact surface between the PCB and a component. The result is either solder not even sticking to one side or a bad solder joint.

The solution for general PCB cleaning is just alcohol. However, don't use the drug-store variety. Rubbing alcohol is about 70% alcohol. While this concentration is sufficient for killing germs and causing great pain to any wound, it is not useful for removing grease and flux.

Instead, go to an electronics supply store (in some cities in California, Fry's Electronics) and purchase alcohol that is intended for PCB cleaning. Such alcohol usually has a 90% concentration and *not intended for consumption nor skin contact*. If you touch it to inhale its vapor in a normal situation, you'll probably be okay. Just don't intentionally try to inhale or rub it on yourself. Absolutely don't drink it, though!

Although I am cost conscious most of the time, I do not recommend the use of regular bathroom tissue paper or even paper towel for cleaning a PCB. The main reason is because of lint. You can purchase inexpensive lint-free cleaning pads from stores.

An acid brush can also be very helpful when you need to clean between component pins. These brushes are inexpensive (US$0.10 or less).

Cotton swabs are also useful for mopping up cleaning solution. Keep plenty around your workbench.

21.1.4 Solding Tools

Now we start to talk about soldering tools.

Unless you plan to solder wires and contacts (switches and etc.) exclusively, be prepared to invest a little money on a good soldering station.

But what's wrong with the inexpensive pencil-style or gun-style soldering tools? The answer is the lack of temperature control. The low power soldering pencils do not have enough heat to flow solder correctly. The result is cold solder joints. Cold solder joints can be the source of many problems to come. The high power soldering guns are too hot for heat sensitive components. Can you guess which components are heat sensitive? It turns out that even pads and traces on the PCB are heat sensitive as well! At 800 °F, pads and traces on a PCB can easily lose their adhesive bond to the fiber glass material. Lift traces and pads are very time consuming to repair.

Even among temperature-controlled soldering stations, there are good ones and bad ones. A good soldering station controls temperature accurately and quickly. In other words, when there is a large heat load, the station quickly converts more energy to heat, but stops immediately when the tip reaches the preset temperature. A bad soldering station, on the other hand, reacts slowly and tends to over shoot. This means a bad soldering station tends to delay reacting to heat loads. This translates to a long time between the tip touching a PCB and a pad eventually heating up. At the same time, the tip temperature over shoots. This means once the tip heats up, it heats up too much and goes beyond the preset temperature.

How can you tell whether a soldering station is good or bad? Unfortunately, the answer is not easily available. Through experience and feedback from others, you can trust Metcal soldering stations. Metcal soldering stations use the skin effect of inductance. Temperature control of Metcal soldering stations occurs directly at the tips. The response is quick and accurate. As you can guess by now, Metcal soldering stations are not cheap. The least expensive model is US$300. If you shop at auction websites, you can typically get one for US$150 or so.

Hakko and Weller soldering stations less than US$200 should be avoided. These soldering stations typically react slowly and over shoot wildly. The manufacturer of Chip-Quik is starting to market their own soldering stations, and these are supposed to be both inexpensive (about US$100 or less) and high quality. Search for the website of Chip Quik and look for their soldering stations.

21.1.5 Soldering Tips

There are many soldering tips available, but you typically don't need more than two. For surface mount work, you need to get a soldering tip with a pointed end. Try to avoid the ones with a long cone. The length of the cone delays temperature feedback and leads to long response time and over shooting. In other words, get a short soldering tip that has a pointed end.

Metcal sells soldering tips that are prebent 30 degrees or so. This is a *good* idea. First, a bent tip makes it easier to access tight space and work under a +3.0 dioptor magnifying glass. This is because a bent tip makes it unnecessary to tilt the soldering pencil itself. Second, a bent tip also makes it easy to "knife" or "slide" solder fine pitch leads of surface mount components. More on this topic later.

Besides a find-tipped soldering tip, you may also want to consider getting a standard tip. Most soldering stations come with standard tips already, so you probably don't need to spend any extra money. A standard tip is bigger and more blunt. This type of tips is useful for general-purposed soldering and it is more effective when the amount of heat mass to heat up is large (due to the larger contact area).

21.1.6 Flux

As long as you have PCB grade alcohol, there is no such thing as too much flux (because you can always remove the excess amount).

Flux is inexpensive, but it is critical to soldering and desoldering. This is because flux is a chemical that "activates" solder. Up to a certain point, flux helps to get rid of impurities between two surfaces to be soldered. It also helps solder flow once molten. Properly flowing molten solder wicks up most metal like water wicks up tissue paper.

You can get the usual flux, which must be washed/cleaned because of its corrosive nature. You can also get "no-wash" type flux which can be left on a PCB. For low quantity production, it really doesn't matter since washing and cleaning a PCB don't talk much time.

Be sure to get flux that is intended for electronics soldering, not copper pipe "sweating". The red solution for copper pipe "sweating" is very corrosive and it can easily damage pads and traces on a PCB.

Some people find vaporized flux pleasant smelling. While this *may* be the case for you, please refrain from inhaling flux vapor! Not only is this vapor unhealthy by itself, it also contains lead oxide. All lead (elemental or compound) is bad for your body.

21.1.7 Desolding Braids

Desoldering braids are made for desoldering components (hence the name), but they are best used for mopping excess solder on a PCB. Combined with enough flux on a clean PCB, desoldering braids effectively remove excess solder that can otherise short two traces or pads. For surface mount work, get the thinner type so it is easier to get to tight spots.

21.1.8 Desoldering Plungers/Guns

Also called "solder suckers", desoldering plungers are spring-loaded pumps. You depress the plunger and lock it in position. Then you position the tip where you need to remove solder. Press a button and the plunger creates the suction to vacuum molten solder.

I have had limited success with these plungers, so I cannot recommend them to you. If you do need to remove *through hole* components, it is better to use a desoldering gun.

There are two types of desoldering guns. The first type utilizes a continuous pump to create the vacuum necessary to remove solder. While this type of desoldering gun is effective, they are also quite expensive. The best price is about US$170 for a brand new unit.

The second type utilizes a solenoid actuated plunger for a "one-shot" suction every time the button is pushed. This type is less expensive at about US$100 each (brand new). Because there is no continuous suction, a one-shot desoldering gun may seem more difficult to use. In reality, on the other hand, a one-shot desoldering gun is quite easy to operate. The key is to position the tip correctly to make sure it is perpendicular to the PCB. The tip should also be heated properly so solder flows well. As long as the correct angle and temperature is provided, a one-shot desoldering gun is an effective tool for removing through-hole parts.

21.2 PCB Preparation

21.2.1 Partitioning a PCB

21.3 Soldering

21.4 Diagnosis

21.5 Desoldering

Part VI

Software/Hardware Tools

Chapter 22

AVR Programming Tools

Although there are many available programming tools for the AVR family of processor, I only recommend the use of the Linux-hosted tool set. The main reason is due to the native support of GNU tools.

While I do not want to dictate what kind of environment or tools you can use for your projects, I find it far more effective to ask everyone in the class to use the same OS and the same tool set. If you decide to use an OS or tool chain that differs from my recommendation, you may experience more unexpected problems. I will help you resolve your problems to my best ability, but I will not allocate an "unfair" amount of time so that other groups suffer from your choice of OS or tools.

22.1 Getting Linux

The first step to set up your development machine is to download and install Linux. You have many choices of Linux, I recommend only one distribution that I can effectively support.

I choose the Debian distribution http://www.debian.org because it is relatively easy to setup, and it is a breeze to maintain and update. It also has all the necessary packages for AVR program development.

You do not need a brand new PC to install Debian Linux. For what we are doing, you can easily use an older PC (Pentium MMX/II class, about 166MHz to 300MHz, 64MB of SDRAM and about 2GB of hard disk space). I do recommend the use of notebook/laptop computers because you can easily bring your project to the lab and back home. You can often find a notebook computer that has sufficient resources for our project for about US$150 for a P2 class to US$300 for a low-end P3 class (at auction sites or specialized dealers selling refurbished computers).

It is best to get a computer with a 100Mbps network port already built-in. This facilitates many network enabled benefits that come with Debian Linux (and most generic Linux tools). If your computer has no network port, you can find brand-new network interface cards (NICs) for less than US$20 in most computer/electronics stores. Fry's regularly sells refurbished PCMCIA/cardbus 100Mbps LAN NICs at US$10 each.

For best compatibility with Linux, not all brands are equal. Intel chipset-based network cards tend to be easier to deal with. The most popular Intel chipset is the ExpressPro 100. You can find PCMCIA network cards using this chipset for about US$20 at Fry's. The card comes with a dongle, so you must be careful with it (otherwise the dongle connector can easily be damaged!).

22.1.1 VMWare Workstation (non-free)

VMWare is not Linux, but it is a tool that allows you to run Linux as a window in a Win32 environment. Alternatively, it allows you to run Windows in a Linux environment. With student discount, it is about US$150 (last I checked anyway).

The main advantage of VMWare is that you don't need to dual boot. You can remain in the "host" operating system that you feel comfortable with (for most people, it is Windows). You can use the "guest" OS only when you need to. You also have access to both OSes *at the same time*. Copying and pasting across the OSes can be tricky, but the two OSes can share the same file system(s), which makes it easy to transfer files from one OS to another.

Another cool thing about VMWare is that the "guest OS" can be completely shielded. The so called file system for the guest OS can be just a huge file in the "host OS". This means the "guest OS" can never mess up the "host OS".

VMWare also allows the "guest OS" share the same network connection to the internet with the "host OS". This is very cool because now your "guest OS" can access the internet resources.

I only have experience using Linux as my "host OS" and Windows 98 as my "guest OS". If you want to set up VMWare in the opposite direction, I can only give you a few pointers.

22.1.2 Installing Debian over the Web

This is the best option if you have broadband internet connection. Go to the URL http://www.debian.org/distrib/netinst and click on either a "Minimal CD" or "Floppy Disks". Most people prefer the floppy disks option.

As a shortcut, go to the URL http://ftp.us.debian.org/debian/dists/stable/main/disks-i386/current/images-1.44/ and download the files of `.bin` extension. These files are floppy disk images. Most people only need `root.bin` and `rescue.bin`, but you may need the `driver-?.bin` if you have some exotic hardware that is not supported by compiled-in options.

Once you have downloaded the floppy disk images, download the DOS utility `rawrite2.exe` from http://ftp.us.debian.org/debian/dists/stable/main/disks-i386/current/dosutils/rawrite2.exe. This command line utility program allows you to re-create floppy disks based on the `.bin` files that you just downloaded earlier. Refer to http://ftp.us.debian.org/debian/dists/stable/main/disks-i386/current/dosutils/rawrite2.txt for some documentation. You can also invoke `rawrite2` without any parameters to see the included help message.

Make sure you label each floppy disk properly. On the target computer, boot with the `rescue` disk, then it will ask you to insert the `root` disk. The rest of the process is fairly easy if you follow the instructions at the Debian website.

The only tricky part if you are using a notebook computer is the need to set up PCMCIA before you attempt to install the base system.

22.1.3 Dual Booting

If you have a machine with an existing OS that you want to keep, but you still want to install Linux in the remaining space, you need to be more careful.

First, you need to make sure there is enough space for Linux. It is best to reserve about 2GB for the main OS/user space, 128MB for the swap file and about 20MB for the boot partition. You need to make sure the boot partition is a physical partition and it is within the first 1GB of the physical disk (or even closer if you have an earlier BIOS), but the other two partitions can be anywhere, even as logical partitions.

If you know that you have enough disk space, but the partitions are not sized/structured correctly, you can use a tool such as Partition Magic (from PowerQuest) to repartition the hard disk. I cannot guarantee the safety of your existing data, so you should back up the drives if there is anything you don't want to lose. This is how I would set up the new partitions:

1. first physical partition be a 20MB partition for the Linux boot partition, it should be an extension-2 (ext2) type.

2. second physical partition be the boot partition for your other OS. It should contain the system files to start up your other. OS. For most Windows users, this is your `C:` drive.

3. make the rest of the partitions an extended partition.
4. reserve 128MB (or more) as the Linux swap partition inside the extended partition
5. reserve 2GB (or more) as the main Linux OS/user partition

Until you get the Linux OS setup, set the boot/active flag on the partition containing your other OS. This way, you can still always boot to your other OS.

Do not change or do anything to the master boot record (MBR), or you may not be able to boot at all!

After following these steps, you can install Debian Linux. Be careful do not let the installer repartition again, just initialize (format) the predesignated partitions and associated them to the correct Linux use.

After, I repeat, *after*, you get Debian Linux installed, you can set up LILO to recognize both the Linux boot partition as well as the other OS's partition. At this point, you can consider setting the Linux boot partition (the small 20MB partition) as the boot/active partition. Again, do not mess with the master boot record, just change the boot/active partition to the Linux boot partition.

22.1.4 Specify Minimal Packages

Because we need to upgrade the whole system to the "unstable" distribution eventually, don't even bother to install packages that you don't need to get the computer started. This means don't install XWindows, Gnome, KDE and other goodies.

This means in the initial set up, don't bother to run `tasksel` or `dselect`. Just install the bare minimum, then follow the instructions of the next subsection to upgrade the distribution.

22.1.5 Upgrading to the Unstable Distribution

Once you have the "stable" distribution installed and running, you can upgrade to the "unstable" distribution. The main reason to upgrade is to get the latest and greatest tools for AVR related software development.

To do this, log in as root. First, edit `/etc/apt/sources.list` and replace all the "stable" references to "unstable" (except for the security entry). Save the file. Then, issue the following commands:

```
apt-get update
apt-get dist-upgrade
```

This will bring in the "unstable" files to replace your "stable" files. Yes, this is a good thing because you'll have the latest updates.

After the system is updated to the "unstable" distribution, you can use `dselect` to get the optional packages.

22.1.6 Get the Rest of the Packages

`dselect` is a command-line tool that lets you update and select packages to install. You probably should select the "update" command once in a while to update your catalog of packages. Use the "select" command to select packages.

For a relatively low-end machine with an 800x600 display, it is relatively pointless to get XWindow, KDE or Gnome. Linux comes with 6 virtual text mode consoles, which is more than sufficient for most people. I understand that text mode consoles do not look "sexy", but they are just as functional (if not more functional than xterms on a low-res screen!).

With a low-res. screen, the only reason why you may want to consider getting XWindow is for viewing PDF files. You don't need Gnome or KDE for this purpose. Just install one of the many windows managers (fluxbox, fvwm, twm and etc.).

By *not* running XWindow and window managers, you are easily saving 200MB of memory requirement. This is what makes a low-end machine with 64MB of memory (or even less!) practical.

You can change the console text mode from 25 lines to 50 lines. by looking up the man pages of `consolechars`.

If you have a fancy screen (1024x768 at least) and enough memory (128MB or more), you can consider installing an GUI on top of Linux. You don't need anything like Gnome or KDE unless you plan to use Linux as your desktop OS (to run office applications). Unless you have a preference for either KDE or a bare-minimum GUI frontend, I personally recommend installing `gnome-core` (and related files). This should automatically trigger the inclusion of most GUI tools that you may need.

22.1.7 Get the AVR tools

Use `dselect` to get the AVR tools, particularly, get `binutils-avr`, `gcc-avr`, `avr-libc`, and `uisp`. Some of these may be automatically selected or recommended when other packages are selected.