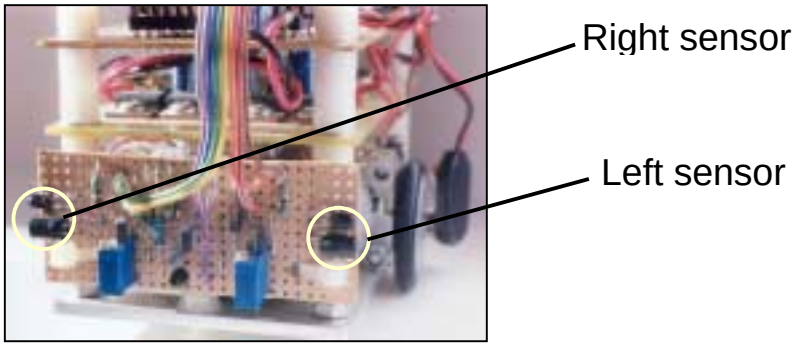
# The low-level control system of a maze-solving MicroMouse robot, George II

Based on a presentation given at MINOS ’02 on 6th April 2002

## Sensor system

George II has three IR reflective sensors at the front: two look sideways and provide distance information, the third looks forwards and is a simple yes/no ‘bump’ sensor. The side sensors also detect openings in the wall. Each IR LED is driven from a latched bit of the microcontroller data bus and can be switched on and off by software. The photodiodes are similarly addressable. The analogue signal from each photodiode is sampled and each sample converted to an 8-bit number by an ADC chip.




## Drive system

George II is of the ‘wheelchair’ type with a DC motor driving the wheel on each side and is steered by varying the motor speed. Pulse Width Modulation (PWM) is used for speed control, provided by special circuits within the microcontroller. The hardware interface between the microcontroller and the motors consists of a pair of ‘H-bridge’ circuits allowing four modes of operation: forward/reverse/coast/brake. Each motor shaft has an 8-slot tachometer fitted, providing 1 pulse for each 0.8 mm of wheel travel.

## Straight-line control

The first objective has been to develop a feedback control system capable of driving the mouse in a straight line down the centre line between two maze walls. It should also be able to ‘acquire’ the centre line from an off-centre start as well as track once locked on. This has been achieved using the well-known ‘PID’ algorithm (Proportional-Integral-Differential). Software has been written to show the contributions from each of the three PID components. Control of the mouse when negotiating corners is another issue, because the data from the side sensors becomes confusing and largely unusable as the corner is turned.

**Defining ‘e’ and calculating a value for it.**

e = 0 when running on centre line

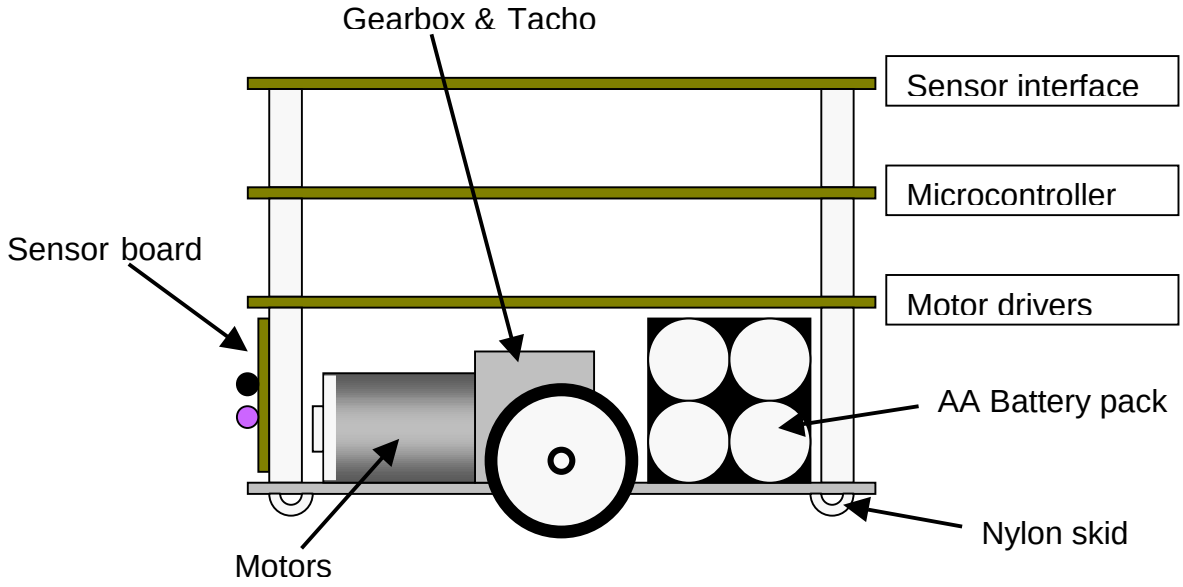
e < 0 when running to the left of centre

e > 0 when running to the right of centre

With an 8-bit ADC, the sensor value varies from 08H (far from wall) to b0H (stuck against wall). A sensor value < 4 means that a wall opening has been located. The value for e is calculated by performing the following sequence of operations:

1. Read Left sensor value (LED off)
2. Turn on Left LED and wait
3. Read Left sensor value (LED on)
4. Subtract ‘dark’ from ‘lit’ value and scale by 2
5. Repeat for Right sensor
6. Subtract Left value from Right value. Result = ‘e’, an 8-bit signed number

This sequence is repeated in an endless loop, and includes checks for wall openings on either side. Only one LED is switched on at a time to conserve power and is held on for a short period before a Lit reading is taken. Both Lit and Dark readings are taken and subtracted to provide a measure of ambient light compensation. The final value for e is also independent of minor variations in the distance between the walls.




**The PID algorithm**

$$c(t) = K_P e(t) + K_I \int e(t)\,dt + K_D \frac{de}{dt}$$

c = control (steering) effort

e = measured lateral error from centre line

K = tuning constants

The first, **Proportional term** simply provides a control signal proportional to the size of the error. A control system can use this term alone but only if it responds very quickly to the steering output, and the robot is relatively slow moving. Otherwise the best you can get is a mouse wandering from side to side as it goes along tracing out a sine wave shaped track. This effect and others can be simulated using a mobile robot simulator downloaded from the Internet at: **http://www.liorelazary.com/Robotics/robot_pid_simulators.htm** The steering signal calculated by the microcontroller is $c = K_P e$. A simple 8-bit x 8-bit multiply is performed followed by an addition to the base PWM value controlling the motor speed of the left-hand wheel. The robot steers to the left as this wheel is slowed down relative to the right hand wheel, and to the right as it is driven faster than the right-hand wheel. A typical value for $K_P$ is likely to be less than 1.0.

The **Differential or derivative term** is necessary for smooth straight-line performance. If the P term makes the mouse behave as if it had coil springs mounted between itself and the walls on each side, then the D term provides the 'shock-absorbers' or dampers. It is derived from: $D = (e_{NEW} – e_{PREVIOUS})$. A record is kept of the last calculated error value, and is subtracted from the one just calculated. This gives a measure of how fast the error is *changing* as the mouse moves along. The control signal now becomes: $c = K_P e + K_D D$. The D term will increase the steering effort if the error is getting worse, but will reduce it if the error is getting smaller. A typical value for $K_D$ is likely to be quite a bit greater than 1.0 because the calculated changes in e between sample intervals are likely to be small.

The **Integral term** may be omitted if the drive systems on each side of the mouse are evenly matched so that the wheels revolve at the same speed when the same PWM signal or voltage value is applied to each motor. If not, which is very likely, then you will have a steering 'bias'. With no control system, the mouse will travel in a circle whose diameter depends on the size of the bias. With a PD controller, the mouse will travel in a straight line, but at a fixed offset from the centre line. The I term is used to remove this offset, and is derived from: $I = I + e$, in other words, an accumulator function. The control signal now becomes: $c = K_P e + K_D D + K_I I$. The value for I will rapidly increase and then stabilize at a constant level as e reduces to zero. This provides a constant fixed bias to the control system acting to cancel out the mechanical one present. A typical value for $K_I$ is likely to be much smaller than 1.0 to ensure that the I term doesn't apply too much control before the correct value is reached.

## Velocity control

So far, the wheel speed is determined entirely by feedback from the IR lateral position sensors modifying a fixed ‘reference’ PWM value. We can use the motor tachos to provide feedback for a velocity control loop for each wheel. Speed is calculated in part by the microcontroller hardware using its count/capture registers. Measured speed is compared with a fixed reference set by user, and provides another input to the PWM motor control. This velocity control loop on each wheel compensates for falling battery voltage, will compensate for differences between the drives, and provides a measure of ‘Traction control’. The tricky bit is ensuring that it doesn’t interfere with the steering control loop. It is a useful technique, but may not be appropriate for the MicroMouse environment.

## Practical maths

The various constants used in a theoretical PID controller are typically small, positive or negative numbers, accurate perhaps to several decimal places. Arithmetic with such ‘Floating-Point’ numbers is difficult and time-consuming on a small micro, but good results can be obtained with plain ‘Fixed-Point’ 8- and 16-bit 2’s complement signed numbers. My microcontroller (an 8051 type) can handle 8-bit add, subtract, multiply and divide using its own instruction set, and 16- and 32-bit operations using a subroutine library. This library is tacked on the end of the MOUSEMON monitor/loader program, the source code for which can be downloaded from my web site. There are ways to overcome the lack of floating-point operation. In order to multiply by 0.1 for example, divide by 10 instead.

## Conclusion

George II is now able to race down corridors at (relatively) high speed under perfect control. The next problem is to sort out the cornering....