Vehicle Obstacle Course Summer Laboratory

Aims

The purpose of this lab is to provide experience of being taught in English, team working, open ended labs and problem based learning. This sort of lab is common in the University of Birmingham.

Specifically, the aim is to develop a simple vehicle from the components given to you in the lab.

Objective

You will be given a model car and some components. By the end of the week, the car is expected to navigate its way across an obstacle course similar to that shown in Figure 1. The course is a flat floor, the obstacles cannot be moved.

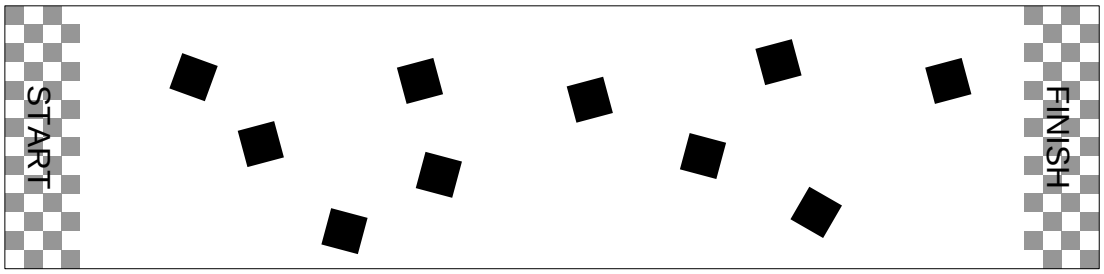


Figure 1

You will be given the chassis of a 1:20 scale remote control Mini Cooper S on which to base your project vehicle, this provides you with movement, steering and a power supply. You are also provided a set of components such as switches, buttons, ICs, a microprocessor, and other parts to help you.

Introduction to PICs

Programmable Interface Controllers are integrated circuits which have the ability to execute the program loaded onto them. The PIC you will be using for this lab is the 18F2525; it has several integrated peripherals such as a serial interface, an analogue to digital converter (ADC), and a pulse width modulator (PWM).

The PIC runs off a 5V supply, and needs several external components to make it work. This PIC has its own internal clock; many PICs use an external crystal.

Task 1

The circuit in Figure 2 can be used to power up the PIC and demonstrate a simple program.

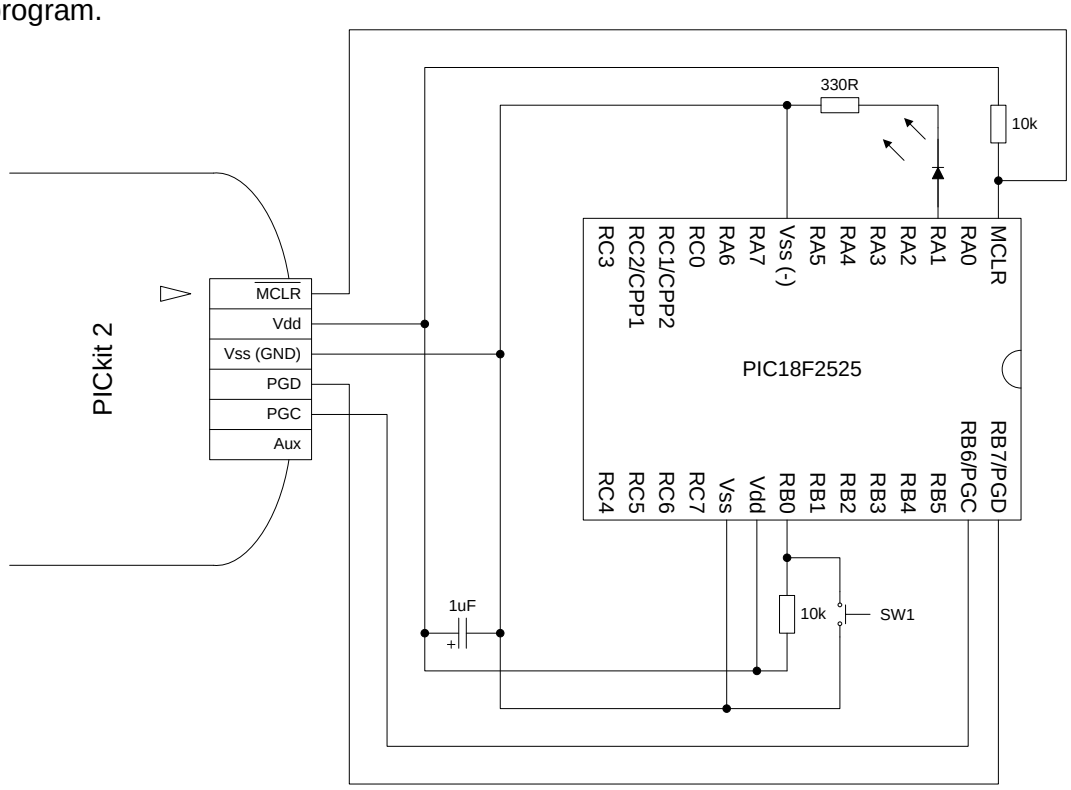


Figure 2 - PIC, programmer, and LED

The program we are going to write will cause the LED to light up when SW1 is pressed.

The first job is to set up the PIC program C files. These contain a commented-out header which holds useful information regarding the program e.g. name, date, revision, device, and function.

Next, the *configuration bits* of the PIC need to be defined. These set up aspects such as the oscillator, code protection, EEPROM protection, and some of the internal timers.

```
////////////////////////////////////////////////////////////////////
//                                                                //
//      AUTHOR:     Your Name                                     //
//      DATE:       12/06/09                                      //
//      REVISION:   1.0                                           //
//      DEVICE:     PIC18F2525                                    //
//      FILENAME:   LED_ON.c                                      //
//                                                                //
/////// DESCRIPTION ////////////////////////////////////////////////
// Turn on LED on RA1 on SW1 press                                //
////////////////////////////////////////////////////////////////////

#include <p18F2525.h>

//********** Configuration bits **********//
#pragma config OSC    = INTIO7       //Internal oscillator, output to pin 10
#pragma config FCMEN  = OFF          //Fail-safe clock monitor disabled
#pragma config IESO   = OFF          //Int.-ext. oscillator switch over disabled
#pragma config PWRT   = OFF          //Power-up timer  disabled
#pragma config BOREN  = OFF          //Brown-out reset disabled
#pragma config WDT    = OFF          //Watch-dog timer disabled
#pragma config MCLRE  = ON           //Master clear enabled
#pragma config LPT1OSC= OFF          //T1 oscillator disabled
#pragma config PBADEN = OFF          //Port B analogue-digital disabled on reset
#pragma config STVREN = OFF          //Stack overflow reset disabled
#pragma config LVP    = OFF          //Low voltage programming disabled
#pragma config XINST  = OFF          //XINST disabled
#pragma config DEBUG  = OFF          //Background debugger disabled

#pragma config CP0    = OFF          //Code protection disabled
#pragma config CP1    = OFF
#pragma config CP2    = OFF

#pragma config WRT0   = OFF          //Write protection disabled
#pragma config WRT1   = OFF
#pragma config WRT2   = OFF

#pragma config WRTB   = OFF          //Boot block write protection
#pragma config WRTC   = OFF          //Config reg write protection
#pragma config WRTD   = OFF          //Data EEPROM write protection

#pragma config EBTR0  = OFF          //Table read write protection
#pragma config EBTR1  = OFF
#pragma config EBTR2  = OFF

#pragma config EBTRB  = OFF          //Boot block table write protection

#pragma config CPB    = OFF          //Boot block code protection disabled
#pragma config CPD    = OFF          //Data EEPROM code protection disabled
```

Open the workspace `LED_ON.mcw` provided for you in the Lab File directory. The workspace configures the compiler for the PIC18F2525 and sets up an interface with the PICkit 2 for you.

The next stage is to define LED and SW1 in the code to make the program easier to read and understand. We are also defining 1 and 0 as ON and OFF respectively.

```
// Definitions ////////////////////////////
#define LED           LATAbits.LATA1
#define SW1           PORTBbits.RB0

#define ON            1
#define OFF           0
```

Copy the above code into your program.

Next we write the actual program code itself. C18 requires `#pragma code` to be written once, before any functions. Like any other C program, the first procedure to be executed is the *main* one.

The first task is to configure the ports we want to use i.e. PORT A and PORT B in terms of *input* or *output.* Any unused pins on these ports are configured as outputs. The configuration is done via the TRIS register for each port (each bit corresponds to a pin on the port); 0 represents an output, 1 an input. All of the pins on PORTA are outputs. Only RB0, the least significant bit on PORTB is an input – the rest are unused.

Next, the LED is explicitly switched OFF otherwise its state is undefined.

The code then enters a loop. The loop runs while the operand (1 in this case) is true, this technique is used to keep code repeating forever.

The LED is switched ON using a simple `if` statement. When the SW1 pin is grounded by pressing the button, the `if` statement runs and powers the LED. There is no code to turn the LED OFF again.

```
#pragma code
// Code        ////////////////////////////

void main (void)
{
        TRISA = 0x00;                                   //All pins O/P
        TRISB = 0x01;                                   //All pins but RA0 O/P

        LED = OFF;

        while (1)
        {
                if (SW1 == 0)
                        LED = ON;
        }
}
```

The code can now be compiled by either clicking *Build All* or pressing F10.

The PICkit 2 programmer provided will program the PIC and also provide you with power for the simple circuits. All the functions can be reached in the ***Programmer*** dropdown menu. ***Program*** will load the code onto the PIC, while ***Set Vdd On*** and ***Set Vdd Off*** switch the power ON and OFF respectively.

If the circuit works, move on to the next task.

If it doesn't work:

- Check that the MCLR pin is tied to 5v via a 10K resistor
- Check that the LED and SW1 are wired up correctly as per figure 2
- Check that the program code refers to the pins you are using

Task 2

This next task involves flashing your LED ON and OFF at a frequency of 1Hz. This will require you to use a software delay. The software delay functions are held in a header file called `delays.h` – you need to include this file in the same way `p18F2525.h` is included at the top of your program.

The oscillator in the PIC runs at 8Mhz, this is then divided by four, leading to a program clock speed of 2 MHz. From this you can calculate the program clock period.

You need a software delay of 0.5s to complete the task.

The header file provides 5 different delay options in multiples of clock cycles.

```
Delay1TCY      (unsigned char);              //unsigned char ×       clock cycles
Delay10TCYx    (unsigned char);              //unsigned char × 10    clock cycles
etc
```

Example:

```
void main (void)
{
        TRISA = 0x00;                                   //All pins O/P
        TRISB = 0x01;                                   //All pins but RA0 O/P

        LED = OFF;

        while (1)
        {
                LED = ON;
                Delay1KTCYx    (unsigned char);
                LED = OFF;
                Delay1KTCYx    (unsigned char);
        }
}
```

`Unsigned char` represents multiples of clock cycles and has limits.
You will need to consult `C:\MCC18\h\delays.h` for more details.

Your circuit is unchanged, so any issues you have now will be program related. If your task works, integrate it with Task 1 so the LED flashes when you push the button. Use your existing workspace to do this.

Task 3

Now you are familiar with programming the PIC, you need to start using it to control the motors on the car.

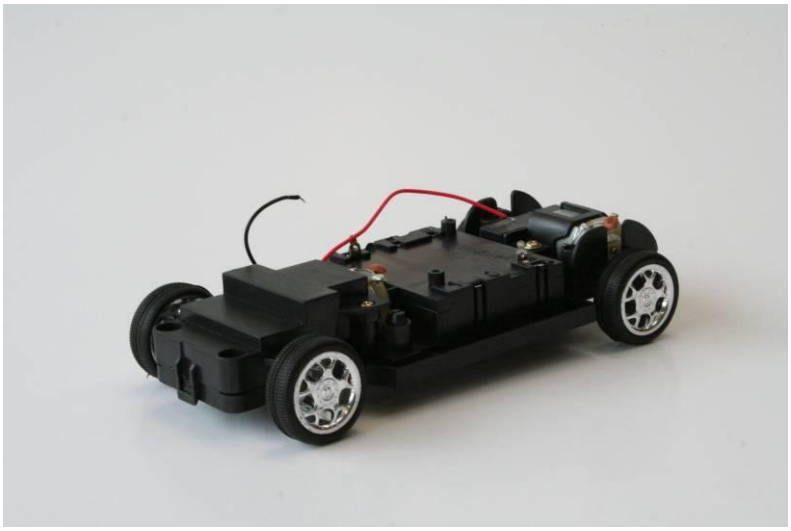


Figure 3 - The chassis of the car

The PIC doesn't have the power to drive the motors on its own. It requires some kind of interface. One component we can use is a MOSFET. Add the components in Figure 4 to your circuit. The PICkit has been removed from the figure to make it easier to read, keep it connected to your circuit. It is important you use the car batteries to power your circuit once the motor is connected as the PICkit 2 cannot supply the required current.

Datasheets have been provided to provide pin outs and other data for all of your components.

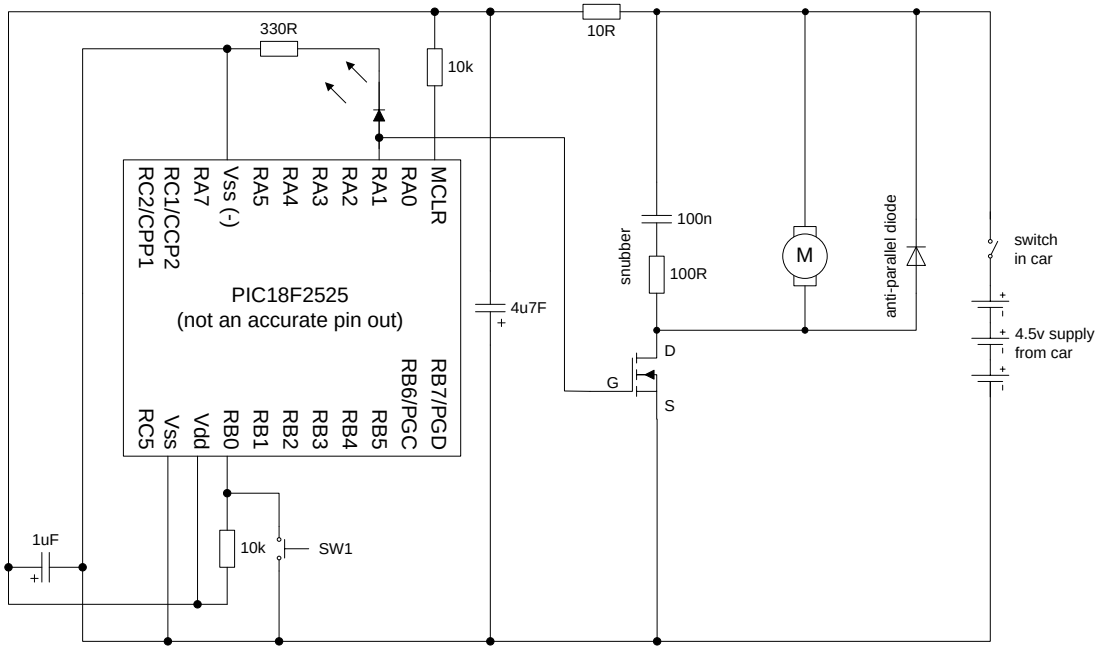


Figure 4 - driving a motor using a MOSFET

Task 4

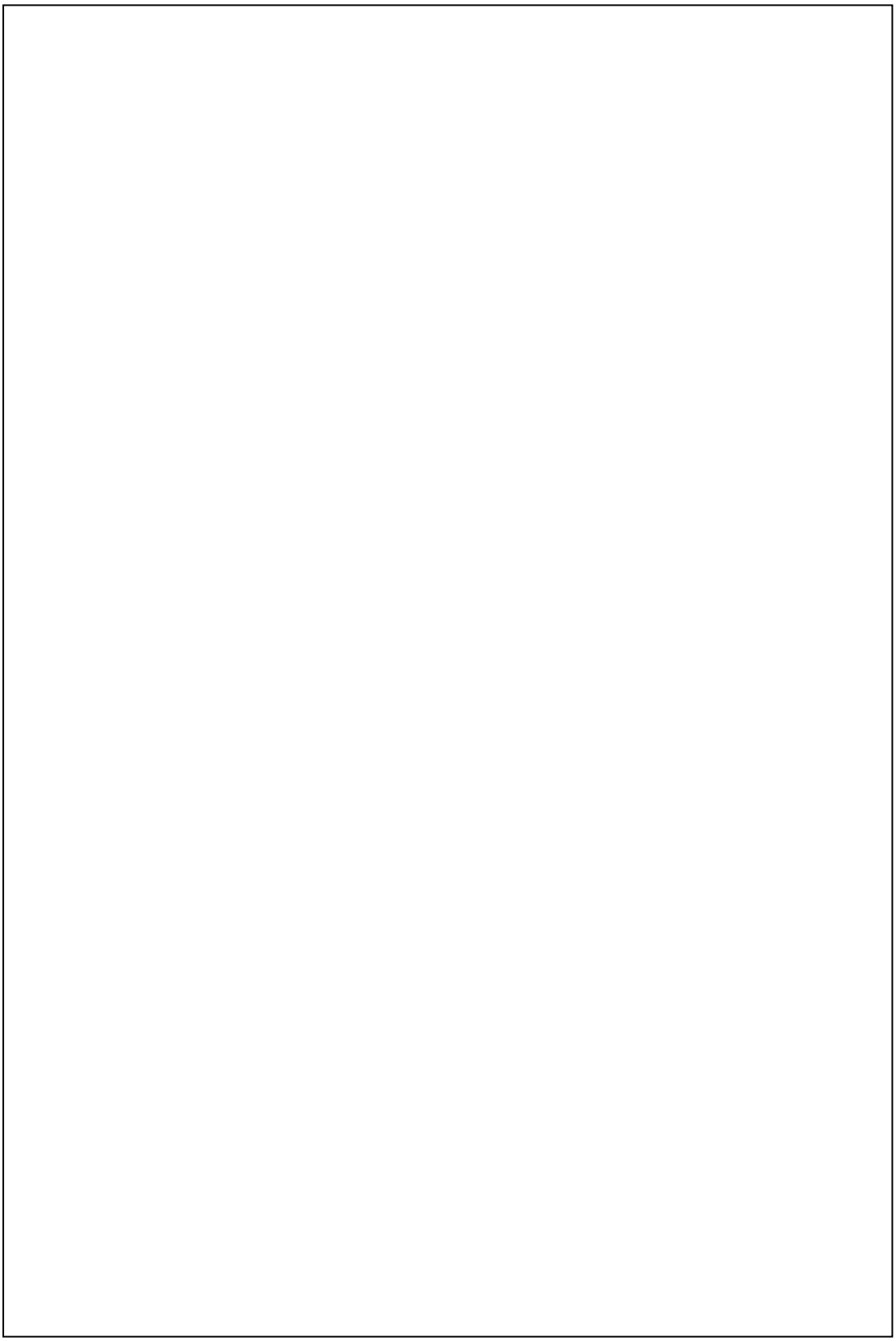
It is likely you will want the car to run backwards as well as forwards.

Relays are commonly used to reverse the polarity of circuits – mainly motors. You have been supplied two relays for this purpose. Their coils, although low current, should *not* be driven using the PIC itself. In the space below, design a circuit which allows the PIC to control the drive and direction of the motor.

Use these components:

- Relay × 1
- MOSFET × 2
- Diode × 2
- 100R Resistor × 1
- 100nF Capacitor × 1
- Motor × 1

Consider carefully your use of the protection diodes.
Draw your circuit in the following box.



Task 5

A modern method of DC motor control is Pulse Width Modulation (PWM). Full power is provided to the motor, but only for a fraction of the time (Figure 5).

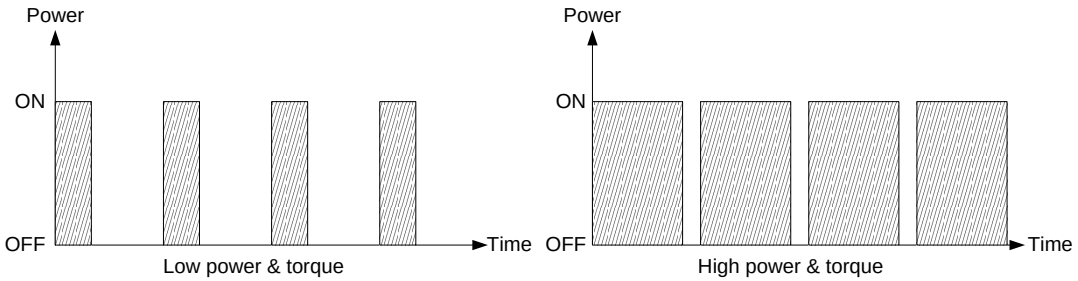


Figure 5 - Low power and High power PWM

The PIC has two PWM channels built in. Once set up, user control is achieved by changing the duty cycle, the pulsing is done in hardware. To make PWM work, *Timer2* must be set up, and the PWM period and duty cycle need to be set. These files will help you:

```
C:\MCC18\doc\periph-lib\PWM
C:\MCC18\doc\periph-lib\Timer
```

Further assistance may be provided by the PIC18F2525 datasheet. You may want to consider reducing the power going to the drive in order to prevent loss of traction, or conserve battery power. Reducing power going to the steering system will reduce mechanism and motor stress (but may stop it from working properly).

The PIC18F2525 also has an analogue to digital converter which converts a voltage into a 10-bit number in the program. The analogue inputs cover most of PORT A. It may be useful for you to link an analogue input to the duty-cycle of the PWM. The analogue input can be provided via the supplied potentiometer which can provide an adjustable potential divider fed by the supply rails.

A summary of the AD converter's function is provided in Figure 6. The conversion is started and takes some time to complete. Its completion is indicated via a flag which indicates when the conversion can be read back reliably. Each box represents a line of C code.

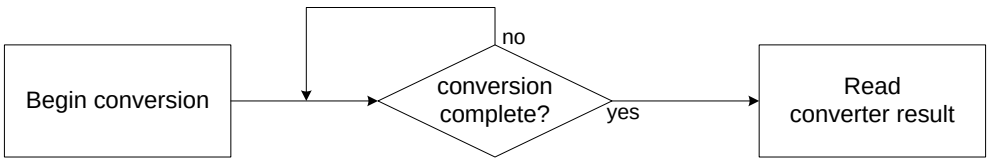


Figure 6 - summary of AD converter function

The datasheet to the PIC and the file below will help you:

```
C:\MCC18\doc\periph-lib\AD Converter
```

Task 6

There are enough components for you to make two drive circuits to operate the steering and the drive. Use what you have learned from the previous tasks to create a wired remote control for the car. Four buttons have been provided for you to do this, but it is possible to open the existing remote and fit wires to the levers.

Hint: most commercial electronics work on an *active low* basis as practiced earlier.

Task 7

You have been provided with several micro switches. Now it is up to you. Use the components you have been given, along with any other items you can get hold of to equip the car so it can make its way across the obstacle course.

You can do this using manual control if you like, but consider making the car able to detect and negotiate obstacles by itself. Ingenuity (both complex and simple), and quality of execution, will be rewarded.

Hint: the pin out on the micro switches is written on the casing e.g. *N.C. = normally closed*.

Presentation

At the end of the week, each group will make a presentation to the tutors on the progress you have made.

Your presentation should be brief and aimed at an educated, but not particularly technical, audience. There is no compulsory content, your presentation could include, but is not limited to

- Who you are
- What the project is about
- Reasons for design choices (e.g. car control method)
- How your car works
- How your method is different from other groups
- Problems you encountered
- What you would do differently next time
- What you have learned
- Demonstration

Of course, the presentation should be in English!