



CITY *of* BRISTOL
COLLEGE

College Green Centre

Faculty of Computing and Digital Communication

Documentation Standards:

**Guidelines for Writing a C++ Software Report
using an
Object Oriented Methodology**

Document No: DS02/OOSWR	Date: 1 September 2003
Issue: 02	Author(s): Ian Shere

Summary

The purpose of these guidelines are to provide an aid for writing reports on programs written in the High Level Object Orient Language of 'C++'. It describes the format required, what each section is for and how it should be written.

In the early stages of learning OOP it was suggested that you write programs in a linear format of a single file with the Class code above that of the client code (see Appendix A for template). When writing your code this way you can use the documentation standard set out in DS01/SWR (Guidelines for Writing a Software Report for Programs written in 'C') and ignore documentation of the Classes.

However, in later assignments and examples you will be required to produce 'project files' using the Turbo C++ IDE. It is for this method of developing your code, these guidelines must be followed.

The OOP software report will consist of 6 sub-reports within the overall document submitted. Each sub-report should have its own Title page and Contents List, in addition to a overall Title page and Contents List showing these sub-reports only.

Contents

Summary	i
Contents	ii
1 OOP Sub-Reports	1
1.1 Title Page	1
1.2 Contents Listing	1
1.3 Sub-Report Titles	1
2 Program Description / Introduction	1
3 Requirements Specification	2
4 Class Design Reports and Specification	2
5 Client Program Design	3
5.1 Function Breakdown Chart	3
5.2 Program Variables	3
5.2.1 Global Variables	4
5.2.2 Defined Constants	4
5.3 Externals	4
5.3.1 Header Files Required	4
5.3.2 External Functions Called	4
5.4 Function Blocks	4
5.4.1 Function 'Name'	4
5.4.2 Description of the Functions Operation	5
5.4.3 Calling function and Called Functions	5
5.4.4 Parameters and Return Values	5
5.4.5 Local Variables	5
5.4.6 Global Variables	5
5.4.7 Reference to Process Specification for Function	5
5.4.8 Process Specifications	5
6 Source Code	6
6.1 Class Header Files	6
6.2 Class Member Functions	6
6.3 Client Code	7
6.4 Main Function	7
7 Conclusion and Critique	7
Appendix A: Code Template for C++ programs in a linear format	8
Appendix B: An example of Requirements Specification	9

OOP Software Report Guidelines

1 OOP Sub-Reports

In order to provide full documentation of the development of the proposed program a number of sub-reports will be grouped under a Software Report portfolio. The portfolio and each sub-report will have a title page and contents list and following standard reporting format.

1.1 Title Page

This page should contain the following:

- Faculty of Computing and Digital Communication
- Course Name and Year
- Module Name
- Software Program Title
- Name of Author
- Date

1.2 Contents Listing

This is prepared after the rest of the report. It is a numbered list showing the section and appendices numbers on the left and starting page number on the right (remember to number all pages). It allows the reader to see the structure of the report and where to find what interests them.

1.3 Sub-Report Titles

Each of the sub-reports required are described in these guidelines, they are:

- Program Description / Introduction
- Requirements Specification
- Class Design Reports and Specification
- Client Program Design
- Source Code
- Conclusion and Critique

2 Program Description / Introduction

This sub-report should describe what the program is intended to do, i.e. its objectives, it should also state why it was chosen and the tasks it is designed to carry out. This is a top level description of what the program does to meet the objective, not an in depth description of how the program works. This sub-report should include the 'purpose statement' (see OOD Task 1) of the product. This may be a short one or two page report.

3 Requirements Specification

The requirements specification is a detailed description of what the program should do to meet the objectives and task described in section 4 above. Each of the following sub-heading should be included with the exception of the 'Figures & Diagrams', although it is anticipated that all programs will have some form of HCI and therefore a diagram of this would be necessary.

The requirement specification sub-heading are:

- Introduction
- Purpose
- Specification
- Prioritisation of Specification
- Figures & Diagrams

To help to establish the contents of the Requirements Specification a sample is attached, see Appendix B.

4 Class Design Reports and Specification

Much of the class report can be generated from the OMT CASE Tool used. It should however start with an 'Introduction' to briefly explain the design of the classes used and the necessary Class Diagrams.

If the design is completed properly using the OMT CASE Tool the all the information normally required in your 'C' Software Reports, e.g. Variable name, type and description, is included in the reports printed from the Dictionary menu. The following information must be completed during the design in the Class Editor, Attribute Editor or Operations Editor Windows for each class, attribute of each class and each class operation.

Class Editor:	Name Description Code-filename
Attributes Editor:	Name Description Constraints Code-declaration
Operation Editor:	Name Description Code-declaration

The reports **required** are listed below under the appropriate category under which they are found in the Reports dialogue box of the Dictionary menu, (with the

exception of Introduction).

Introduction

Class Diagrams:

to show Association, Aggregation, Inheritance

Item Reports in hierarchical order:

item details (brief)

item catalogue

Class specific reports:

class properties

class associations

code generation (this should be included in the report to confirm that all required data had been entered).

Diagram Reports:

Diagram Detail

Diagram Consistency Checking

5 Client Program Design

Here you include a 'standard' Software Report as a sub-report. A brief overview of the requirements of this report is included here together with additional information and guidelines. The full standard is found in document No. DS01/SWR.

5.1 Function Breakdown Chart

The functional breakdown chart (or hierarchy chart or structure chart) shows graphically the C++ functions of the client code that are used in the program and from where they are called.

Functions that are not defined within your program are External Functions and need not be shown on the chart. Also if a function calls a function more than once, then that function is only shown once.

5.2 Program Variables

Under this heading all global variables, defined constants and defined data types (e.g. structures or user defined class objects) declared in your program should be listed and described. These appear before the function main() in the source code. Any variable listed outside of any function is global. Global variables, constants declared using #define and defined data types are accessible by all functions.

Local variables, those declared within functions (including main()) are not shown in this section but under the appropriate Functional Block.

5.2.1 *Global Variables*

Here you list the data type, identifier, any default value and a short description of each variable used. The reason for this is to provide an understanding of why global variables are used by the functions and why. Warning - keep Global Variable to a minimum!

5.2.2 *Defined Constants*

Under this heading, list all the names of the constants created using #define preprocessor command together with a short description of what they represent. The use of constant variables (i.e. const int my_val = 36) is not recommended without justification.

5.3 **Externals**

As mentioned earlier, External Functions are those not defined by the programmer and therefore defined in a header file (note: this does not include Class operations). In this section those header files, declared for use by the preprocessor command #include, are listed. This should be followed by a short description of the header files purpose. Also each external function used should be listed together with a description.

5.3.1 *Header Files Required*

A listing of header files included in your source code including those created by you for your classes, for example:

stdio.h	standard input / output functions
math.h	mathematical functions

5.3.2 *External Functions Called*

Each external function used should be listed with a short description. This listing should take the format of the function prototype as given in the header file which should also be shown. Do not include commonly used functions like getch().

5.4 **Function Blocks**

This section lists and describes each user defined function in the client program (i.e. those shown in the functional breakdown chart). This, therefore excludes any external functions or user Class methods. For each function the following should be given

5.4.1 *Function 'Name'*

The full prototype declaration of the function

5.4.2 Description of the Functions Operation

A short description of the function purpose.

5.4.3 Calling function and Called Functions

List the functions that call this function and list function that are called from within it.

5.4.4 Parameters and Return Values

Any variables that are passed or returned by the function.

5.4.5 Local Variables

Identify local variables declared in the function with their data type and a description.

5.4.6 Global Variables

A listing of global variables used by the function.

5.4.7 Reference to Process Specification for Function

Each function must also be described using one of the following methods of a Process Specification and a clear reference as to the page showing it should be made. Process specifications can be in the form of either a flowchart, pseudo code or decisions table for each function.

5.4.8 Process Specifications

In this section a program flowchart, pseudo code or decision table of each user defined function is show. The flowcharts should be drawn in accordance with the standards set. Remember that a flowchart for a function should not be more than one A4 page. If you go over the page then ensure that there is only one point of entry to the function and one point of exit. The off-page connector should clearing indicate the continuing page entry process.

Notwithstanding, all efforts should be made to keep to one page, it is often an indication that if more is needed then the function itself may be two large and contain more than one concept, hence increasing coupling.

Pseudo code should be structured and non-program dependent.

Decision tables can be either limited or extended but all stage of consolidation should be shown.

6 Source Code

This sub-report should only include the source code of your program, clearly indexed and commented. The format of the program should adhere to the 'C++ Programming Documentation Standard'. Warning - make sure it is the code that is executed as errors introduced during 'tidying up' will cause suspicion of the executable file.

As your program will consist of a number of files, attention should be paid to grouping appropriate pages. Each class header file should be printed on a separate page, followed by its methods code (the .CPP file). Client code should be printed separately with each function on a separate page.

6.1 Class Header Files

Each Class declaration must have its own header file and each class should be on a new page. The layout is demonstrated in the following example.

```
/*===== Header File Information =====  
NAME OF SOURCE FILE:      ANA_IN.H  
  
PROGRAMMER NAME:         A.N.Other  
  
CLASS DECLARATION OF:    Ana_In  
  
DATE:                    3/4/02                               */  
  
#ifndef __ANA_IN_H_  
#define __ANA_IN_H_  
  
//class definition for Ana_In  
  
class Ana_In{  
private:  
    int ch, i, j;  
    int Eye, Eye_old, Eye1, Eye2;  
    char key;  
  
public:  
    Ana_In();           //default constructor  
  
    ~Ana_In();         //destructor  
  
    void setAna_Input(void); /*reads input from keyboard to select  
                             channel and generates a random value  
                             to represent input from either MARCOs  
                             'Eye' or analogue joystick. Draws a  
                             graph of this input. */  
  
};  
#endif  
//===== end of file =====
```

6.2 Class Member Functions

Each Class member function should be clearly layed out to the standards set in the 'C++ Programming Documentation Standard', using the accepted prefixes for getter

and setter methods. Constructors and Destructors must also be shown.

The layout of the code should be as for that of user functions in the client code including a 'Program Information' header and the function information section.

6.3 Client Code

Functions should be shown in alphabetical order or that in which they are declared under the Function Prototype heading. For clarity, each function should start on a new page (also gives an indication if the function is too long if it goes over a single page).

6.4 Main Function

The 'main' function should provide sufficient code to indicate the function of the program, as sort of 'road map'. For example it may contain the 'switch / case' statements that are used by the menu of a menu driven program.

7 Conclusion and Critique

In this section you should evaluate your project and identify areas you think you could improve and things you would do differently if presented with the problem again. Again this is a short sub-report often of only one or two pages.

Appendix A: Code Template for C++ programs in a linear format

```
/*  NAME OF SOURCE FILE:
    NAME OF PROGRAMMER:
    DATE:
    PROGRAM DESCRIPTION:
*/
//  LIBRARIES
//  CLASS DECLARATIONS
//  MEMBER FUNCTION DEFINITIONS
//  DEFINITIONS;
//  GLOBAL VARIABLES;
//  FUNCTION PROTOTYPES;
//===== MAIN BLOCK=====
void main(void)
{
    // Main client code goes here
}
/* *****FUNCTION*****
 *  FUNCTION NAME;
 *
 *  PARAMETERS;
 *
 *  OPERATION;
 *
 *  RETURN VALUES;
 *****
*/
```

Appendix B: An example of Requirements Specification

Faculty of Computing and Digital Communication

**HND in Computing
(Software Engineering) 2003**

Object Oriented Programming

**Software Project - Robot Simulation
Requirements Specification**

By: A. N. Other

April 2003

Contents

Introduction	1
Purpose	2
Specification	2
Prioritisation of Specification	4
Figures	4

Information Technology - Object Oriented Programming

Software Project - Robot Simulation

Requirements Specification

Introduction

The Engineering Faculty of the University of the West of England teaches the 'C' programming language to all its 1st year students. So that this module is engineering related and reflects the use of 'C' in the "real world", the module is based around the control of a small wheeled robot, known as MARCO (**M**icroprocessor **A**pplications for **R**obot **C**ontrol). Connection between the PC and MARCO (fig 1) is achieved through a specialised I/O card (PC74 I/O card) in the PC, an interface box (known as the *Rack* - fig 2), this allows devices, such as the robot to be attached to the I/O card. The *rack* provides all necessary connection to the I/O card such as digital and analogue devices (e.g. joysticks) and the robot itself via a breakout box (fig 3). A system block diagram is shown in figure 4.

MARCO hardware comprises of:

1. A pair of DC motors to give forward and backward motion and steering capabilities.
2. Two micro switches attached to 'left & right bumpers' for collision detection.
3. Two infra-red opto switches mounted on the underside with a transmit and receive circuit to allow the following of a white line.
4. A photodiode to detect a light source MARCO's 'eye'.
5. A stepper motor that allows the photodiode to rotate.
6. Two infra-red switches to detect the ends of the sweeping arc of the stepper motors.

Students are required to write a number of programs to control and use the various components of MARCO. These programs include driving MARCO with both digital and analogue joysticks (connected directly into the *rack*), make MARCO follow a white line using the two infra-red switches underneath and detecting light intensity using MARCO's 'eye'.

The programs themselves are specified in 'Laboratory Session Sheets' and cover topics such as control constructs, use of conditional operators, bit masking and scaling techniques. These Laboratory Session Sheets can be made available if required.

The various functions required to access the I/O card are written in a header file called RACK.H, thus requiring students to only make function calls to five basic functions contained in it:

- | | |
|------------------------------|---|
| 1. int ReadSwitch(void) | returns the value of the digital input of the <i>rack</i> |
| 2. void WriteMotor(int) | writes a value to the digital output of the <i>rack</i> |
| 3. int ReadAD(int ch) | - where ch is a channel number from 0 - 3,
returns a value from the analogue input of the
<i>rack</i> |
| 4. void WriteAD(int ch, int) | writes a value to the analogue output of the <i>rack</i> |

5. void ResetRack(void) resets all input and output value to zero.

The full specification of MARCO, the rack, breakout box and functions can be found in the “Handbook for Computers for Control” issued to all students and can be made available for this project if required.

Currently students can only develop their programs when in the designated ‘MARCO’ laboratory as the specialised equipment of the PC74 I/O card, the *rack* and MARCO is only available in this room.

To allow some development of programs outside the MARCO laboratory, a second header file has been written called DUMRACK.H. This gives a text interface to the *rack* functions and requires for example input from the keyboard for the value returned by the digital input of the *rack*. The use of this header file is very limited and only allows the development of the most basic program.

Purpose

The purpose of this project is to provide students with a simulation of the *rack* and MARCO so that they are able to continue to develop programs outside of the MARCO laboratory in a more realistic environment. The outcome should be the production of a header file that can be included in students ‘C’ programs that will produce a graphical representation of the *rack* on the screen, an animated robot and allow the operation through the keyboard and hopefully the mouse (to simulate the analogue input).

Specification

The following specification details the intended functions of the final header file. However, as time is limited full animation of the robot and use of the mouse may not be achieved.

1. Final program must be in a header file that can directly replace the pre-processor file of RACK.H.
2. The *rack* should be displayed on the top half of the screen and look similar to that shown in figure 2.
3. A text ‘window’ or viewport should be below the graphical image of the *rack* to allow student program output to be displayed.
4. A third ‘window’ in the bottom right-hand corner of the screen in which the robot can be animated (see suggested screen lay in fig 5).
5. All operation to be via the five function calls in RACK.H.
6. Each section of the *rack* is to react as in the ‘real world’. That is to say that the input and output LEDs of the digital ports to display red when on (1) and hollow when off (0), and the analogue output to show the red and green bar LEDs according to the output value, lowest green bar for 0 and highest red bar for 4095.

7. Digital input switches of *rack* D0 to D7 to be activated by the student selecting which joystick action and IR or micro switch is connected to which input or simply pressing a number key from 0 to 7. This could be achieved by clicking the mouse on a selection from a connection menu and then clicking on the appropriate “switch” or pressing the corresponding numeric key. This of course would require a fourth ‘window’ on the screen. Alternatively, the following connections should be given in the student handbook for when DUMRACK.H is being used.
 - 7.1 Keyboard numbers 0 to 7
 - 7.2 Cursor keys
 - up = D0
 - down = D1
 - right = D2
 - left = D3
 - 7.3 Animated robot bumpers
 - left = D4
 - right = D5
 - 7.4 Animated robot IR switches
 - left = D6
 - right = D7
8. Analogue input to A0, A1 and A2 should again if possible be accessed by selection from the connection menu. If this is not achieved then A0 should be used for the photo-diode connection and A1 and A2 for the analogue joystick connections.
9. Digital output for the DC motors will be represented as if the left motor positive connection is to P3 on the *rack*, left motor negative to P2, right motor positive to P1 and right motor negative to P0.
10. Analogue output is only displayed on the *rack* simulation by the ‘illumination’ of the coloured bar LEDs.
11. The ‘robot’ should move in accordance with the instructions of WriteMotor or WriteDA from the input received via ReadSwitch and ReadAD. If it reaches the bounds of the ‘window’ it should stop, unless ‘bumpers’ have been ‘connected’ at which time it would react accordingly.
12. For bumper simulation a four square blocks should appear in the animation window.
13. For determining whether the robot is on a light or dark surface, white lined box should appear around the robot a random distance from it.
14. For following a white line a ‘Le Mans’ track should appear in the animation window, the robot should be automatically placed on the track.
15. For the detection of light, a random number should be generated and the incremented by the + key and decremented by the -. The bounds of the number to be predefined with this program.
16. The header file should be named as RACKSIM.H.

Prioritisation of Specification

For the purpose of this project, it has been decided that a working model of the rack display run from a menu system and operating from the keyboard as opposed to a 'student program' will be the main priority.

Figures

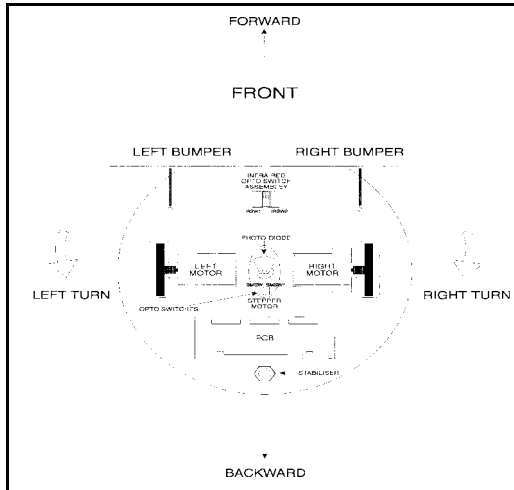


Figure 2 - MARCO

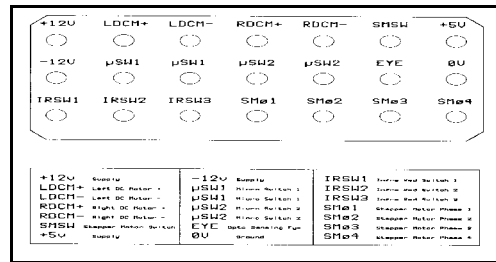


Figure 3 - Breakout Box

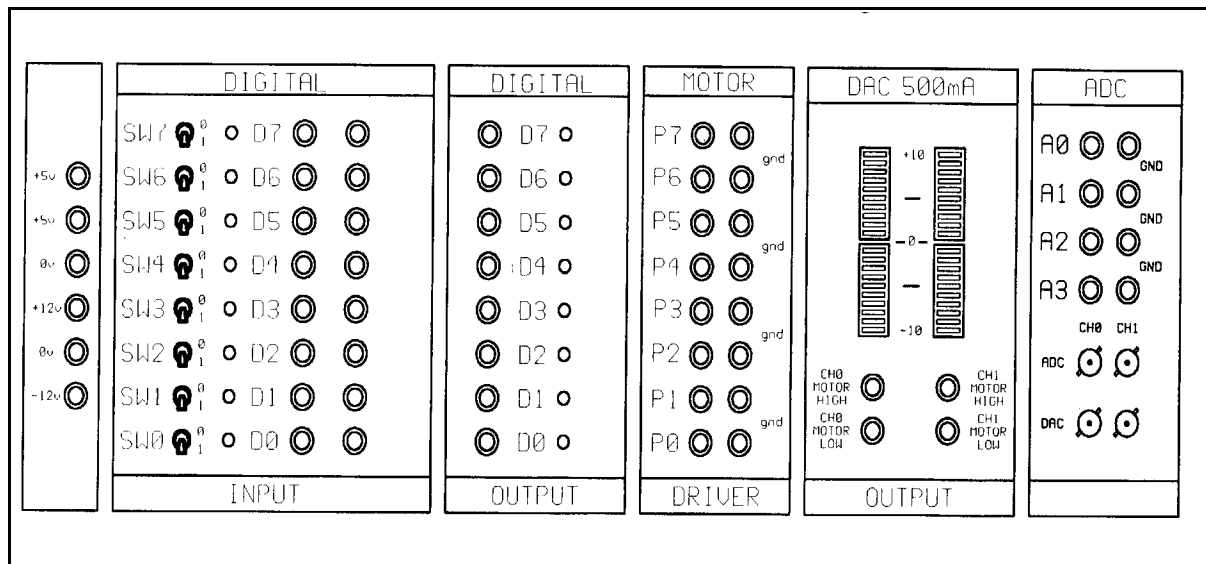


Figure 2 - The Rack

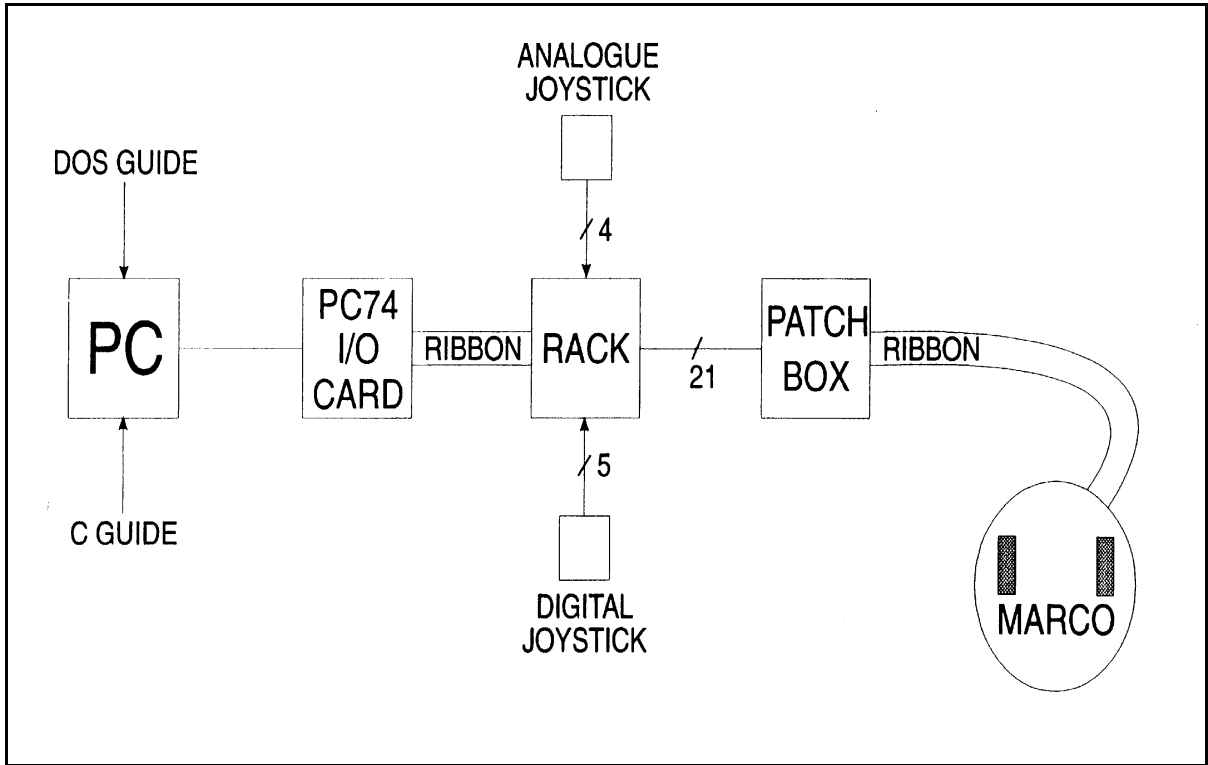


Figure 5 - System Block Diagram

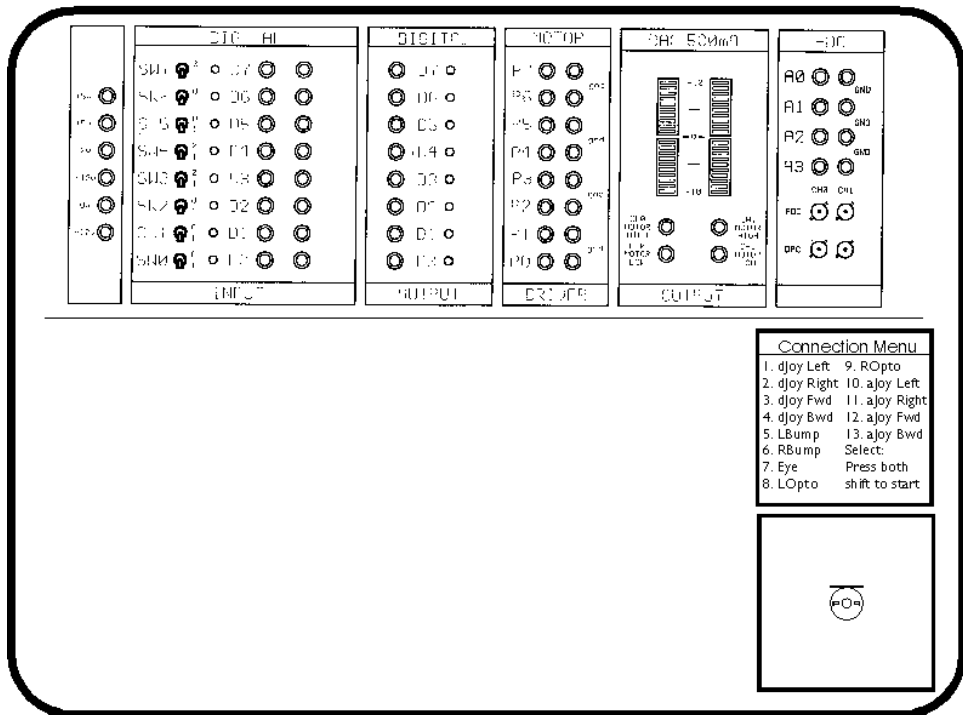


Figure 6 - User Interface Screen Layout