## 0.1 Module Requirements: Menu Navigation

The Menu Navigation module aims to allow end user to easily interface with the system in both the set-up of run time behaviour as well as the entering and exiting of run time sequences. As per specification, the Menu Navigation module interacts with user through the use of an LCD output and user inputs to:

1. Switch between run-time modes Manual, Assisted, and Full Auto
2. Change important global variables such as 'Max Speed' and 'IR Sample Rate' where it is appropriate to do so
3. Enter and exit a 'Motor On' phase safely

To clarify the use of terms in this section:

- **Motor On Mode** refers to mode in which motor is turned on and user input will affect the real time movement of the robot
- **Menu Mode** or **Motor Off Mode** refers to the mode the system takes when motors are off and menu variables can be changed
- **Run-time mode** refers to modes Manual, Assisted, Full Auto, and Factory which dictate behaviour when motors are both on and off. This is referred to in code as **menu_ref_1**. On the LCD this is displayed on the top line
- **Submenu** refers to the variables that can be altered in Manual and Factory Mode and are displayed on the bottom line of the LCD. These variables include Max Speed, Yaw Rate, IR sample rate, etc. This is referred to in code through **menu_ref_2**

### 0.1.1 Functional Requirements

**Inputs** Inputs mainly come in the form of user inputs through analogue signals from joysticks and falling edge triggers from buttons.

Analogue signals are received through the PORT A bits 1 and 2, representing the y and x axes respectively. They are converted from analogue to digital and then parsed through the module.

The buttons control the more significant status changes of the system and in being push buttons need to be more deliberate inputs than the joystick (which may easily be accidentally altered). Trigger signals are created when a button is pressed which creates a falling edge by shorting to ground and generating a low value in PORTB bits 0 or 1. Originally, this used a hardware interrupt on PORT B, but due to integration difficulties was reimplemented within the wider loops.

An additional receive input (relevant to this module, though others are also present, see Communications) takes the IR raw data periodically and displays

this signal to the LCD while in Assisted or Full Auto mode and while motors are on. This gives the user additional feedback while operating in these modes that would not have been relevant for Manual Mode.

**Process** The module consists of while loops that continue to loop contingent upon the statuses of certain set flags, namely RUN and Menu_ref_1, to be discussed later. These loops though programmed for different states, all perform a similar process of:

- Read ADC values
- Dependent on values, perform an action
- Update Commander display and Robot if a change has occurred

In the interest of only making large changes when it is safe to do so (e.g. changing motor status, changing run type), the system uses an indirect or two-step command system. When one of the buttons are pressed a flag unique to each button is raised. This flag will cause the program to change the status when it is safe to do so (i.e. outside of long process such as LCD writing and radio transmission). This is opposed to a method in which an interrupt immediately causes a status change which may cause errors in timing, LCD output, and transmission while also allowing artificial delays to override user input (see Interfaces). Much like joystick signal processing, when a change is found to have occurred, LCD and robot are updated.

A detailed description of state changes and function is given in the Conceptual Design section.

**Outputs** The module generates two outputs, the LCD user display and the transmitted signal. Both are covered in more detail in their respective sections.

### 0.1.2 Non-Function (Quality of Service) Requirements

**Performance** The Menu Navigation runs in a way where its performance (and speed) is high enough as to have a negligible impact on the rest of the system, especially when there are more fragile modules (such as radio communication and PWM generation) that would be affected adversely by mistiming and interruptions at play. The primary operation of the menu navigation is independent of the rest the system (especially due to the indirect command sequence outlined later) and runs sufficiently fast due to the low levels of computational requirements. It uses primarily 'if' statements to parse inputs accordingly, which while extending code length, has very low computational time. Large quantities of memory is accessed indirectly through pointers and pointers-to-pointers that result in more efficiently compiled machine code and simpler code readability.

When the time does come for a command of any form to be transmitted, and thus break independence from the system, the module makes sure to only do it

at a safe time, and to only do it when absolutely necessary (See Communications and LCD).

**Interfaces** The software design is one which prevents human error resulting in erroneous hardware behaviour. Users are given ample time and warning of system changes and can safely break from undesired modes of behaviour.

Upon powering on, a welcome message is displayed. During this time necessary initialisation delays are covered and prevents premature user activity; as a design choice, the use *should not* be able to make changes of any kind immediately after the system is powered on.

Part of creating intuitive interfacing between user and product is to respond to human response time and the need for tolerance in human error and imprecision. As a result, delays were implemented for three reasons: prevention of overly rapid menu, debouncing, and safety. They were implemented as follows:

- A delay occurred with each increment or decrement of a global variable (such as 'Max. speed') and sub-menu change (see Menu_ref_2)to prevent these values from changing too rapidly for the user to respond or accurately select desired value
- A delay was implemented with each button press or joystick move while in menu (in menu, joystick functions more like button than variable voltage source) to ignore any additional signals that may be produced
- Whenever the system entered a 'Motor On' mode, an initial delay with respective start-up message 'Giddy up' is initialised to prevent movement for 2 seconds. This allows user time to ready themselves. When motors are turned off and the system is asked to return to its Menu state, motors are turned off *before* a 2 second delay with message 'Whoa!' is initiated to prevent user accidentally turning motors back on, which may happen if user for example panics.

Furthermore, by design the user is restricted from changing run-time modes while the robot can also move. Two separate modes were created, a Motor On mode, in which user input only influences real time movement (as well as both starting and stopping), and Menu Mode, in which user can select run time mode and change submenu variables.

**Design Constraints** Ideally the module makes extensive use of hardware interrupts for buttons to reduce computational time in loops created from the constant polling of button status and this was the original implementation, however it was found that in migrating from the PIC18F452 to the PIC18F4520 that hardware interrupts simply would not function during integration. This certainly creates a minor constraint on ideal functionality and future work should look into resolving this.

## 0.2 Conceptual Design: Software Module Menu Navigation

### 0.2.1 Using Menu Navigation - an outline

When User powers on the commander, they are greeted with a welcome screen 'Bow for I am Charlemagne'. They by default enter the Menu mode in Manual mode and the LCD will display after the welcome 'MANUAL' in the first line and 'Max speed: 100' in the second. If they move the joystick left and right, they will increment and decrement the value of Max Speed from 0-100. If the joystick button (Button 1) is pressed, the commander will switch to Factory mode. Here the user may also move the joystick up and down to scroll through the submenus and thus choose what value they wish to change (Max Speed, Yaw rate, etc.). If the joystick button is pressed again, they will enter Assisted mode where no value changes can be made, likewise if they press it again to enter Auto mode. If the button is pressed once more they return to Manual mode.

If at any point the Motor Button is pressed (button 2) then the Robot will be switched on to the run time mode the commander was in. Upon pressing this button, a start up delay and message is shown ('Giddy up') after which motors are activated and motor on run-time behaviour commences, during which the phrase 'Taste my lance!' is shown. If the motor button is pressed any time after the initial press, motors are turned off and an end delay and message ('Whoa!') is initiated during which user forcibly has no control. While motors are on, user cannot make any changes to Factory mode variables nor change Run-time modes. By default if motors are turned on in Factory mode, Robot enters motor on run time in Manual.

### 0.2.2 Design Rationale

**General Design** The focus of the design was intuitive user control; we wanted to minimise the number of possible inputs, bringing it down to only three main sources of user inputs (four with the power switch) which are the analogue joystick inputs (x-y), joystick push button, and separate push button. We thought about how the user held the commander and which inputs had room for error. From this we established that the act of changing variables and changing submenus had the most room for error, changing of run-time modes to have the second, and turning the motors on and off (switching between Motor On Mode and Menu Mode) to have the least room for error. Thus analogue joystick inputs were assigned to the changing of variables and submenus, the joystick button (controlled by the same (left) hand) were assigned to Run-time mode changes, and finally the motor switching was assigned to the separate push-button which was controlled by the right hand.

The use of buttons allow for simplified interaction, but limits the depth of the user input. Buttons can only represent a single action and for this design asks the commander to 'go to the next menu/mode'. This created some limitation

in meeting the specification of being able to go to any run-time mode from any other run-time mode as our design acts as a mono-directional scrolling interface going from Manual-Factory-Assist-Auto-Manual. However we determined that due to the low number of run-time modes available, this would not be perceived by the end user as an inconvenience. The use of a separate motor button (Button 2) allows for the decoupling of tuning and movement from the most significant mode change of the system, that is to put the system into and out of a movement state. It minimises confusion for the user, performing in a robust manner no matter how unorthodox user input is, and prioritises safety above all else.

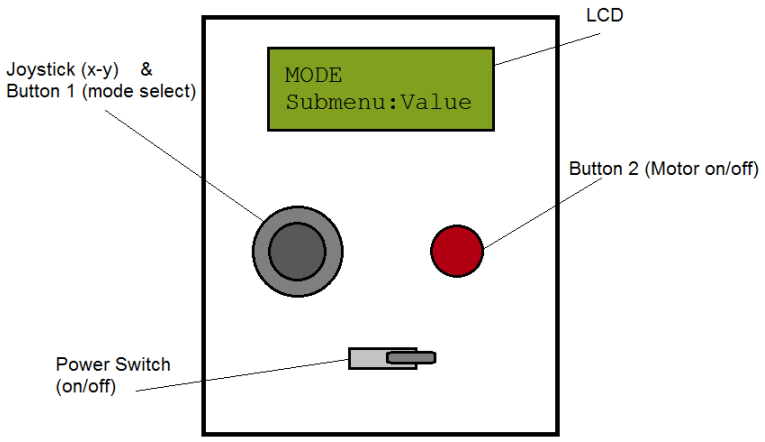


Figure 1: Controller Design

**Display of Information** The similar mentality of keeping things simple and intuitive was used in displaying the information. While in Menu mode, the only three things that would need to be displayed to the user are current run-time mode, submenu, and submenu value. This was also influenced by limited LCD space. When in a Motor On mode, the LCD is used to inform the user when the motors are turning on, when they are on and when they are turning off to give the user indication of status without visual feedback from the robot. Further information is given in LCD.

We also believed that due to the nature of Assisted mode, there needed to be more information given to the user in regards to the robot's parallel distance from tilt, especially if the user wishes to operate the robot without the robot in sight. As a result, when in Assisted robot will send back IR data to be displayed on the LCD.

### 0.2.3 Function

**Variables** The structure of the module is a large while loop that runs as long as the commander is powered with smaller while loops representing the separate modes that the commander runs in. These smaller while loops keep running contingent upon three main variables:

- RUN - Determines status of Motors (on/off) and the respective behaviour
- menu_ref_1 - Integer determines Run-time Mode (0-3)
- menu_ref_2 - Integer that determines submenu and submenu variable to be operated on (0-7)

Both menu_ref values work in a circular manner, if they are decremented/incremented beyond a minimum/maximum value, they will become the maximum/minimum value.
Important submenu values and their corresponding string (e.g. "Max Speed:") are stored in arrays 'values[ ]' and 'stringtab[ ]' respectively, with each position of each array corresponding to a certain variable. For example, user adjusts speed by adjusting values[0], and then prints it with a combination of values[0] and stringtab[0]. Within the code, the exact variable to be operated on is determined by menu_ref_2 which dictates which value of each array is being accessed.

e.g.

```
menu_ref_2=0;
values[menu_ref_2]=50;
//MAX SPEED = 50
```

Using this form of referencing simplifies the code and naturally allows for the incremental and decremental style of our code.

**Pseudocode** The program continuously polls current status, anticipates the user input expected for each status, and executes the necessary response. Function `LCD_disp(i,j)` is explored in more detail in LCD, but generally, it takes in two arguments, normally menu_ref_1 and menu_ref_2 as i and j, and writes to the LCD the run time mode on the first line and the submenu and submenu value on second

```
/*By default set menu_ref_1 to MANUAL, menu_ref_2 to MAX SPEED and RUN=0*/
intialistion();
welcome();
LCD_disp(menu_ref_1, menu_ref_2);

Main loop:
  if (RUN==0) GOTO Menu_mode;
  else GOTO Motor_On_mode;
```

```
/*User pushes left/right on joystick to decrement/increment a value and up/down
to scroll through submenus*/

Menu_mode:
  if (Menu button has been pressed)
  {
  Next menu_ref_1;
  clear menu button flag;
  }

  if (menu_ref_1==MANUAL)
  {
    if (joystick pushed LEFT/RIGHT && values[menu_ref_2]>0)
    {
        values[menu_ref_2]=values[menu_ref_2]-/+5;
    }
    LCD_disp(menu_ref_1, menu_ref_2);  //Update LCD
    delay;                             //Delay to prevent flickering
  }

  else if (menu_ref_1==FACTORY)
  {
    if (joystick pushed LEFT/RIGHT && values[menu_ref_2]>0)
    {
        values[menu_ref_2]=values[menu_ref_2]-/+5;
    }
    else if (joystick pushed UP/DOWN)
    {
      Next/Previous submenu;              //menu_ref_2 ++/--
    }
    LCD_disp(menu_ref_1, menu_ref_2);
    delay;
  }
  else
  {
    display runtime mode only;            //Assisted and Auto have no user
                                          //inputs in Menu
  }

  if (Motor button has been pressed since start of loop)
  {
  RUN=1;
  clear motor button flag;
  }
  GOTO Main Loop;
```

```
Motor_On_mode:
  //Simply run in chosen run-time mode until motor button is pressed again
  motor_on(menu_ref_1);
  if (Motor button has been pressed since start of loop)
  {
  RUN=0;
  clear motor button flag;
  }
  GOTO Main Loop
```

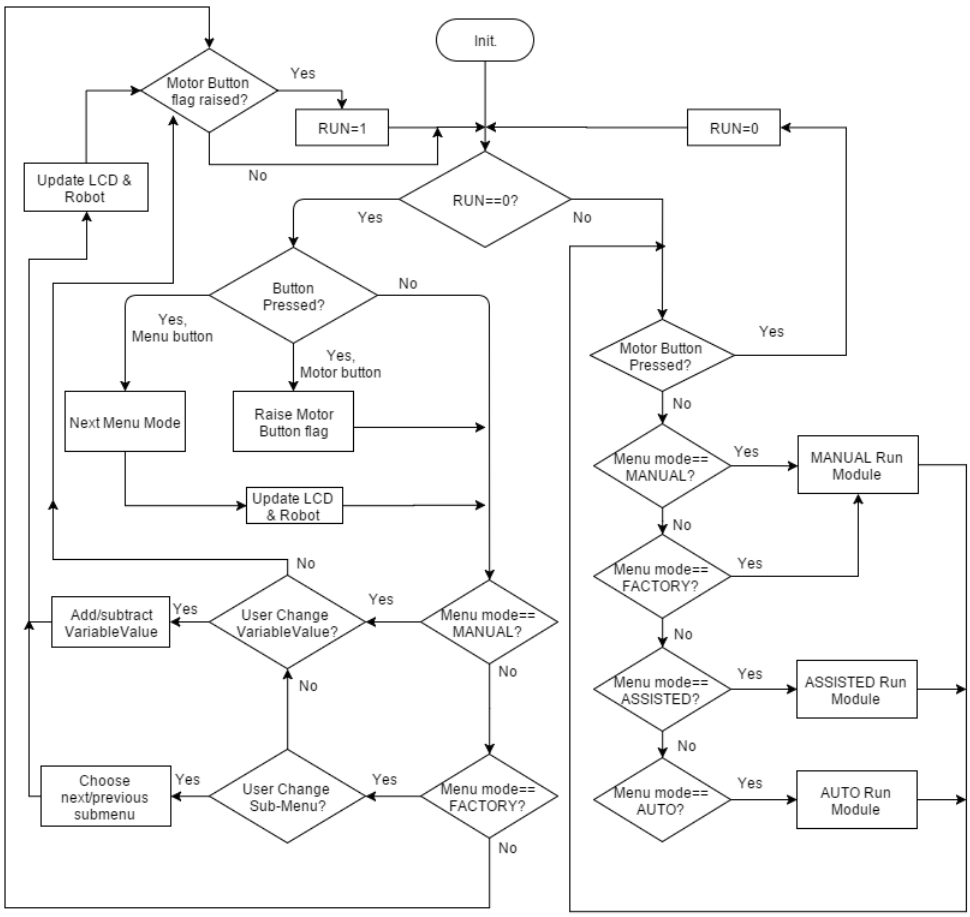


Figure 2: Menu Navigation Flow Chart

### 0.2.4 Constraints on Module X Performance

State any constraints that may prevent the design from satisfying its requirements.