```
#define START PORTD.B0
#define AUTOM PORTD.B1
#define EXEC PORTD.B2
#define RUNLED LATD.B4
#define AUTOLED LATD.B5
#define EXECLED LATD.B6
#define FAULTLED LATD.B7

// LCD module connections
sbit LCD_RS at RC4_bit;
sbit LCD_EN at RC5_bit;
sbit LCD_D4 at RC0_bit;
sbit LCD_D5 at RC1_bit;
sbit LCD_D6 at RC2_bit;
sbit LCD_D7 at RC3_bit;

sbit LCD_RS_Direction at TRISC4_bit;
sbit LCD_EN_Direction at TRISC5_bit;
sbit LCD_D4_Direction at TRISC0_bit;
sbit LCD_D5_Direction at TRISC1_bit;
sbit LCD_D6_Direction at TRISC2_bit;
sbit LCD_D7_Direction at TRISC3_bit;
// End LCD module connections

char keypadport at PORTB;

void init_ports(void){
//  ADCON1 = 0x0D;     // could be here or in init_adc sets analog/digital
  TRISA = 0x03;        // AN0 and AN1 inputs.
  TRISB = 0x0f;        // half inputs half outputs for keypad (not critical)
  TRISC = 0x40;        // serial TX bit 7, RX input bit 6, bits 0-5 for LCD
  TRISD = 0x07;        // 3 button inputs, 4 leds outputs
}

void init_serial(void){
  SPBRG = 7;           // for the BRGH = 0  at a 10MHz Osc
// SPBRG = 31;         // alternate for the BRGH = 1  at a 10MHz Osc
  TXSTA = 0x20;        // enable transmitter, BRGH = 0
  RCSTA = 0x80;        // enable serial port
  BAUDCON = 0x00;      // all zero - particularly BR16 bit for 8 bit baud gen
}

void init_adc(void){
  ADCON0 = 0x01;     // adc ON (default is channel 0)
  ADCON1 = 0x0D;     // could be here or in init_ports
  ADCON2 = 0x81;     // minimum Fosc/N is N=8 for a 10MHz PIC18F4420
}
```

```
int read_x(void){
  unsigned int h, l;  // bytes to hold high and low values of adc result
  long value;
  ADCON0 = 0x01;       // set channel to 0 corresponding to RA0
  ADCON0.GO = 1;       // or ADCON1 = ADCON1|0b00000010;
// this next line of code will loop endlessless in the mikroC debugger
// as the adc operation is not simulated- comment out in mikroC debugger
  while(ADCON0.DONE == 1)  // or could be ADCON0.DONE (or .GO) & 0b00000010
  {
  }                  // do nothing - or coulbd be while (ADCON0.DONE == 1);
  h = ADRESH;                // in mikoC debugger stop here to enter values
  l = ADRESL;                // next line done differently in read_y
  value = h*256 + l;          // h should be no more than 3
  value = value - 512;           // subtract half full scale gives +/- 511
  value = (value * 400) / 819;   // 0.8*1024 = 819.2 range of input
                              // 400 is out range ie. +/-200
                              // value must be long to fit 511*400 = 204400
  return (int)value;             // strictly should cast (convert) long to int
}

int read_y(void){
  unsigned int h, l;  // bytes to hold high and low values of adc
  long value;
  ADCON0 = 0x05;        // set channel to 1
  ADCON0.GO = 1;        // or ADCON1 = ADCON1|0b00000010;
// this next line of code will loop endlessless in the mikroC debugger
// as the adc operation is not simulated- comment out in mikroC debugger
  while(ADCON0.DONE == 1)  // or could be ADCON0.DONE (or .GO) & 0b00000010
  {
  }                  // do nothing - or coulbd be while (ADCON0.DONE == 1);
  h = ADRESH;             // in mikoC debugger stop here to enter values
  l = ADRESL;             // next line done differently in read_x
  value = (h<<8) + l;       // must use ( ) as + has higher precedence than <<
  value = value - 512;           // subtract half full scale gives +/- 511
  value = (value * 400) / 819;   // 0.8*1024 = 819.2 range of input
                              // 400 is out range ie. +/-200
                              // value must be long to fit 511*400 = 204400
  return value;             // without cast (convert) should work
}

int start(void){

  if(START == 0)      // active low push button
    return 1;
  else
    return 0;
}
```

```
int auto_m(void){
  if(AUTOM == 0)      // active low push button
    return 1;
  else
    return 0;
}

int exec(void){
  if(EXEC == 0)     // active low push button
    return 1;
  else
    return 0;
}

void indicator(unsigned short n, unsigned short on_off){
  if (n == 0)
    RUNLED = on_off;
  if (n == 1)
    AUTOLED = on_off;
  if (n == 2)
    EXECLED = on_off;
  if (n == 3)
    FAULTLED = on_off;
}

char cmdstr[15] = "uses array ";
char cmdstr2[15] = "uses pointer ";

void command(char cmd[]){              // passing string array type ie name
  int i = 0;
  while (cmd[i] != '\0')
  {
    if(TXSTA.TRMT == 1)
    {
      TXREG = cmd[i];
      i++;
    }
  }
}

void command2(char *cmd){              // passing pointer to string
 while (*cmd != '\0')
  {
    if(TXSTA.TRMT == 1)
    {
      TXREG = *cmd;
      cmd++;
    }
   }
}
```

```
void main() {
  int x, y;
  char str[9], kp;
  init_ports();
  init_serial();
  init_adc();
  LCD_init();
  LCD_Out(1,1,"UAV Control");

  while(1)
  {
    x = read_x();
    y = read_y();
    if(start())           // if start returns a 1
    {
      IntToStr(x,str);
      LCD_out(2,1,str);
      indicator(0,1);
      indicator(2,0);
    }

    if(auto_m())          // if auto_m returns a 1
    {
      IntToStr(y,str);
      LCD_out(2,9,str);
      indicator(1,1);
      indicator(2,0);
    }

    if(exec())          // if exec returns a 1
    {
      indicator(0,0);
      indicator(1,0);
      indicator(2,1);
      command(cmdstr);
      command2(cmdstr2);
    }

    kp = Keypad_Key_Press();   // OshonSoft simulator does not detect keys using this
    if(kp)
      indicator(3,1);
    else
      indicator(3,0);
  }

}
```