PROGRAMMING ARDUINO

The sketch written for the Arduino microprocessor is developed and uploaded by using the Arduino IDE software. It checks the actuators states and set the indicators or active the stepper motor. Much code is derived from the Arduino code examples published on web, so it's open code.

It starts with the constant declarations and variable assignments:

```
/*
  Remote_control.ino
  To manually control towards front-panel and towards USB remote control the uCF·SPS module,
  a Syringe Pumping System developed by the
  Applied Electrochemistry Research Group on the University of Huelva

  Programmed by Juan Daniel Mozo Llamazares, José Ignacio Otero Bueno and Angel García Barrios
  February, 2014
 */

// pin assignation (constants).
// give it a name:
const int botonONOFF = 2;              // button Run/Stop
const int ledONOFF = 3;                // Red LED Start flag
const int ledLOAD = 4;                 // Yellow LED LOAD flag / R-L motion pin
const int Paso = 5;                    // motor clock pin(one step)
const int pinHalf = 6;                 // full/half step pin
const int botonLOAD = 7;               // LOAD button
const int FinVaciar = 8;               // Dispense limit switch
const int FinLOAD = 9;                 // LOAD limit switch
const int Buzz = 10;                   // buzzer
const int ledPRUEBA = 13;              // Arduino ON BOARD led
const int Veloc = A0;                  // analogical potential read pin (speed)

// other constants
const int Prueba = LOW;                // Demo mode (activa el pin 13 en lugar del motor)
const long debounceDelay = 50;         // stores Debounce time in millisec

// variables
unsigned long previousTime = 0;        // stores previous step time in microsec
unsigned long currentTime = 0;         // stores actual time in microsec
unsigned long interval = 500;          // stores interval-between-steps in microsec
unsigned long previousTimeVolt = 0;    // stores voltmeter' previous refresh time in microsec
long lastDebounceTime = 0;             // stores last LED turnover time
int ledState = LOW;                    // stores led's state
int runMotor = LOW;                    // stores motor's state (HIGH = run)
int runLOAD = LOW;                     // stores running direction (HIGH = LOAD = LEFT)
int botonState = LOW;                  // stores limit switches state
int botonState1 = LOW;                 // stores ONOFF button state
int previousBotonState1 = LOW;         // stores ONOFF button previous state (to Debounce)
int botonState2 = LOW;                 // stores LOAD button state
int previousBotonState2 = LOW;         // stores LOAD button previous state (to Debounce)
int volt = 0;                          // stores analog speed setting (0 - 1024)
char serialData = 0;                   // stores data read from USB (one byte 0 - 255)
int remote = LOW;                      // stores control active mode (to speed's question)
```

A set of remote commands is provided to enable the remote control via USB connection. Almost all are one character commands for easy programming.

```
/* USB command set
fn,n    == Freq (hl = millisec 0 to 1024)    replace manual panel control (hi and low byte required)
g       == Go run                            start motor to difuse
l       == Load                              start motor to withdraw
s       == Stop                              stop motor
r       == Remote                            enable remote control (disable speed reading)
m       == Manual                            disable remote control (enable speed reading)
u       == Up                                increases speed
d       == Down                              decreases speed
*/
```

Then there is the `setup` routine, therein it sets the digital pins as input or output and the serial port is initialized.

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pins as input or output.
  pinMode(ledONOFF, OUTPUT);
  pinMode(ledLOAD, OUTPUT);
  pinMode(Paso, OUTPUT);
  pinMode(pinHalf, OUTPUT);
  pinMode(Buzz, OUTPUT);
  pinMode(ledPRUEBA, OUTPUT);
  pinMode(botonONOFF, INPUT);
  pinMode(botonLOAD, INPUT);
  pinMode(FinVaciar, INPUT);
  pinMode(FinLOAD, INPUT);

  // initialize the serial port
  Serial.begin(9600);
}
```

The `loop` routine is ever run. It starts by reading the actual time (in microseconds) and loading it in the `currentTime` variable. This value is used to compare below with the motor step's period. As time goes by, the routine is checking several things.

```
// the loop routine runs over and over again forever:
void loop() {
  // first read the actual time                    ------------------------------------------------
  currentTime = micros();
```

First it checks for a command waiting on the serial port. If any, the variables values changes depending on the command received by a switch/case structure. When the command has a valid meaning and has acted accordingly, a handshaking is returned echoed the same command.

```
  // check if data is waiting on serial port       ------------------------------------------------
  while (Serial.available() > 0) {
    serialData = Serial.read();
    switch (serialData) {
      case 's':                                    // stop
        runMotor = LOW;
        runLOAD = LOW;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'm':                                    // manual mode
        remote = LOW;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'r':                                    // remote mode
        remote = HIGH;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'l':                                    // load
        runMotor = HIGH;
        runLOAD = HIGH;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'g':                                    // go run
        runMotor = HIGH;
        runLOAD = LOW;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'u':                                    // up volt (speed)
        volt += 5;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'd':                                    // down volt (speed)
        volt += -5;
        Serial.print("Ok, ");                      // Handshaking Ok
        Serial.println(serialData);
        break;
      case 'f':                                    // frequency (speed)
        int HByteRead = Serial.parseInt();         // read the message completely
```

```
        int LByteRead = Serial.parseInt();
        LByteRead = constrain(LByteRead, 0, 255);   // avoid out-of-range values
        HByteRead = constrain(HByteRead, 0, 3);
        volt = LByteRead + (HByteRead * 256);       // compose two Bytes to calculate volt

        Serial.print("Ok, ");                       // Handshaking Ok
        Serial.print(serialData);
        Serial.print(HByteRead,DEC);
        Serial.print(',');
        Serial.print(LByteRead,DEC);
        Serial.print(',');
        Serial.println(volt,DEC);
        break;
    }
    int serialDatatmp = Serial.read();              // empty serial port (CR)
  }
```

For ‘f’ command (sets a new value of frequency) it must be included the new setting, replacing the analog read from the 10-bits A/D converter build in the Arduino Duemilanove. A 10-bits number reaches 1024 but serial port supports only bytes, so two bytes are needed which must be composed and constrained.

Then it checks if the control panel’s buttons are pressed. To prevent unwanted multiple activation, a debouncing procedure is implemented. A single push generates only a toggle in the variables setting. The On/Off button toggles the `runMotor` variable but, if `runMotor` results `LOW` the `runLoad` goes also `LOW` and motor always stops. The Load button toggles the `runLoad` variable but, if `runLoad` results `HIGH` the `runMotor` goes also `HIGH` and motor starts.

```
  // check control panel knobs                    ------------------------------------------------
  // check ONOFF button setting --------------------
  int reading = digitalRead(botonONOFF);
  if (reading != previousBotonState1) {             // with Debounce
    lastDebounceTime = millis(); }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != botonState1) {
      botonState1 = reading;
      if (botonState1 == HIGH) {                    // changes the runMotor value if button is pushed
        runMotor = !runMotor; }
      if (runLOAD == HIGH && runMotor == LOW) {     // if loading and run stop all
        runLOAD = LOW; }}}
  previousBotonState1 = reading;
  // check LOAD button setting ---------------------
  reading = digitalRead(botonLOAD);
  if (reading != previousBotonState2) {             // with Debounce
    lastDebounceTime = millis(); }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != botonState2) {
      botonState2 = reading;
      if (botonState2 == HIGH) {                    // changes the runLOAD value if button is pushed
        runLOAD = ! runLOAD; }
      if (runLOAD == HIGH) {                        // check if goes LOAD and start the motor too
        runMotor = HIGH; }}}
  previousBotonState2 = reading;
```

Then the micro-switches are checked. If they are pressed a `HIGH` is read and we must stop the motor by setting `LOW` at `runMotor`. If load switch is pressed, `runLoad` goes `LOW` also. An audio alarm is activated to alert than the user attention is required. A different tone is used to distinguish both events.

```
  // check the limit switches                     ------------------------------------------------
  // check the DISPENSE limit switch setting -------
  botonState = digitalRead(FinVaciar);
  // motor must be running and dispensing to check the limit switch
  if (botonState == HIGH && runMotor == HIGH && runLOAD == LOW) {
    tone(Buzz,250,500);                             // audio alarm
    runMotor = LOW; }                               // if limit switch is pressed stop the motion


  // check the LOAD limit switch setting -----------
```

```
  botonState = digitalRead(FinLOAD);
  // motor must be running and loading to check the limit switch
  if (botonState == HIGH && runMotor == HIGH && runLOAD == HIGH) {
    tone(Buzz,523,250);                              // audio alarm
    runLOAD = LOW;                                   // if limit switch is pressed stops and load
    runMotor = LOW; }
```

Once they have been checked all the actuators, the LED are setting to reflect the variables. Remember that `runLoad` affects to LED and also to the direction setting on motor driver. To fully configure the motor driver the step-mode pin must be sets. We have programmed a half-step mode for infusion motion (more resolution) and a full-step mode for withdraw motion (more speed) so the `pinHalf` always has the inverse setting of the `runLoad` variable and no need to program anything else. As `pinHalf` setting is code-controlled, other configurations are possible but more code is required.

```
  digitalWrite(ledONOFF, runMotor);                  // write RUN setting on LED
  digitalWrite(ledLOAD, runLOAD);                    // write LOAD setting on LED and set direction
  digitalWrite(pinHalf, !runLOAD);                   // sets HALF step to dispense and FULL to load
```

It's time to check if we have to move the motor. But first we have to calculate the period of motor movement. In remote mode the setting is read from serial port and you don't have to do anything. In manual mode we must read the analog port to check the potentiometer setting. The reading is done twice every second. In addition we write the reading in the serial USB port, so the remote control software knows the data.

In any case, the `interval` between motor steps is calculated from the `volt` variable to infuse fluids. To withdraw, a fixed `interval` is assigned to get a quick filling of syringes.

A Demo mode is provided, in which there is full operation but, instead of activating the motor windings, a flash is generated in the Arduino onboard LED, which is associated to pin 13. In that case a bigger interval is needed to see the blink properly. This mode is advisable while debugging routine, eliminating the higher power consumption the electronics can be powered directly from the USB port.

```
  // calculate motor rotation speed                  ------------------------------------------------
  if (remote == LOW) {                               // manual mode
    volt = analogRead(Veloc);                        // read potentiometer value each 500 millisec
    // check if refresh time is passed (500 ms)
    if (currentTime - previousTimeVolt >= 500000L) {
      previousTimeVolt = currentTime;
      Serial.print('v');                             // write in serial USB to PC synchronization
      Serial.println(volt); }}

  if (Prueba == LOW) {
    interval = (1025 - volt) * 250L;                 // calculates motor steps interval in microsec
    if (runLOAD == HIGH) interval = 3500L; }         // full speed to load
  else {                                             // in demo mode calculates bigger intervals (LED)
    interval = (1024 - volt) * 1000L;
    if (runLOAD == HIGH) interval = 25000L; }
```

Once the `interval` has been calculated, it is compared with the time since the last check only if the motor must be activated.

If the interval has passed, the current time is stored in the variable `previousTime` and a pulse is generated on pin `Paso`. The motor driver CLK input is flange active on rising edge, so it is advisable to stay the pin `LOW` and generate an `L-H-L` sequence.

In Demo mode a blink routine is programmed by inverting the `ledState` variable and writing it on pin 13 every `interval`.

```
  // check if elapsed time is bigger than calculated interval to motor step up ----------------------
  if (currentTime - previousTime >= interval && runMotor == HIGH) {
    previousTime = currentTime;                    // stores actual time (reset time)
    if (Prueba == HIGH) {
      ledState = !ledState;                        // turn the LED OFF and ON every interval
      digitalWrite(ledPRUEBA, ledState); }
    else {
      digitalWrite(Paso,HIGH);                     // set a pulse on Paso (Motor goes on)
      delay(1);
      digitalWrite(Paso,LOW); }
  }
}
```

After the last checking the `loop` routine ends and it starts over again.