System and Code Documentation

Installation and User Guide:

The Fi-Ventilator runs on Arduino based code, where the 4 inputs are regulated using control knobs; these 4 inputs are tabulated below.

Knob	1	2	3	4
Parameter	P_{insp}	FiO_2	Breathing Rate	IE Ratio
Range (units)	0 to 40 (cmH_2O)	21 to 100 (%)	6 to 40 (Breaths/Minute)	1 to 6 (N/A)

Table 1: System Knob Parameters and Ranges

Each knob controls an important factor of the ventilator system, and the parameters for each knob are clearly defined in *Table 2*.

Parameter	***Definition***
P_{insp}	The lung inspiration gauge pressure, where the gauge means relative to atmospheric pressure
FiO_2	The fraction of inspired oxygen into the lungs
Breathing Rate	The number of breaths per minute
IE Ratio	The ratio of inspiratory to expiratory time defined as E/I. For example, an IE Ratio of 2 means the expiration time is twice the inspiration time.

Table 2: Definition of Knob Parameters

In addition, there is a manual control for the expiratory pressure for the exit circuit; this valve can be adjusted between 0 and 25 cmH_2O gauge pressure. To use the control module, the operator must turn the dials related to each parameter to put the ventilator in ideal operating conditions for the patient.

The internal circuitry and code are all developed and produced together and combined into the control module. The Arduino hardware contains connections to the DC motor encoder, LCD display and all of the electronic components of the device, such as sensors and solenoid valves; as a result, the system is built so that the operator only needs to be powered on and off. The LCD display will show the relative parameters, such as Tidal Volume, while the knobs can be adjusted to alter these parameters based on the patient's needs. Therefore, no installation is necessary from the operator, as the system comes ready to use.

Arduino Code:

```
/*
* Name: Ventilator_Preformance.ino
* Authors: Luca Scotzniovsky, Clemente Guasch, Imran Matin
* Description: This program contains the controller code for a ventilator.
* TODO Notes: Motor specifications and pressure sensors unknown.
*             Pins are not determined.
*             Interrupt Syntax: attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
*             Add support for 2nd pressure sensor.
*             Alarms:
*             Buzzer to sound alarm
*             Button to deactive alarm or internally turned
*             Display:
*             Input: (4 potentiometers, 2 pressure sensor)
*             Output: (4 potentiometers, 1 tidal volume)
*
*             Hardware:
*               2/3 LCD displays, 4 potentiometers, 2 pressure sensors, 1 buzzer, 1 DC
motor with encoder, 2 solenoid valves
*/


// include mathematical functions
#include <math.h>
// include the library code for LiquidCrystal display
```

```
#include <LiquidCrystal.h>



// Initalize pins for sensors and potentiometers
const int PINSP_PIN = A1;
const int IE_RATIO_PIN = A3;
const int FIO2_PIN = A4;
const int BPM_PIN = A5;
const int PATIENT_PRESSURE_PIN = A6;



// Volume flow rate from the O2 chamber to piston in cm^3/sec
const double VOLUME_RATE_O2_TO_PISTON = 12015.4145;
// Piston Outer Diameter in cm
const double OD = 10.16;
// Piston Inner Diameter in cm
const double ID = 9.398;
// Cross-sectional Area of Piston in m^2
const double AP = M_PI * pow(((OD + ID)/4), 2) * pow(10,-4);
// TODO: Time to pause program for in microseconds, subject to change
const int T_PAUSE = 100;



/*
* Potentiometer knob values.
*/
// Initializing inspiratory pressure
int pInsp = 0;
// Initializing insp-exp ratio (E/I)
int ieRatio = 0;
// Initializing oxygen fraction (lowest is at 21% for pure air) (Fraction of inspired
oxygen)
int FiO2 = 0;
// Initializing breaths per minute (respiratory rate)
int bpm = 0;



// Inspiratory time
double tInsp = 0;
```

```
// Expiratory time
double tExp = 0;
// store pressure reading from patient pressure sensor
int patientPressure = 0;
// flag to open/close patient solenoid valve
bool patientSolValve = 0;
// Height the piston moves from O2 tank
double hO2 = 0;
// Volume going out of O2 tank into piston.
double vO2 = 0;
// Time duration for O2 Solenoid valve to be open
unsigned long tSolO2 = 0;
// Flag to open and close O2 solenoid valve
bool O2SolValve = 0;
// Velocity of piston for bringing the piston back to inital positon
long vPiston = 0;
// The volume of air that enters the lung
double tidalVolume = 0;
// variables to handle timing
unsigned long startTime = 0;
unsigned long currTime = 0;
unsigned long timeElapsed = 0;


// TODO
// flag to handle motor torque, -1 = counterclockwise, 0 = stopped, 1 = clockwise
// motor mechanics unknown, this is a temporary solution
int motorflag = 0;
// Distance that piston travels once the lungs have filled up
double delta_h = 0;


// initialize the library with the numbers of the interface pins
// TODO: Choose correct pins on arduino and LCD
LiquidCrystal lcd(7,8,9,10,11,12);


/*
* Function Name: setup
```

```
* Description: Setup code to run once for initalizing ventilator control values upon
power up.
*/
void setup() {
 // set up the LCD's number of columns and rows:
 lcd.begin(16, 2);
 lcd.clear();

 // Set patient pressure sensor pin to INPUT
 pinMode(PATIENT_PRESSURE_PIN, INPUT);

 // Set potentiometer knobs mode to INPUT
 pinMode(PINSP_PIN, INPUT);
 pinMode(IE_RATIO_PIN, INPUT);
 pinMode(FIO2_PIN, INPUT);
 pinMode(BPM_PIN, INPUT);
}



/*
* Function Name: loop
* Description: Code that will run repeateadly once setup is completed.
*/
void loop() {
 // Sets the cursor to col 0 and row 0
 lcd.setCursor(0,0);
 // start motor
 motorflag = 1;



 /*
  * Current state: All solenoid valves closed, motor torque is on, bag is inflated
  */



 // read in potentiometer analog input
 // TODO: Confirm if need to transform value to match pInsp
 bpm = analogRead(BPM_PIN);
```

```
 ieRatio = analogRead(IE_RATIO_PIN);
 patientPressure = analogRead(PATIENT_PRESSURE_PIN);
 pInsp = convertAnalogToCMH20(analogRead(PINSP_PIN));

  //  Inspiratory  time;  need  to  determine  actual  BPM  and  IE  using  potentiometer
relations
 tInsp = calculateTInsp(bpm, ieRatio);
 // Determining expiratory time given BPM and Tinsp
 tExp = calculateTExp(tInsp, ieRatio);

 // Begin bag compression if condition satisfied
 if (patientPressure >= pInsp) {
   // open patient solenoid valve and
   patientSolValve = 1;

   // air leaves piston and enters lung until lung capacity reached
   startTime = micros();
   do {
     currTime = micros();
     timeElapsed = currTime - startTime;
     patientPressure = analogRead(patientPressureAnalogPin);
   } while ((patientPressure < pInsp) || (timeElapsed < tInsp));

   // close patient solenoid valve and turn off motor
   patientSolValve = 0;
   motorflag = 0;

   // TODO: Motor specifications unknown
   // calculate delta_h displacement of motor shaft

   // calculate amount of air that entered lungs
   tidalVolume = calculateTidalVolume(delta_h);

   // Display Tidal Volume
   lcd.setCursor(0, 0);
   lcd.print("TidalVol: ");
   lcd.print(tidalVolume);
   lcd.print("mL");
```

```
  delay(T_PAUSE);



  /*
      * Current state: All solenoid valves closed, motor torque is off, bag is
compressed
   */



  // TODO: Confirm if need to transform value
  FiO2 = analogRead(FIO2_PIN);

  // Calculating height the piston moved
  hO2 = calculateHO2(FiO2, delta_h);

  // calculate duration for O2 Solenoid valve to be open
  tSolO2 = calculateO2SolenoidTime(hO2);

  // Open O2 solenoid valve for tSol2 seconds to allow O2 into bellow
  if (tSolO2 > 0) {
    // Open O2 Solenoid valve
    O2SolValve = 1;

    // Keep O2 Solenoid valve open for correct duration
    startTime = micros();
    do {
        currTime = micros();
        timeElapsed = currTime - startTime;
    } while (timeElapsed < tSolO2);

    // Close O2 Solenoid valve
    O2SolValve = 0;
  }



  /*
    * Current state: All solenoid valves closed, motor torque is off, bag is partially
inflated with O2
   */
```

```
   // calculate velocity of piston for bringing the piston back to inital positon
   vPiston = calculateVPiston(delta_h, hO2, tSolO2, tExp);

   // TODO: Move in reverse direction at vPiston velocity
   // motorflag = -1;
   // motorVelocity = vPiston;

   // Continue moving piston until reaches inital position
   // do {
   //   motorPos = analogRead(MOTOR_POS_PIN);
   // } while (motorPos != 0);

   // Turn motor off
   // motorflag = 0;

   delay(T_PAUSE);

   /*
     * Current state: All solenoid valves closed, motor torque is off, bag is fully
inflated
    */
 }
}



/*
* Function Name: convertAnalogToCMH20
* Description: Converts pInsp analog reading into units cmH20.
*/
int convertAnalogToCMH20(int pInsp) {
 return (pInsp * 40)/1023;
}

/*
* Function Name: calculateHO2
```

```
* Description: Calculating height the piston moves based on FiO2 and delta_h input. In
units centimeters.
*/
double calculateHO2(int FiO2, double delta_h) {
 return delta_h/.79 * (FiO2 - .21);
}

/*
* Function Name: calculateO2SolenoidTime
* Description: Calculating duration for O2 solenoid valve to be open in microseconds.
*/
int calculateO2SolenoidTime(double hO2) {
 // calculates volume going out of O2 tank into piston
 double vO2 = hO2 * AP;
 return (vO2 / VOLUME_RATE_O2_TO_PISTON) * pow(10,6);
}

/*
* Function Name: calculateVPiston
* Description: Calculating velocity of piston for bringing the piston back to inital
positon. Brings in flow from the atmosphere.
* In units centimeters per second.
*/
long calculateVPiston(double delta_h, double hO2, unsigned long tSolO2, double tExp) {
 // Piston stops when the reference height is reached
 double h_air = delta_h - hO2;
 return h_air / (tExp - tSolO2 - (T_PAUSE*2));
}

/*
* Function Name: calculateTInsp
* Description: Determining Inspiratory time in microseconds.
*/
double calculateTInsp(double bpm, double ieRatio) {
 return (60 / (bpm * (1 + ieRatio))) * pow(10,6);
}

/*
* Function Name: calculateTExp
```

```
* Description: Determining Expiratory time in microseconds.
*/
double calculateTExp(double tInsp, double ieRatio) {
 return (tInsp * ieRatio) * pow(10,6);
}

/*
* Function Name: calculateTidalVolume
* Description: calculate the amount of air entering the lungs in millileters.
*/
double calculateTidalVolume(double delta_h) {
 return (AP * pow(10, 4)) * delta_h;
}
```