This code is provided as supplementary information for the manuscript

E. Castro-Camus et al. *Simple ventilators for emergency use based on Bag-Valve pressing systems: Lessons learned and future steps*. (UNDER REVIEW 2020); Preprint available at https://doi.org/10.1101/2020.04.29.20084749

it should be used in accordance with all the instructions and disclaimers described in the manuscript. Its use as part of medical devices is strongly discouraged and has not been approved by any relevant regulatory agency or ethics committee for use or trail on humans to the date of this release. This code comes with no warranty, has not been tested for long term reliability, it is only provided for the purpose of further development and investigation.

The code is designed for the ArduinoUNO platform.

```
//==========START: DEFINE PINS AND VARIABLES ==========
#define pwmPin 5 // pin for the DC motor power control
#define motor_in1 4 // motor direction control
#define motor_in2 7 // motor direction control
#define ledPinOne A2 // LED 1
#define ledPinTwo A3 // LED 2
#define buzzerPin 6 // Buzzer
#define modePin 8 // Mode switch
#define pressurePin A0 //pressure sensor
#define potentiometerPin A1 //potentiometer
#define interruptPinOne 2 // Upper (stretched) endswitch
#define interruptPinTwo 3 // Lower (squeezed) endswitch
#include <LiquidCrystal_I2C.h> // load display library
LiquidCrystal_I2C lcd(0x27, 16, 2); // address and size of the display

unsigned long timestamp_squeezed = 0; // counter of time (refreshed when squeeze button
pressed)
unsigned long T0_inhale = 0; // inhale time (measured)
unsigned long timestamp_stretched = 0; // Counter of time (refreshed when release button
pressed)
unsigned long T0_exhale = 0; // exhale time (measured)
long dev_inhale = 0; // deviation of T0_inhale from the desired value
long dev_exhale = 0; // deviation of T0_exhale from the desired value
int count_stretched = 0; // counter (increasing when stretched switch pressed - important for
corect strat of the code)
int count_squeezed = 0; // counter (increasing when squeeze switch pressed - important for
corect strat of the code)
int count_temp = 0; // temporary counter; increased when upper endswitch not reached in time -
> used in endswitch_check()

unsigned long paceMillis = 0; // time when the display was last refreshed when changing the
pace
unsigned long alarmMillis = 0; // time when the alarm sequence was last through
unsigned long printMillis = 0; // time when the values were last printed out
long numb_time = 1000;  // set the time for which endswitch is ignored after pressed
volatile unsigned long last_millis = 0; // time of the last stretched event
volatile unsigned long last_millis0 = 0; // time og the last squeezed event
unsigned long last_millisPEEP = 0; // time of the last PEEP maintained event

int potValue; // variable to store potentiometer value => set pace
int pace = 10; // set pace BPM => set by potentiometer
int pace_old = pace; // old set pace
int pace_measured = 0; // measured/actual
double coef = 1; // coeficient for adjusting the voltage in the loop - stat value should be 1!
int motorValue = 255; // maximum motor value
int motorSlow = motorValue; // value which is being set in the back-fed loop
int motorTemp = 0; // temp value which is increasing during squeezing
int factor = 1; // factor for 2:1 or 1:1 mode
bool modeValue; // which mode is chosen
float squeezing_time = 1; // squeezing timer
float tau = 1; // period of squeezing depending on the set pace

volatile bool stretched = false; // true when stretched
volatile bool squeezed = false; // true when squeezed
bool PEEPed = true; // true when PEEP maintained
bool inhaled = false; // true when inhale was detected
bool alarm = false; // true when alarm event happened

int press_read = 0; // value read out from the pressure sensor (arb. units)
```

```
float pressure = 0; // pressure value (mbar)
int max_p = 0; // maximum pressure in a cycle
int min_p = 0; // minimum pressure in a cycle
//==========END: DEFINE PINS AND VARIABLES ==========
//Choose if to control peep actively and set values;
bool doPEEP = true; // imporatnt - mechanical PEEP valve might still be needed
int PEEPvalue = 5; //The value of PEEP (mbar)
int InhaleValue = 1; //Pressure drop indicating inhale (mbar)
//==========START: ADDITIONAL FUNCTIONS ==========
void squeezedInterrupt() { // ROUTINE FOR UPPER SWITCH -> WHEN SQUEEZED
  if ((millis() - last_millis0) >= numb_time){
    bool in1_old = digitalRead(motor_in1);
    digitalWrite(motor_in1, !in1_old); //Turn the direction of the motor
    digitalWrite(motor_in2, in1_old); //Turn the direction of the motor
    count_temp = 0; // for what_if scenarios
    squeezed = true;
    last_millis0 = millis();
  }
}
void stretchedInterrupt() { // ROUTINE FOR LOWER SWITCH -> WHEN RELEASED
  if ((millis() - last_millis) >= numb_time){
    if (count_stretched > 0){
      bool in1_old = digitalRead(motor_in1);
      digitalWrite(motor_in1, !in1_old); //and brake the motor
    }
    stretched = true;
    last_millis = millis();
  }
}
void mode(bool modeValue){
  if (modeValue == LOW){ // MODE 1:1
    if (alarm == false){
      analogWrite(ledPinOne, 255);
      analogWrite(ledPinTwo, 0);
    }
    factor = 1;
  }
  else{
    if (alarm == false){ // MODE 1:2
      analogWrite(ledPinOne, 0);
      analogWrite(ledPinTwo, 255);
    }
    factor = 2;
  }
}
void buzzer(bool alarm){
  if (alarm == true){
    analogWrite(pwmPin,0); // stop the motor!
    motorValue = 0;
    coef = 0;
    motorSlow = 0;
    if (millis() - alarmMillis < 250){
      digitalWrite(buzzerPin, HIGH);
      analogWrite(ledPinOne, 255);
      analogWrite(ledPinTwo, 0);
    }
    else if (millis() - alarmMillis < 500){
      digitalWrite(buzzerPin, LOW);
      analogWrite(ledPinOne, 0);
      analogWrite(ledPinTwo, 255);
    }
    else{
      alarmMillis = millis();
    }
  }
}
void endswitch_check(){
  if (millis() - timestamp_stretched > 1.5*60000/pace_old/(factor+1) && timestamp_stretched >
timestamp_squeezed){// If squeezing is taking 1.5x longer than set => release - exhale!
    bool in1_old = digitalRead(motor_in1);
    digitalWrite(motor_in1, !in1_old); //Turn the direction of the motor
    digitalWrite(motor_in2, in1_old); //Turn the direction of the motor
    squeezed = true; // go into UPPER EDNSWITH REACHED part
    count_temp = count_temp +1; // counter for how many times in a row the upper endswitch was not
reached (see Interrupt() for reset)
    last_millis0 = millis();
  }
  if (count_temp > 2){ // if upper endswitch was not reached 2x in a row
```

```
    alarm = true; // trigger the alarm!
  }
  //if one of the end switches was detected twice in a row!!! One could be broken!
  //abs(count_stretched-count_squeezed) < 10000 for the case if the counter exceeds maximum
intiger!
  if(abs(count_stretched-count_squeezed) > 1 && abs(count_stretched-count_squeezed) < 10000){
    // one of the counters was not detected!
    alarm = true;
  }
  //if neither of the endswitches was pressed for more than 15 s
  if (millis() - timestamp_squeezed > 15000){
    alarm = true;
  }
  if (millis() -timestamp_stretched > 15000){
    alarm = true;
  }
}
//==========END: ADDITIONAL FUNCTIONS ==========
//==========START: setup() ==========
void setup() {
  #include <math.h>
  pinMode(pwmPin, OUTPUT);
  pinMode(motor_in1, OUTPUT);
  pinMode(motor_in2, OUTPUT);
  pinMode(interruptPinOne, INPUT_PULLUP);
  pinMode(interruptPinTwo, INPUT_PULLUP);
  pinMode(modePin, INPUT_PULLUP);
  pinMode(potentiometerPin, INPUT);
  pinMode(pressurePin,INPUT);
  pinMode(ledPinOne, OUTPUT);
  pinMode(ledPinTwo, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(interruptPinOne), squeezedInterrupt, HIGH);
  attachInterrupt(digitalPinToInterrupt(interruptPinTwo), stretchedInterrupt, HIGH);
  lcd.init();
  lcd.backlight();
  digitalWrite(motor_in1, LOW);
  digitalWrite(motor_in2, HIGH);
  Serial.begin(9600);
}
//==========END: setup() ==========
//==========START: loop() ==========
void loop() {
  if (count_stretched == 0){
    pace_old = pace;
    analogWrite(pwmPin,motorValue); // start the motor
    lcd.setCursor(0, 0);
    lcd.print("     eARM     ");
    lcd.setCursor(0, 1);
    lcd.print("  HELLO WORLD!  ");
  }
  else{
    endswitch_check();
    if (millis() - timestamp_stretched > 0 && timestamp_stretched > timestamp_squeezed &&
alarm == false){
      //While squeezing (timestamp_stretched>timestamp_squeezed) increase motor power with
time
      squeezing_time = (float)(millis()-timestamp_stretched);
      tau = (float)(60000/pace_old/(factor+1));
      motorTemp = motorSlow + 0.5*(255-motorSlow)*(sin(squeezing_time*PI/(3*tau)));
      analogWrite(pwmPin,motorTemp);
    }
  }
  //==========START: UPPER EDNSWITH REACHED => Squeezed completely ==========
  if(squeezed == true){
    modeValue = digitalRead(modePin); // mode change has an effect only after the next squeeze
    mode(modeValue);
    pace = 30 - (int)map(potValue, 0, 1023, 0, 22);// map the pace between 8-30
    pace_old = pace;
    //==========START: adjust the motor speed for next squeeze ==========
    T0_inhale =  (long)(millis() - timestamp_stretched);//note the time it took to squeeze
    if (count_stretched > 0){ //if the lower switch has been already touched at least once
      dev_inhale = tau - T0_inhale; // tau calculated before squeezed was true
      if (abs(dev_inhale) > 0){
        if (abs((float)dev_inhale/T0_inhale) > 0.6){
          coef = abs(coef - coef * 0.36*2*dev_inhale/abs(dev_inhale));
        }
        else{
```

```
          coef = abs(coef - coef
*2*(float)dev_inhale/T0_inhale*abs((float)dev_inhale/T0_inhale));
          if (count_temp == 2){coef = coef -
4*(float)dev_inhale/T0_inhale*abs((float)dev_inhale/T0_inhale);}
        }
        float motorSlow_old = motorSlow;
        motorSlow = motorValue * coef;
        if (motorSlow_old == motorSlow){ //if motorSlow did not change despite deviation (due
to limited "resolution")
          motorSlow = motorSlow - 1*dev_inhale/abs(dev_inhale); //change for minimum value
          coef = (float)motorSlow/(float)motorValue;
          }
        if (motorSlow > 255){
          motorSlow = 255;
          coef = 1;
        }
        else if (motorSlow < 10){
          motorSlow = 10;
          coef = (float)motorSlow/(float)motorValue;
        }
      }
    }
    //==========END: adjust the motor speed for next squeeze ==========
    squeezed = false; // reset the condition
    count_squeezed = count_squeezed +1; // note that the switch was reached
    timestamp_squeezed =  millis(); // note the time
    PEEPed = false; //important to reset the condition for PEEP
  }
  //==========END: UPPER EDNSWITH REACHED => Squeezed completely ==========
  //==========START: LOWER EDNSWITH REACHED => Stretched completely ==========
  if (count_squeezed > 0){//if upper switch has been reached at least once
    if (stretched == true){
      T0_exhale =  millis() - timestamp_squeezed; // note stretch time when endswitch reached
      if(count_stretched > 0){ // if lower switch has been reached at least once
        dev_exhale = (factor*T0_inhale-T0_exhale);
        last_millis = millis(); // since the motor is topped; ignore any potential signal from
lower switch
        if(dev_exhale <= 0 or inhaled == true){
           // if time for neccessary inhale passed or inhale was detected=> start squeezing
          if (digitalRead(motor_in2) == digitalRead(motor_in1)){
            //if motor is blocked - unblock and continue in the other direction
            bool in2_old = digitalRead(motor_in2);
            digitalWrite(motor_in2, !in2_old); // continue in the other direction
          } //if not blocked, it is already moving in the correct direction
          analogWrite(pwmPin,motorSlow); // start squeezing with set initial speed
          timestamp_stretched = millis(); // note when inhaling started: important for timer
of inhale
          count_stretched = count_stretched +1; // increase the counter of this switch
          inhaled = false; // important to reset the condition
          stretched = false; // iimportant to reset the condition to exit LOWER EDNSWITH
REACHED
          //=======start: modify display =======
          lcd.setCursor(0, 0);
          lcd.print("                ");
          lcd.setCursor(0, 0);
          String StringRow1 = "";
          pace_measured = (int)round((60000/(T0_inhale + T0_exhale + dev_exhale)));
          if (modeValue == LOW) {StringRow1 = "I:E 1:1 ";}
          else {StringRow1 = "I:E 1:2 "; }
          StringRow1 = StringRow1 + pace_measured + "/" + pace + "BPM";
          lcd.print(StringRow1);
          if ((pace == pace_old)){
            String StringRow2 = "Pmx/mn ";
            StringRow2 = StringRow2  + max_p + "/" + (int)round(pressure) + "mbar";
            lcd.setCursor(0, 1);
            lcd.print("                ");
            lcd.setCursor(0, 1);
            lcd.print(StringRow2);
            max_p = 0;
          }
          //=======end: modify display =======
        }
        //==========START: maintaining PEEP and checking for inhale ==========
        else{//if the condition to start squeezing again has not been met yet
          if (doPEEP == true){ // if PEEP control is on
            analogWrite(pwmPin,motorValue); //full speed
            if (pressure <= PEEPvalue or PEEPed == false){
              if (PEEPed == false){//if peep has not been maintained yet
```

```
                if (pressure < PEEPvalue && pressure >0){
                  // if pressure dropped below the set PEEPvalue
                  if (digitalRead(motor_in2) == digitalRead(motor_in1)){//if motor is blocked
                    bool in2_old = digitalRead(motor_in2);
                    digitalWrite(motor_in2, !in2_old); // continue in the other direction
                    last_millisPEEP = millis();
                  }
                }
                else{ // if pressure is higher than PEEPvalue => block the motor
                  if (digitalRead(motor_in2) != digitalRead(motor_in1)){ //if motor not
already blocked
                    if (millis() - last_millisPEEP > 25){// some buffer time
                      bool in2_old = digitalRead(motor_in2);
                      digitalWrite(motor_in2, !in2_old); // block the motro again - maintain
PEEP
                      PEEPed = true; // maintain PEEP only once per cycle. if pressure < 0 =>
support inhale
                    }
                  }
                }
              }
              else{ // if PEEP has been maintained
                if (pressure <= PEEPvalue-InhaleValue){ //if pressure drops again =>
spontaneous inhale
                  if (millis() - last_millisPEEP > 500){
                    inhaled = true; //start squeezing to suport inhailing!
                  }
                }
              }
            }
          }
          else{// if PEEP is not beeing maintained - check only for potential inhale and
triger
            if (pressure <= -InhaleValue){
              inhaled = true;
            }
          }
        }
      //==========END: maintaining PEEP and checking for inhale ==========
      }
      else{ //if this endswitch has not been ever reached
        analogWrite(pwmPin,motorValue);
        timestamp_stretched = millis();
        stretched = false; //reset the value for next occurance
        count_stretched = count_stretched +1; //increase the counter
        last_millis = millis(); // make sure that the numb time is adjuted correctly
      }
    }
  }
  //==========END: LOWER ENDSWITCH WAS REACHED ==========
  buzzer(alarm); // see buzzer void below
  //==========START: READING AND PRINTING OUT VALUES ==========
  potValue = analogRead(potentiometerPin);
  pace = 30 - (int)map(potValue, 0, 1023, 0, 22);
  if (pace != pace_old){ //When changing the set pace: show current value.
    if (millis() - paceMillis > 200){
      String StringRow2 = "SET PACE: ";
      StringRow2 = StringRow2 + pace + " BPM";
      lcd.setCursor(0, 1);
      lcd.print(StringRow2);
      paceMillis = millis();
    }
  }
  press_read = analogRead(pressurePin);
  pressure = ((float)press_read-515.0)/1024.0*5.0/0.0286; // (mbar) depends on the sensor used
  if (millis() - printMillis > 60){
     Serial.print(pressure*10);
     Serial.print(",");
     Serial.print(coef*100);
     Serial.print(",");
     Serial.println(motorTemp);
     printMillis = millis();
  }
  if (pressure > max_p && press_read > 2){
    max_p = pressure;
    min_p = pressure;
  }
  if (pressure < min_p){
```

```
    min_p = pressure;
  }
  //==========END: READING AND PRINTING OUT VALUES ==========
}
//==========END: loop() ==========
```