Digitizing vents graph images.

June 17, 2020

Contents

1 Introduction

The program consists of 3 parts:

1. The pre-processing part:
    - Identifying the screen in a given image
    - Crop out the rest of image.
    - Straighten the image
2. The digitization part.
    - Identify parts of the image with pressure, volume flow rate and total volume (if present) graphs using opencv algorithms like Canny edge detection, Hough Line Transform and contour detection.
    - Find bounding boxes around the region of interest, digitize the graph with respect to the base of this box.
3. Fitting the equation parameters.
    - Fit the equation of motion [Equation 1] to the digitized data and obtain the parameters of interest.

2 Details

2.1 The pre-processing part

The identification and cropping steps make use of object detection and instance segmentation methods of neural networks. We have used the RetinaNet model from ImageAI for object detection and Mask-RCNN pre-trained model from GluonCV for instance segmentation. Once the region of interest is obtained, rest of the image is cropped.

The image straightening step is performed using the Canny edge detection, Hough line transform and perspective warping. All of these algorithms are used from the opencv implementation, this step is general in the sense that it should work for all types of monitors.

The program for this part can be found in `MaskRCNN.ipynb`

2.2 Digitization part

This part is not general in the sense that the program currently available only works for specific types of ventilator images. This task is achieved by a c++ program:

The programs which produces the executable is available in `Blue.cc`, `GreenOnWhite.cc` and `Line.cc`.

This requires the external dependencies:

1. OpenCV C++ libraries.
2. GNU GSL scientific library

This program also has several other dependencies such as Eigen but these are used as template header only libraries which is provided inside the source tree.

The main program for digitization is found in `MainDigitizerGreenOnWhite.hh`.
(similar programs are also found for other types of images.)

This implements many algorithms for pixel by pixel filtering, pre-processing steps and deciding bounding boxes on the graphs. It also implements the digitization functions for both the pressure and volume flow graphs.

2.3 Fitting the equation parameters

This step involves two further parts:

1. Smooth out the digitized data obtained from the previous step and use cubic spline interpolation to remove noise and perform interpolation to obtain continuous curve even in regions where digitized data might be rare.
2. Fit the ventilator equation of motion [Equation 1] parameters to the cleaned data.
3. Optionally, analyze the data after Fourier transforms.

The main program required for achieving this step is available in `Calculator2.hh` and `fitter2.hh`.

The main ventilator equation of motion is:

$$\begin{aligned} \Delta P &= EV + RF, \\ \text{Where } F &= \frac{dV}{dt} \equiv \dot{V}. \end{aligned} \tag{1}$$

The digitization procedure yields ΔP and F, but unfortunately, V is missing and needs to be calculated by integrating F. This step is unreliable and error prone due to imperfections in picking the integration domains, hence a better approach is to differentiate the equation to obtain:

$$\dot{\Delta P} = EF + R\dot{F}. \tag{2}$$

Here, all the parameters are only local and do not require integration (this also removes any errors due to a constant shift bias) although now we have to evaluate numerical derivatives. But it is far more easier and reliable to evaluate derivatives than integrals and hence this [Equation 2] equation will be used in the program to estimate E and R and more importantly the ratio $\frac{R}{E}$.

To fit the parameters, overlapping, sliding windows are chosen in the pressure and flow rate graphs (obtained by the previous step), numerical derivatives are calculated and finally the parameters are fit using gradient descent technique.

3 Details of select parts of the program

The main program which produces the executable is like:

```
#include "./GreenOnWhite.hh"

int main (int argc, char ** argv) {
        CPPFileIO::GetArgs
                args (argc,argv)
        ; //
        GreenOnWhite::DIGITIZER::TYPE_MAIN_DIGITIZER
                slave
        ; //
        for (size_t i=1;i<args();i++) {
                slave.Work1(args(i));
        }
        return 0 ;
}
```

here, `slave.Work1(args(i));` calls the main function which does all the work. This function is called on every filename mentioned on the command line, this behavior can be easily changed in the program. All the real work is done by this function which is found in the file `MainDigitizerGreenOnWhite.hh`. The final results (R/E) is written by the lines:

```
TYPE_INTERPOLATOR calc(output,outpoints);
calc.TO_FILE(filename+".pars");
```

found in the same header file.

The final digitized output is normalized to 1 (units are arbitrary), the units can be restored by using some references for each graph. For instance, PEEP can be used to restore normalization on the pressure graph, the PEEP from the image can be read using OCR techniques, lets call this $\text{PEEP}_{\text{Truth}}$. This quantity can also be calculated from the digitized data, a function which does this can be found in `Fitter::MinPeepSlab()` in the header file `Calculator2.hh`, lets call this $\text{PEEP}_{\text{Calc}}$. The pressure graph can then be rescaled by the fraction $\frac{\text{PEEP}_{\text{Truth}}}{\text{PEEP}_{\text{Calc}}}$ to restore the normalization of the pressure graph. Similar approach might be feasible to also restore the units of volume flow graph.