School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

A Computer Model of Electrocardiogram Signals

by

Andrew Chin Hou Au

Thesis submitted as a requirement for the degree of

Bachelor of Software Engineering

Submitted:	August 2019	Student ID:	z5020593
Supervisor:	Dr. Socrates Dokos	Topic ID:	

Contents

Chapter 1

Project Description

Computer modelling of Electrocardiogram Signals (ECGs) is a field of computer science involved with the generation of synthetic ECG signals. Computer models of ECGs are useful for developing signal processing techniques as algorithms can be tested against generated regular and irregular ECG signals.

Various models that utilize different mathematical backgrounds to generate synthetic ECGs have been introduced [1][2][3][4]. Features of interest to clinical ECGs commonly include noise filtering or parameter fitting, for which extensions [5], [6] have been introduced to [1].

Computer models of ECGs are typically proprietary and carry licensing requirements that make it difficult for users to access. Whilst open source implementations exist, they lack extensions introduced over the last two decades.

This thesis project aims to develop an open source alternative to a dynamic ECG model that implements the McSharry model[1] capable of parameter fitting to a sample ECG with an Extended Kalman Filter. Chapter 2 contains a progress report for the results obtained so far. Chapter 3 will then contain a reflection on the progress and thesis project followed by Chapter 4, which contains a summary of tasks completed and an adjusted project plan taking into account the current progress.

Chapter 2

Progress Report

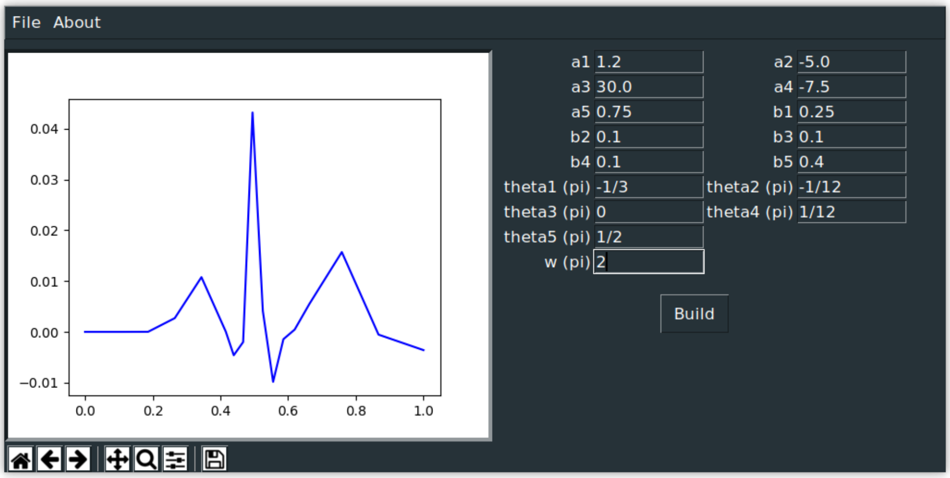
2.1 Initial Results

With the equations, variables and constants present in [1], a basic dynamic computer model capable of generating ECGs without noise was implemented in the previous semester.

The computer model was implemented in Python3 with tkinter, the standard graphical user interface (GUI) package for Python. The mathematical and graphing components were implemented with SciPy, a Python-based ecosystem of open-source software for mathematics, science, and engineering. NumPy was used for its ability to solve ordinary differential equations and matplotlib was used for its plotting capabilities.

Git was used for version control in development. Pip was used as a package installer to install additional software (SciPy/NumPy/matplotlib). Virtualenv was also used to create an isolated Python environment for software development.

The basic implementation allowed the generation of an ECG based on 16 variables that could be entered through a form. A form and canvas to hold the ECG plot was created through tkinter widgets such as Frames and Entries. A button is presented to the user which collects values from the form, solves an ordinary differential equation and then projects the result into the canvas area.



a1	1.2	a2	-5.0
a3	30.0	a4	-7.5
a5	0.75	b1	0.25
b2	0.1	b3	0.1
b4	0.1	b5	0.4
theta1 (pi)	-1/3	theta2 (pi)	-1/12
theta3 (pi)	0	theta4 (pi)	1/12
theta5 (pi)	1/2		
w (pi)	2		

Figure 2.1: GUI application for simple McSharry model

The basic model was used as a base for experience with SciPy and with tkinter components for building GUI applications.

The basic model had a number of shortcomings:

- Looks a bit old
- The values for θ_i and ω only accept numerical inputs, rather than mathematical expressions, which are required for different values of π
- The implementation lacks the ability to import and export form parameters
- Lacks ability to import sample ECG data (from Physionet)
- Lacks common graphing components (units, labels, etc)
- The generated ECG plot wasn't smooth

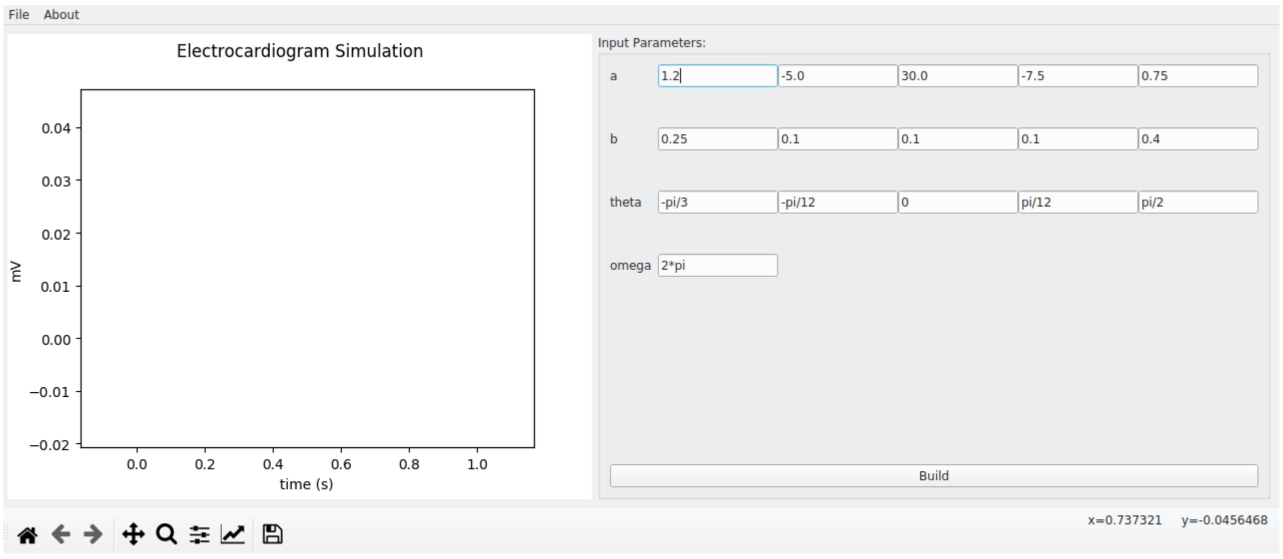
2.2 Technologies

After reflecting on the initial results new packages were introduced to cover the shortcomings of the basic model.

The total amount of technologies used are as follows:

- Python3
- SciPy (NumPy+matplotlib)
- PyQt5
- numexpr
- Git
- pip
- virtualenv

Notably, tkinter was replaced with PyQt5 and numexpr was introduced as a new package.



Input Parameters:

a	1.2	-5.0	30.0	-7.5	0.75
b	0.25	0.1	0.1	0.1	0.4
theta	-pi/3	-pi/12	0	pi/12	pi/2
omega	2*pi				

x=0.737321 y=-0.0456468

Figure 2.2: GUI application using PyQt

2.3 Graphical User Interface (GUI)

Tkinter was used as the GUI in the initial results. After some consideration, tkinter was dropped in favor of PyQt.

2.3.1 Tkinter, Tcl/Tk and ttk

Tkinter is a Python based binding of Tcl/Tk, which is a free open-source, cross-platform widget toolkit to build GUI applications. Tkinter is the de-facto standard for GUI development in Python and comes with most Python distributions.

Tkinter offers features through widgets that can be managed with layout managers. The pack layout manager is used to organises widgets horizontally or vertically. The grid manager organises widgets into grids, where each cell of the grid can be configured to shrink or expand under different conditions.

The initial results used a mix of tkinter pack and grid layouts to create a form and a canvas to plot an ECG. However, some difficulty was experienced trying to make widgets with the application window. This was eventually completed, but required a fair amount of configuration to achieve the functionality. This was made more difficult by the lack of adequate documentation for tkinter features.

While using tkinter it was noted that some legacy features were included as defaults in tkinter. Applications commonly use menus that contain commands such as save, open, exit, etc. Tkinter by default includes a tear out menu feature that allows the user to move the menu into another window when selected.

A concern with tkinter was making it appear more modern. Starting from Tk version 8 themed widgets are available through ttk. Ttk makes widgets look better, but only to a certain subset of widgets of tkinter. For instance, widgets such as Entries are covered, but Menus are not. While there are methods to customise the appearance, the effort placed into customizing the appearance of tkinter would be considerable.

2.3.2 Qt and PyQt

Qt is a modern open-source cross-platform widget toolkit for creating GUI applications. Qt requires a commercial license for commercial usage, but is free for open-source usage. It has bindings for different languages including C++ and Python. Qt offers PySide and PyQt bindings for Python.

Qt comes with a variety of widgets and layouts for different purposes. The central feature of Qt is a signal and slot based mechanism that allows widgets to communicate to another.

In usage Qt offered an instinctive system of widgets and layouts with clear documentation that aided greatly in development. Custom layouts not found in tkinter such as form and groupbox helped to simplify code. Appearance customisation was deemed unnecessary as the application appeared quite modern.

In using grid layouts, less lines of configuration were required to achieve widget resizing according to window size. Size hint options were also useful to create minimum sized windows and layouts.

Considering all these aspects, it was decided that PyQt would be used over tkinter.

2.4 Dynamic Generation of ECG

McSharry et al. proposed a synthetic ECG generator based on three dynamic equations of motion. The model generates a quasi-periodic trajectory in a three dimensional (3D) state space with coordinates (x, y, z). The trajectory revolves around a unit radius around the (x, y) plane, which corresponds to a single heartbeat. The points P, Q, R, S and T are described by events placed at fixed angles along the unit circle labelled as $\theta_P, \theta_Q, \theta_R, \theta_S$ and θ_T.

The equations of motion are given by three ordinary differential equations:

$$\dot{x} = \alpha x - \omega y$$
$$\dot{y} = \alpha y + \omega x$$
$$\dot{z} = -\sum_{i \in P,Q,R,S,T} a_i \Delta\theta_i \exp\left(-\frac{\Delta\theta_i^2}{2b_i^2} \right) - (z - z_0)$$

where $\alpha = 1 - \sqrt{x^2 + y^2}$, $\Delta\theta_i = (\theta - \theta_i) \bmod 2\pi$, $\theta = \text{atan2}(y, x)$ and ω is the angular velocity of the trajectory as it moves around the x, y plane. The constants a_i and b_i for $i = 1..5$ are based on the morphology of a healthy ECG.

The ordinary differential equation above was solved with the *solve_ivp* function available from NumPy. *Solve_ivp* takes a function, time span, initial value and times to evaluate the solution. The function to compute the positions is written as follows:

```
def odefcn(T, Y, a, b, w, events):
    '''
    Function to solve ODE with scipy.integrate.solve_ivp()
    http://web.mit.edu/~gari/www/papers/ieeetbe50p289.pdf
    '''
    x, y, z = Y
    dy = np.zeros(3)

    theta = np.arctan2(y, x)
    alpha = 1.0 - np.sqrt(x**2 + y**2)
    dy[0] = alpha*x - w*y
    dy[1] = alpha*y + w*x

    dy[2] = -(z - 0)
    for i in range(0, 5):
        dtheta = theta - events[i]
        dy[2] -= a[i] * dtheta * np.exp(-(dtheta**2) / (2 * b[i]**2))

    return dy
```

The function *solve_ivp* is run with 100 points to make the resulting plot more smooth:

```
tspan = np.array([-1.0, 1.0])
y0 = np.array([-1.0, 0.0, 0.0])
teval = np.linspace(0, 1, num=100)
print('building...')
sol = solve_ivp(fun=lambda t, y: odefcn(t, y, a, b, w, evt),
                t_span=tspan, y0=y0, t_eval=teval)
return sol
```

2.5 Input Validation and Evaluation

In initial results inputs were entered as strings and converted into numbers. However inputs do not accept mathematical expressions such as 2*pi*. To improve this aspect input validation and evaluation was introduced.

Fortunately, Qt inputs come with a validator property that can be customised using regular expressions. A regular expression was introduced to force inputs to only accept numbers, decimals, mathematical operators ($+, -, /, *$) and the letters "*pi*".

```
pi_re = QRegExp(r"[\d.+\-*/()pi\s]*")
```

This allows basic input of mathematical expressions that involve *pi*.

For convenience mathematical expressions involving variables should follow the same rules as algebra. e.g. $2a = 2 * a$. A custom replacer function was used to replace instances of *pi* depending on the mathematical operators around it.

```
pi_regx = re.compile(r"(pi\s*|[0-9.]+)(?=(\s*pi|[0-9.]))")
def pirepl(word):
    def repl(matchobj):
        if matchobj.group(1).strip() == 'pi':
            return "{}*".format(matchobj.group(1))
        elif matchobj.group(2).strip() == 'pi':
            return "{}*".format(matchobj.group(1))
        else:
            return matchobj.group(0)

    return re.sub(pi_regx, repl, word)
```

For instance, $2pi$ becomes $2 * \pi$ and $pi/2$ becomes $\pi/2$.

Values inside the forms are strings, not numbers. In order to use mathematical expressions with *pi*, values involving *pi* need to be replaced with a valid expression, then evaluated into a number.

Numexpr is a third-party open-source package that can perform fast numerical expression evaluation. For our purposes we will use numexpr to evaluate "*pi*" with mathematical operators and return a number. Numexpr doesn't handle *pi* by default, but accepts a map of custom expressions to values. Doing so allows us to evaluate expressions like "$2pi$" to 6.28318530718.

Invalid expressions given to numexpr would cause an error that stops the program. To handle this we simply catch any error that arises and create a message explaining that an error occurred due to an invalid expression.

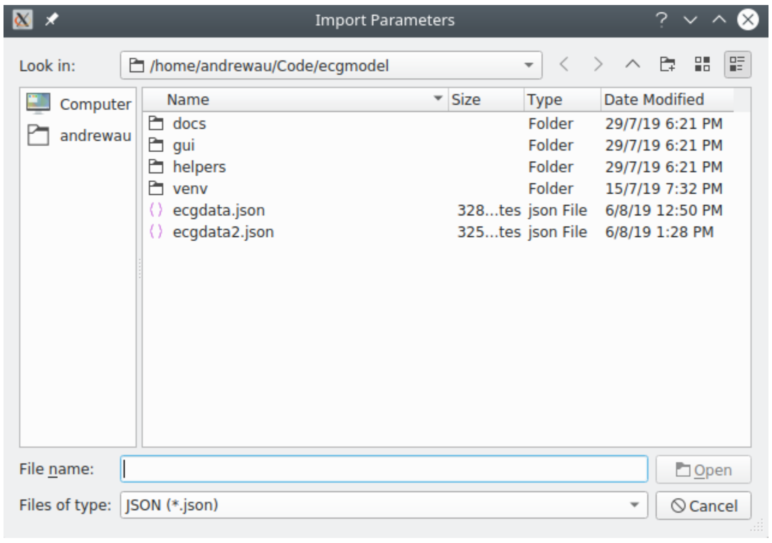
2.6 Parameters

To reduce the number of manual inputs, an import/export feature was introduced. This would allow a user to save the current values of the form to a file that can be loaded into the model later.

The JSON file format was chosen to contain the values of the exported parameters for its readability. Another good option would have been column separated values (csv) files.

When exporting parameters a typical save file window is shown with the ability to rename the file or place it in a different directory. Importing a file opens a similar window that only lists JSON files.

Input parameters to the form are saved as an array of string values. When a file is imported, the arrays and their contents are read and evaluated to numbers. If the file doesn't contain the right number of parameters or if the parameter doesn't contain a valid mathematical expression a message will appear explaining that the parameters are invalid.



Look in: /home/andrewau/Code/ecgmodel

Name	Size	Type	Date Modified
docs		Folder	29/7/19 6:21 PM
gui		Folder	29/7/19 6:21 PM
helpers		Folder	29/7/19 6:21 PM
venv		Folder	15/7/19 7:32 PM
ecgdata.json	328...tes	json File	6/8/19 12:50 PM
ecgdata2.json	325...tes	json File	6/8/19 1:28 PM

File name:

Files of type: JSON (*.json)

Figure 2.3: Import window for exported parameters

2.7 Importing ECG samples

ECG samples in the CSV file format can be downloaded from Physionet through their ATM service. Physionet provides multiple options, including time length of sample.

CSV files from Physionet contain a header for each column of comma separated values. Each column contains time and amplitude for the ECG. NumPy can be used to import CSV data using $numpy.genfromtext$. Headers in the file are skipped as the information isn't needed.

Imported samples have varying lengths of time. When a sample is imported into the model, a window with a slider to select the length of time is shown. The selected time length is returned using NumPy style indexing and then projected onto the canvas.

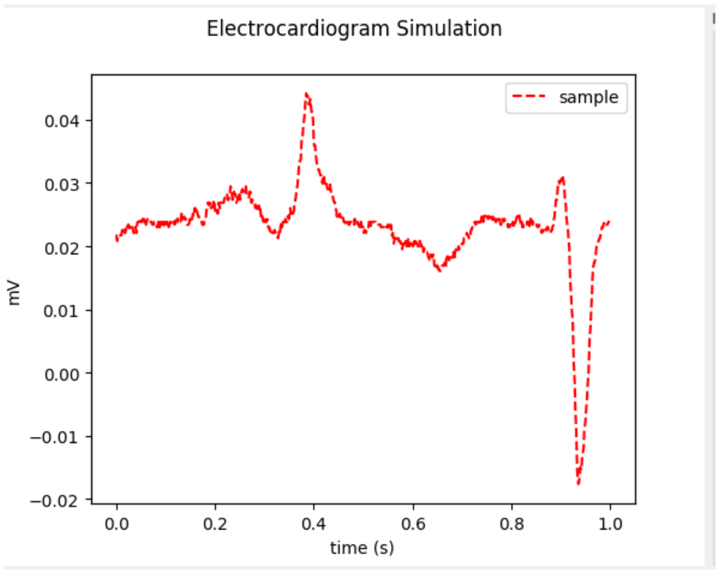


Figure 2.4: Imported ECG sample from Physionet (ANSI/AMII EC13 Database

2.8 Matplotlib

Matplotlib is a plotting library for Python that belongs to the SciPy ecosystem of software. Matplotlib comes with bindings for different GUI libraries including tkinter and PyQt. A canvas containing a figure can be created with matplotlib.

Matplotlib has an object-oriented interface. Figure objects contain Axes objects which stores Line objects for a plot. An Axes object also contains Axis objects, which describe the labels, legends and units for the Axes.

Two Axes were created with one containing the generated ECG and the other containing an ECG sample. By making an Axes share the drawing area as the other, both graphs can coexist with the same scaling. When generating or importing a new ECG the relevant ECG Axes object has its Lines replaced with the new data.

Legends were added to the graph plot by searching for Line objects in all Axes. Lines from generated ECGs are displayed as ”estimate”, while Lines for the imported sample is displayed as ”sample”.

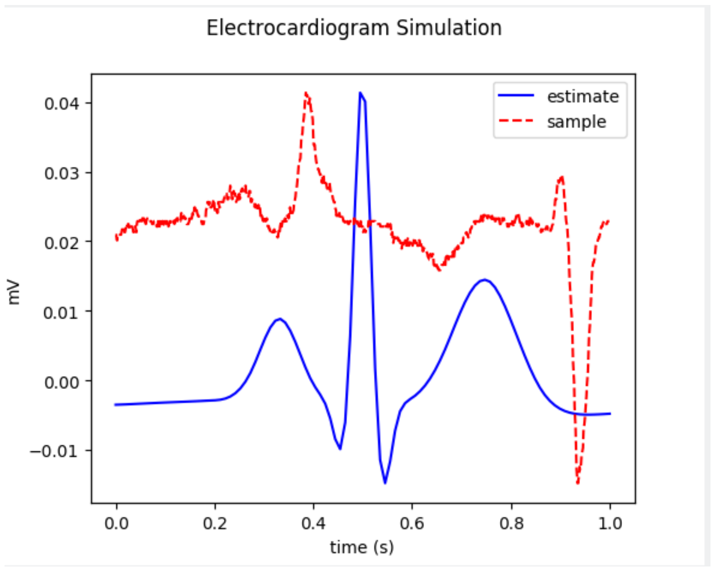


Figure 2.5: Plot of imported sample and generated ECG

2.9 Kalman Filtering

Kalman Filtering is the chosen method for parameter fitting against an imported ECG sample.

A Kalman filter only works with linear models. However the ECG model based on [1] is non-linear. This being the case, an Extended Kalman Filter is required. In order to implement an Extended Kalman Filter, a linearized model must be produced from the McSharry model. An Extended Kalman Filter for an ECG has been described in [6].

This task has not yet been completed as time is needed to understand and implement the components. Completion for this task is likely to occur in the next semester.

Chapter 3

Reflections

3.1 Thesis structure

The thesis project places an emphasis on open-ended design based on the understanding of an advanced problem. These issues tend to require critical analysis and understanding.

Typically, academic and industry projects have fixed requirements. For instance, a certain language or framework must be used. The thesis project on the other hand is open-ended, meaning that any choice of language and framework can be used to solve an issue.

The amount of reading is also more specialised than projects found in undergraduate courses so far. A good understanding of a concept is important to implementing a system. The lack of a good understanding of Kalman Filters has lead to a delay and change in the intended schedule. For the future, it would be best to read more in earnest.

An aspect that remains unchanged from other projects is project management. This means some form of version control, time management and agile methodology.

3.2 Importance of proper documentation

When implementing software documentation is vital. Tkinter's documentation was somewhat lacking and a bit incomplete. PyQt documentation led to the documentation of Qt written in C++. Fortunately, the PyQt binding followed very closely to Qt in C++.

Matplotlib was at times confusing with two different APIs. One API was for MATLAB style interactive commands, whereas the other was an object oriented style that was used for the implementation. The distinction between Axes and Axis objects was also confusing at times. There were many examples with matplotlib documentation which greatly helped in development.

In the future, write documentation that would help others understand and use what you've created.

3.3 Implementation Choices

GUI development began with tkinter rather than with PyQt. Over time it became clear that tkinter would be more difficult to use due to inadequate documentation. Tkinter also lacked features that would have made development easier.

The initial implementation was developed on until an import/export feature was introduced. In hindsight, it would have been better to research more thoroughly into the capabilities of tkinter before starting with it.

Chapter 4

Task Summary and Revised Plan

4.1 Task summary

Table 4.1: Summary of tasks completed

Task	**Status**
Initial results (generate ECG on a plot)	Complete
Tkinter import/export parameter for ECG	Complete
Transition from Tkinter to Qt	Complete
Validate parameter values (number input and pi)	Complete
PyQt dialogs (parse warning, import/export parameter)	Complete
PyQt import sample data (using csv from Physionet)	Complete
Matplotlib display two graphs	Complete
PyQt Slider (slider to select timeframe of ECG sample	Complete
PyQt Graph/Matplotlib features (title, legends, labels)	Complete
Kalman Filtering	Delayed

This semester saw improvements to the initial results of the previous semester. Various improvements such as UI, graph smoothness and graphing features were added. Features were also added in anticipation of parameter fitting to ECG samples from Physionet.

Kalman Filter was planned but has been delayed, likely to see completion early next semester.

4.2 Revised Plan

Weeks 1 - 3

- Complete implementation of Extended Kalman Filtering.
- Adjustments to UI.
- Decide evaluation method for parameter fitting (testing various ECG signals).

Weeks 4 - 6

- Adjustments to UI for different operating systems.
- Testing ECG generation and parameter fitting over samples from Physionet.

Weeks 7

- Packaging and distribution of software package.

Weeks 8 - 10

- Write final report.

Bibliography

[1] P. E. McSharry, G. D. Clifford, L. Tarassenko, and L. A. Smith, "A dynamical model for generating synthetic electrocardiogram signals," *IEEE Transactions on Biomedical Engineering*, vol. 50, pp. 289–294, March 2003.

[2] P. Kovcs, "Ecg signal generator based on geometrical features," *Annales Universitatis Scientiarum Budapestinensis de Rolando Etvs Nominatae. Sectio Computatorica*, vol. 37, 01 2012.

[3] J. Kubicek, M. Penhaker, and R. Kahankova, "Design of a synthetic ecg signal based on the fourier series," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1881–1885, Sep. 2014.

[4] P. Dolinsk, I. Andras, L. Michaeli, and D. Grimaldi, "Model for generating simple synthetic ecg signals," *Acta Electrotechnica et Informatica*, vol. 18, pp. 3–8, 09 2018.

[5] G. D. Clifford and P. E. McSharry, "Method to filter ecgs and evaluate clinical parameter distortion using realistic ecg model parameter fitting," in *Computers in Cardiology, 2005*, pp. 715–718, Sep. 2005.

[6] R. Sameni, M. B. Shamsollahi, and C. Jutten, "Filtering electrocardiogram signals using the extended kalman filter," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pp. 5639–5642, Jan 2005.