# Design and Evaluation of a MATLAB® Simulated Blood Gas Monitor

by

Marguerite C Wellstein

A Thesis Submitted to the Faculty of the

Milwaukee School of Engineering in Partial

Fulfillment of the Requirements for the

Degree of Master of Science in Perfusion

Milwaukee, Wisconsin

February 24, 2017

## Abstract

Immersive perfusion simulation is becoming more prevalent in perfusion education and training. This is best thought to be achieved through high fidelity simulations that mimic as much of the operating room environment as possible. Thus, this type of simulation would include all equipment, both application and monitoring. MSOE currently has an in-house developed hemodynamic perfusion simulator. It allows students to practice tasks such as initiating bypass, maintaining arterial pressure, and weaning from bypass. Currently this simulator is not paired with blood gas monitoring equipment. The goal of this project was to add blood gas monitoring capabilities to the simulator.

The blood gas monitor simulator was developed by modeling the physiologic systems that interact with blood gas monitors and creating an interactive program that would display the values achieved with normal blood gas analysis. The physiologic models were represented by a set of equations that were entered into MATLAB® as a function within a programed graphical user interface (GUI). When the code initiated the function, all relevant values were collected from the interactive control and used in the calculations to determine dependent blood gas values. All values were then displayed.

Testing of the system through trial runs specific to the programmed models found that the virtual simulation would accurately represent the normal physiologic responses to changes in blood gas values, as well as the effects of cardiopulmonary bypass on those mechanisms. The addition of the simulated continuous blood gas monitor has created a more immersive simulation that allows students to practice assessing blood gas values and improve situational awareness that is necessary when these values change.

## Acknowledgements

This master's thesis would not have been possible without the help, and guidance from my thesis committee that assiduously supported my effort; their contributions are truly appreciated. I would like to thank each member of my committee: Dr. Ron Gerrits for his extensive review, editing work, and patience; Kirsten Kallies, CCP for her patience and diligence in reviewing my project, and providing guidance; and Jonathan Howard, CCP for his encouragement and contributions in design and review. I would also like to thank Lauren Mason, my parents, and my sisters for supporting my efforts during this process, and for being patient with me throughout this stressful endeavor. Finally, I would like to thank Frank Wellstein and Jean Leuhring for their unwavering support and encouragement throughout my educational pursuit. I will be indebted to all of the aforementioned people who so generously donated their time and supported this work.

# Table of Contents

**List of Figures** ..... 5
**List of Tables** ..... 6
**Nomenclature** ..... 7
**1.0 Introduction** ..... 8
**2.0 Background** ..... 10
2.1 Use of Simulation Technology in Perfusion Education ..... 12
2.2 Simulation at Milwaukee School of Engineering ..... 14
2.3 Continuous Inline Blood Gas Monitoring ..... 15
**3.0 Project Goal** ..... 16
**4.0 Theoretical Design** ..... 17
4.1 Acid-Base Balance ..... 20
4.2 The Oxyhemoglobin Dissociation Curve ..... 24
4.3 Non-Gas Values Measured by CDI$^{TM}$ 500 Inline Monitoring System ..... 27
*4.3.1 Potassium ($K^+$)* ..... 27
*4.3.2 Temperature* ..... 28
*4.3.3 Hematocrit* ..... 29
**5.0 Implementation of Design** ..... 30
5.1 Acid-Base Model ..... 31
5.2 Oxyhemoglobin Dissociation Curve Model ..... 32
5.3 Program Development ..... 34
5.4 Simulation Testing ..... 39
**6.0 Results** ..... 41
6.1 User Interface Testing ..... 41
6.2 Physiologic Model Testing ..... 42
*6.2.1 Acid-Base Model* ..... 42
*6.2.2 The Oxyhemoglobin Dissociation Curve Model* ..... 47
**7.0 Discussion** ..... 52
7.1 Future Improvements ..... 53
**8.0 Conclusion** ..... 54
**References** ..... 55
**Appendix A: MATLAB Code** ..... 57

## List of Figures

Figure 1: Standard Cardiopulmonary Bypass Pump Schematic. ..... 11

Figure 2: Simulation Facility with Orpheus System. ..... 13

Figure 3:The Display of the Terumo CDI$^{TM}$ 500. ..... 17

Figure 4: The Relationship Between Blood pH, $pCO_2$, and $HCO_3^-$ ..... 23

Figure 5: The Oxyhemoglobin Dissociation Curve. ..... 25

Figure 6: Monitor Display Design Created in MATLAB® GUIDE. ..... 35

Figure 7: The User Controlled Interface. ..... 35

Figure 8: Pseudocode for Simulation Program. ..... 37

Figure 9: The Simulation Windows Shown on Two Monitors. ..... 42

Figure 10: The Effect of $HCO_3^-$ on the pH. ..... 43

Figure 11: The Effect of $PCO_2$ when $HCO_3^-$ is Held Constant. ..... 44

Figure 12: The Effect of $PCO_2$ and $HCO_3^-$ on pH. ..... 44

Figure 13: The Effect of $pCO_2$ on Base Excess. ..... 45

Figure 14: The Effect of $pO_2$ on Base Excess. ..... 46

Figure 15: The Effect of $HCO_3^-$ on Base Excess. ..... 47

Figure 16: The Estimated Oxygen Saturation from $pO_2$ Using the Severinghaus Equation. ..... 48

Figure 17: The Effect of Flow on the Venous Saturation. ..... 49

Figure 18: The Effect of Hemoglobin Concentration on the Venous Saturation. ..... 50

Figure 19: The Effect of $pO_2$ on Venous Saturation. ..... 50

Figure 20: The Effect of Oxygen Consumption on Venous Saturation. ..... 51

## List of Tables

Table 1: Operational and Display Ranges of the Terumo CDI™ 500. ..... 18

Table 2: Normal Blood Gas Values. ..... 19

Table 3: Independent Variables and their Corresponding Mechanism of Control During Cardiopulmonary Bypass. ..... 30

## Nomenclature

| | |
|---|---|
| **$C_aO_2$** | Arterial Blood Oxygen Content |
| **$CO_2$** | Carbon Dioxide |
| **$C_vO_2$** | Venous Blood Oxygen Content |
| **$FiO_2$** | Fraction of Inspired Oxygen |
| **$H^+$** | Hydrogen ion |
| **$H_2CO_3$** | Carbonic acid |
| **$H_2O$** | Water |
| **$HCO_3^-$** | Bicarbonate Ion |
| **$K^+$** | Potassium Ion |
| **$pCO_2$** | Partial Pressure of Carbon Dioxide |
| **$pO_2$** | Partial Pressure of Oxygen |
| **$SO_2$** | Oxygen Saturation |
| **SVC** | Superior Vena Cava |
| **$S_vO_2$** | Venous Oxygen Saturation |
| **$VO_2$** | Oxygen Consumption |
| | |
| **Hct** | Hematocrit |
| **Hgb** | Hemoglobin |
| **uicontrols** | User Interface Controls |
| | |
| **AmSECT** | American Society of Extracorporeal Technology |
| **BE** | Base Excess |
| **CBGM** | Continuous Inline Blood Gas Monitor |
| **CPB** | Cardiopulmonary Bypass |
| **EKG** | Electrocardiogram |
| **GUI** | Graphical user Interface |
| **GUIDE** | Graphical User Interface Development Environment |
| **IVC** | Inferior Vena Cava |
| **MSOE** | Milwaukee School of Engineering |
| **pH** | Potential of Hydrogen |
| **RBC** | Red Blood Cells |
| **ZBUF** | Zero Balance Ultrafiltration |

## 1.0 Introduction

Simulation technology has become a staple in the medical field. Creating situational scenarios for healthcare providers can help reduce iatrogenic risk to patients and reinforce understanding of medical equipment. Increased understanding has led to more specialized simulation modules that are profession and task based.

Cardiopulmonary bypass simulators have become a popular method of teaching perfusion by combining artificial hemodynamic systems and physiological models to create an immersive learning experience [1]. They provide risk free, reproducible environments for hands-on learning and evaluation. Physiological modeling software used in perfusion simulators such as Orpheus have allowed for customizable case simulations [1]. These scenarios are also useful for practicing perfusionists to simulate unique or catastrophic events they may not experience in their practice.

The perfusion simulator at the Milwaukee School of Engineering (MSOE) was designed and implemented by Jonathan Howard [2]. Howard's design was a mock loop comprised of mechanical components that provided an analogous representation of the arterial and venous systems. Later, Caleb Varner added the representation of the pulmonary system and variable venous compliance [3]. The current simulator allows students to practice initiating and weaning from CPB in variable scenarios while monitoring simulated patient pressures. However, there is more to perfusion than just hemodynamics and initiating or weaning from CPB.

The current simulator does not provide patient scenarios while on bypass or assessing the adequacy of perfusion. An addition of simulated monitoring systems such as a continuous inline blood gas monitor (CBGM) would add a new dimension to the

simulator. Monitoring and troubleshooting blood gas values would provide students with a more realistic operating environment and help in developing situational awareness.

The goal of this project was to design and program a graphical user interface (GUI) application to simulate a CBGM. This included two GUI components and multiple functions built into the programs code. The program would collect data from an interactive user interface, then use them to calculate and model physiologic conditions representative of the cardiac patient population displayed through blood gas values. All pertinent values would be displayed on the second GUI that acted as a simulation monitor to the student. The simulator would allow students to practice assessment on blood gas parameters while on pump and in a controlled environment. A CBGM displays several pertinent values that help to assess the adequacy of perfusion during a case. The application would be displayed in a similar manner to CBGM used clinically and would allow an instructor to alter the parameters.

## 2.0 Background

Cardiopulmonary bypass is a medical technique required in a multitude of cardiovascular surgeries. It is the use of a physiologically analogous mechanical circuit to replicate the function of the heart and lungs of the patient. The basic procedural setup consists of venous cannulas placed in the superior vena cava (SVC) and inferior vena cava (IVC) that are attached to an extracorporeal circuit's venous line, which drains deoxygenated blood into a filtered reservoir. From the reservoir, a mechanical pump, the "heart", is used to move the blood through tubing to the oxygenator, the "lung". As blood travels through the artificial lung, the blood is oxygenated and carbon dioxide is removed via diffusion. Some current oxygenator models have a built-in heat exchanger and arterial filter to maintain desired temperature and filter blood before its return to the patient. The oxygenated blood is returned to the patient through the arterial line to arterial cannula placed in the aorta. This procedure effectively circumvents the native heart and lungs (Figure 1).

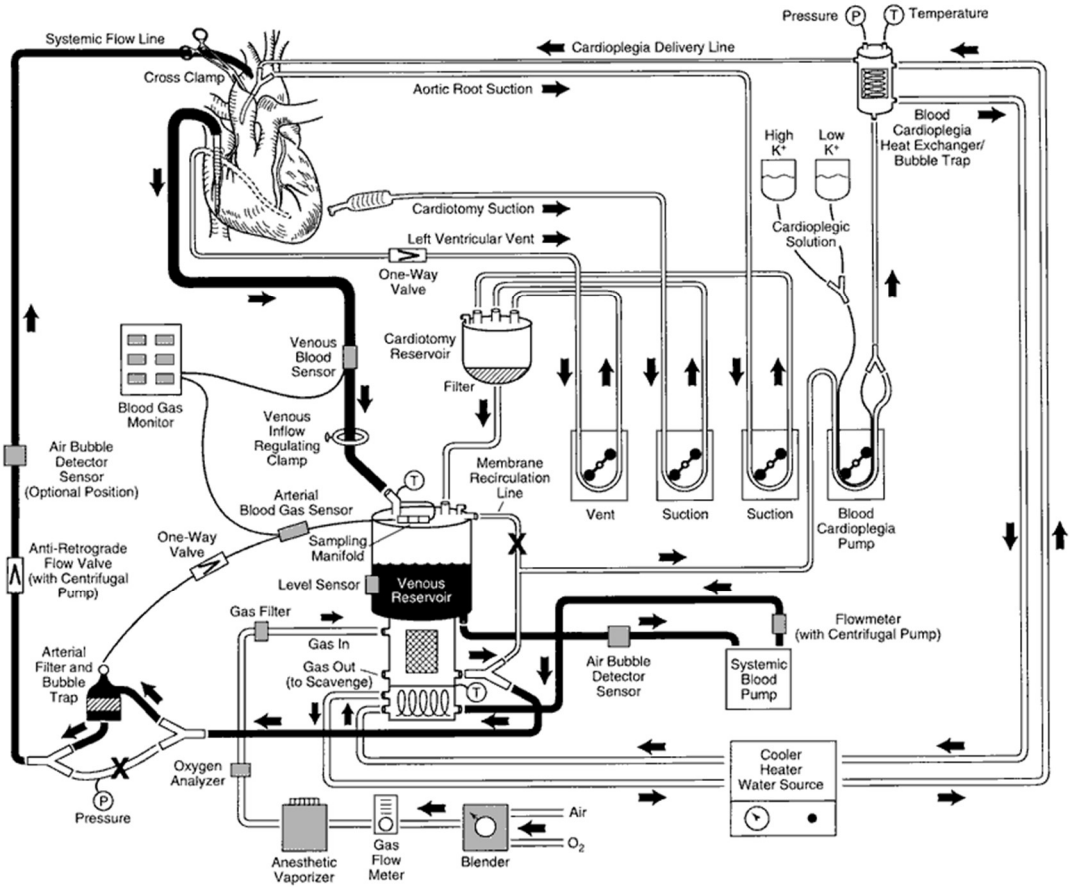


**Figure 1: Standard Cardiopulmonary Bypass Pump Schematic [4].**

In addition to simulating the function of heart and lungs, the CPB circuit includes multiple occlusive pumps, a cardioplegia delivery system, and ancillary monitoring equipment. The occlusive pumps are used as a means to pull shed blood from the surgical field to a cardiotomy. The cardiotomy, often built into the venous reservoir, is used to filter out gross emboli and salvage blood. The cardioplegia system is used to deliver pharmacological agents to protect myocardium and establish quiescence of the heart during surgery. Some of the monitoring equipment used with CPB include pressure, blood gas, and electrocardiogram (EKG). The information provided can be used to assess hemodynamics, electrical activity and homeostatic mechanisms. In order to simulate CPB

realistically, a multifaceted model must be created to account for the multiple aspects described.

### 2.1 Use of Simulation Technology in Perfusion Education

There is a long history of medical simulation which predates perfusion as a profession. Simulations have been used as educational implements for understanding physiologic systems, predict potential outcomes, and to learn/practice technical abilities [5]. Simulation technology has become increasingly profession-specific as knowledge expands and equipment changes. Perfusion is still a relatively new profession that has evolved drastically in the past 50 years. Increased understanding has led to development of CPB-specific simulations as a tool for this growing profession.

There are multiple types of CPB simulators available, including virtual and high fidelity. Virtual simulators are solely computer-based programs that run interactive scenarios where a user provides feedback with a computer interface. High fidelity simulators such as the Orpheus, created by Ulco Technologies, combine programmed physiological models with mechanical systems in an operating theater to create more realistic operating room experiences (Figure 2) [1, 6]. Simulated cases can be run with a normal CPB setup as the interface which reinforces physical perfusion techniques [1]. The reproducibility of the scenarios also provides a more standard way to evaluate students' performance and promote safe practices [5, 6]. These attributes make simulators a desirable method for teaching and mastering procedural skills before students enter the operating room.

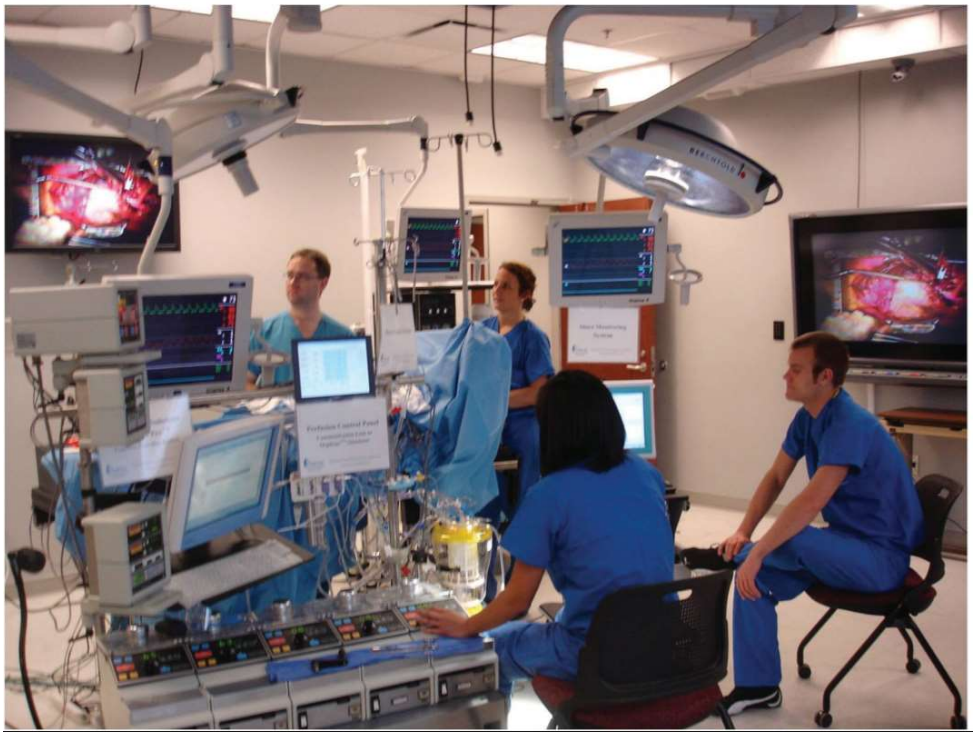

**Figure 2: Simulation Facility with Orpheus System [6].**

While CPB simulators are becoming a more prevalent educational tool, perfusion is a niche field that is technologically complex. That combination makes it difficult to get companies to make supporting products, such as simulators, because there is little market for it. Those companies that have created high fidelity CPB simulators sell them at some staggering costs, which make this technology out of reach for many.

A potential solution to the expense is to build a simulator. Mock loop simulators can be constructed in a cost-effective manner using off-the-shelf mechanical components that replicate hemodynamic behavior. Supporting software can also be developed and incorporated with the mock loop to create a more realistic simulation to rival costly high fidelity simulators.

## 2.2 Simulation at Milwaukee School of Engineering

The MSOE Perfusion program is clinically driven and provides students didactic lessons with concurrent clinical experience. The available caseloads of MSOE's affiliated hospitals and small class sizes provide an abundant amount of clinical opportunities for students to hone their abilities. The relative size of the program and institution make an investment into a high fidelity simulator not financially feasible. This problem was approached by the faculty and students in inventive way. In partial fulfillment of requirements for the program, many students have used their knowledge to contribute to a student designed simulator.

The current simulator was designed and implemented by Jonathan Howard to simulate the mechanical function of the heart and the vasculature. It is a hydrodynamic loop, which consists of a variety of components that mimic the variable hemodynamic functions, such as pressure, flow, capacitance and resistance [2]. Its purpose was to be used to practice initiating CPB, maintaining arterial pressure, and weaning from CPB. The system was designed to be simple and open to allow for additions and improvements. Varner later contributed to the Howard simulator by implementing a mechanical representation of the pulmonary circulation, variable venous compliance and additional pressure monitoring [3]. These additions allowed for more scenarios to be simulated during specified tasks.

The development and construction of the mock loop simulator was successful but its implementation into the curriculum has been less effective. The current setup is used for catastrophic event simulations where students primarily practice cutting components in and out of the circuit. In an effort to bridge the gap, Posch developed a curriculum in

which he outlined teaching objectives to be used with the Howard-Varner simulator [7]. The curriculum's outlined objectives inadvertently showed limitations of the current simulator (*i.e.*, continuity of case simulation and monitoring capabilities).

Both Howard and Varner recognized the need for simulation of patient monitoring devices used in clinical practice, such as a continuous inline blood gas monitor (CBGM). Incorporating an interactive CBGM simulation with physiologic values and scenarios would allow for simulations to be presented as full cases instead of individual tasks. It provides students an opportunity during a case simulation to practice scanning the circuit and troubleshooting potential problems observed via blood gas values.

**2.3 Continuous Inline Blood Gas Monitoring**

The use of CBGM in CPB cases is a topic of debate among many perfusionists. It is not a current standard of practice for perfusion but is a recommended guideline by the American Society of Extracorporeal technology (AmSECT) [8]. Some argue that the cost is unnecessary, and if proper blood gas assessments are done during a case, that there is no need for a CBGM. However, the CBGM can offer real time blood gas data trends, which provides a dynamic picture, reflecting the dynamic nature of the body. Studies have shown improved postoperative outcomes for patients who are continually monitored, as well as improved blood gas management during CPB [9-11].

For simulation purposes, having a simulated CBGM is a necessity for the simple reasons that using blood in a simulation is not safe or sustainable. By simulating these values, an instructor can create realistic scenarios that can occur while on pump.

## 3.0 Project Goal

The scope of this thesis was to design and create a graphical user interface that simulates the function of a continuous blood gas monitor, such that it could be used in conjunction with the Howard-Varner mock circulatory loop or for in-class demonstrations. Specific goals were to:

1. add the ability to monitor blood gas values that simulate physiological states during CPB;
2. create a user friendly interface that instructors and students can use;
3. allow for variable adjustments to simulate different scenarios that may occur in a case;
4. create built-in scenarios that will automatically adjust blood gas values;
5. create an accessible program that allows for use in a class setting or in combination with the Howard-Varner mock loop.

## 4.0 Theoretical Design

The human body is a complex system consisting of numerous homeostatic feedback loops [12]. The body continually compensates in order to stay in an optimal range to function. If this regulation is not maintained, it can cause deleterious effects to the patient. This concept is crucial in perfusion. CPB is not as functionally effective as the native lungs or heart. Those organs have more roles than pumping or oxygenating blood. Bypass essentially removes certain compensatory mechanisms of the body, which can be detrimental if prolonged. Proper monitoring of patient vitals and blood gases can be used to attenuate these negative effects of CPB. In order to implement appropriate homeostatic feedback mechanisms into the simulator, it is helpful to overview the variables measured via CBGM devices and the mathematical relationships connecting pertinent variables.


**Figure 3: The Display of the Terumo CDI™ 500.**

Based on the project goals, the simulator will be designed to mirror that of a Terumo CDI™ 500 monitor (Figure 3). Listed in Table 1 are the physiologic parameters displayed on a CDI and the operating ranges. Note that although these are called inline blood gas monitors, they actually measure a few variables, such as potassium, pH, and temperature, which are not blood gases. This project was meant to incorporate these variables as well.

**Table 1: Operational and Display Ranges of the Terumo CDI™ 500 [13]**.

| Displayed Parameters | System Operating Ranges | System Display Ranges |
|---|---|---|
| pH | 6.8 to 7.8 pH units | 6.5 to 8.5 units |
| $pCO_2$ | 10 to 80 mm Hg (1.3 to 10.7 kPa) | 10 to 200 mm Hg (1.3 to 26.7 kPa) |
| $pO_2$ | 20 to 500 mm Hg (2.7 to 66.7 kPa) | 10 to 700 mm Hg (1.3 to 93.3 kPa) |
| $K^+$ | 3.0 to 8.0 mmol/L | 1.0 to 9.9 mmol/L |
| Temperature (T) | 15° to 40° C | 1° to 45° C |
| Oxygen saturation ($SO_2$) | 60 to 100% | 35 to 100% |
| Hematocrit (Hct) (15° < T <40° C) | 17 to 38% | 12 to 45% |
| Total hemoglobin (Hgb) | 5.6 to 12.6 g/dL | 4.0 to 15.0 g/dL |
| Oxygen consumption ($VO_2$) | 10 to 400 mL/min | 10 to 400 mL/min |
| Base Excess (BE) | -25 to 25 mEq/L | -25 to 25 mEq/L |
| Bicarbonate ($HCO_3$) | 0 to 50 mEq/L | 0 to 50 mEq/L |
| Blood flow (Q) | 0 to 9.9 L/min | 6.5 to 8.5 units |

The only "patient" specifications required are the operational values of a CDI 500 CBGM display. The simulator should cover similar ranges that can be displayed on such a monitor (Table 1), but should ensure that under normal "patient" conditions, the values displayed are within the normal ranges (Table 2). The simulator design was meant to include a range for each of these physiologic values that a perfusionist might encounter, as discussed in the sections to follow. Each section will discuss the physiological homeostatic models pertinent to blood gas assessment and its indications during CPB, which form the underlying models and relationships for programming into a simulator.

**Table 2: Normal Blood Gas Values [12, 14, 15].**

| Blood Gas Variable | Normal Value |
|---|---|
| pH | 7.35 - 7.45 |
| Partial Pressure of Oxygen | 100 - 400 mmHg |
| $O_2$ Arterial Saturation | 96 - 100% |
| Partial Pressure of Carbon Dioxide | 35 - 45 mmHg |
| Base Excess | ± 2.5 |
| Bicarbonate ion | 22 – 28 mEq/L |
| Potassium | 3.5 – 5 mEq/L |
| Hemoglobin | 12 - 16 g/dL |
| Hematocrit | 22 – 38% |
| Mixed Venous Saturation | 73 – 77% |
| Temperature | 35 - 37° C (normothermic) |
| $O_2$ Consumption | 200 – 250 mL/min |
| Flow rate | 2 – 2.5 L/min/$m^2$ |

### 4.1 Acid-Base Balance

The acid-base balance is commonly referred to in terms of a pH value. The potential of hydrogen (pH) is defined by Encyclopedia Britannica as the quantitative measure of the acidity or basicity of aqueous or other liquid solutions [16]. It is derived from the concentration of hydrogen ions shown in Equation (1):

$$\mathrm{pH} = -\log[\mathrm{H}^+]. \tag{1}$$

In human blood, the measure can vary between 7.35 to 7.45 [17]. The small range is optimal environment to sustain the metabolic function of cells [14]. It is maintained by chemical and physiological buffers of the body.

The most effective chemical buffers in blood are the red blood cells (RBC) which facilitate the bicarbonate buffer system [14]. Carbon Dioxide ($CO_2$) is released from tissues as a result of cellular metabolic activity. It enters the RBC and binds with water ($H_2O$) in the presence of carbonic anhydrase to form carbonic acid ($H_2CO_3$). The dissociation of the weak acid results in a hydrogen ($H^+$) and bicarbonate ion ($HCO_3^-$) [18]. The chemical reaction occurring as a result of the buffer can be seen in Equation (2):

$$\mathrm{CO_2} + \mathrm{H_2O} \leftrightarrow \mathrm{H_2CO_3} \leftrightarrow \mathrm{H^+} + \mathrm{HCO_3^-}. \tag{2}$$

The pH is inversely related to the $H^+$ concentration. An increase in $H^+$ ions decreases the pH (acidic), while an increase in $HCO_3^-$ ions increases the pH (alkalotic). To maintain electrical neutrality, the $H^+$ ions bind to the Hemoglobin in the RBC and the bicarbonate diffuses into the plasma as chloride ions diffuse in, thus completing a

chloride shift [18]. This quick response buffer allows for the transport and removal of $CO_2$ without causing a major change in pH.

The normal ratio of bicarbonate to carbonic acid is 24:1.2 or 20:1. Deviations from this ratio affect the blood pH [17]. This relationship is described in the Henderson-Hasselbach equation [12], Equation (3):

$$\mathrm{pH} = \mathrm{pK} + \log\frac{\mathrm{HCO_3^-}}{0.03\mathrm{xpCO_2}}. \tag{3}$$

The Henderson-Hasselbach equation is derived with respect to mass balance [12]. Mass balance is the concept that the concentration of $H^+$ and $HCO_3^-$ are proportional to the concentration of carbonic acid [12]. The proportion of dissociated ions to the acid is called the dissociation constant (pK) which is 6.1 for bicarbonate [12]. The concentration of $CO_2$ is used in this equation because it is directly proportional to the concentration of undissociated carbonic acid. The more common form of measurement of $CO_2$ is as a partial pressure so the solubility coefficient (0.03mmol/mmHg) is used to determine the concentration [12]. From Equation (3), the pH can be calculated if the concentration of $HCO_3^-$ and partial pressure of $CO_2$ are known.

Physiological buffers of the body that help to regulate pH are the functions of the kidneys and lungs. The bicarbonate buffer system is fast acting, but the kidneys and lung can compensate for acid-base disturbances [17]. The lungs regulate the removal of $CO_2$. If there is excess $H^+$ due to high $CO_2$, it can be reduced with respiration within a couple of minutes [12]. The kidneys can excrete or retain $H^+$ ion, retain $HCO_3^-$, and regenerate $HCO_3^-$ from $H_2O$ and $CO_2$ to compensate for an alkalotic or acidotic state [17]. However, this process can take up to days to occur.

This is where the pH becomes very important for a perfusionist. During CPB, the heart and lungs of a patient are bypassed [19]. The role of the perfusionist during surgery is not only to oxygenate the blood but to take on the compensatory role of the lungs. Perfusionists are able to control the $pCO_2$ of the patient by adjusting the sweep rate, which is the flow of mixed oxygen and medical air through the oxygenator. Increasing the rate removes more $CO_2$ from the blood. The concentration of $HCO_3^-$ can also be altered by the perfusionist during CPB with the addition of a sodium bicarbonate solution. Understanding the relationship between pH, $CO_2$, and $HCO_3^-$ is necessary in maintaining physiologically normal pH during a case. Additionally, this knowledge will help with managing patients with comorbidities that may affect the function of the lungs or kidneys and the normal pH state. Figure 4 shows the relationship of the pH, $CO_2$, and $HCO_3^-$ , along with corresponding conditions.

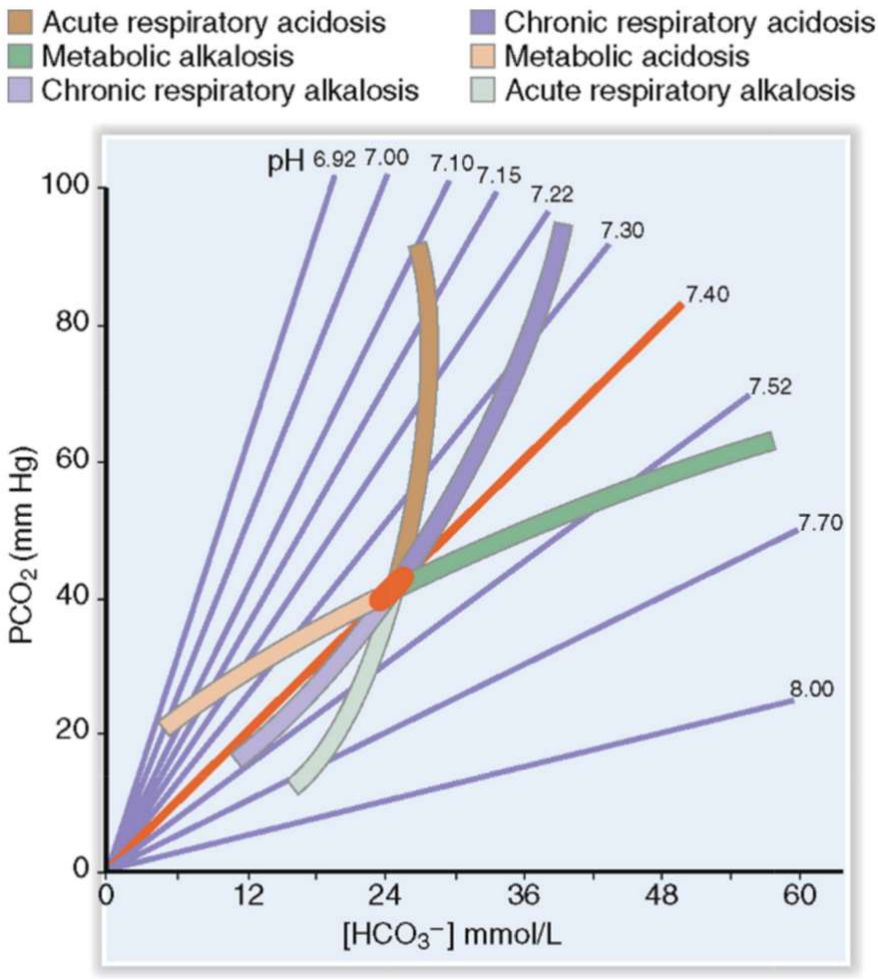


**Figure 4: The Relationship Between Blood pH, $pCO_2$, and $HCO_3^-$ [17].**

Another value used in assessment of the acid-base balance is the base excess (BE). The BE is a reference variable derived from the pH, $pCO_2$, hemoglobin concentration and arterial saturation, as seen in Equation (4). Equation (4) is the Zander variation of the Siggaard-Andersen equation [20]. BE is defined as the concentration of titratable H+ required to return pH to 7.4 when PCO2 is constant at 40 mmHg [21]. The normal range is +/- 2 mEq/L [14]. This value is primarily used to assess the metabolic contribution to an acid-base disorder. If the value of BE is greater than 2mEq/L, it is indicative of metabolic Alkalosis, where a value less than -2mEq/L indicates metabolic

acidosis [17]. Recognizing the BE value is helpful in classifying the cause of the disorder and taking appropriate measure to fix it. Equation (4) follows:

$$\mathrm{BE} = \left(1 - 0.0143 * \mathrm{Hgb}\frac{\mathrm{g}}{\mathrm{dL}}\right) * \left[\left\{0.03\ \frac{\mathrm{mmol}}{\mathrm{mmHg}} * \mathrm{pCO_2} * 10^{\mathrm{pH}-6.1} - 24.26\right\} + (9.5 + 1.63 * \mathrm{Hgb}) * (\mathrm{pH} - 7.4)\right] - 0.2 * \mathrm{Hgb} * (1 - \mathrm{S_aO_2}). \quad (4)$$

Simulated changes in pH will allow students to troubleshoot different scenarios that can be present during a surgical case. A low pH is indicative of acidosis, but supporting values such as $CO_2$ and $HCO_3^-$ can help to determine the cause and find an appropriate response. The simulator will use Equations (3) and (4) to replicate scenarios that represent acid-base disorders that can occur, such as respiratory acidosis or metabolic alkalosis. This allows an instructor to change values that a student is observing on a monitor. It will test the ability of the student to recognize changes in their scan of the CPB circuit and understanding of the body's compensatory mechanisms.

### 4.2 The Oxyhemoglobin Dissociation Curve

The oxyhemoglobin dissociation curve, shown in Figure 5, is representative of the nonlinear relationship between hemoglobin (Hgb) and oxygen ($O_2$) in the blood. As the partial pressure of oxygen ($pO_2$) increases, so does the percentage of Hgb bound to $O_2$, also referred to as saturation [12]. The curve shows another homeostatic mechanism of the body. The plateaued region at the top of the curve from 60 mmHg of $pO_2$ and up shows that the oxygen saturation is around 90% or greater. The normal alveolar $pO_2$ is around 104 mmHg with a saturation of 95-97% [12].

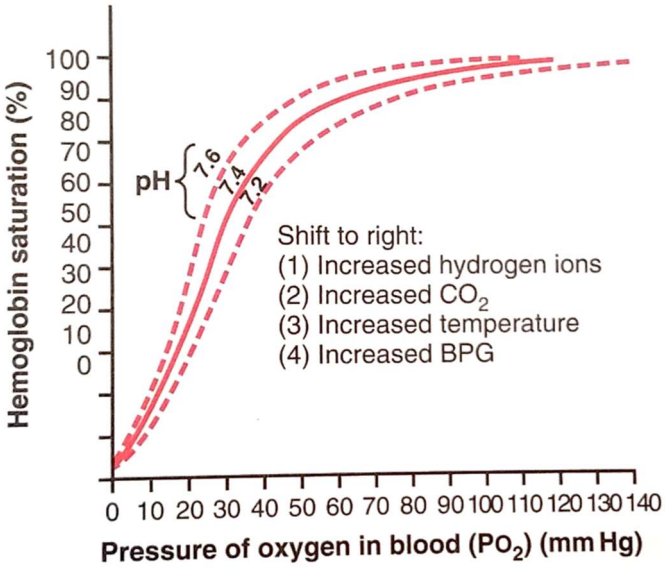


**Figure 5: The Oxyhemoglobin Dissociation Curve [12].**

The plateau of the curve is significant because it shows that the body can maintain relatively normal saturations when there are large shifts in $pO_2$. The steep portion of the curve is an important indicator of the release of $O_2$ in the tissues. As blood transports $O_2$ through the capillaries, the $pO_2$ follows the gradient for diffusion from hemoglobin into the tissues. The result is a drop in $pO_2$ to around 40 mmHg, which is normal for venous blood [12]. Equation (5) is a simplified predictive model developed by Severinghaus [22] that calculates the saturation based on the $pO_2$ following the dissociation curve:

$$SO_2(\%) = \left[\left(\left(\left(PO_2{}^3 + 150 * PO_2\right)^{-1} * 23400\right) + 1\right)^{-1}\right]. \quad (5)$$

There are many factors that can affect the oxyhemoglobin dissociation curve, such as temperature, $pCO_2$, and pH [12]. Changes in these variables can change the Hgb affinity for $O_2$, which can result in shifts in the oxyhemoglobin dissociation curve [14].

Equation (5) is limited in regards to reflecting shifts in the curve, but it provides a simple but realistic physiological model to estimate the arterial saturation and can be easily implemented for simulation purposes.

Although the arterial saturation is important to determine if hemoglobin is binding to oxygen for transport, it does not provide the $O_2$ content being delivered to the body. A more telling sign of $O_2$ transport is the mixed venous saturation ($S_VO_2$). The $S_VO_2$ can help indicate whether the $O_2$ supply is meeting the demands of the body. The normal $S_VO_2$ is around 75%; if this value experiences a change greater than 10%, it can be indicative of an issue with perfusion [14]. The gas transfer equations – Equation (6), (7), (8), (9) -- and blood gas values can give more insight into the oxygen transport of the patient [14]:

$$O_2 \text{ Capacity } \left(\frac{\text{mL } O_2}{\text{dL}}\right) = 1.34*\text{Hgb} + 0.003*pO_2, \tag{6}$$

$$O_2 \text{ Content } \left(\frac{\text{mL } O_2}{\text{dL}}\right) = 1.34*\text{Hgb}*\%\text{ Saturation(decimal form)} + 0.003*pO_2, \tag{7}$$

$$O_2 \text{ Saturation} = O_2 \text{ Content} / O_2 \text{ Capacity}, \tag{8}$$

$$\text{and } O_2 \text{ Consumption } \left(\frac{\text{mL } O_2}{\text{min}}\right) = (\Delta O_2\text{Content})* \text{ Flow(L/min)}*10\text{(dL/L)}. \tag{9}$$

Hgb is responsible for about 97% of the oxygen transported from the lungs to the tissues [12]. The saturation of Hgb does not ensure adequate oxygen delivery. During CPB, hemodilution can drop the Hgb concentration. Decreased amounts of Hgb reduce the ability to transfer $O_2$. A scenario like this can be approached by increasing the flow to the patient to increase the $O_2$ delivered or add additional Hgb to increase the carrying

capability. Looking at all the values is important before choosing a course of action. The role of the simulation is to reinforce assessment before action.

The relationship between the oxyhemoglobin dissociation curve and $O_2$ transfer equations is important in providing adequate perfusion to the patient. Understanding the interplay of variables can help in diagnosing cause of changes in $S_VO_2$ or determining if the supply of $O_2$ matches the need. Simulating these variables using Equations (5) through (9) will provide a model that can reflect scenarios such as hypoxia that can be attributed to the multiple factors that affect $O_2$ transfer.

## 4.3 Non-Gas Values Measured by CDI™ 500 Inline Monitoring System

### *4.3.1 Potassium ($K^+$)*

Potassium ($K^+$) is the major intracellular ion. Ninety-eight percent of the potassium in the body is found intracellularly [14]. The normal concentration for $K^+$ in blood is 3.5-5.0 mEq/L, while the intracellular concentration is approximately 140 mEq/L [12]. Conditions such as hyperkalemia and hypokalemia can inhibit the propagation of action potentials [12]. The presence of increased extracellular $K^+$, hyperkalemia, can affect the heart's contraction, rhythm and conduction [12]. A reduced concentration gradient can partially depolarize the membrane, causing a low intensity action potential which can make the contraction of the heart weaker [12]. If the concentration of $K^+$ doubles or triples in the blood, it can cause weak contractions and arrhythmias that can be fatal. This is why it is important to monitor $K^+$ concentration during surgery to avoid impairing function when coming off CPB.

Increased levels of $K^+$ can be caused by multiple conditions or actions. Often during CPB, a cardioplegia solution containing $K^+$ is used to arrest the heart by depolarizing the cells [19]. It is reversible but can increase the amount of $K^+$ in the blood. Hormonal and acid-base responses to CPB can also affect $K^+$ concentration. Cortisol and aldosterone increase the urinary secretion of $K^+$, but kidney dysfunction can result in accumulation [17, 19]. Conditions affecting the concentration of $K^+$ while on CPB can be managed pharmacologically by giving sodium bicarbonate or insulin and glucose, which pulls $K^+$ into the cells [14]. The practice of zero balance ultrafiltration (Z-BUF), which entails hemoconcentrating blood and replacing volume with low $K^+$ solution, is also effective in reducing the concentration [19].

Simulation of ion would be difficult in a mathematical model because of the multitude of factors that can affect it. However, since the simulation is specific to CPB, the instructor can change the value within the operational range to reflect changes that may occur in a surgery or exhibit preexisting conditions of the patient.

### *4.3.2 Temperature*

The temperature of a patient can affect many of the blood gas parameters during CPB. Variations in temperature can cause shifts in the oxyhemoglobin dissociation curve and alter the acid-base relationship by affecting the solubility of gases [12, 17, 19]. Temperature is difficult to model because of its numerous direct and indirect effects.

For simulation purposes, certain temperature effects can be simulated without creating a complex model, but instead by changing values that can be affected by it. An example is the $O_2$ consumption, where a decrease in temperature is associated with

decreased metabolic activity [19]. Changing the temperature and $O_2$ consumption will simulate to a student a change in metabolism due to temperature.

*4.3.3 Hematocrit*

The hematocrit is the percentage of whole blood that is made up of red blood cells. In an average adult, this percentage can range from 38 to 54% [18]. Hematocrit (Hct) indicates the volume of red blood cells, which can approximate oxygen carrying capacity of the blood.

During CPB, the hemodilution from the CPB prime can cause the hematocrit to drop to a range of 22 to 34% [14]. The increase in non-cellular volume can within limits overcome the decreased O2 capacity by increasing the cardiac output from the pump. Moderate hemodilution is seen as an advantage to decrease the viscosity of blood to improve blood flow to microvasculature. However, if the patient becomes too dilute, this can reduce the oxygen delivery to the tissues [19].

A change in Hct can be telling of other conditions, such as anemia or polycythemia [12]. Continuous monitoring of Hct during CPB is a standard in clinical practice [8]. For simulation purposes, the value of Hct will be determined by tripling the Hgb concentration. This estimation is commonly used because the Hgb is the determining factor in oxygen transport.

## 5.0 Implementation of Design

Prior to modeling the systems outlined in Section 4.0, the variables for simulation needed to be classified as independent or dependent. The independent variables were those associated with parameters that can be controlled or affected directly by the actions of the perfusionist. Identifying the variables in such a manner allows for students to enact changes to maintain normal values. The dependent variables are the variables that are indirectly affected. In Table 3, the independent variables are listed with their associated CPB control mechanism. From the independent variables, the remaining variables were derived.

**Table 3: Independent Variables and their Corresponding Mechanism of Control During Cardiopulmonary Bypass [12, 19].**

| Independent Variables | CPB Mechanisms of Control |
|---|---|
| $pCO_2$ | Sweep rate through an oxygenator can alter the amount of $pCO_2$. |
| $pO_2$ | The fraction of inspired oxygen supplied to the oxygenator can be increased or decreased which proportionally changes $PO_2$ in the blood. |
| $HCO_3^-$ | Bicarbonate ion concentration can be altered by adding Sodium bicarbonate. |
| Hgb | The concentration of Hgb can be changed through hemodilution or transfusion. |
| Flow | CPB pump speed is variable. |
| $K^+$ | Potassium concentration can be altered pharmacologically (*i.e.*, insulin, $NaHCO_3$, KCl) or through methods like Z-BUFing. |
| $VO_2$ | Patient $O_2$ consumption can be altered with changed in temperature and use of anesthetic agents. |
| Temperature | The temperature of the patient can be adjusted or maintained during CPB with the use of a heater cooler system. |

The models represented in the simulation are the acid-base and oxyhemoglobin dissociation curve as described in Sections 4.1 and 4.2. Some of the independent variables are not modeled in this simulation due to the complexity of their interactions with other physiologic systems. For simulation purposes, such variables as potassium and temperature were left autonomous from all other variables, but their values could still be changed by the user. They can be adjusted to simply mimic some physiological scenarios seen during CPB, such as cardioplegia delivery, but will have no effect on other values. The use of these values would be instructor dependent.

**5.1 Acid-Base Model**

In order to reproduce the Acid-Base model, the Henderson-Hasselbach equation, Equation (3), was used to determine the pH of the system and the Sigaard-Andersen equation, Equation (4), was used to determine BE. The equations were used to solve for pH and BE because the $HCO_3^-$ and the $pCO_2$ can be altered by actions of the perfusionist. The $HCO_3^-$ concentration can be increased by the addition of sodium bicarbonate to the system [19]. The renal system of the body can also be a determinant of the $HCO_3^-$ concentration -- if there is impaired function the concentration may be low [12]. As for $pCO_2$, the pressure can be changed by altering the sweep rate of the CPB circuit [19]. Since the oxygenators function via diffusion, the pressure gradient across the membrane affects the rate of diffusion [19]. The gas being supplied to the oxygenator is a mixture of oxygen and medical air which contains the normal concentration of carbon dioxide in the room air, which creates the gradient of $CO_2$ from the blood to the oxygenator. Increased sweep rate maintains a larger pressure gradient, which causes a decrease in $pCO_2$ [19].

The use of $pCO_2$ and $HCO_3^-$ as independent variables allows the simulated model to more accurately resemble CPB.

### 5.2 Oxyhemoglobin Dissociation Curve Model

To simulate oxygen delivery and consumption of a patient, Equation (5) and Equations (6) through (9) were used to solve for the saturation of arterial and venous blood. This model was split into two separate equations. The arterial saturation was estimated using Equation (5), which uses the $pO_2$ as its determining factor. The $pO_2$ can be altered by changing the fraction of inspired oxygen ($FiO_2$), which proportionally alters the partial pressure [19].

The model for the venous saturation has multiple determinants. While the measured saturation is typically used to determine oxygen consumption, $S_vO_2$ is a dependent variable that cannot be directly changed. This model was altered to match the effects of CPB and other surgical factors. Independent variables listed in Table 3 that affect the venous saturation include $pO_2$, Hgb, flow, and $VO_2$. The Hgb concentration can be affected by hemodilution, ultrafiltration, blood salvaging, or addition of red blood cells. While the concentration of Hgb may not affect the arterial saturation, it does limit the oxygen carrying capabilities of the blood. If there is not enough oxygen delivered, the $S_vO_2$ will reflect this even if the flow to the patient and metabolic consumption are normal [12]. Blood flow affects the rate at which the oxygen is delivered to the tissues and is controlled by the speed of arterial pump [19]. If the flow is too low, the $S_vO_2$ would decrease because more of the oxygen is extracted from the available content [12]. The final determinant is the consumption of oxygen. Patient metabolism can be affected

by manageable factors, such as anesthetic agents and temperature, or concurrent conditions such as sepsis. In order to solve for $S_vO_2$, Equation (8) was specified for saturation in Equation (10):

$$S_vO_2 = \text{Venous } O_2 \text{ Content} / O_2 \text{ Capacity}. \quad (10)$$

Equation (6) was substituted into Equation (10) to create Equation **Error! Reference source not found.**:

$$S_vO_2 = \text{Venous } O_2 \text{ Content} /(1.34 * \text{Hgb} + 0.003 * pO_2) . \quad (11)$$

Equation (7) could not be used because the equation contained $S_vO_2$ as a variable. Instead, Equation (9) was rearranged to solve for venous oxygen content in Equation (12):

$$C_vO_2 \left(\frac{\text{mL } O_2}{\text{dL}}\right) = C_aO_2 - \frac{VO_2}{(Q * 10)} . \quad (12)$$

Equations (7) and (12) were then substituted into Equation **Error! Reference source not found.** to create the final venous saturation equation, Equation (13):

$$S_vO_2 = \frac{(1.34 * \text{Hgb} * S_aO_2 + 0.003 * pO_2) - \frac{VO_2}{(\text{Flow} * 10)}}{(1.34 * \text{Hgb} + 0.003 * pO_2)}. \quad (13)$$

Equation (13) provides the model to depict $S_vO_2$ changes due to multiple variables that can be controlled during CPB. The model allows students to evaluate changes in the $S_vO_2$ and alter them via CPB practices.

### 5.3 Program Development

To create a user friendly interface, the CBGM simulation was developed using the high level language in MATLAB® version 9.1 [23]. This platform was chosen for familiarity and the intuitive nature of the language. The current platform also has a GUI development environment (GUIDE), which was used to develop and easily edit the layout of the GUI [23]. This environment will also make it easier for additional developments to be carried out in the future.

Before programing, the function and variables of a program were defined. From the equations listed in Sections 5.1 and 5.2, a programmed script was developed to have the independent variables as inputs and dependent variables as outputs of the equations. The inputs needed to be collected from the user interface.

A goal of this simulation is to create an interactive user interface that can be adjusted by an instructor and display blood gas to the student for evaluation. To meet those needs, the program was designed to have two windows. The GUIs were created using GUIDE. One window was designed (Figure 6) to display the blood gas values in the fashion of a Terumo CDI$^{TM}$ 500 display (Section 4.0, Figure 3). The second window was designed to be the controlling GUI that had the capabilities to alter the blood gas values (Figure 7).

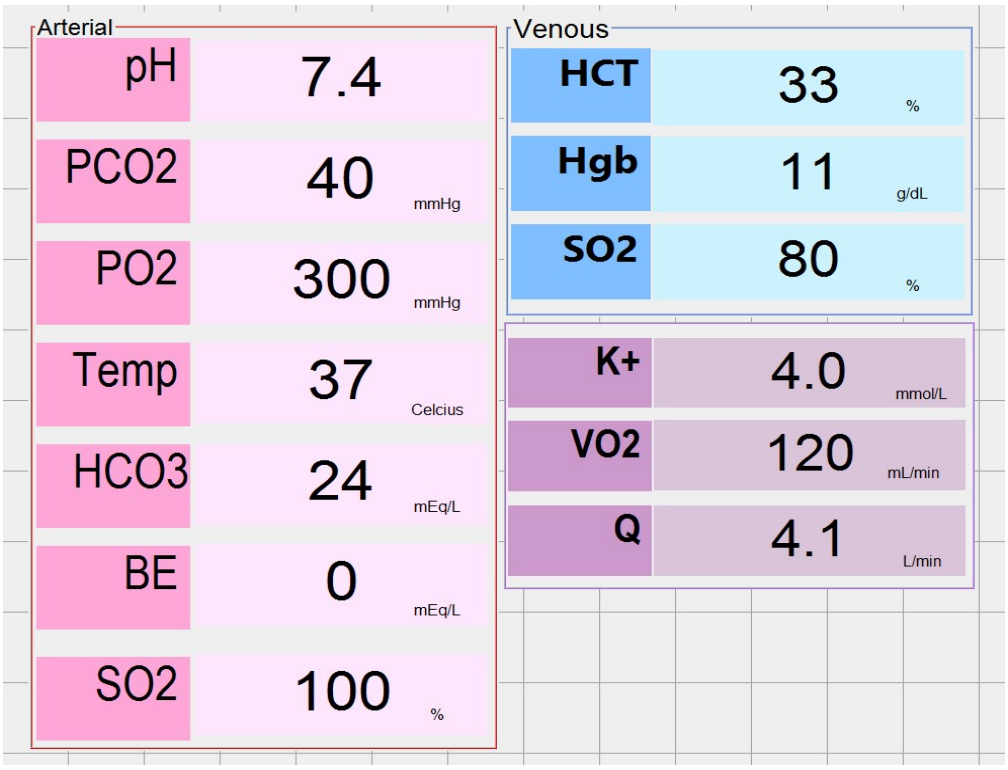


**Figure 6: Monitor Display Design created in MATLAB® GUIDE.**

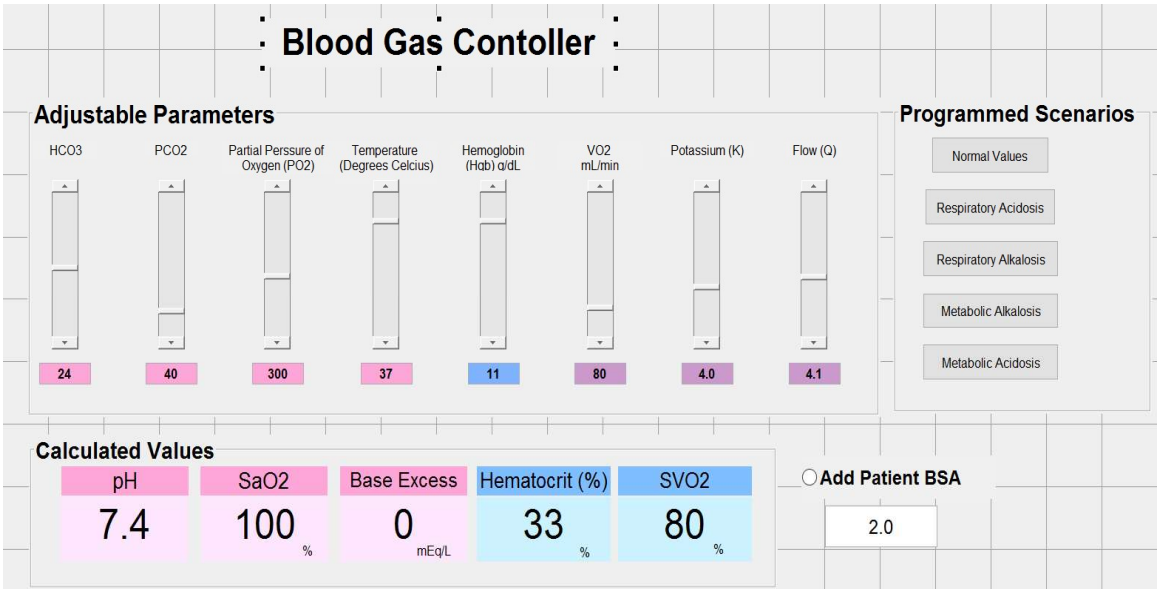


**Figure 7: The User Controlled Interface.**

The display window (Figure 7) has no interactive controls. Its purpose was only to display the data from the control GUI. Static text boxes and panels were used in the display layout. Each component in the display and design window was edited in the property inspector tables or through code. The design of the control window in Figure 7 has many user interface controls (uicontrols). Each independent variable listed in Table 3 was given its own edit box and slider bar uicontrols to change the values easily. The operational values from Table 1 were used to create ranges of values for each independent variable. Additionally, pushbutton controls were created with preset values for easy simulation of specific scenarios. The option for body surface area was also added for creating a patient-specific case; the only variable affected by this option is the $VO_2$, which is then indexed.

After the GUI layouts were complete, they were saved to a single file called CILBGM. The display was saved under the name of CILBGM.fig and the control under CILBGMC2.fig. Saving the work in GUIDE produced a programing script and MATLAB® file of the same name containing the code for each of the GUI (Appendix A). The interactive uicontrol component code was included in the main GUI code for each window. Each uicontrol has a callback function that runs the specific code when the uicontrol is activated [23]. The pseudocode for the simulation is shown in Figure 8.

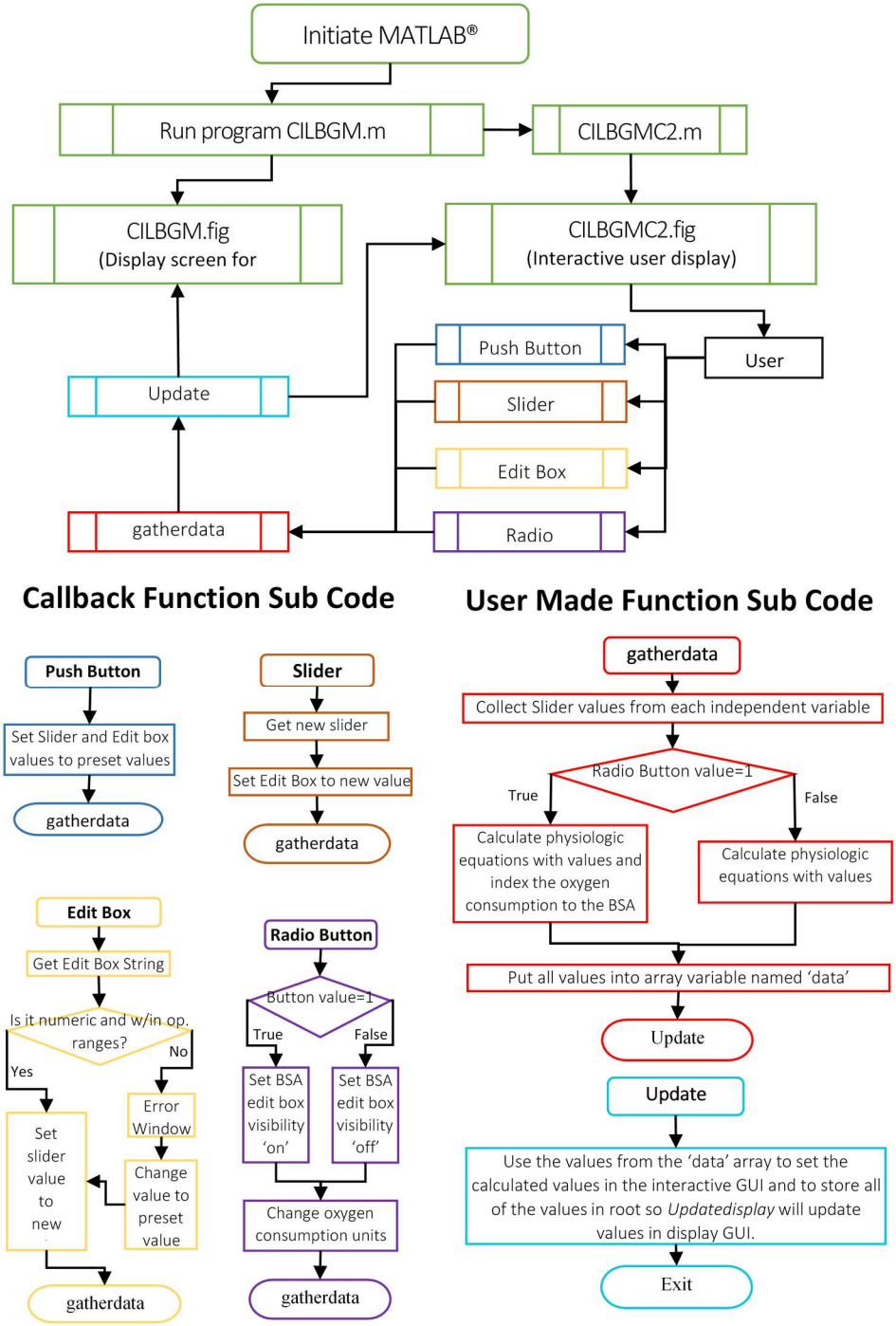


**Figure 8: Pseudocode for Simulation Program.**

The uicontrol code for all the interactive controls in the controller were changed to make sure the proper components were interacting. For the slider bar components, when the slider was moved, the value of the slider changed. However, the value of the edit box associated with the slider would not reflect the value change. Code was added to the slider callback function so that when the slider was activated, the value of the slider would be used to set the text string within the box. The components have different data types and needed to be corrected when sharing data. The edit box had the same problem--when a value was entered, the value did not change the position of the slider. The callback function of the edit box was given code to set the value of the slider when activated. This process was repeated for all the sliders and edit boxes on the controller.

Another control featured on the controlling GUI were multiple pushbuttons. Each pushbutton was created to simulate particular scenarios when the button was pushed. In order to do that, code was added to the pushbutton callback functions to set the variables of all of the independent components to the specified values of the code. These values were numbers chosen out of the normal physiologic range to show how values can be indicative of different conditions.

A final uicontrol on the controlling GUI was a radio button. When activated, an edit box was to display that allowed addition of, or editing, of the BSA. This function was a bit more challenging because the edit box had to be set to invisible in its opening function. The callback function of the radio button needed an “if” statement. When the radio button was active, the value was one, and when it was inactive, it was zero. The if statement was set up so that if the value was one, the edit box visibility would be turned on, but if zero, it would remain off.

After the independent controls worked together, the models needed to be implemented. The function *gatherdata* was created to collect the current value of each independent value, calculate the values of the dependent values, and then create an array containing all of data. This function was used in every unicontrol callback code to ensure that the simulation values were up to date with every user change. From these data, all of the desired values can be displayed.

In order to update the values in the controller, another user function, called Update, was created to take the data array produced from *gatherdata*. This function prevented calculation interruptions by updating all the values at once, after the calculations had been complete. This function also stored the data array in the root and called a function in the display GUI. A final function called *UpdateDisplay*, created in the code of the display GUI, was designed to get the stored data array and update all the values in the display. Additional features were added to the code including error warnings for using incorrect values or values out of the operational range.

### 5.4 Simulation Testing

In order to test the simulation, each programed component needed to be tested. This was accomplished by running the simulation and changing the values of the independent variables. If the change in value appeared in each GUI window, and the components associated with that variable, then the code was functioning properly. Also tested were the error warnings by inputting improper number or characters into the edit boxes.

Testing of the modeling equations was performed by running the simulation and collecting the data points of the related variables. The independent variables were isolated within their specific model to determine the relationship between that variable and dependent variable. The values were collected at multiple points within the simulation's operational range and graphed to visualize if the model reflects behavior seen in physiology. The graphs are shown in Section 6.2.

# 6.0 Results

## 6.1 User Interface Testing

Testing the interactive components showed that they met the goals of the project. When the program CILBGM.m was run in the editor of MATLAB®, it displayed the two GUI windows (Figure 9). When any independent variables were changed in the controller, the program appropriately updated the display and any dependent variables. If values were changed out of range, or incorrect characters were used, error boxes displayed, the program was paused, and the variable was reset to its default value. The pushbuttons returned the values set in the code and updated all of the displayed data. The radio button activation resulted in the pop-up of an editable box for BSA input and the units of consumption were indexed. All the functional components of the program completed their desired task.

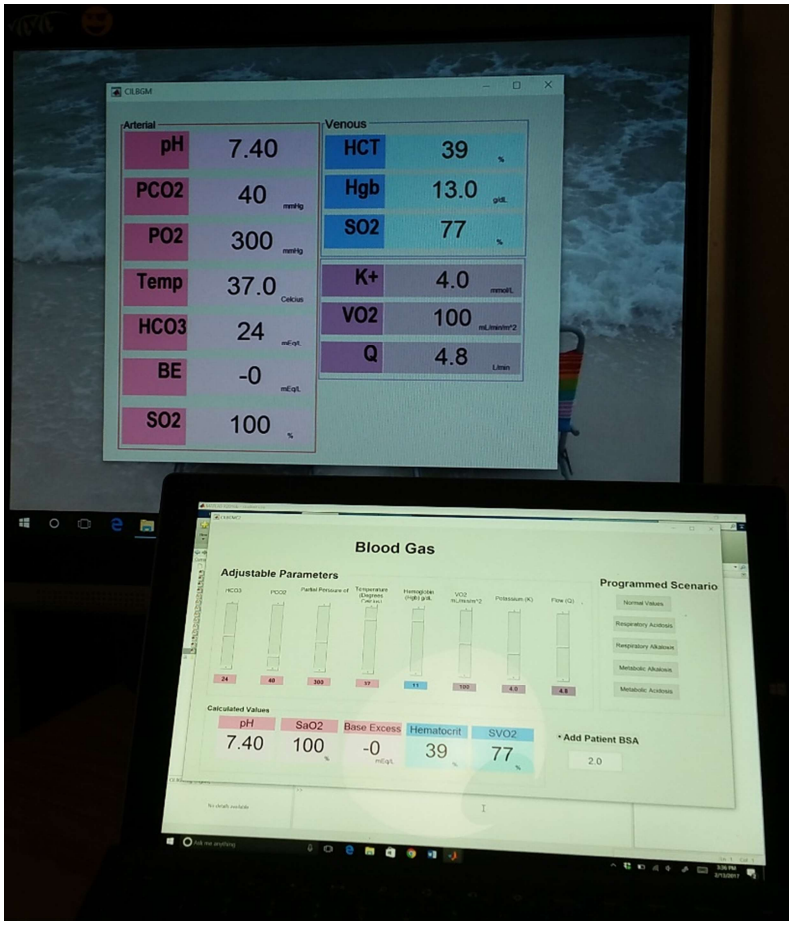


**Figure 9: The Simulation Windows Shown on Two Monitors.**

## 6.2 Physiologic Model Testing

### *6.2.1 Acid-Base Model*

The results of simulating the acid-base model are shown in Figure 10 through Figure 15. The graphs show the relationship of the independent variable with the dependent variable. Figure 10 was generated by holding the $pCO_2$ constant as the concentration of the bicarbonate ion is altered. As shown, the increase in $HCO_3^-$ causes an increase in the pH. In Figure 11, the $HCO_3^-$ was held constant as $pCO_2$ was changed. The increase in $pCO_2$ results in a drop in pH, which is consistent with the physiology that is described in Section 4.1. Figure 12 was generated when both the $pCO_2$ and the $HCO_3^-$

concentration were altered. Their combined effect on pH is a good representation of how the body can compensate to maintain a normal pH. The ranges exceed normal values, but the responses correspond with bodily function.

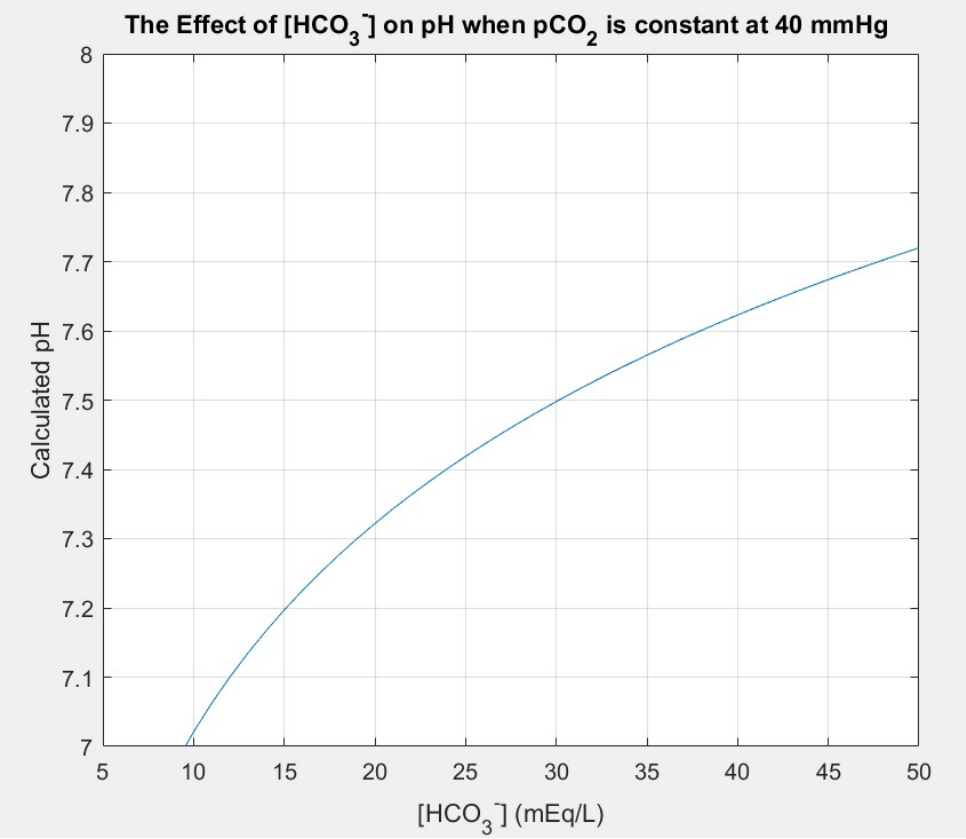


**Figure 10: The Effect of $HCO_3^-$ on the pH.**

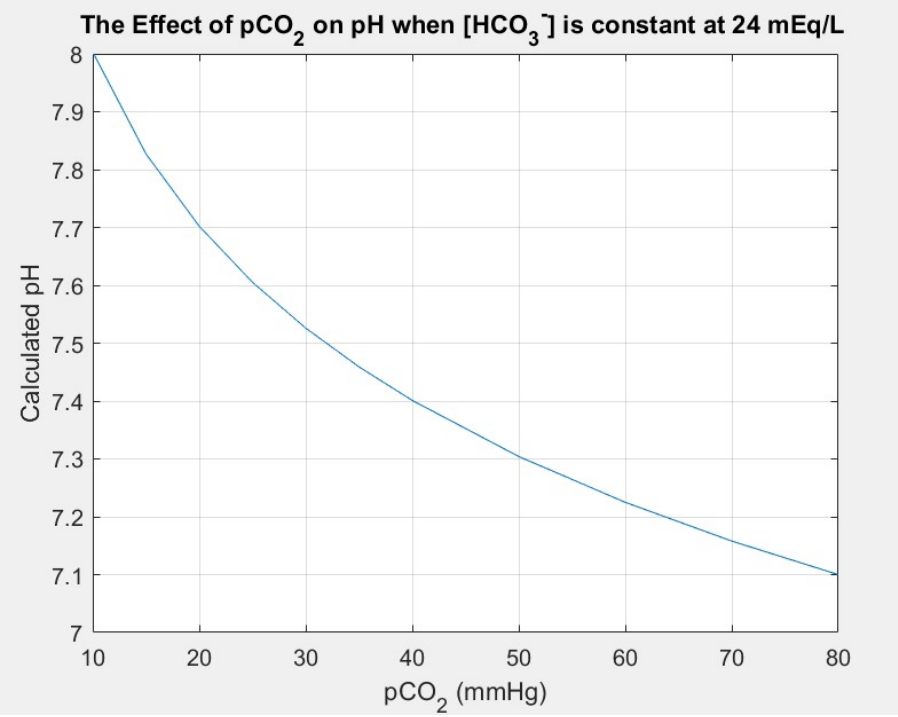


**Figure 11: The Effect of $PCO_2$ when $HCO_3^-$ is Held Constant.**

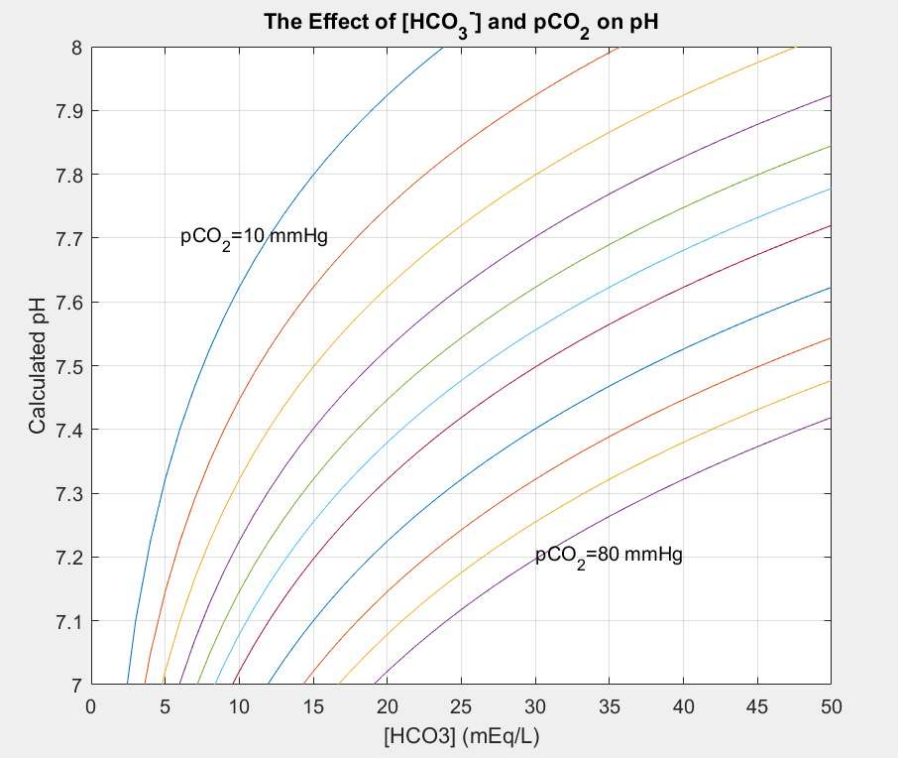


**Figure 12: The Effect of $PCO_2$ and $HCO_3^-$ on pH.**

The BE was also graphed against its determining variables and results are shown in Figure 13 to Figure 16. Each graph isolated one independent variable of the BE. In Figure 13, the increase in $pCO_2$ causes a shift from base excess to base deficit. At around 40 mmHg, which is considered normal, the BE is approximately zero. In Figure 14, the relationship shows the increased $pO_2$ reduces the base deficit when all other variables were held constant.

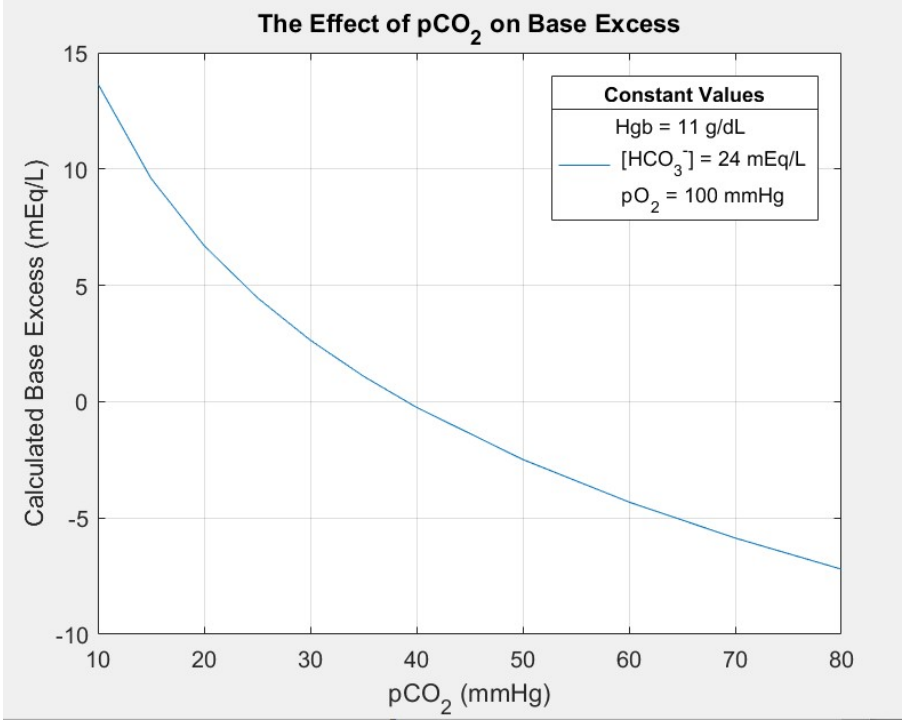


**Figure 13: The Effect of $pCO_2$ on Base Excess.**

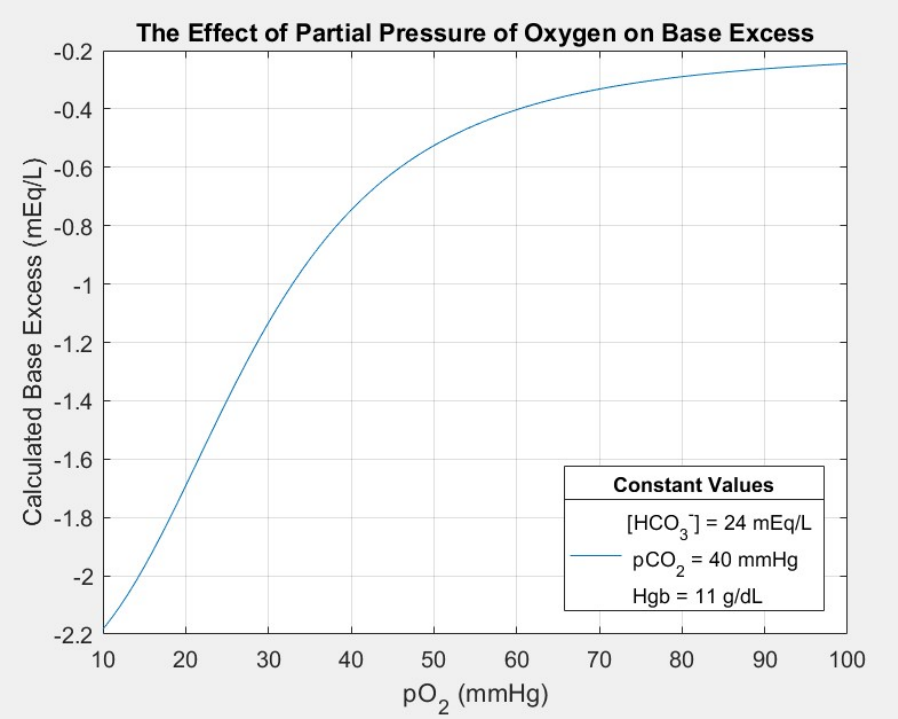


**Figure 14: The Effect of $pO_2$ on Base Excess.**

Figure 15 illustrates the strong effect the increasing of the $HCO_3^-$ has on BE. Recall that BE is an indicator for metabolic contribution to pH, so it makes sense that the $HCO_3^-$ is a dominant factor in BE. The $HCO_3^-$ is the main chemical buffer in the body regulated by the renal system. If that buffering system is not working properly, the result will be evident in the BE.

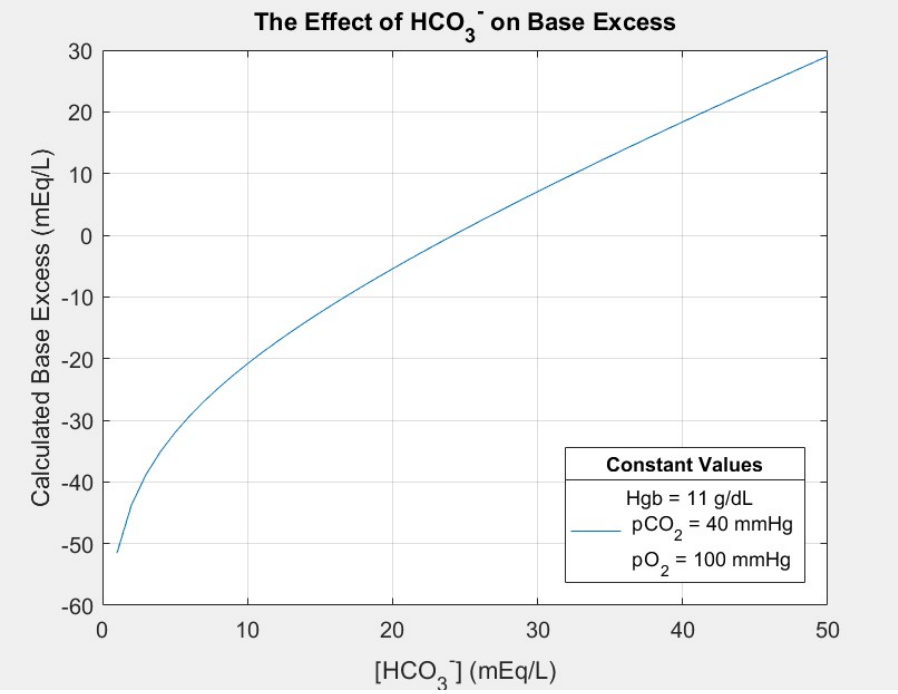


**Figure 15: The Effect of $HCO_3^-$ on Base Excess.**

*6.2.2 The Oxyhemoglobin Dissociation Curve Model*

The dissociation curve and oxygen transfer equations – Equations (5) and (13), respectively-- were tested in the same manner as the acid-base model. The results are shown in Figure 16 to Figure 20. The first graph in the series is the oxyhemoglobin dissociation curve as estimated by the simple Severinghaus equation, Equation (5). The equation appears to be fairly accurate in determining the saturation from the partial pressure. Indicated in Figure 16 are two reference points that list the saturation at the $pO_2$. The values are associated with the normal arterial saturation and normal venous saturation.

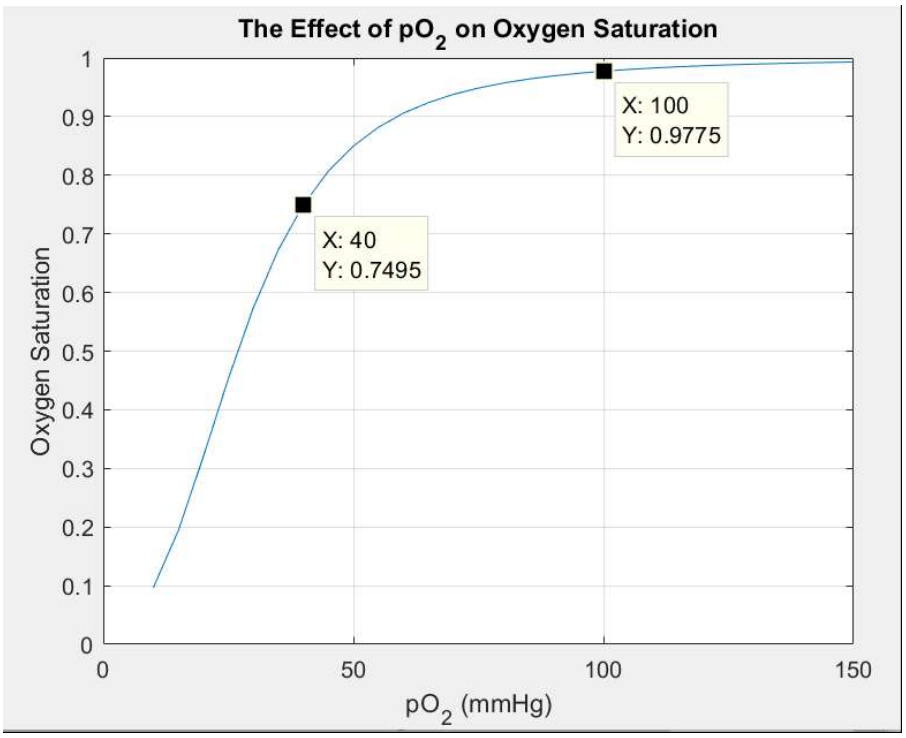


**Figure 16: The Estimated Oxygen Saturation from $pO_2$ using the Severinghaus Equation.**

The remainder of the graphs display the effects of the independent variables on the venous saturation. Increasing the flow (Figure 17), the Hgb (Figure 18), and the $pO_2$ (Figure 19) all increase the $S_vO_2$. The response is reasonable because each variable increased the oxygen delivery capability. One thing to notice is that a decrease in flow (Figure 17) or $pO_2$ (Figure 19) past a certain value, the saturation values become negative in the graph. This cannot happen in real-life and is a consequence of assuming a constant metabolic rate, even when it cannot be met with delivery of oxygen. Because these negative values occur at extremely low flow and $pO_2$, they should not be encountered in the simulation. The graphs also display the many ways the body can compensate to maintain oxygen delivery. The only graph to display a negative correlation to venous saturation is that of oxygen consumption (Figure 20). The response is logical when other

variables are held constant -- the supply is constant but the demand increases. More oxygen is pulled from the blood, which causes the decrease in venous saturation.

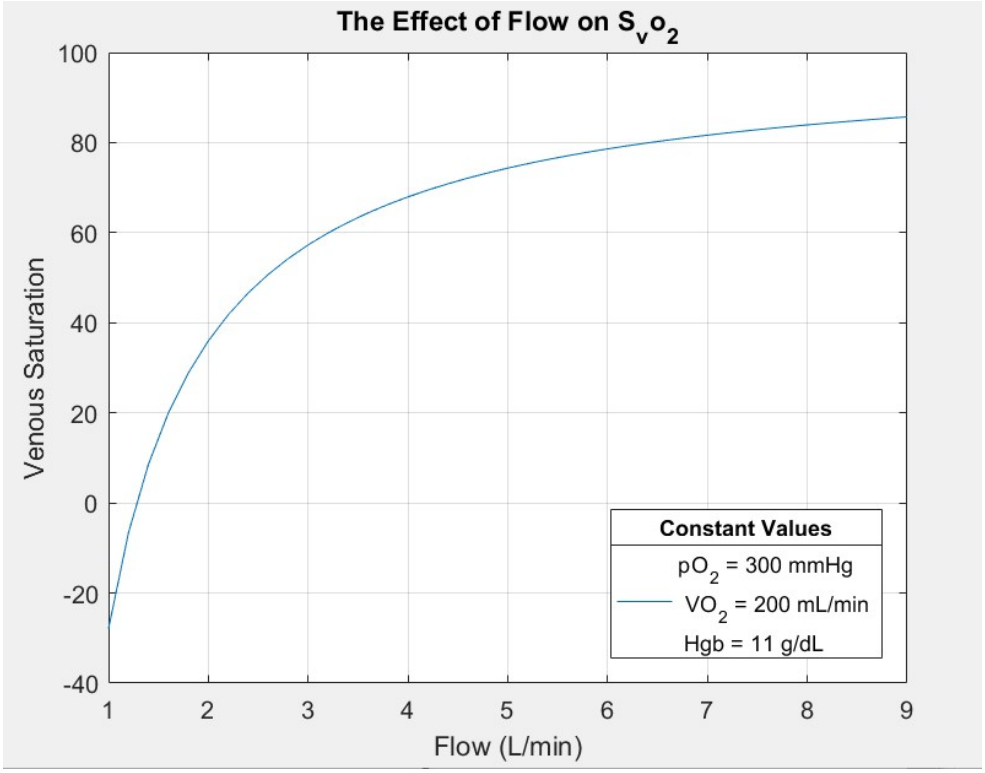


**Figure 17: The Effect of Flow on the Venous Saturation.**

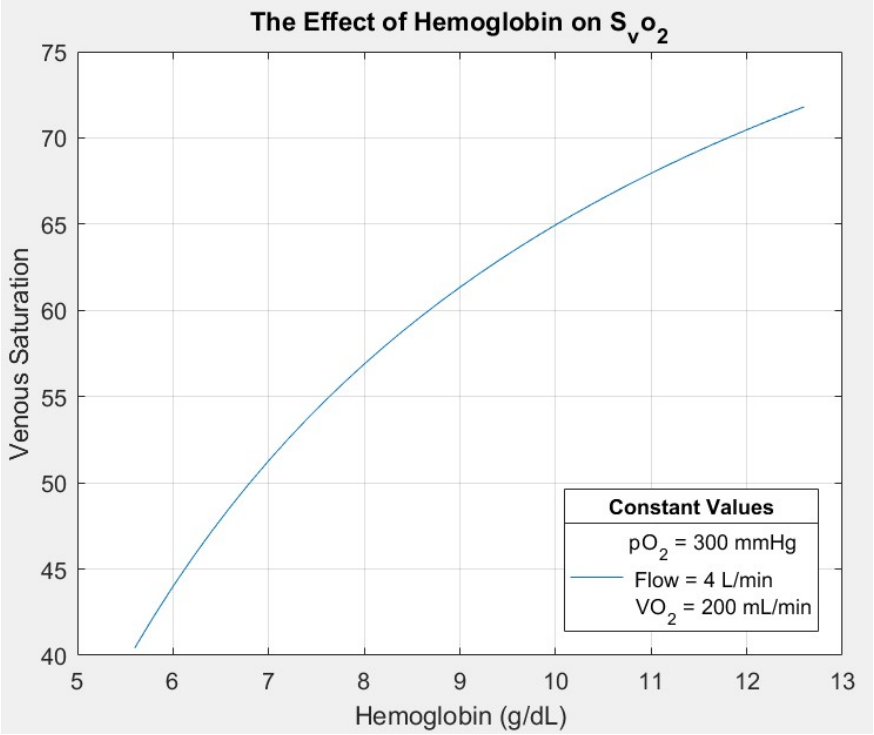


**Figure 18: The Effect of Hemoglobin Concentration on the Venous Saturation.**

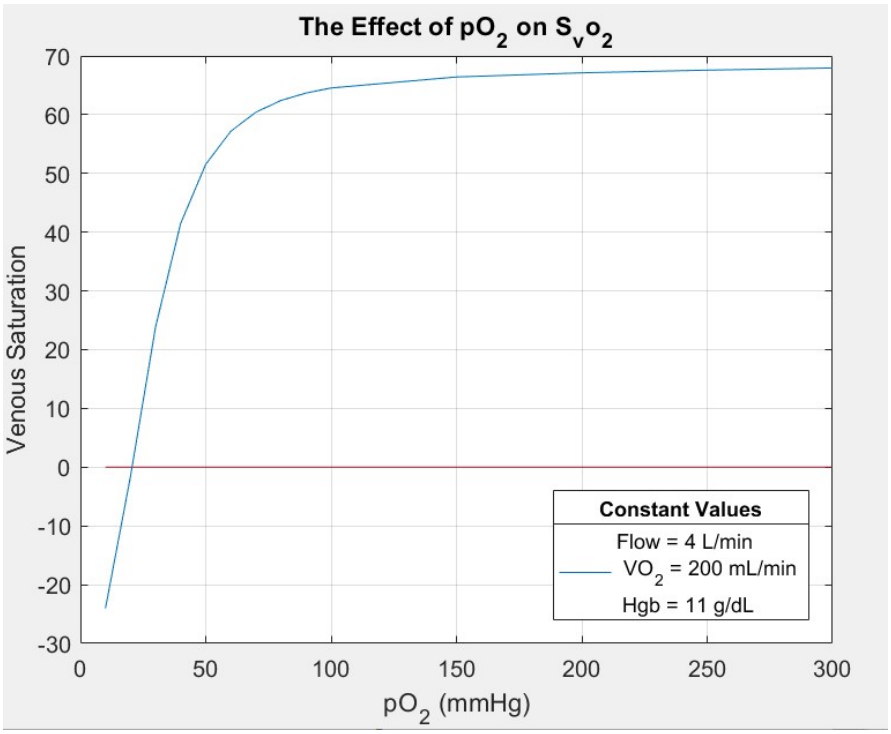


**Figure 19: The Effect of $pO_2$ on Venous Saturation.**

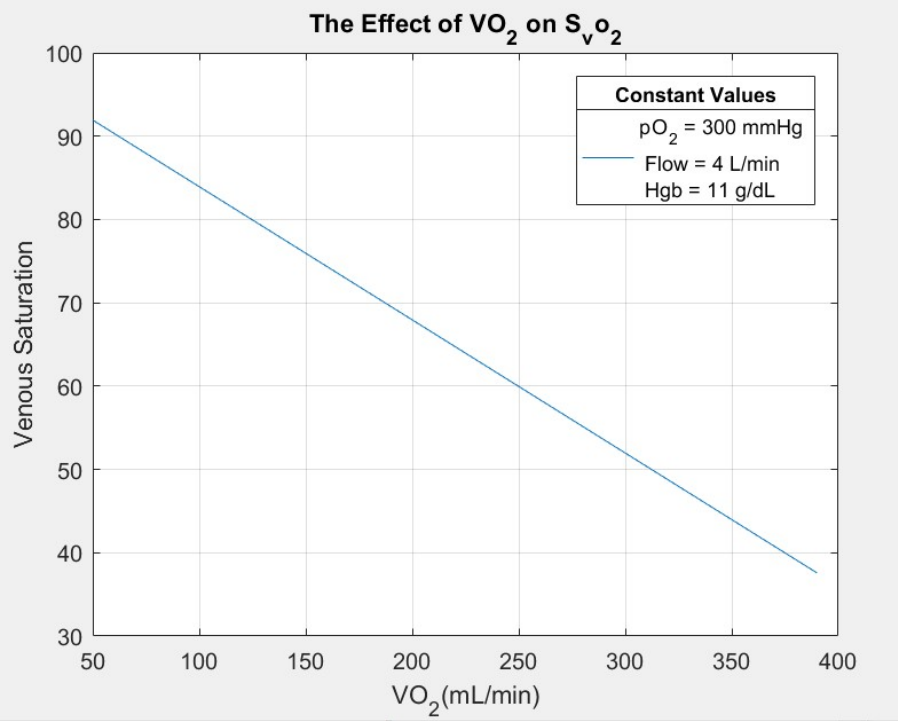


**Figure 20: The Effect of Oxygen Consumption on Venous Saturation.**

## 7.0 Discussion

The virtual simulation designed in this project was done to provide a more complete representation of perfusion when used with the Howard-Varner simulator. From the results, the modeled systems are expected to present physiologically realistic scenarios and responses to user input. Students can now practice monitoring patient blood gas as a part of the simulation, which was not available in the original simulator. Instructors can also use this virtual simulation as an adjunct to class lectures.

The specifications of the project were met by limiting the value ranges in the code to that of the operational ranges from Table 1. Additional code was included to present error messages if the user were to exceed the range or use inappropriate characters. These features were tested during simulation and shown to be effective.

Understanding the homeostatic functions of the body will build students' ability to assess and compensate while on bypass. The modeled systems of simulator were able to depict the relationship of the blood gas values to that of the controlling features of the CPB setup. By changing the values of the simulation, multiple scenarios can be depicted and students can assess and determine the course of action. The instructor will be able to simulate the effect of the action by altering the corresponding variable of the correction. Students will be able to see the result of their choice on the display monitor.

With the addition of the virtual simulator to the current mechanical model, students will be enabled to not only initiate or terminate bypass, but also to simulate situations that occur while on bypass. The software program will prepare students for clinical situations and help them to develop their understanding of homeostatic mechanisms in relation to CPB. By effectively adding this aspect, the simulation at

Milwaukee School of Engineering is now more immersive for students. This software can be incorporated into the simulation and classroom curriculum, which will increase students' understanding and allow them to be more clinically prepared before using their knowledge in the operating room.

### 7.1 Future Improvements

Although the addition of the virtual simulation increases the possible scenarios and utility of the original simulator, there are multiple additions and improvements that can be made. Many improvements were outlined by Howard and Varner [2, 3].

A limiting factor of this project and design are the models used. The simulation was more simplified to establish that a simulation of this kind was feasible. In a future, incorporating more dynamic feedback systems would be beneficial in producing the more realistic simulation for students. A future system could include correction equations to account for changes in temperature for scenarios such as deep hypothermia and pH stat blood management [19]. Additionally, this program could be used as a launching point to create virtual simulation displays for other monitoring systems, such as EKG. Simulating changes in EKG would allow students more opportunities to interpret and understand the meaning behind the changes.

Overall, the simulator program needs to become more computer-based. Simulating scenarios from one modeled program using interactive feedback loops will create the most realistic environment for students to learn and practice skills. It will reduce the necessity of changing components manually and can provide more specific patient scenarios.

## 8.0 Conclusion

The goal of this thesis project was to design and implement a graphical user interface that simulates the function of a continuous blood gas monitor, such that it could be used in conjunction with the Howard-Varner mock circulatory loop. With the successful addition of this program, students now have a more complete simulation model which can include perfusion practices while on CPB.

The additions to the Howard-Varner simulator were successfully incorporated and provided a physiologically accurate blood gas model. It is now possible for students to assess and diagnose changes in blood gas values, and then through CPB controls, to treat the cause. While there are still plenty of improvements that can be made, the current simulator is an effective tool in honing a variety of skills necessary to the practice of perfusion. The perfusion program at MSOE can now incorporate a simulated blood gas monitor into the simulator program and core curriculum. This simulation can be used to teach students and bridge the gap between clinical evaluation and physiological understanding.

## References

[1] Morris, Richard W., Pybus DA.. 2007. ""Orpheus" Cardiopulmonary Bypass Simulation System," *The Journal of Extra-Corporeal Technology* Vol.39 (4), pp. 228-233. Web. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4680687/>

[2] Howard, Jonathan R. 2010. *Design and Evaluation of a Hemodynamic Perfusion Simulator.* Thesis. Milwaukee School of Engineering.

[3] Varner, Caleb S. 2013. *Addition of Pulmonary Circulation and Variable Venous Compliance to the Howard Perfusion Simulator.* Thesis. Milwaukee School of Engineering.

[4] Hensley, Frederick A., Martin DE, Gravlee GP. 2003.*A Practical Approach to Cardiac Anesthesia.* 3rd ed. Philadelphia, Pennsylvania: Lippincott Williams & Wilkins.

[5] Kunkler, Kevin. 2006. "The Role of Medical Simulation: An Overview," *The International Journal of Medical Robotics and Computer Assisted Surgery* Vol.2 (3), pp. 203-10.

[6] Sistino, JJ, Michaud N, Sievert A, Shackelford A. 2011. "Incorporating High Fidelity Simulation into Perfusion Education," *Perfusion* Vol.26 (5), pp. 390-4.

[7] Posch, Benjamin. 2014. *Development of Simulation Curriculum for the Education of Cardiopulmonary Perfusionists.* Thesis. Milwaukee School of Engineering.

[8] Baker, Robert A., Bronson SL, Dickinson TA *et al.* 2013. "Report from AmSECT's International Consortium for Evidence-Based Perfusion: American Society of ExtraCorporeal Technology Standards and Guidelines for Perfusion Practice: 2013," *The Journal of Extra-Corporeal Technology* Vol.45 (3), pp. 156-66.

[9] Trowbridge, C. C., Vasquez M, Stammers AH *et al*. 2000. "The Effects of Continuous Blood Gas Monitoring During Cardiopulmonary Bypass: A Prospective, Randomized Study--Part II," *The Journal of Extra-Corporeal Technology* Vol.32 (3), pp. 129-137.

[10] Trowbridge, C. C., Vasquez M, Stammers AH *et al.* 2000. "The Effects of Continuous Blood Gas Monitoring During Cardiopulmonary Bypass: A Prospective, Randomized Study--Part I," *The Journal of Extra-Corporeal Technology* Vol.32 (3), pp. 120-128.

[11] Ottens, Jane, Tuble SC, Sanderson AJ, Knight JL, Baker RA. 2010. "Improving Cardiopulmonary Bypass: Does Continuous Blood Gas Monitoring Have a Role to Play?" *The Journal of Extra-Corporeal Technology* Vol.42 (3), pp. 191-198.

[12] Hall, John E., Guyton,Arthur C. 2011.*Guyton and Hall textbook of medical physiology.* 12th ed. Philadelphia, PA: Saunders Elsevier.

[13] CDI Blood Parameter Monitoring System 500. Biomedical Safety & Standards. 2012 Aug 15;Sect. 42 (111).

[14] Lich, Brian V., Brown DM2004.*The Manual of Clinical Perfusion.* 2nd ed. Fort Myers, Forida: Perfusion.com, Inc.

[15] Edwards Lifesciences. "Normal Hemodynamic Parameters and Laboratory Values." 2009. Web. February 4, 2017. < http://ht.edwards.com/scin/edwards/sitecollectionimages/edwards/products/presep/ar04313hemodynpocketcard.pdf>.

[16] Anonymous "pH." 2017. Web. February 2, 2017. < https://www.britannica.com/science/pH>.

[17] Damjanov, Ivan. 2009.*Pathophysiology.* Philadelphia, Pennsylvania: Saunders.

[18] Dailey, John F. 2001.*Blood.* 2nd ed. Ipswich, Massachusetts: Medical Consulting Group.

[19] Gravlee, Glenn P., Davis RF, Stammers AH, Ungerleider RM. 2008. *Cardiopulmonary Bypass.* 3. ed. ed. Philadelphia, Pennsylvania: Kluwer, Lippincott, Williams and Wilkins.

[20] Lang, Werner, Zander R. 2002. "The Accuracy of Calculated Base Excess in Blood," *Clinical Chemistry and Laboratory Medicine* Vol.40 (4), pp. 404-410.

[21] Sirker, A. A., Rhodes A, Grounds RM, Bennett ED. 2002. "Acid−base Physiology: the ‘Traditional’ and the 'Modern’ Approaches," *Anaesthesia* Vol.57 (4), pp. 348-56.

[22] Severinghaus, J.W. 1979. "Simple, Accurate Equations for Human Blood O2 Dissociation Computations," *Journal of Applied Physiology* Vol.46 (3), pp. 599-602.

[23] The Mathworks I. *9.1MATLAB Release 2016b.* Natick, Massachusetts, United States : The Mathworks, Inc.

## Appendix A: MATLAB Code

CILBGM.m

```
function varargout = CILBGM(varargin)
% CILBGM MATLAB code for CILBGM.fig
%      CILBGM, by itself, creates a new CILBGM or raises the existing
%      singleton*.
%
%      H = CILBGM returns the handle to a new CILBGM or the handle to
%      the existing singleton*.
%
%      CILBGM('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CILBGM.M with the given input
arguments.
%
%      CILBGM('Property','Value',...) creates a new CILBGM or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before CILBGM_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to CILBGM_OpeningFcn via varargin.
%
%
%
% Edit the above text to modify the response to help CILBGM

% Last Modified by GUIDE v2.5 09-Feb-2017 13:31:51

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CILBGM_OpeningFcn, ...
                   'gui_OutputFcn',  @CILBGM_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before CILBGM is made visible.
function CILBGM_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to CILBGM (see VARARGIN)


% Choose default command line output for CILBGM
handles.output = hObject;

CILBGMC2; % starts the Control figure created with the name CILBGMC2.

% Update handles structure
guidata(hObject, handles);

% sets the Display as the current figure in the stored data so it can
be
% retrieved on any gui.
setappdata(0  , 'Display'    , gcf);
%sets the UpdatDisplay as the callback for the function in this figure
setappdata(gcf, 'UpdateDisplay', @UpdateDisplay)



% --- Outputs from this function are returned to the command line.
function varargout = CILBGM_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function UpdateDisplay
% This function was created to update the data displayed in the CILBGM
% figure to match that created by the controller.

% This equation gets the stored data that saved the display figure with
the
% tag 'Display'.
Display=getappdata(0,'Display');

% Data is then the set to the data stored in the gui. When calling the
% UpdateDisplay function the (Display,'data') must be set using
setappdata.
data=getappdata(Display,'data');
VO2units=getappdata(Display,'VO2units');

handles= guihandles(CILBGM);    % sets the handles to that of figure
CILBGM
%                                 which is being used as the display
%
%   The following are the positions on the collected data in the data
%   array.
%
%   d(1)= pH value
%   d(2)= PCO2 slider value
%   d(3)= PO2 slider value
```

```
%   d(4)= Temp slider Value
%   d(5)= Hgb slider Value
%   d(6)= SVO2 slider Value
%   d(7)= K slider value
%   d(8)= Q slider value
%   d(9)= HCT calculated value
%   d(10)= HCO3 calculated value
%   d(11)= BE calculated Value
%   d(12)= SaO2 calculated Value
%   d(13)= VO2 calculated Value
%
% the following code set the values of the Display to those from the
data
% array that was built in CILBGMC2

set(handles.dVO2units,'String',VO2units);
set(handles.dHCO3text,'String',sprintf('%0.0f',data(1)))
set(handles.PCO2text,'String',sprintf('%0.0f',data(2)))
set(handles.PO2text,'String',sprintf('%0.0f',data(3)))
set(handles.Temptext,'String',sprintf('%0.1f',data(4)))
set(handles.pHtext,'String',sprintf('%0.2f',data(10)))
set(handles.dBEtext,'String',sprintf('%0.0f',data(11)))
set(handles.dSO2text,'String',sprintf('%0.0f',data(12)))
set(handles.dHCTtext,'String',sprintf('%0.0f',data(9)))
set(handles.Hgbtext,'String',sprintf('%0.1f',data(5)))
set(handles.dVO2text,'String',sprintf('%0.0f',data(6)))
set(handles.Ktext,'String',sprintf('%0.1f',data(7)))
set(handles.SVO2text,'String',sprintf('%0.0f',data(13)))
set(handles.Qtext,'String',sprintf('%0.1f',data(8)))
```

CILBGMC2.m

```
function varargout = CILBGMC2(varargin)
% CILBGMC2 MATLAB code for CILBGMC2.fig
%      CILBGMC2, by itself, creates a new CILBGMC2 or raises the
existing
%      singleton*.
%
%      H = CILBGMC2 returns the handle to a new CILBGMC2 or the handle
to
%      the existing singleton*.
%
%      CILBGMC2('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in CILBGMC2.M with the given input
arguments.
%
%      CILBGMC2('Property','Value',...) creates a new CILBGMC2 or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before CILBGMC2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to CILBGMC2_OpeningFcn via
varargin.
```

```
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help CILBGMC2

% Last Modified by GUIDE v2.5 07-Feb-2017 21:15:42

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CILBGMC2_OpeningFcn, ...
                   'gui_OutputFcn',  @CILBGMC2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before CILBGMC2 is made visible.
function CILBGMC2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to CILBGMC2 (see VARARGIN)

% Choose default command line output for CILBGMC2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);




% --- Outputs from this function are returned to the command line.
function varargout = CILBGMC2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
```

```
varargout{1} = handles.output;



function HCO3edit_Callback(hObject, eventdata, handles)
% hObject    handle to HCO3edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% If the edit callback is activated this code will run.
%
% The string from the edit box is converted to a number then set to the
% variable HCO3edit
HCO3edit=str2double(get(handles.HCO3edit,'String'));
% If this value from the string is not a number or is out of the value
% range of the slider then an error display .
if (isnan(HCO3edit)==1)||(HCO3edit >
get(handles.HCO3slider,'Max'))||...
        (HCO3edit < get(handles.HCO3slider,'Min'))
        E=errordlg(...
            sprintf('A numeric value within the range of \n0 to 50
mEq/L of must be entered!',...
            'HC03 Error')); %error display code
        waitfor(E); % the error box will remain until closed
        set(handles.HCO3edit,'String','24') % the value will be cahnged
to a normal value
        set(handles.HCO3slider,'Value',24)
else % if the input data is a number with in the proper range then the
value
    %will set the slider value and location.
set(handles.HCO3slider,'Value',HCO3edit)
end

% collects data from all sliders and calculates it
data=gatherdata(handles);
% takes the calculated data and updates the displays
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function HCO3edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to HCO3edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function PCO2edit_Callback(hObject, eventdata, handles)
% hObject    handle to PCO2edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% If the edit callback is activated this code will run.
%
% The string from the edit box is converted to a number then set to the
% variable PCO2edit
PCO2edit=str2double(get(handles.PCO2edit,'String'));
% If this value from the string is not a number or is out of the value
% range of the slider then an error display .
if (isnan(PCO2edit)==1)||(PCO2edit >
get(handles.PCO2slider,'Max'))||...
        (PCO2edit < get(handles.PCO2slider,'Min'));
        E=errordlg(...
            sprintf('A numeric value within the range of \n10 to 80
mmHg of must be entered!',...
            'PCO2 Error'));
        waitfor(E);
        set(handles.PCO2edit,'String','40')
        set(handles.PCO2slider,'Value',40)
else% if the input data is a number with in the proper range then the
value
    %will set the slider value and location.
set(handles.PCO2slider,'Value',PCO2edit)
end
data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function PCO2edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PCO2edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function PO2edit_Callback(hObject, eventdata, handles)
% hObject    handle to PO2edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% If the edit callback is activated this code will run.
%
% The string from the edit box is converted to a number then set to the
```

```
% variable PCO2edit
PO2edit=str2double(get(handles.PO2edit,'String'));
% If this value from the string is not a number or is out of the value
% range of the slider then an error display .
if (isnan(PO2edit)==1)||(PO2edit > get(handles.PO2slider,'Max'))||...
        (PO2edit < get(handles.PO2slider,'Min'));
        E=errordlg(...
            sprintf('A numeric value within the range of \n20 to 500
mmHg of must be entered!',...
            'PO2 Error'));
        waitfor(E);
        set(handles.PO2edit,'String','300')
        set(handles.PO2slider,'Value',300)
else% if the input data is a number with in the proper range then the
value
    %will set the slider value and location.
set(handles.PO2slider,'Value',PO2edit)
end
data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function PO2edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PO2edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Tempedit_Callback(hObject, eventdata, handles)
% hObject    handle to Tempedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% If the edit callback is activated this code will run.
%
% The string from the edit box is converted to a number then set to the
% variable Tempedit
Tempedit=str2double(get(handles.Tempedit,'String'));
% If this value from the string is not a number or is out of the value
% range of the slider then an error display .
if (isnan(Tempedit)==1)||(Tempedit >
get(handles.Tempslider,'Max'))||...
        (Tempedit < get(handles.Tempslider,'Min'));
        E=errordlg(...
```

```
            sprintf('A numeric value within the range of \n15 to 40 C
of must be entered!',...
            'Temp Error'));
        waitfor(E);
        set(handles.Tempedit,'String','37')
        set(handles.Tempslider,'Value',37)
else % if the input data is a number with in the proper range then the
value
    %will set the slider value and location.
set(handles.Tempslider,'Value',Tempedit)
end

data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);


% --- Executes during object creation, after setting all properties.
function Tempedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Tempedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Hgbedit_Callback(hObject, eventdata, handles)
% hObject    handle to Hgbedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% If the edit callback is activated this code will run.
%
% The string from the edit box is converted to a number then set to the
% variable Hgbedit
Hgbedit=str2double(get(handles.Hgbedit,'String'));
% If this value from the string is not a number or is out of the value
% range of the slider then an error display .
if (isnan(Hgbedit)==1)||(Hgbedit > get(handles.Hgbslider,'Max'))||...
        (Hgbedit < get(handles.Hgbslider,'Min'));
        E=errordlg(...
            sprintf('A numeric value within the range of \n5.6 to 12.6
g/dL of must be entered!',...
            'Hgb Error'));
        waitfor(E);
        set(handles.Hgbedit,'String','11')
        set(handles.Hgbslider,'Value',11)
else % if the input data is a number with in the proper range then the
value
    %will set the slider value and location.
set(handles.Hgbslider,'Value',Hgbedit)
```

```
end


data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function Hgbedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Hgbedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function VO2edit_Callback(hObject, eventdata, handles)
% hObject    handle to VO2edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
VO2edit=str2double(get(handles.VO2edit,'String'));
if (isnan(VO2edit)==1)||(VO2edit > get(handles.VO2slider,'Max'))||...
        (VO2edit < get(handles.VO2slider,'Min'));
        E=errordlg(...
            sprintf('A numeric value within the range of \n10 to 400
mL/min of must be entered!',...
            'VO2 Error'));
        waitfor(E);
        set(handles.VO2edit,'String','150')
        set(handles.VO2slider,'Value',150)
else
set(handles.VO2slider,'Value',VO2edit)
end
data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function VO2edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to VO2edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Kedit_Callback(hObject, eventdata, handles)
% hObject    handle to Kedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Kedit=str2double(get(handles.Kedit,'String'));
if (isnan(Kedit)==1)||(Kedit > get(handles.Kslider,'Max'))||...
        (Kedit < get(handles.Kslider,'Min'));
        E=errordlg(...
            sprintf('A numeric value within the range of \n1.0 to 9.9
mmol/L of must be entered!',...
            'K+ Error'));
        waitfor(E);
        set(handles.Kedit,'String','3.8')
        set(handles.Kslider,'Value',3.8)
else
set(handles.VO2slider,'Value',Kedit)
end
data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function Kedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Kedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Qedit_Callback(hObject, eventdata, handles)
% hObject    handle to Qedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Qedit=str2double(get(handles.Qedit,'String'));
if (isnan(Qedit)==1)||(Qedit > get(handles.Qslider,'Max'))||...
        (Qedit < get(handles.Qslider,'Min'));
        E=errordlg(...
```

```
            sprintf('A numeric value within the range of \n0 to 9.9
L/min of must be entered!',...
            'Flow Error'));
        waitfor(E);
        set(handles.Qedit,'String','4.0')
        set(handles.Qslider,'Value',4.0)
else
set(handles.Qslider,'Value',Qedit)
end

data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);




% --- Executes during object creation, after setting all properties.
function Qedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Qedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%% SLIDERS
% --- Executes on slider movement.
function HCO3slider_Callback(hObject, eventdata, handles)
% hObject    handle to HCO3slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.HCO3edit,'String',sprintf('%0.0f',get(handles.HCO3slider,'V
alue')))

data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);




% --- Executes during object creation, after setting all properties.
function HCO3slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to HCO3slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
```

```
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function PCO2slider_Callback(hObject, eventdata, handles)
% hObject    handle to PCO2slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.PCO2edit,'String',sprintf('%0.0f',get(handles.PCO2slider,'V
alue')))

data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function PCO2slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PCO2slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function PO2slider_Callback(hObject, eventdata, handles)
% hObject    handle to PO2slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.PO2edit,'String',sprintf('%0.0f',get(handles.PO2slider,'Val
ue')))


data=gatherdata(handles);
 %collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function PO2slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PO2slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Tempslider_Callback(hObject, eventdata, handles)
% hObject    handle to Tempslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.Tempedit,'String',sprintf('%0.1f',get(handles.Tempslider,'V
alue')))

data=gatherdata(handles);
Update(handles,data);


% --- Executes during object creation, after setting all properties.
function Tempslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Tempslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Hgbslider_Callback(hObject, eventdata, handles)
% hObject    handle to Hgbslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.Hgbedit,'String',sprintf('%0.0f',get(handles.Hgbslider,'Val
ue')))

data=gatherdata(handles);
%collects data from all sliders and calculates it
Update(handles,data);




% --- Executes during object creation, after setting all properties.
function Hgbslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Hgbslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function VO2slider_Callback(hObject, eventdata, handles)
% hObject    handle to VO2slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.VO2edit,'String',sprintf('%0.1f',get(handles.VO2slider,'Val
ue')))

data=gatherdata(handles);

%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function VO2slider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to VO2slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Kslider_Callback(hObject, eventdata, handles)
% hObject    handle to Kslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.Kedit,'String',sprintf('%0.0f',get(handles.Kslider,'Value')
))

data=gatherdata(handles); %collects data from all sliders and
calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function Kslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Kslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function Qslider_Callback(hObject, eventdata, handles)
% hObject    handle to Qslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.Qedit,'String',sprintf('%0.1f',get(handles.Qslider,'Value')
))

data=gatherdata(handles);
%data=gatherdata(handles); %collects data from all sliders and
calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function Qslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Qslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

%% BUTTONS

% --- Executes on button press in NVButton.
function NVButton_Callback(hObject, eventdata, handles)
% hObject    handle to NVButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.HCO3slider,'Value',24)
set(handles.HCO3edit,'String','24')
set(handles.PCO2slider,'Value',40)
set(handles.PCO2edit,'String','40')
set(handles.PO2slider,'Value',300)
set(handles.PO2edit,'String','300')
set(handles.Tempslider,'Value',37)
set(handles.Tempedit,'String','37')
set(handles.Hgbslider,'Value',11)
set(handles.Hgbedit,'String','11')
set(handles.VO2slider,'Value',150)
set(handles.VO2edit,'String','150')
set(handles.Kslider,'Value',4.0)
```

```
set(handles.HCO3edit,'String','4.0')
set(handles.Qslider,'Value',4.1)
set(handles.Qedit,'String','4.1')

data=gatherdata(handles);

Update(handles,data);




% --- Executes on button press in RAButton.
function RAButton_Callback(hObject, eventdata, handles)
% hObject    handle to RAButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.HCO3slider,'Value',24)
set(handles.HCO3edit,'String','24')
set(handles.PCO2slider,'Value',60)
set(handles.PCO2edit,'String','60')
set(handles.PO2slider,'Value',100)
set(handles.PO2edit,'String','100')
set(handles.Tempslider,'Value',37)
set(handles.Tempedit,'String','37')
set(handles.Hgbslider,'Value',11)
set(handles.Hgbedit,'String','11')
set(handles.VO2slider,'Value',150)
set(handles.VO2edit,'String','150')
set(handles.Kslider,'Value',4.0)
set(handles.HCO3edit,'String','4.0')
set(handles.Qslider,'Value',4.1)
set(handles.Qedit,'String','4.1')

data=gatherdata(handles);

Update(handles,data);



% --- Executes on button press in RAKButton.
function RAKButton_Callback(hObject, eventdata, handles)
% hObject    handle to RAKButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.HCO3slider,'Value',24)
set(handles.HCO3edit,'String','24')
set(handles.PCO2slider,'Value',25)
set(handles.PCO2edit,'String','25')
set(handles.PO2slider,'Value',300)
set(handles.PO2edit,'String','300')
set(handles.Tempslider,'Value',37)
set(handles.Tempedit,'String','37')
set(handles.Hgbslider,'Value',11)
set(handles.Hgbedit,'String','11')
```

```
set(handles.VO2slider,'Value',150)
set(handles.VO2edit,'String','150')
set(handles.Kslider,'Value',4.0)
set(handles.HCO3edit,'String','4.0')
set(handles.Qslider,'Value',4.1)
set(handles.Qedit,'String','4.1')

data=gatherdata(handles);
Update(handles,data);


% --- Executes on button press in MAButton.
function MAButton_Callback(hObject, eventdata, handles)
% hObject    handle to MAButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.HCO3slider,'Value',16)
set(handles.HCO3edit,'String','16')
set(handles.PCO2slider,'Value',40)
set(handles.PCO2edit,'String','40')
set(handles.PO2slider,'Value',300)
set(handles.PO2edit,'String','300')
set(handles.Tempslider,'Value',37)
set(handles.Tempedit,'String','37')
set(handles.Hgbslider,'Value',11)
set(handles.Hgbedit,'String','11')
set(handles.VO2slider,'Value',150)
set(handles.VO2edit,'String','150')
set(handles.Kslider,'Value',4.0)
set(handles.HCO3edit,'String','4.0')
set(handles.Qslider,'Value',4)
set(handles.Qedit,'String','4.0')

data=gatherdata(handles);
Update(handles,data);



% --- Executes on button press in MAKButton.
function MAKButton_Callback(hObject, eventdata, handles)
% hObject    handle to MAKButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.HCO3slider,'Value',34)
set(handles.HCO3edit,'String','34')
set(handles.PCO2slider,'Value',41)
set(handles.PCO2edit,'String','41')
set(handles.PO2slider,'Value',300)
set(handles.PO2edit,'String','300')
set(handles.Tempslider,'Value',37)
set(handles.Tempedit,'String','37')
set(handles.Hgbslider,'Value',11)
set(handles.Hgbedit,'String','11')
set(handles.VO2slider,'Value',150)
set(handles.VO2edit,'String','150')
```

```
set(handles.Kslider,'Value',4.0)
set(handles.HCO3edit,'String','4.0')
set(handles.Qslider,'Value',4.1)
set(handles.Qedit,'String','4.1')

data=gatherdata(handles);
Update(handles,data);


function Update(handles,data)
%the purpose of this function is to update the visible variables on the
%control figure then call the update function for the display figure.
The
%UpdateDisplay function call back was saved in the gui data for CILBGM.
THe
%data adjusted and calculated variables are set with setappdata to the
%CILBGM figure. They can be accessed  in the UpdateDisplay function.

set(handles.HCTtext,'String',sprintf('%0.0f',data(9)))
set(handles.pHtext,'String',sprintf('%0.2f',data(10)))
set(handles.BEtext,'String',sprintf('%0.0f',data(11)))
set(handles.SO2text,'String',sprintf('%0.0f',data(12)))
set(handles.SVO2text,'String',sprintf('%0.0f',data(13)))


Display = getappdata(0, 'Display');
UpdateDisplay = getappdata(Display, 'UpdateDisplay');
setappdata(Display,'data',data);

if get(handles.BSARB,'Value')
    setappdata(Display,'VO2units','mL/min/m^2');
else
    setappdata(Display,'VO2units','mL/min');
end
feval(UpdateDisplay);


% --- Executes on button press in BSARB.
function BSARB_Callback(hObject, eventdata, handles)
% hObject    handle to BSARB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(hObject, 'Value')
    Visibility = 'on';
    set(handles.VO2units,'String',sprintf('VO2\nmL/min/m^2'))
    currentvalue=get(handles.VO2slider,'Value');
    max=get(handles.VO2slider,'Max');
    min=get(handles.VO2slider,'Min');
    BSA=str2double(get(handles.BSAedit,'String'));
    newmax=max/BSA;
    newvalue=currentvalue/BSA;
    set(handles.VO2slider,'Value',newvalue,'Max',newmax,...
        'SliderStep',[1/(newmax-min) 3/(newmax-min)])
    set(handles.VO2edit,'String',sprintf('%0.0f',newvalue))
else
    Visibility = 'off';
    set(handles.VO2units,'String',sprintf('VO2\nmL/min'))
```

```
    currentvalue=get(handles.VO2slider,'Value');
    oldmax=get(handles.VO2slider,'Max');
    newvalue=400*currentvalue/oldmax;
    set(handles.VO2slider,'Max',400,'SliderStep',[1/390 5/390],...
        'Value',newvalue)
    set(handles.VO2edit,'String',sprintf('%0.0f',newvalue))
end
data=gatherdata(handles);
set(handles.BSAedit,'Visible',Visibility)
Update(handles,data)


% Hint: get(hObject,'Value') returns toggle state of BSARB



function BSAedit_Callback(hObject, eventdata, handles)
% hObject    handle to BSAedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(handles.BSARB, 'Value')
    BSAedit=str2double(get(handles.BSAedit,'String'));
    if isnan(BSAedit)==1
        E=errordlg('A number must be entered!','BSA Error');
        waitfor(E);
        set(handles.BSAedit,'String','2.0')
    else
        currentvalue=get(handles.VO2slider,'Value');
        oldmax=get(handles.VO2slider,'Max');
        min=10;
        BSA=str2double(get(handles.BSAedit,'String'));
        newmax=400/BSA;
        newvalue=newmax*currentvalue/oldmax;
        set(handles.VO2slider,'Value',newvalue,'Max',newmax,...
            'SliderStep',[1/(newmax-min) 3/(newmax-min)])
        set(handles.VO2edit,'String',sprintf('%0.0f',newvalue))
    end
data=gatherdata(handles);
end
%collects data from all sliders and calculates it
Update(handles,data);



% --- Executes during object creation, after setting all properties.
function BSAedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BSAedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

set(hObject,'Visible','Off')
```

gatherdata.m

```
function [d]=gatherdata(handles)
% the purpose of this function is to collect all the values from the
% sliders and edit boxes to e used for calculations.

d=[get(handles.HCO3slider,'Value');get(handles.PCO2slider,'Value');...
    get(handles.PO2slider,'Value');get(handles.Tempslider,'Value');...
    get(handles.Hgbslider,'Value');get(handles.VO2slider,'Value');...
    get(handles.Kslider,'Value');get(handles.Qslider,'Value')];

d(9)=3*d(5); %HCT
d(10)=6.1+log10(d(1)/(0.03*d(2))); %pH

O2capacity=1.34*d(5);
% The arterial saturation estimated from the PO2
Sat=(((d(3)^(3)+150*d(3))^(-1)*23400)+1)^(-1);  %SaO2
d(12)=Sat*100;

d(11)=(1-0.0143*d(5))*((0.03*d(2)*10^(d(10)-6.1)-
24.26)+(9.5+1.63*d(5))...
    *(d(10)-7.4))-0.2*d(5)*(1-Sat); %BE

if get(handles.BSARB,'Value')
    BSA=str2double(get(handles.BSAedit,'String'));
    d(13)=((1.34*d(5)*Sat+0.003*d(3))-
((d(6)*BSA)/(d(8)*10)))/(1.34*d(5)+0.003*d(3))*100;
else
  d(13)=((1.34*d(5)*Sat+0.003*d(3))-
(d(6)/(d(8)*10)))/(1.34*d(5)+0.003*d(3))*100; %SVO2
end



% the Calculated oxygen content for arterial and venous blood
%CaO2=((1.34*d(5)*SaO2)+(0.003*d(3)));
%CvO2=((1.34*d(5)*(SVO2/100))+(0.003*d(3)));
% the Venous Saturation


% The data created is an 13x1 arrary with the following data:
%
%   d(1)= HCO3 value
%   d(2)= PCO2 slider value
%   d(3)= PO2 slider value
%   d(4)= Temp slider Value
%   d(5)= Hgb slider Value
%   d(6)= VO2 slider Value
%   d(7)= K slider value
%   d(8)= Q slider value
%   d(9)= HCT calculated value
```

```
%   d(10)= HCO3 calculated value
%   d(11)= BE calculated Value
%   d(12)= SaO2 calculated Value
%   d(13)= SVO2 calculated Value
```

**Perfusion**

**Thesis Approval Form**

**Master of Science in Perfusion – MSP**

**Milwaukee School of Engineering**

This thesis, entitled "Design and Evaluation of a MATLAB® Simulated Blood Gas Monitor," submitted the student Marguerite Wellstein, has been approved by the following committee:

Faculty Advisor: _____ Date: _____
Dr. Ronald Gerrits, Ph.D.

Faculty Member: _____ Date: _____
Kirsten Kallies CCP, LP

Faculty Member: _____ Date: _____
Jonathan Howard CCP, LP