8-2021

Refinement and automation using algorithmic control of BreathForce, a respiratory training system for patients with spinal cord Injuries.

Anna Goestenkors
University of Louisville

REFINEMENT AND AUTOMATION USING ALGORITHMIC CONTROL OF BREATHFORCE, A RESPIRATORY TRAINING SYSTEM FOR PATIENTS WITH SPINAL CORD INJURIES

Anna Goestenkors

Bioengineering B.S., University of Louisville, 2020

A Thesis

Submitted to the Faculty of the

University of Louisville

J.B. Speed School of Engineering

As Partial Fulfillment of the Requirements

For the Professional Degree

MASTER OF ENGINEERING

Department of Bioengineering

May 2021

REFINEMENT AND AUTOMATION USING ALGORITHMIC CONTROL OF BREATHFORCE, A RESPIRATORY TRAINING SYSTEM FOR PATIENTS WITH SPINAL CORD INJURIES

REFINEMENT AND AUTOMATION USING ALGORITHMIC CONTROL OF BREATHFORCE

Submitted by: Anna P Goestenkors
Anna Goestenkors

A Thesis Approved on

28 May 2021
(Date)

by the Following Reading and Examination Committee:

Thomas Roussel, PhD, Thesis Director

Alexander Ovechkin, MD, PhD, Committee Member

Jonathan Kopechek
Jonathan Kopechek, PhD, Committee Member

ACKNOWLEDGMENTS

Throughout the past year, I have received so much support from many people in my life. I would like to thank Dr. Roussel for always being a supportive thesis mentor and helping me along every step of the way. In addition to being a great mentor, he took the time to write numerous letters of recommendation and support me in figuring out the next step in my career. I would like to thank my committee members, Dr. Ovechkin and Dr. Kopechek, for agreeing to be on my committee and support me throughout this project. I would also like to thank Dr. Soucy and Alexa Melvin for being amazing mentors and giving me such great advice. I would also like to thank my family for constantly encouraging me and being my testing volunteers. I would not have been able to do this without them. Finally, I would like to thank my friends for their constant support.

ABSTRACT

Spinal cord injuries (SCI) can lead to impaired respiratory and cardiovascular function and a general decrease in lung compliance. This can complicate breathing as well as impair the ability to sigh, cough, and clear secretions, leading to increased risk of respiratory infections. Respiratory training has been shown to combat these effects. BreathForce is under active development to create a user-centric inspiratory-expiratory device that is an affordable option for at-home training. This study reports on the refinement of valve design and automation incorporated into BreathForce to enhance and enforce clinical practices and processes as part of the respiratory training protocol used with SCI patients.

The system establishes resistance to flow using a custom designed (SolidWorks Flow Sim 2020) proportional valve driven by a 180-degree servo motor (Towerpro MG996R). Computational Fluid Dynamics (CFD) methods were used to evaluate the downstream to upstream pressure differential as each modified valve design was rotated from completely closed to completely open. Boundary conditions were set at the inlet and outlet of the device to imitate the peak volumetric flow rate of a healthy adult male weighing 70 kg (0.167 L/sec). The static pressure at the inlet and outlet of the device as well as the pressure differential were output parameters for each incremental position of the proportional valve. A microprocessor (Feather M0, Adafruit) was used to automate respiratory training. The original system calculated target expiratory and inspiratory

training pressures but required the clinicians to manually set the valve position. An algorithm was developed to automatically set the target valve position for training based on a measurement of the maximum inhalation and exhalation pressures. The pressure drop generated by the user was measured during normal breathing as the servo motor incrementally moved the valve from open to closed. Once the generated pressure was within ninety percent of the target pressure (~ 15% of max capacity), the servo motor was stopped, and that valve position was stored. Healthy volunteers were used to validate system operation. Data was saved to an included SD card and real time clock (Adalogger FeatherWing, Adafruit) to record maximum and minimum pressures generated, as well as session training data at approximately 20 Hz.

The simulation goal was to develop a valve geometry that maintained resistance to flow over the widest range of valve body rotation (0 to approximately 180 degrees). Seventeen design iterations were created and tested via CFD. The algorithm successfully located the optimal valve position for both expiration and inspiration training based on individual users maximum expiratory and inspiratory pressures measured on system startup. Additionally, a simple feedback algorithm was included to adjust the valve position in small increments during training based upon the percentage of target pressure the user was generating. Since pressure drop is related to volumetric flow, if a user generated an artificially high pressure (hyperventilation, coughing) during training, continuous adjustment of the valve position aided users in reaching appropriate target pressures.

Flow simulations set the stage for continued refinement of the custom valve designs which are currently 3D printed. The inherent print resolution limitations of this

manufacturing method are acceptable only for prototyping, and as the product moves towards manufacturability, the valve structure will be injection molded. Each training session begins with a measurement of maximum and minimum pressures, so the target training pressure the user experiences automatically increases as the user gains in their respiratory capacity. Building in automation proved successful in enforcing clinical protocols developed at the Frazier Rehabilitation Institute and refinements will continue as the system moves to the clinic for evaluation with patients under IRB approval.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Spinal cord injury is defined as damage to the spinal cord or the cauda equina, the collection of nerves at the end of the spinal canal. The spinal cord can be directly damaged or be compromised as a result of damage to the vertebrae, ligaments, or discs in the spinal column. Injury may occur for example, from a significant blow to the body or from a wound that directly penetrates and/or lacerates the spinal cord. Additional damage can follow in the days and weeks after the initial injury due to inflammation and swelling of the tissue surrounding the cord [1].

In the United States alone, there are approximately fifty-four new cases of spinal cord injury for every one million people each year [2]. This number does not account for fatalities due to severe spinal cord trauma. While the average age of injury has for many years hovered around twenty-nine years old, the average age is now trending towards forty-three, with males accounting for approximately seventy-eight percent of spinal cord injury cases. Motor vehicle accidents followed by falls are the two largest percentages of spinal cord injuries. Other incidents that cause spinal cord injuries include violence, sports, and medical/surgical incidents. The graphical breakdown of spinal cord injury causes since 2015 can be viewed in Figure 1.1 (next page) [2].

Spinal cord injury can be classified by injury type and level of injury. The two types of spinal cord injury are referred to as *complete and incomplete*. Complete spinal cord injury is defined as a loss of sensory and motor function below the level of injury as a

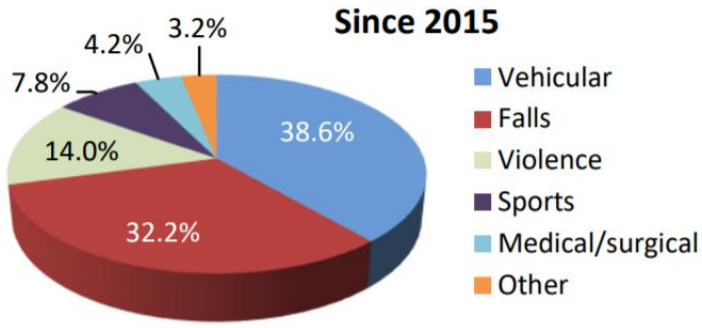


Figure 1.1 - Motor vehicle accidents constitute the largest cause of spinal cord injury since 2015.

whole. Complete injuries results in paraplegia or quadriplegia. Paraplegia refers to paralysis and loss of mobility in the lower limbs and sometimes the lower abdomen due to injury in the thoracic and lumbar segments of the spinal cord. Quadriplegia is paralysis and loss of mobility in all four limbs due to injury to the cervical segment of the spinal cord [3].

Incomplete injury occurs when there is partial damage to the spinal cord that leads to some remaining sensory and/or motor function [3]. The resulting paralysis or loss of voluntary motor movement in this type of injury greatly depends on the level and severity of the injury. The level of injury is based on the section of the spinal cord that is damaged. The four sections of the spinal column include the cervical, thoracic, lumbar, and sacral, and injuries at each level affects different groups of nerves. Depending on the particular nerves that are damaged, an individual may experience varying degrees of severity of paralysis or loss of motor function as well as other symptoms from different organ systems in the body [4].

Since 2010, forty five percent of spinal cord injuries can be categorized as incomplete quadriplegia. Twenty-one percent are categorized as incomplete paraplegia while twenty

percent are categorized as complete paraplegia. Finally, a smaller fourteen percent are categorized as complete quadriplegia [5].

The different levels of paralysis combined with motor function loss result in average annual expenses for individuals that are greatly dependent on the severity of their injury. Table 1.1 displays the average cost of the first year and each subsequent year post injury for different types of injury previously discussed [5].

Table 1.1 - Spinal cord injury has a significant financial impact.

Severity of Injury	Average Cost of the First Year Following Spinal Cord Injury	Average Cost of Each Subsequent Year
High Tetraplegia (C1-C4)	$ 1,064,716	$ 184,891
Low Tetraplegia (C5-C8)	$ 769,351	$ 113,423
Paraplegia	$ 518,904	$ 68,739
Incomplete Motor Function	$ 347,484	$ 42,206

Table 1.1 shows that medical costs for high tetraplegia patients are significantly higher compared to low tetraplegia. This is due to the greater number of complications found in higher level injuries due to a larger collection of nerve groups that are affected. Additionally, the medical costs for both high and low levels of tetraplegia are higher per year compared to paraplegia due to the paralysis and loss of motor function in four limbs as opposed to two.

1.1 Clinical Presentation of Spinal Cord Injury

As previously stated, resulting symptoms of injury greatly depend on the level of injury and severity of injury. General symptoms that can be seen at any level or severity in different combinations include loss of movement, loss of bladder or bowel control, loss or change in sensation, increased reflex activities or spasm, pain caused by damage to the nerve fibers in the spinal cord, and difficulty breathing, coughing, and clearing secretions from the lungs [1].

Elevated risk of severe respiratory complications is primarily associated with cervical injuries. An injury at the fifth cervical vertebrae allows for independent respiration but results in abdominal muscle paralysis. This causes decreased lung volumes and a weak and ineffective cough. The inability to cough to remove secretions can lead to respiratory infections, pneumonia, or even respiratory failure. Second to paralysis and loss of motor functions, respiratory complications are a major concern following spinal cord injury as they remain the most common cause of mortality in spinal cord injury patients. Additionally, the number of respiratory complications experienced by an individual following spinal cord injury greatly influences length of stay and hospital costs [6].

Current rehabilitative therapy aims to restore pulmonary function in spinal cord injury compromised patients. One method includes intermittent positive pressure breathing, which uses a mouthpiece or a facemask that supports inspiration and then manually assists coughing in an attempt to increase lung volume and exhalation flow. A similar method is called in/exsufflation which also increases inspiratory volume followed by assisted coughing. Another method is breath stacking which uses a resuscitation bag connected to a mouthpiece or facemask. The resuscitation bag delivers two or more

breaths before it allows exhalation to occur and increases lung volume. It also aims to improve clearance of secretions [6].

Respiratory therapies have not been proven yet to be clinically effective following spinal cord injury [7]. However, exercise has been proven to increase fitness and respiratory function following spinal cord injury. Strength training can take place in a clinical setting to improve the function of the pectoralis major in order to improve expiratory function and the ability to cough and rid the body of secretions [7]. BreathForce was designed with the goal of creating an inspiratory and expiratory training device that allows for exercise-like training to take place in a home-setting with a simplified user interface. It was also developed with reduced costs in mind to make the device affordable for end users.

1.2 Project Goals

The original goal of BreathForce was to develop a respiratory training device using off the shelf parts combined with a differential pressure instrumentation configuration to evaluate and capture breath cycles generated by the user, and a simple and intuitive control interface to guide users through therapy sessions. While this goal was met, the inclusion of manually adjustable airway restriction valves proved to be troublesome for SCI patients, many of which suffer from limitation of fine motor control. A second-generation device that addressed this major complication of the original device would offer more effective training via automation of valve position and provide a more efficient and consistent experience for end users. With this in mind, the goals of this project were to eliminate the manual valve of the original device by creating a custom flow resistance mechanism, improve the user experience, and evaluate options to

automate the control of the valve position at the start and during a training session. The following specific aims were outlined.

Specific Aim 1: *Design and simulate proportional valve structures that provide geometries that create the widest range of resistance over the largest area of the proportional valve from fully closed to fully open*. The valve should allow for electromechanical adjustment prior to training and continual adjustment of the valve throughout training. Using instantaneous pressure measurements, the simulation data could be used to identify the appropriate valve position that corresponds to the recommended training level chosen by the clinician.

Specific Aim 2: *Develop an algorithm to identify the training session starting valve position that creates an appropriate resistance for the user.* Automation of this step of the protocol will provide a more efficient and consistent experience when using the device. Using maximum expiratory and minimum inspiratory pressures measured at the start of a session, the algorithm will provide a repeatable method to set the initial valve position.

Specific Aim 3: *Automate the device operation during training to ensure users are reaching, but not exceeding, their target inspiratory and expiratory pressure levels.* An algorithm will be developed to continuously monitor the pressure throughout a training session and continually adjust the valve position to account for when users breathe too forcefully, and the measured pressure exceeds the recommended resistance level.

II. BACKGROUND

2.1 Physiology of the Spinal Cord

The spinal cord is a vital part of the central nervous system. It lies within the vertebral column, which is protected by numerous vertebrae, and covered by three membranes, the dura mater, arachnoid, and the pia mater [8]. White and gray matter comprise the cord, creating an H-shaped cross section. The gray matter surrounds the white matter as shown in Figure 2.1 [9].

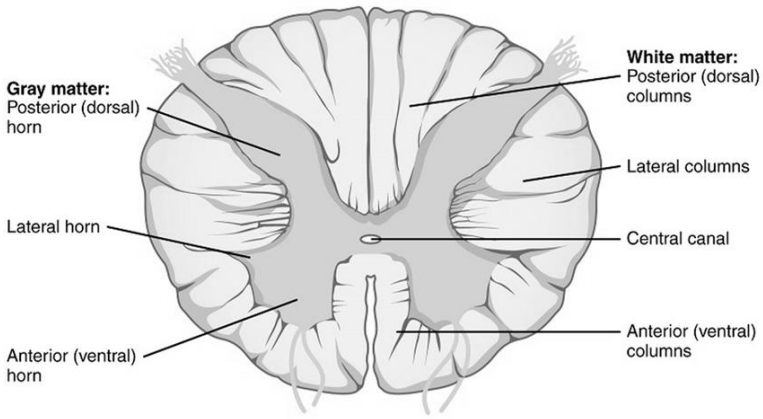


Figure 2.1 - The white matter encompasses the gray matter within the spinal cord.

The gray matter is made up of motor and sensory neurons, interneurons, neuroglia cells, and, for the most part, unmyelinated axons. This tissue makes up the dorsal and ventral horns that are also shown in Figure 2.1. The white matter is made up of

interconnecting fiber tracts that are mostly myelinated sensory and motor axons. The central canal sits at the center of the cord containing cerebrospinal fluid [10].

It is the white matter that carries information between sensory receptors and the higher levels of the central nervous system through the interconnecting fiber tracts. These tracts are categorized as either ascending or descending tracts. The ascending tracts transmit signals from peripheral sensory receptors up to the higher level while the descending tracts transmit signals from the higher levels out to the periphery. The spinal cord can also independently act via reflex arcs, which allow the body to react via this feedback mechanism without input from the brain. In reflex arcs, signals are carried from the sensory receptors to the spinal cord and synapsed on an interneuron. That signal is then carried to a motor neuron which stimulates the effector muscles needed to respond to the sensory input [10].

The spinal column is composed of thirty-three vertebrae that are divided into four sections: cervical, thoracic, lumbar, and sacral [11]. There are thirty-one pairs of spinal nerves coming from the spinal cord that are labeled according to which section they originate in. There are eight cervical nerves that originate from an enlargement of the cord due to their extension into the upper extremities and large neural output and input. There are twelve thoracic nerves and sacral nerves. Between the thoracic and sacral nerves are five lumbar nerves which also originate from an enlargement due to their extension into the lower extremities and large neural output and input [10].

These spinal nerves originate as two roots, ventral and dorsal. The dorsal root transmits signals to the brain while the ventral root transmits signals from the brain. These roots conjoin to form the spinal nerve which eventually forms branches after

exiting the spinal canal. These branches contain motor and sensory fibers [11]. Injuries to the cervical nerves are considered the most severe, as they greatly affect the central nervous system [12]. The third, fourth, and fifth cervical nerves control the function of diaphragm that stretches the rib cage to allow for breathing [13]. When the spinal cord is injured, afferent signals can no longer travel from the brain past the injury. Damage to the cervical nerves can cause weakness or loss of function in the diaphragm and abdominal muscles required for respiration and coughing [12].

2.2 The Physiology of Respiration

The respiratory system consists of the upper respiratory tract, which includes the nose, nasal cavity, sinuses, and pharynx, and the lower respiratory tract, which includes the larynx, trachea, bronchial tree, and lungs. These structures, as shown in Figure 2.2, are involved in the passage of air from the environment into the lungs [14].

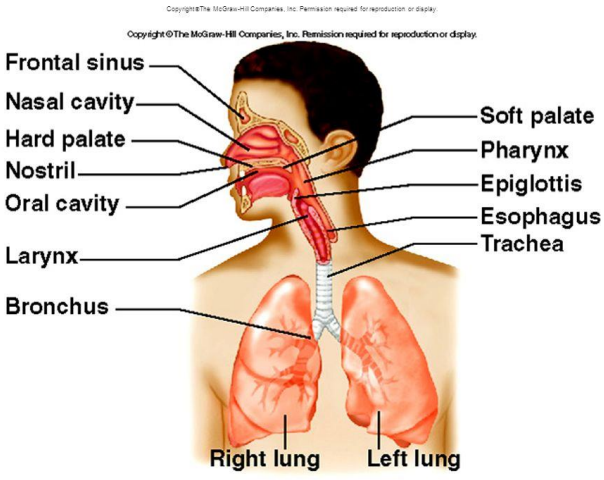


Figure 2.2 - The upper and lower respiratory tracts function to pass air from the environment to the lungs.

Respiration consists of two phases: inspiration and expiration. Inspiration is a product of atmospheric pressure acting as a force to move air into the lungs when a decrease in pressure inside the lungs is created by the diaphragm. When the muscle fibers of the diaphragm are signaled to contract, it moves downward. This movement makes the thoracic cavity bigger and causes the pressure drop within the lungs which then begins to fill with air. External intercostal muscles may also contribute to this pressure drop by contracting to elevate the ribs and sternum, further increasing the size of the thoracic cavity [14].

Expiration is a product of the elastic recoil of lung tissue and abdominal organs and surface tension acting as the forces required to push air out of the respiratory tract. The external intercostal muscles and the diaphragm will relax after inspiration, and elastic tissues within the lungs will cause elastic recoil. This recoil returns the lungs to their initial size and shape. Additionally, surface tension between the alveolar linings within the lungs shrink the alveoli. All of these actions increase the lung pressure above atmospheric pressure which causes the air that traveled into the lungs during inspiration to be forced out. One inhalation and one exhalation are considered a single respiratory cycle [14].

The respiratory cycle can be evaluated and described in terms of air volumes and capacities shown in Figure 2.3 (next page). Spirometry is the measurement technique used to quantify several types of respiratory volumes that are moved in and out of the lungs. *Tidal volume* is the volume of air that enters or leaves the respiratory tract and alveoli during a single respiratory cycle. For an adult, 500 milliliters is considered an average tidal volume during resting inspiration and that same volume exits during a

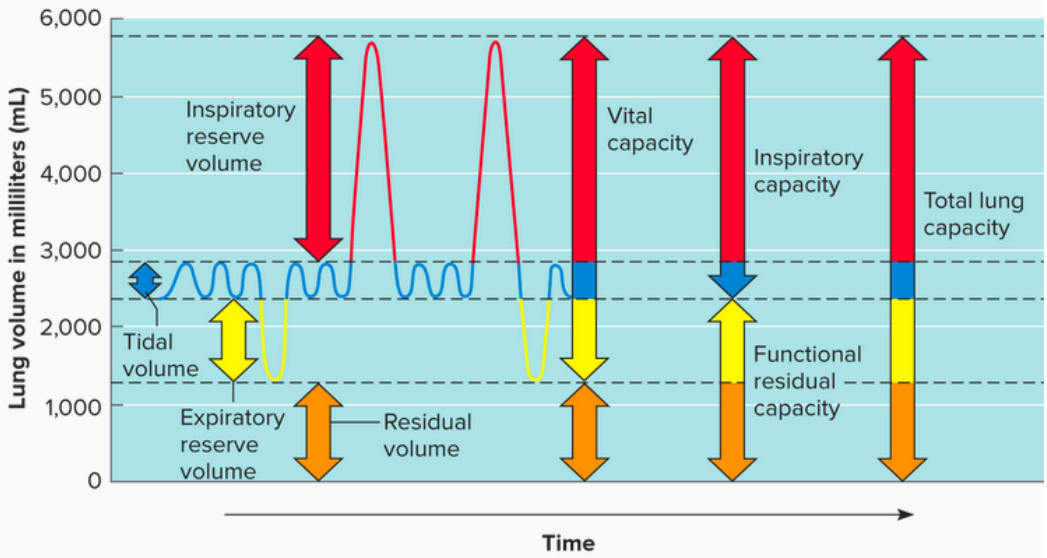


Figure 2.3 - The lung volumes and capacities can give clinicians an idea of someone's lung function.

resting expiration. The *inspiratory reserve volume* is the volume of air that is inhaled in addition to the tidal volume during a forced maximal inspiration and is generally around three thousand milliliters. Contrary to that, the *expiratory reserve volume*, usually around 1,100 milliliters, is the amount of air that can be exhaled in addition to the tidal volume during a maximal forced expiration. *Residual volume*, usually around 1,200 milliliters, is the amount of air that remains in the lungs following a maximal forced inspiration. This is the only volume that cannot be measured with spirometry and must be measured through gas dilution techniques instead [14].

There are four respiratory capacities that can be used to describe the respiratory cycle by combining the respiratory volumes. The *inspiratory capacity* is the sum of the tidal volume and the inspiratory reserve volume. The *functional residual capacity* is the sum of the expiratory reserve volume and the residual volume. The *vital capacity* is the sum of the tidal volume, the inspiratory reserve volume, and the expiratory reserve

volume. Finally, the *total lung capacity* is the sum of the vital capacity and residual volume [14].

2.3 The Mechanics of Respiratory Function Following Spinal Cord Injury

Spinal cord injuries that occur at the third, fourth, or fifth cervical vertebrae and damage any of the corresponding spinal nerves can greatly change the mechanics of the respiratory cycle [16]. Lung compliance is defined as how easily the lungs are able to expand as a result of the pressure changes that occur during the respiratory cycle [14].

Lung compliance is reduced within a month of the initial spinal cord injury and will not change thereafter during the first-year post-injury. This reduction in compliance is believed to be caused by reduced lung volume combined with changes in surface tension that alters the mechanical properties of the lungs. Overall, this reduced compliance is a complicated issue because in spinal cord injury, the compliance of the abdominal compartment of the chest wall is quite high, but the rib cage compartment can become stiff. This stiffness can be a result of muscle spasticity as well as abnormalities in the rib articulations with the spine and sternum. These abnormalities develop as a result of poor inspiratory muscle function that results in a reduction in lung capacity [16].

Quadriplegia causes changes in the respiratory system which in turn causes an individual to exert more energy than is achieved during ventilation. This leads to fatigue of the respiratory muscles. Additionally, weakness or loss of external intercostal muscles function as well as high compliance of the abdominal wall can cause the upper anterior rib cage to move inward during inhalation. This decreases the effectiveness of the diaphragm and reduces the rib cage expansion during inhalation [16].

Overall, the changes in compliance in the abdominal and rib cage compartments as well as respiratory muscle function create abnormalities in the lung volumes and therefore to the vital lung capacities discussed previously. These abnormalities are dependent on the severity and level of injury, body mass index, time since injury, and the development or previous existence of other respiratory conditions. However, on average a decrease in total lung capacity, expiratory reserve volume, and function residual capacity is observed. Additionally, there is commonly an increase in residual volume. Overall, these changes cause less efficient respiration and an increase in exerted energy [17].

The impaired respiratory muscle function and decrease in lung compliance can interfere with the ability to sigh, cough, and clear secretions. This leads to an increase in respiratory infections and causes those infections to be much more persistent when they occur. Respiratory illnesses are a major factor in the mortality rate of spinal cord injury patients, and therefore, respiratory function rehabilitation is a major topic in spinal cord injury research [16].

2.4 Respiratory Function Rehabilitation Methods

Respiratory therapists can attempt to manage or improve the respiratory function of spinal cord injury patients in a number of ways. Respiratory muscle training is often used to improve the muscle strength and endurance. It has been previously shown that able-bodied individuals can train and enhance the respiratory muscles similar to training for the skeletal muscles (i.e. weightlifting). However, these improvements to capacity require a significant amount of training. Spinal cord injury leads to respiratory muscle

weakness that can lead to respiratory muscle fatigue. Training has been shown to enhance respiratory muscle strength and prevent fatigue [18].

Respiratory muscle training commonly centers around inspiration or expiration against an adjustable resistance for a specified amount of time each day for multiple weeks. Studies by Gross et al [19] and Rutchik et al [20] showed improvement in both strength and endurance as well as increases in maximum inspiratory pressures, functional vital capacity, and total lung capacity. Another study by Kogan et al [21] showed similar results as well as thickening of the diaphragm. While these studies clearly show respiratory muscle training is effective, the best (or even a recommended) protocol/method of training has not been established. Additionally, long-term effects of training as well as the effects of abandoning the training method have not been studied or established [18].

One example from a variety of commercially available devices that promote the concept of respiratory muscle training is “The Breather.” This device claims to independently train inspiratory and expiratory muscles in patients suffering from spinal cord injury, COPD, multiple sclerosis, and many other conditions. As shown in Figure 2.4 (next page), it is a small hand-held device with a manually adjustable valve that restricts the flow of air as the user breathes [22].

Another respiratory rehabilitation method is called abdominal binding, which is commonly used in tetraplegia. This method includes using inductance pneumography belts as abdominal support during spontaneous breathing. This support increases the abdominal pressure and forces the diaphragm to remain in contact with the rib cage over

Figure 2.4 - The Breather has a manually adjustable valve to provide resistance during respiratory muscle training.

a larger region. The support also causes the upper rib-cage area to increase during tidal inhalation, showing that the larger region of contact diminished distortion of the respiratory system as whole. Unfortunately, this method does reduce the functional residual capacity, and needs to be studied more to ascertain its effects on gas exchange [18].

Respiratory muscle pacing is a relatively new technique used to restore inspiratory muscle function in tetraplegic subjects following spinal cord injury. Intramuscular electrodes are placed near the motor points of the phrenic nerve in each hemidiaphragm and stimulated to active the diaphragm. This provides full ventilatory support and allows the patient to breathe without the support of the ventilator. Respiratory muscle pacing restores the inspiratory muscle function and provides a more natural breathing experience as the subjects are breathing through their nose. Additionally, placement of epidural electrodes to directly stimulate the spinal cord has been used to restore the ability to cough. The stimulation in that region results in powerful contraction of the abdominal muscles and could result in less occurrences of respiratory infections seen in chronic spinal cord injury cases [18].

After spinal cord injury, many individuals are left without or a significant decrease in the ability to cough. This results in secretions that accumulate in the respiratory tract and cause respiratory infections. The assisted cough technique is used to directly combat this issue, and there are several ways to apply the method. Epidural stimulation, while technically considered muscle pacing, also falls under this method. Quad cough, though often ineffective, involves the subject, in a supine position, expanding their lungs to total capacity and then coughing. A therapist applies forceful pressure to the abdomen in rhythm with the cough. A mechanical insufflation-exsufflation device can also be used [18]. A device is used to gradually inflate the lungs and then quickly changes to negative pressure. This causes the subject to rapidly exhale, which essentially simulates a cough [20]. One example of this type of device is the CoughAssist Mechanical Insufflation-Exsufflation Device produced by J.H. Emerson Co. (Figure 2.5) [24].

Figure 2.5 - The CoughAssist Mechanical Insufflation-Exsufflation Device helps subjects simulate a cough.

Unfortunately, respiratory rehabilitation is quite expensive. For example, while the mechanical insufflation and exsufflation devices are effective, they can cost between $1,500 and $5,000 [25]. In 2009, the cost to implant a spinal cord stimulation system in the United States was estimated to be between $32,882 to $57,896, depending on insurance. The annual maintenance cost at that time was estimated to be between $5,071 and $7,277. The average cost for complications was estimated as $9,649 to $21,390 [26].

While these various techniques have been shown to be effective, respiratory rehabilitation would benefit from a method that allows patients to complete their therapy every day, following a proven and effective protocol, and at a low cost. While "The Breather" provides an inexpensive option at a cost of $47.95 per unit, it contains a manually adjustable valve (difficult for SCI patients to use) with low precision settings that could lead to inconsistent configurations and therefore limit the success of rehabilitation sessions [22]. Additionally, there is no way for supporting respiratory therapists to confirm that patients completed training, nor obtain confirmation that training sessions were completed correctly. While users may set the valve position to apply the correct amount of resistance, confirmation that the appropriate pressure was reached when using the respiratory system to rehabilitate the muscles is not provided. Respiratory muscle pacing requires an expensive and invasive implantation process. It is also subject to complications that require significant costs to maintain these systems. The cost is also highly variable depending upon the insurance that the patient has [26]. Cough assistance machines are expensive as well, which limits the number of patients that can afford to obtain a machine for home use [25]. They may also use the machines incorrectly, limiting their effectiveness.

III. MATERIALS & METHODS

Software developed for the original BreathForce prototype followed a scripted respiratory training protocol developed by researchers at Frazier Rehabilitation Institute while the mechanical prototype components incorporated a manually adjusted resistance mechanism during inspiration and expiration using off the shelf parts. The original training protocol was used as a guide to develop improved software with new features to automate several steps and incorporate a feedback system into the training regimen. The mechanical resistance mechanism was completely abandoned in favor of a 3D printed valve with a servo-driven custom proportional valve body that would provide high resolution adjustments to the flow resistance that users experience when using the device. Validation tests were completed to test the efficacy of the new software that adjusted the valve position for initial setup and during data collection with healthy volunteers.

3.1 Device Hardware

3.1.1 The First-Generation of BreathForce

The first generation of BreathForce was designed using off-the-shelf parts that included an inspiratory valve and an expiratory valve combined with a tee connector that contained two one-way valves. A differential pressure sensor was included to measure the pressure inside the device with respect to atmospheric pressure. The inspiratory and expiratory threshold valves contained threshold settings that could be manually adjusted by the clinicians or users to adjust the resistance to flow during the training sessions [27].

The two one-way valves on opposite sides of the tee connector provided independent resistance levels according to the threshold settings based on the measured maximum inspiratory and expiratory capacities. As the user inhales, the air flows through the inspiratory one-way check valve into the tee connector while the expiratory resistance device blocks airflow through that portion. During exhalation the opposite effects occur, where the air flows through the tee connector and the one-way valve of the expiratory one-way check valve while the one-way valve in the inspiratory muscle trainer device blocks airflow. The first-generation device and the airflow paths are shown in Figure 3.1 [27].

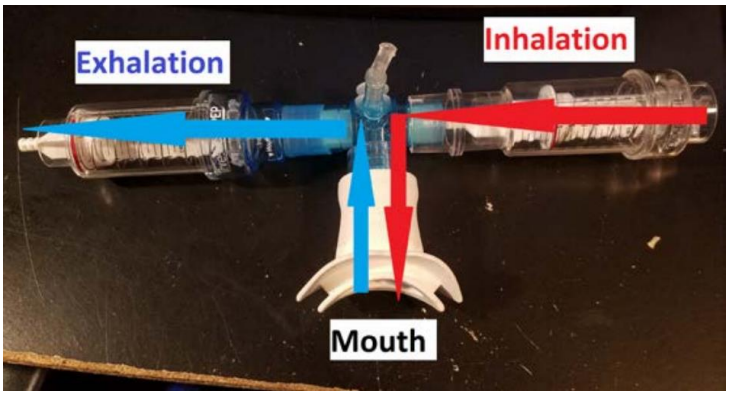


Figure 3.1 – The first-generation of BreathForce used off-the-shelf parts to create a respiratory training device that provided resistance for inspiration and expiration.

3.1.2 The Proportional Valve vs. Threshold Valve

By definition a proportional control valve changes the fluid flow rate through changes in size of the flow passage via a restriction system [28]. The new goal was to include one valve that provided resistance for both inspiration and expiration in the second-generation

device. Threshold valves that were used in the first-generation device were not capable of this as they open when the pressure reaches a certain level (threshold, or cracking pressure), and therefore only work for one direction of airflow. A custom-designed proportional valve would provide resistance in both directions and require only one adjustment mechanism. The geometry of the custom-designed proportional valve is unique, and while the airflow may remain laminar, there is no simple technique to quantify changes in area of the flow restriction path. General fluid transport equations do not apply to these custom designs; therefore simulations are appropriate and required in order to evaluate the pressure differential of individual designs prior to fabrication.

3.1.3 The Second-Generation of BreathForce

As shown in Figure 3.2, the first iteration of a bi-directional respiratory training device was developed in SolidWorks (Dassault Systèmes, Waltham, MA) in order to provide users with a device that contained an integrated airway restriction mechanism for both inspiration and expiration. This prototype features a one-inch diameter inlet and outlet with a custom designed proportional control valve that incrementally opens and closes when rotated by a servo motor. Two pressure measurement ports (1 mm diameter) extrude form the flow volume upstream and downstream from the adjustable valve body to allow for the measurement of differential pressure measurement using a pressure sensor ((+/- 5 PSI, NSCDRRN005PDUNV, Honeywell, Charlotte, NC).

To provide a mechanism for electronic control of valve position, a platform and mounting features for the servo motor (Towerpro MG996R) were included as well as a custom gear attachment for the servo horn (24T). A section at the center of the device was removed to expose gear teeth on the internal movable valve that would interface with

the gear driven by the servo. The ratio of the servo gear to the movable valve gear was designed to be 1:1.67, providing approximately 240 degrees of internal rotation to 180 degrees of rotation of the servo motor. Turning the servo motor incrementally open and closes the proportional valve.

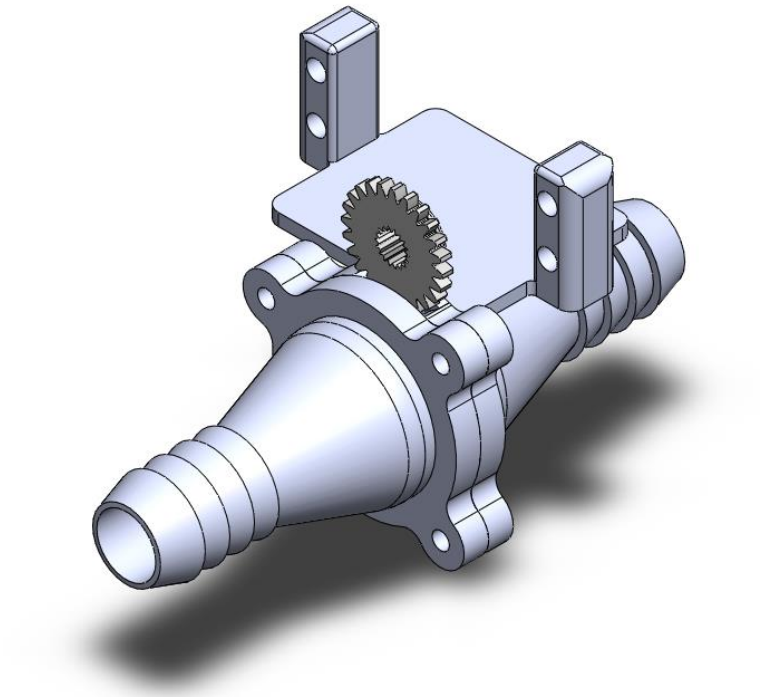


Figure 3.2 - The 2nd generation design of BreathForce featuring an inlet, outlet, servo motor platform, and driving gear for internal proportional valve.

When the general layout of the second-generation prototype was finalized, focus was turned to the design of the geometric features of the adjustable proportional control valve. A planar valve body was chosen to directly interface with the direct drive of the servo gear and to simplify the fabrication process during prototyping. The design should maintain resistance to flow over the widest range of valve body rotation (zero to one

hundred and eighty degrees) to allow the clinicians ample opportunity to find a valve position that produces the correct amount of resistance for each user.

The first design produced in SolidWorks featured a proportional valve design produced with the end of the inlet acting as the fixed portion and the disc acting as the free portion as seen in Figure 3.3 and Figure 3.4, respectively.

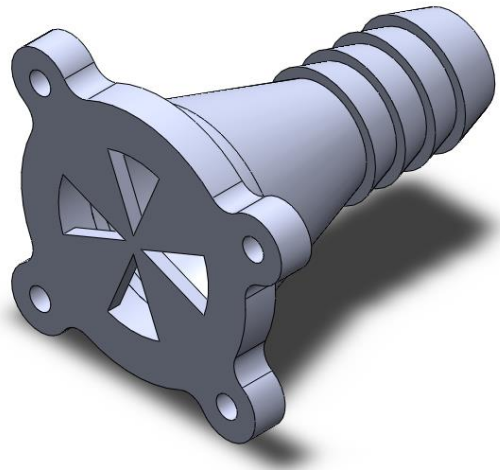


Figure 3.3- The end of the inlet acts as the fixed portion of the proportional valve.

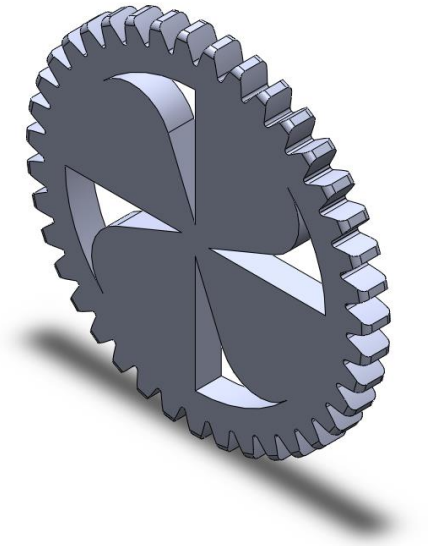


Figure 3.4- The disc acts as the rotating (free) valve body of the proportional valve.

As the free portion is turned and the cut-out sections align, air more easily passes through the device leading to little to no pressure differential between the inlet and outlet of the device and less resistance. When the cut-outs do not align, a smaller volume of air can pass through the device leading to a greater pressure differential between the inlet and outlet and greater resistance. The SolidWorks drawings of the original design components and assembly can be seen in Appendix I.

Figure 3.5 shows the pressure differential response of the first proportional valve design as well as an idealized response. The idealized response includes a linearized decline in pressure as the proportional valve is opened as it provides resistance to flow over a wide range of valve body rotation. The area under the curve of the first bi-directional proportional control valve design is approximately 145 while it is approximately 1025 for the idealized response which indicates there is room for

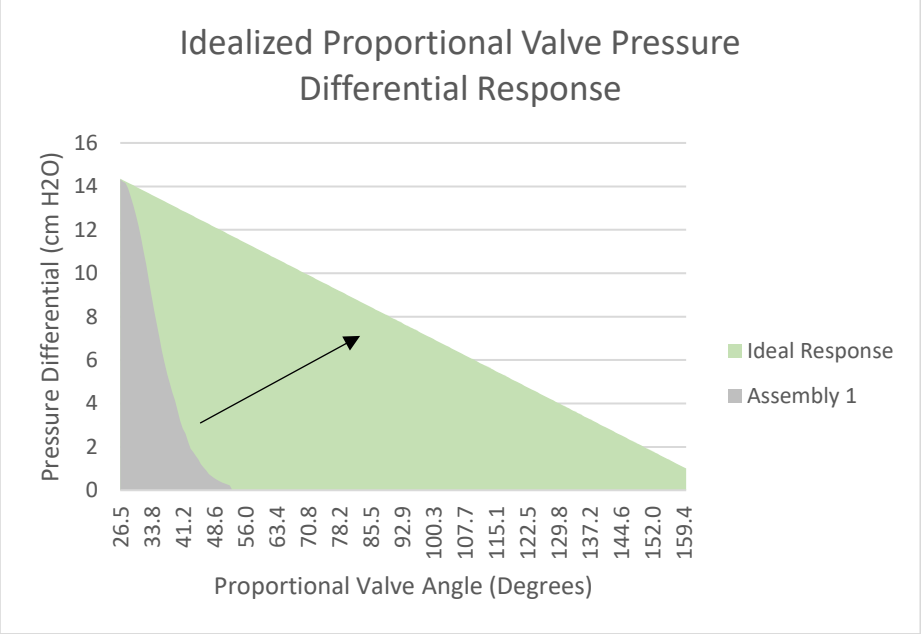


Figure 3.5– The idealized response that maintains resistance to flow while experiencing a linearized decline is shown in green.

improvement from the original design. Visualization of simulation results will demonstrate increases in both the range of valve angle of rotation and the area of similar curves based on iterations of the flow restriction geometry.

3.1.4 Pressure Differential Analysis via CFD

SolidWorks Flow Simulation 2020 was used to study changes in the pressure differential as each valve body design was incrementally moved from completely closed to completely open. A simplified three-dimensional model of the overall design prototype was used to simplify the geometry required to initiate flow simulations (Figure 3.6). The platform for the servo motor was removed in addition to the gear attachment and gear teeth on the free portion of the proportional valve.

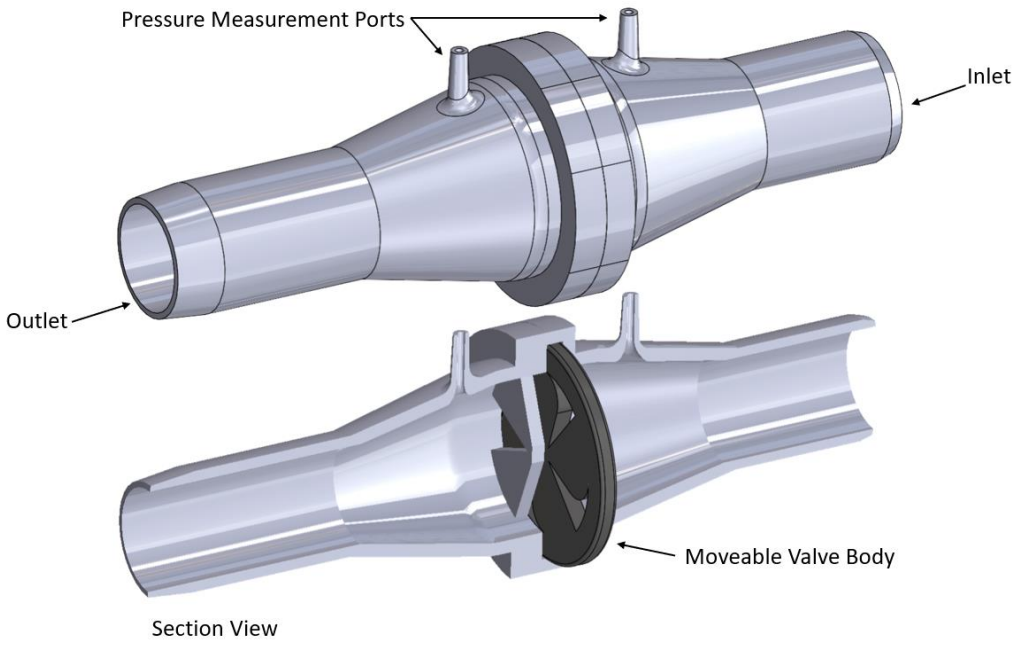


Figure 3.6 – Simplified version of the valve including a section view showing the movable valve body.

In order to complete a parametric study, an angle mate that establishes a relationship between the original and target rotation of the valve body was included. This parameter iteratively sets the position of the free valve body in relation to the position of the fixed portion on the inlet. This mate was then made into a global variable for use in the simulation where the software could rotate the valve angle between iterations. Rotation of the servo position adjusts the primary gear attachment which in turn rotates the free portion of the valve. The gear ratio is such that one degree of movement on the servo motor is equivalent to 0.615385 degrees of movement of the free portion of the valve. Multiples of 0.615385 degrees were used to incrementally change the valve position. An example of a proportional valve design moving from completely closed to open in multiples of ten is shown in Figure 3.7.

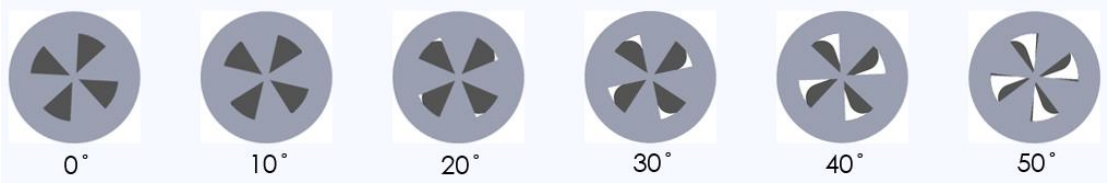


Figure 3.7 – A proportional valve design in SOLIDWORKS is moved from completely closed to completely open using an angle mate.

Additionally, the software requires that the assembly be completely sealed in order to run a flow simulation. Lids were created to seal the inlet, outlet, two pressure ports, and the opening for the servo motor gear attachment.

Boundary conditions were set at the inlet and outlet of the device. When participants use the device, they will be breathing into the inlet, so a volumetric flow rate boundary

condition was applied. In order to simulate and visualize the pressure differential created with each valve position, a peak volumetric flow rate was used.

An adult male weighing 70 kg expires a volume of approximately 500 milliliters following resting inspiration for approximately two to three seconds [29]. Using these parameters, a volumetric flow rate during exhalation of an average adult male can be calculated using the following unit conversions.

$$Volumetric\ Flow\ Rate = \frac{500\ mL}{1\ expiration} \times \frac{1\ expiration}{3\ seconds} \times \frac{1\ L}{1000\ mL} = 0.167\ L/s$$

The value of three seconds was used as the estimate for the time to complete a full exhalation. The resulting flow rate was set as a boundary condition at the inlet. Atmospheric pressure was set as a boundary condition at the outlet.

The input parameter of the parametric study was the global variable of the angle mate between the fixed and free portion of the proportional valve. Multiples of 0.615385 degrees were added to ascertain the pressure differential at each valve position from completely closed to completely open. The angle values for the completely closed and open positions differed for each design due to different design features in each. The completely closed position was defined as the position that generated the largest pressure differential value while the completely open position was defined as the position that generated the smallest pressure differential value. The output parameters of each study were the static pressure at the inlet, outlet, and the pressure ports as well as the calculated differential pressure between the inlet and outlet and the calculated differential pressure between the two ports. The automatically calculated mesh with a refinement level of seven (about 100,000 cells) was used to generate pressure differential results for each of

these output parameters at each valve position. Overall, seventeen proportional valve designs were simulated, and the fixed and free portion of each design is included in Appendix II.

3.1.5 Printed Circuit Board and Components

The circuit board designed to support the second-generation device was inherited for this project (Figure 3.8). The circuit included a 2.5 mm power input jack and on-off power switch for a 5-volt, 2.5 Amp DC power supply. An embedded microcontroller (Feather M0 Bluefruit, Adafruit Industries, New York, NY) was used to orchestrate the respiratory training protocol. A real-time clock with flash storage daughterboard (Adalogger FeatherWing - RTC + SD, Adafruit Industries, New York, NY) was included to save data with time stamps from the real time clock. A manual reset button was

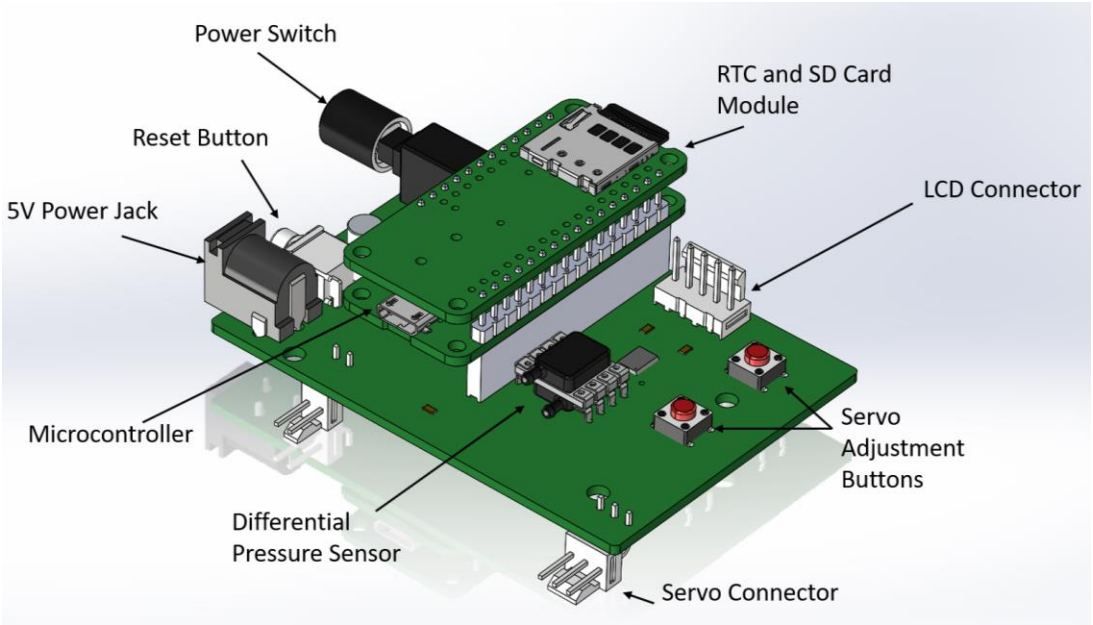


Figure 3.8 - The 3D rendering of the printed circuit board shows the various components.

incorporated into the design for programming, and two manual switches were included on the circuit board to control servo motor movement.

A differential pressure sensor (+/- 5 PSI, NSCDRRN005PDUNV Honeywell, Charlotte, NC) was incorporated into the circuit in order to measure the pressure difference between the inlet and the outlet ports of the breathing apparatus. Additionally, a 3.2-inch display (uLCD-32PTU-AR, 4D Systems, Minchinbury NSW, Australia) was included to provide input control via resistive touch input and visual feedback for the user, which included both instructions for operating the device as well as feedback during implementation of the protocol. The layout and 2D rendering of the printed circuit board and components can be viewed in Appendix III.

3.2 Device – Software

The original prototype software was developed on the Arduino platform (Arduino MEGA, Arduino.cc), and the flexibility of a software driven mechanism that included the custom protocol developed at Frazier Rehab for respiratory training proved to be extremely successful. Based on feedback from users and clinicians, changes to refine the flow of the training enabled by the device could be acted on rapidly with simple changes to the code. The most recent version of the original prototype software was used as a base for this project and was refactored to include both manual and automated control of the servo position, as well as enhancement to the respiratory protocol to take advantage of the system-controlled proportional valve.

3.2.1 Software Criteria

The primary software development goal was to include code that allows the system to set the position of the proportional valve for training based on the pressures measured by the pressure sensor. This is a significant advancement beyond the original manually adjusted valve position of the original prototype. In the original software, the training pressures were calculated, and the clinician or user manually adjusted the valve position. The updated software was also required to include a method to continuously adjust the valve body angle during the training portion of the protocol. This adjustment would be used to actively increase or decrease the valve resistance created by the proportional valve to assist the user in maintaining the training pressure values. Additionally, the software for the touchscreen display was to be replicated and updated to include these additions to the protocol. The existing software prior to any updates can be seen in Appendix IV.

3.2.2 Updating the Touchscreen Display

The 4D Workshop Integrated Development Environment (IDE) and Arduino IDE were used to program the touchscreen display and microcontroller, respectively. The 4D Workshop (4DW) was used to create multiple forms that act as a dynamic user interface on the touchscreen and to program the display to report messages in response to certain user inputs to the microcontroller. These inputs were used to prompt the user regarding the various functions of the device. Additionally, data was programmed to be sent from measurements performed in the microcontroller code to the touchscreen for display.

Forms from the first iteration of the device were replicated. Insignificant changes such as making custom software buttons that were larger and easier to read were made to

some of the existing forms. New form pages were created to facilitate the location of the proportional valve position as well as test the pressure differential created at different proportional valve positions.

3.2.3 Updating the Microcontroller Code

The Arduino IDE was used to develop the previous code, and it was also used to add new functions in order to provide a valve test program, to locate the position of the proportional valve, and to provide continuous adjustment of the valve position during training.

The test program was created with a function that allowed for the servo motor to be manually turned clockwise or counterclockwise (opening and closing the valve) in increments of one degree. The user could then measure the pressure differential generated by the proportional valve in cm H_2O at a particular valve position. The pressure sensor reports values in PSI, but code was included to convert cm H2O which are standard for respiratory measurements. This conversion can be viewed in Appendix V.

The process for recording the maximum expiration and inspiration pressures would be the same as the previous iteration of software. It would use a moving window average to find the values. The target pressures would then be calculated by multiplying those maximum pressures by the training percentage chosen by the clinician or user.

The new function to locate the proportional valve position for training measured the pressure differential in cm H2O while the servo motor moved the valve from an open position to a closed position in increments of one degree. As the user is asked to breathe in and out normally, the proportional valve closes, which increased the resistance to flow

and increased the measured pressure differential. Once the user was within ninety percent of the calculated exhalation training pressure, the servo motor stopped, and the servo position was saved. The same process was repeated for the calculated inhalation training pressure. These two valve rotation positions would serve as the starting point for training.

During the training portion of the protocol, an algorithm was included to continuously adjust the servo motor to the appropriate proportional valve positions during inspiration and expiration. When the pressure differential was positive and within one hundred and ten percent of the exhalation training pressure, the servo motor set the proportional valve to the position located for the exhalation training pressure. When the pressure differential was negative and within one hundred and ten percent of the inhalation training pressure, the servo motor set the proportional valve to the position located for the inhalation pressure. If the pressure differential exceeded one hundred and ten percent of either training pressures, the servo motor opened the proportional valve ten degrees to decrease the resistance and the pressure differential. This addition to the code was included to ensure that the user does not experience too much resistance (higher than prescribed), which leads to early fatigue and low compliance.

3.2.4 Experimental Testing with Healthy Volunteers

To test the additions to the existing code, six volunteers, three healthy females and three heathy males completed the entire protocol with the new additions. Each completed the protocol three times for two minutes, each using a different training percentage. The chosen training percentages were ten, fifteen, and twenty percent. The results were saved to the SD card for later evaluation by clinicians.

IV. RESULTS/DISCUSSION

4.1 Proportional Valve Computational Fluid Dynamics Results

The flow simulations produced the pressure differential between the inlet and the outlet of the breathing apparatus at each input angle. The results shown in this section represent significant developments made during this study. Results for all seventeen assemblies can be seen in Appendix VI. The original design or first assembly, shown in Figure 4.1, was tested first and produced the data seen in Figure 4.2 (next page). While the results showed a relatively gradual decline in pressure, that gradual decline only spanned approximately forty degrees. The servo motor allows for up to one hundred and eighty degrees of movement. It was important to find a design that allowed for adjustments over a larger portion of that available space, so the valve could be adjusted to

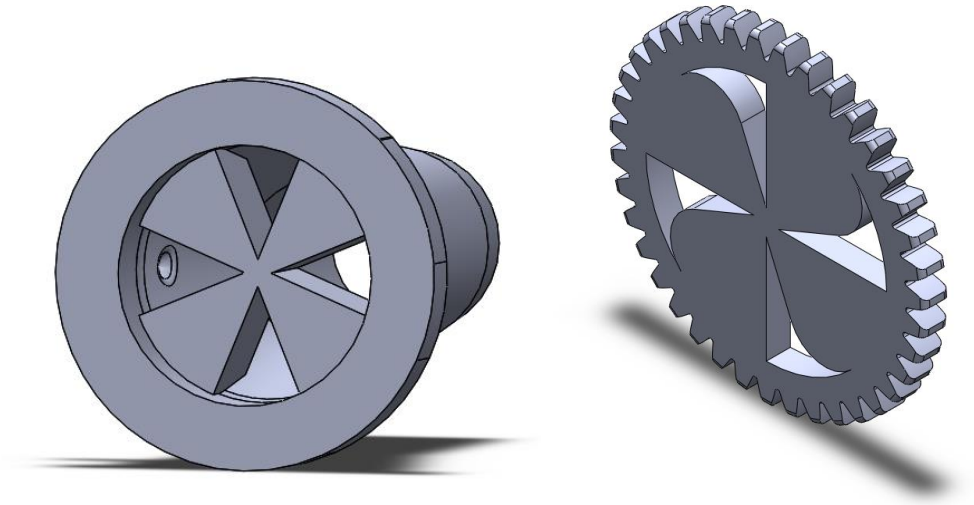


Figure 4.1 – The original design featured four cut-outs on both portions.

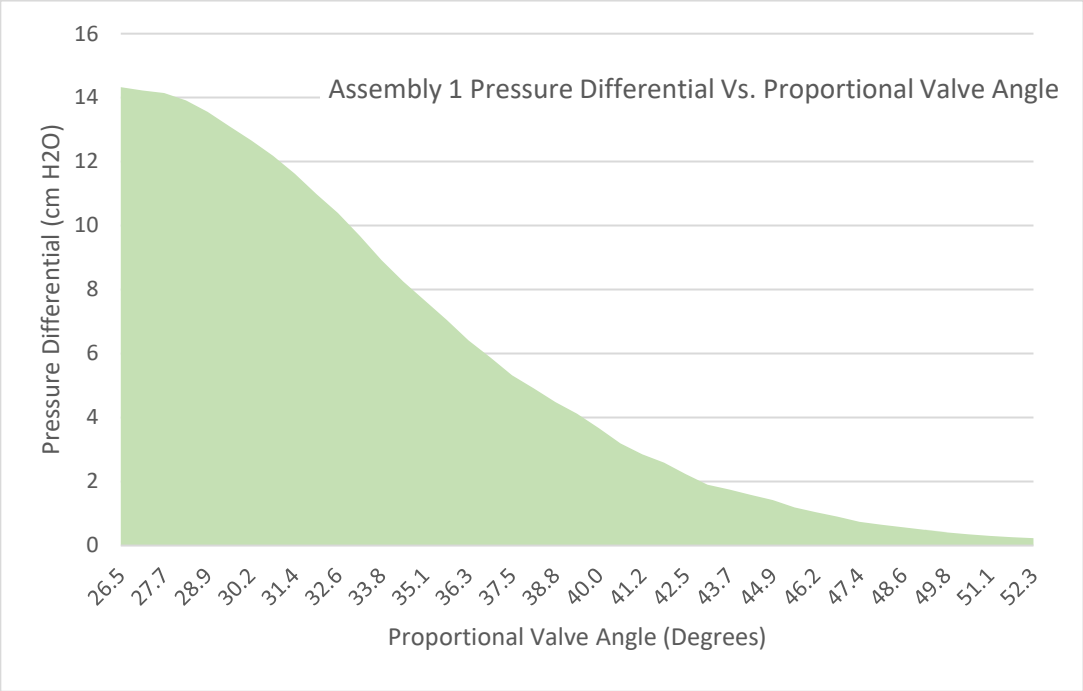


Figure 4.2 – The original design resulted in a low maximum pressure differential and resistance to flow over only forty degrees.

the most efficient location for each individual participant. Additionally, the pressure differential generated, even completely closed (the most resistive position) was relatively low at 14.3 cm H_2O.

In order to increase resistance to air flow to raise the pressure differential, a grate design was created for Assembly Three. Additionally, to attempt to provide resistance over a larger area of the valve, the grate was designed to cover one hundred and eighty degrees of the proportional valve as shown in Figure 4.3 (next page). A cutout matching the size of the grate was made on the fixed portion of the proportional valve. A flow simulation was completed and produced the results seen in Figure 4.4 (next page). While the grate design resulted in an increase in the pressure differential, with a maximum value of 30.9 cm H_2O, the design only provided resistance to flow over thirty degrees.

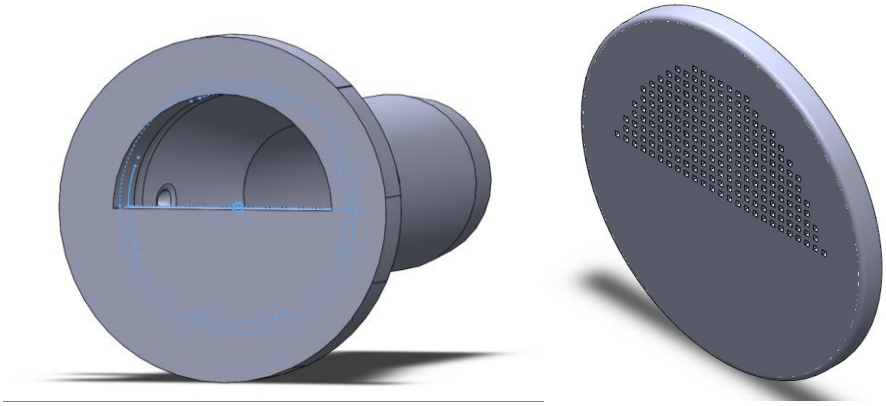


Figure 4.3 – The third assembly featured a grate design on the free portion to increase resistance to flow.

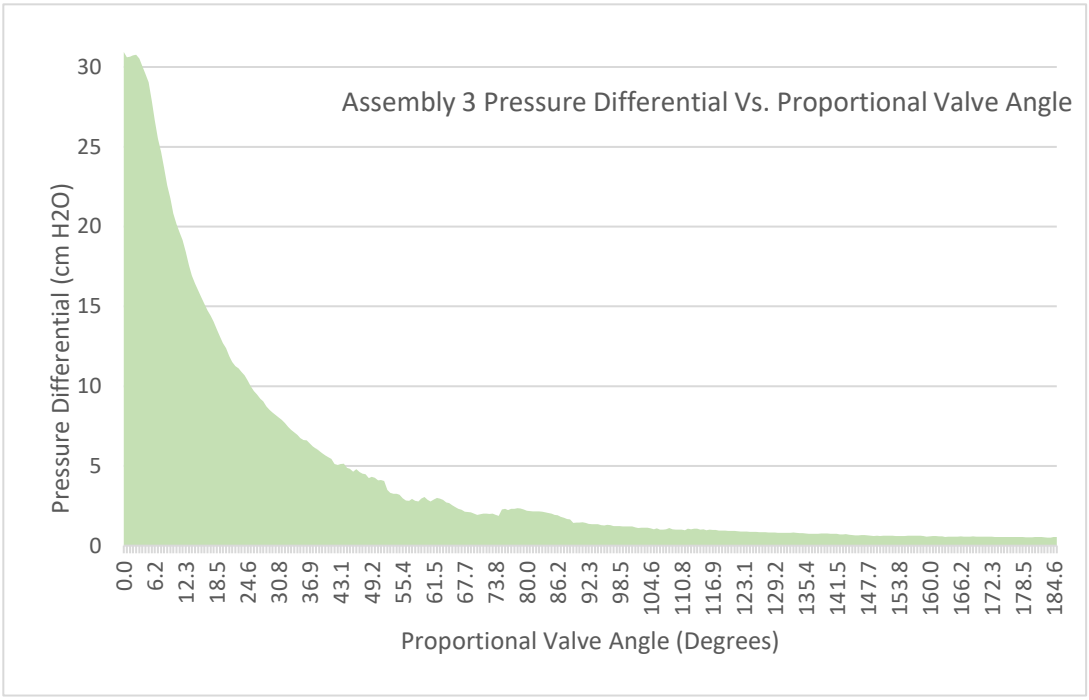


Figure 4.4 – The third assembly resulted in a high maximum pressure differential, but only resisted airflow over 30 degrees.

Continuing with the grate concept, the variable pattern feature in SolidWorks was used to create a grate in a semi-circular pattern on the free portion of the valve. As the free portion was rotated, the size of the cutouts within the grate pattern changed. In the first iteration of this design, Assembly Five, as the grate pattern lined up with the cutout

on the fixed portion, the holes became larger. In the second iteration, Assembly Six (Figure 4.5), the holes became smaller. Of the two, Assembly Six created a larger maximum pressure differential as shown in Figure 4.6. However, the resistance to flow only spanned approximately twenty degrees, limiting the adjustments available for participants during the training regimen.

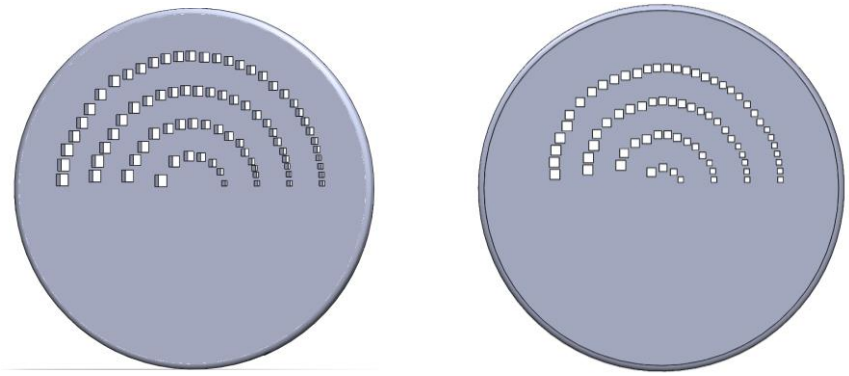


Figure 4.5 – Assembly Five and Six featured the same patterns, but in a reverse order to ascertain which produced the greatest resistance to flow over the largest area of the proportional valve.

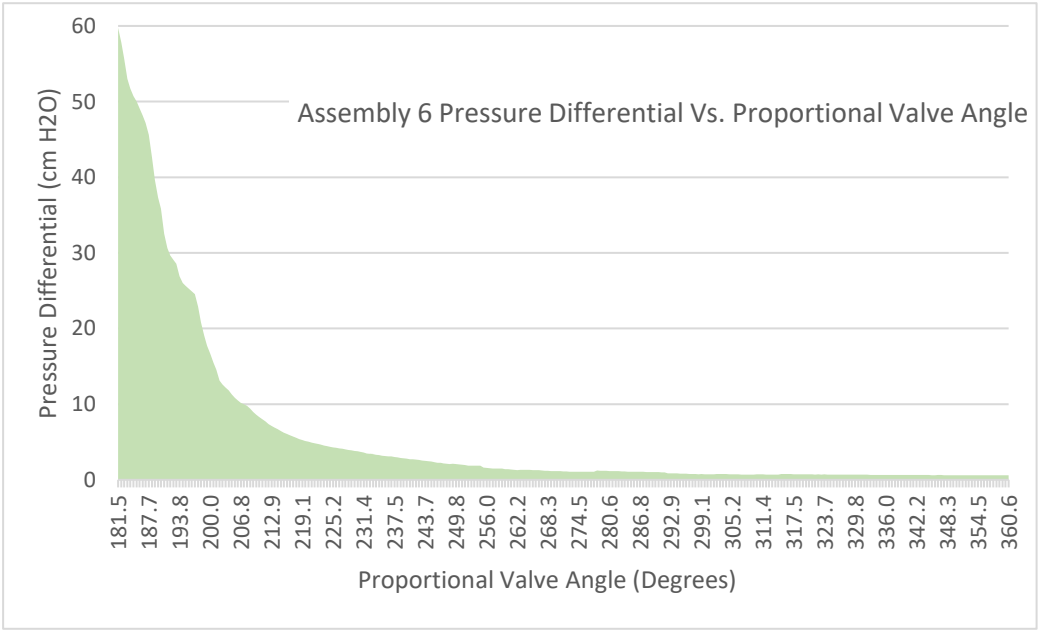


Figure 4.6 – Assembly Six resulted in a maximum pressure differential of 59.7 cm H2O while Assembly Five resulted in a maximum pressure differential of 38.0 cm H2O.

In order to maintain the increased pressure differential but create a more gradual decline in pressure, different patterns were tested on the fixed portion of the proportional valve. Additionally, the cut-outs of the grate design were rounded. Flow simulations were also used to test whether a change in the thickness of the free portion of the valve would have any effect on the generated pressure differential. Assembly Eight (Figure 4.7) was created with a matching grate pattern on both the fixed and free portions of the proportional valve. Additionally, as the free portion turned and more openings lined up, the thickness of the free portion continuously increased.

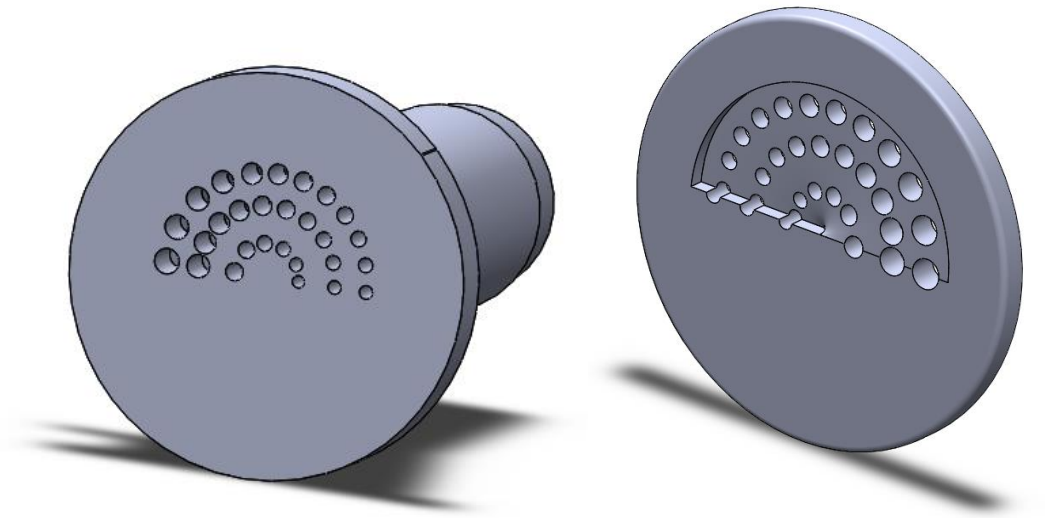


Figure 4.7 – Assembly Eight featured matching grate designs on both portions with a change in thickness on the free portion.

Assembly Nine (Figure 4.8, next page) was created for comparison by adjusting only the thickness of the free portion. To test if increasing the thickness created a difference, the thickness over the grate in Assembly Nine was set equal to the thinnest portion of Assembly Eight.

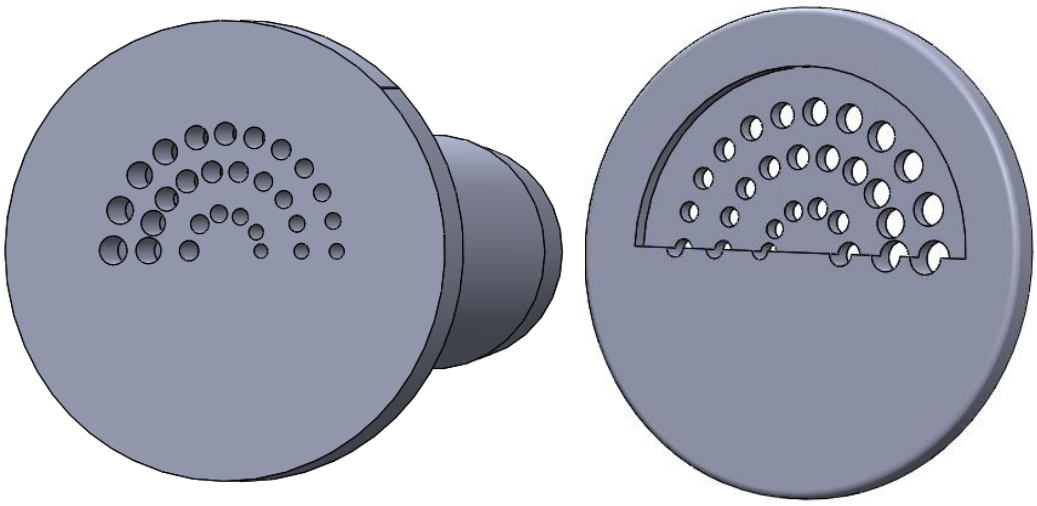


Figure 4.8 – Assembly Nine was the same design as Assembly Eight with the thickness over the grate equivalent to the thinnest portion of Assembly Nine.

Overall, the increase in thickness over the grate did create a resistance to flow over a larger portion of the valve rotation. Assembly Eight results (Figure 4.9, next page) show that the matching grate pattern with increasing thickness generated a maximum pressure differential of 7.34 cm H_2O and a gradual decline in pressure that spanned approximately sixty degree. Assembly Nine results (Figure 4.10, next page) show that the matching grate pattern without increasing thickness generated a maximum pressure differential of 6.69 cm H_2O with a steeper decline than Assembly Eight.

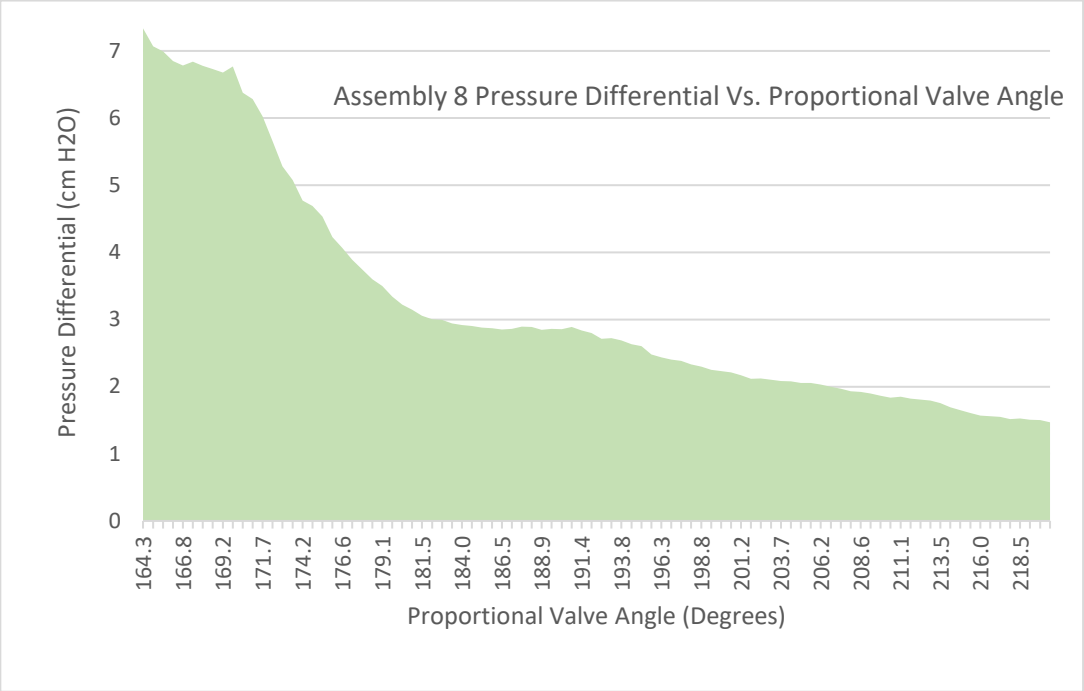


Figure 4.9 – Assembly Eight featured an increasing thickness over the grate and resistance to flow over fifty-five degrees.

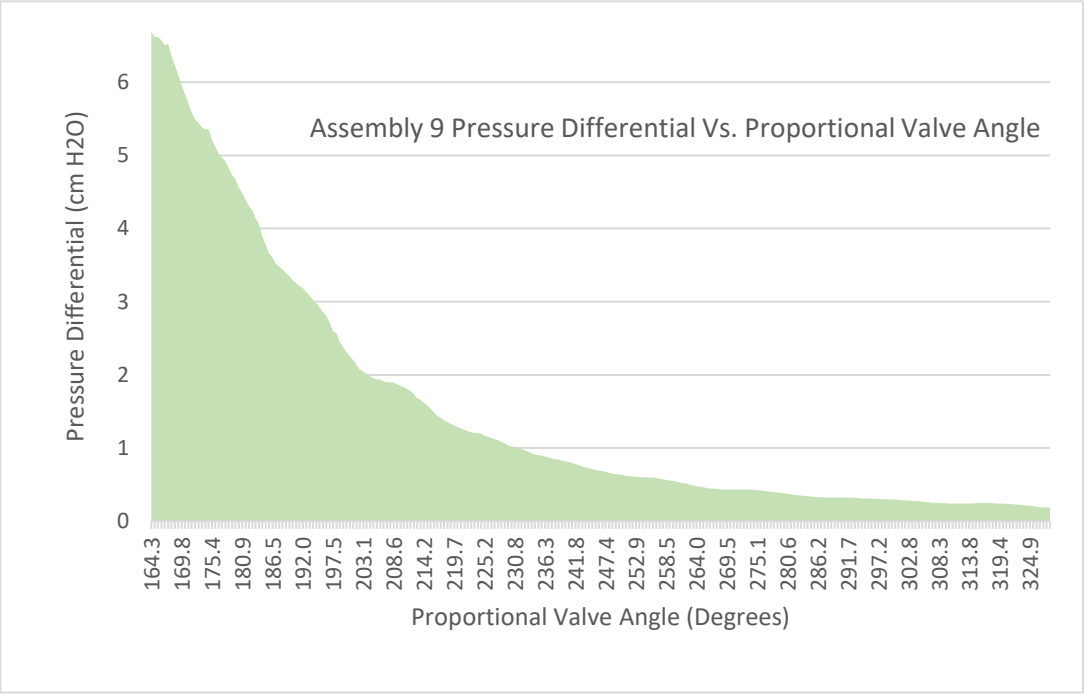


Figure 4.10 – Assembly Nine show resistance to flow only over forty degrees.

Assembly Eleven, Twelve, and Thirteen feature matching square grate designs with the same increasing thickness as the proportional valve opens. The sizes of the cut-outs were altered to determine the effects different sizes and increasingly large cut-outs had on the generated pressure differential. Assembly Eleven and Twelve both featured cut-outs of increasing size. However, the cut-outs in Assembly Eleven were 0.10 millimeters larger than the cut-outs in Assembly Twelve. Assembly Thirteen featured cut-outs all 0.40 millimeters by 0.40 millimeters.

The results for all three assemblies featured gradual declines in the pressure curves with discontinuities appearing in the trend in the data due to the cut-outs aligning and misaligning as the proportional valve opened. Assembly Thirteen (Figure 4.11) featured the highest maximum pressure differential of 60.7 cm H_2O (Figure 4.12, next page).

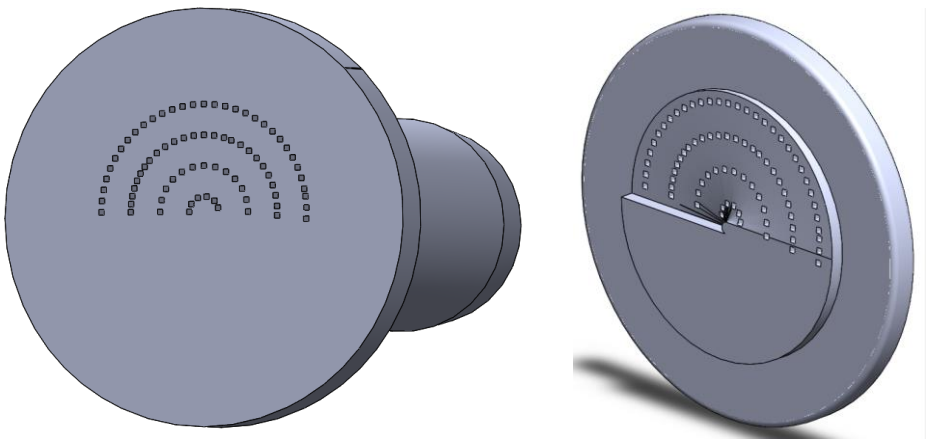


Figure 4.11 – Assembly Thirteen featured matching grate designs of .40 by .40-millimeter square cut-outs.

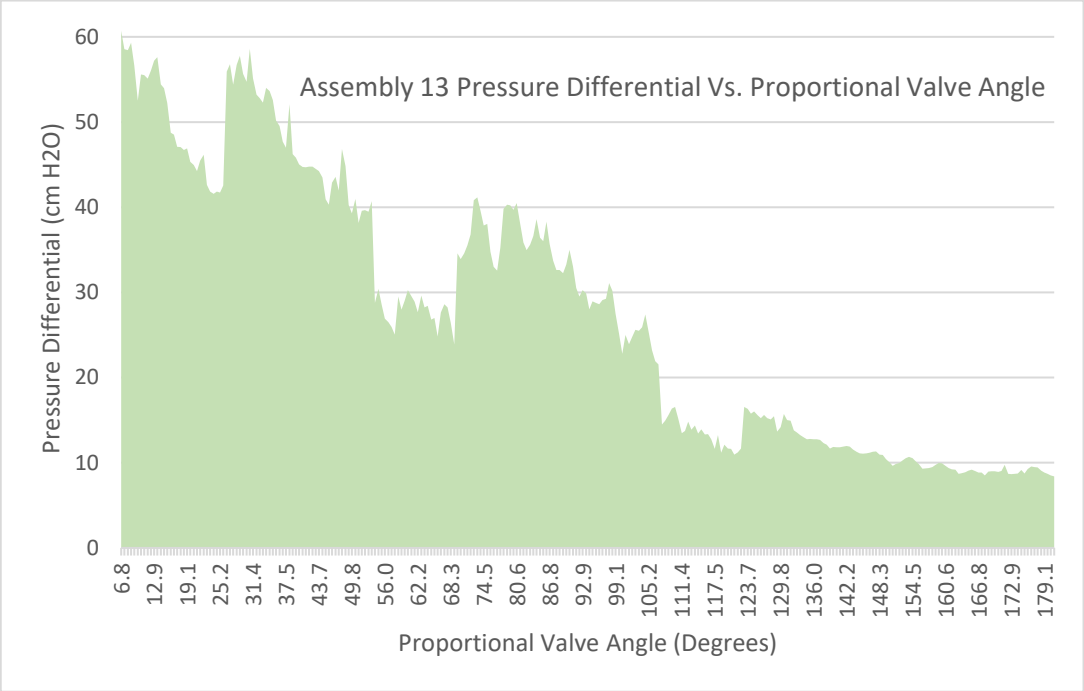


Figure 4.12 – Assembly Thirteen featured high resistance to flow over 180 degrees.

While the grate design seemed promising in terms of a high-pressure differential and resistance to flow as the proportional valve is opened, the ability to 3D print the design was an issue. The available 3D printer (3D40, Dremel, Racine, WI**)** was capable of reliably creating a minimum hole size of around 1.1 mm. A design was created with matching grate designs on both portions of the proportional valve. Assembly Fourteen (Figure 4.13, next page) featured the increasing thickness on the free portion and increasing size in cut-outs as previously tested. However, these cut-outs were 1.10 millimeters to 1.90 millimeters. Assembly Eleven and Twelve featured cut-outs between 0.40 and 0.90 millimeters.

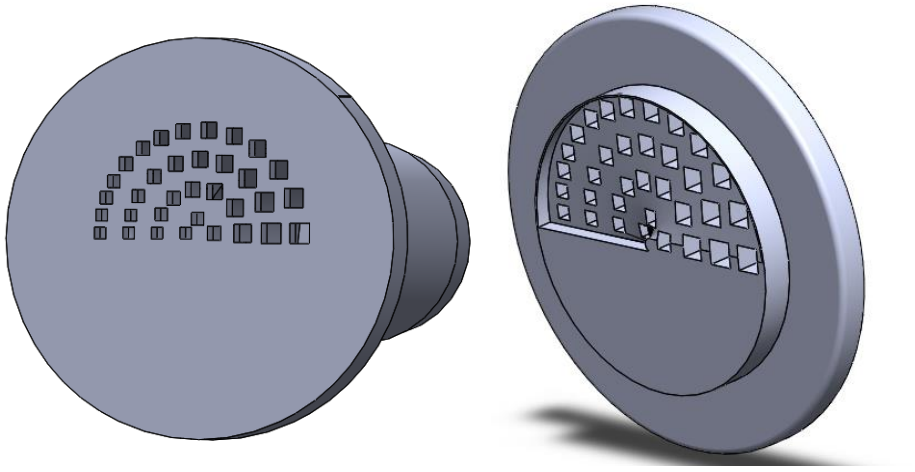


Figure 4.13 - Assembly Fourteen featured matching grate designs with larger cut-outs of increasing size and increasing thickness over the grate.

Additionally, since previous studies showed cut-outs of the same size generated a higher maximum pressure differential, Assembly Fifteen (Figure 4.14) was created featuring cut-outs that were all 1.10 millimeters.

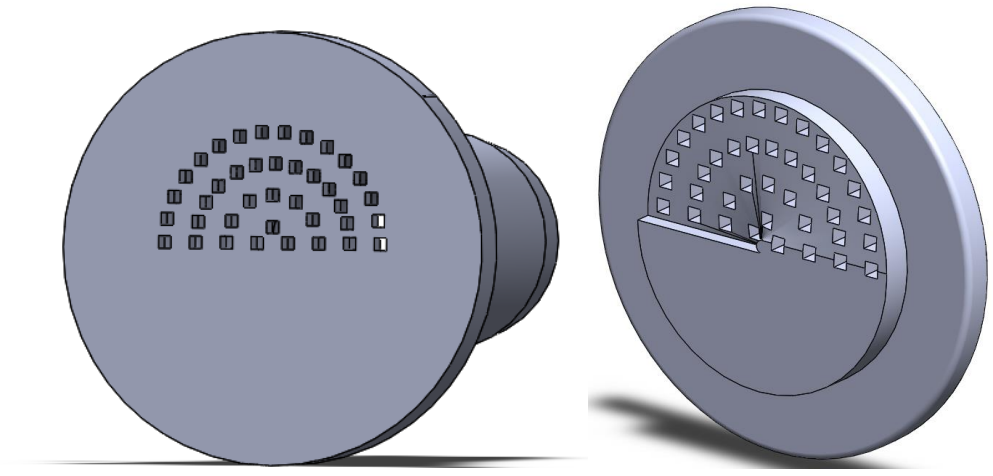


Figure 4.14 – Assembly Fifteen featured matching grate designs with larger cut-outs of all the same size and increasing thickness over the grate.

Overall, both Assembly Fourteen (Figure 4.15) and Fifteen (Figure 4.16, next page) resulted in a decreased maximum pressure differential and a much more rapid decrease in the pressure differential when opening the proportional valve. Assembly Fourteen resulted in a maximum pressure differential of 8.55 cm H_2O. Assembly Fifteen resulted in a maximum pressure differential of 19.7 cm H_2O. Both of these values are less than thirty percent of the pressure differential generated by Assembly Thirteen. While the original grate designs resulted in the desired design criteria, the version that is capable of being fabricated does not produce the same results.

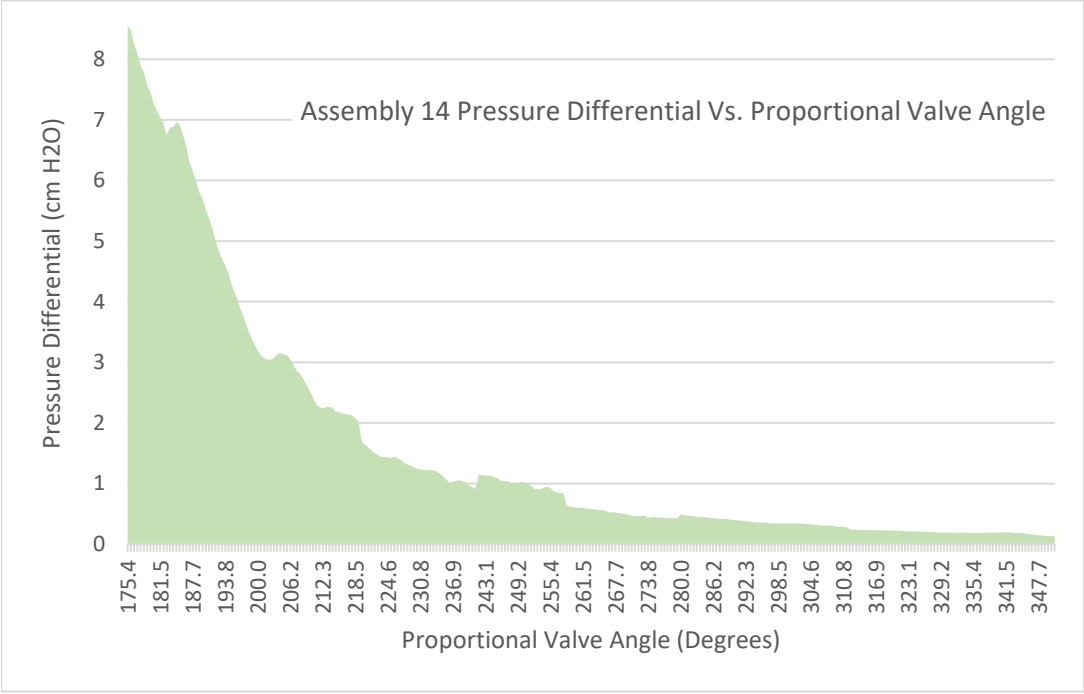


Figure 4.15 – Assembly Fourteen resulted in a much lower maximum pressure differential and lower resistance to air flow over the proportional valve area.

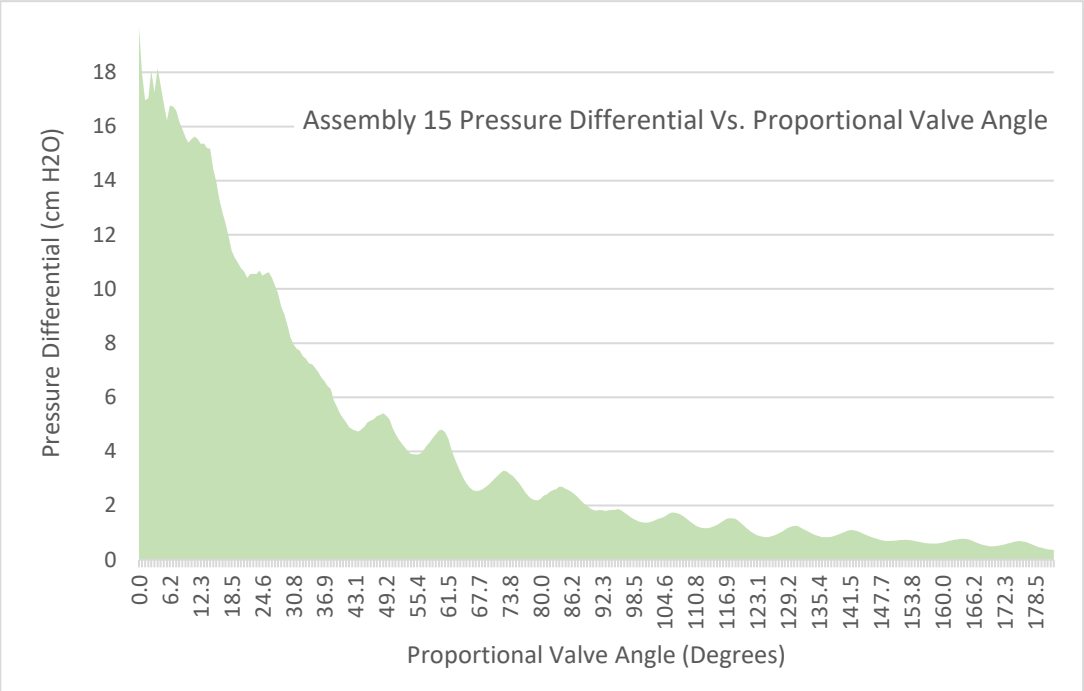


Figure 4.16 - Assembly Fifteen results were more consistent with the design criteria than Assembly Fourteen but not as consistent as Assembly Thirteen.

The final design, Assembly Seventeen, featured a yin-yang shaped cut-out on the fixed portion of the proportional valve and a grate of thirteen 1.0 millimeters holes evenly spaced out with the boundaries of the yin-yang shape on the free portion. As the proportional valve opens, a change occurs in the amount of air passed through the opening. The yin-yang design (Figure 4.17) would, in theory, create gradually smaller increases in the amount of air allowed to flow as the valve was opened. Additionally, the increased thickness on the free portion of the valve was incorporated to aid in creating an increased pressure differential compared to previous design iterations.

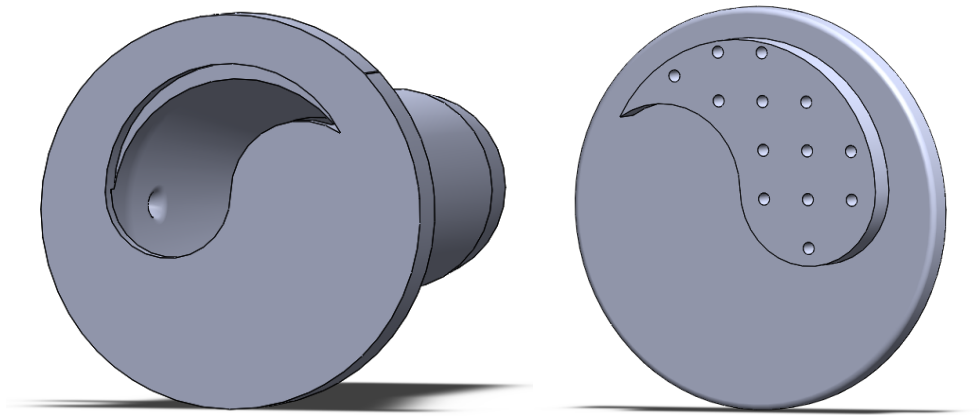


Figure 4.17 – Assembly Seventeen featured a yin-yang shaped cut-out to facilitate smaller increases in air flow as the valve was opened.

Despite the discontinuities in the graph of pressure versus valve position, assumed to be caused by the spacing of the grate cut-outs in the moveable valve body, this design resulted in a maximum pressure differential of 208.6 cm H_2O, the highest of all seventeen assemblies. Additionally, the decline in the pressure differential was gradual and very similar to the results observed in Assembly Thirteen (Figure 4.18, next page).

Overall, this simulation and design study showed that the original servo-driven valve design limited the ability to adjust proportional valve position during the training regimen. The functionality of the valve would be optimized if small changes in the pressure differential were available over the full one hundred and eighty degrees of servo rotation. While the use of grates (a grid of holes) appeared to provide an acceptable geometry to meet the design criteria, the maximum resolution of the 3D printer used to create a printed prototype for testing limited the size of the holes that make up the grating design. When the system was evaluated with the grating hole sizes that could be successfully fabricated, the results indicated that desired gradual decline in resistance to

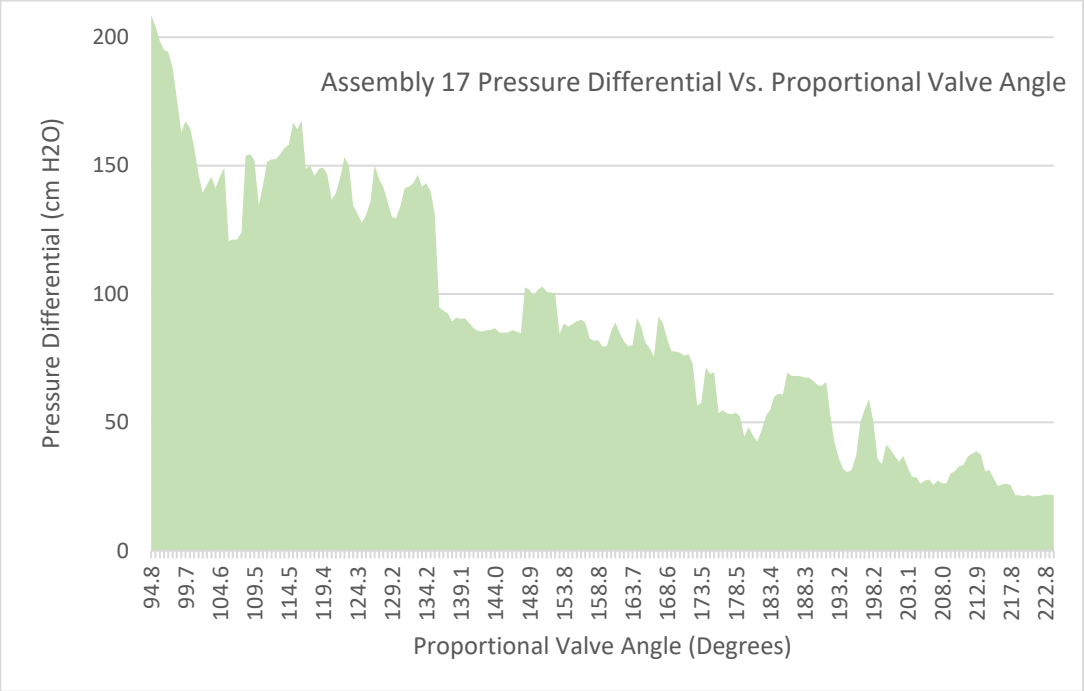


Figure 4.18 – Assembly Seventeen created the resistance to flow over approximately 130 degrees of the proportional valve.

flow as the valve was opened was unattainable. In future work, injection molding could be a way to fabricate these designs for testing.

The final yin-yang shaped design appeared to provide the most gradual decrease in pressure as the proportional valve is opened. This functionality should allow a wide range of adjustability for the system to optimize the valve position during a therapy session. Future studies on the design of the proportional valve will build on this information and continue to create subtle iterations of the yin-yang design. Figure 4.19 (next page) features the results of all of the assemblies normalized to each other in terms of position and is an excellent representation of how the subtle changes in the fixed and movable valve body affects the drop in pressure versus valve position. Zero on the X-axis of the graph represents the position on each assembly that the proportional valve is

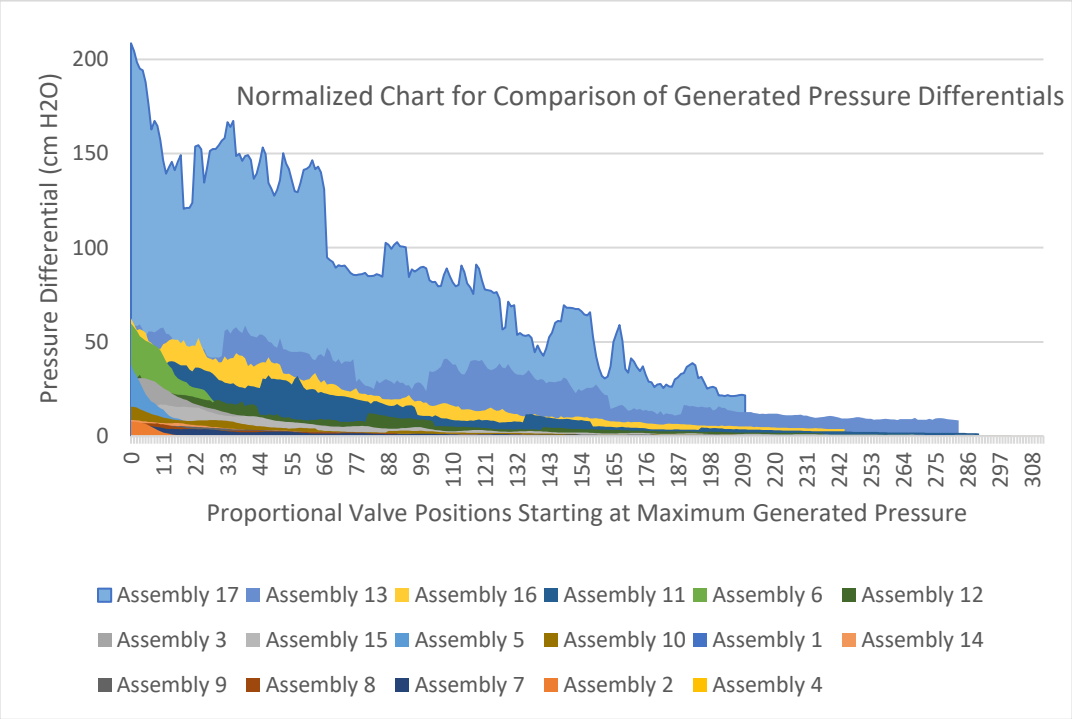


Figure 4.19 – Normalized to each other, Assembly 17 clearly generated the largest maximum pressure differential and created the largest resistance to flow over the proportional valve.

completely closed and where the maximum pressure differential is generated for the simulated maximum flow rate.

4.2 Updated Touchscreen Display and Microcontroller Code

The LCD touchscreen begins with the "start-up" page seen in Figure 4.20 (next page) and performs tests to check the function of the SD card and RTC function as well as calibrates the pressure sensor as shown in the previous code. The start-up page features two buttons, the Test Program button and a Start button. The Test Program button pulls up the Test Program page as shown in Figure 4.21 (next page).

The Test program, which was added to this iteration of the platform software, allows for the user to open or close the proportional valve using the servo motor in one-

degree increments. Pressing the “Measure” button directors the microprocessor to use the differential pressure sensor to measure the pressure for five seconds.

Figure 4.20 – The “start-up” page of the touchscreen is displayed while the program performs necessary functions for set-up.

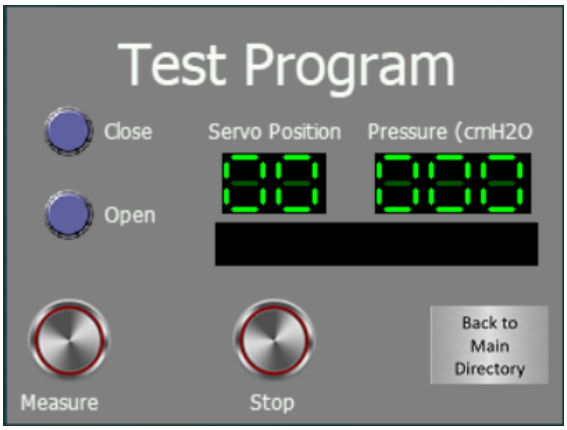


Figure 4.21 – The Test Program page was included for future testing of proportional valve designs.

If the user presses the Start button it takes them to the Directory page (Figure 4.22) as in the previous set-up. This page allows the user to choose to make a file to save all of the session results to, calculate the maximal expiratory pressure and the maximal inspiratory pressure, locate the valve positions for the training regimen, or complete the training regimen.

If the user chooses the Make File Session button it takes them to the Making New File page seen in Figure 4.23. Pressing the Start button on this page will cause the code to run as in the previous set-up to create a .csv file to save all of the data from the session to the flash drive.

Figure 4.22 – The Directory page presents the user with the necessary options to complete the device protocol.

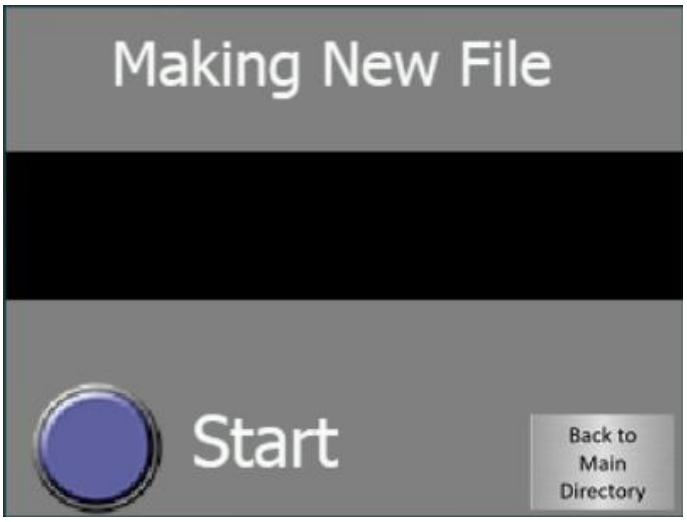


Figure 4.23 – This page walks the user through the steps being completed to read previous files and make a new file.

Once the file is created, the user can press the Back to Main Directory button to return to the Directory page seen in Figure 4.22. If the user chooses the Expiratory/Inspiratory (PE/PI_{max}) button, they will first be taken to a page to choose the training percent for the session as shown in Figure 4.24. They are able to use the slider to choose the training percent, and the current value of that slider will also be displayed in the LED digits indicator along with the chosen value from the previous session.

After choosing the appropriate training percentage and pressing next, the user is taken to the PI/PE_{max} page (Figure 4.25, next page). On this page the user will first inhale as

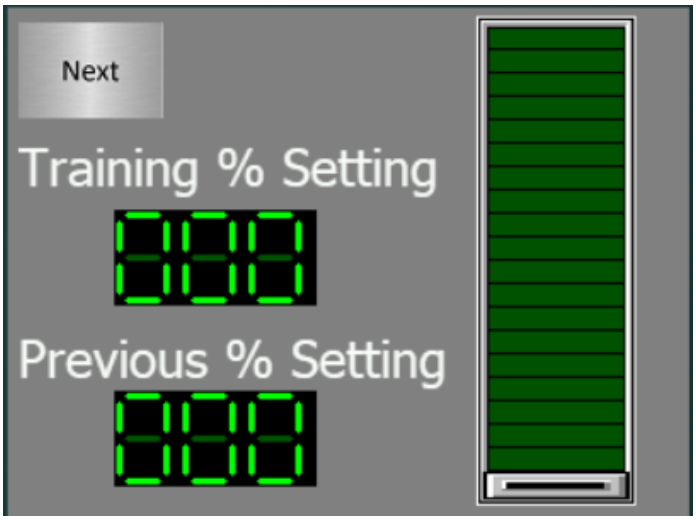


Figure 4.24 – The user uses the slider to set the training percentage for the session.

forcefully as they can from the device after pressing the PI_{max} button and then will exhale as forcefully as they can into the device after pressing the PE_{max} button. The program uses a running window average as in the previous set-up to calculate the maximum values. When the user presses the result button, the maximum values are displayed as well as the training values for exhalation and inhalation. These training values are the percentage chosen of the calculated maximum values.

The user can choose to return to the Main Directory page. They would then choose the Locate Valve Position button. This button pulls up the Valve Position Location Page (Figure 4.26).

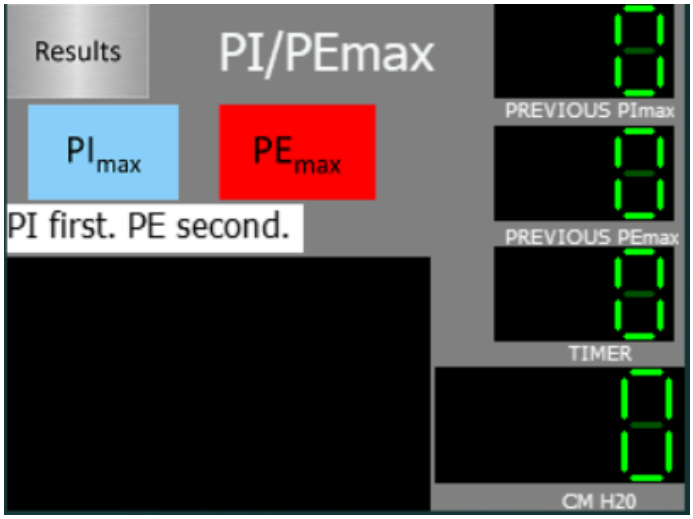


Figure 4.25 – The PI/PEmax page directs the user on when to breath to measure their maximal inspiratory and expiratory pressures.

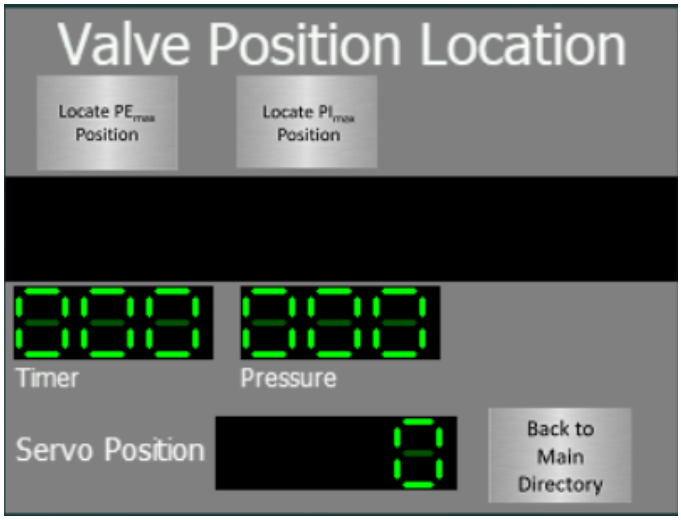


Figure 4.26 – The Valve Position Location page directs the user through the steps to locate the expiratory and inspiratory valve positions for training.

This page was added to the set-up and locates a position for both inhalation and exhalation during the training regimen. The user breathes in and out of the device normally, and as they breathe the device measures the pressure differential and moves the servo motor in one-degree increments starting at seventy-five degrees. This movement is slowly closing the proportional valve and steadily increasing resistance to flow. The proportional valve will continue to close until the pressure differential being currently measured is within ninety percent of the previously calculated training pressures. A flow chart depicting the logic of the code is shown in Figure 4.27.

Once completed, the user can choose the Back to Main Directory button to go to the Directory page again. They can finally choose the Training button and go to the Training page (Figure 4.28, next page).

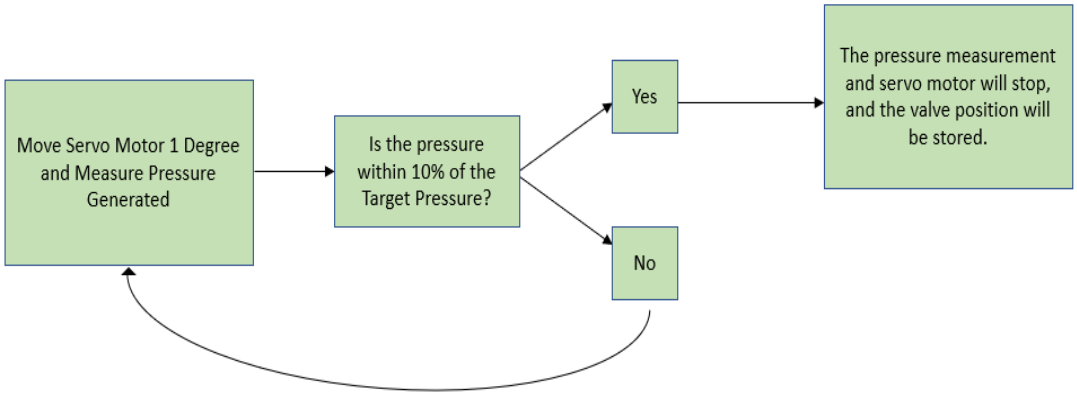


Figure 4.27 – The flow chart shows that if the pressure is not within ten percent of the target pressure, the process begins again with another servo motor movement and pressure measurement.

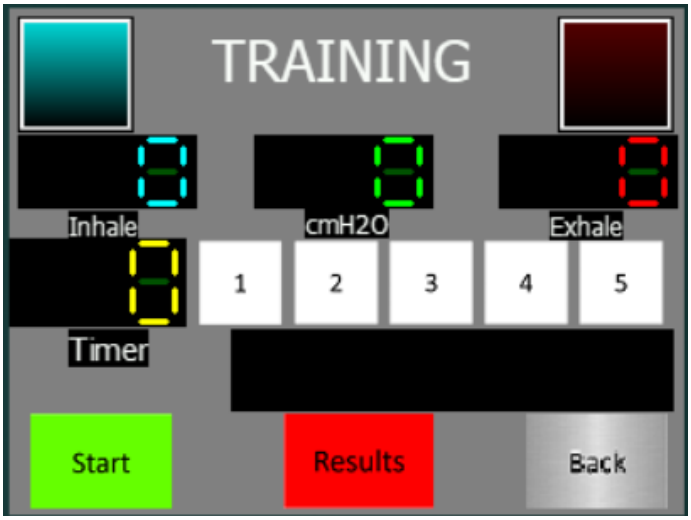


Figure 4.28 – The training page has multiple forms of feedback to direct users through the training session.

This page operates in a very similar manner to the original software. However, this generation includes new code to continuously adjust the valve position throughout the training regimen as previously stated. Figure 4.29 (next page) shows the flow chart depicting the logic of the code that adjusted the servo motor according to the percentage of the target pressure the user is generating. The code measures the pressure differential and adjusts the valve position depending on what percentage range of the target pressures the pressure is within. If the pressure is greater than zero but less than one hundred and ten percent of the target expiratory pressure, the valve is set at the expiratory valve position. If the pressure is less than zero but greater than one hundred and ten percent of the target inspiratory pressure, the valve is set at the inspiratory valve position. If the pressure exceeds one hundred and ten percent of either target pressure, the valve opens ten degrees to decrease the resistance to air flow and decrease the pressure differential.

Once the user completes the training regimen, they can choose the Results button to see a summary of the session as shown in the previous set-up. The Session Results page (Figure 4.30) was only minimally altered to include the time spent training instead of the maximum inhalation and exhalation values generated during the training regimen.

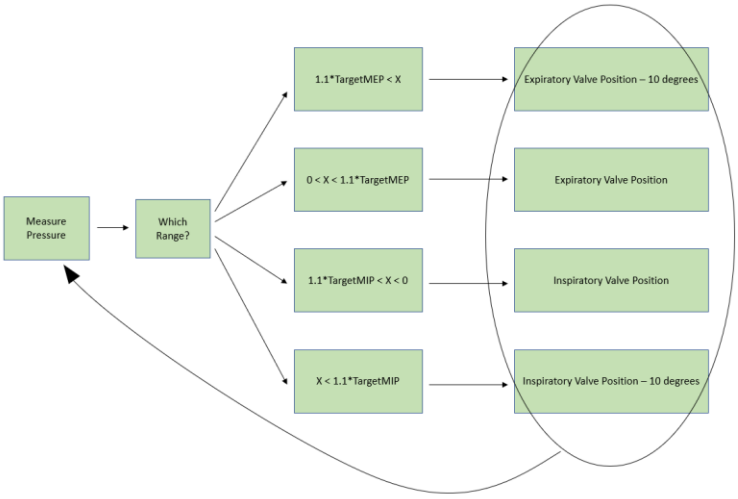


Figure 4.29 – The servo motor moved based on the percentage of the target pressure the user is generating.

Figure 4.30 – The Results page produces a summary of the previous and current training session.

The complete code for the new system configuration can be viewed in Appendix VII while a set of directions for use can be viewed in Appendix VIII.

4.3 Results of Experimental Testing with Healthy Volunteers

Six healthy volunteers completed the entire device protocol at different training percentages. The main purpose of these tests was for prototype evaluation with the inclusion of the new programming to automate the entirety of the protocol. However, due to the volunteers being tested at various percentages, it is also possible to evaluate the effects of those different percentages on their respiratory function and the effects multiple sessions have on the valve location process.

Volunteers A, C, and F were healthy males with no history of spinal cord injury or respiratory dysfunction. Volunteers B, D, and E were healthy females with no history of spinal cord injury or respiratory dysfunction. Volunteers completed three two-minute respiratory training sessions, unless the valve location process set the valve positions as completely closed for both inspiration and expiration. Once this situation occurred, their respiratory training was suspended. The training sessions were conducted at ten, fifteen, and twenty percent of the maximal expiration and inspiration pressures.

Table 4.1 and Table 4.2 (next page) show the summary of each training session for Volunteers A and B. The summary includes the training percentage, the maximal pressures, the training time they completed, the target and average pressures, and the valve positions.

Table 4.1 – Volunteer A Summary Data

Training Load (%)	10	15	20
PEmax (cm H_2O)	34	35	39
PImax (cm H_2O)	-32	-36	-35
Training Time (seconds)	120	120	120
Target Expiration Pressure (cm H_2O)	3.4	5.25	4.6
Average Expiration Pressure (cm H_2O)	2.23	3.14	4.32
Target Inspiration Pressure (cm H_2O)	-3.2	-5.4	-5.4
Average Inspiration Pressure (cm H_2O)	-2.2	-3.23	-4.06
Expiratory Valve Position (degrees)	152	136	152
Inspiratory Valve Position (degrees)	178	178	178

Table 4.2 – Volunteer B Summary Data

Training Load (%)	10	15	20
PEmax (cm H_2O)	80	61	63
PImax (cm H_2O)	-21	-53	-25
Training Time (seconds)	120	120	120
Target Expiration Pressure (cm H_2O)	8.00	9.15	12.6
Average Expiration Pressure (cm H_2O)	6.59	6.05	9.48
Target Inspiration Pressure (cm H_2O)	-2.10	-7.95	-5
Average Inspiration Pressure (cm H_2O)	-3.12	-5.45	-5.66
Expiratory Valve Position (degrees)	166	152	178
Inspiratory Valve Position (degrees)	91	162	166

Figure 4.31, **Error! Reference source not found.** (next page), and Figure 4.33 (next page) show three breath cycles during the respiratory muscle training for Volunteer A at the three training percentages. The graphs of the breath cycles also include lines dictating the target and average expiratory and inspiratory pressures.

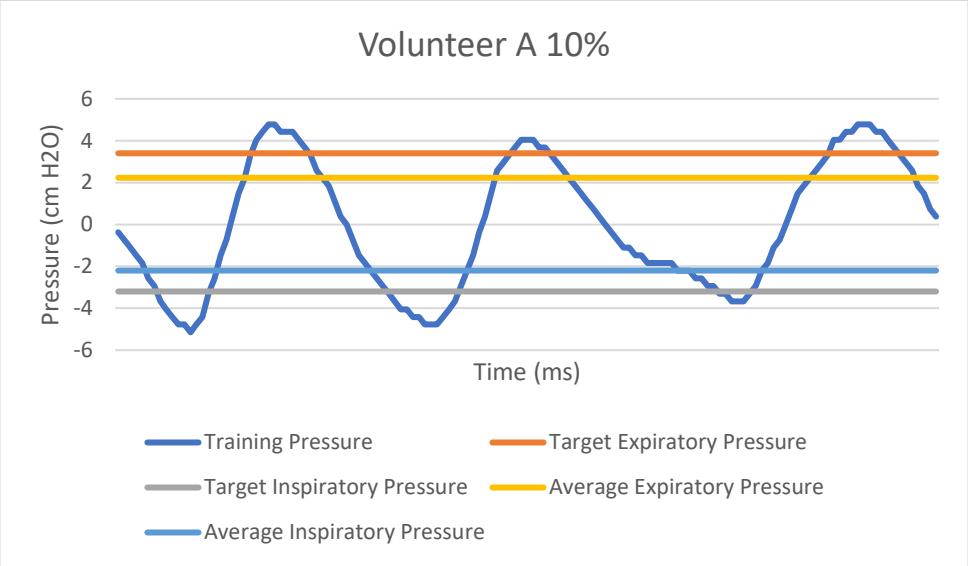


Figure 4.31 – Volunteer A Training Session at 10%

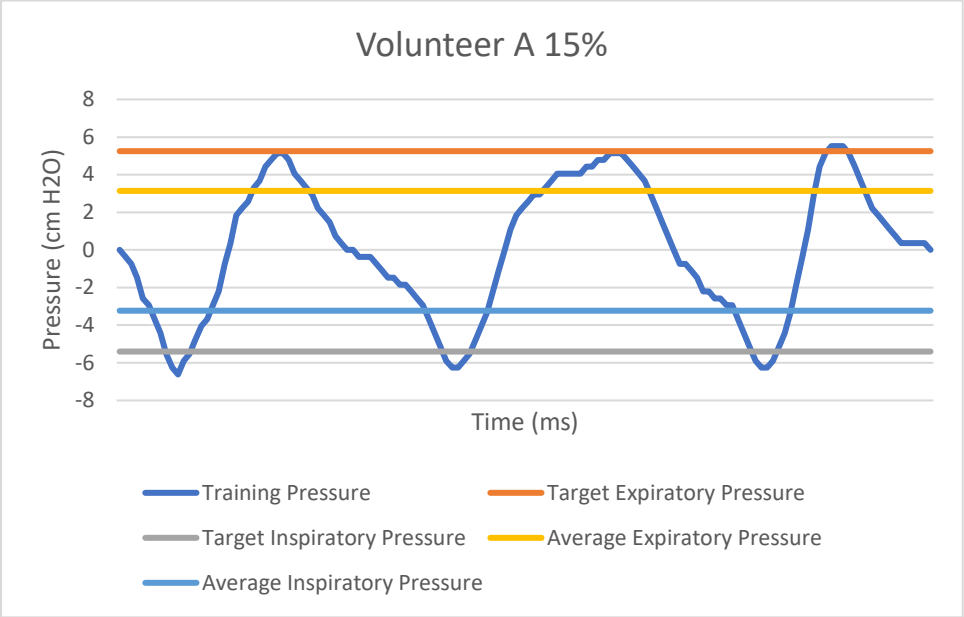


Figure 4.32 – Volunteer A Training Session at 15%

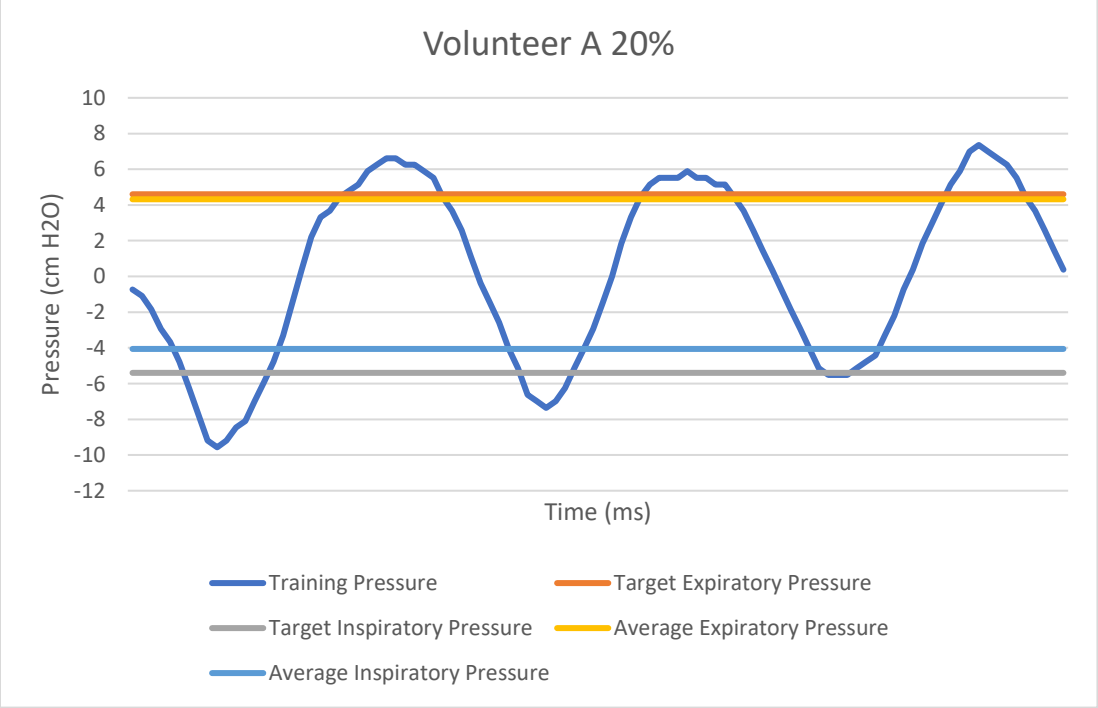


Figure 4.33 – Volunteer A Training Session at 20%

The average expiratory pressure for each session is the average of all positive pressure values recorded during the training session while the average inspiratory pressure for each session is the average of all the negative pressure values recorded during the training session. Comparing the average pressures to the target pressures allows for observation of how far the patient deviated from the measured target pressures they were aiming to hit. Volunteer A generated larger deviations, or greater differences between the average and target pressures, at fifteen percent but had lower deviations at twenty percent.

Figure 4.34, **Error! Reference source not found.** (next page), and **Error! Reference source not found.** (next page) show three breath cycles during the respiratory muscle training for Volunteer B at the three training percentages.

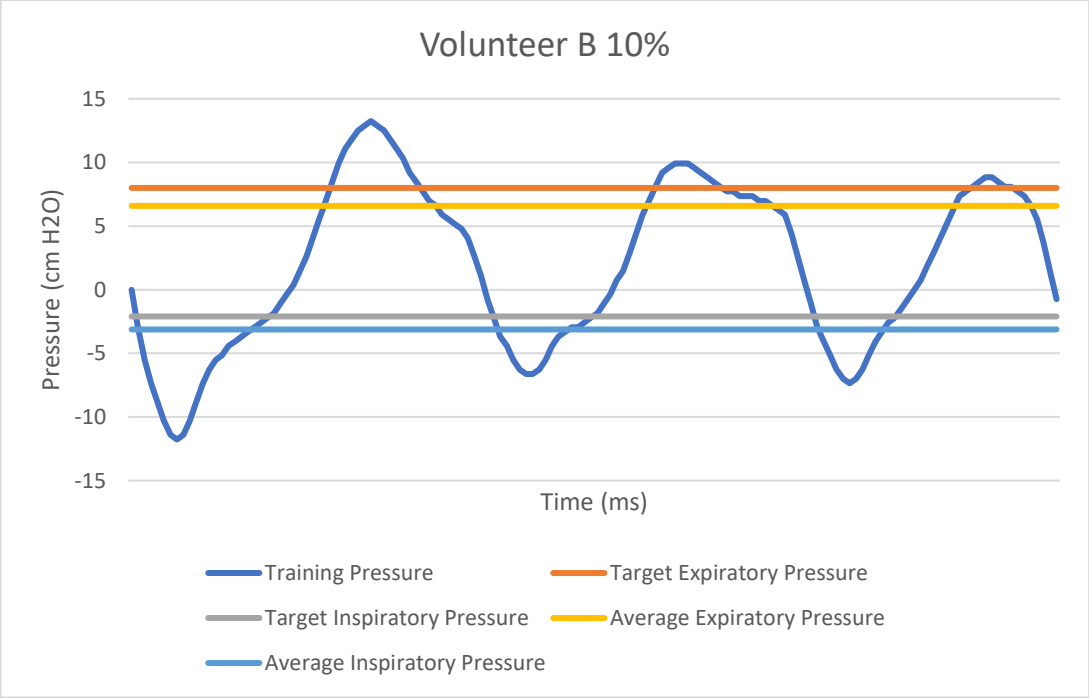


Figure 4.34 – Volunteer B Training Session at 10%

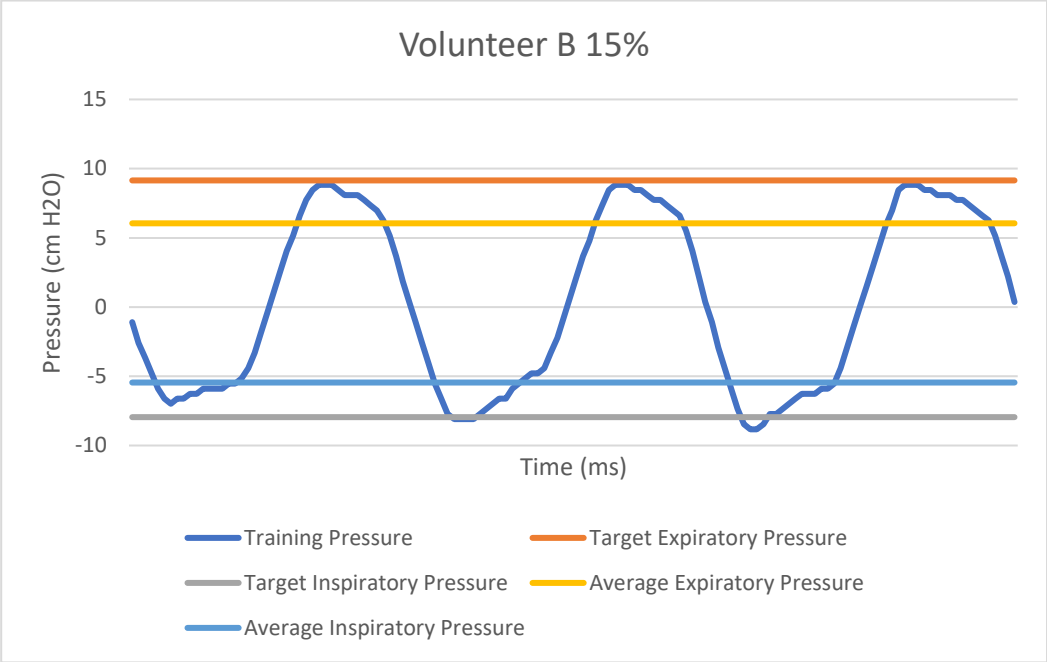


Figure 4.35 – Volunteer B Training Session at 15%

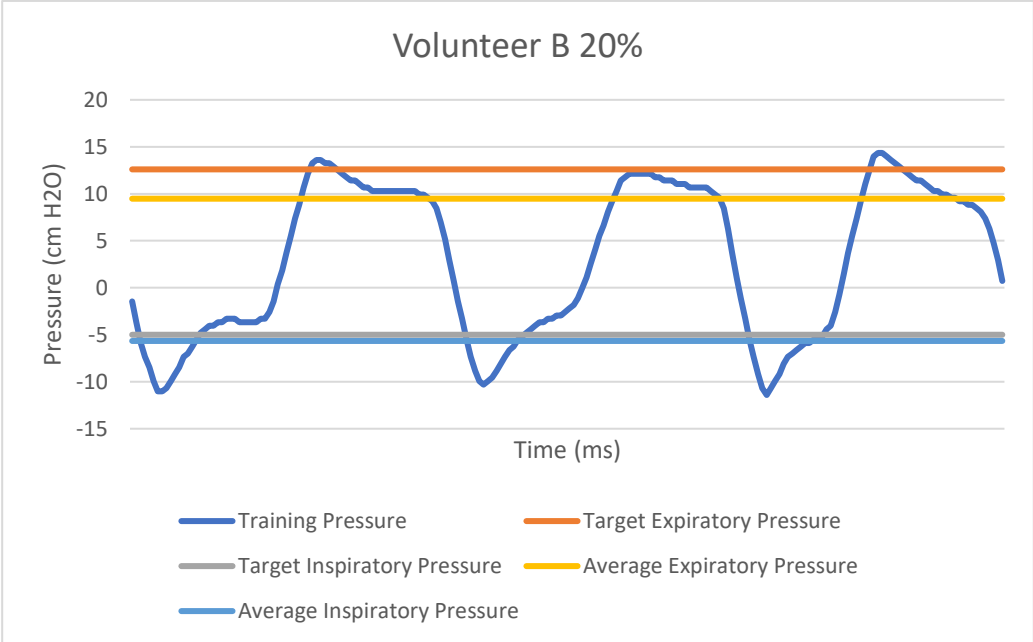


Figure 4.36 – Volunteer B Training Session at 20%

In some instances, volunteers reached the maximum valve position for both their inspiratory and expiratory positions. When this occurred, both valve positions were 178 degrees, and the respiratory muscle training was ended after the completion of that trial. Completing the next test at a higher percentage would have meant higher target pressures. The volunteer could not generate ninety percent of the calculated target pressures with the valve almost completely closed during the valve location process. They would not have been able to generate ninety percent of even greater calculated target pressures. One example of this instance was seen with Volunteer C. As shown in Table 4.3 the valve positions were 178 degrees, the maximum, and 161 degrees for expiration and inspiration respectively. Upon completing the protocol again, their maximal pressures were slightly greater along with the training percentage, and the valve locations were set to 178 degrees.

Table 4.3 – Volunteer C Summary Data

Training Load (%)	10	15	20
PEmax (cm H_2O)	76	81	N/A
PImax (cm H_2O)	-55	-64	N/A
Training Time (seconds)	120	120	N/A
Target Expiration Pressure (cm H_2O)	7.6	12.15	N/A
Average Expiration Pressure (cm H_2O)	4.48	3.92	N/A
Target Inspiration Pressure (cm H_2O)	-5.5	-9.6	N/A
Average Inspiration Pressure (cm H_2O)	5.32	-4.62	N/A
Expiratory Valve Position (degrees)	178	178	N/A
Inspiratory Valve Position (degrees)	161	178	N/A

Figure 4.37 and Figure 4.38 contain graphs of the volunteer's respiratory cycles at ten and fifteen percent. During the respiratory muscle training at fifteen percent, the volunteer has a much greater deviation between the target and average pressures, and they indicated they were feeling quite fatigued during that training session.

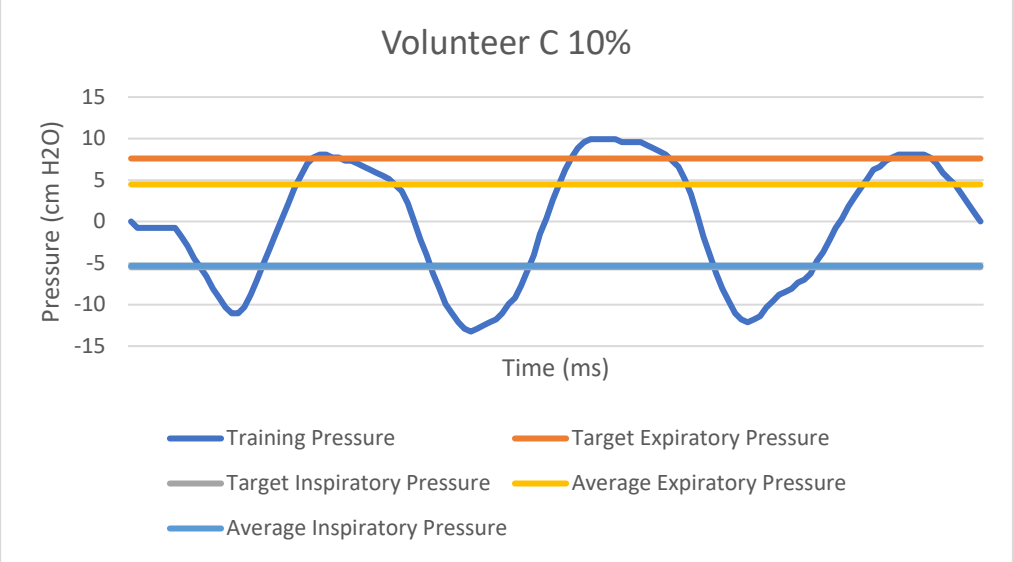


Figure 4.37 – Volunteer C Training Session at 10%

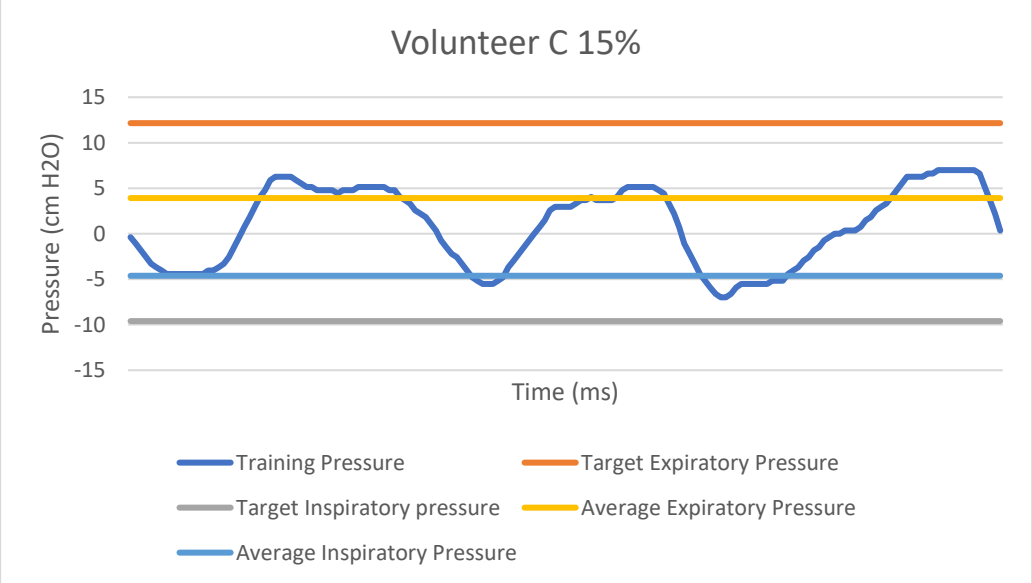


Figure 4.38 – Volunteer C Training Session at 15%

Volunteers D and E also reached the maximum valve positions for both inspiration and expiration during the respiratory training session at fifteen percent. Volunteer F reached the maximum valve position during their first session at ten percent. Summary tables and graphs characterizing their training sessions can be found in Appendix VIII.

4.4 Interpretation of Experimental Testing with Healthy Volunteers

Overall, there were deviations from the target expiratory and inspiratory pressures, but that was expected. During the first iteration of BreathForce, healthy test subjects had larger deviations than subjects with spinal cord injury. This was suspected to be a product of their healthy respiratory systems leading to an increased respiratory range.

Table 4.4 shows the average deviations for all three training percentages. A trend shows that the average deviations increase from ten to fifteen percent but decrease at twenty percent.

Table 4.4 – Average Deviations at Each Training Percent

Training Load (%)	10	15	20
Expiratory Pressures (cm H2O)	1.58	3.73	1.70
Inspiratory Pressures (cm H2O)	2.82	2.99	1.00

This could be the product of two things. The volunteers were becoming fatigued by the third training session, so their respiratory range was decreased. Additionally, they were becoming more comfortable operating and reading the device and were better able to match the generated pressure shown on the touchscreen to the target pressures.

Additionally, three volunteers were not able to complete the training session at twenty percent of their maximal pressures while the fourth volunteer was not able to complete the training session at fifteen and twenty percent due to reaching the maximum valve position for both the expiratory and inspiratory pressures at ten percent. There are two possible reasons for the volunteers not being able to generate a pressure within ten percent of their maximal pressures even with the valve completely closed. The first is that the design of the proportional valve is still not generating enough resistance to increase the pressure differential. More designs and iterations of the yin-yang shape are still being tested to increase the maximum pressure differential generated and maintain resistance to air flow over a larger area of the valve.

Another possible reason is that the program is generating artificially high maximal inspiratory and expiratory pressures. When the maximal inspiratory and expiratory pressures are calculated, the volunteers are instructed to breath as forcefully as they can in or out of the device. The program moves the servo to almost close the valve completely. Currently, the program moves the servo to one hundred and seventy degrees. It is possible, that the valve is closed too much and, depending on how the volunteers breathe into the device, generating artificially high pressures. Then when the target pressures are calculated as fifteen or twenty percent of these maximal values, they are too high for the volunteer to generate breathing normally into the device, even when completely closed.

A future study will be conducted to determine at which position the valve should be set to generate accurate maximal pressures for all subjects. The first iteration included a specific mouthpiece for measuring the maximal pressures that had a leak valve with an

inner diameter of 2.58 mm. A future potential study could include the use of SolidWorks to calculate the area of the open proportion of the proportional valve at each incremental valve position to locate a valve position with an area equivalent to that leak valve and use that position within the programming.

Another issue encountered was that adding the programming to continuously adjust the servo motor and valve position during training decreased the resolution of the data acquisition during the training session. Each time a session is completed, and a file is created, the program calculates the amount of data points stored per second and stores that in the file as well. It was found that while the original program had a sampling rate of approximately twenty points per second, the new programming had decreased the resolution to approximately seven points per second. Changes were made to the programming including creating a subroutine for moving the servo motor during the respiratory training session and changing the if-else statements that directed the servo movement to case statements. These changes can be seen in Appendix IX. Overall, the resolution was improved, and the sampling rate was increased to thirty-four points per second. This increase greatly improved the accuracy of the data.

Additionally, the test data revealed that at the gain setting used on the circuit board, the analog measurement channel had a resolution of 0.37 bytes. In order to reduce unnecessary noise, a dead band was incorporated into the code to move the servo motor only when the pressure is outside the bit noise floor of the analog channel. The dead band was classified as any pressure greater than -0.74 cm H_2O and less than 0.74 cm H_2O. The code would then direct that if the user was within the dead-band range, the servo motor

would not move. Figure 4.39 shows the flow chart depicting the logic of the final code including the dead band.

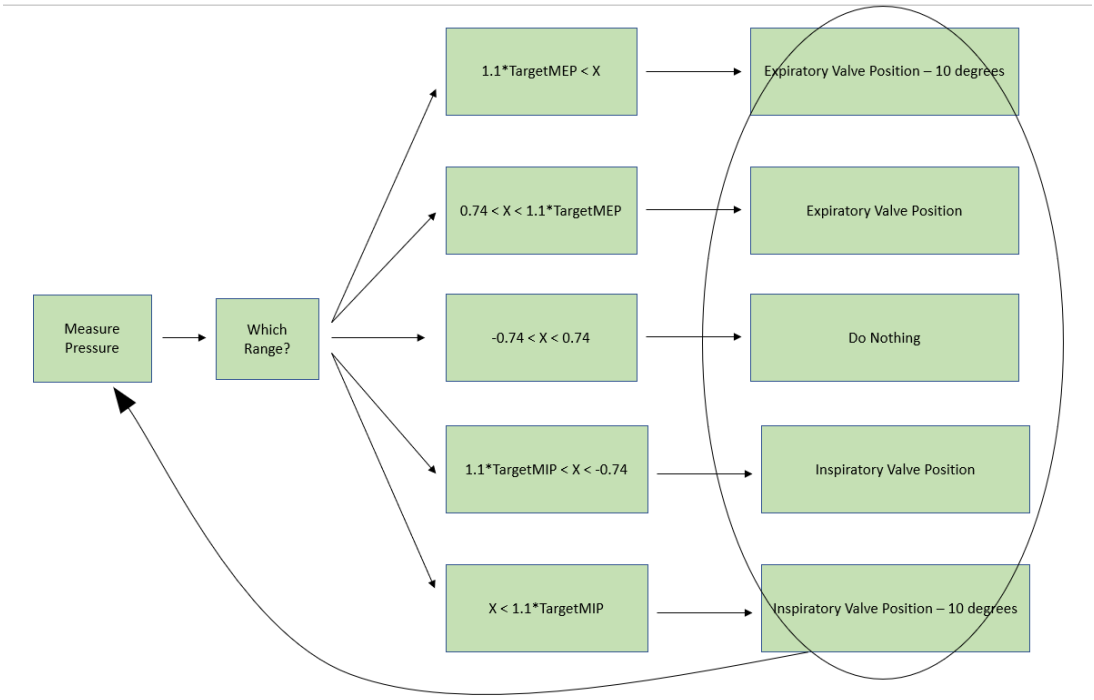


Figure 4.39 – The updated code included a dead band as shown in the flow chart.

Finally, it was noted that during the testing, the LED indicating that the device was receiving power would dim upon movement of the servo. Additionally, the servo movement would also cause the touchscreen to freeze. Further testing revealed that the movement of the servo motor was causing the current usage to spike. Future hardware iteration will be completed to compensate for the power requirements of the device.

V. CONCLUSION

5.1 Proof of Concept

Overall, the criteria for this project were met and the design and operation of BreathForce was refined and completely automated. After testing seventeen different proportional valve designs using flow simulations, a design was found with a high maximum pressure differential that maintained resistance over a large area of the proportional valve as it was opened incrementally. Microcontroller code was created to fully automate the entire device protocol including the system-controlled movement of the proportional control valve. The software calculates the maximal and target pressures, locates the valve position for both expiration and inspiration, and continuously adjusts the valve position during training to aid the user in generating pressure within ten percent of the target pressures. Experimental testing with healthy volunteers revealed that the new additions had lowered the sample rate to seven points per second. Changes were made to the logic of the program that increased the sampling rate to thirty-four points per second.

The complete automation of the device will allow for at-home use that enforces the device protocols and leads to consistent effective use of the device for respiratory muscle training. More consistent, effective use of the device will enhance respiratory muscle rehabilitation and respiratory infection prevention.

5.2 Future Development of BreathForce

Currently, the continued automation and development of BreathForce is limited by the device hardware. The next step would be to create a truly active valve that would start each inspiration or expiration completely open and close the valve incrementally until the subject reaches the target pressures. However, the current microcontroller does not possess enough processing power to acquire data and send commands fast enough to accurately complete that process. Prior to the addition of programming that directs a truly active valve, the hardware must be redesigned to include two microcontrollers. One would be used to acquire data and send commands to the servo motor while the other would be used to send commands to the touchscreen.

Additionally, future studies need to be completed to determine the position of the servo motor and proportional valve during the measurement of the maximal pressures as well as the most effective proportional valve design. These studies need to be completed to calculate accurate maximal pressures and aid subjects in reaching the target pressures during the valve location process and respiratory training session.

Finally, the newest software for the second-generation device produces high fidelity data of the participants' respiratory cycles. The clinicians will need to analyze that data to determine if the data gives them the same information as the first-generation device or if there have been alterations due using a restriction valve instead of a check valve. Additionally, it needs to be determined whether the second generation provides an equivalent or better experience and training session through application of the device in a clinical setting with spinal cord injury patients and analysis of data. Mastering the ability to adapt the design based on CFD simulation results in this study allows for continued

refinement of the hardware to continue improvement of the device per the guidelines of the clinicians.

REFERENCES

1. Kupfer, Mendel, et al. "Spinal Cord Injury." CURRENT Diagnosis & Treatment: Physical Medicine & Rehabilitation Eds. Ian B. Maitin, and Ernesto Cruz. McGraw-Hill, 2014, https://accessmedicine-mhmedical-com.echo.louisville.edu/content.aspx?bookid=1180&sectionid=70377348.
2. National Spinal Cord Injury Statistical Center, Facts and Figures at a Glance. Birmingham, AL: University of Alabama at Birmingham, 2020.
3. Keenan, Mary Ann E., et al. "Chapter 12. Rehabilitation." *Current Diagnosis & Treatment in Orthopedics, 5e* Eds. Harry B. Skinner, and Patrick J. McMahon. McGraw-Hill, 2014, https://accessmedicine-mhmedical-com.echo.louisville.edu/content.aspx?bookid=675&sectionid=45451718.
4. "Types and Levels of Spinal Cord Injuries." *Spinal Cord Injury Levels and Types | Shepherd Center Rehabilitation*, Shepherd Center, 2021.
5. "Costs of Living with Spinal Cord Injury." *Reeve Foundation*, Reeve Foundation, 2021, www.christopherreeve.org/living-with-paralysis/costs-and-insurance/costs-of-living-with-spinal-cord-injury.
6. Berlowitz, David J et al. "Respiratory problems and management in people with spinal cord injury." Breathe (Sheffield, England) vol. 12,4 (2016): 328-340. doi:10.1183/20734735.012616

7. Terson de Paleville, Daniela G L et al. “Respiratory motor control disrupted by spinal cord injury: mechanisms, evaluation, and restoration.” Translational stroke research vol. 2,4 (2011): 463-73. doi:10.1007/s12975-011-0114-0
8. Nógrádi A, Vrbová G. Anatomy and Physiology of the Spinal Cord. In: Madame Curie Bioscience Database [Internet]. Austin (TX): Landes Bioscience; 2000- 2013. Available from: https://www.ncbi.nlm.nih.gov/books/NBK6229/
9. A., Rajalakshmi. (2018). Influence of Early Acquaintance with Dikshitar's Nottuswaras on Cognitive Development, Communication and Social-Emotional Learning in Preschool Children. 10.13140/RG.2.2.13266.20169.
10. Harrow-Mortelliti M, Reddy V, Jimsheleishvili G. Physiology, Spinal Cord. [Updated 2021 Feb 17]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2021 Jan-. Available from: https://www.ncbi.nlm.nih.gov/books/NBK544267/
11. “Spine Anatomy, Anatomy of the Human Spine.” Mayfieldclinic.com, Mayfield Brain and Spine , Sept. 2018, mayfieldclinic.com/pe-anatspine.htm#:~:text=Vertebrae%20are%20the%2033%20individual,sacrum%20and%20coccyx%20are%20fused.
12. Garshick, Eric, et al. “Respiratory Health and Spinal Cord Injury.” Respiratory Health and Spinal Cord Injury | Model Systems Knowledge Translation Center (MSKTC), Model Systems Knowledge Translation Center (MSKTC), 2015,

msktc.org/sci/factsheets/respiratory#:~:text=in%20your%20body.-,How%20does%20the%20respiratory%20system%20work%3F,levels%20to%20 ontract%20the%20diaphragm.

13. Slosar, Paul. "Cervical Spinal Nerves." *Spine - Health*, Veritas Health, LLC, 31 May 2019, www.spine-health.com/conditions/spine-anatomy/cervical-spinal-nerves.
14. Shier, David, et al. *Holes Essentials of Human Anatomy & Physiology + Lab Manual*. 14th ed., McGraw-Hill College, 2016.
15. Editorial, AnaesthesiaUK. "Spirometry ." *Anaesthesia UK : Tests of Pulmonary Function*, AnaethesiaUK, 11 Feb. 2004, www.frca.co.uk/article.aspx?articleid=100023.
16. Brown, Robert et al. "Respiratory dysfunction and management in spinal cord injury." Respiratory care vol. 51,8 (2006): 853-68;discussion 869-70.
17. Stepp, Evan L et al. "Determinants of lung volumes in chronic spinal cord injury." Archives of physical medicine and rehabilitation vol. 89,8 (2008): 1499-506. doi:10.1016/j.apmr.2008.02.018
18. Brown, Robert et al. "Respiratory dysfunction and management in spinal cord injury." Respiratory care vol. 51,8 (2006): 853-68;discussion 869-70.
19. Gross D, Ladd HW, Riley EJ, Macklem PT, Grassino A. The effect of training on strength and endurance of the diaphragm in quadriplegia. Am J Med 1980;68(1):27–35.

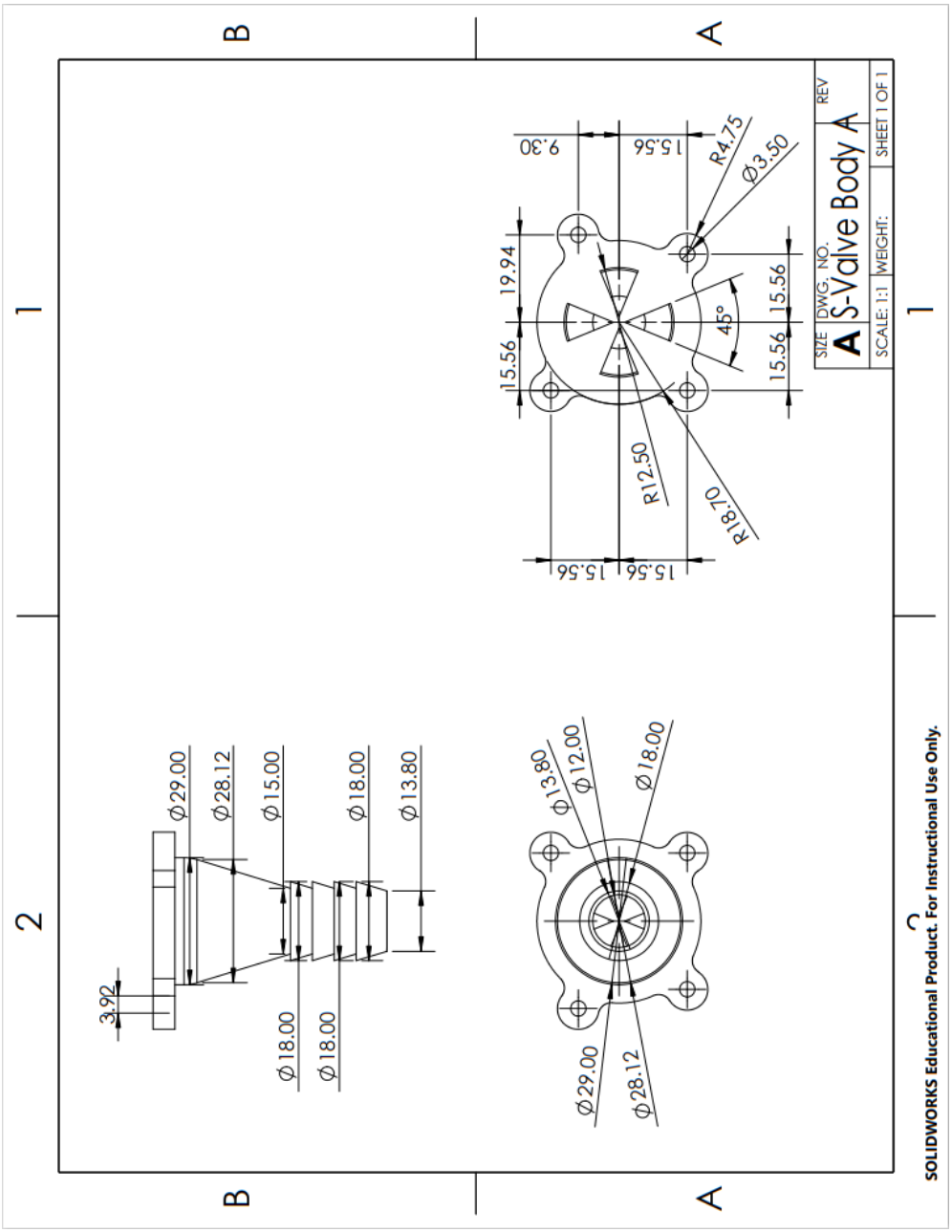
20. Rutchik A, Weissman AR, Almenoff PL, Spungen AM, Bauman WA, Grimm DR. Resistive inspiratory muscle training in subjects with chronic cervical spinal cord injury. Arch Phys Med Rehab 1998;79(3):293–297.
21. Kogan I, McCool FD, Liberman SL, Garshick E, Shannon K, Frisbee JH, Brown R. Diaphragm hypertrophy during inspiratory muscle training in tetraplegia (abstract). Am J Respir Crit Care Med. 1996;153(4):A25.
22. "The Breather." *PN Medical*, PN Medical, www.pnmedical.com/product/the-breather/.
23. Homnick, Douglas N. "Mechanical insufflation-exsufflation for airway mucus clearance." Respiratory care vol. 52,10 (2007): 1296-305; discussion 1306-7.
24. "Used EMERSON CA-3000 Cough Assist Device For Sale - DOTmed Listing #906026." *DOTmed.com*, DOTmed.com, Inc., www.dotmed.com/listing/cough-assist-device/emerson/ca-3000/906026.
25. "Respironics Cough Assist T70." *No Insurance Medical Supplies*, www.noinsurancemedicalsupplies.com/respironics-cough-assist-t70/?sku=1098160&gclid=EAIaIQobChMIkNjkw6TY7wIVicDACh3isQ_YEAQYAyABEgI2vfD_BwE.
26. Kumar, Krishna, and Sharon Bishop. "Financial impact of spinal cord stimulation on the healthcare budget: a comparative analysis of costs in Canada and the United States." Journal of neurosurgery. Spine vol. 10,6 (2009): 564-73. doi:10.3171/2009.2.SPINE0865
27. Tran, Kevin L. *Design, Development, and Characterization of BreathForce, A*

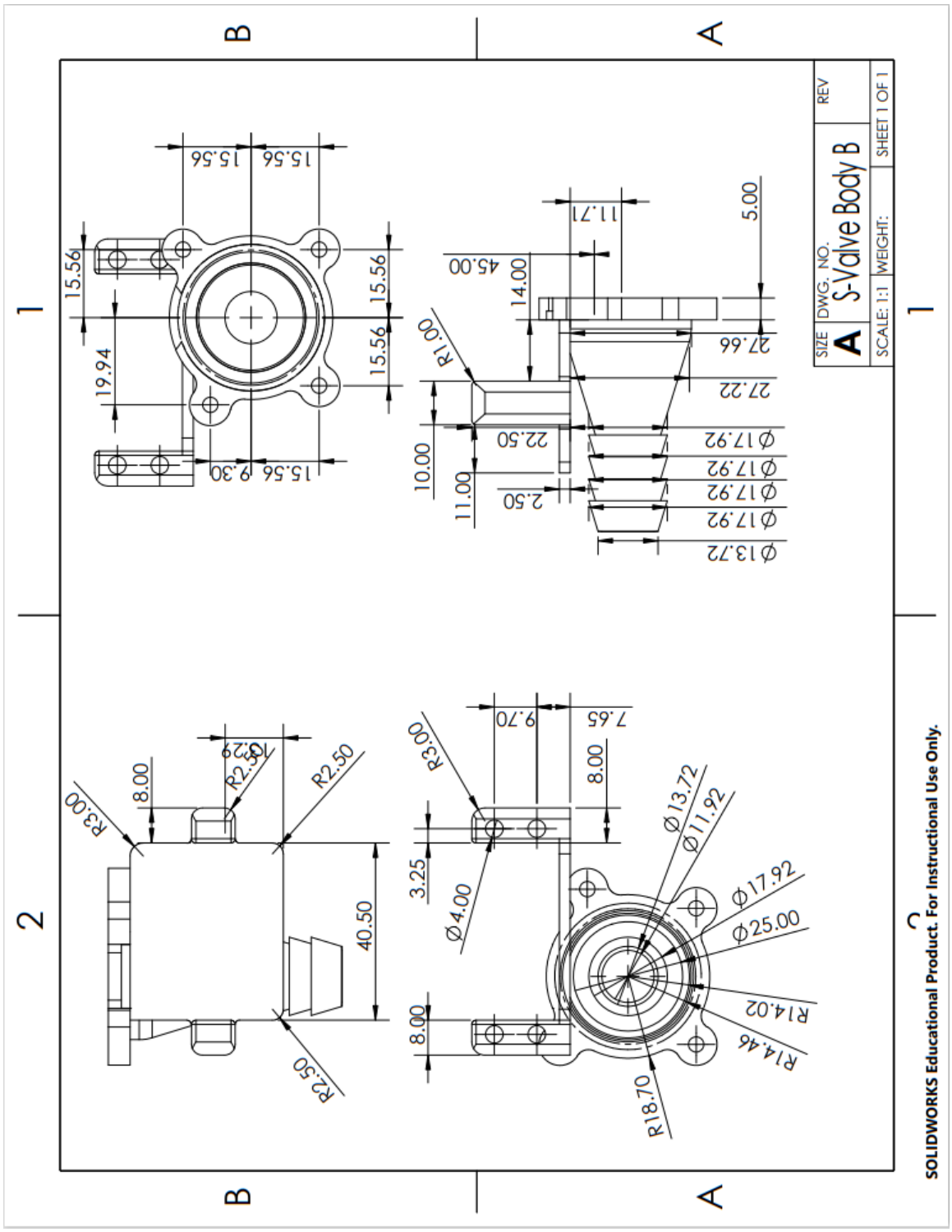
Respiratory Training System for Patients with Spinal Cord Injuries. MEng Thesis. University of Louisville, 2017 Dec.

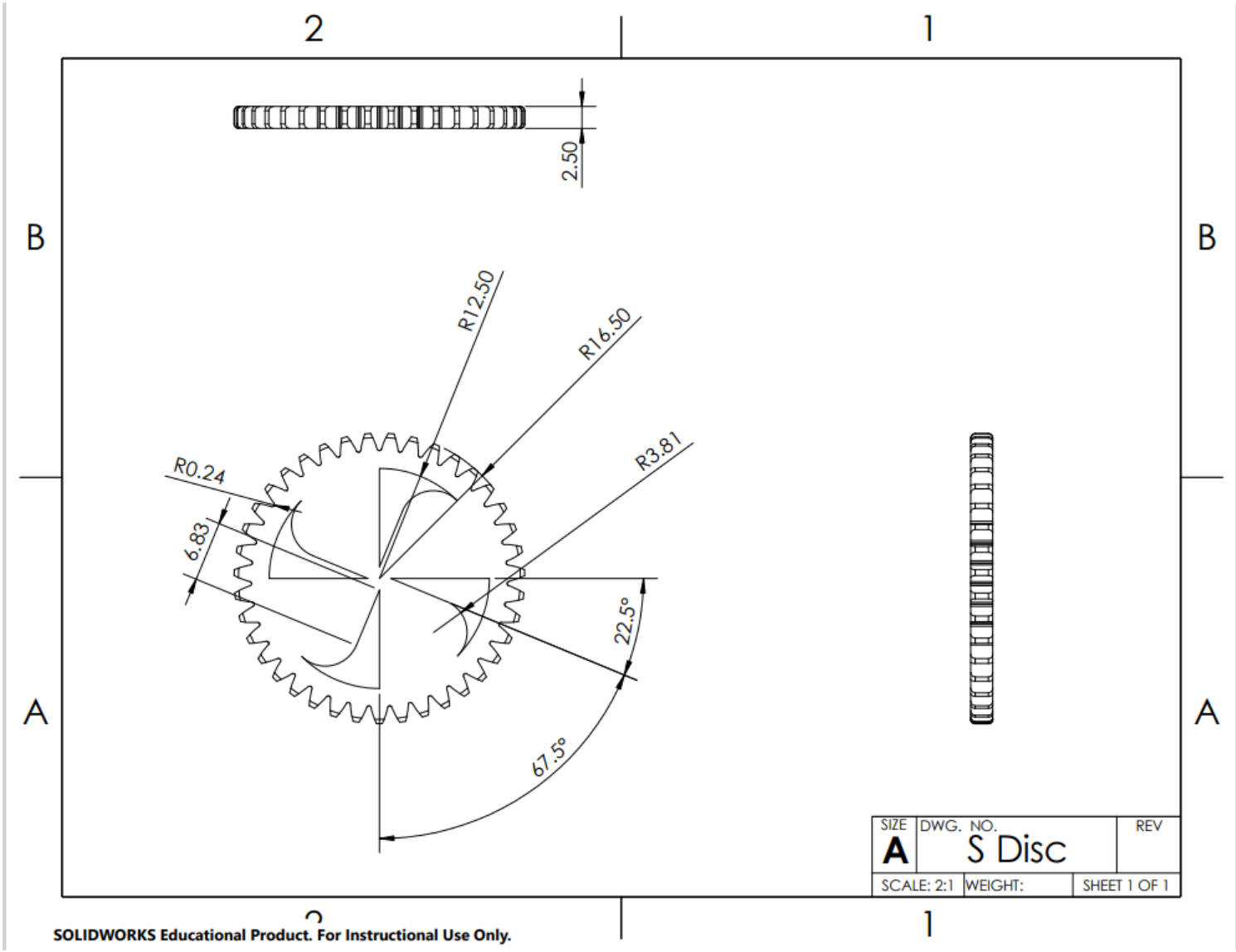
28. "Engineering Essentials: Flow-Control Valves." *Hydraulics Pneumatics*, Endeavor Business Media, 1 Jan. 2012, www.hydraulicspneumatics.com/technologies/hydraulic-valves/article/21885085/engineering-essentials-flowcontrol-valves.
29. Hallett S, Toro F, Ashurst JV. Physiology, Tidal Volume. [Updated 2020 Jun 1]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2021 Jan.

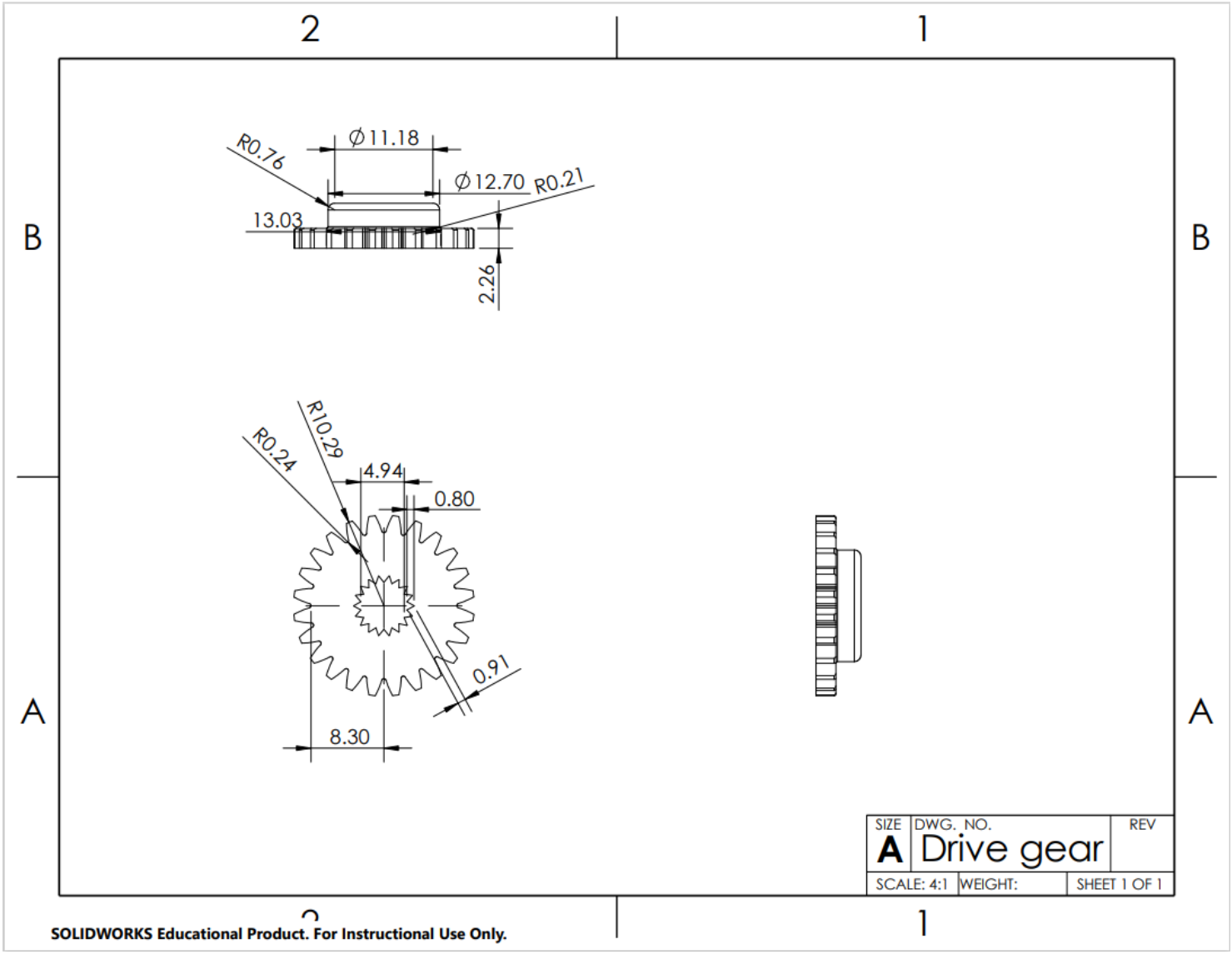
VI. APPENDIX I

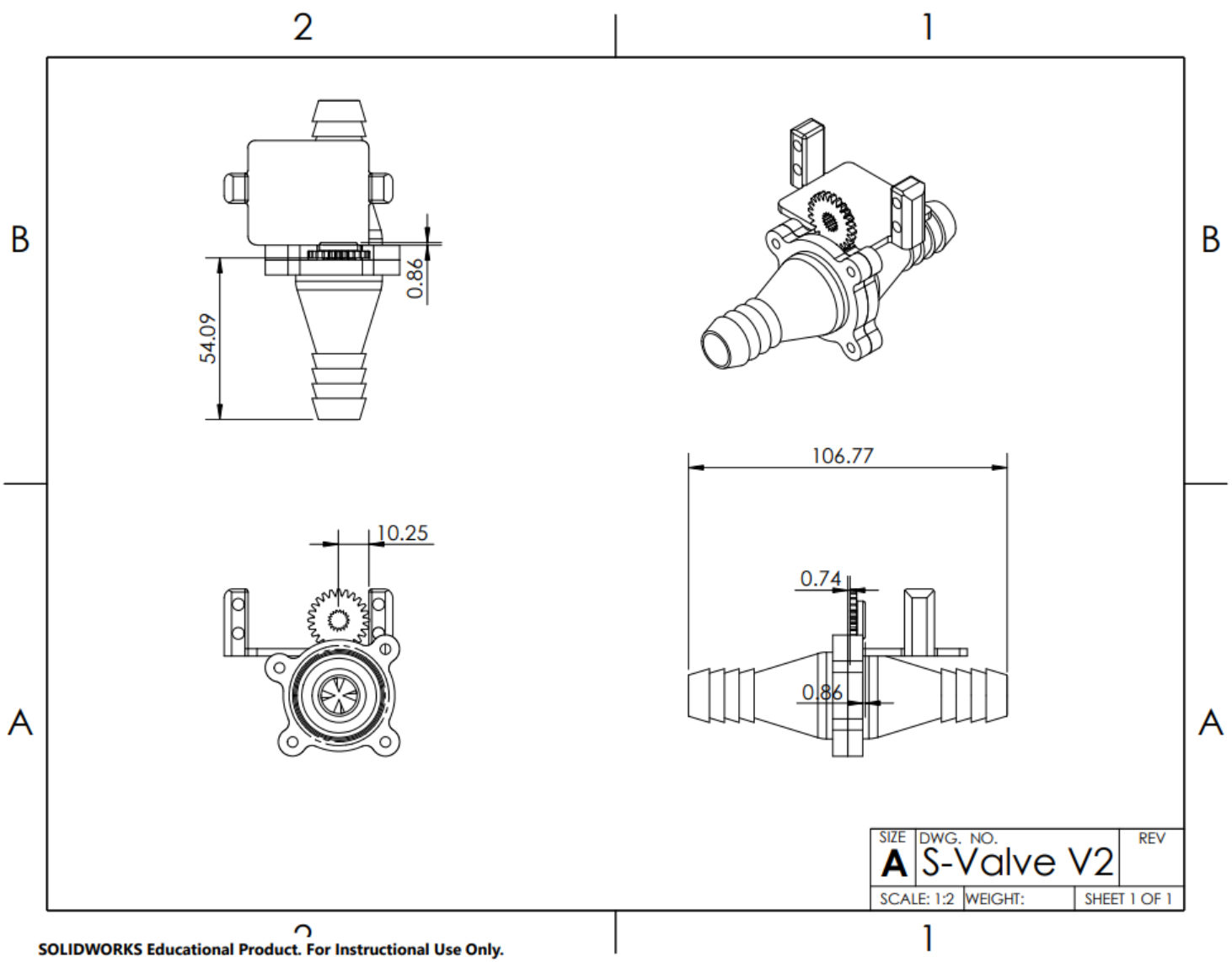
The schematic drawings of the original BreathForce design:




SIZE
A
DWG. NO.
S-Valve Body B
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

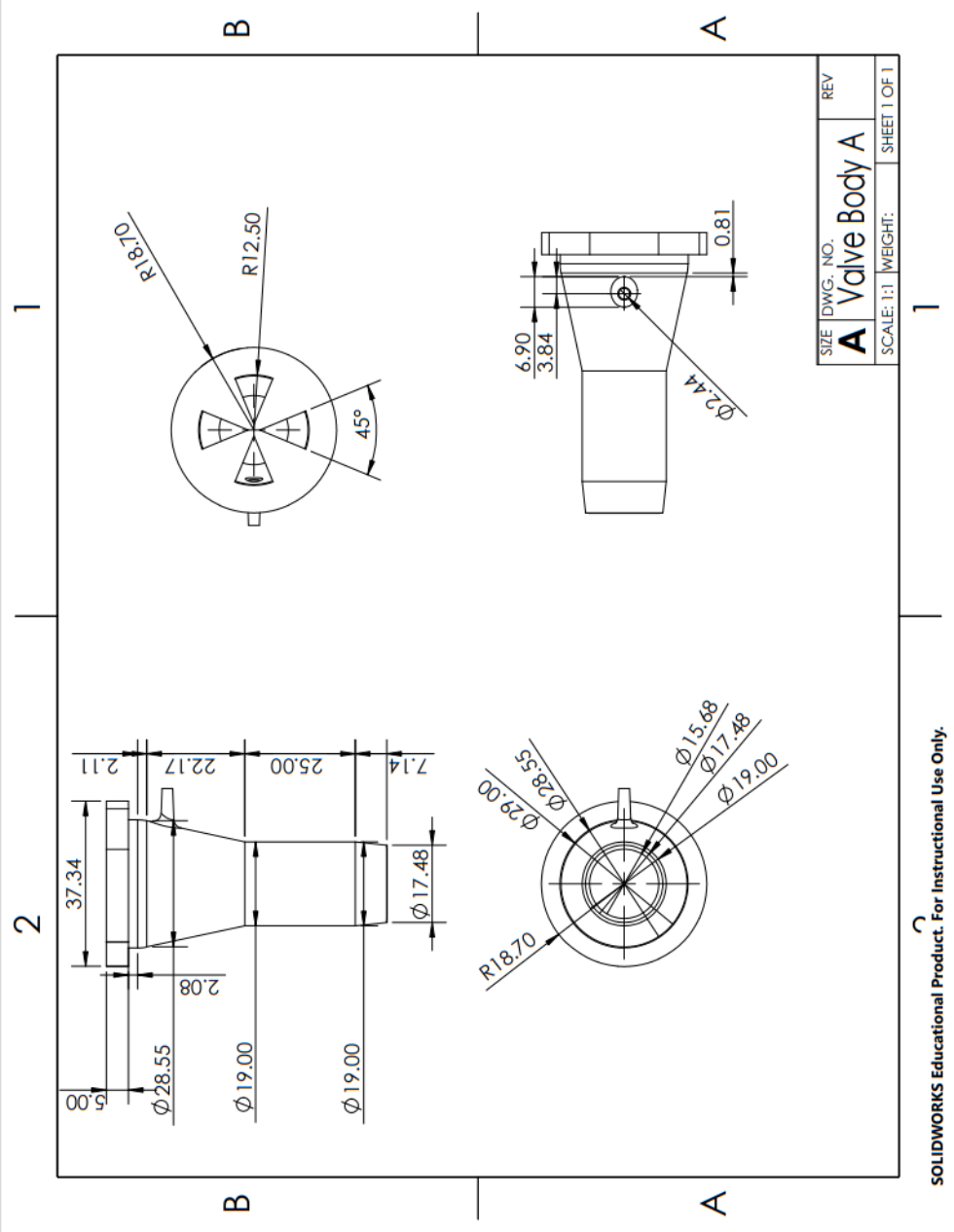

2
1
B
A
2.50
R12.50
R16.50
R3.81
R0.24
6.83
22.5°
67.5°
SIZE
A
DWG. NO.
S Disc
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

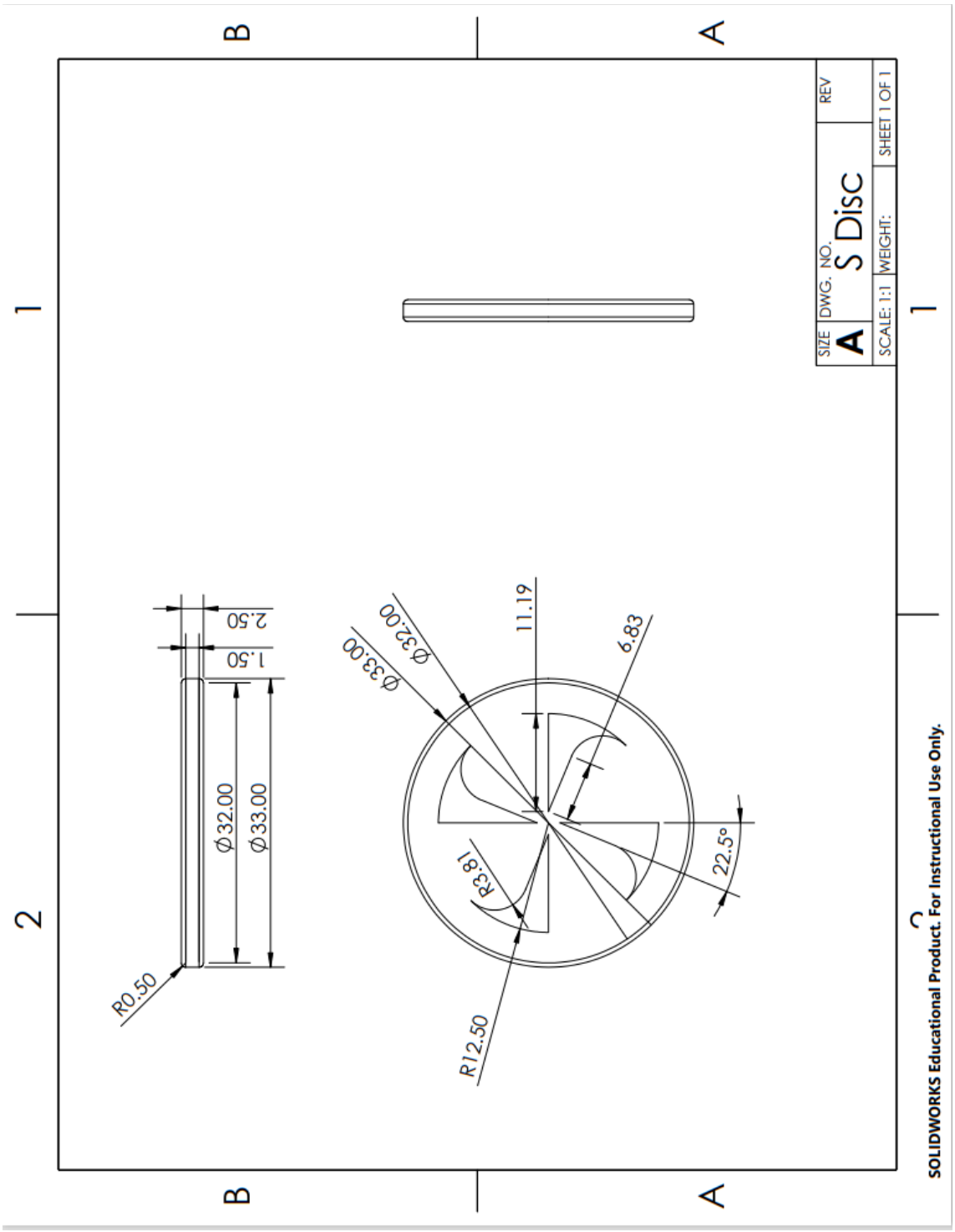

2
1
B
A
R0.76
Ø11.18
Ø12.70
R0.21
13.03
2.26
R10.29
R0.24
4.94
0.80
0.91
8.30
SIZE
A
DWG. NO.
Drive gear
REV
SCALE: 4:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

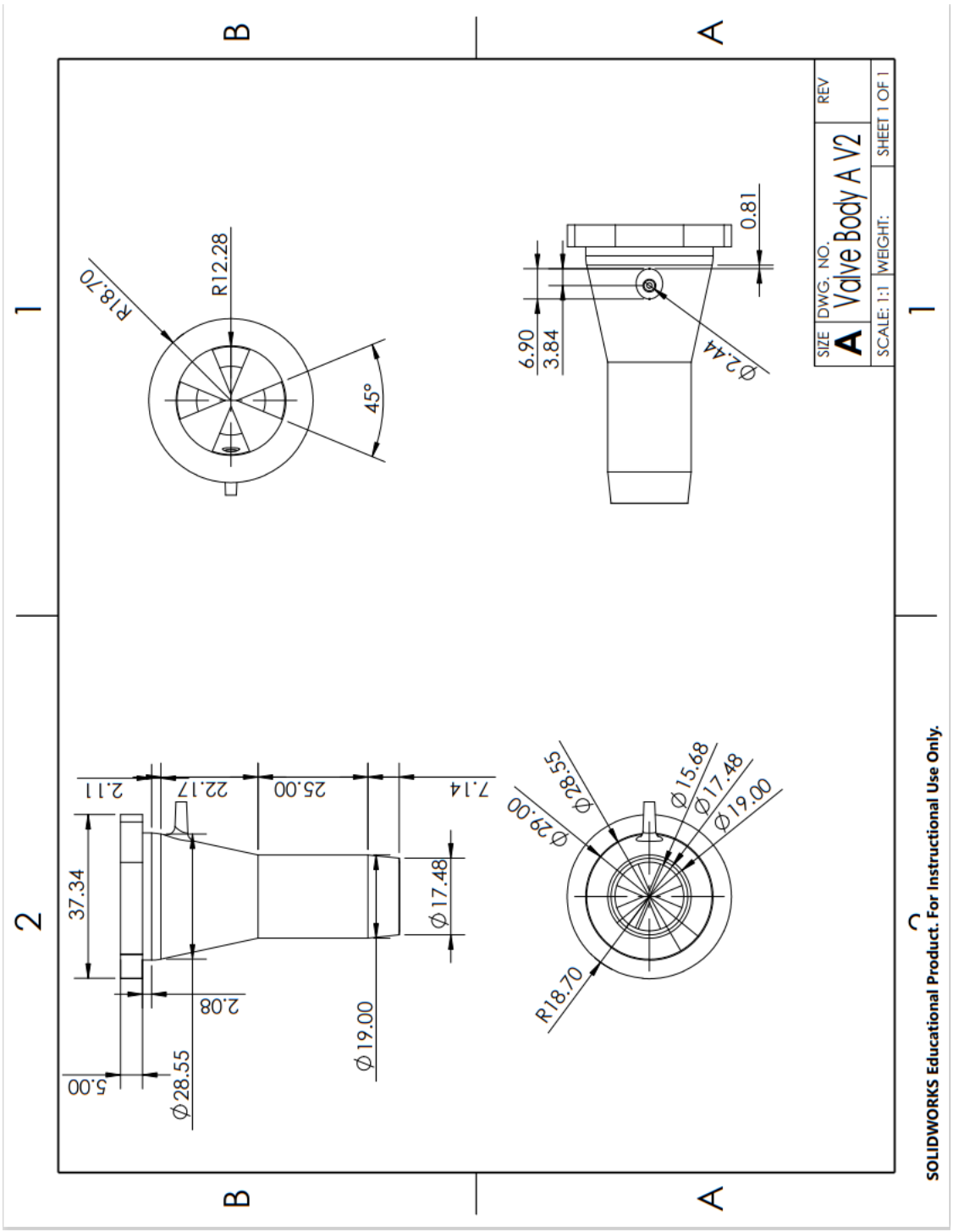

2
1
B
A
54.09
0.86
10.25
106.77
0.74
0.86
SIZE
A
DWG. NO.
S-Valve V2
REV
SCALE: 1:2
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

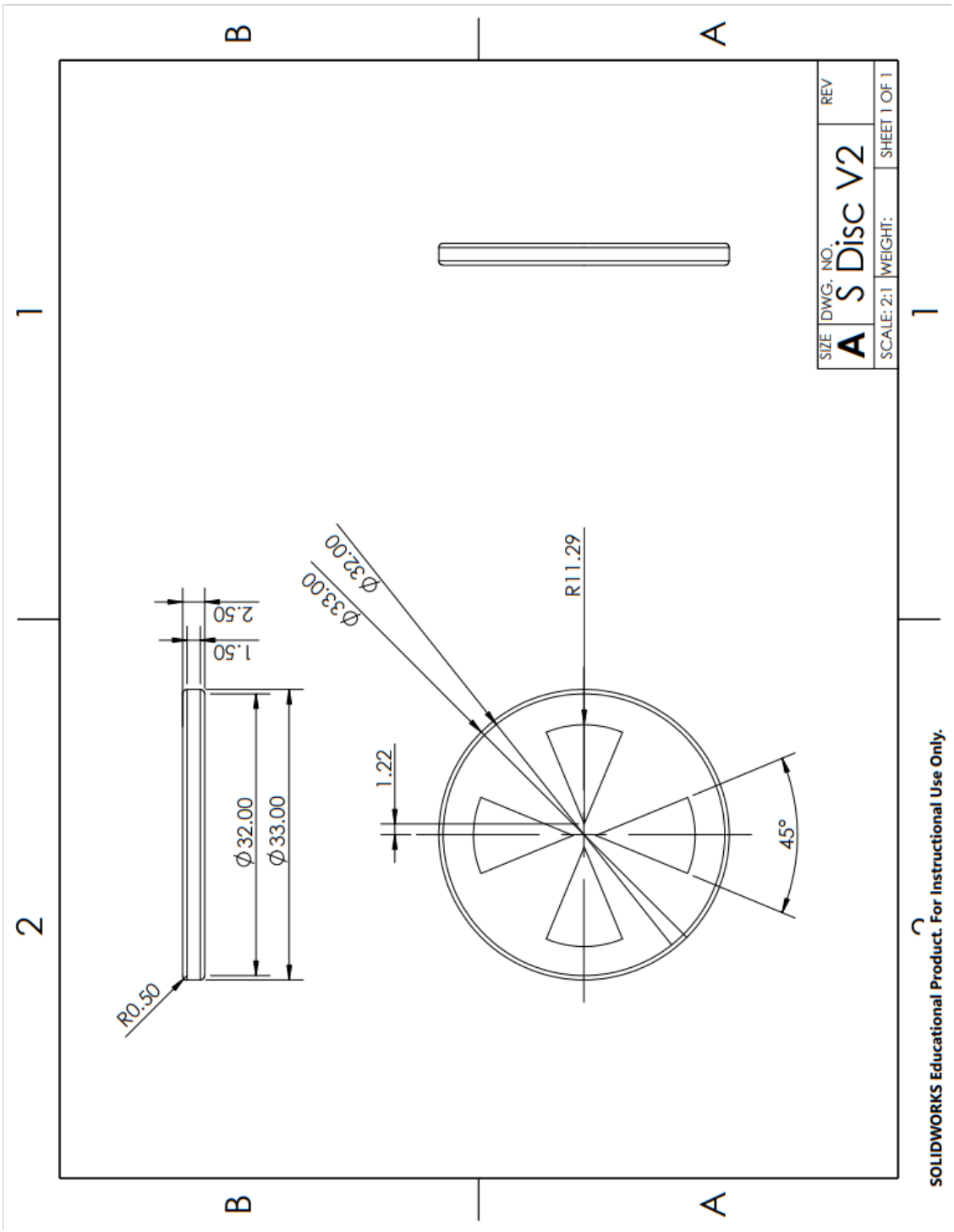
VII. APPENDIX II

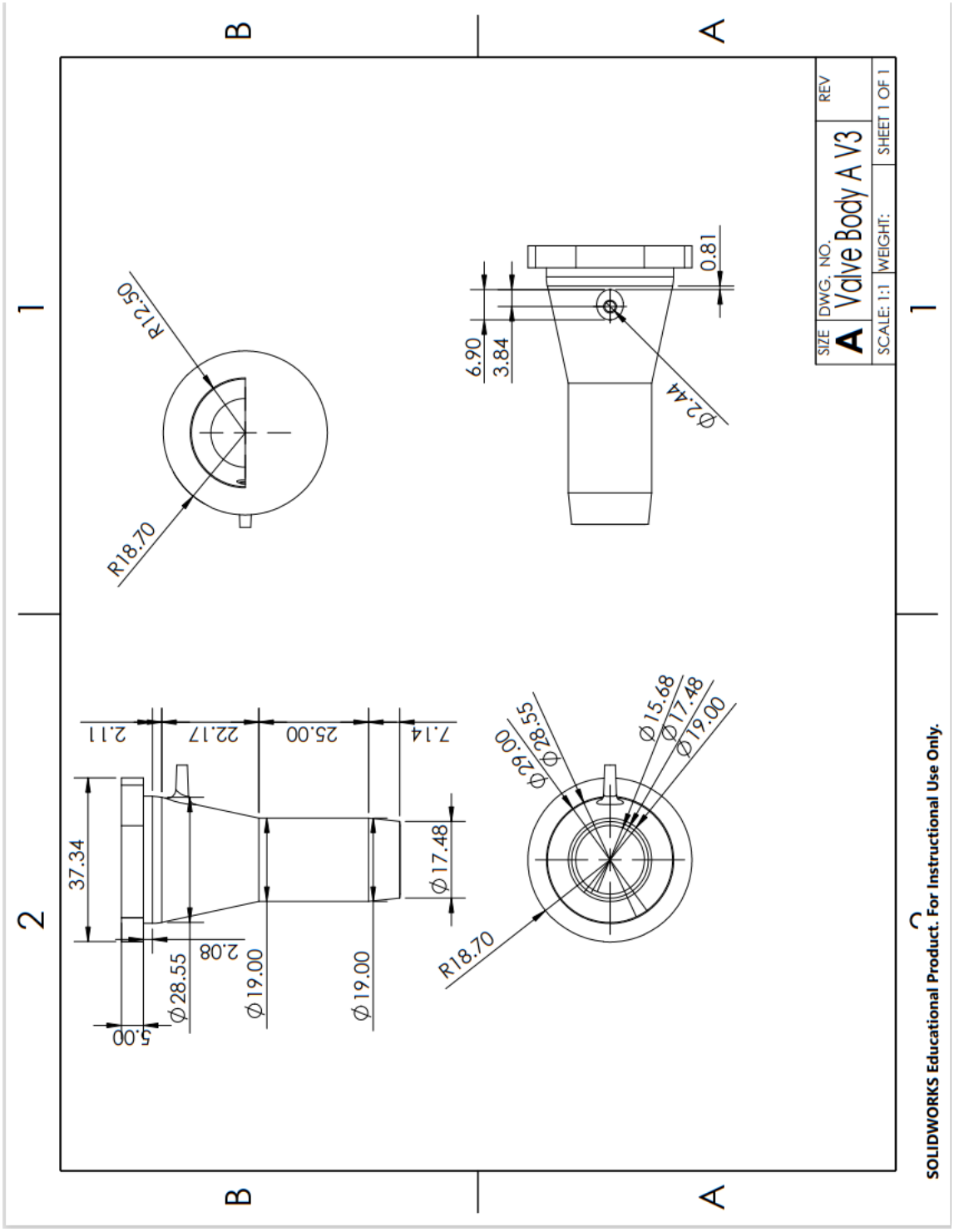
The schematic drawings of the seventeen proportional valve designs tested:

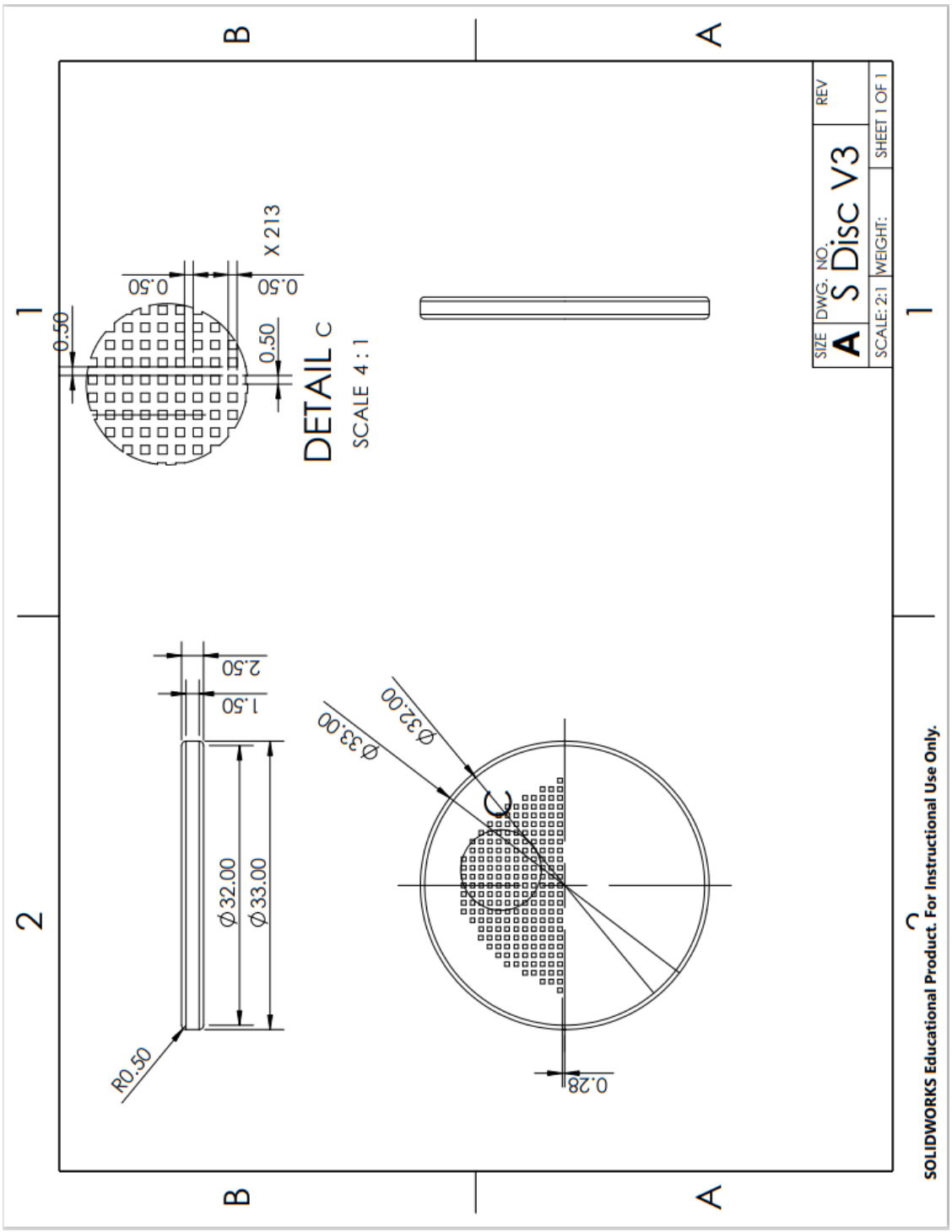


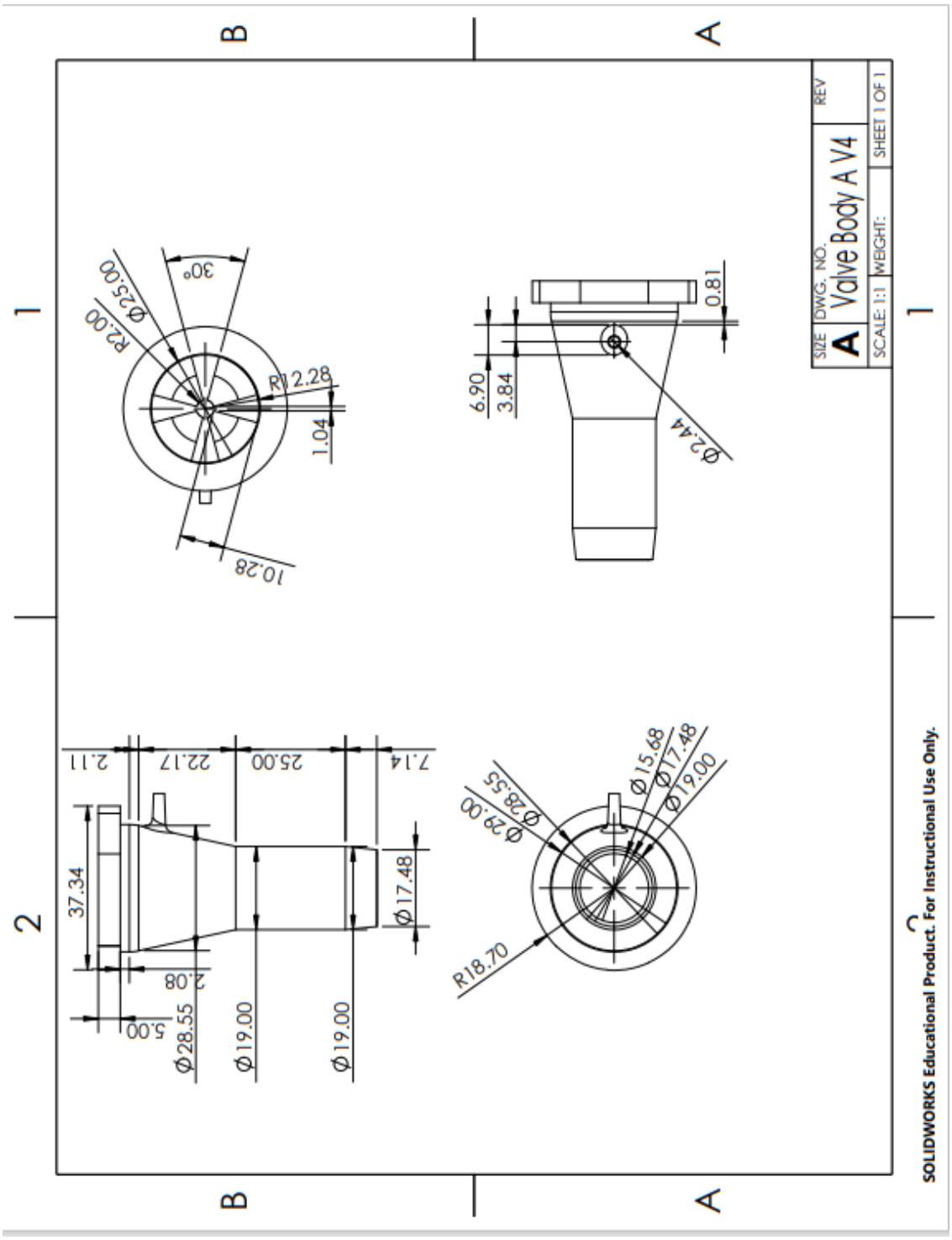

B
A
1
2
SIZE
A
DWG. NO.
S Disc
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø33.00
Ø32.00
11.19
6.83
R3.81
22.5°
R12.50

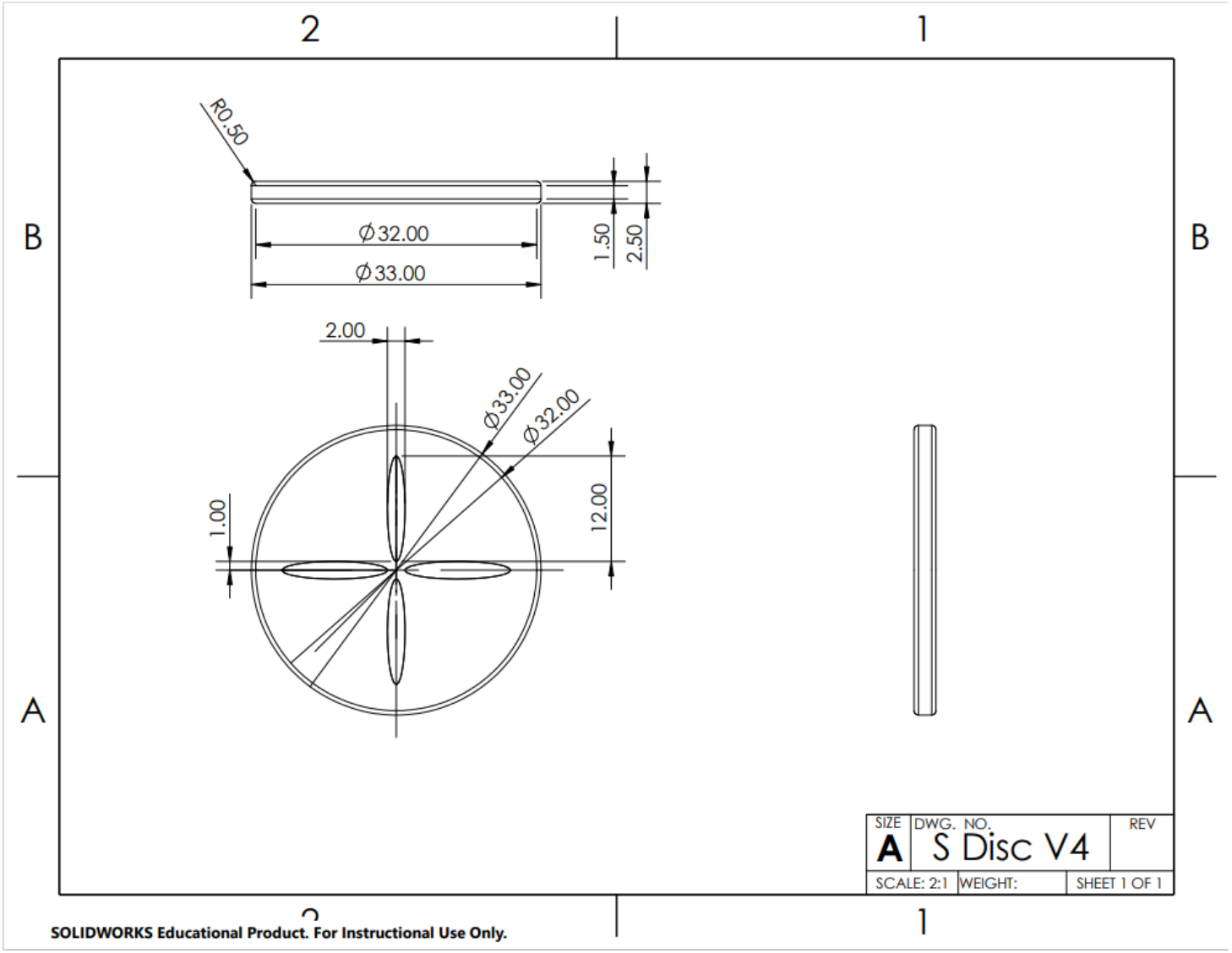

SIZE
A
DWG. NO.
Valve Body A V2
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
R18.70
R12.28
45°
6.90
3.84
0.81
Ø2.44
Ø28.00
Ø28.55
Ø15.68
Ø17.48
Ø19.00
R18.70
2.11
22.17
25.00
7.14
37.34
Ø17.48
Ø19.00
2.08
5.00
Ø28.55

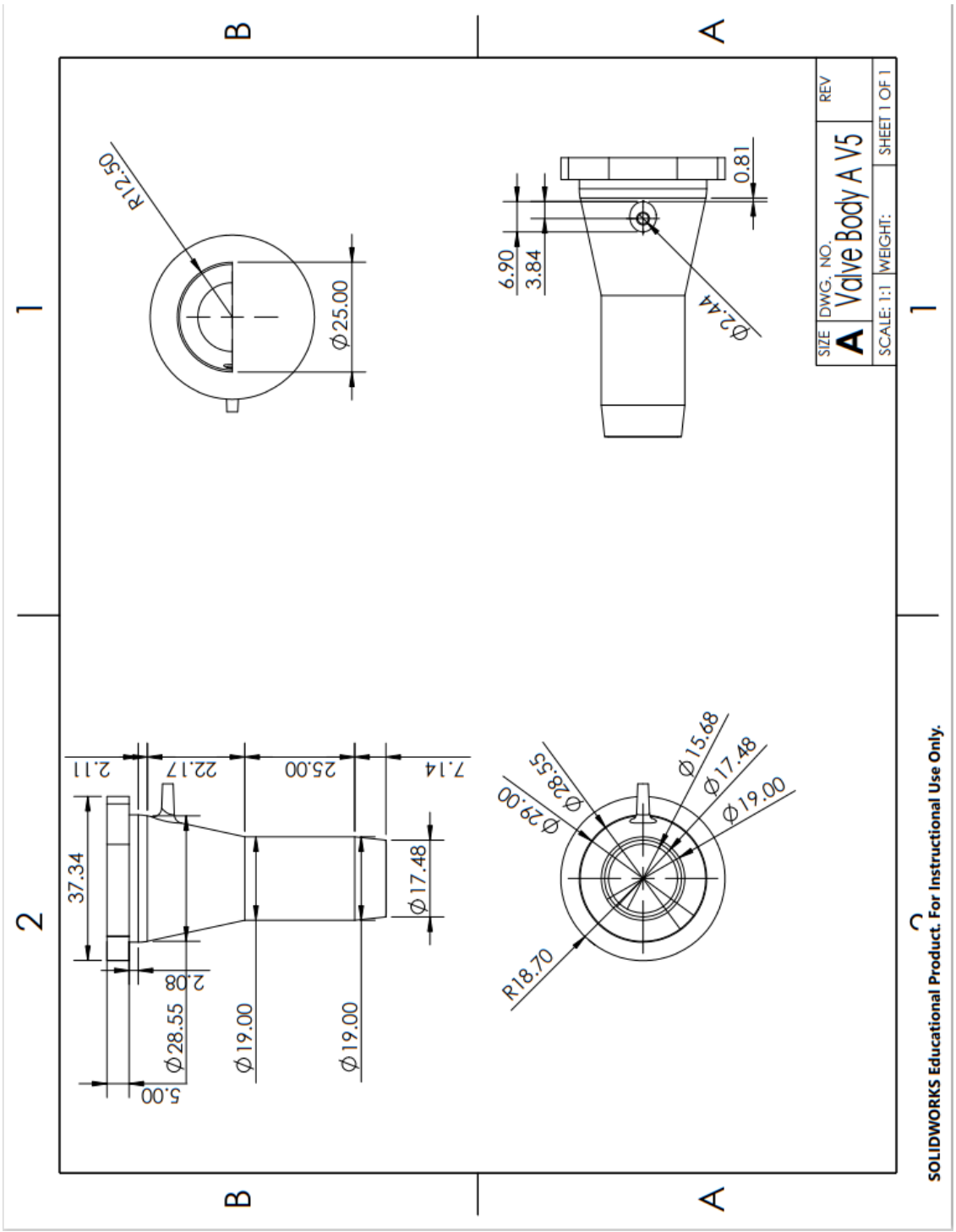

B
A
1
2
SIZE
A
DWG. NO.
S Disc V2
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø33.00
Ø32.00
R11.29
1.22
45°
SOLIDWORKS Educational Product. For Instructional Use Only.

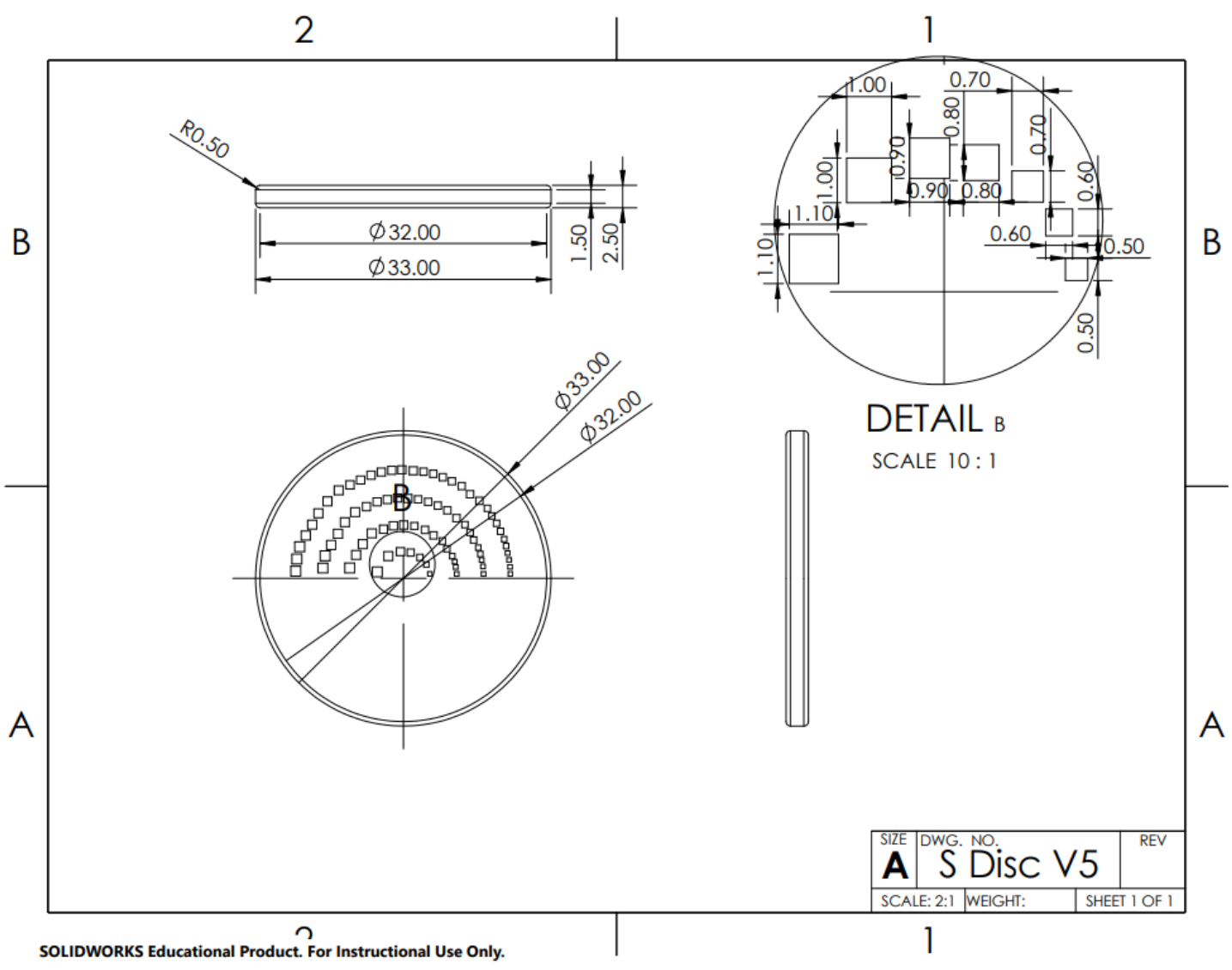

SIZE
A
DWG. NO.
Valve Body A V3
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
R2.50
R18.70
0.81
6.90
3.84
Ø2.44
2.11
22.17
25.00
7.14
37.34
Ø17.48
5.00
Ø28.55
2.08
Ø19.00
Ø19.00
Ø29.00
Ø28.55
Ø15.68
Ø17.48
Ø19.00
R18.70

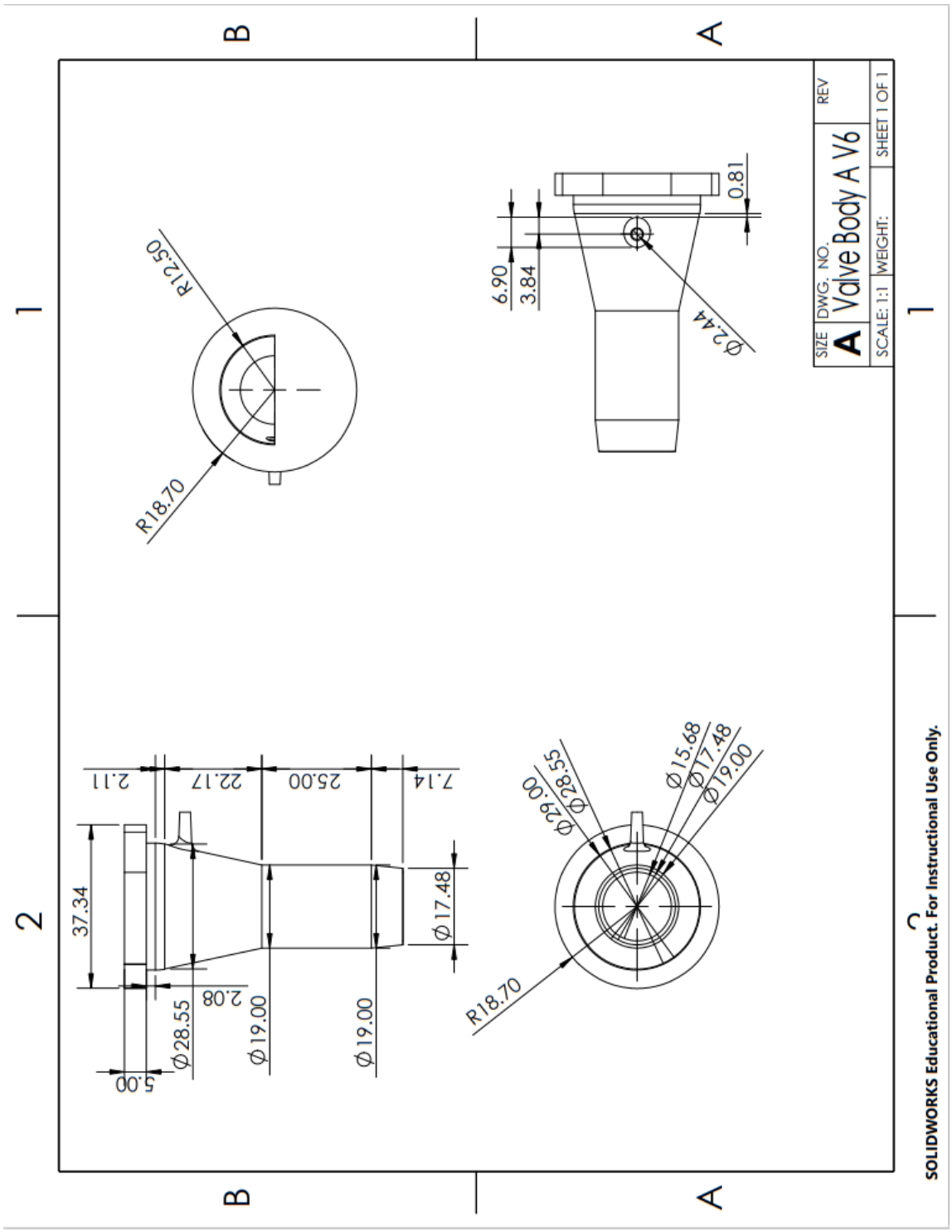

DETAIL C
SCALE 4 : 1
X 213
0.50
0.50
0.50
0.50
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø33.00
Ø32.00
0.28
SIZE
A
DWG. NO.
S Disc V3
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

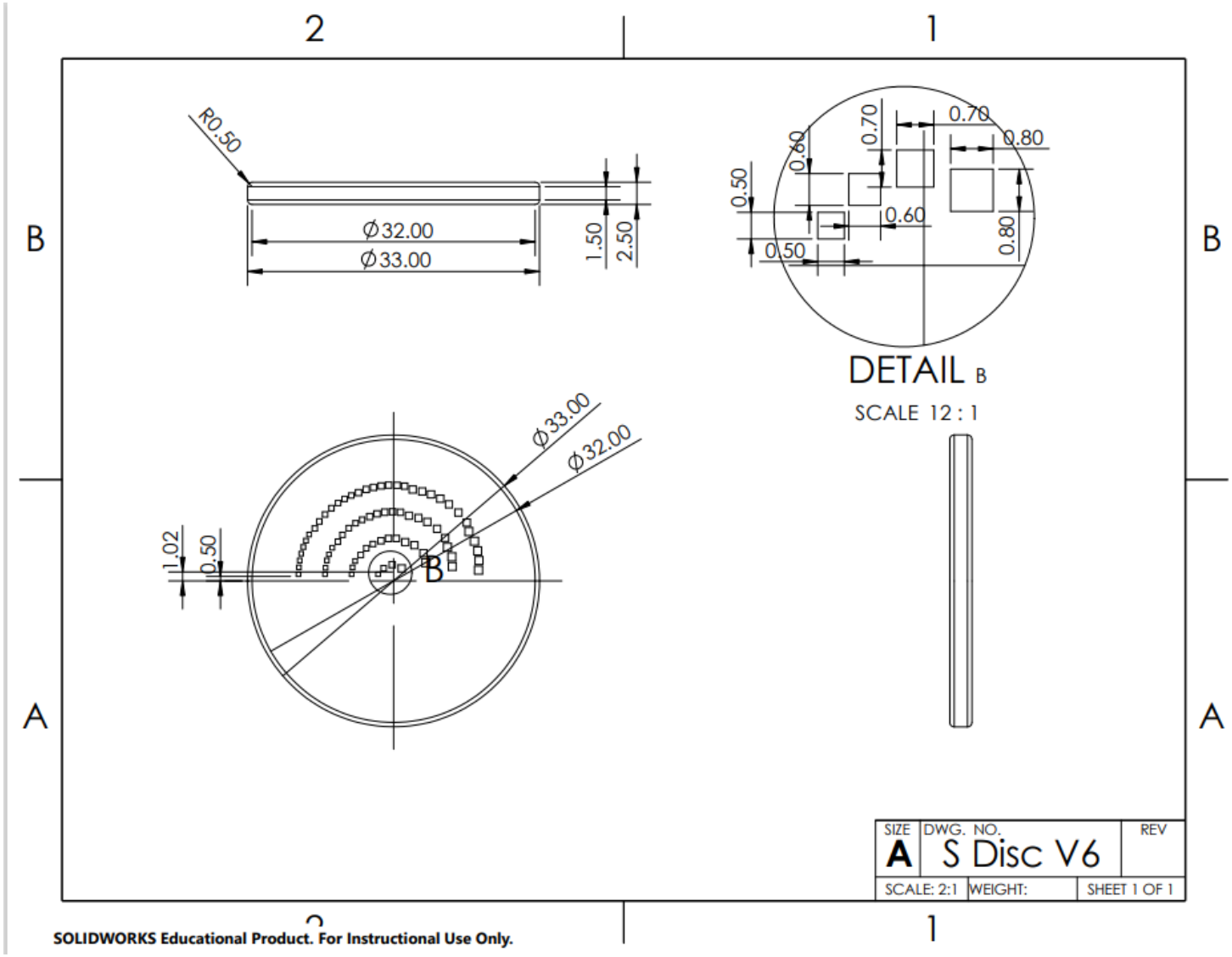

SIZE
A
DWG. NO.
Valve Body A V4
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
30°
R2.00
Ø25.00
R12.28
1.04
10.28
0.81
6.90
3.84
Ø2.44
2.11
22.17
25.00
7.14
37.34
2.08
5.00
Ø28.55
Ø19.00
Ø19.00
Ø17.48
Ø15.68
Ø17.48
Ø19.00
Ø28.00
Ø28.55
R18.70

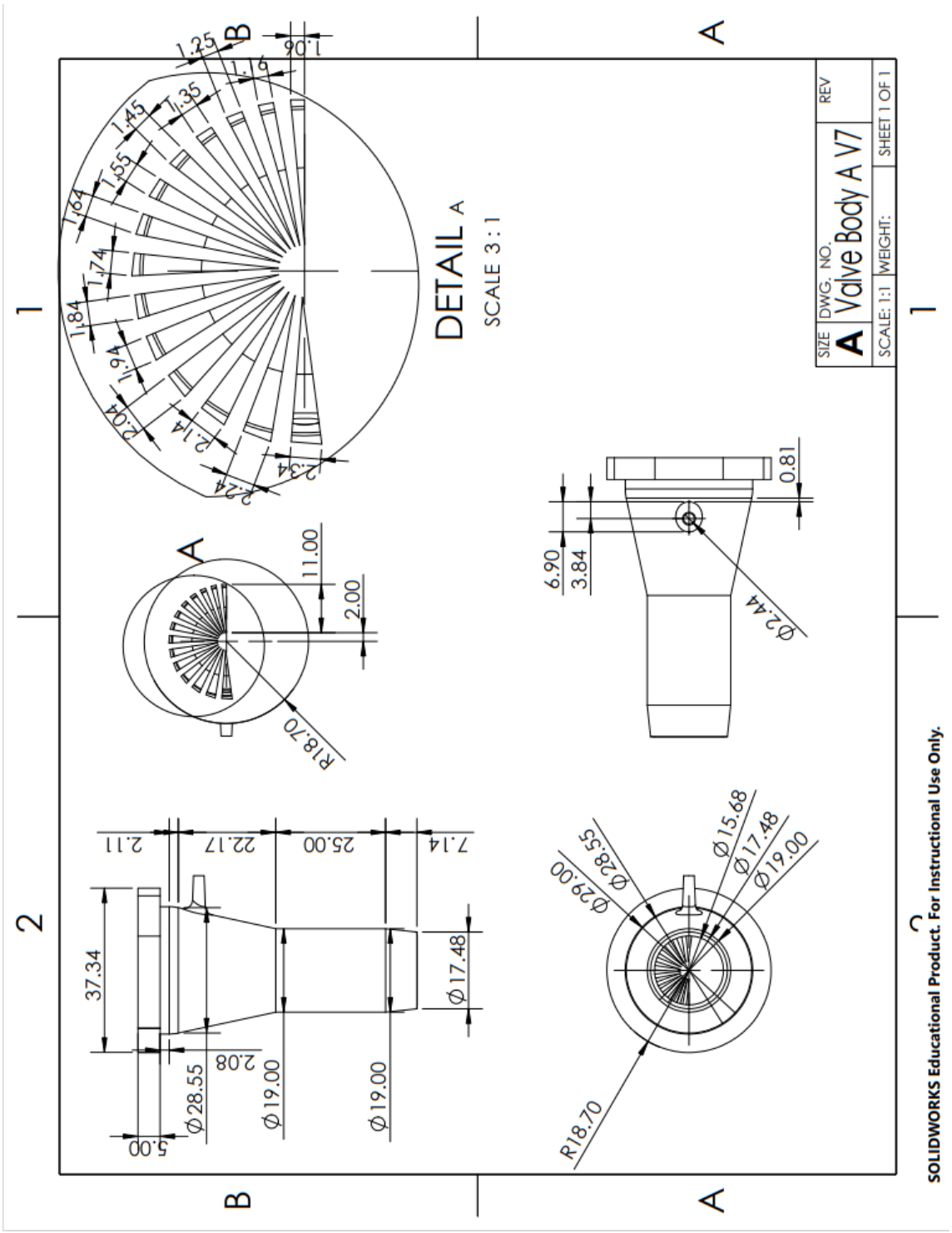

2
1
B
A
R0.50
Ø32.00
Ø33.00
1.50
2.50
2.00
Ø33.00
Ø32.00
1.00
12.00
SIZE
A
DWG. NO.
S Disc V4
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

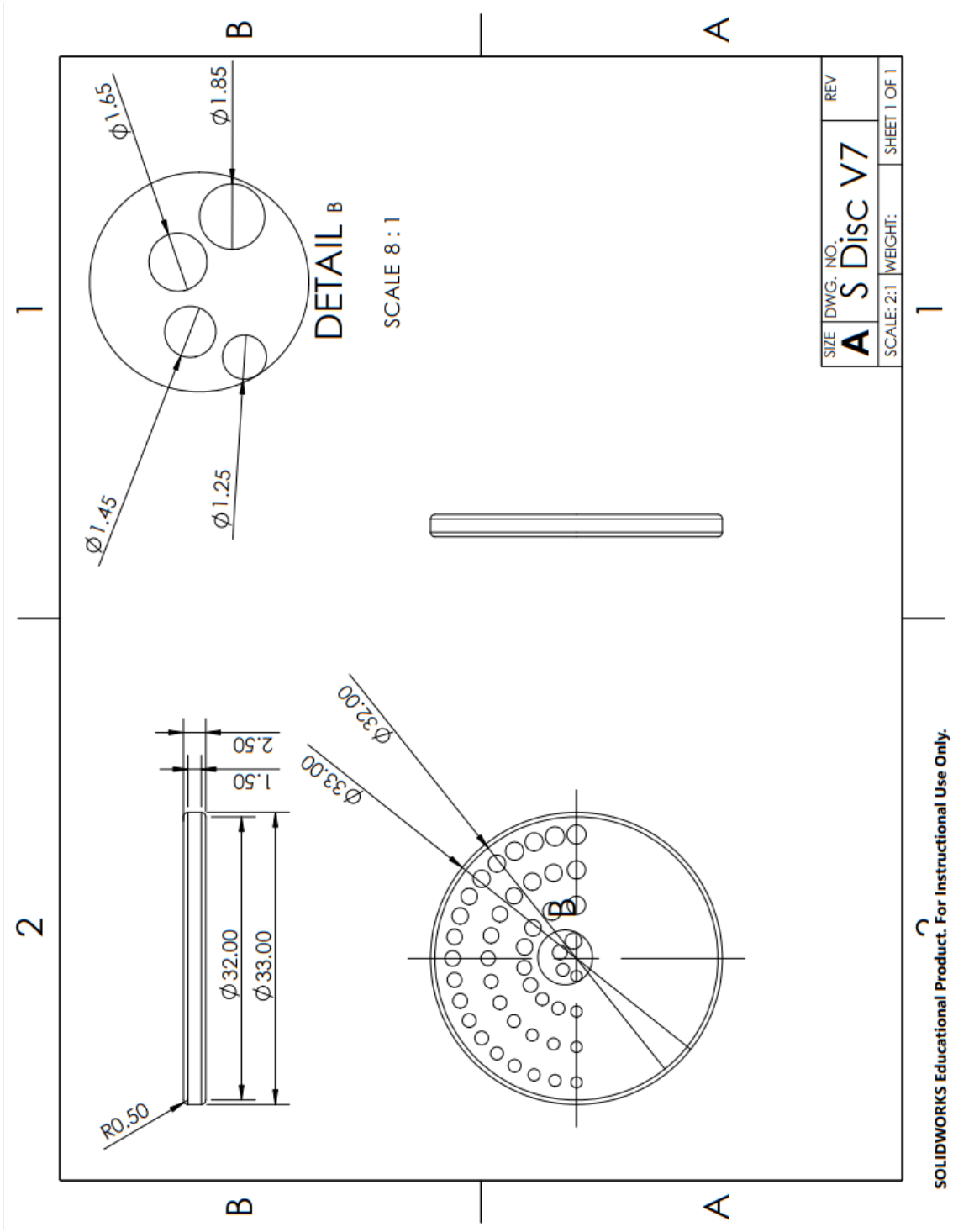

SIZE
A
DWG. NO.
Valve Body A V5
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
R12.50
Ø25.00
6.90
3.84
0.81
Ø2.44
2.11
22.17
25.00
7.14
37.34
2.08
Ø28.55
Ø19.00
Ø19.00
Ø17.48
5.00
Ø15.68
Ø17.48
Ø19.00
Ø28.55
Ø29.00
R18.70

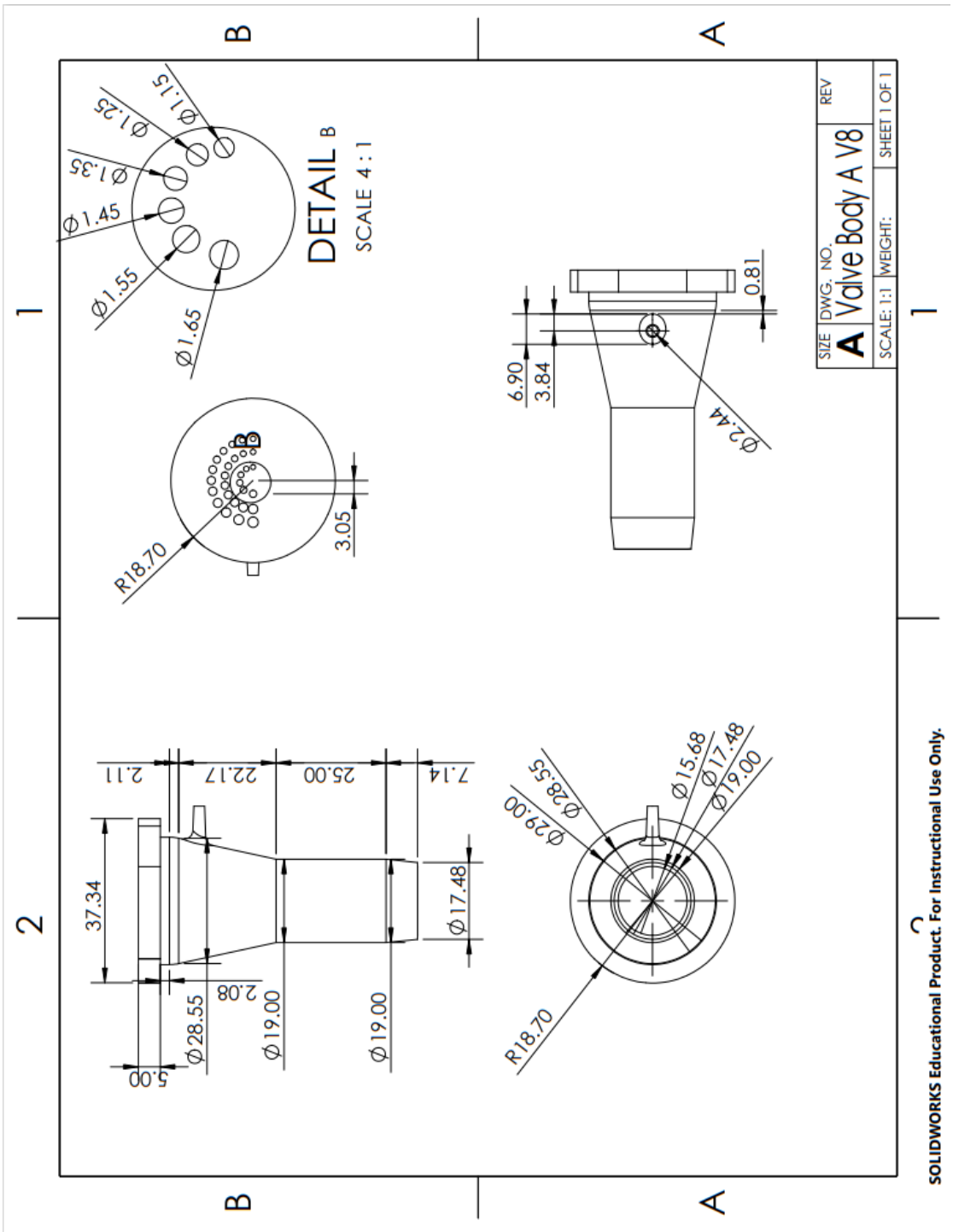

2
1
B
A
R0.50
Ø32.00
Ø33.00
1.50
2.50
1.00
0.70
0.80
0.90
0.70
1.00
0.90
0.80
0.60
1.10
0.60
0.50
1.10
0.50
DETAIL B
SCALE 10 : 1
Ø33.00
Ø32.00
B
SIZE
A
DWG. NO.
S Disc V5
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

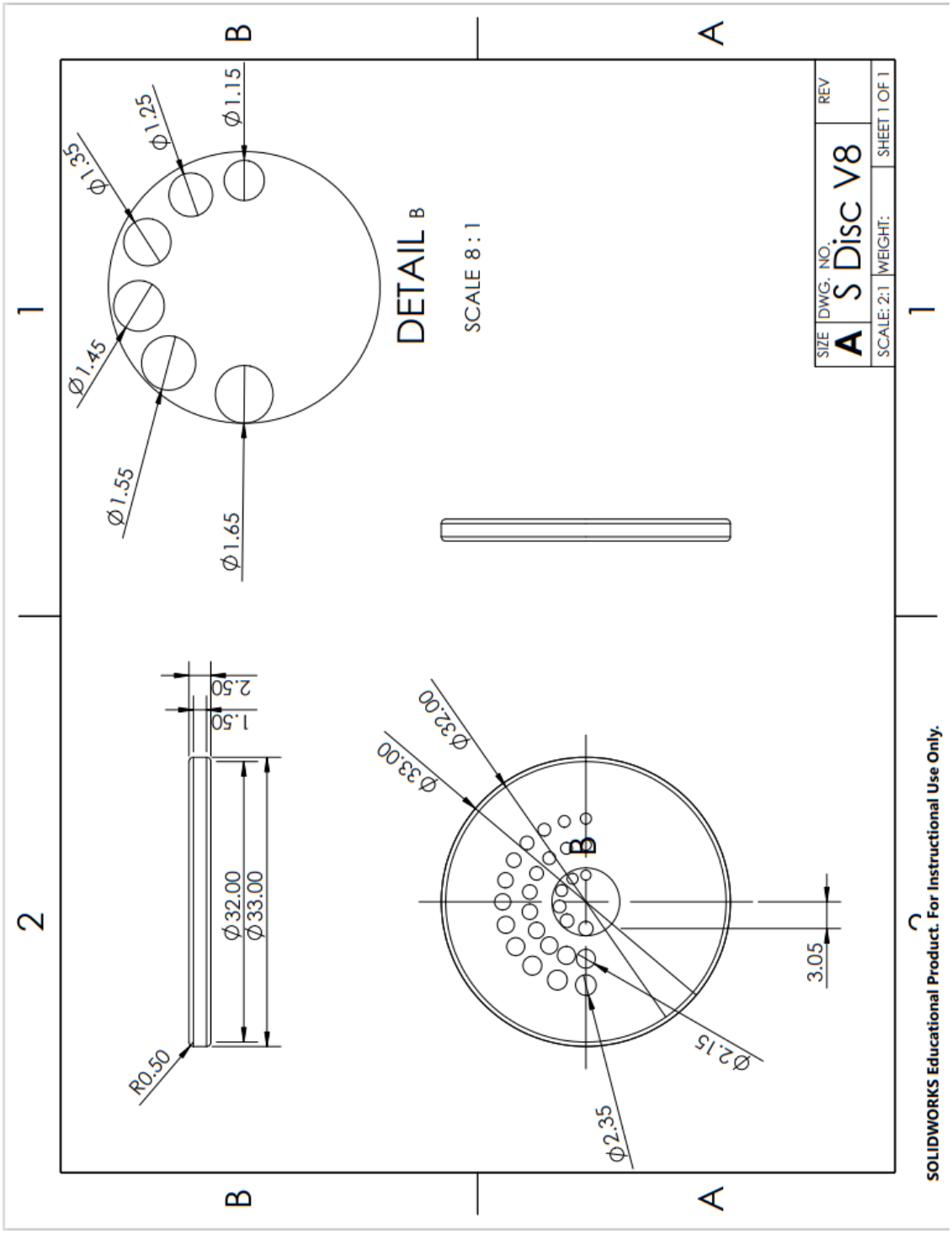

SIZE
A
DWG. NO.
Valve Body A V6
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
R2.50
R18.70
6.90
3.84
0.81
Ø2.44
2.11
22.17
25.00
7.14
37.34
Ø17.48
Ø28.55
2.08
Ø19.00
Ø19.00
5.00
Ø29.00
Ø28.55
Ø15.68
Ø17.48
Ø19.00
R18.70

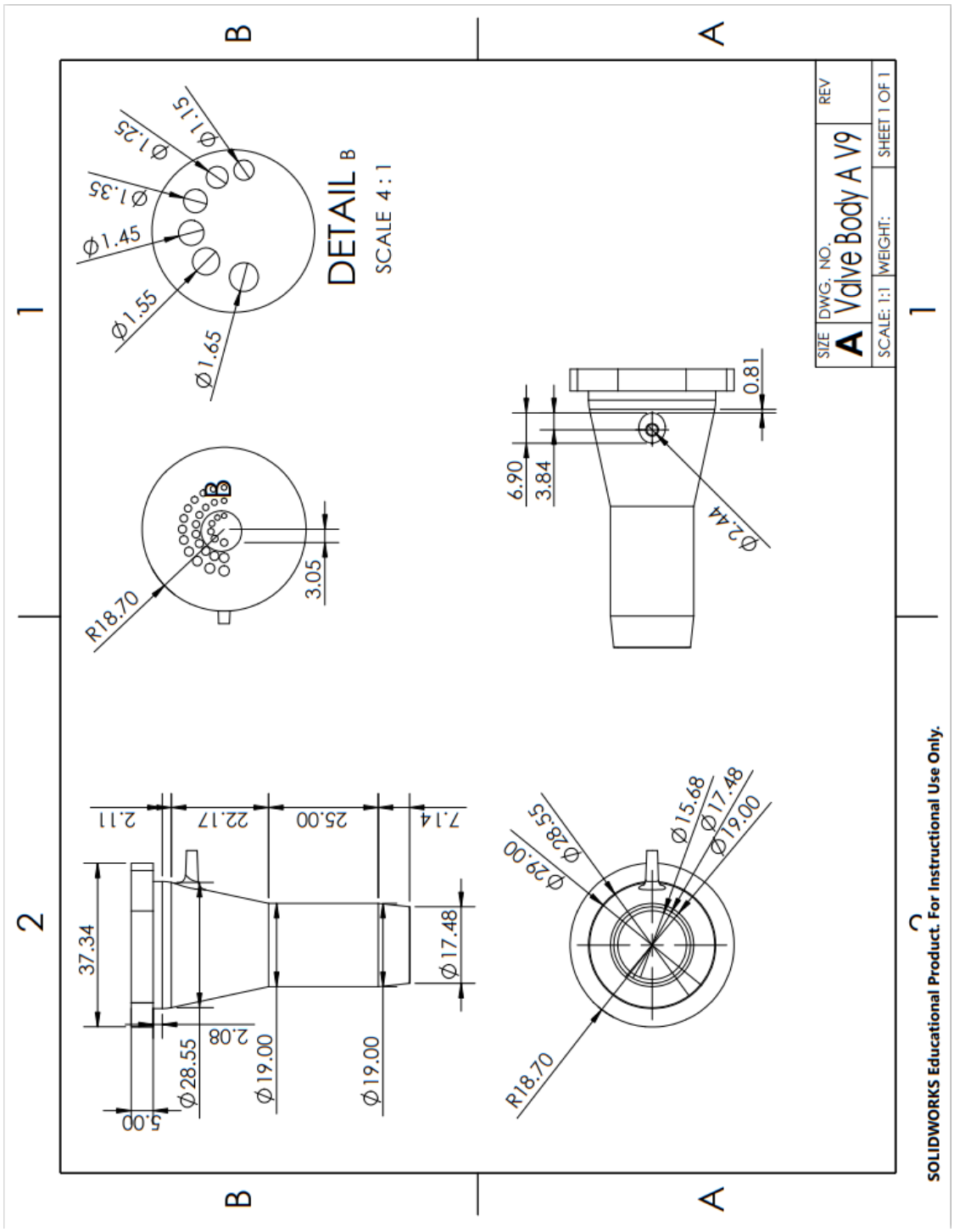

2
1
B
A
R0.50
Ø32.00
Ø33.00
1.50
2.50
0.70
0.70
0.80
0.60
0.50
0.60
0.80
0.50
DETAIL B
SCALE 12 : 1
Ø33.00
Ø32.00
1.02
0.50
B
SIZE
A
DWG. NO.
S Disc V6
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

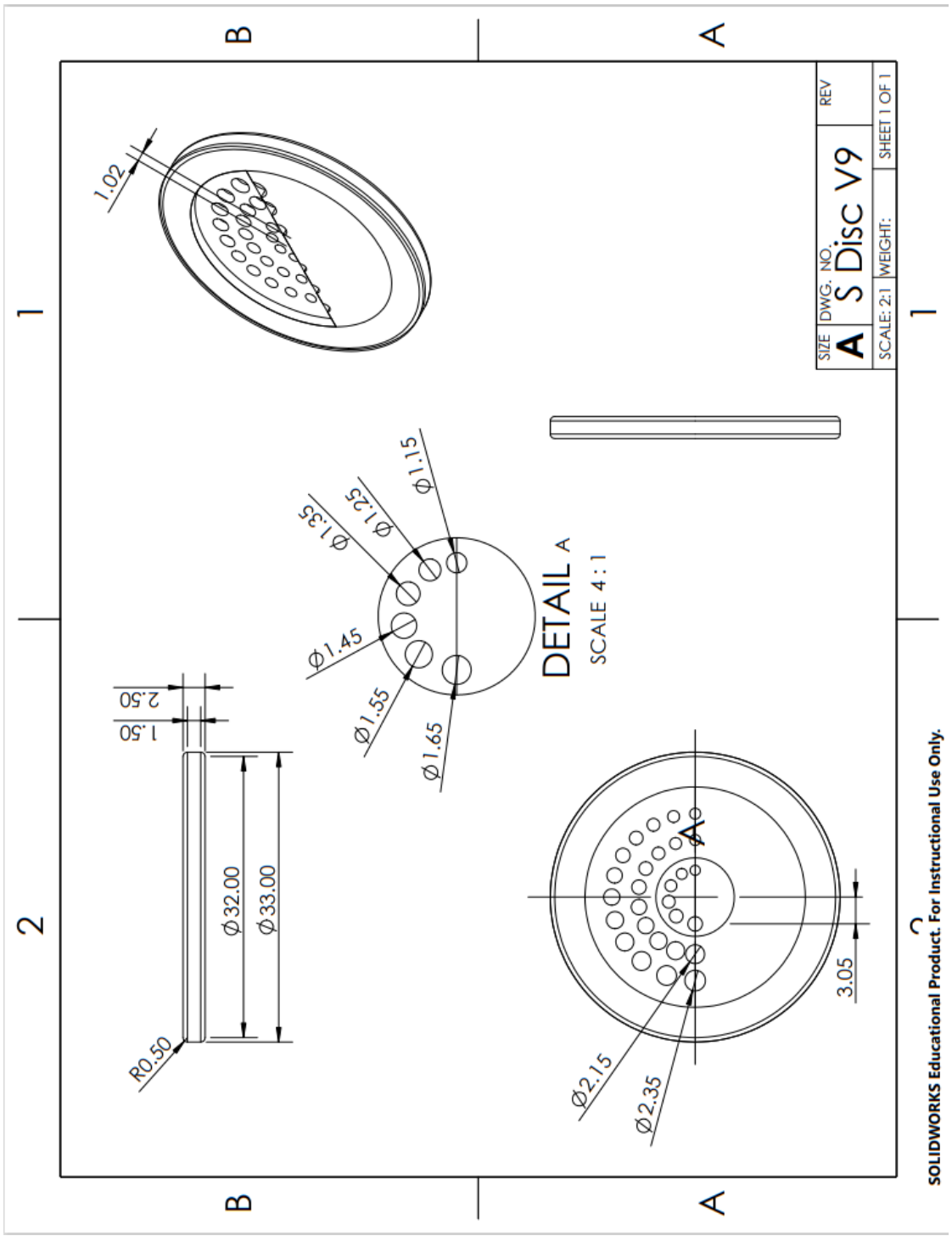

DETAIL A
SCALE 3 : 1
SIZE A
DWG. NO. Valve Body A V7
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

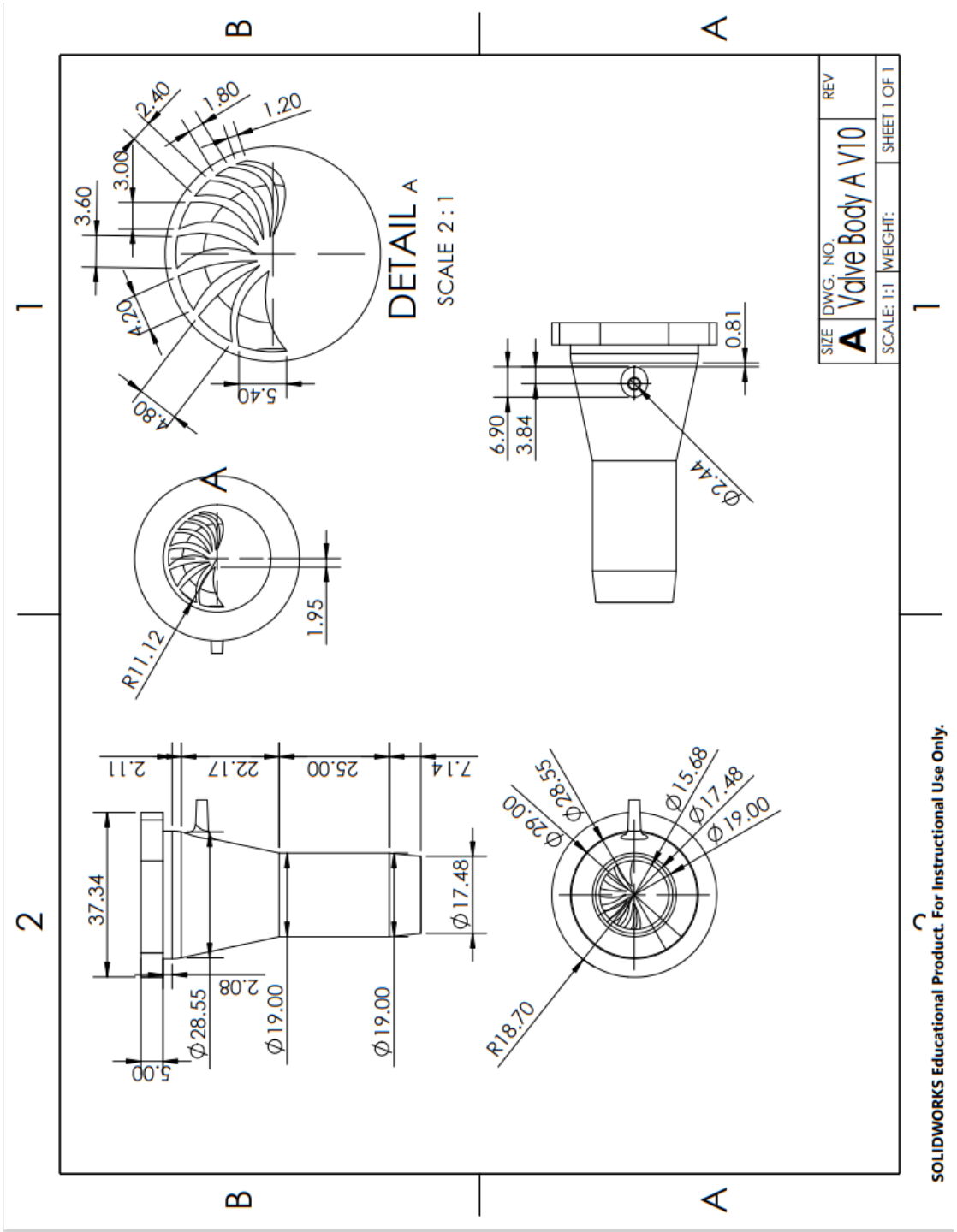

DETAIL B
SCALE 8 : 1
Ø1.65
Ø1.85
Ø1.45
Ø1.25
Ø32.00
Ø33.00
2.50
1.50
Ø32.00
Ø33.00
R0.50
SIZE
A
DWG. NO.
S Disc V7
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

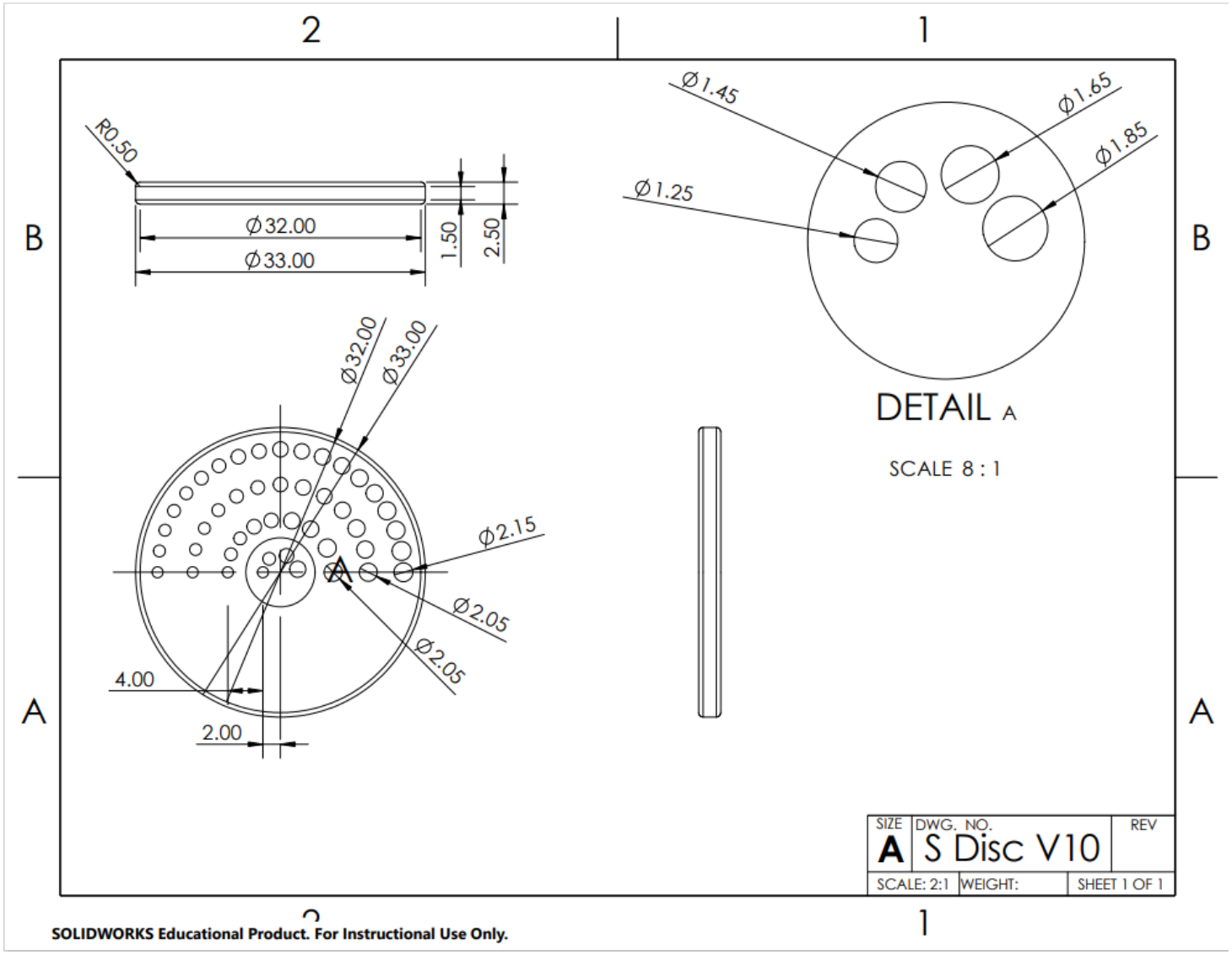

DETAIL B
SCALE 4 : 1
Ø1.15
Ø1.25
Ø1.35
Ø1.45
Ø1.55
Ø1.65
R18.70
3.05
0.81
6.90
3.84
Ø2.44
SIZE
A
DWG. NO.
Valve Body A V8
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
2.11
22.17
25.00
7.14
37.34
Ø17.48
2.08
Ø28.55
Ø19.00
Ø19.00
5.00
Ø15.68
Ø17.48
Ø19.00
Ø28.55
Ø29.00
R18.70
SOLIDWORKS Educational Product. For Instructional Use Only.

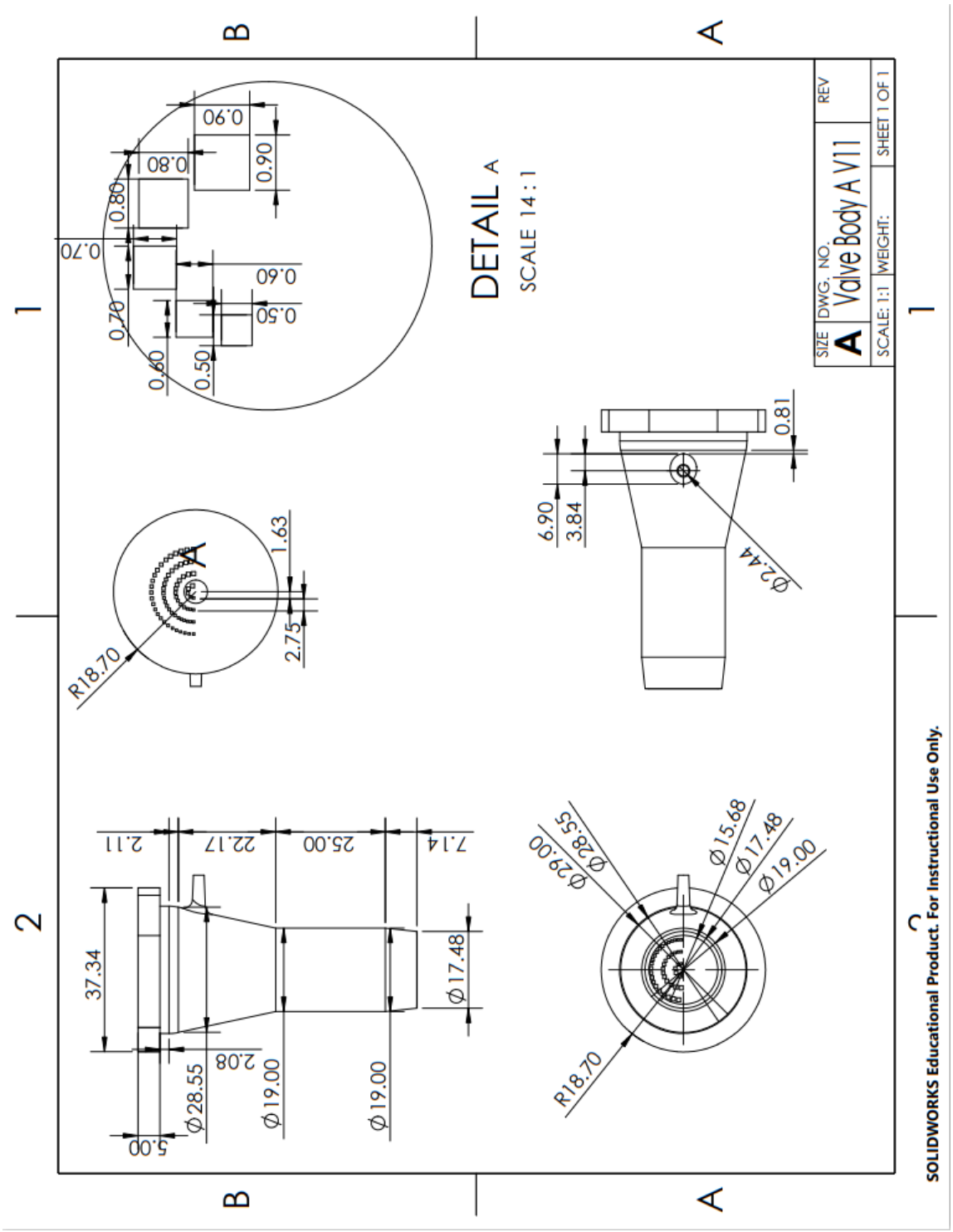

DETAIL B
SCALE 8 : 1
SIZE A
DWG. NO. S Disc V8
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
Ø1.15
Ø1.25
Ø1.35
Ø1.45
Ø1.55
Ø1.65
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø32.00
Ø33.00
Ø2.15
Ø2.35
3.05
SOLIDWORKS Educational Product. For Instructional Use Only.

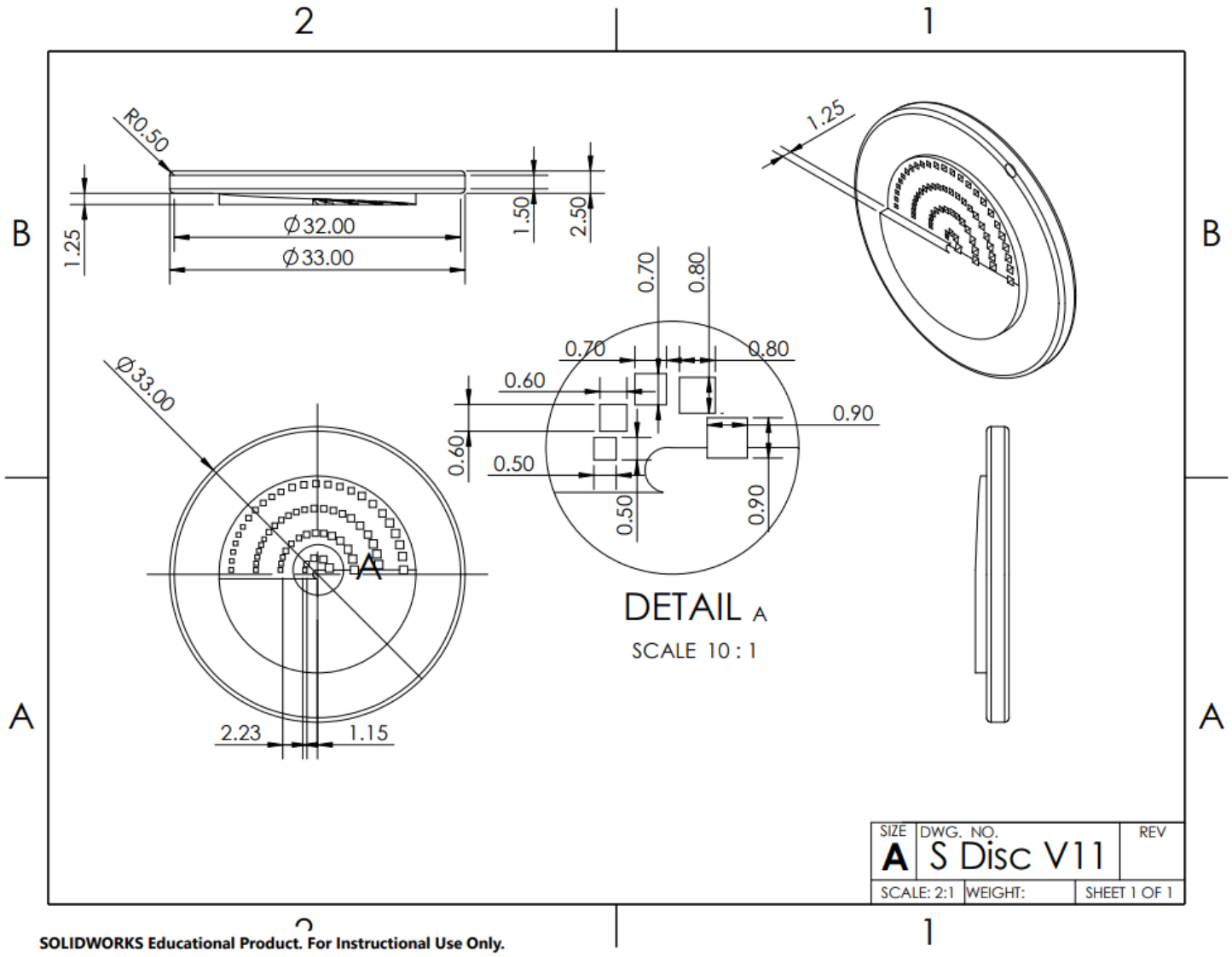

DETAIL B
SCALE 4 : 1
Ø1.15
Ø1.25
Ø1.35
Ø1.45
Ø1.55
Ø1.65
R18.70
3.05
0.81
6.90
3.84
Ø2.44
2.11
22.17
25.00
7.14
37.34
Ø17.48
2.08
Ø28.55
Ø19.00
Ø19.00
5.00
Ø29.00
Ø28.55
Ø15.68
Ø17.48
Ø19.00
R18.70
SIZE
A
DWG. NO.
Valve Body A V9
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

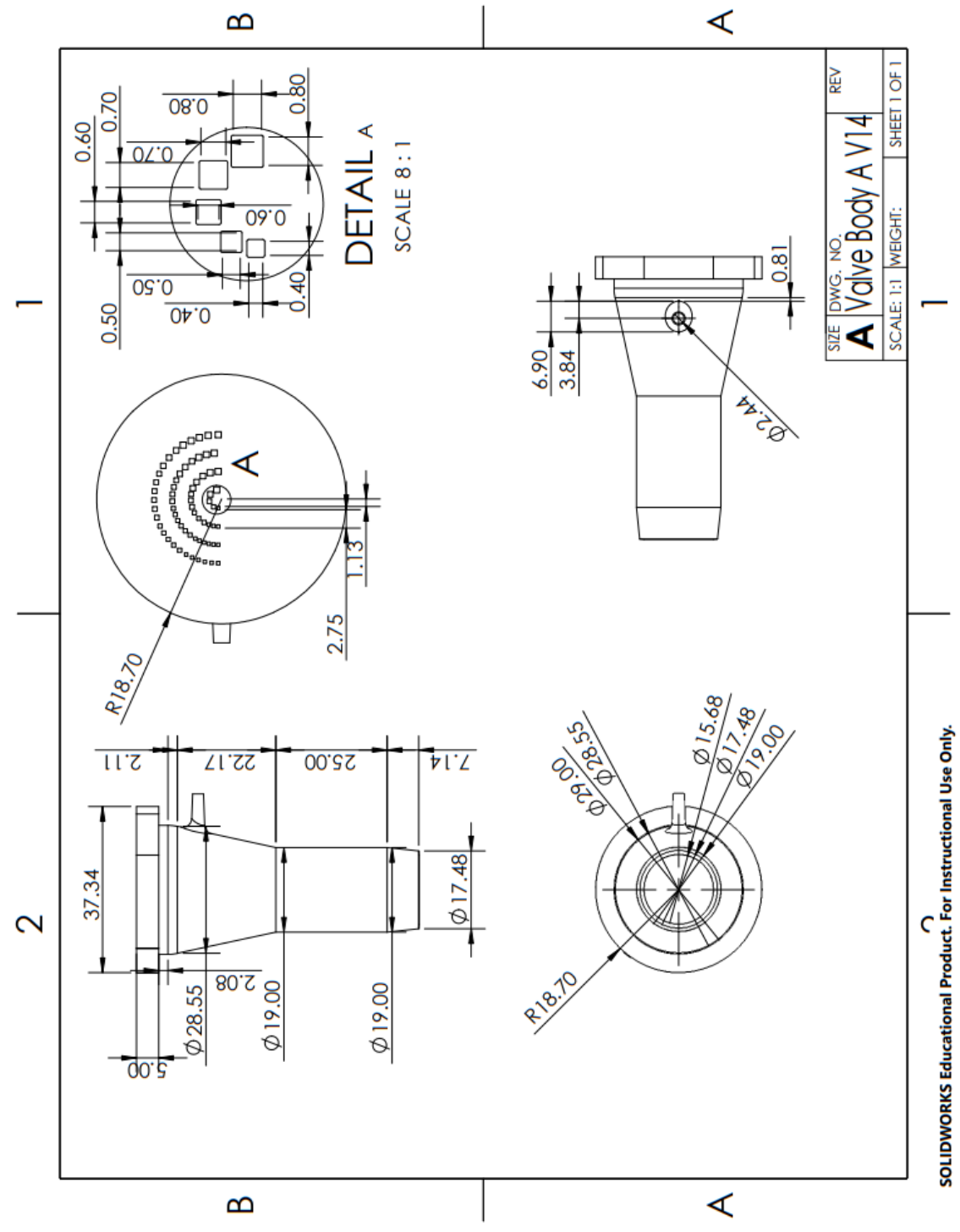

1.02
DETAIL A
SCALE 4 : 1
Ø1.15
Ø1.25
Ø1.35
Ø1.45
Ø1.55
Ø1.65
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø2.15
Ø2.35
3.05
SIZE
A
DWG. NO.
S Disc V9
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

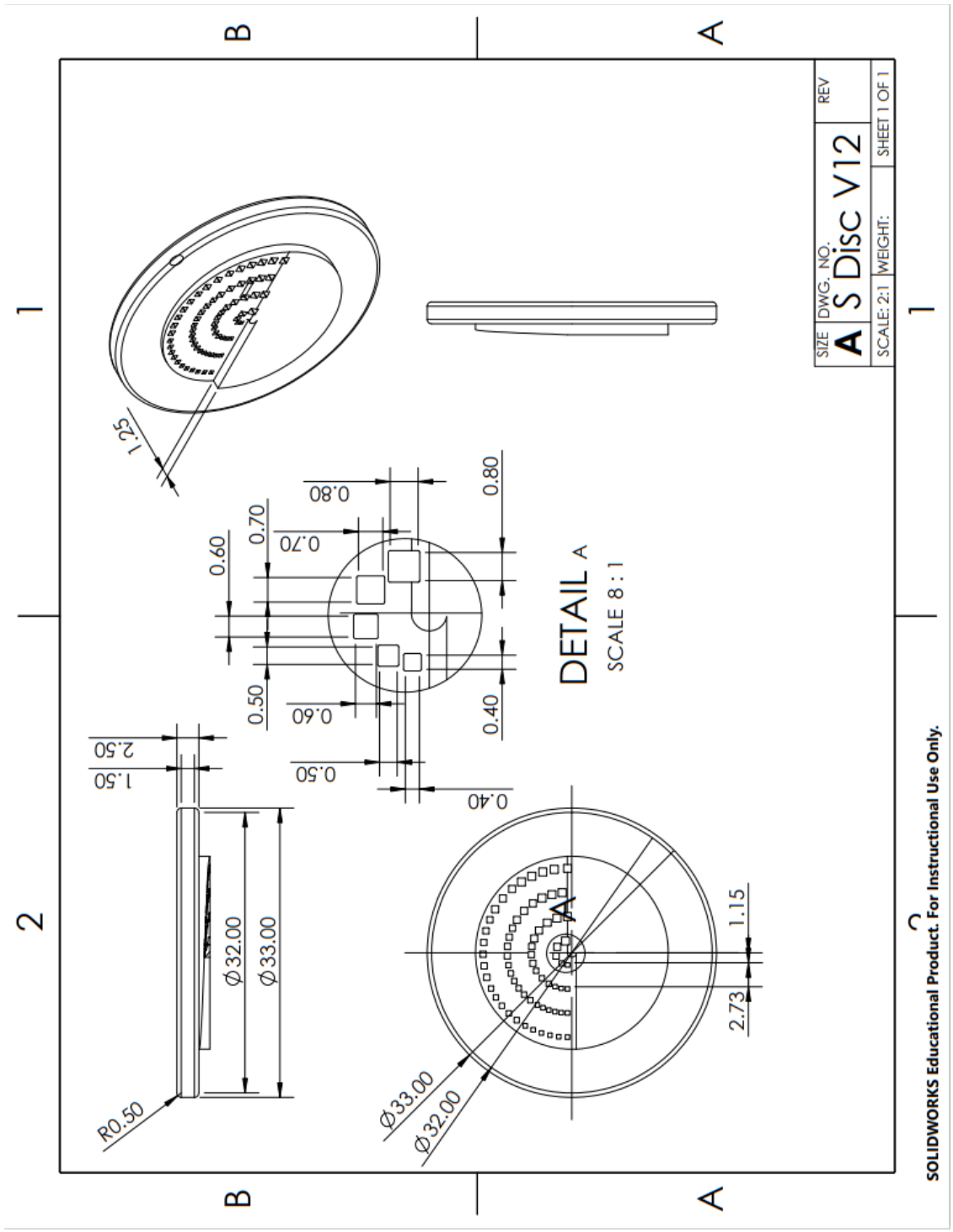

DETAIL A
SCALE 2 : 1
SIZE
A
DWG. NO.
Valve Body A V10
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

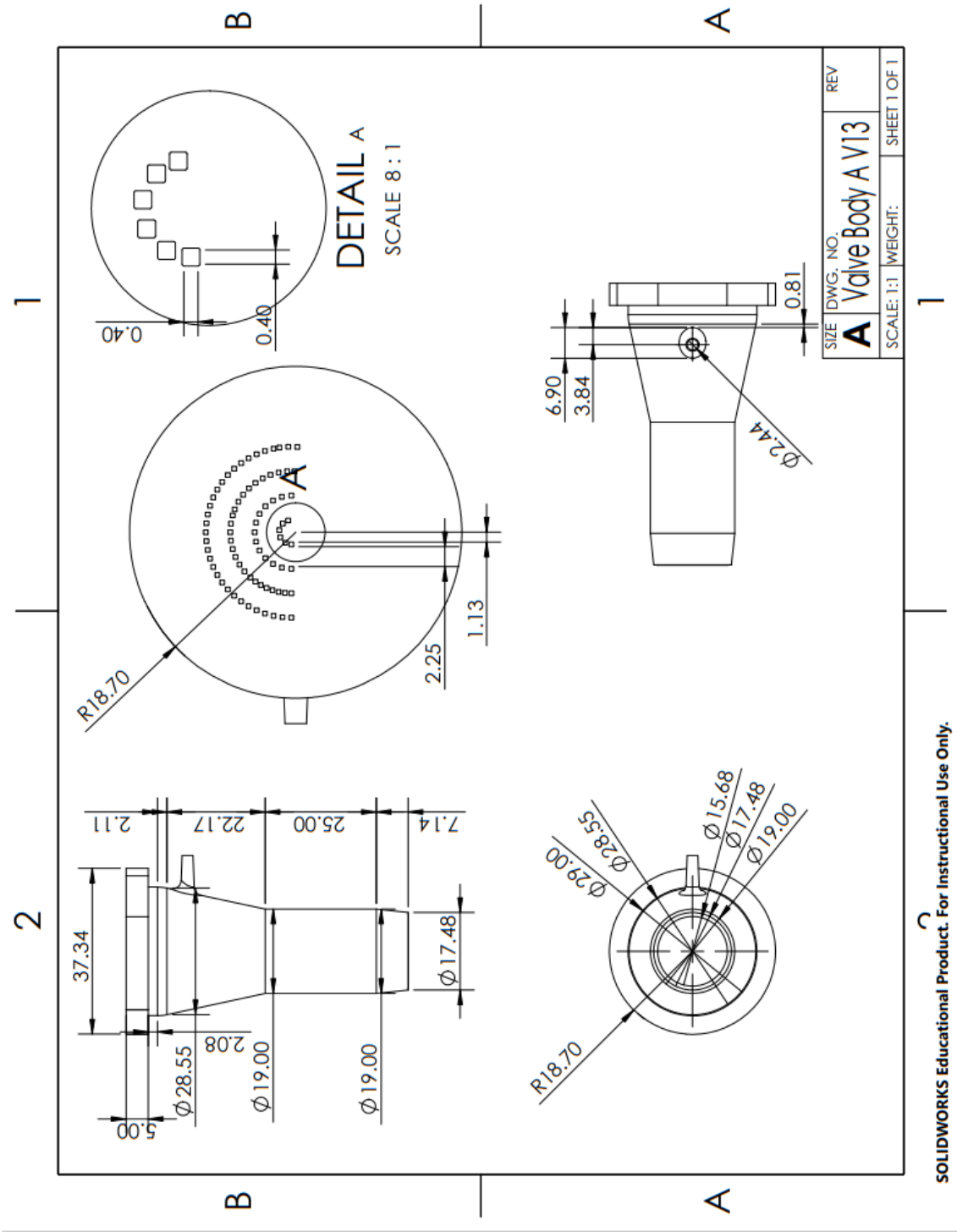

2
1
B
A
R0.50
Ø32.00
Ø33.00
1.50
2.50
Ø1.45
Ø1.65
Ø1.85
Ø1.25
DETAIL A
SCALE 8 : 1
Ø32.00
Ø33.00
Ø2.15
Ø2.05
Ø2.05
4.00
2.00
SIZE
A
DWG. NO.
S Disc V10
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

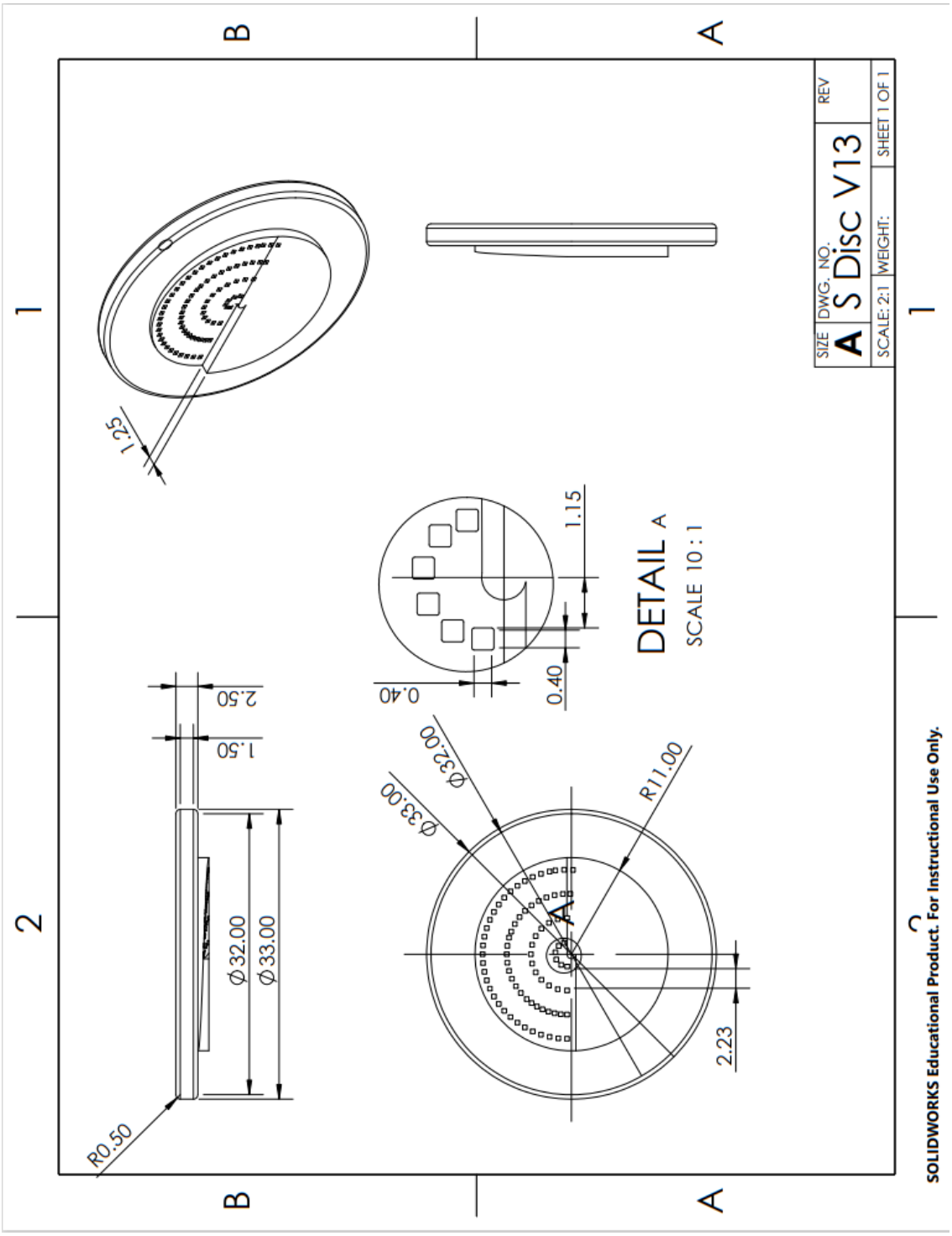

DETAIL A
SCALE 14 : 1
SIZE
A
DWG. NO.
Valve Body A V11
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
R18.70
2.75
1.63
6.90
3.84
0.81
Ø2.44
Ø29.00
Ø28.55
Ø15.68
Ø17.48
Ø19.00
37.34
2.11
22.17
25.00
7.14
Ø17.48
Ø19.00
Ø19.00
Ø28.55
2.08
5.00
0.90
0.80
0.70
0.60
0.50

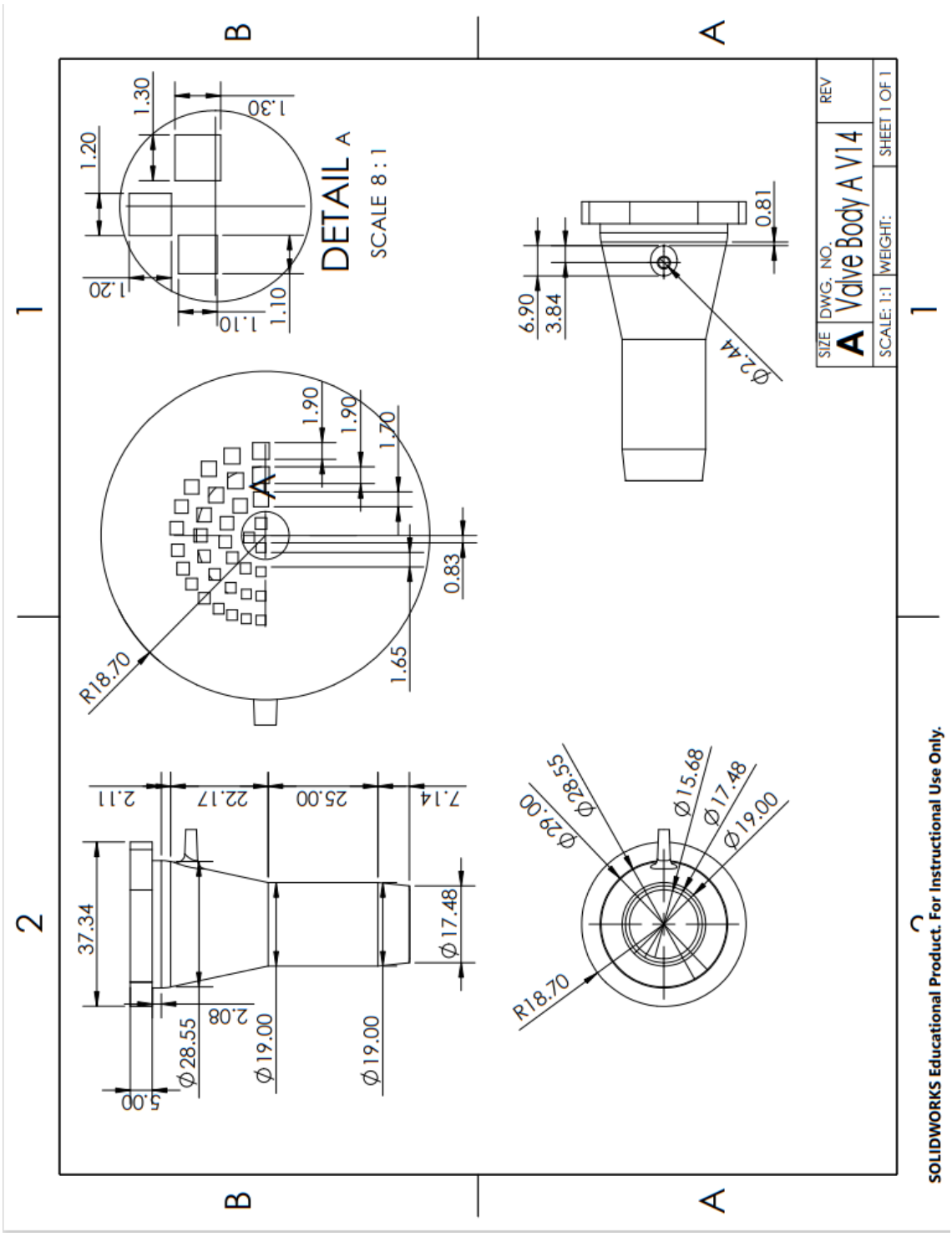

2
1
B
A
R0.50
1.25
Ø32.00
Ø33.00
1.50
2.50
1.25
0.70
0.80
0.70
0.80
0.60
0.90
0.60
0.50
0.50
0.90
Ø33.00
A
2.23
1.15
DETAIL A
SCALE 10 : 1
SIZE
A
DWG. NO.
S Disc V11
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

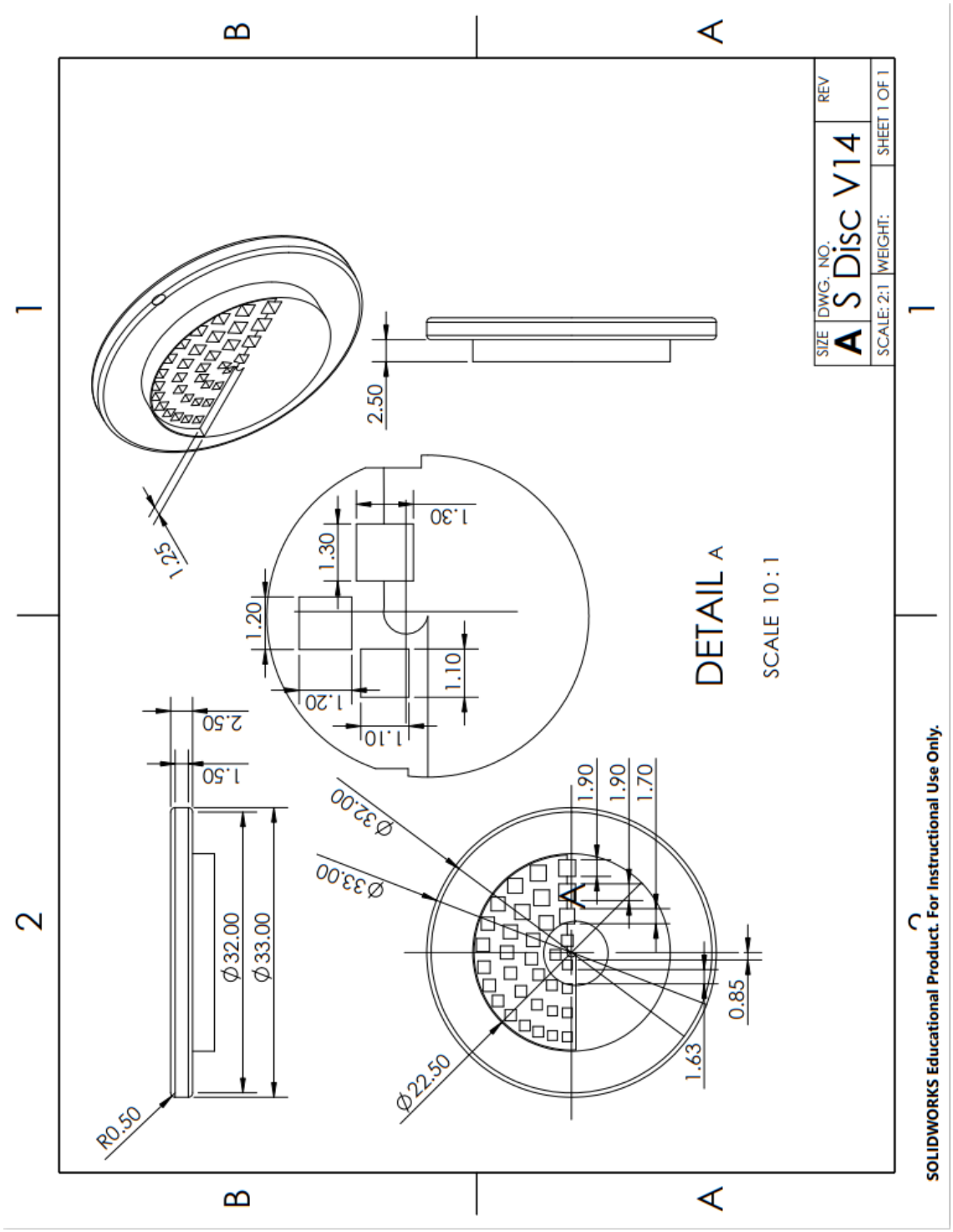

DETAIL A
SCALE 8 : 1
SIZE A
DWG. NO. Valve Body A V14
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

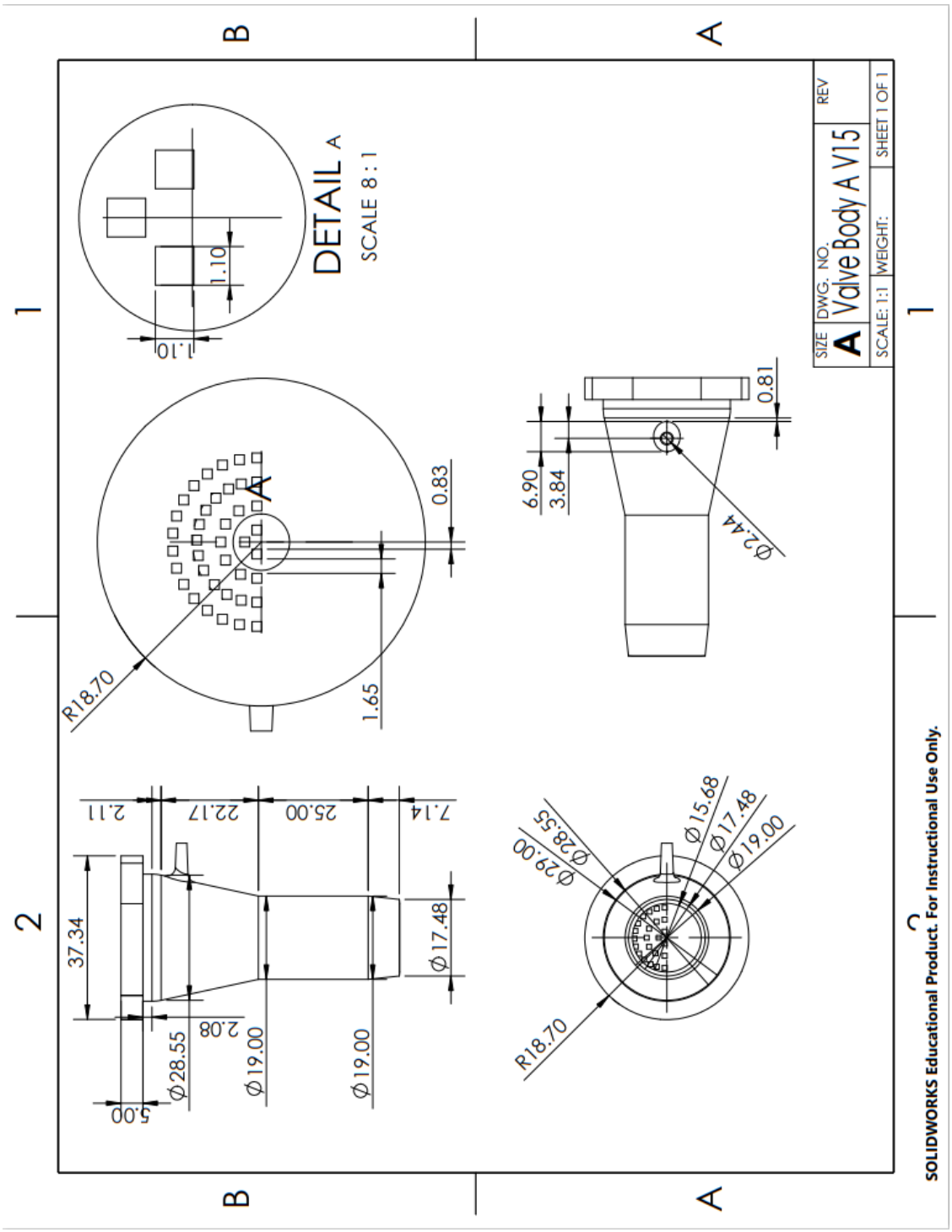

SIZE
A
DWG. NO.
S Disc V12
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
DETAIL A
SCALE 8 : 1
1.25
0.80
0.70
0.60
0.50
0.40
1.50
2.50
Ø32.00
Ø33.00
R0.50
1.15
2.73
SOLIDWORKS Educational Product. For Instructional Use Only.

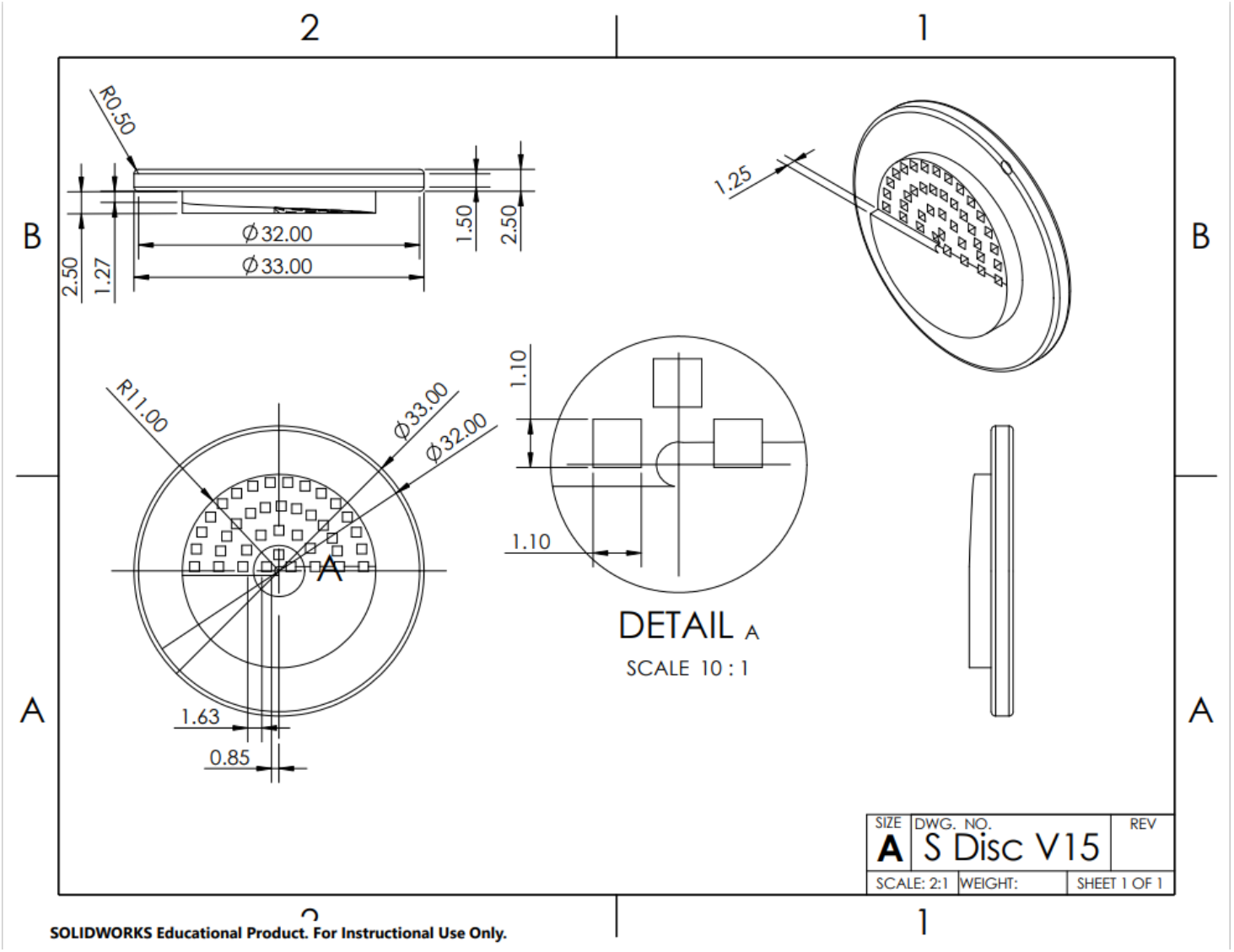

DETAIL A
SCALE 8 : 1
0.40
0.40
R18.70
2.25
1.13
0.81
6.90
3.84
Ø2.44
Ø15.68
Ø17.48
Ø19.00
Ø28.55
Ø29.00
R18.70
37.34
2.11
22.17
25.00
7.14
Ø17.48
Ø28.55
2.08
Ø19.00
Ø19.00
5.00
SIZE
A
DWG. NO.
Valve Body A V13
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

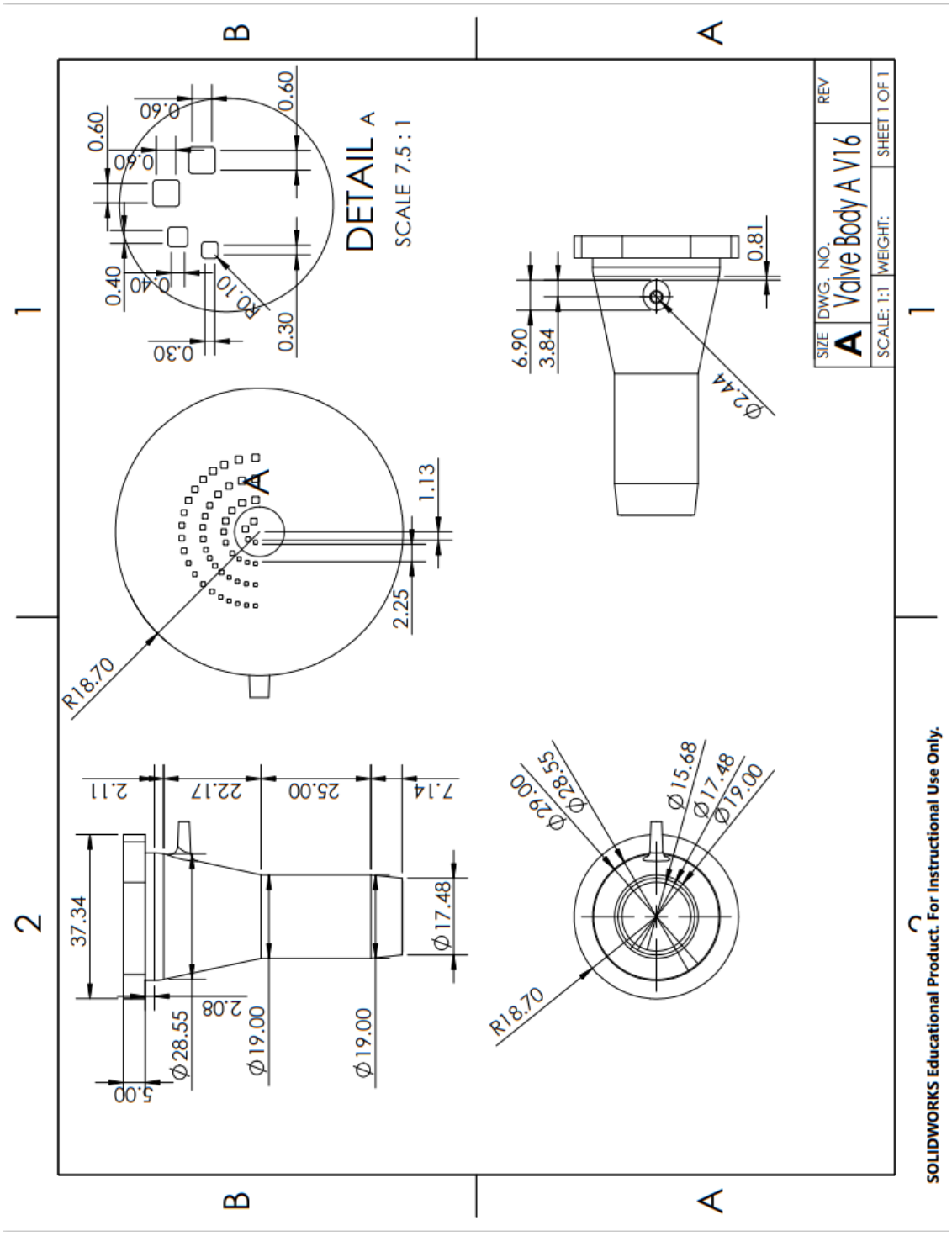

B
A
1
2
SIZE
A
DWG. NO.
S Disc V13
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
1.25
DETAIL A
SCALE 10 : 1
1.15
0.40
0.40
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø32.00
Ø33.00
R11.00
2.23
SOLIDWORKS Educational Product. For Instructional Use Only.

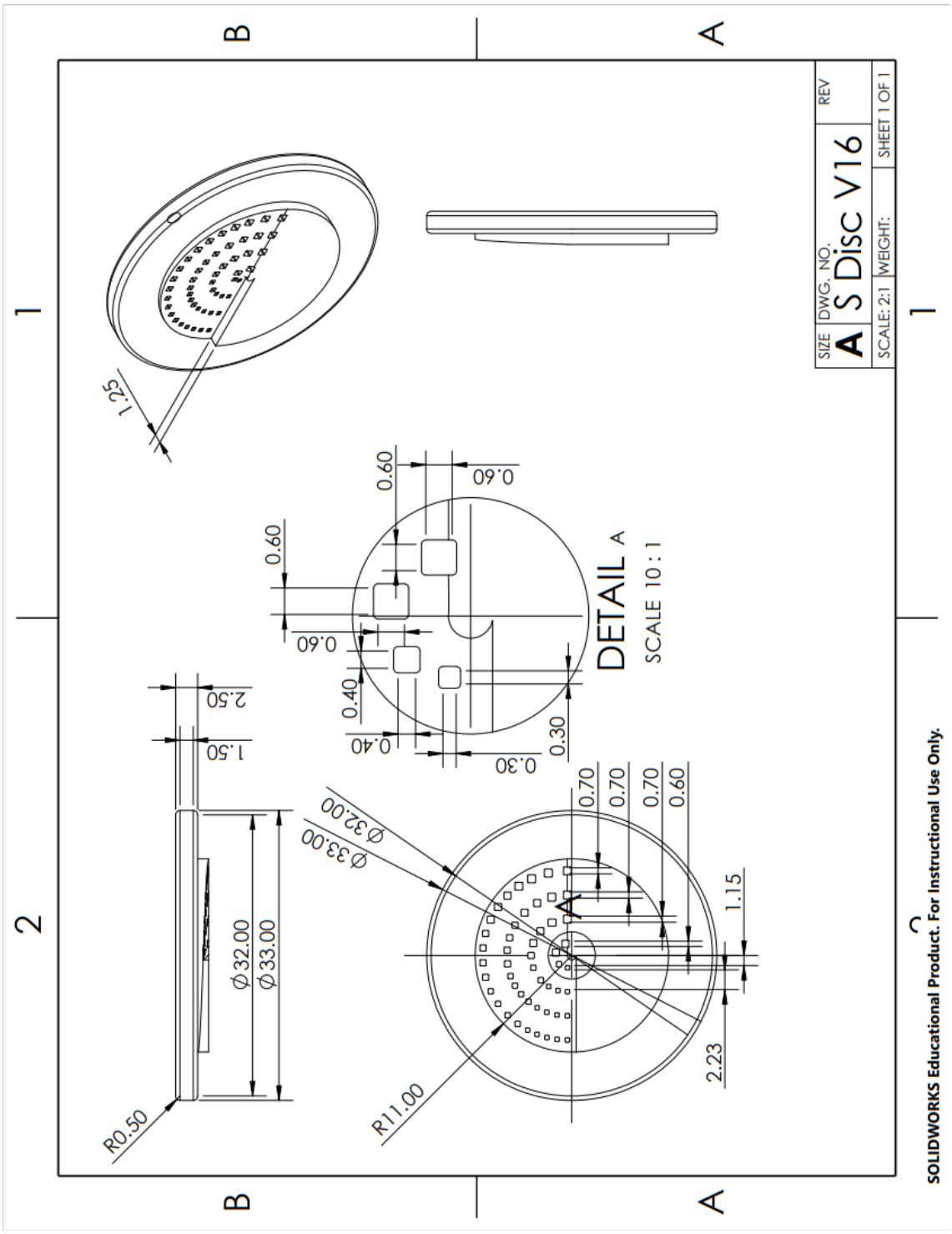

DETAIL A
SCALE 8 : 1
SIZE
A
DWG. NO.
Valve Body A V14
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

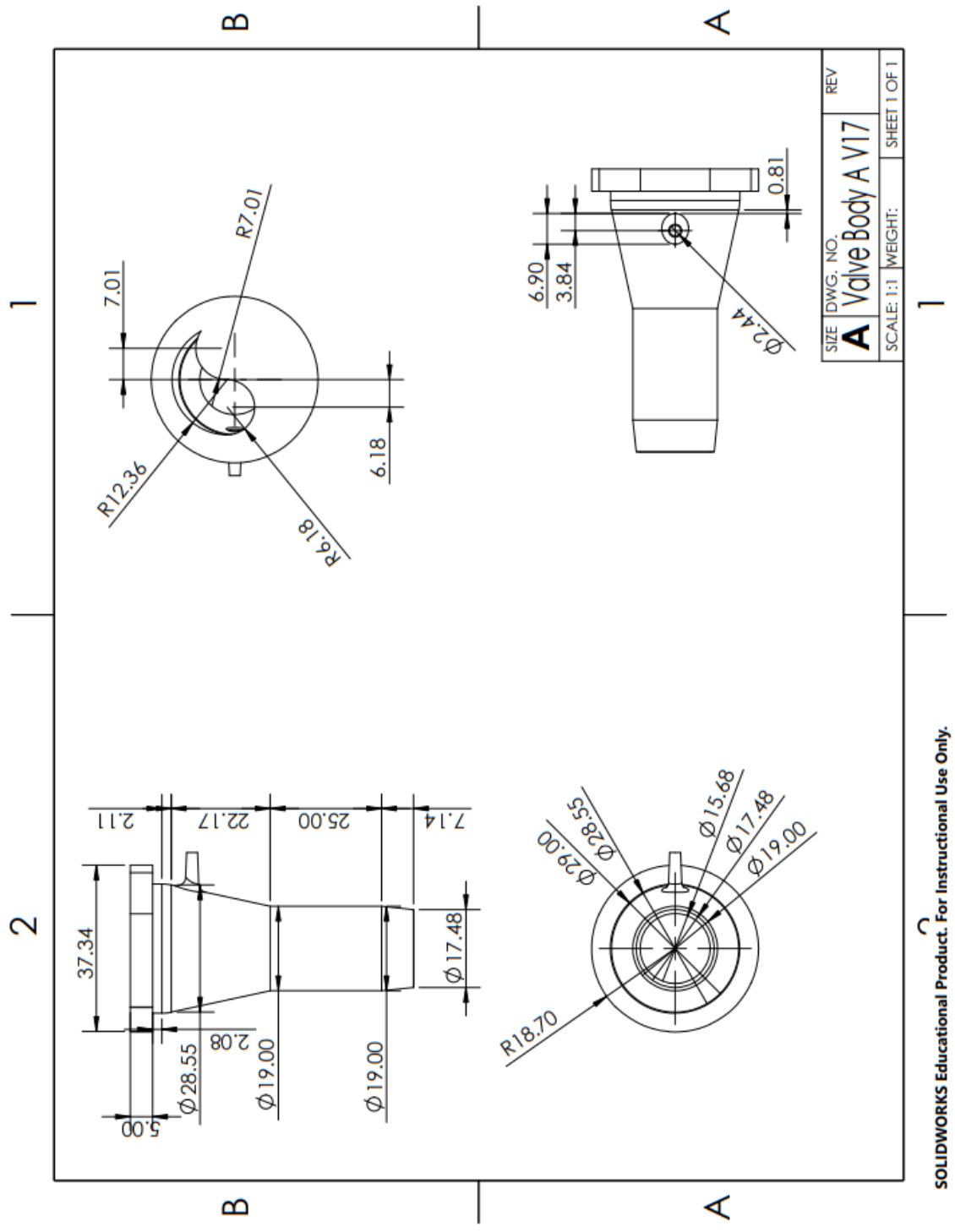

SIZE
A
DWG. NO.
S Disc V14
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
DETAIL A
SCALE 10 : 1
SOLIDWORKS Educational Product. For Instructional Use Only.
2.50
1.25
1.30
1.30
1.20
1.10
1.20
1.10
2.50
1.50
Ø32.00
Ø33.00
R0.50
Ø32.00
Ø33.00
1.90
1.90
1.70
0.85
1.63
Ø22.50

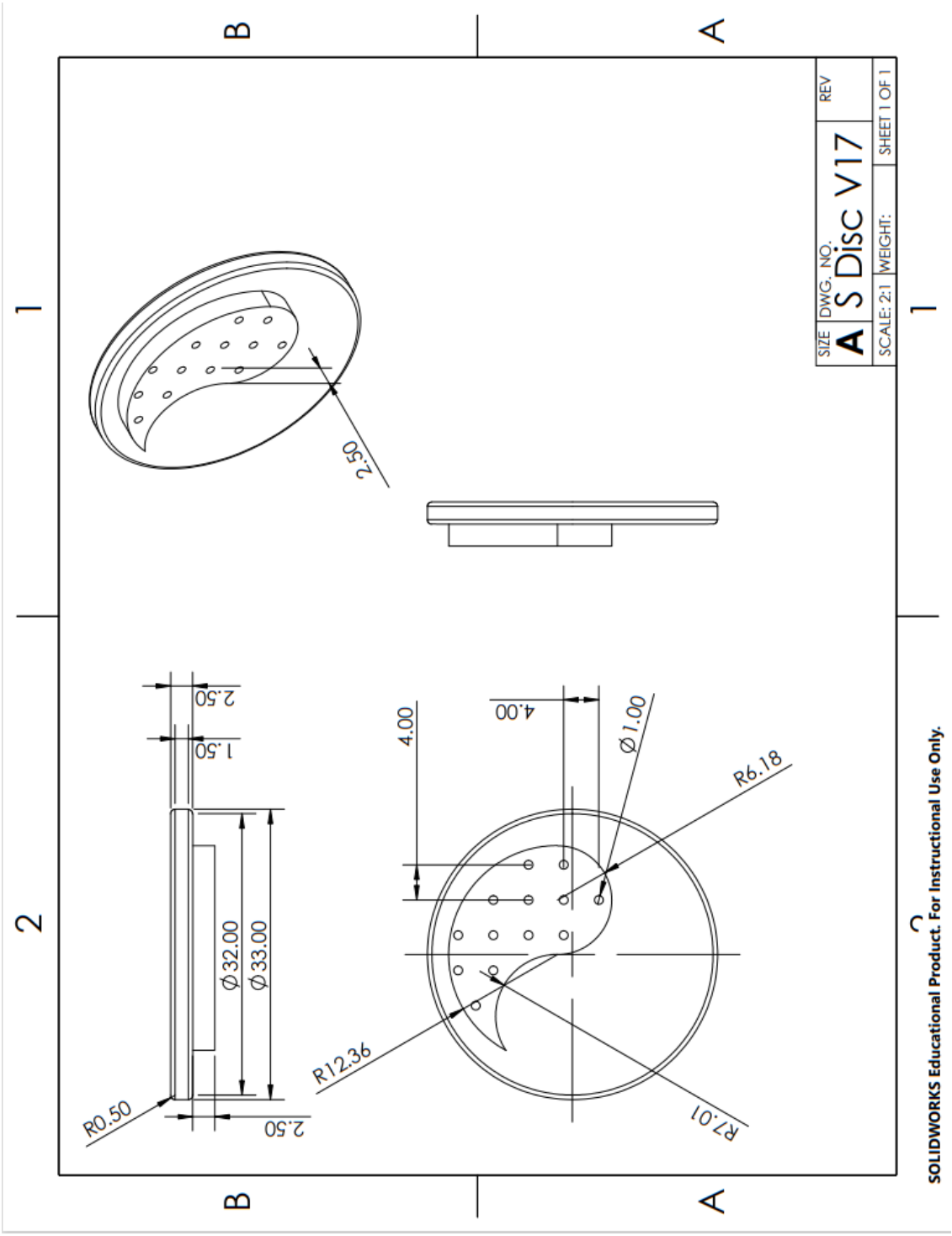

DETAIL A
SCALE 8 : 1
1.10
1.10
R18.70
1.65
0.83
A
6.90
3.84
0.81
Ø2.44
37.34
2.11
22.17
25.00
7.14
Ø28.55
2.08
Ø19.00
Ø19.00
Ø17.48
5.00
Ø29.00
Ø28.55
Ø15.68
Ø17.48
Ø19.00
R18.70
SIZE
A
DWG. NO.
Valve Body A V15
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.


2
1
B
A
R0.50
Ø32.00
Ø33.00
2.50
1.27
1.50
2.50
1.25
R11.00
Ø33.00
Ø32.00
A
1.63
0.85
1.10
1.10
DETAIL A
SCALE 10 : 1
SIZE
A
DWG. NO.
S Disc V15
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.


DETAIL A
SCALE 7.5 : 1
SIZE
A
DWG. NO.
Valve Body A V16
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

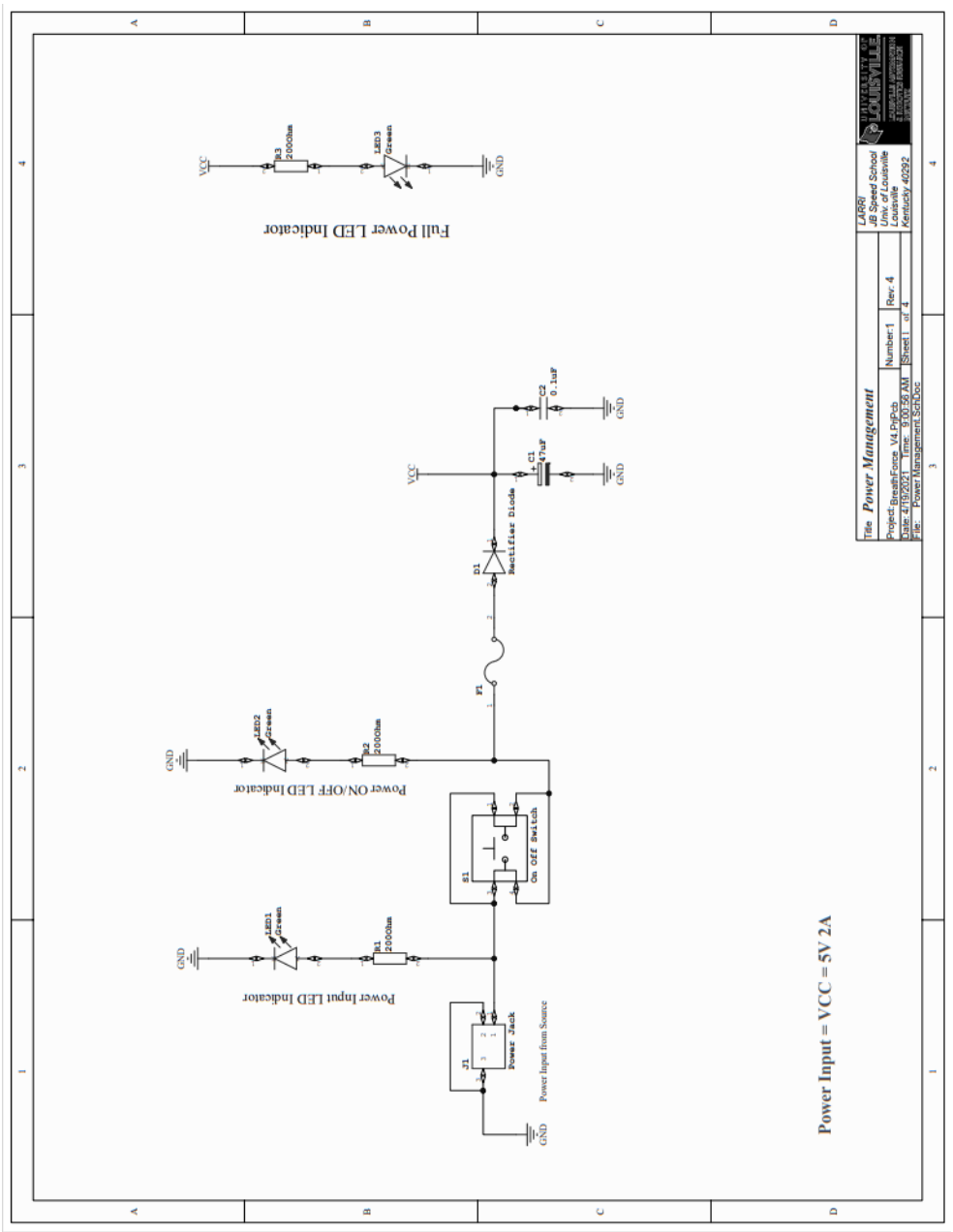

SIZE
A
DWG. NO.
S Disc V16
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
DETAIL A
SCALE 10 : 1
SOLIDWORKS Educational Product. For Instructional Use Only.

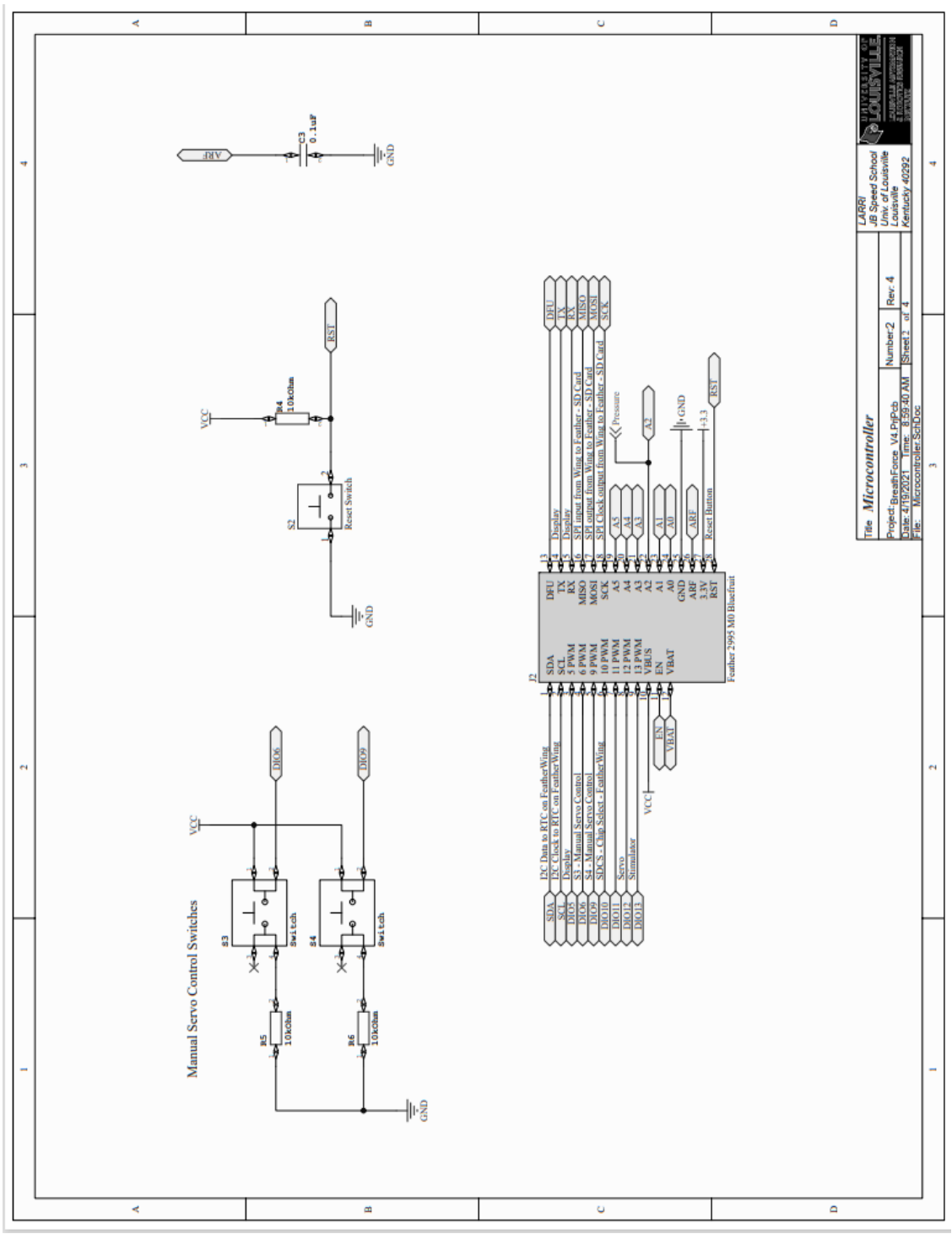

SIZE
A
DWG. NO.
Valve Body A V17
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.
R7.01
7.01
R12.36
R9.18
6.18
0.81
6.90
3.84
Ø2.44
Ø15.68
Ø17.48
Ø19.00
Ø29.00
Ø28.55
R18.70
2.11
22.17
25.00
7.14
37.34
Ø17.48
2.08
Ø28.55
Ø19.00
Ø19.00
5.00

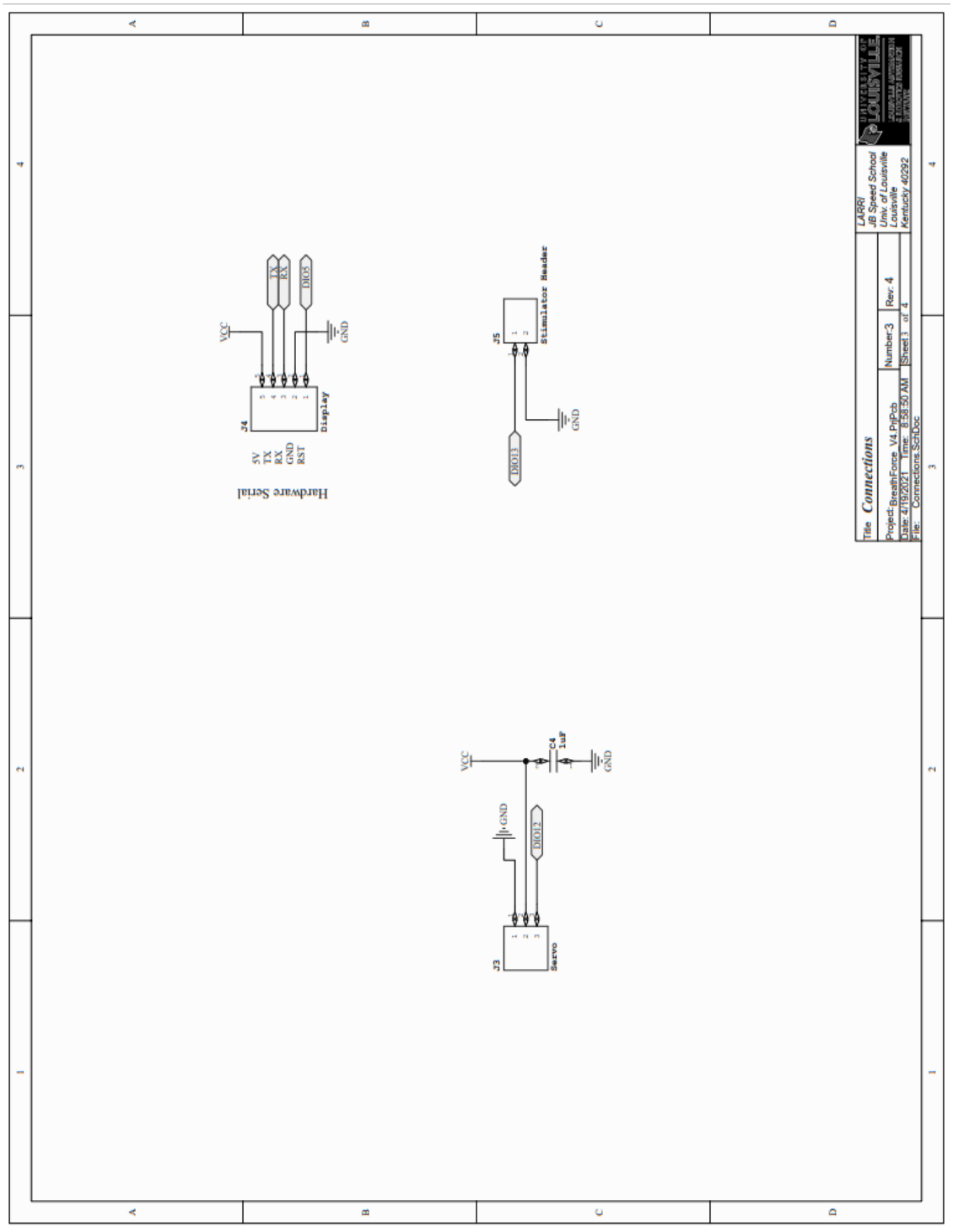

B
A
1
2
SIZE
A
DWG. NO.
S Disc V17
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
2.50
2.50
1.50
Ø32.00
Ø33.00
R0.50
2.50
4.00
4.00
Ø1.00
R6.18
R12.36
R7.01
SOLIDWORKS Educational Product. For Instructional Use Only.

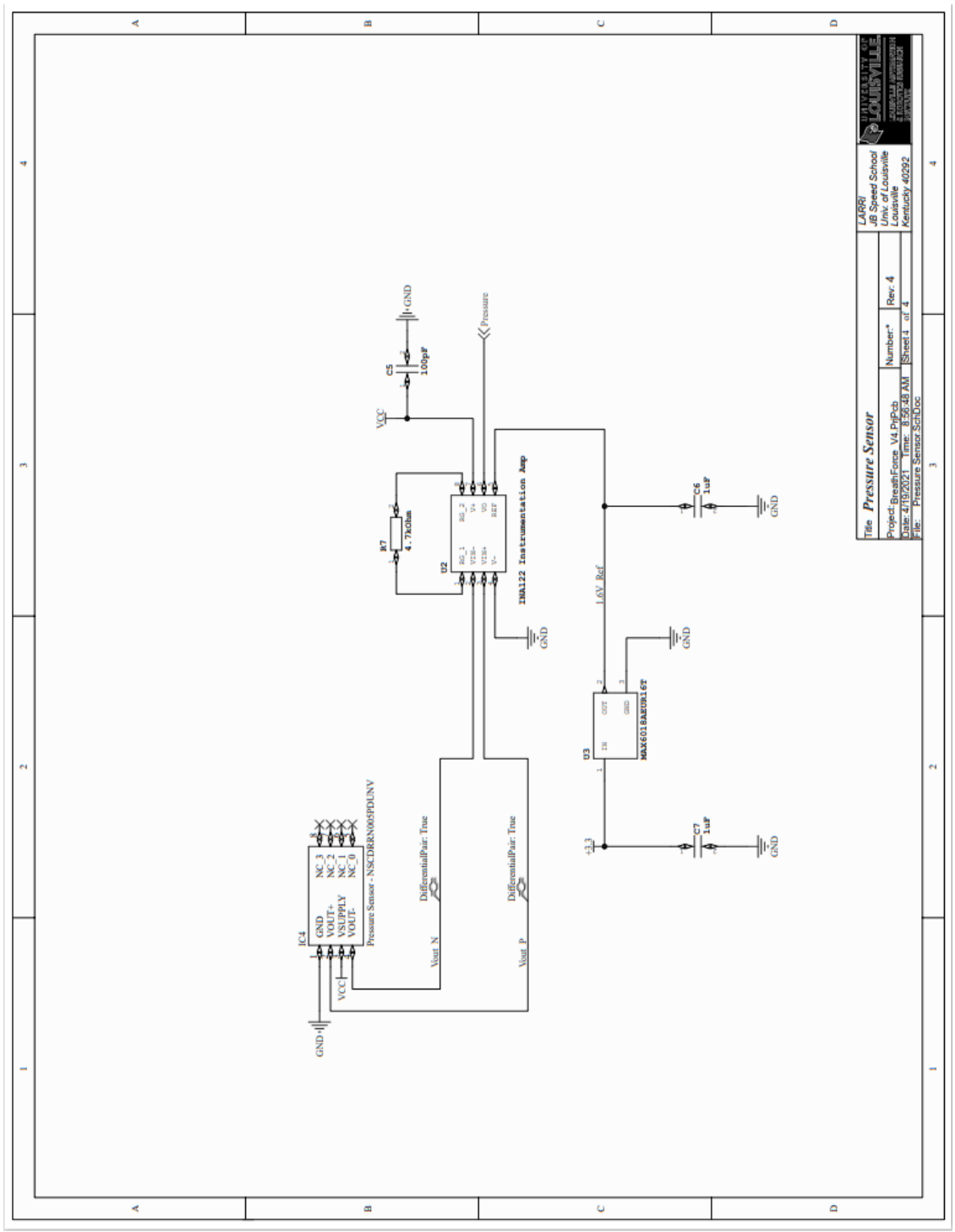
VIII. APPENDIX III

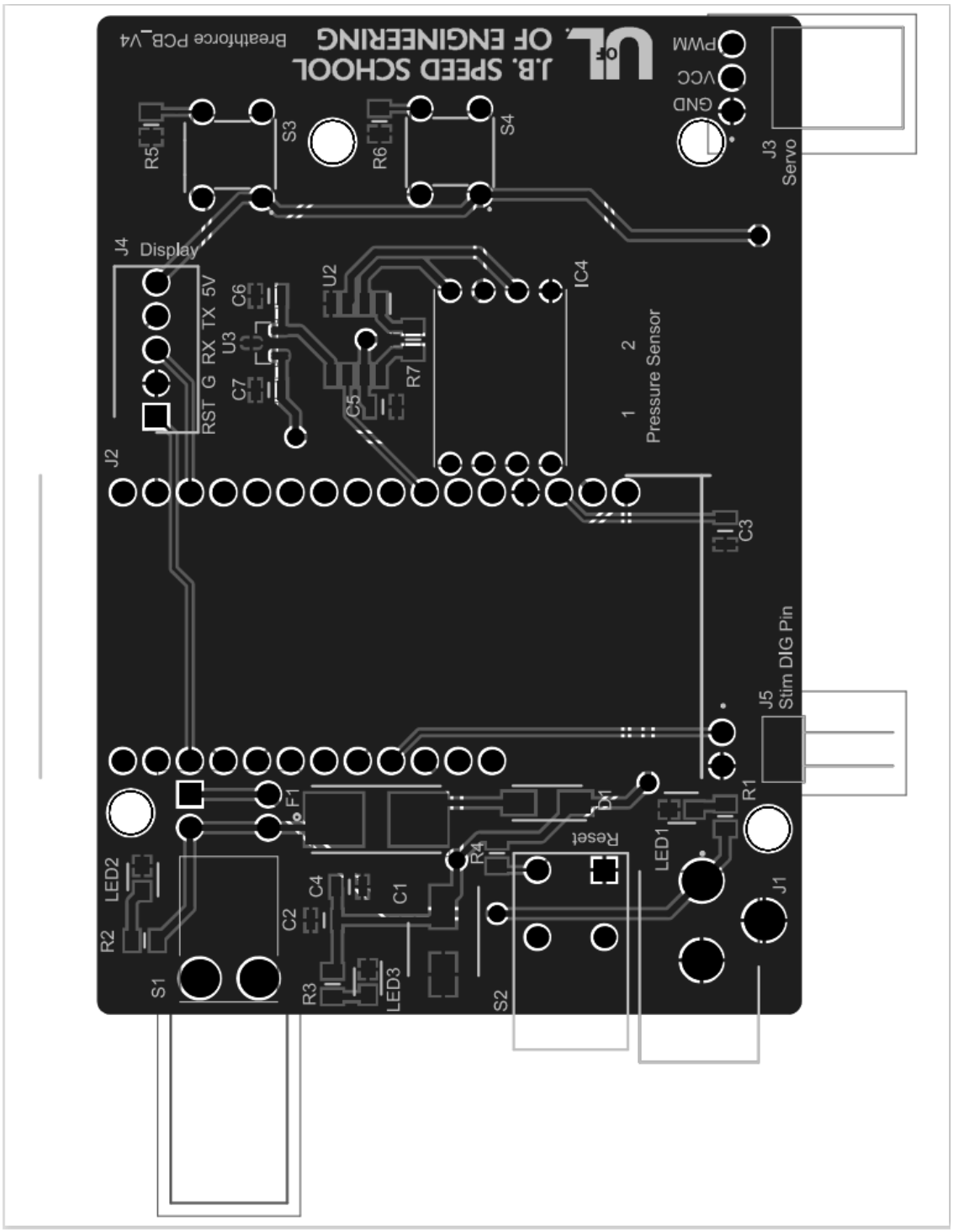
The schematics and 2D layout of the printed circuit board:




Manual Servo Control Switches
Reset Switch
Reset Button
Pressure
Feather 2995 M0 Bluefruit
Microcontroller
Project: BreathForce V4.PrjPcb
Date: 4/19/2021
Time: 8:59:40 AM
Sheet 2 of 4
File: Microcontroller.SchDoc
Number:2
Rev: 4
LARRI
JB Speed School
Univ. of Louisville
Louisville
Kentucky 40292


LARRI
JB Speed School
Univ. of Louisville
Louisville
Kentucky 40292
Title: Connections
Project: BreathForce_V4_PrjPcb
Number: 3
Rev: 4
Date: 4/19/2021 Time: 8:58:30 AM
Sheet 3 of 4
File: Connections.SchDoc
Hardware Serial
5V
TX
RX
GND
RST
J4
Display
VCC
GND
TX
RX
D05
J5
Stimulator Header
D013
GND
VCC
C4
1uF
GND
GND
D012
J3
Servo


Title Pressure Sensor
Project:BreathForce_V4.PrjPcb
Date: 4/19/2021 Time: 8:56:48 AM
File: Pressure Sensor.SchDoc
Number:* Rev: 4
Sheet 4 of 4
LARM
JB Speed School
Univ. of Louisville
Louisville
Kentucky 40292
GND
C5
10pF
VCC
Pressure
R7
4.7kOhm
U2
INA122 Instrumentation Amp
C6
1uF
1.6V Ref
U3
MAX6018AEUB+T
C7
1uF
IC4
Pressure Sensor - NSCDRRN005PDUNV
NC_3
NC_2
NC_1
NC_0
GND
VOUT+
VSUPPLY
VOUT-
DifferentialPair: True
Vout_N
Vout_P


Breathforce PCB_V4
J.B. SPEED SCHOOL OF ENGINEERING
PWM
VCC
GND
J3 Servo
S3
S4
R5
R6
J4 Display
5V TX RX G RST
U2
U3
C6
C7
C5
R7
IC4
Pressure Sensor
J2
C3
J5 Stim DIG Pin
F1
D1
R1
Reset
LED1
R4
LED2
R2
C4
C1
C2
J1
S1
R3
LED3
S2

IX. APPENDIX IV

The first iteration of the BreathForce system code installed in the Arduino Mega MCU:

```
#include <SD.h>
#include <Wire.h>
#include <TimeLib.h>
#include <DS1307RTC.h>  //Run RTClib -> SetTime example to re-calibrate time
tmElements_t tm; //Time things
#include <genieArduino.h>
Genie genie;
#define RESETLINE 4

//psi calculations
const int numReadings = 10;
int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average
float GAIN = 500.0;
float bits = 1023.0;
float resolution = 0.0002584;
float psi;
float offset = 2.5;
float Vcc = 5.0;
float calibration = 0.00;
int count;

int PSIhi;          //to help display on 4d display
int PSIlow;

int MEP = 0;
int MIP = 0;
int TargetMEP;
int TargetMIP;
int TrainingPercent;

int MEParray[105];
int MEPsumarray[95];
```

```
int E;
int MIParray[105];
int MIPsumarray[95];
int I;
long int PSITimer;

//Pressure sensor
int inputPin = A0;

//SD Card
File logfile;
const int chipSelect = 10;
char filename[] = "XXXXXX00.CSV";   //change patient name here. Format: (First thE letters of first and last name)00.CSV
char filename2[] = "XXXXXX00.CSV";
int x;
int xx;                         //helps display previous file
int xxx;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup() {
  //for PSI calculation
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }

  //for 4d display to boot up
  Serial.begin(200000);                             // Serial0 @ 150000 (200K) Baud
  genie.Begin(Serial);                              // Use Serial0 for talking to the Genie Library, and to the 4D Systems display
  genie.AttachEventHandler(myGeniEventHandler);         // Attach the user function Event Handler for processing events
  pinMode(RESETLINE, OUTPUT);                           // Set D4 on Arduino to Output (4D Arduino Adaptor V2 - Display Reset)
  digitalWrite(RESETLINE, 1);                       // Reset the Display via D4
  delay(100);
  digitalWrite(RESETLINE, 0);                       // unReset the Display via D4
  delay (3500);                                   //let the display start up after the reset (This is important)
  genie.WriteContrast(1); //1=ScrEn on, 0 = scrEn off
  delay(500);

  //Starting process
  genie.WriteStr(0, "STARTING UP...");
  delay(300);
```

```
  //RTC Boot Check
  genie.WriteStr(0, "Checking RealTimeClock (RTC) status:...");
  delay(300);
  if (RTC.read(tm)) {
    genie.WriteStr(0, "RTC is running.");
    delay(300);
  }
  else {
    if (RTC.chipPresent()) {
      genie.WriteStr(0, "The DS1307 is stopped. Please run the SetTime example
DS1307RTC to recalibrate");
      while (1) {};
    }
    else {
      genie.WriteStr(0, "DS1307 read error! Please check the circuitry/battery.");
      while (1) {};
    }
  }

  //SD Card Check
  genie.WriteStr(0, "Initializing SD card:...");
  delay(300);

  pinMode(53, OUTPUT);
  if (!SD.begin (SPI_FULL_SPEED, chipSelect)) {
    genie.WriteStr(0, "Failed. SD Card not present: 1) Insert SD. 2) Press reset.");
    while (1) {};
  }
  else {
    genie.WriteStr(0, "Success.");
    delay(300);
  }
  genie.WriteStr(0, "Press 'START' when ready...");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void loop() {
  genie.DoEvents(); // This calls the library each loop to process the queued responses
from the display
}

////////////////////////////////////////////////////////////////////////////////////

void myGeniEventHandler(void) {
```

```
  genieFrame Event;
  genie.DequeueEvent(&Event); // Remove the next queued event from the buffer, and
process it below

  int EVENT_val = 0;
  int slider_val = 0;

  EVENT_val = genie.GetEventData(&Event);

  //If the cmd received is from a Reported Event (Events triggered from the Events tab of
Workshop4 objects)
  if (Event.reportObject.cmd == GENIE_REPORT_EVENT)
  {
    //For Calibration on directory screen
    if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
    {
      if (Event.reportObject.index == 0)
      {
        genie.WriteObject(GENIE_OBJ_FORM, 0, 1);
        delay(1000);
        genie.WriteStr(26, "Calibrating...");
        delay(500);
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, 333);
        delay(1000);
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, 222);
        delay(1000);
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, 111);
        delay(1000);
        long int CALendtime = 0;
        CALendtime = millis() + 1500;
        while (millis() < CALendtime)
        {
          PSIcalculation(); // PSI calculation
          genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, psi);
        }
        delay(500);
        calibration = psi;
        PSIcalculation();
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, psi);
        delay(500);
        genie.WriteStr(26, "Calibrated.");
        delay(1000);
        genie.WriteStr(26, "Make File Session.");
      }
    }
```

```
  //To create new file for session
  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
  {
   if (Event.reportObject.index == 13)
   {
    genie.WriteStr(18, "Checking INFO file...");
    delay(1000);
    if (Event.reportObject.index == 13)
    {
     MakeNewFile();
    }
   }
  }

  //Puts in High (e) and Low (i) psi into file. Then displays results of session and
displays previous session
  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
  {
   if (Event.reportObject.index == 10)
   {
    PullFromNewAndOldFiles();
   }
  }

  //Adjusts LED Target Values to Trigger
  if (Event.reportObject.object == GENIE_OBJ_SLIDER)
  {
   if (Event.reportObject.index == 3)
   {
    slider_val = genie.GetEventData(&Event);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 7, slider_val);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, slider_val);
    TargetMEP = slider_val;
   }
  }

  if (Event.reportObject.object == GENIE_OBJ_SLIDER)
  {
   if (Event.reportObject.index == 2)
   {
    slider_val = genie.GetEventData(&Event);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 5, slider_val);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, slider_val);
    TargetMIP = -1 * slider_val;
   }
  }
```

```
//For MIP
if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
{
 if (Event.reportObject.index == 6)
 {
  TimerMIP(8, 1, 4, 17);
  genie.WriteStr(17, "Done.");
 }
}

//For MEP
if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
{
 if (Event.reportObject.index == 14)
 {
  TimerMEP(8, 1, 4, 17);
  genie.WriteStr(17, "Done.");
 }
}

//   //For 5secMIP
//   if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//   {
//    if (Event.reportObject.index == 0)
//    {
//     TimerMIP(8, 1, 4, 17);
//     genie.WriteStr(17, "Done.");
//    }
//   }
//
//   //For 5secMEP
//   if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//   {
//    if (Event.reportObject.index == 1)
//    {
//     TimerMIP(8, 1, 4, 17);
//     genie.WriteStr(17, "Done.");
//    }
//   }
//
//   //For Valsalva
//   if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//   {
//    if (Event.reportObject.index == 22)
//    {
```

```
//      TimerMEP(8, 1, 4, 17);
//      genie.WriteStr(17, "Done.");
//    }
//  }
//
//  //For StimMIP
//  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//  {
//    if (Event.reportObject.index == 6)
//    {
//      TimerMIP(8, 1, 4, 17);
//      genie.WriteStr(17, "Done.");
//    }
//  }
//
//  //For StimMEP
//  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//  {
//    if (Event.reportObject.index == 14)
//    {
//      TimerMEP(8, 1, 4, 17);
//      genie.WriteStr(17, "Done.");
//    }
//  }
//
//  //For Stim5secMIP
//  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//  {
//    if (Event.reportObject.index == 0)
//    {
//      TimerMIP(8, 1, 4, 17);
//      genie.WriteStr(17, "Done.");
//    }
//  }
//
//  //For Stim5secMEP
//  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//  {
//    if (Event.reportObject.index == 1)
//    {
//      TimerMIP(8, 1, 4, 17);
//      genie.WriteStr(17, "Done.");
//    }
//  }
//
//  //For StimValsalva
```

```
//    if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
//    {
//      if (Event.reportObject.index == 22)
//      {
//        TimerMEP(8, 1, 4, 17);
//        genie.WriteStr(17, "Done.");
//      }
//    }

//PSI Training Timers
if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
{
  if (Event.reportObject.index == 15)
  {
    PSITimer = 180500;
    count = 5;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer / 1000);
  }
}
if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
{
  if (Event.reportObject.index == 16)
  {
    PSITimer = 240500;
    count = 4;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer / 1000);
  }
}
if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
{
  if (Event.reportObject.index == 17)
  {
    PSITimer = 300500;
    count = 3;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer / 1000);
  }
}

//Shows results of MEP/MIP on MEP/MIP result
if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
{
  if (Event.reportObject.index == 8)
  {
    delay(500);
    genie.WriteObject(GENIE_OBJ_FORM, 7, 1);
    delay(500);
```

```
      genie.WriteStr(12, MEP);
      delay(500);
      TargetMEP = MEP * TrainingPercent / 100;
      genie.WriteStr(13, TargetMEP);
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, TargetMEP);
      delay(500);
      genie.WriteStr(14, abs(MIP));
      delay(500);
      TargetMIP = MIP * TrainingPercent / 100;
      genie.WriteStr(15, abs(TargetMIP));
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, abs(TargetMIP));
    }
   }
   //LUNG TRAINING button takes to Form 3
   if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)
   {
    if (Event.reportObject.index == 5)
    {
     genie.WriteObject(GENIE_OBJ_FORM, 3, 1);
     delay(100);
    }
   }

   //For PSI training screen. Write to excel and give option to end or continue.
   if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
   {
    if (Event.reportObject.index == 1)
    {
     logfile = SD.open(filename, FILE_WRITE);
     TimerAndRecordPSI ();
     logfile.close();
     delay(500);
     genie.WriteStr(2, "Done.");
     delay(1000);
     genie.WriteStr(2, "RESULTS to see results \nor START to repeat training.");
    }
    genie.WriteObject(GENIE_OBJ_4DBUTTON, 1, 0);
   }
  }
}

//PSI calculation//////////////////////////////////////////////////////////////////////////////
int PSIcalculation() {
 total = total - readings[readIndex];       //psi readings + calculations
 readings[readIndex] = analogRead(inputPin);
 total = total + readings[readIndex];
```

```
  readIndex = readIndex + 1;
  if (readIndex >= numReadings) {
    readIndex = 0;
  }
  psi = total / numReadings;
  psi = psi / bits;              //bits = 1023.0
  psi = psi * Vcc;               //Vcc=5.0
  psi = psi - offset;            //offset = 2.5
  psi = psi / resolution;        //resolution = 0.0002584
  psi = psi / GAIN;              //GAIN
  psi = psi * 10 * 1.36;         //1.36 is conversion from 1 mmHg = 1.36 cmH2O
  psi = psi - calibration;
}

//Makes new file name based off INFO file and setsup CVS file//////////////////////////////
void MakeNewFile() {

  genie.WriteStr(18, "Patient Info:...");
  delay(1000);
  logfile = SD.open("INFO.TXT");
  filename[0] = logfile.read();
  filename[1] = logfile.read();
  filename[2] = logfile.read();
  filename[3] = logfile.read();
  filename[4] = logfile.read();
  filename[5] = logfile.read();
  filename2[0] = filename[0];
  filename2[1] = filename[1];
  filename2[2] = filename[2];
  filename2[3] = filename[3];
  filename2[4] = filename[4];
  filename2[5] = filename[5];
  logfile.close();
  genie.WriteStr(18, filename);
  delay(1000);

  genie.WriteStr(18, "Making new file:...");
  for (uint8_t i = 1; i < 100; i++) {
    filename[6] = i / 10 + '0';
    filename[7] = i % 10 + '0';
    if (! SD.exists(filename)) {
      logfile = SD.open(filename, FILE_WRITE);
      logfile.print(",,");
      logfile.print("Date(D/M/Y)/Time:,");          //A1:B1
      logfile.print(tm.Day);
      logfile.print("/");
```

```
logfile.print(tm.Month);
logfile.print("/");
logfile.print(tmYearToCalendar(tm.Year));
logfile.print(",");
logfile.print(tm.Hour);
logfile.print(":");
logfile.print(tm.Minute);
logfile.print(":");
logfile.println(tm.Second);
logfile.print(",,");
logfile.print("Highest Exp. PSI:");                  //A2:B2
logfile.print(",");
logfile.print("       ");
logfile.print(",");
logfile.println("cm H2O");
logfile.print(",,");
logfile.print("Lowest Insp. PSI:");                  //A3:B3
logfile.print(",");
logfile.print("       ");
logfile.print(",");
logfile.println("cm H2O");
logfile.print(",,");
logfile.print("MEP:");                  //A2:B2
logfile.print(",");
logfile.print("       ");
logfile.print(",");
logfile.println("cm H2O");
logfile.print(",,");
logfile.print("MIP:");                  //A3:B3
logfile.print(",");
logfile.print("       ");
logfile.print(",");
logfile.println("cm H2O");
logfile.print(",,");
logfile.print("Training %");
logfile.print(",");
logfile.print("     ");
logfile.print(",");
logfile.println("%");
logfile.print(",,");
logfile.print("# of points:");                  //A4:B4
logfile.print(",");
logfile.print( "=COUNT(B:B)");
logfile.print(",");
logfile.println("points");
logfile.print(",,");
```

```
    logfile.print("Time Duration");               //A5:B5
    logfile.print(",");
    logfile.print("=(MAX(A:A)-MIN(A:A))/1000");
    logfile.print(",");
    logfile.println("seconds");
    logfile.print(",,");
    logfile.print("Points per second:");          //A6:B6
    logfile.print(",");
    logfile.print("=D7/D8");
    logfile.print(",");
    logfile.println("points/s");
    logfile.print("Time(ms)");      //A8:B8
    logfile.print(",");
    logfile.println("cm H2O");
    logfile.close();
    break; // leave the loop!;
   }
  }
  delay(1000);
  genie.WriteStr(18, filename);
  delay(1000);

  genie.WriteStr(18, "Checking old file...");
  delay(1000);
  filename2[6] = filename[6];
  filename2[7] = filename[7];
  xx = filename2[6] - '0';   //Renames filename to go back 1 previous session to display on
result scrEn
  xxx = filename2[7] - '0';
  x = (xx * 10) + xxx - 1;
  filename2[6] = x / 10 + '0';
  filename2[7] = x % 10 + '0';
  logfile = SD.open(filename2, FILE_READ);
  logfile.seek(20);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
```

```
  MEP = logfile.readStringUntil(',').toInt();
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  MIP = logfile.readStringUntil(',').toInt();
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  TrainingPercent = logfile.readStringUntil(',').toInt();
  logfile.close();

  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, MEP);    //MEP
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 13, abs(MIP));    //MIP
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, TrainingPercent);    //% Training

  genie.WriteStr(18, filename2);
  delay(1000);

  genie.WriteStr(18, "Proceed to MEP/MIP.");
  delay(100);
}

//MIP calculating//////////////////////////////////////
void TimerMIP (int a, int b, int c, int d) {
  genie.WriteStr(d, "Starting...");
  delay(1000);
  genie.WriteStr(d, "3...");
  delay(1000);
  genie.WriteStr(d, "2...");
  delay(1000);
  genie.WriteStr(d, "1...");
  delay(1000);
  genie.WriteStr(d, "Now.");
  delay(500);

  long int MIPendtime = 0;
  I = 0;
  MIP = 0;
  MIPendtime = millis() + 10500;
  while (millis() < MIPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, a, (MIPendtime - millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_SCOPE, b, abs(psi));
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, c, abs(psi));
    MIParray[I] = psi;
```

```
    I++;
  }
  for (I = 0; I < 96; I++)
  {
    MIPsumarray[I] = (MIParray[I] + MIParray[I + 1] + MIParray[I + 2] + MIParray[I +
3] + MIParray[I + 4] + MIParray[I + 5] + MIParray[I + 6] + MIParray[I + 7] +
MIParray[I + 8] + MIParray[I + 9]) / 10;
    delay(5);
  }
  int idy;
  for (byte idy = 0; idy != 96; idy++)
  {
    if (MIPsumarray[idy] < MIP) {
      MIP = min(MIPsumarray[idy], MIP);
    }
    delay(5);
  }
}
//MEP calculating//////////////////////////////////////////////////////////////
void TimerMEP (int a, int b, int c, int d) {
  genie.WriteStr(d, "Starting...");
  delay(1000);
  genie.WriteStr(d, "3...");
  delay(1000);
  genie.WriteStr(d, "2...");
  delay(1000);
  genie.WriteStr(d, "1...");
  delay(1000);
  genie.WriteStr(d, "Now.");
  delay(500);

  long int MEPendtime = 0;
  E = 0;
  MEP = 0;
  MEPendtime = millis() + 10500;
  while (millis() < MEPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, a, (MEPendtime - millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_SCOPE, b, abs(psi));
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, c, abs(psi));
    MEParray[E] = psi;
    E++;
  }
  for (E = 0; E < 96; E++)
  {
```

```
    MEPsumarray[E] = (MEParray[E] + MEParray[E + 1] + MEParray[E + 2] +
MEParray[E + 3] + MEParray[E + 4] + MEParray[E + 5] + MEParray[E + 6] +
MEParray[E + 7] + MEParray[E + 8] + MEParray[E + 9]) / 10;
    delay(5);
  }
  int idx;
  for (byte idx = 0; idx != 96; idx++)
  {
    if (MEPsumarray[idx] > MEP) {
      MEP = max(MEPsumarray[idx], MEP);
    }
    delay(5);
  }
}
//Timer and PSI session calculating////////////////////////////////////////////////////////////
void TimerAndRecordPSI () {
  long int Pendtime = 0;
  while (count > 0)
  {
    delay(500);
    genie.WriteStr(2, "Starting...");
    delay(1000);
    genie.WriteStr(2, "3...");
    delay(1000);
    genie.WriteStr(2, "2...");
    delay(1000);
    genie.WriteStr(2, "1...");
    delay(1000);
    genie.WriteStr(2, "Now...");
    delay(500);
    Pendtime = millis() + PSITimer;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer / 1000);
    while (millis() < Pendtime)
    {
      PSIcalculation();
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, (Pendtime - millis()) / 1000);
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 1, abs(psi));
      genie.WriteObject(GENIE_OBJ_SLIDER, 1, psi + 40);
      if (psi > PSIhi) {
        PSIhi = psi;
      }
      if (psi < PSIlow) {
        PSIlow = psi;
      }
      if (psi > (TargetMEP * 0.90)) {
        genie.WriteObject(GENIE_OBJ_USER_LED, 0, 1);
```

```
      }
      else {
        genie.WriteObject(GENIE_OBJ_USER_LED, 0, 0);
      };

      if (psi < (TargetMIP * 0.90)) {
        genie.WriteObject(GENIE_OBJ_USER_LED, 1, 1);
      }
      else {
        genie.WriteObject(GENIE_OBJ_USER_LED, 1, 0);
      };
      logfile.print (millis());
      logfile.print (",");
      logfile.println(psi);
    }
//    delay(1000);
//    genie.WriteStr(2, "1-minute rest");
//    delay(1000);
//    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, 60);
//    Pendtime = millis() + 60500;
//    while (millis() < Pendtime)
//    {
//      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, (Pendtime - millis()) / 1000);
//    }
//    delay(1000);
    count = count - count ; //supposed to be count - 1 when doing longer consecutive
sessions.
  }
}

//Pulls information from old file to display on session results scrEn////////////////
void PullFromNewAndOldFiles() {
  delay(500);
  logfile = SD.open(filename, FILE_WRITE);
  logfile.seek(20);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(PSIhi);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(PSIlow);
  logfile.readStringUntil(',');
```

```
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.print(MEP);
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.print(MIP);
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.print(TrainingPercent);
logfile.close();

genie.WriteObject(GENIE_OBJ_FORM, 4, 1);
delay(1000);
genie.WriteStr(5, filename);
delay(500);

logfile = SD.open(filename, FILE_READ);
logfile.seek(20);
genie.WriteStr(1, logfile.readStringUntil(','));    //DATE
delay(500);
genie.WriteStr(4, logfile.readStringUntil(','));    //TIME
delay(500);
logfile.readStringUntil(',');
logfile.readStringUntil(',');
genie.WriteStr(6, logfile.readStringUntil(','));    //EXHALE
delay(500);
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
genie.WriteStr(3, logfile.readStringUntil(','));    //INHALE
delay(500);
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
genie.WriteStr(19, logfile.readStringUntil(','));    //MEP
delay(500);
logfile.readStringUntil(',');
logfile.readStringUntil(',');
logfile.readStringUntil(',');
genie.WriteStr(21, logfile.readStringUntil(','));    //MIP
delay(500);
```

```
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(23, logfile.readStringUntil(','));    //% Training
  delay(500);
  logfile.close();

  xx = filename[6] - '0';   //Renames filename to go back 1 previous session to display on
result scrEn
  xxx = filename[7] - '0';
  x = (xx * 10) + xxx - 1;
  filename[6] = x / 10 + '0';
  filename[7] = x % 10 + '0';
  genie.WriteStr(7, filename);
  delay(500);

  logfile = SD.open(filename, FILE_READ);
  logfile.seek(20);
  genie.WriteStr(8, logfile.readStringUntil(','));
  delay(500);
  genie.WriteStr(9, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(10, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(11, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(20, logfile.readStringUntil(','));    //MEP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(22, logfile.readStringUntil(','));    //MIP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(24, logfile.readStringUntil(','));    //% Training
  delay(500);
```

```
  logfile.close();
}
```

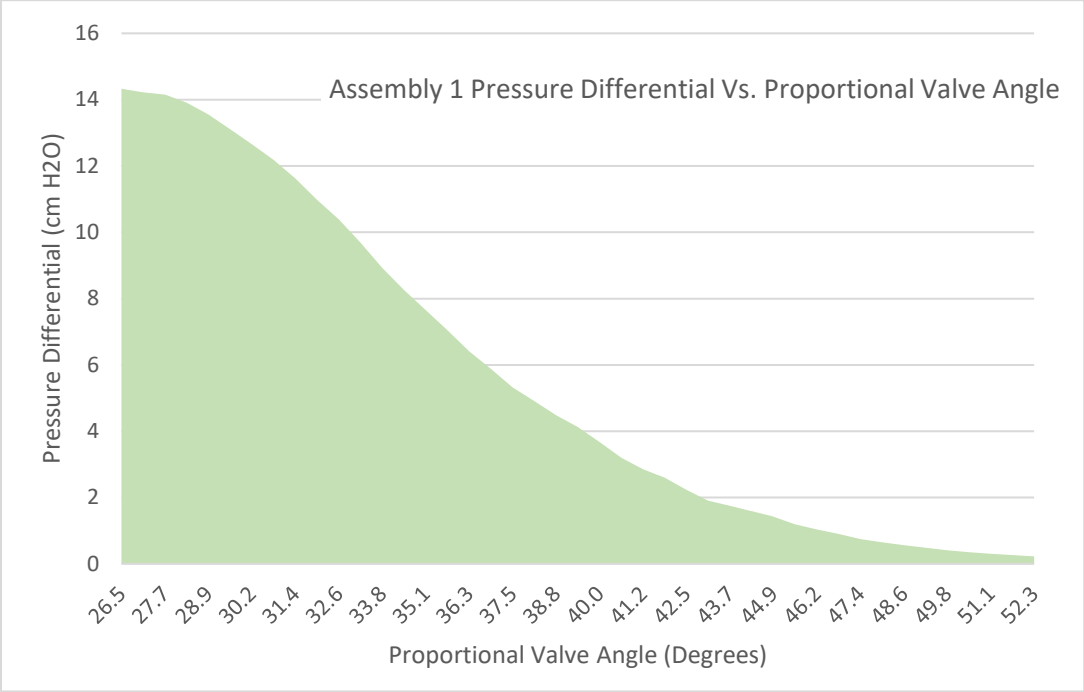
X. APPENDIX V

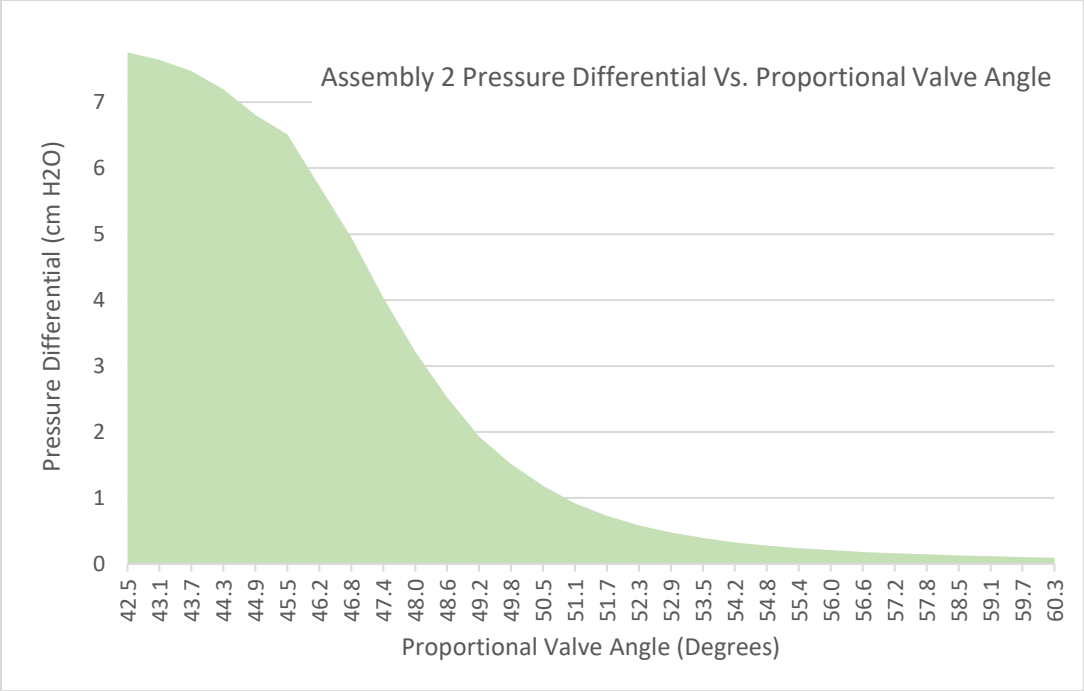
Conversion from PSI to cm H_2O used in the Arduino code:

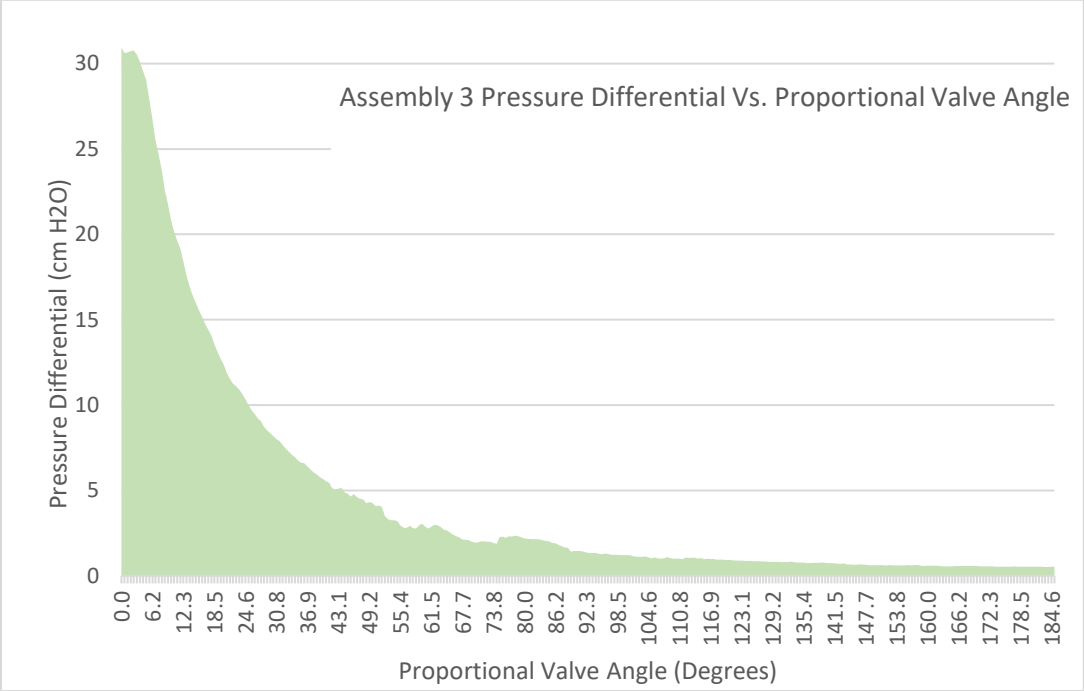
```
psi = total / numReadings;
  psi = psi / bits;            //bits = 1023.0
  psi = psi * Vcc;             //Vcc=3.3
  psi = psi - offset;          //offset = 2.5
  psi = psi / resolution;      //resolution = 0.0002584
  psi = psi / GAIN;            //GAIN
  psi = psi * 1.36;       //1.36 is conversion from 1 mmHg = 1.36 cmH2O
  psi = psi - calibration;
```

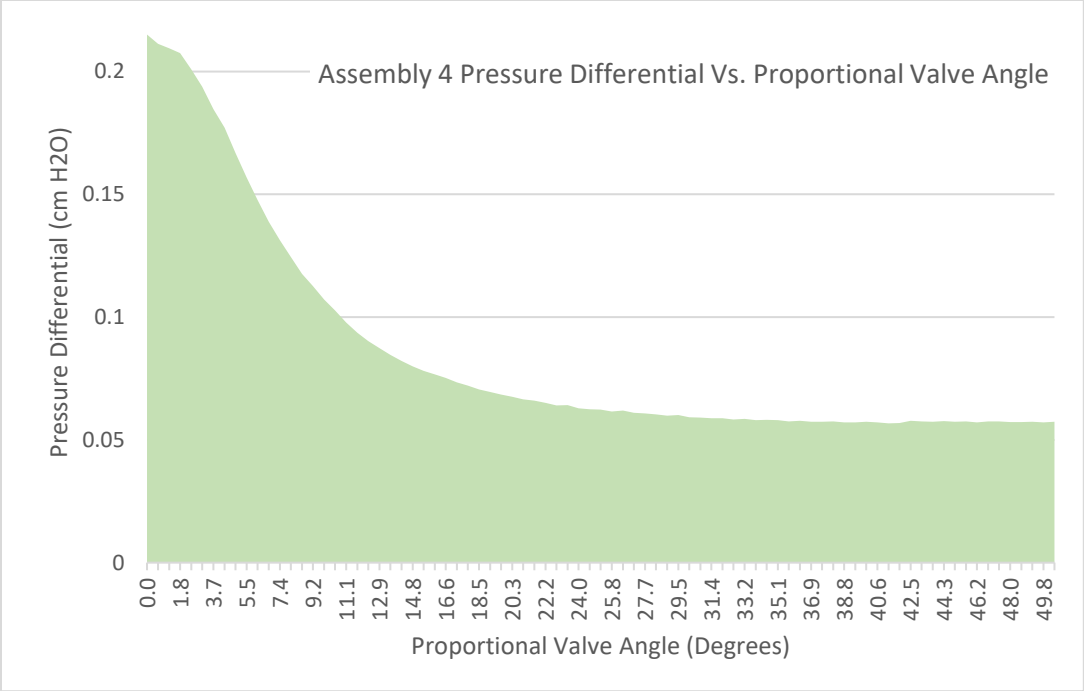
XI. APPENDIX VI

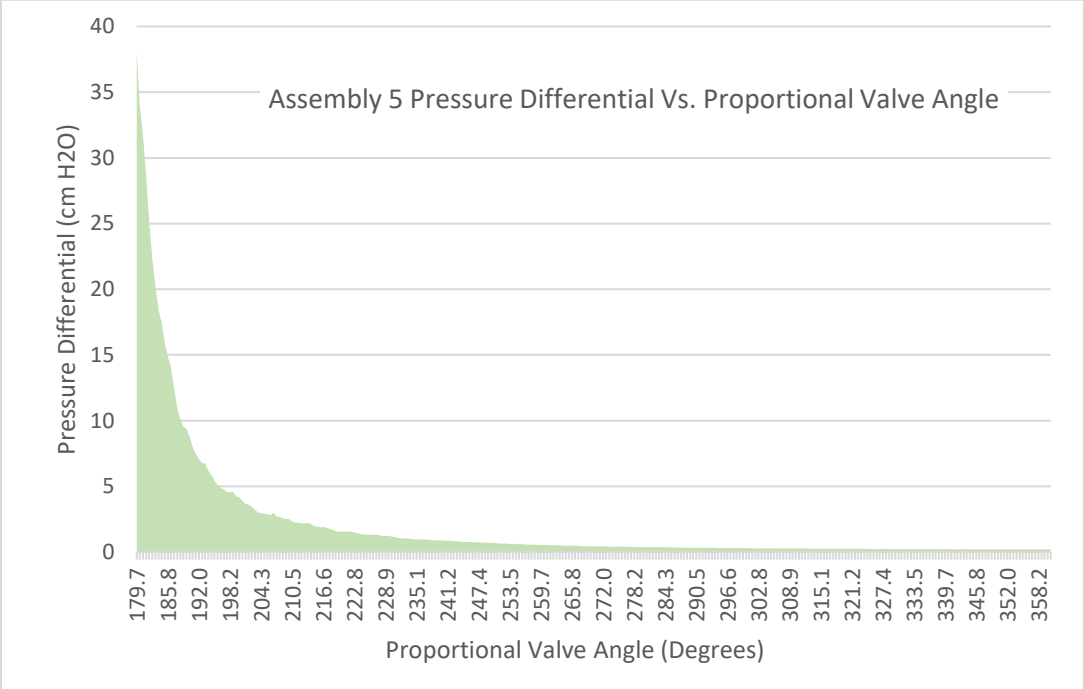
The computation fluid dynamics study results for the seventeen designs:

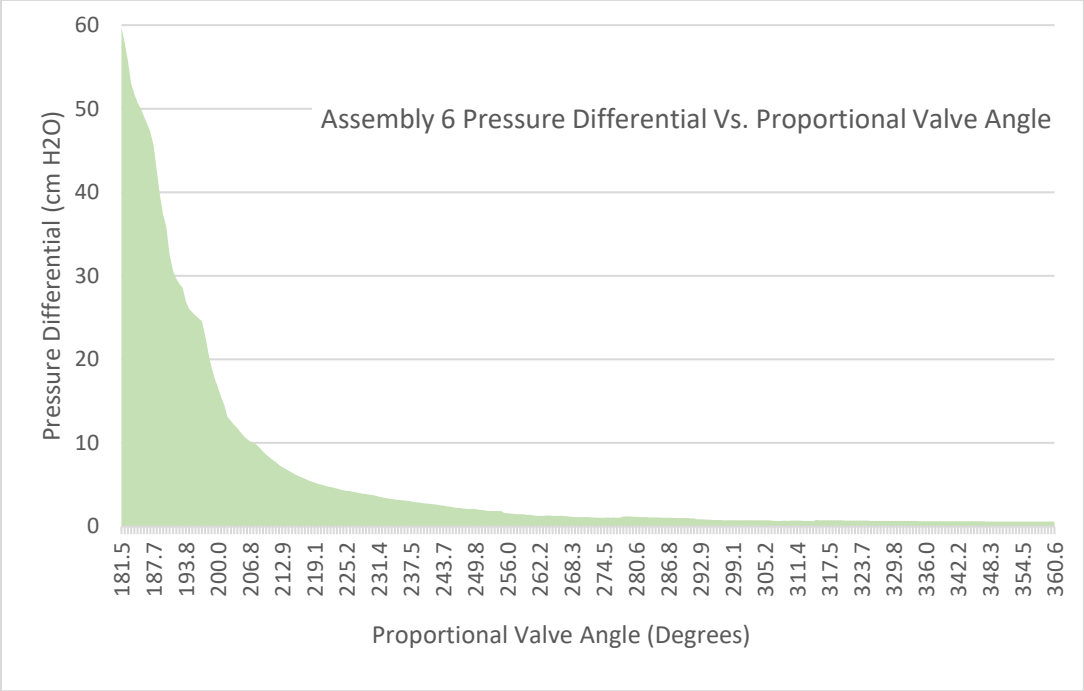


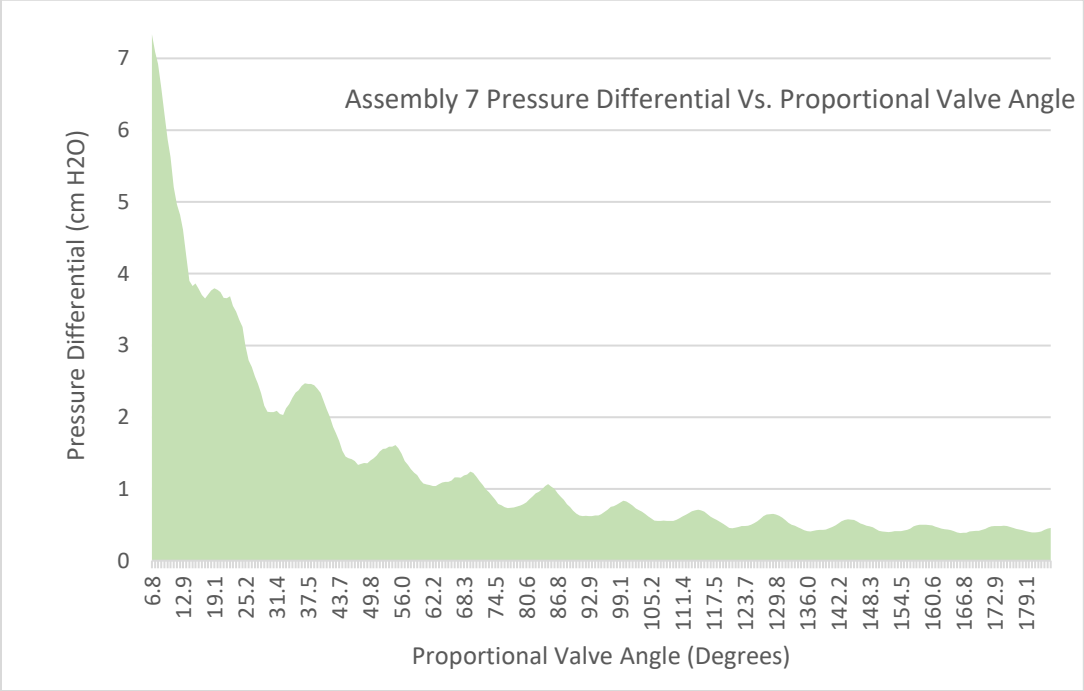

Assembly 2 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0
1
2
3
4
5
6
7
42.5
43.1
43.7
44.3
44.9
45.5
46.2
46.8
47.4
48.0
48.6
49.2
49.8
50.5
51.1
51.7
52.3
52.9
53.5
54.2
54.8
55.4
56.0
56.6
57.2
57.8
58.5
59.1
59.7
60.3
Proportional Valve Angle (Degrees)

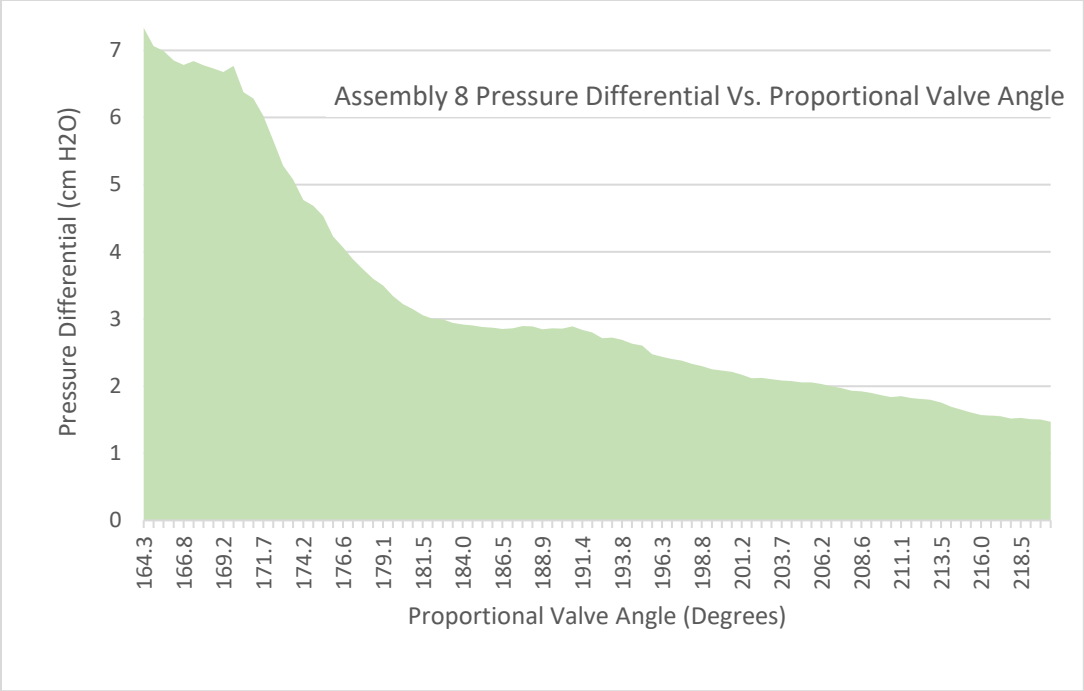

Assembly 3 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0
5
10
15
20
25
30
0.0
6.2
12.3
18.5
24.6
30.8
36.9
43.1
49.2
55.4
61.5
67.7
73.8
80.0
86.2
92.3
98.5
104.6
110.8
116.9
123.1
129.2
135.4
141.5
147.7
153.8
160.0
166.2
172.3
178.5
184.6
Proportional Valve Angle (Degrees)

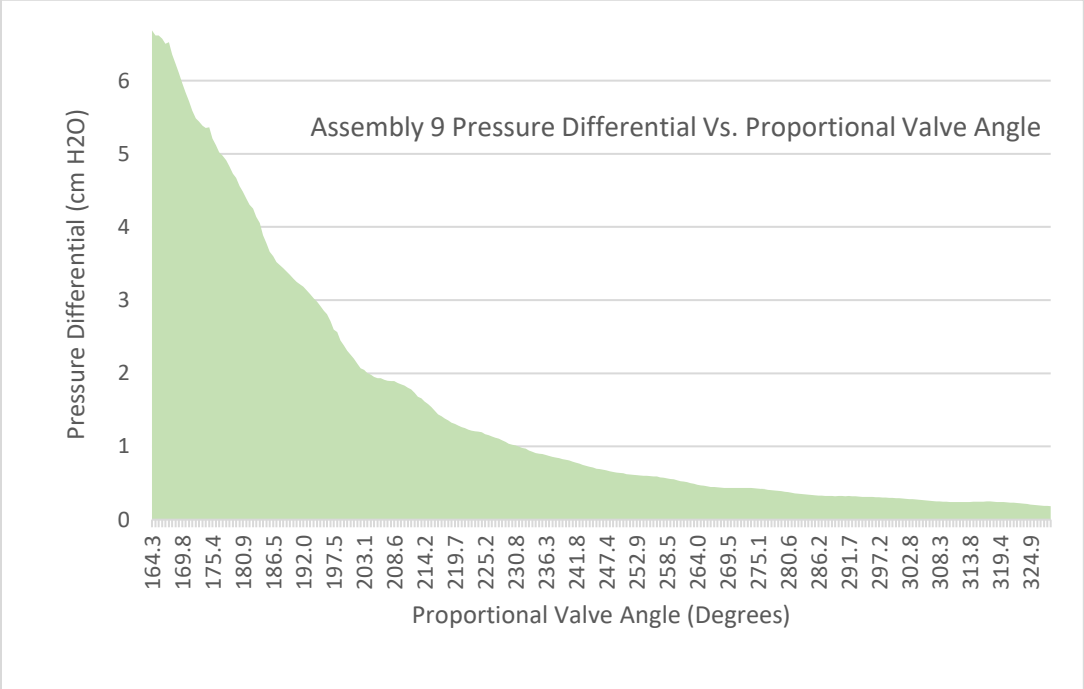

Assembly 4 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0.2
0.15
0.1
0.05
0
0.0
1.8
3.7
5.5
7.4
9.2
11.1
12.9
14.8
16.6
18.5
20.3
22.2
24.0
25.8
27.7
29.5
31.4
33.2
35.1
36.9
38.8
40.6
42.5
44.3
46.2
48.0
49.8
Proportional Valve Angle (Degrees)

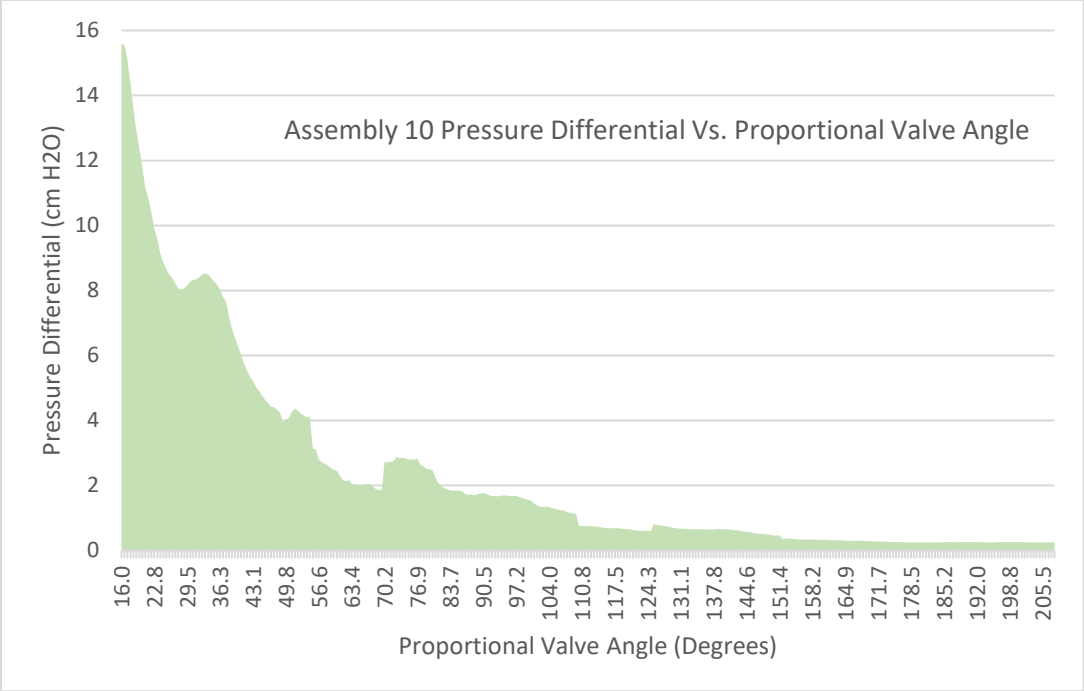

Assembly 5 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
40
35
30
25
20
15
10
5
0
179.7
185.8
192.0
198.2
204.3
210.5
216.6
222.8
228.9
235.1
241.2
247.4
253.5
259.7
265.8
272.0
278.2
284.3
290.5
296.6
302.8
308.9
315.1
321.2
327.4
333.5
339.7
345.8
352.0
358.2
Proportional Valve Angle (Degrees)

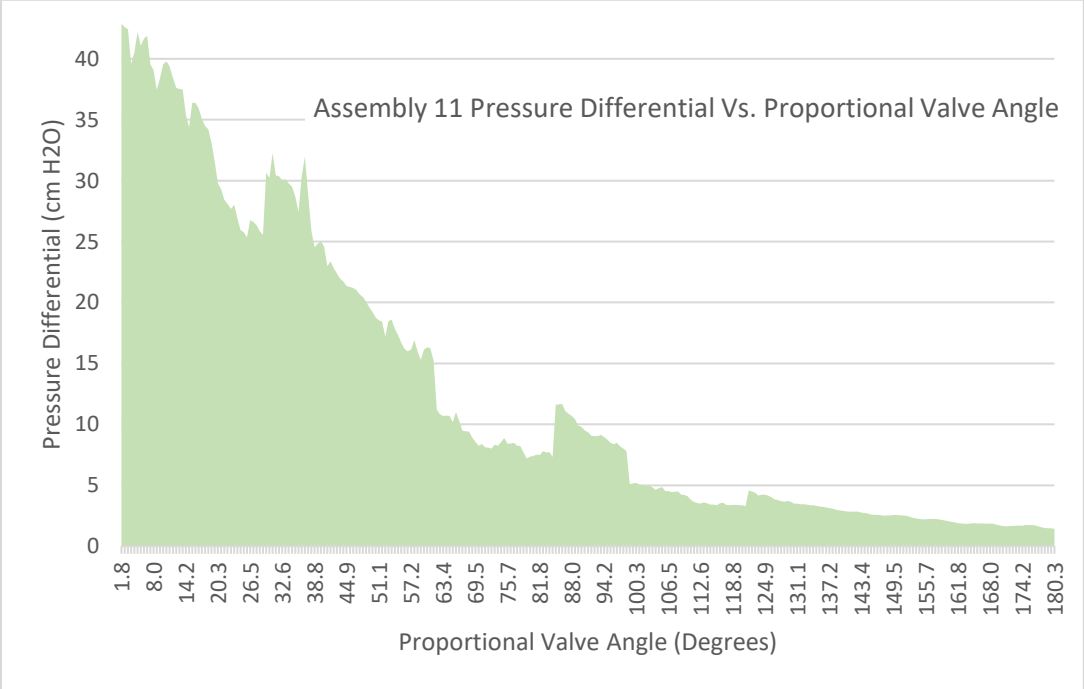

Assembly 6 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
60
50
40
30
20
10
0
181.5
187.7
193.8
200.0
206.8
212.9
219.1
225.2
231.4
237.5
243.7
249.8
256.0
262.2
268.3
274.5
280.6
286.8
292.9
299.1
305.2
311.4
317.5
323.7
329.8
336.0
342.2
348.3
354.5
360.6
Proportional Valve Angle (Degrees)

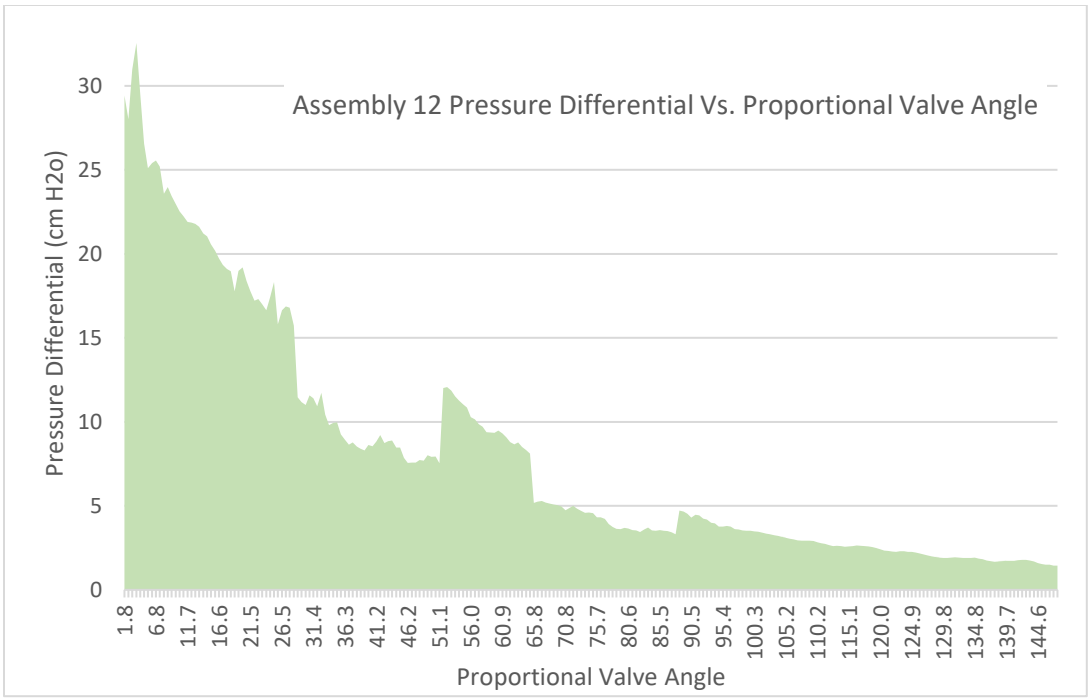

Assembly 7 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
7
6
5
4
3
2
1
0
6.8
12.9
19.1
25.2
31.4
37.5
43.7
49.8
56.0
62.2
68.3
74.5
80.6
86.8
92.9
99.1
105.2
111.4
117.5
123.7
129.8
136.0
142.2
148.3
154.5
160.6
166.8
172.9
179.1
Proportional Valve Angle (Degrees)

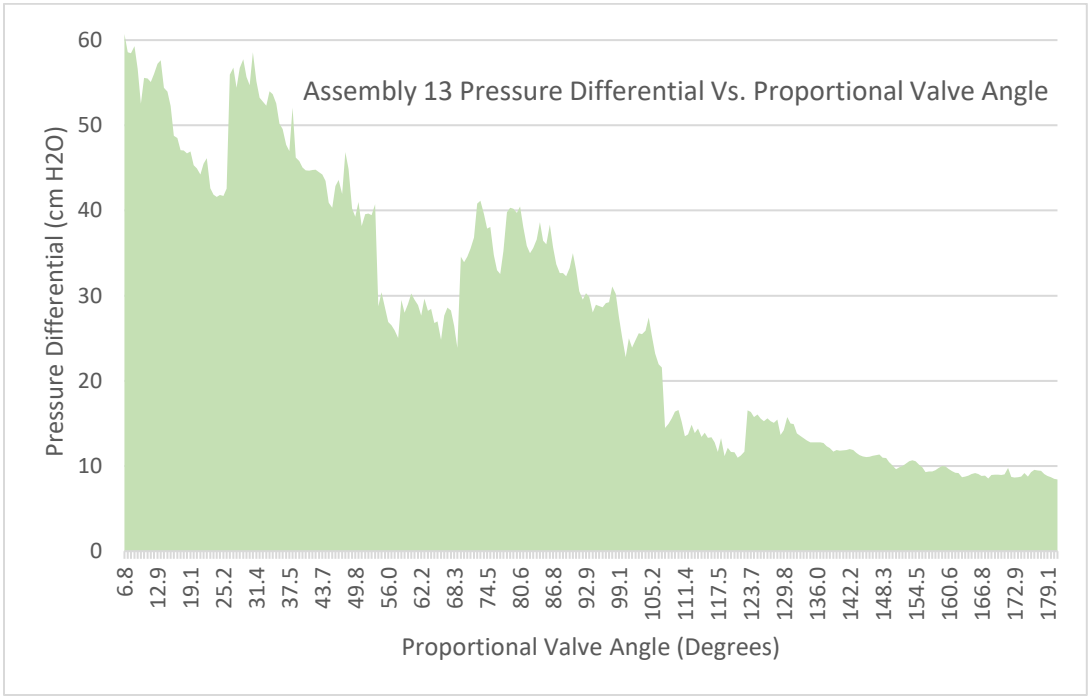

Assembly 8 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0
1
2
3
4
5
6
7
164.3
166.8
169.2
171.7
174.2
176.6
179.1
181.5
184.0
186.5
188.9
191.4
193.8
196.3
198.8
201.2
203.7
206.2
208.6
211.1
213.5
216.0
218.5
Proportional Valve Angle (Degrees)

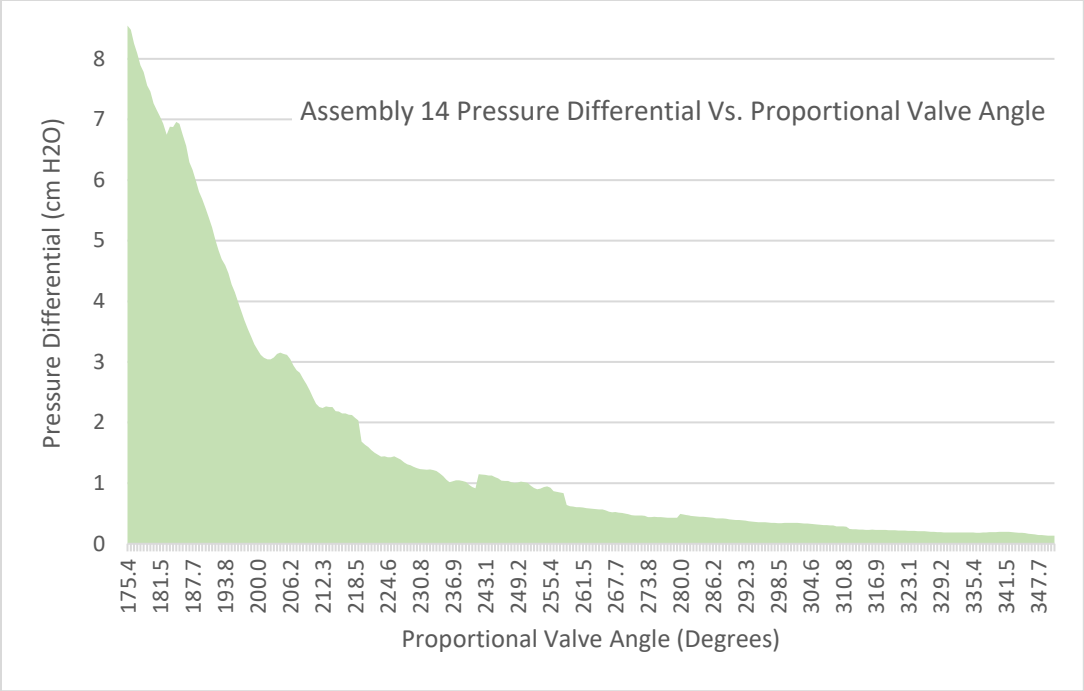

Assembly 9 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0
1
2
3
4
5
6
164.3
169.8
175.4
180.9
186.5
192.0
197.5
203.1
208.6
214.2
219.7
225.2
230.8
236.3
241.8
247.4
252.9
258.5
264.0
269.5
275.1
280.6
286.2
291.7
297.2
302.8
308.3
313.8
319.4
324.9
Proportional Valve Angle (Degrees)

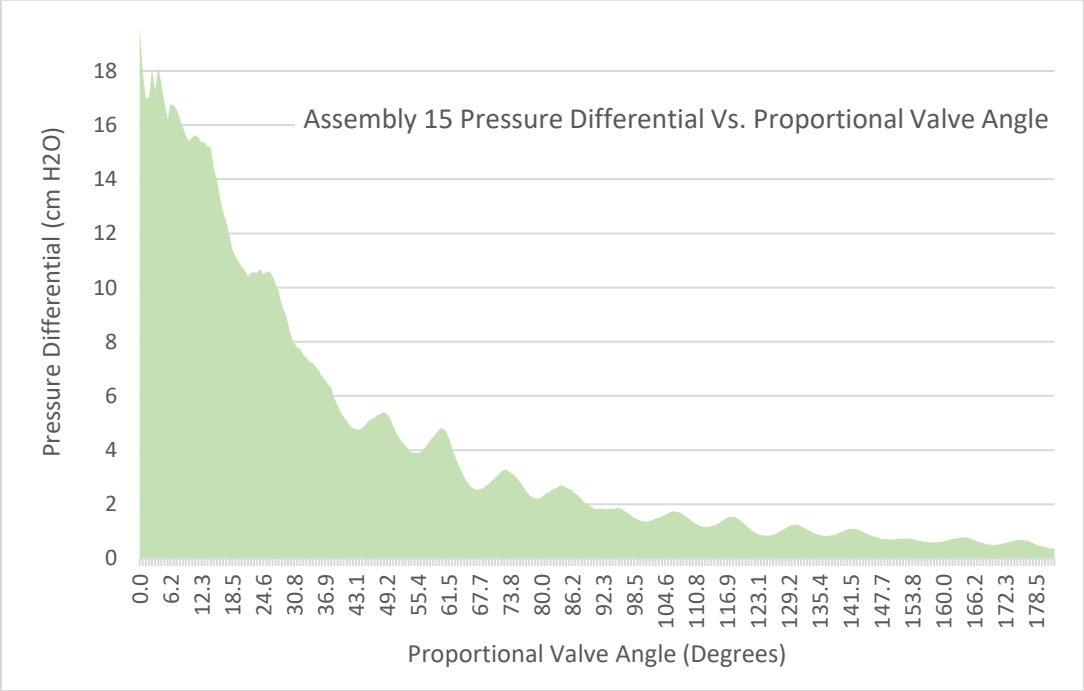

Assembly 10 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
16
14
12
10
8
6
4
2
0
16.0 22.8 29.5 36.3 43.1 49.8 56.6 63.4 70.2 76.9 83.7 90.5 97.2 104.0 110.8 117.5 124.3 131.1 137.8 144.6 151.4 158.2 164.9 171.7 178.5 185.2 192.0 198.8 205.5
Proportional Valve Angle (Degrees)

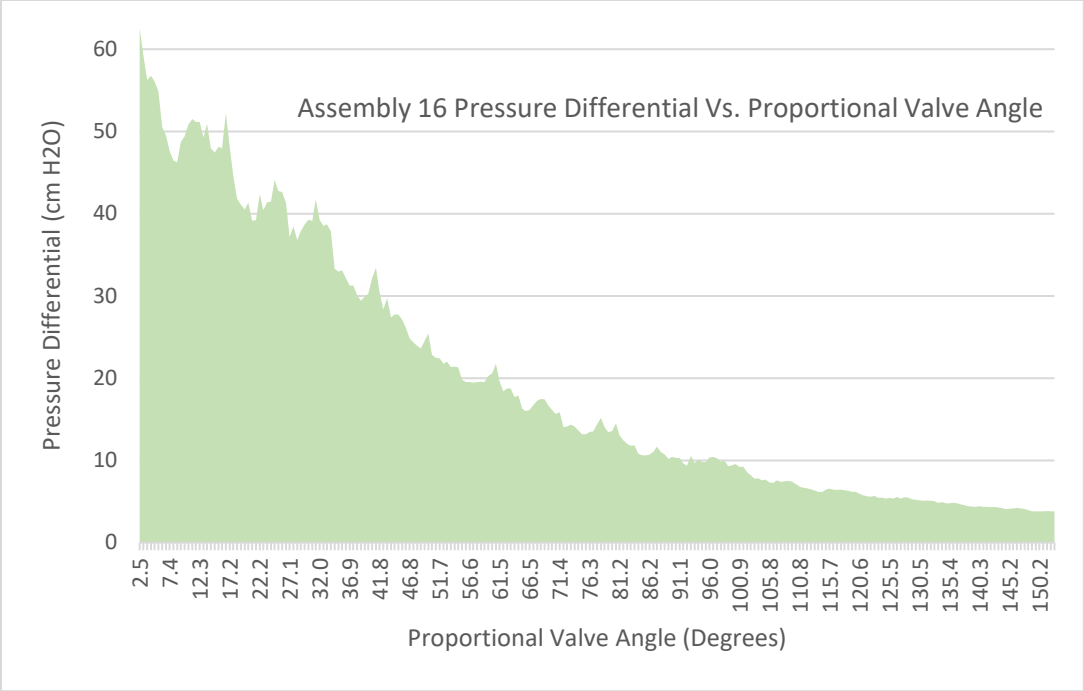

Assembly 11 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
40
35
30
25
20
15
10
5
0
1.8 8.0 14.2 20.3 26.5 32.6 38.8 44.9 51.1 57.2 63.4 69.5 75.7 81.8 88.0 94.2 100.3 106.5 112.6 118.8 124.9 131.1 137.2 143.4 149.5 155.7 161.8 168.0 174.2 180.3
Proportional Valve Angle (Degrees)

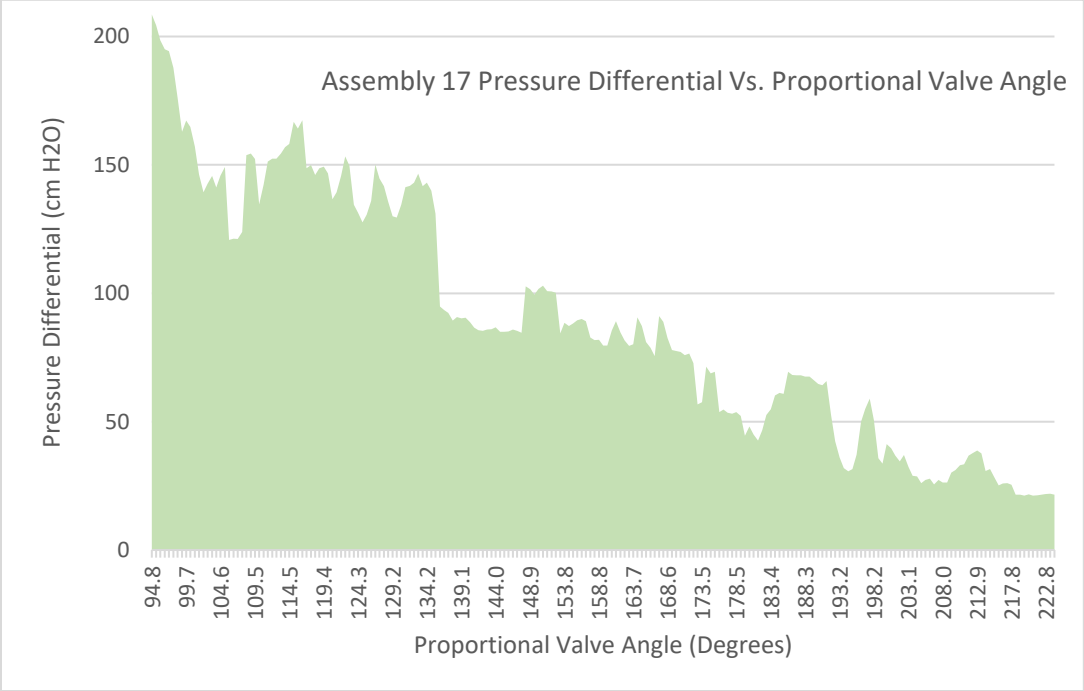

Assembly 12 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2o)
30
25
20
15
10
5
0
1.8
6.8
11.7
16.6
21.5
26.5
31.4
36.3
41.2
46.2
51.1
56.0
60.9
65.8
70.8
75.7
80.6
85.5
90.5
95.4
100.3
105.2
110.2
115.1
120.0
124.9
129.8
134.8
139.7
144.6
Proportional Valve Angle


Assembly 13 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
60
50
40
30
20
10
0
6.8
12.9
19.1
25.2
31.4
37.5
43.7
49.8
56.0
62.2
68.3
74.5
80.6
86.8
92.9
99.1
105.2
111.4
117.5
123.7
129.8
136.0
142.2
148.3
154.5
160.6
166.8
172.9
179.1
Proportional Valve Angle (Degrees)


Assembly 14 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0
1
2
3
4
5
6
7
8
175.4
181.5
187.7
193.8
200.0
206.2
212.3
218.5
224.6
230.8
236.9
243.1
249.2
255.4
261.5
267.7
273.8
280.0
286.2
292.3
298.5
304.6
310.8
316.9
323.1
329.2
335.4
341.5
347.7
Proportional Valve Angle (Degrees)


Assembly 15 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
0
2
4
6
8
10
12
14
16
18
0.0
6.2
12.3
18.5
24.6
30.8
36.9
43.1
49.2
55.4
61.5
67.7
73.8
80.0
86.2
92.3
98.5
104.6
110.8
116.9
123.1
129.2
135.4
141.5
147.7
153.8
160.0
166.2
172.3
178.5
Proportional Valve Angle (Degrees)


Assembly 16 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
60
50
40
30
20
10
0
2.5
7.4
12.3
17.2
22.2
27.1
32.0
36.9
41.8
46.8
51.7
56.6
61.5
66.5
71.4
76.3
81.2
86.2
91.1
96.0
100.9
105.8
110.8
115.7
120.6
125.5
130.5
135.4
140.3
145.2
150.2
Proportional Valve Angle (Degrees)


Assembly 17 Pressure Differential Vs. Proportional Valve Angle
Pressure Differential (cm H2O)
200
150
100
50
0
94.8
99.7
104.6
109.5
114.5
119.4
124.3
129.2
134.2
139.1
144.0
148.9
153.8
158.8
163.7
168.6
173.5
178.5
183.4
188.3
193.2
198.2
203.1
208.0
212.9
217.8
222.8
Proportional Valve Angle (Degrees)

XII. APPENDIX VII

The second configuration of BreathForce system code installed in the Adafruit Feather M0 Bluefruit Microcontroller

```
#include <SD.h>
#include <Wire.h>
#include <TimeLib.h>
#include <SPI.h>
#include "RTClib.h"
RTC_PCF8523 rtc; // runs RTC, if you need to recalibrate run the PCF8523 example
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
#include <genieArduino.h>
Genie genie;
#define RESETLINE 5

//SD card variables
// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

//Connecting Servo
#include <Servo.h>
int servoPin = 12;
Servo myservo;
int openvalve = 0; //WILL CHANGE DEPENDING ON THE DESIGN AND POSITION
int almostclosevalve=170;
int closevalve = 180; //WILL CHANGE DEPENDING ON THE DESIGN AND
POSITION
//All of this code is to add the buttons on the PCB to increment by 1
const byte Button1=6;
const byte Button2=9;
byte CurrentButtonState1=HIGH;
byte PreviousButtonState1=HIGH;
byte CurrentButtonState2=HIGH;
byte PreviousButtonState2=HIGH;
```

```
int sPosition=0;
int sIncrement=1;
int spPosition=0;
int spIncrement=1;



//psi calculations
const int numReadings = 10;
int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average
float GAIN = 47.55;
float bits = 1023.0;
float resolution = 0.00025075;
float psi;
float offset = 1.08;
float Vcc = 3.3;
float calibration = 0.00;
int count;

int PSIhi;          //to help display on 4d display
int PSIlow;

int MEP = 0;
int MIP = 0;
int FMIP =0;
int MPF = 0;
int MPS = 0;
int TargetMEP;
int TargetMIP;
int TrainingPercent;
int MatchPressure;

int MEParray[105];
int MEPsumarray[95];
int E;
float MIParray[105];
float MIPsumarray[95];
int I;
int MParray[105];
int MPsumarray[95];
int J;
int h;
long int PSITimer;
```

```
int R;
int RCP;
int MEPfinalpos;
int MIPfinalpos;
int b;
int TP;
int TEP;

int BreathTime=0;
int TotalTime=0;


//Pressure sensor
int inputPin = A2;

//SD Card
File logfile;
const int chipSelect = 10;
char filename[] = "SADATA00.CSV";   //change patient name here. Format: (First thE
letters of first and last name)00.CSV
char filename2[] = "SADATA00.CSV";
int x;
int xx;                         //helps display previous file
int xxx;

////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup() {
  //for PSI calculation
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;

  //connecting servo
  myservo.attach(servoPin);
  pinMode(Button1, INPUT);  // Manual valve adjustment (clockwise)
  pinMode(Button2, INPUT);  // Manual valve adjustment (counterclockwise)
  }

  //for 4d display to boot up
  Serial1.begin(200000);                          // Serial0 @ 150000 (200K) Baud
  genie.Begin(Serial1);                          // Use Serial1 on the Feather for talking to
the Genie Library, and to the 4D Systems display
  genie.AttachEventHandler(myGenieEventHandler);         // Attach the user function
Event Handler for processing events
  pinMode(RESETLINE, OUTPUT);                        // Set Resetline (D5) on Feather to
Output (4D Arduino Adaptor V2 - Display Reset)
```

```
  digitalWrite(RESETLINE, 0);                    // Reset the Display via Resetline
  delay(100);
  digitalWrite(RESETLINE, 1);                    // unReset the Display via Resetline
  delay (3500);                                  //let the display start up after the reset (This is
important)
  genie.WriteContrast(1); //1=ScrEn on, 0 = scrEn off
  delay(500);

  //Starting Process
  //These messages are being written to the string object on form 0, the index number for
the string object is 23
  genie.WriteStr(23, "Starting up. Please wait.");
  delay(1200);

  //RTC Boot Check
  genie.WriteStr(23, "Checking RealTimeClock status:....");
  delay(1200);

  if (rtc.begin())
    {
     genie.WriteStr(23, "Real Time Clock is working.");
     delay(1200);
    }

    else
    {
     if(! rtc.initialized() || rtc.lostPower())
     {
      genie.WriteStr(23, "RTC is down. Run pcf8523 sketch. System halted.");
      while (1){};
     }
     else {
      genie.WriteStr(23, "RTC read error. Check battery/circuit. System Halted");
      while(1){};
     }
    }

   //SD Card Check
  genie.WriteStr(23, "Initializing SD card:...");
  delay(1200);

  pinMode(10,OUTPUT);
  if (!SD.begin(chipSelect))
  {
    genie.WriteStr(23, "Failed. SD Card not present. Insert SD and press reset. System
Halted.");
```

```
  while (1) {};
 }
 else
 {
  genie.WriteStr(23, "Success.");
  delay(1200);
 }
 genie.WriteStr(23, "Opening Valve");
 myservo.write(openvalve);
 delay(1200);
 genie.WriteStr(23, "Calibrating...");
 delay(1000);
 long int CALendtime=0;
 CALendtime = millis() + 1500;
 while (millis() < CALendtime)
{
 PSIcalculation(); // PSI calculation
}
delay(1200);
calibration=psi;
PSIcalculation;
delay(1200);
genie.WriteStr(23, "Calibrated.");
delay(1200);
genie.WriteStr(23, "Press 'Start' when ready...");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////

void loop()
{
 CurrentButtonState1 = digitalRead(Button1);
    if (CurrentButtonState1 != PreviousButtonState1 && CurrentButtonState1 == LOW)
    {
     sPosition += sIncrement;
     myservo.write(sPosition);
     delay(10);

     }
     PreviousButtonState1 = CurrentButtonState1;
     CurrentButtonState2 = digitalRead(Button2);
     if (CurrentButtonState2 != PreviousButtonState2 && CurrentButtonState2 ==LOW)
     {
       sPosition -= sIncrement;
       myservo.write(sPosition);
```

```
       delay(10);
       }
      PreviousButtonState2 = CurrentButtonState2;
  genie.DoEvents(); //This calls the library each loop to process the responses from the
display.

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////

void myGenieEventHandler(void)
{
  genieFrame Event;
  genie.DequeueEvent(&Event); //This removes the queued event from the buffer to
process it below

  int EVENT_val=0;
  int slider_val=0;

  EVENT_val = genie.GetEventData(&Event);

  //If the cmd received is from a Reported Event (Events triggered from the Events tab of
Workshop4 objects)
  if (Event.reportObject.cmd == GENIE_REPORT_EVENT)
  {
   if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
   {
    if (Event.reportObject.index == 4)
    {
    spPosition=spPosition+spIncrement;
    myservo.write(spPosition);
    delay(1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, spPosition);
    }
    if (Event.reportObject.index == 5)
    {
    spPosition=spPosition-spIncrement;
    myservo.write(spPosition);
    delay(1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, spPosition);
    }
   }

   if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
   {
```

```
  if (Event.reportObject.index == 1)
  {
   TestProgram();
   genie.WriteStr(25, "Idle");
  }
  if (Event.reportObject.index == 3)
  {
   return;
  }
 }

  //To create a new file for the session which is form 1, Userbutton0
  if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
  {
   if (Event.reportObject.index == 2)
   {
    genie.WriteStr(24, "Checking INFO file...");
    delay(1000);
    MakeNewFile();
   }
  }
       //Puts in High (e) and Low (i) psi into file. Then displays results of session and
displays previous session
  if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 12)
   {
    genie.WriteObject(GENIE_OBJ_FORM, 7, 1);
    PullFromNewAndOldFiles();
   }
  }

  //Controls Training% for PSItraining %
  if (Event.reportObject.object == GENIE_OBJ_SLIDER)
  {
   if (Event.reportObject.index == 0)
   {
    slider_val = genie.GetEventData(&Event);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, slider_val);
   TrainingPercent = slider_val;
   }
  }
  //For MIP
  if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 6)
```

```
  {


    myservo.write(almostclosevalve);
    TimerMIP(4, 5, 0);
    genie.WriteStr(0, "Done.");
  }
}

 //For MEP
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 7)
  {
    TimerMEP(4, 5, 0);
    genie.WriteStr(0, "Done. Proceed to results.");
  }
}

//Shows results of MEP/MIP on MEP/MIP result
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 5)
  {
    delay(500);
    genie.WriteObject(GENIE_OBJ_FORM, 4, 1);
    delay(500);
    genie.WriteStr(1, MEP);
    delay(500);
    TargetMEP = MEP * TrainingPercent / 100;
    genie.WriteStr(2, TargetMEP);
    delay(500);
    FMIP = -1 * MIP;
    genie.WriteStr(3, FMIP);
    delay(500);
    TargetMIP = FMIP * TrainingPercent / 100;
    genie.WriteStr(4, TargetMIP);
    delay(500);
    myservo.write(openvalve);
  }
}

//Moves to form 5 to calibrate valve positions
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 2)
```

```
    {
      delay(500);
      genie.WriteObject(GENIE_OBJ_FORM, 5, 1);
      delay(500);
      genie.WriteStr(5, "Recalibrating pressure sensor. Please   wait."); //Keep couple
spaces between please and wait so it fits the string on the display correctly.
      Recalibrating();
      genie.WriteStr(5, "Press the 'Locate PEmax Position'         button when ready.");
    }
   }



   //Calibrates the Valve Position
   h=75;
   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
      {
       if (Event.reportObject.index == 20)
       {
         myservo.write(h);
        TimerMatchPressureMEP();
         genie.WriteStr(5, "Done. Press the 'Locate PImax Position' button when ready.");
       }
      }


      if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
      {
       if (Event.reportObject.index == 21)
       {
         delay(1200);
         genie.WriteStr(5, "Recalibrating pressure sensor and servo motor. Please wait.");
         myservo.write(h);
         delay(1200);
         Recalibrating();
         delay(1200);
         TimerMatchPressureMIP();
         genie.WriteStr(5, "Done, Proceed to the training page.");
       }
      }



      if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
   {
    if (Event.reportObject.index == 3)
```

```
  {
   delay(500);
   genie.WriteObject(GENIE_OBJ_FORM, 6, 1);
   delay(500);
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 9, TargetMEP);
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 7, abs(TargetMIP));
   genie.WriteStr(22, "Chose the number of minutes to          train. Then press start.");
  }
 }

  //PSI Training Timers
  //There are options for 1, 2, 3, 4, or 5 minutes of training. Each button needs to identify
how much time there is and write it the LED on the training page.
  //The PSITimer is in milliseconds with an extra half second added on.
  if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 17)
   {
    PSITimer = 300500;
    count =5;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
   }
  }

   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 16)
   {
    PSITimer = 240500;
    count =4;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
   }
  }

   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 15)
   {
    PSITimer = 180500;
    count =3;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
   }
  }

  if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
```

```
    if (Event.reportObject.index == 14)
    {
      PSITimer = 120500;
      count=2;
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
    }
   }

   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
   {
    if (Event.reportObject.index == 13)
    {
      PSITimer = 60500;
      count=1;
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
    }
   }

    //For PSI training screen. Write to excel and give option to end or continue.
   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
   {
    if (Event.reportObject.index == 11)
    {
      logfile = SD.open(filename, FILE_WRITE);
      TimerAndRecordPSI ();
      logfile.close();
      delay(500);
      genie.WriteStr(22, "Done.");
      delay(1000);
      genie.WriteStr(22, "Click results to see results. Click start to train again.");
    }
   }
  }
}
void MakeNewFile()
{

  genie.WriteStr(24, "Patient Info:...");
  delay(1200);
  logfile = SD.open("INFO.TXT");
  if (logfile.available())
  {
    genie.WriteStr(24, "File exists.");
  }
  else
  {
```

```
  genie.WriteStr(24, "File does not exist.");
  delay(1200);
  genie.WriteStr(24, "Check SD Card");
  delay(3600);
  return;
}
filename[0] = logfile.read();
filename[1] = logfile.read();
filename[2] = logfile.read();
filename[3] = logfile.read();
filename[4] = logfile.read();
filename[5] = logfile.read();
filename2[0] = filename[0];
filename2[1] = filename[1];
filename2[2] = filename[2];
filename2[3] = filename[3];
filename2[4] = filename[4];
filename2[5] = filename[5];
logfile.close();
genie.WriteStr(24, filename);
delay(1000);

genie.WriteStr(24, "Making new file:...");
for (uint8_t i = 1; i < 100; i++) {
  filename[6] = i / 10 + '0';
  filename[7] = i % 10 + '0';
  if (! SD.exists(filename)) {
    DateTime now = rtc.now();
    logfile = SD.open(filename, FILE_WRITE);
    logfile.print(",,");
    logfile.print("Date(D/M/Y)/Time:,");          //A1:B1
    logfile.print(now.day(), DEC);
    logfile.print("/");
    logfile.print(now.month(), DEC);
    logfile.print("/");
    logfile.print(now.year(), DEC);
    logfile.print(",");
    logfile.print(now.hour(), DEC);
    logfile.print(":");
    logfile.print(now.minute(), DEC);
    logfile.print(":");
    logfile.println(now.second(), DEC);
    logfile.print(",,");
    logfile.print("Training Time:");                //A2:B2
    logfile.print(",");
    logfile.print("       ");
```

```
    logfile.print(",");
    logfile.println("seconds");
    logfile.print(",,");
    logfile.print("PEmax:");                  //A2:B2
    logfile.print(",");
    logfile.print("       ");
    logfile.print(",");
    logfile.println("cm H2O");
    logfile.print(",,");
    logfile.print("PImax:");                  //A3:B3
    logfile.print(",");
    logfile.print("       ");
    logfile.print(",");
    logfile.println("cm H2O");
    logfile.print(",,");
    logfile.print("Training %");
    logfile.print(",");
    logfile.print("     ");
    logfile.print(",");
    logfile.println("%");
    logfile.print(",,");
    logfile.print("# of points:");               //A4:B4
    logfile.print(",");
    logfile.print( "=COUNT(B:B)");
    logfile.print(",");
    logfile.println("points");
    logfile.print(",,");
    logfile.print("Time Duration");               //A5:B5
    logfile.print(",");
    logfile.print("=(MAX(A:A)-MIN(A:A))/1000");
    logfile.print(",");
    logfile.println("seconds");
    logfile.print(",,");
    logfile.print("Points per second:");          //A6:B6
    logfile.print(",");
    logfile.print("=D6/D7");
    logfile.print(",");
    logfile.println("points/s");
    logfile.print("Time(ms)");     //A8:B8
    logfile.print(",");
    logfile.println("cm H2O");
    logfile.close();
    break; // leave the loop!;
  }
}
```

```
  delay(1000);
  genie.WriteStr(24, filename);
  delay(1000);

  genie.WriteStr(24, "Checking old file...");
  delay(1000);
  filename2[6] = filename[6];
  filename2[7] = filename[7];
  xx = filename2[6] - '0';   //Renames filename to go back 1 previous session to display on
result scrEn
  xxx = filename2[7] - '0';
  x = (xx * 10) + xxx - 1;
  filename2[6] = x / 10 + '0';
  filename2[7] = x % 10 + '0';
  logfile = SD.open(filename2, FILE_READ);
  logfile.seek(20);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  MEP = logfile.readStringUntil(',').toInt();
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  MIP = logfile.readStringUntil(',').toInt();
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  TrainingPercent = logfile.readStringUntil(',').toInt();
  logfile.close();

//writing the previous values to form 7 led digits
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, MEP);
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, abs(MIP));
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 1, TrainingPercent); //Previous %
training display in form 2

  genie.WriteStr(24, filename2);
  delay(1000);

  genie.WriteStr(24, "Proceed to PE/PImax.");
  delay(100);
```

```
}
int PSIcalculation() {
  total = total - readings[readIndex];        //psi readings + calculations
  readings[readIndex] = analogRead(inputPin);
  total = total + readings[readIndex];
  readIndex = readIndex + 1;
  if (readIndex >= numReadings) {
    readIndex = 0;
  }
  psi = total / numReadings;
  psi = psi / bits;             //bits = 1023.0
  psi = psi * Vcc;              //Vcc=3.3
  psi = psi - offset;           //offset = 2.5
  psi = psi / resolution;       //resolution = 0.0002584
  psi = psi / GAIN;             //GAIN
  psi = psi * 1.36;        //1.36 is conversion from 1 mmHg = 1.36 cmH2O
  psi = psi - calibration;
}
void PullFromNewAndOldFiles() {
  delay(500);
  logfile = SD.open(filename, O_RDWR);
  logfile.seek(20);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(TotalTime/1000);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(MEP);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(FMIP);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(TrainingPercent);
  logfile.close();

  genie.WriteObject(GENIE_OBJ_USERBUTTON, 12, 1);
  delay(1000);
```

```
  genie.WriteStr(6, filename);
  delay(500);

  logfile = SD.open(filename, FILE_READ);
  logfile.seek(20);
  genie.WriteStr(7, logfile.readStringUntil(','));    //DATE
  delay(500);
  genie.WriteStr(8, logfile.readStringUntil(','));    //TIME
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(9, logfile.readStringUntil(','));    //Total Time
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(11, logfile.readStringUntil(','));    //MEP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(12, logfile.readStringUntil(','));    //MIP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(13, logfile.readStringUntil(','));    //% Training
  delay(500);
  logfile.close();

  xx = filename[6] - '0';   //Renames filename to go back 1 previous session to display on
result scrEn
  xxx = filename[7] - '0';
  x = (xx * 10) + xxx - 1;
  filename[6] = x / 10 + '0';
  filename[7] = x % 10 + '0';
  genie.WriteStr(14, filename);
  delay(500);

  logfile = SD.open(filename, FILE_READ);
  logfile.seek(20);
  genie.WriteStr(15, logfile.readStringUntil(','));
  delay(500);
  genie.WriteStr(16, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
```

```
  logfile.readStringUntil(',');
  genie.WriteStr(17, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(19, logfile.readStringUntil(','));    //MEP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(20, logfile.readStringUntil(','));    //MIP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(21, logfile.readStringUntil(','));    //% Training
  delay(500);
  logfile.close();
}
void Recalibrating() {
  long int RCendtime = 0;
  R = 0;
  RCP = 0;
  RCendtime = millis() + 5500;
  while (millis() < RCendtime)
  {
    PSIcalculation();
  }
}
void TestProgram()

{
  long int TPendtime = 0;
  TP = 0;
  TEP = 0;
  TPendtime = millis() + 5500;
  while (millis() < TPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 13, abs(psi));
    genie.WriteStr(25, "Measuring");
  }

}
void TimerAndRecordPSI () {
```

```
long int Pendtime = 0;
int scount=0;
int smove=0;
while (count > 0)
{
  BreathTime=PSITimer;
  TotalTime=TotalTime+BreathTime;
  Pendtime = millis() + PSITimer;
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer / 1000);
  while (millis() < Pendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, (Pendtime - millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 8, abs(psi));
    if (psi > (TargetMEP * 0.90)) {
      genie.WriteObject(GENIE_OBJ_USER_LED, 1, 1);
    }
    else {
      genie.WriteObject(GENIE_OBJ_USER_LED, 1, 0);
    }

    if (psi < (TargetMIP * 0.90)) {
      genie.WriteObject(GENIE_OBJ_USER_LED, 0, 1);
```

```
}
else {
  genie.WriteObject(GENIE_OBJ_USER_LED, 0, 0);
}

if (psi>0 && psi<=1.1*TargetMEP)
{
  myservo.write(MEPfinalpos);
  delay(100);
  genie. WriteStr(22, "Normal Expiratory Position");
}
else if (psi<0 && psi>=1.1*TargetMIP)
{
  myservo.write(MIPfinalpos);
  delay(100);
  genie.WriteStr(22, "Normal Inspiratory Position");
}

else if (psi>1.1*TargetMEP)
{
  myservo.write(MEPfinalpos-10);
  delay(100);
  genie.WriteStr(22, "Too much expiratory pressure.");
```

```
      }
      else if (psi < 1.1*TargetMIP)
      {
        myservo.write(MIPfinalpos-10);
        delay(100);
        genie.WriteStr(22, "Too much inspiratory pressure.");
      }
      else
      {
        genie.WriteStr(22, "Transitioning..");
      }



      logfile.print (millis());
      logfile.print (",");
      logfile.println(psi);
    }
    count = count - count ; //supposed to be count - 1 when doing longer consecutive
sessions.
  }
}
void TimerMEP (int a, int b, int c) {
  genie.WriteStr(c, "Starting...");
  delay(1000);
  genie.WriteStr(c, "3...");
  delay(1000);
```

```
  genie.WriteStr(c, "2...");
  delay(1000);
  genie.WriteStr(c, "1...");
  delay(1000);
  genie.WriteStr(c, "Now.");
  delay(500);

  long int MEPendtime = 0;
  E = 0;
  MEP = 0;
  MEPendtime = millis() + 5500;
  while (millis() < MEPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, a, (MEPendtime - millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, b, abs(psi));
    MEParray[E] = psi;
    E++;
  }
  for (E = 0; E < 96; E++)
  {
    MEPsumarray[E] = (MEParray[E] + MEParray[E + 1] + MEParray[E + 2] +
MEParray[E + 3] + MEParray[E + 4] + MEParray[E + 5] + MEParray[E + 6] +
MEParray[E + 7] + MEParray[E + 8] + MEParray[E + 9]) / 10;
    delay(5);
  }
  int idx;
  for (byte idx = 0; idx != 96; idx++)
  {
    if (MEPsumarray[idx] > MEP) {
      MEP = max(MEPsumarray[idx], MEP);

    }
    delay(5);
  }
}
void TimerMIP (int a, int b, int c) {
  genie.WriteStr(c, "Starting...");
  delay(1000);
  genie.WriteStr(c, "3...");
  delay(1000);
  genie.WriteStr(c, "2...");
  delay(1000);
  genie.WriteStr(c, "1...");
  delay(1000);
  genie.WriteStr(c, "Now.");
```

```
  delay(500);

  long int MIPendtime = 0;
  I = 0;
  MIP = 0;
  MIPendtime = millis() + 5500;
  while (millis() < MIPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, a, (MIPendtime - millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, b, abs(psi));
    MIParray[I] = psi;
    I++;
  }
  for (I = 0; I < 96; I++)
  {
    MIPsumarray[I] = (MIParray[I] + MIParray[I + 1] + MIParray[I + 2] + MIParray[I +
3] + MIParray[I + 4] + MIParray[I + 5] + MIParray[I + 6] + MIParray[I + 7] +
MIParray[I + 8] + MIParray[I + 9]) / 10;
    delay(5);
  }
  int idy;
  for (byte idy = 0; idy != 96; idy++)
  {
    if (MIPsumarray[idy] < MIP) {
      MIP = min(MIPsumarray[idy], MIP);

          }
    delay(5);
  }
}
void TimerMatchPressureMEP() {
  genie.WriteStr(5, "Starting...");
  delay(1000);
  genie.WriteStr(5, "3...");
  delay(1000);
  genie.WriteStr(5, "2...");
  delay(1000);
  genie.WriteStr(5, "1...");
  delay(1000);
  genie.WriteStr(5, "Now.");
  delay(500);
 long int MEPMatchPressureEndTime=0;
  J=0;
  MPF=0;
  MEPMatchPressureEndTime=millis() + 105500;
```

```
  while (millis() < MEPMatchPressureEndTime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, (MEPMatchPressureEndTime -
millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, abs(psi));
    delay(1000);

  if (abs(psi)<= .9*TargetMEP)
  {
    h=h+1;
    myservo.write(h);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 26, h);
    MEPfinalpos=h;
  }
  else if (abs(psi) >= .9*TargetMEP)
  {
    break;

  }
  else
  {
    break;
  }
  }
}
void TimerMatchPressureMIP() {
  genie.WriteStr(5, "Starting...");
  delay(1000);
  genie.WriteStr(5, "3...");
  delay(1000);
  genie.WriteStr(5, "2...");
  delay(1000);
  genie.WriteStr(5, "1...");
  delay(1000);
  genie.WriteStr(5, "Now.");
  delay(500);
  long int MIPMatchPressureEndTime=0;
  J=0;
  MPS=0;
  MIPMatchPressureEndTime=millis() + 105500;
  while (millis() < MIPMatchPressureEndTime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, (MIPMatchPressureEndTime -
millis()) / 1000);
```

```
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, abs(psi));
  delay(1000);

 if (psi> .9*TargetMIP)
 {
  h=h+1;
  myservo.write(h);
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 26, h);
  MIPfinalpos=h;
 }
 else if (psi <= .9*TargetMIP)
 {
  break;

 }
 else
 {
  break;
 }
 }
}
```

XIII. APPENDIX VIII

BreathForce Training Instructions

1. Plug in the device and wait for the screen to say “Press ‘Start’ When Ready…”.
    a. If the device says
        i. "Failed. SD Card not present. Insert SD and press reset. System Halted."
            1. Unplug the device, insert and SD card, and plug the device back in.
        ii. "RTC read error. Check battery/circuit. System Halted"
            1. Unplug the device and check if the processor battery has died and needs to be replaced. Replace the battery and run the Real Time Clock Set-Up Program titled “pcf8523” in Arduino IDE.
        iii. "RTC is down. Run pcf8523 sketch. System halted."
            1. Run the Real Time Clock Set-Up Program titled “pcf8523” in Arduino IDE.
2. Press Start. This will take you to the Main Directory of the device.
3. Choose “Make File Session” on the Main Directory
4. Press the Start button and wait while a new file is generated to save the data from the session.
    a. If the device says the file does not exist and to check the SD card, make sure:
        i. The correct SD card is being used.
        ii. The text.info file contains the correct origin file name.
5. When the device says "Proceed to PE/PImax." choose “Back to Main Directory”.
6. Choose “PE/PImax”.
7. Use the slider to select the training percent for the session and click “Next”.
8. When ready, choose “PImax”. The device will countdown, and when it displays “Now”, inhale as forcefully as you can while the timer counts down from five seconds.
9. When ready, choose “PEmax”. The device will countdown, and when it displays “Now”, exhale as forcefully as you can while the timer counts down from five seconds.
10. When finished, press “Results”

11. The PImax and PEmax will be displayed as well as the Target PImax and Target PEmax values calculated for the training session.
12. Click “Back to Main Directory” when finished viewing the results.
13. On the Main Directory, choose “Locate Valve Position”.
14. To locate the valve positions for training, first press “Locate PEmax Position”, and wait as the device counts down. When the device says “Now”, breath as you would during the training session in and out of the valve until it says "Done. Press the 'Locate PImax Position' button when ready."
15. When ready, press “Locate PImax Position”, and wait as the device recalibrates and counts down. When the device says “Now”, breath as you would during the training session in and out of the valve until it says "Done, Proceed to the training page."
16. Choose “Back to Main Directory”.
17. On the Main Directory, choose “Training”.
18. Choose the number of minutes you want to train, and press “Start”.
19. Breath in and out of the valve while trying to match the Target PI/PEmax values displayed until the timer runs out.
    a. The LED lights above the Target PI/PEmax values will turn on when you are within 10% of the Target PI/PEmax pressures.
20. When the timer expires, you may:
    a. Start another training session by choosing the number of minutes you wish to train and pressing “Start” again
    b. End the session by pressing “Results”
21. If you choose “Results” the device will display highlights from the current and previous sessions.
22. Turn off the device.

BreathForce Testing Program

To test the pressure at various servo positions use the following process:

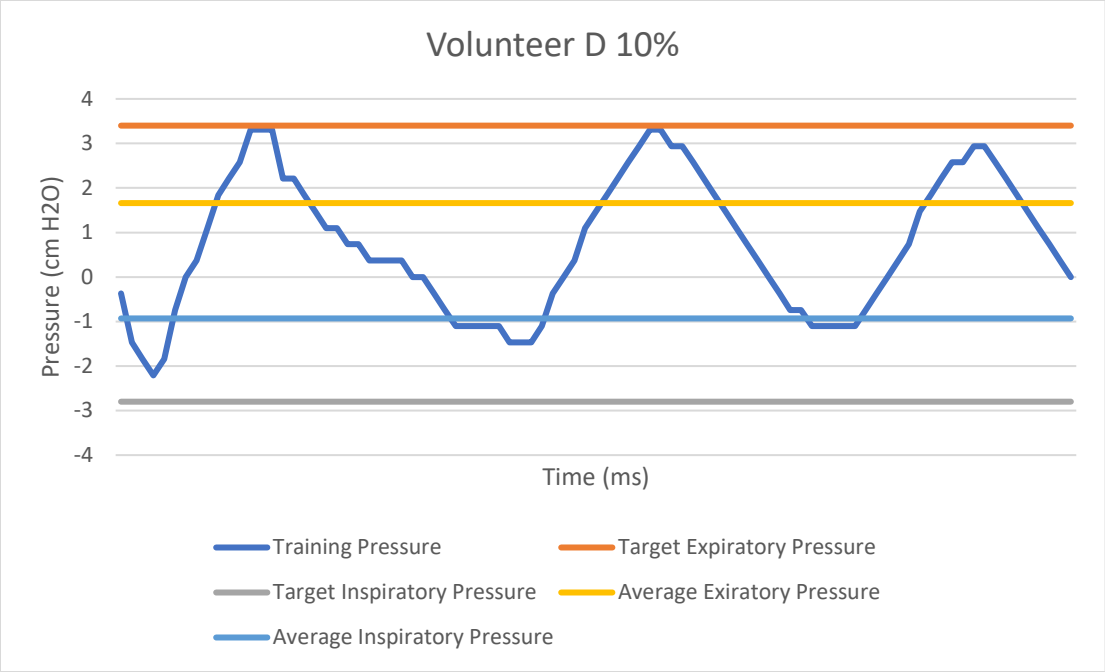
1. Plug in the device and wait for the screen to say “Press ‘Start’ When Ready…”.
    a. If the device says
        i. "Failed. SD Card not present. Insert SD and press reset. System Halted."
            1. Unplug the device, insert and SD card, and plug the device back in.
        ii. "RTC read error. Check battery/circuit. System Halted"
            1. Unplug the device and check if the processor battery has died and needs to be replaced. Replace the battery and run the Real Time Clock Set-Up Program titled “pcf8523” in Arduino IDE.
        iii. "RTC is down. Run pcf8523 sketch. System halted."
            1. Run the Real Time Clock Set-Up Program titled “pcf8523” in Arduino IDE.
2. Press “Test Program”.
3. To move the servo and open or close the valve, press “Open” or “Close”.
4. To measure the pressure, press the measure button.
5. As the measurement is occuring, the device will display “Measuring”. During this time, the servo motor cannot be adjusted.
6. When the device displays “Idle” the servo motor position can be adjusted again.
7. To suspend the measuring process while the display shows “Measuring”, press the stop button.

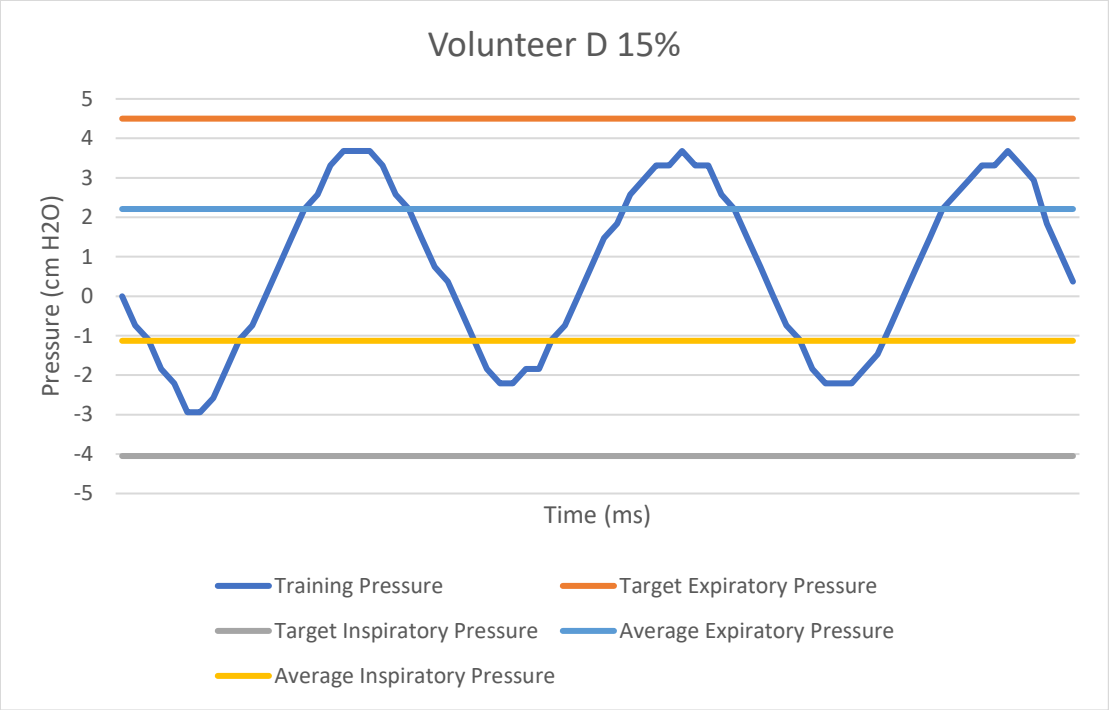
XIV. APPENDIX VIIII

The summary tables and respiratory graphs of Volunteers D, E, and F:

Volunteer D

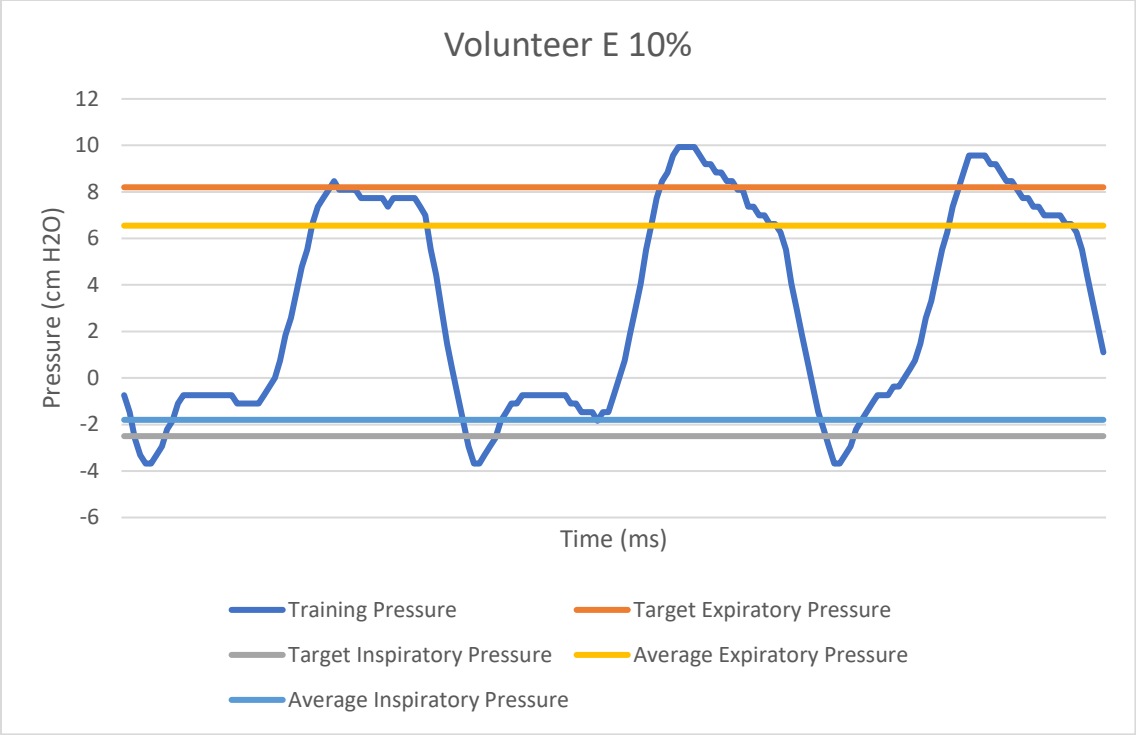
Training Load (%)	10	15	20
PEmax (cm H_2O)	34	30	N/A
PImax (cm H_2O)	-28	-27	N/A
Training Time (seconds)	120	120	N/A
Target Expiration Pressure (cm H_2O)	3.40	4.50	N/A
Average Expiration Pressure (cm H_2O)	1.66	2.21	N/A
Target Inspiration Pressure (cm H_2O)	-2.80	-4.05	N/A
Average Inspiration Pressure (cm H_2O)	-0.93	-1.13	N/A
Expiratory Valve Position (degrees)	157	178	N/A
Inspiratory Valve Position (degrees)	178	178	N/A

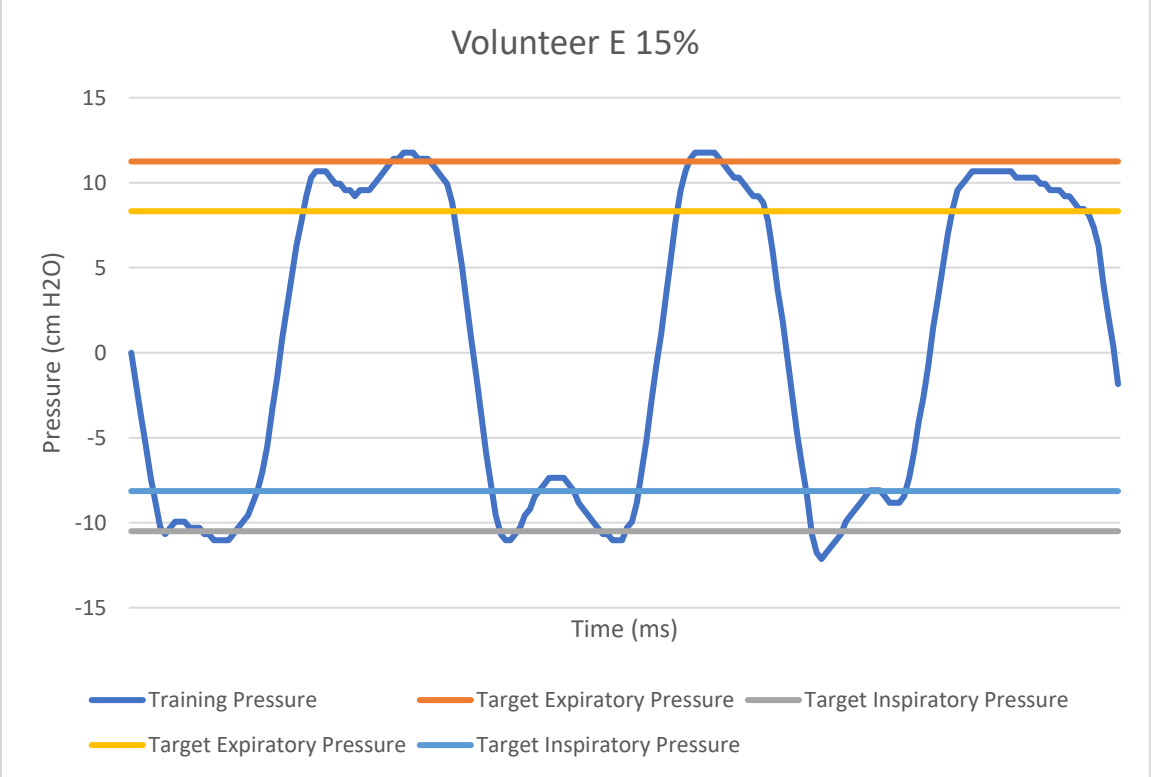

Volunteer D 10%
Pressure (cm H2O)
4
3
2
1
0
-1
-2
-3
-4
Time (ms)
Training Pressure
Target Expiratory Pressure
Target Inspiratory Pressure
Average Exiratory Pressure
Average Inspiratory Pressure



Volunteer E

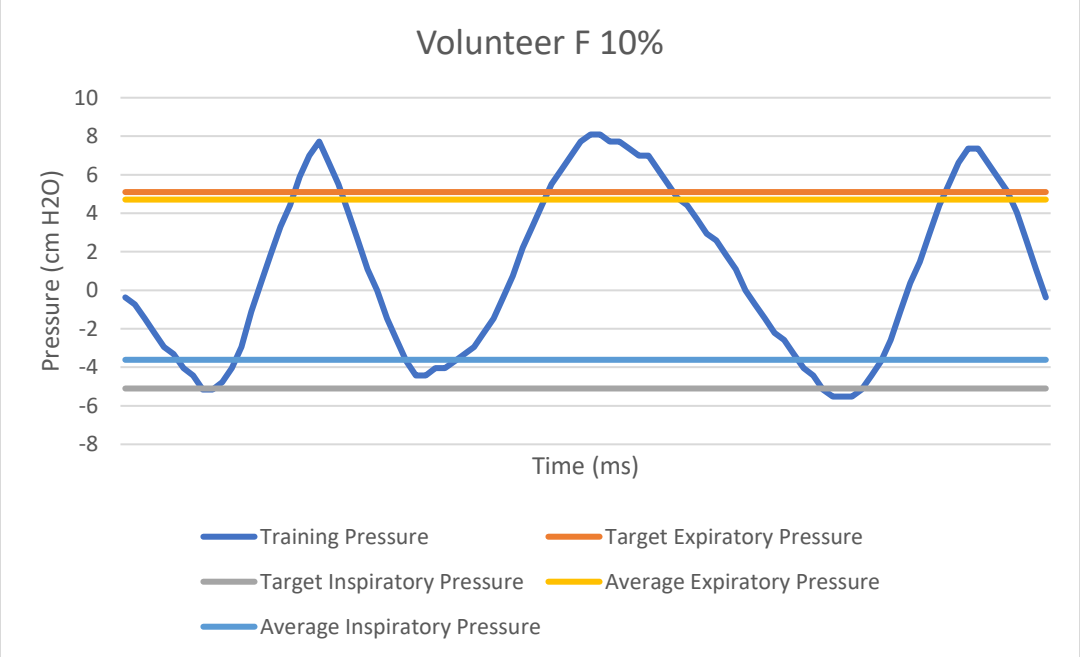
Training Load (%)	10	15	20
PEmax (cm H_2O)	82	75	N/A
PImax (cm H_2O)	-25	-70	N/A
Training Time (seconds)	120	120	N/A
Target Expiration Pressure (cm H_2O)	8.20	11.25	N/A
Average Expiration Pressure (cm H_2O)	6.55	8.32	N/A
Target Inspiration Pressure (cm H_2O)	-2.50	-10.50	N/A
Average Inspiration Pressure (cm H_2O)	-1.8	-8.14	N/A
Expiratory Valve Position (degrees)	177	178	N/A
Inspiratory Valve Position (degrees)	101	178	N/A


Volunteer E 10%
12
10
8
6
4
2
0
-2
-4
-6
Pressure (cm H2O)
Time (ms)
Training Pressure
Target Expiratory Pressure
Target Inspiratory Pressure
Average Expiratory Pressure
Average Inspiratory Pressure


Volunteer E 15%
15
10
5
0
-5
-10
-15
Pressure (cm H2O)
Time (ms)
Training Pressure
Target Expiratory Pressure
Target Inspiratory Pressure
Target Expiratory Pressure
Target Inspiratory Pressure

Volunteer F

Training Load (%)	10	15	20
PEmax (cm H_2O)	51	N/A	N/A
PImax (cm H_2O)	-51	N/A	N/A
Training Time (seconds)	120	N/A	N/A
Target Expiration Pressure (cm H_2O)	5.10	N/A	N/A
Average Expiration Pressure (cm H_2O)	4.71	N/A	N/A
Target Inspiration Pressure (cm H_2O)	-5.10	N/A	N/A
Average Inspiration Pressure (cm H_2O)	-3.61	N/A	N/A
Expiratory Valve Position (degrees)	178	N/A	N/A
Inspiratory Valve Position (degrees)	178	N/A	N/A



XV. APPENDIX X

The final configuration of the BreathForce system code installed in the Adafruit Feather M0 Bluefruit Microcontroller

```
#include <SD.h>
#include <Wire.h>
#include <TimeLib.h>
#include <SPI.h>
#include "RTClib.h"
RTC_PCF8523 rtc; // runs RTC, if you need to recalibrate run the PCF8523 example
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
#include <genieArduino.h>
Genie genie;
#define RESETLINE 5

//SD card variables
// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

//Connecting Servo
#include <Servo.h>
int servoPin = 12;
Servo myservo;
int openvalve = 0; //WILL CHANGE DEPENDING ON THE DESIGN AND POSITION
int almostclosevalve=170;
```

```
int closevalve = 180; //WILL CHANGE DEPENDING ON THE DESIGN AND POSITION
//All of this code is to add the buttons on the PCB to increment by 1
const byte Button1=6;
const byte Button2=9;
byte CurrentButtonState1=HIGH;
byte PreviousButtonState1=HIGH;
byte CurrentButtonState2=HIGH;
byte PreviousButtonState2=HIGH;
int sPosition=0;
int sIncrement=1;
int spPosition=0;
int spIncrement=1;
//psi calculations
const int numReadings = 10;
int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average
float GAIN = 47.55;
float bits = 1023.0;
float resolution = 0.00025075;
float psi;
float offset = 1.08;
float Vcc = 3.3;
float calibration = 0.00;
int count;
int PSIhi;          //to help display on 4d display
int PSIlow;
int MEP = 0;
int MIP = 0;
int FMIP =0;
int MPF = 0;
```

```
int MPS = 0;
int TargetMEP;
int TargetMIP;
int TrainingPercent;
int MatchPressure;
int MEParray[105];
int MEPsumarray[95];
int E;
float MIParray[105];
float MIPsumarray[95];
int I;
int MParray[105];
int MPsumarray[95];
int J;
int h;
long int PSITimer;
int R;
int RCP;
int MEPfinalpos;
int MIPfinalpos;
int b;
int TP;
int TEP;
int BreathTime=0;
int TotalTime=0;
//Pressure sensor
int inputPin = A2;
//SD Card
File logfile;
const int chipSelect = 10;
char filename[] = "SADATA00.CSV";   //change patient name here. Format: (First thE letters of first and last name)00.CSV
char filename2[] = "SADATA00.CSV";
```

```
int x;
int xx;                    //helps display previous file
int xxx;

////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup() {
  //for PSI calculation
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  //connecting servo
  myservo.attach(servoPin);
  pinMode(Button1, INPUT);  // Manual valve adjustment (clockwise)
  pinMode(Button2, INPUT);  // Manual valve adjustment (counterclockwise)
  }
  //for 4d display to boot up
  Serial1.begin(200000);                    // Serial0 @ 150000 (200K) Baud
  genie.Begin(Serial1);                     // Use Serial1 on the Feather for talking to the Genie Library, and to the 4D Systems display
  genie.AttachEventHandler(myGenieEventHandler);         // Attach the user function Event Handler for processing events
  pinMode(RESETLINE, OUTPUT);                    // Set Resetline (D5) on Feather to Output (4D Arduino Adaptor V2 - Display Reset)
  digitalWrite(RESETLINE, 0);                    // Reset the Display via Resetline
  delay(100);
  digitalWrite(RESETLINE, 1);                    // unReset the Display via Resetline
  delay (3500);                             //let the display start up after the reset (This is important)
  genie.WriteContrast(1); //1=ScrEn on, 0 = scrEn off
  delay(500);
  //Starting Process
  //These messages are being written to the string object on form 0, the index number for the string object is 23
  genie.WriteStr(23, "Starting up. Please wait.");
```

```
 delay(1200);
 //RTC Boot Check
 genie.WriteStr(23, "Checking RealTimeClock status:....");
 delay(1200);
 if (rtc.begin())
  {
   genie.WriteStr(23, "Real Time Clock is working.");
   delay(1200);
  }
  else
  {
   if(! rtc.initialized() || rtc.lostPower())
   {
    genie.WriteStr(23, "RTC is down. Run pcf8523 sketch. System halted.");
    while (1){};
   }
   else {
    genie.WriteStr(23, "RTC read error. Check battery/circuit. System Halted");
    while(1){};
   }
  }
  //SD Card Check
 genie.WriteStr(23, "Initializing SD card:...");
 delay(1200);
 pinMode(10,OUTPUT);
 if (!SD.begin(chipSelect))
 {
   genie.WriteStr(23, "Failed. SD Card not present. Insert SD and press reset. System
Halted.");
   while (1) {};
 }
 else
 {
```

```
   genie.WriteStr(23, "Success.");
   delay(1200);
  }
  genie.WriteStr(23, "Opening Valve");
  myservo.write(openvalve);
  delay(1200);
  genie.WriteStr(23, "Calibrating...");
  delay(1000);
  long int CALendtime=0;
  CALendtime = millis() + 1500;
  while (millis() < CALendtime)
 {
  PSIcalculation(); // PSI calculation
 }
 delay(1200);
 calibration=psi;
 PSIcalculation;
 delay(1200);
 genie.WriteStr(23, "Calibrated.");
 delay(1200);
 genie.WriteStr(23, "Press 'Start' when ready...");
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////
void loop()
{
  CurrentButtonState1 = digitalRead(Button1);
     if (CurrentButtonState1 != PreviousButtonState1 && CurrentButtonState1 == LOW)
     {
      sPosition += sIncrement;
      myservo.write(sPosition);
      delay(10);
      }
```

```
    PreviousButtonState1 = CurrentButtonState1;
    CurrentButtonState2 = digitalRead(Button2);
    if (CurrentButtonState2 != PreviousButtonState2 && CurrentButtonState2 ==LOW)
    {
     sPosition -= sIncrement;
     myservo.write(sPosition);
     delay(10);
     }
    PreviousButtonState2 = CurrentButtonState2;
  genie.DoEvents(); //This calls the library each loop to process the responses from the display.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////


void myGenieEventHandler(void)
{
  genieFrame Event;
  genie.DequeueEvent(&Event); //This removes the queued event from the buffer to process it below
  int EVENT_val=0;
  int slider_val=0;
  EVENT_val = genie.GetEventData(&Event);
  //If the cmd received is from a Reported Event (Events triggered from the Events tab of Workshop4 objects)
  if (Event.reportObject.cmd == GENIE_REPORT_EVENT)
  {
   if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
   {
    if (Event.reportObject.index == 4)
    {
    spPosition=spPosition+spIncrement;
    myservo.write(spPosition);
    delay(1000);
```

```
 genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, spPosition);
 }
 if (Event.reportObject.index == 5)
 {
 spPosition=spPosition-spIncrement;
 myservo.write(spPosition);
 delay(1000);
 genie.WriteObject(GENIE_OBJ_LED_DIGITS, 12, spPosition);
 }
}
if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
{
 if (Event.reportObject.index == 1)
 {
   TestProgram();
   genie.WriteStr(25, "Idle");
 }
 if (Event.reportObject.index == 3)
 {
   return;
 }
}
 //To create a new file for the session which is form 1, Userbutton0
 if (Event.reportObject.object == GENIE_OBJ_4DBUTTON)
 {
   if (Event.reportObject.index == 2)
   {
     genie.WriteStr(24, "Checking INFO file...");
     delay(1000);
     MakeNewFile();
   }
 }
```

```
        //Puts in High (e) and Low (i) psi into file. Then displays results of session and displays previous session
    if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
    {
     if (Event.reportObject.index == 12)
     {
      genie.WriteObject(GENIE_OBJ_FORM, 7, 1);
      PullFromNewAndOldFiles();
     }
    }
    //Controls Training% for PSItraining %
    if (Event.reportObject.object == GENIE_OBJ_SLIDER)
    {
     if (Event.reportObject.index == 0)
     {
      slider_val = genie.GetEventData(&Event);
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0, slider_val);
     TrainingPercent = slider_val;
     }
    }
    //For MIP
    if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
    {
     if (Event.reportObject.index == 6)
     {
      myservo.write(almostclosevalve);
      TimerMIP(4, 5, 0);
      genie.WriteStr(0, "Done.");
     }
    }
     //For MEP
    if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
    {
```

```
  if (Event.reportObject.index == 7)
  {
    TimerMEP(4, 5, 0);
    genie.WriteStr(0, "Done. Proceed to results.");
  }
}
//Shows results of MEP/MIP on MEP/MIP result
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 5)
  {
    delay(500);
    genie.WriteObject(GENIE_OBJ_FORM, 4, 1);
    delay(500);
    genie.WriteStr(1, MEP);
    delay(500);
    TargetMEP = MEP * TrainingPercent / 100;
    genie.WriteStr(2, TargetMEP);
    delay(500);
    FMIP = -1 * MIP;
    genie.WriteStr(3, FMIP);
    delay(500);
    TargetMIP = FMIP * TrainingPercent / 100;
    genie.WriteStr(4, TargetMIP);
    delay(500);
    myservo.write(openvalve);
  }
}
//Moves to form 5 to calibrate valve positions
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 2)
  {
```

```
      delay(500);
      genie.WriteObject(GENIE_OBJ_FORM, 5, 1);
      delay(500);
      genie.WriteStr(5, "Recalibrating pressure sensor. Please   wait."); //Keep couple spaces between please and wait so it fits the string on the display correctly.
      Recalibrating();
      genie.WriteStr(5, "Press the 'Locate PEmax Position'         button when ready.");
     }
   }
   //Calibrates the Valve Position
   h=75;
   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
     {
      if (Event.reportObject.index == 20)
      {
       myservo.write(h);
      TimerMatchPressureMEP();
       genie.WriteStr(5, "Done. Press the 'Locate PImax Position' button when ready.");
      }
     }
     if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
     {
      if (Event.reportObject.index == 21)
      {
       delay(1200);
       genie.WriteStr(5, "Recalibrating pressure sensor and servo motor. Please wait.");
       myservo.write(h);
       delay(1200);
       Recalibrating();
       delay(1200);
       TimerMatchPressureMIP();
       genie.WriteStr(5, "Done, Proceed to the training page.");
      }
```

```
    }
  if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
 {
  if (Event.reportObject.index == 3)
  {
   delay(500);
   genie.WriteObject(GENIE_OBJ_FORM, 6, 1);
   delay(500);
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 9, TargetMEP);
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 7, abs(TargetMIP));
   genie.WriteStr(22, "Chose the number of minutes to          train. Then press start.");
  }
 }
  //PSI Training Timers
  //There are options for 1, 2, 3, 4, or 5 minutes of training. Each button needs to identify how much time there is and write it the LED on the training page.
  //The PSITimer is in milliseconds with an extra half second added on.
  if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 17)
   {
    PSITimer = 300500;
    count =5;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
   }
  }
   if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
  {
   if (Event.reportObject.index == 16)
   {
    PSITimer = 240500;
    count =4;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
```

```
  }
}
 if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 15)
  {
    PSITimer = 180500;
    count =3;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
  }
}
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 14)
  {
    PSITimer = 120500;
    count=2;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
  }
}
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 13)
  {
    PSITimer = 60500;
    count=1;
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer/1000);
  }
}
 //For PSI training screen. Write to excel and give option to end or continue.
if (Event.reportObject.object == GENIE_OBJ_USERBUTTON)
{
  if (Event.reportObject.index == 11)
```

```
      {
        logfile = SD.open(filename, FILE_WRITE);
        TimerAndRecordPSI ();
        logfile.close();
        delay(500);
        genie.WriteStr(22, "Done.");
        delay(1000);
        genie.WriteStr(22, "Click results to see results. Click start to train again.");
      }
    }
  }
}
void MakeNewFile()
{
  genie.WriteStr(24, "Patient Info:...");
  delay(1200);
  logfile = SD.open("INFO.TXT");
  if (logfile.available())
  {
    genie.WriteStr(24, "File exists.");
  }
  else
  {
    genie.WriteStr(24, "File does not exist.");
    delay(1200);
    genie.WriteStr(24, "Check SD Card");
    delay(3600);
    return;
  }
  filename[0] = logfile.read();
  filename[1] = logfile.read();
  filename[2] = logfile.read();
  filename[3] = logfile.read();
```

```
filename[4] = logfile.read();
filename[5] = logfile.read();
filename2[0] = filename[0];
filename2[1] = filename[1];
filename2[2] = filename[2];
filename2[3] = filename[3];
filename2[4] = filename[4];
filename2[5] = filename[5];
logfile.close();
genie.WriteStr(24, filename);
delay(1000);
genie.WriteStr(24, "Making new file:...");
for (uint8_t i = 1; i < 100; i++) {
  filename[6] = i / 10 + '0';
  filename[7] = i % 10 + '0';
  if (! SD.exists(filename)) {
    DateTime now = rtc.now();
    logfile = SD.open(filename, FILE_WRITE);
    logfile.print(",,");
    logfile.print("Date(D/M/Y)/Time:,");          //A1:B1
    logfile.print(now.day(), DEC);
    logfile.print("/");
    logfile.print(now.month(), DEC);
    logfile.print("/");
    logfile.print(now.year(), DEC);
    logfile.print(",");
    logfile.print(now.hour(), DEC);
    logfile.print(":");
    logfile.print(now.minute(), DEC);
    logfile.print(":");
    logfile.println(now.second(), DEC);
    logfile.print(",,");
    logfile.print("Training Time:");                //A2:B2
```

```
logfile.print(",");
logfile.print("        ");
logfile.print(",");
logfile.println("seconds");
logfile.print(",,");
logfile.print("PEmax:");                  //A2:B2
logfile.print(",");
logfile.print("        ");
logfile.print(",");
logfile.println("cm H2O");
logfile.print(",,");
logfile.print("PImax:");                  //A3:B3
logfile.print(",");
logfile.print("        ");
logfile.print(",");
logfile.println("cm H2O");
logfile.print(",,");
logfile.print("Training %");
logfile.print(",");
logfile.print("     ");
logfile.print(",");
logfile.println("%");
logfile.print(",,");
logfile.print("# of points:");               //A4:B4
logfile.print(",");
logfile.print( "=COUNT(B:B)");
logfile.print(",");
logfile.println("points");
logfile.print(",,");
logfile.print("Time Duration");               //A5:B5
logfile.print(",");
logfile.print("=(MAX(A:A)-MIN(A:A))/1000");
logfile.print(",");
```

```
    logfile.println("seconds");
    logfile.print(",,");
    logfile.print("Points per second:");          //A6:B6
    logfile.print(",");
    logfile.print("=D6/D7");
    logfile.print(",");
    logfile.println("points/s");
    logfile.print("Time(ms)");      //A8:B8
    logfile.print(",");
    logfile.println("cm H2O");
    logfile.close();
    break; // leave the loop!;
   }
  }
  delay(1000);
  genie.WriteStr(24, filename);
  delay(1000);
  genie.WriteStr(24, "Checking old file...");
  delay(1000);
  filename2[6] = filename[6];
  filename2[7] = filename[7];
  xx = filename2[6] - '0';   //Renames filename to go back 1 previous session to display on
result scrEn
  xxx = filename2[7] - '0';
  x = (xx * 10) + xxx - 1;
  filename2[6] = x / 10 + '0';
  filename2[7] = x % 10 + '0';
  logfile = SD.open(filename2, FILE_READ);
  logfile.seek(20);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
```

```
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  MEP = logfile.readStringUntil(',').toInt();
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  MIP = logfile.readStringUntil(',').toInt();
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  TrainingPercent = logfile.readStringUntil(',').toInt();
  logfile.close();
//writing the previous values to form 7 led digits
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 3, MEP);
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 2, abs(MIP));
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 1, TrainingPercent); //Previous %
training display in form 2
  genie.WriteStr(24, filename2);
  delay(1000);
  genie.WriteStr(24, "Proceed to PE/PImax.");
  delay(100);
}
int PSIcalculation() {
  total = total - readings[readIndex];       //psi readings + calculations
  readings[readIndex] = analogRead(inputPin);
  total = total + readings[readIndex];
  readIndex = readIndex + 1;
  if (readIndex >= numReadings) {
    readIndex = 0;
  }
  psi = total / numReadings;
```

```
  psi = psi / bits;            //bits = 1023.0
  psi = psi * Vcc;              //Vcc=3.3
  psi = psi - offset;          //offset = 2.5
  psi = psi / resolution;       //resolution = 0.0002584
  psi = psi / GAIN;             //GAIN
  psi = psi * 1.36;        //1.36 is conversion from 1 mmHg = 1.36 cmH2O
  psi = psi - calibration;
}
void PullFromNewAndOldFiles() {
  delay(500);
  logfile = SD.open(filename, O_RDWR);
  logfile.seek(20);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(TotalTime/1000);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(MEP);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(FMIP);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.print(TrainingPercent);
  logfile.close();
```

```
  genie.WriteObject(GENIE_OBJ_USERBUTTON, 12, 1);
  delay(1000);
  genie.WriteStr(6, filename);
  delay(500);
  logfile = SD.open(filename, FILE_READ);
  logfile.seek(20);
  genie.WriteStr(7, logfile.readStringUntil(','));    //DATE
  delay(500);
  genie.WriteStr(8, logfile.readStringUntil(','));    //TIME
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(9, logfile.readStringUntil(','));    //Total Time
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(11, logfile.readStringUntil(','));    //MEP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(12, logfile.readStringUntil(','));    //MIP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(13, logfile.readStringUntil(','));    //% Training
  delay(500);
  logfile.close();
  xx = filename[6] - '0';   //Renames filename to go back 1 previous session to display on
result scrEn
  xxx = filename[7] - '0';
```

```
  x = (xx * 10) + xxx - 1;
  filename[6] = x / 10 + '0';
  filename[7] = x % 10 + '0';
  genie.WriteStr(14, filename);
  delay(500);
  logfile = SD.open(filename, FILE_READ);
  logfile.seek(20);
  genie.WriteStr(15, logfile.readStringUntil(','));
  delay(500);
  genie.WriteStr(16, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(17, logfile.readStringUntil(','));
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(19, logfile.readStringUntil(','));    //MEP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(20, logfile.readStringUntil(','));    //MIP
  delay(500);
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  logfile.readStringUntil(',');
  genie.WriteStr(21, logfile.readStringUntil(','));    //% Training
  delay(500);
  logfile.close();
}
void Recalibrating() {
```

```
  long int RCendtime = 0;
  R = 0;
  RCP = 0;
  RCendtime = millis() + 5500;
  while (millis() < RCendtime)
  {
    PSIcalculation();
  }
}
void TestProgram()
{
  long int TPendtime = 0;
  TP = 0;
  TEP = 0;
  TPendtime = millis() + 5500;
  while (millis() < TPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 13, abs(psi));
    genie.WriteStr(25, "Measuring");
  }
}
void TimerAndRecordPSI () {
  long int Pendtime = 0;
  int scount=0;
  int smove=0;
  int c;
  int prevCase = -1;  // To make sure case starts not being equal to any other possible case (something is written to LCD)
  int TargetMEPNinety=TargetMEP * 0.90;
  int TargetMIPNinety=TargetMIP * 0.90;
  int TargetMIPOneTen=TargetMIP * 1.1;
  int TargetMEPOneTen=TargetMEP * 1.1;
```

```
  BreathTime=PSITimer;
  TotalTime=TotalTime+BreathTime;
  Pendtime = millis() + PSITimer;
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, PSITimer / 1000);
  while (millis() < Pendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 6, (Pendtime - millis()) / 1000);   // write elapsed time (countdown)
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 8, abs(psi));                       // write current pressure in cmH2O
    logfile.println((String)millis()+","+psi);                                  // writes entry to sd card
    // LED indicator during training
    if (psi > (TargetMEPNinety)) {
     // Check to see if user is in "the window" for exhale
     genie.WriteObject(GENIE_OBJ_USER_LED, 0, 0);
     genie.WriteObject(GENIE_OBJ_USER_LED, 1, 1);
    }
    else if (psi < (TargetMIPNinety)) {
     // Check to see if user is in "the window" for inhale
     genie.WriteObject(GENIE_OBJ_USER_LED, 0, 1);
     genie.WriteObject(GENIE_OBJ_USER_LED, 1, 0);
    }
    else {
     // user NOT in "the window"
     genie.WriteObject(GENIE_OBJ_USER_LED, 1, 0);
     genie.WriteObject(GENIE_OBJ_USER_LED, 0, 0);
    }
    // End LED indicator during training
    // Classify to ID state we are in
    if ( ( psi >= 0.74 ) && ( psi <= TargetMEPOneTen ) )
       {
         c = 0;
```

```
   }
else if ( ( psi <= -0.74 ) && ( psi >= TargetMIPOneTen ) )
   {
    c = 1;
   }
else if ( psi > TargetMEPOneTen )
   {
    c = 2;
   }
else if ( psi < TargetMIPOneTen )
   {
    c = 3;
   }
else
   {
    // This catches "dead band" -0.74 < psi < 0.74
    c = 4; // the "none of these" case
   }
// END CLASSIFICATION

switch (c)
  {
   case 0:
    // 0.74 to 110% of Target MEP
    TrainingServoWrite(MEPfinalpos);
    if (prevCase != c)
      {                                    // only write to LCD if case has changed!
      genie.WriteStr(22, "Normal Expiratory Position");
      prevCase = c;                        // update prevCase to current case
      }
    break;
   case 1:
    // 0.74 to 110% of Target MIP
```

```
  TrainingServoWrite(MIPfinalpos);
  if (prevCase != c)
    {                                   // only write to LCD if case has changed!
      genie.WriteStr(22, "Normal Inspiratory Position");
      prevCase = c;                     // update prevCase to current case
    }
    break;
case 2:
  // PSI is > 110% of Target MEP
  TrainingServoWrite(MEPfinalpos-10);
  if (prevCase != c)
    {                                   // only write to LCD if case has changed!
      genie.WriteStr(22, "Too much expiratory pressure.");
      prevCase = c;                     // update prevCase to current case
    }
    break;
case 3:
  // PSI is < 110% of Target MIP
  TrainingServoWrite(MIPfinalpos-10);
  if (prevCase != c)
    {                                   // only write to LCD if case has changed!
      genie.WriteStr(22, "Too much inspiratory pressure.");
      prevCase = c;                     // update prevCase to current case
    }
    break;
case 4:
  // DEADBAND - Moving from Exhale to Inhale or vice versa
  if (prevCase != c)
    {   // only write to LCD if case has changed!
      genie.WriteStr(22, "Transitioning..");
      prevCase = c;      // update prevCase to current case
    }
    break;
```

```
        default:
          genie.WriteStr(22, "Do Nothing.");
          break;
       }
    }
}
void TimerMEP (int a, int b, int c) {
 genie.WriteStr(c, "Starting...");
 delay(1000);
 genie.WriteStr(c, "3...");
 delay(1000);
 genie.WriteStr(c, "2...");
 delay(1000);
 genie.WriteStr(c, "1...");
 delay(1000);
 genie.WriteStr(c, "Now.");
 delay(500);
 long int MEPendtime = 0;
 E = 0;
 MEP = 0;
 MEPendtime = millis() + 5500;
 while (millis() < MEPendtime)
 {
  PSIcalculation();
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, a, (MEPendtime - millis()) / 1000);
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, b, abs(psi));
  MEParray[E] = psi;
  E++;
 }
 for (E = 0; E < 96; E++)
 {
```

```
    MEPsumarray[E] = (MEParray[E] + MEParray[E + 1] + MEParray[E + 2] +
MEParray[E + 3] + MEParray[E + 4] + MEParray[E + 5] + MEParray[E + 6] +
MEParray[E + 7] + MEParray[E + 8] + MEParray[E + 9]) / 10;
    delay(5);
  }
  int idx;
  for (byte idx = 0; idx != 96; idx++)
  {
    if (MEPsumarray[idx] > MEP) {
      MEP = max(MEPsumarray[idx], MEP);
    }
    delay(5);
  }
}
void TimerMIP (int a, int b, int c) {
  genie.WriteStr(c, "Starting...");
  delay(1000);
  genie.WriteStr(c, "3...");
  delay(1000);
  genie.WriteStr(c, "2...");
  delay(1000);
  genie.WriteStr(c, "1...");
  delay(1000);
  genie.WriteStr(c, "Now.");
  delay(500);
  long int MIPendtime = 0;
  I = 0;
  MIP = 0;
  MIPendtime = millis() + 5500;
  while (millis() < MIPendtime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, a, (MIPendtime - millis()) / 1000);
```

```
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, b, abs(psi));
    MIParray[I] = psi;
    I++;
  }
  for (I = 0; I < 96; I++)
  {
    MIPsumarray[I] = (MIParray[I] + MIParray[I + 1] + MIParray[I + 2] + MIParray[I +
3] + MIParray[I + 4] + MIParray[I + 5] + MIParray[I + 6] + MIParray[I + 7] +
MIParray[I + 8] + MIParray[I + 9]) / 10;
    delay(5);
  }
  int idy;
  for (byte idy = 0; idy != 96; idy++)
  {
    if (MIPsumarray[idy] < MIP) {
      MIP = min(MIPsumarray[idy], MIP);
          }
    delay(5);
  }
}
void TimerMatchPressureMEP() {
  genie.WriteStr(5, "Starting...");
  delay(1000);
  genie.WriteStr(5, "3...");
  delay(1000);
  genie.WriteStr(5, "2...");
  delay(1000);
  genie.WriteStr(5, "1...");
  delay(1000);
  genie.WriteStr(5, "Now.");
  delay(500);
  long int MEPMatchPressureEndTime=0;
  J=0;
```

```
  MPF=0;
  MEPMatchPressureEndTime=millis() + 105500;
  while (millis() < MEPMatchPressureEndTime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, (MEPMatchPressureEndTime -
millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, abs(psi));
    delay(1000);
  if (abs(psi)<= .9*TargetMEP)
  {
    h=h+1;
    myservo.write(h);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 26, h);
    MEPfinalpos=h;
  }
  else if (abs(psi) >= .9*TargetMEP)
  {
    break;
  }
  else
  {
    break;
  }
  }
}
void TimerMatchPressureMIP() {
  genie.WriteStr(5, "Starting...");
  delay(1000);
  genie.WriteStr(5, "3...");
  delay(1000);
  genie.WriteStr(5, "2...");
  delay(1000);
```

```
  genie.WriteStr(5, "1...");
  delay(1000);
  genie.WriteStr(5, "Now.");
  delay(500);
  long int MIPMatchPressureEndTime=0;
  J=0;
  MPS=0;
  MIPMatchPressureEndTime=millis() + 105500;
  while (millis() < MIPMatchPressureEndTime)
  {
    PSIcalculation();
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 10, (MIPMatchPressureEndTime -
millis()) / 1000);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 11, abs(psi));
    delay(1000);
  if (psi> .9*TargetMIP)
  {
    h=h+1;
    myservo.write(h);
    genie.WriteObject(GENIE_OBJ_LED_DIGITS, 26, h);
    MIPfinalpos=h;
  }
  else if (psi <= .9*TargetMIP)
  {
    break;
  }
  else
  {
    break;
  }
  }
}
void TrainingServoWrite(int x)
```

```
{
  myservo.write(x);
}
```