CLEMSON UNIVERSITY – INTELLIGENT MATERIALS, SYSTEMS, AND ENVIRONMENTS

ComforTable

The smart solution for nightstands

Charreau Bell, Christopher Felegie, Nivedhitha Giri, Jessica Merino, Bryan Willimon

12/6/2010

Bedside tables to facilitate independence among the older population and rehabilitative patients are non-existent. The current design creates general-purpose bedside tables that are unwieldy and hard to use. For this reason, the automated ComforTable has been designed. The system is a unification of a nightstand with a table that can be delivered from within the nightstand for efficient storage. The nightstand features a rotating drawer system for maximum storage and organization of small items, as well as a continuum surface table which is able to sense the user and respond to user needs. The system is also equipped with artificial intelligence which is able to predict the needs of the user as well as tailor its actions based on a reward from the user. A prototype has been designed to successfully implement each of these functions, and is actuated by innovative pneumatic muscles.

1 Introduction and Objectives

Originally conceived as a helper for the elderly, the ComforTable suite has developed into a ubiquitous system that can be used in a variety of environments. The purpose of the system is two-fold: 1) to increase user independence by aiding the user in retrieving small items commonly found in a nightstand or in a room by delivering them via continuum table, and 2) to ensure the user's comfort with the ComforTable by creating a system responsive and predictive of user interaction and schedule.

2 Scenario

At 7:00am, 62-year old Shelly awakes from her slumber. She automatically throws her hand towards the nightstand next to her, fumbling through the clutter of items next to her to find her glasses. Finally, she finds them by knocking them on the ground. Certainly, she knows that familiar clatter. Groaning as she painfully pushes herself from the bed, she leans down to pick up her glasses, searching blindly to find her glasses. Finally, she finds them and begins her day.

Every day has the same schedule; first the glasses at 7am, then the pills from the bathroom at 7:30am. She entertains herself with leisurely reading of a book from the bookshelf at 8, and so forth throughout the day.

Already on the brink of being placed in assisted living, Shelly's fall in the shower seals her fate, as she is no longer mobile, and is confined to a bed. Without the help of a nurse, simple tasks such as putting pressure on her hip as she rolls in the bed to reach her glasses, or attempting to retrieve her pills from her bathroom cabinet is altogether impossible. A few days after her fall, she hears a knock on her door, and she resignedly waits to be toted to her final assisted living facility, Restful Shores. However, she is shocked to find her beloved daughter leading two burly men into the room with a large wooden piece of furniture between them. "Over there – next to the bed," her daughter instructs.

"Shelby, what is this?" Shelly says with confusion. "I already have a nightstand – the same one I've had for the last 25 years!" Shelby smiles understandingly at her suddenly belligerent mother, and starts placing each item in the current nightstand into the revolving drawer system of the new nightstand.

"Mom," she says, "this is a ComforTable. It's here to retrieve for you everything you need so that you'll be able to stay here on your own, or at least with minimal help. All you do is select items from this touchscreen, and the ComforTable will deliver them. Well, I've gotta go; I'm running late for a meeting, but give it a try and see if you like it."

Shelly's frown said just about everything about her attitude towards this new piece of technology. All she could think to herself was, "What's this NEW silly thing made by those young whippersnappers? My old nightstand was just fine!" It was late, so Shelly eventually fell asleep according to her normal routine, and woke up to attempting to fetch her glasses. Disoriented from the general episode of waking up, her eye caught view of the touchscreen with the large button that read “glasses.” Throwing her hand towards it, she pushed the large button. With a soft mechanical whirr, she saw her glasses emerge from the side of the nightstand on a flexible continuum table. It stopped shortly near her. Shocked from the sudden emergence of the table, she could barely move. The table seemed to sense her hesitation and gently arched its position so the glasses were closer to her, as if offering them to her. Cautiously, she took her glasses from the table and eyed it suspiciously. “Thanks,” she said warily, and was shocked to observe that the table turned blue, as if it *appreciated* what she had said. It then returned to the nightstand. Still skeptical of this new technology, she avoided it for the rest of the day, and went to bed according to her nightly routine.

The next morning, she woke up with her usual disorientation, but as she opened her eyes, she found the table offering her the glasses, as if it knew her desires. “Thanks,” she said again, and the table again glowed blue as if it understood her gratitude. From then on, she began ordering items out of the nightstand according to her daily routine, and as the days grew on, the nightstand seemed to always deliver items slightly ahead of the time she needed them. Even better, it seemed to change its actions based on her feedback; it she rewarded it positively, it glowed and repeated the same actions. If she rewarded it negatively, it refrained from repeating the same actions.

Several days later, her daughter returned to find her mother happily interacting with the new technology, sometimes even stroking it as if it was a pet. “Man,” Shelly thought to herself, “Mom looks really happy with that nightstand. Perhaps even ComforTable.”

3 youTube Link

A demonstration by the system prototype can be seen on the following web address: http://www.youtube.com/watch?v=PcjWBbOEZrM

4 Source Code

The source code for the control software and the artificial intelligence can be found in Appendix C. The Arduino source code controlling the ComforTable and nightstand can be found in Appendices D and E, respectively.

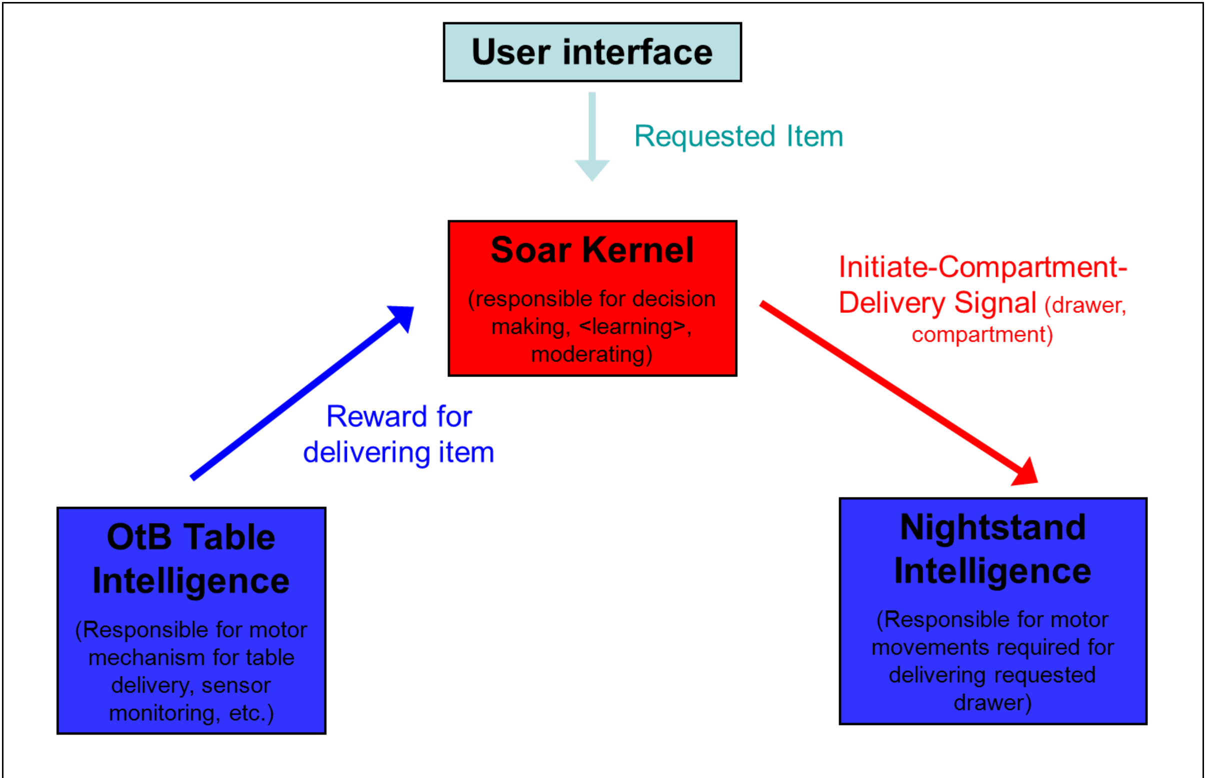
5 Model Implementation

5.1 Overview

The system has been subdivided into three parts: the ComforTable control software (CTCS) and artificial intelligence (AI), the continuum table, and the nightstand drawer system. The three are connected at two interfaces: one between the control software and the nightstand, and one between the control software and the continuum table. A description of the each of the component parts can be found in the following sections.

5.2 ComforTable and Nightstand Control Software and Artificial Intelligence

The CTCS has four different components: the user interface, implemented in C++; the Soar Kernel used to achieve the AI goals of the system; a serial interface to the nightstand to deliver drawers; and a serial interface to the table to report back the reward given. A flowchart of this information is shown below.

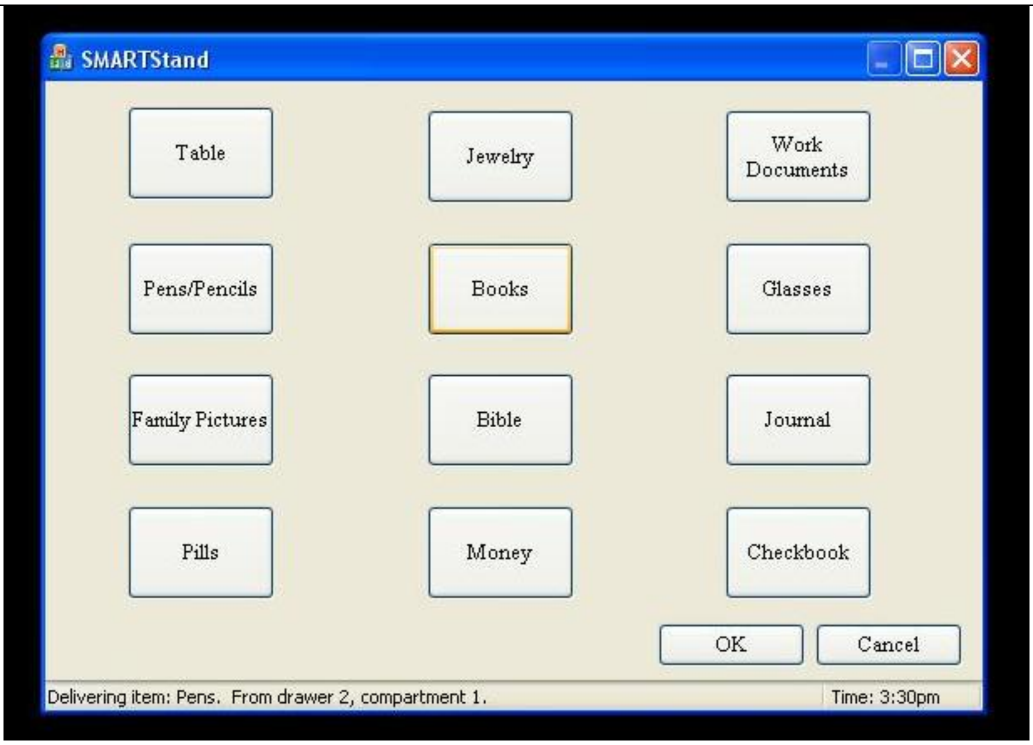

User interface
Requested Item
Soar Kernel
(responsible for decision making, <learning>, moderating)
Initiate-Compartment-Delivery Signal (drawer, compartment)
Reward for delivering item
OtB Table Intelligence
(Responsible for motor mechanism for table delivery, sensor monitoring, etc.)
Nightstand Intelligence
(Responsible for motor movements required for delivering requested drawer)

CTCS information flow diagram

Each part of the system will be described in further detail in the following sections. Information about the Arduino code used to implement the functionality of the continuum table ("OtB Table" in diagram above) and the nightstand intelligence can be found in Sections 5.3 and 5.4, respectively.

5.2.1 User Interface

The user interface was implemented in C++, and supports the Windows operating system. The purpose of this interface is to enable the user to select items from the nightstand, but also to inform the user of events. For example, when the user selects "table," the system will deliver the table, and also display the drawer and the compartment in which it is located in the nightstand. It also has other messages, including that it is waiting to be rewarded for a particular action, or that the reward has been received. The time is also shown for the user. The user interface is shown in the diagram below.



Delivering item: Pens. From drawer 2, compartment 1.   Time: 3:30pm

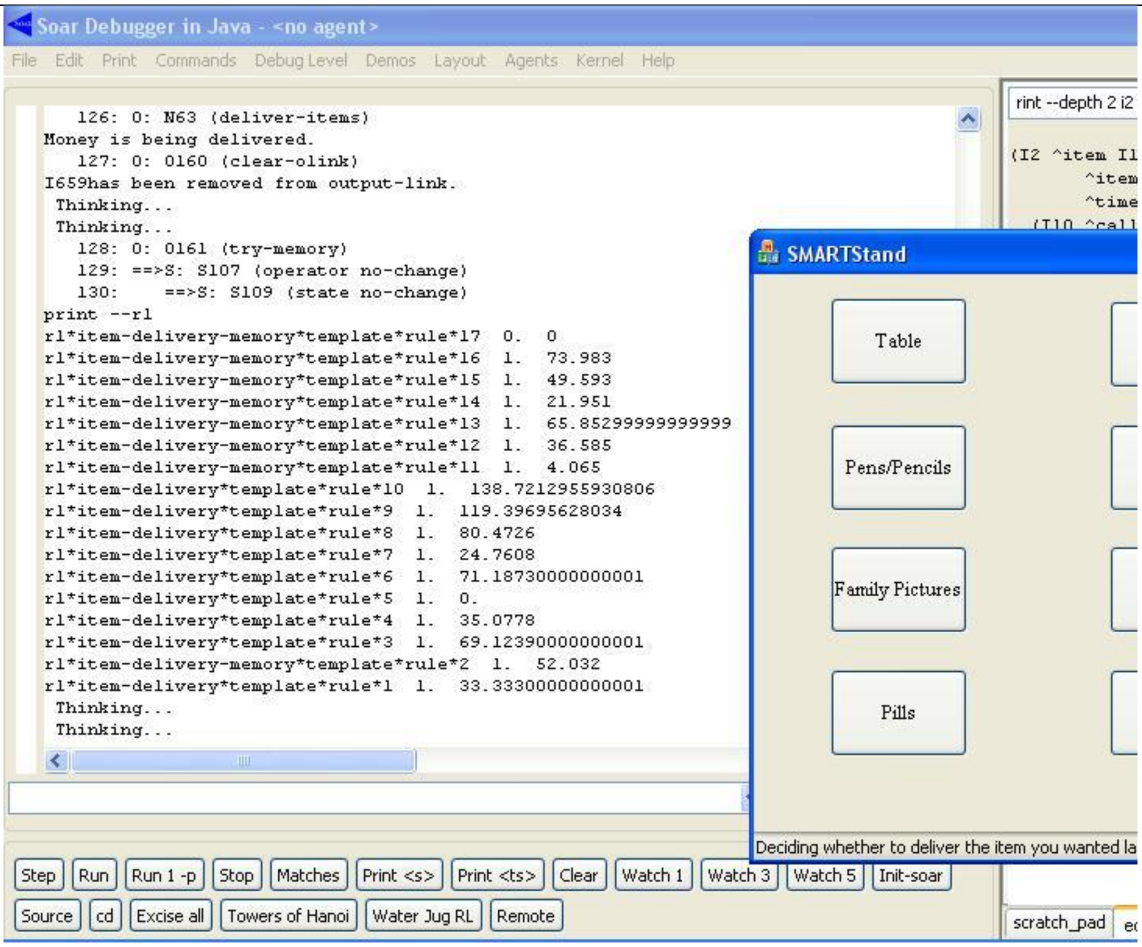
ComforTable user interface

5.2.2 Soar Kernel

Developed at the University of Michigan, the Soar is a cognitive architecture designed for problem-solving applications. By design, it has long-term memory which stores *rules*, or if-then statements relating preconditions to actions. For example, in the context of ComforTable, long-term memory is populated with rules that tell it when to deliver a certain item based on selections the user has made in the past.

Soar also has short-term memory, which reflects its current state. For example, the created Soar agent will be able to keep track of what items are or are not in the nightstand, and what has been delivered. In this way, the agent will know that it can only deliver an item that is located within the nightstand.

Most powerfully used here are Soar's built-in learning mechanisms, which include reinforcement learning and episodic memory. The goal of reinforcement learning is for the agent to learn what the best action to take is based on a reward that is given to it by the user. In this case, the user clicks a button which results in a reward of either 0 or 255. Better rewards have higher values, and thus, the agent will repeat this action more often. An example of this can be seen in the figure below.


Soar Debugger in Java - <no agent>
File Edit Print Commands Debug Level Demos Layout Agents Kernel Help
rint --depth 2 i2
SMARTStand
Table
Pens/Pencils
Family Pictures
Pills
Step Run Run 1 -p Stop Matches Print <s> Print <ts> Clear Watch 1 Watch 3 Watch 5 Init-soar
Source cd Excise all Towers of Hanoi Water Jug RL Remote
scratch_pad

```
   126: O: N63 (deliver-items)
Money is being delivered.
   127: O: O160 (clear-olink)
I659has been removed from output-link.
 Thinking...
 Thinking...
   128: O: O161 (try-memory)
   129: ==>S: S107 (operator no-change)
   130:    ==>S: S109 (state no-change)
print --rl
rl*item-delivery-memory*template*rule*17  0.  0
rl*item-delivery-memory*template*rule*16  1.  73.983
rl*item-delivery-memory*template*rule*15  1.  49.593
rl*item-delivery-memory*template*rule*14  1.  21.951
rl*item-delivery-memory*template*rule*13  1.  65.85299999999999
rl*item-delivery-memory*template*rule*12  1.  36.585
rl*item-delivery-memory*template*rule*11  1.  4.065
rl*item-delivery*template*rule*10  1.  138.7212955930806
rl*item-delivery*template*rule*9  1.  119.39695628034
rl*item-delivery*template*rule*8  1.  80.4726
rl*item-delivery*template*rule*7  1.  24.7608
rl*item-delivery*template*rule*6  1.  71.18730000000001
rl*item-delivery*template*rule*5  1.  0.
rl*item-delivery*template*rule*4  1.  35.0778
rl*item-delivery*template*rule*3  1.  69.12390000000001
rl*item-delivery-memory*template*rule*2  1.  52.032
rl*item-delivery*template*rule*1  1.  33.33300000000001
 Thinking...
 Thinking...
```

```
(I2 ^item I1
     ^item
     ^time
 (I10 ^call
```

Deciding whether to deliver the item you wanted la

Soar Java Debugger displaying reward values for rules

Seen here is the Soar Java Debugger, which can be most simply thought of as a monitor, or a look inside of the running Soar kernel. Towards the top of the image, a statement “deliver-items” and “Money is being delivered” can be seen. This means that a rule has fired, which has selected money as the item to be delivered. Most interesting about this execution is the symbol “N63” preceding. This rule was *not* originally programmed into the system; that is, the agent was able to write this rule itself based on past actions. Also interesting to observe are all the rules with the name structure “rl*item-delivery**template*rule**”. These are the reinforcement learning rules that Soar creates in order to tailor its actions to the user’s rewards.

Episodic memory enables the software to remember what action it took in the past, and therefore, what action might be available for it to take in the future. This is particularly interesting in the case of the ComforTable, because any action or set of actions that can be taken by the user can be used as a trigger to remember some event that occurred in the software’s past. Currently, the trigger is merely time, so as the software executes, Soar constantly tries to remember what it did at that last time. However, in the future, this can be implemented for Soar to remember actions, or anything that can be parameterized by an Arduino. Thus, actions such as the user knocking their glasses off the nightstand, or the several actions the user takes in succession can be used as triggers. The figure above illustrates several instances of reinforcement learning and episodic memory combined; that is, the system will take an action that was not specifically forced by the user, and will receive a reward. Then, this same remembering of the past should be used again.

Lastly, to reiterate, the power of this artificially intelligent agent is in the fact that the programmer does not have to explicitly write out every possible event that the user can take so that the system can appropriately respond. The programmer writes *some* rules, like deliver-items seen above, and the system will be able to write some of its own rules to determine in what specific events that it should take a specific action.

Sample productions and source code can be found in Appendix A of this document. For the full source, please contact iwalker@clemson.edu, kegreen@clemson.edu, or charreau.s.bell@gmail.com.

5.2.3 Serial Interfaces

As shown in the block diagram in Section 5.2, two serial interfaces were necessary to appropriately communicate with the Arduinos controlling the nightstand and the table. In the current prototype, a number is written to the nightstand Arduino so that it can actuate a compartment of a drawer to be delivered. As for the serial connection with the continuum table, after a drawer is delivered, the system waits for 4 seconds to receive a reward. If it does not detect a reward in those 4 seconds, then the system records a reward of 0 and moves along. This will be discussed further in Section 6.

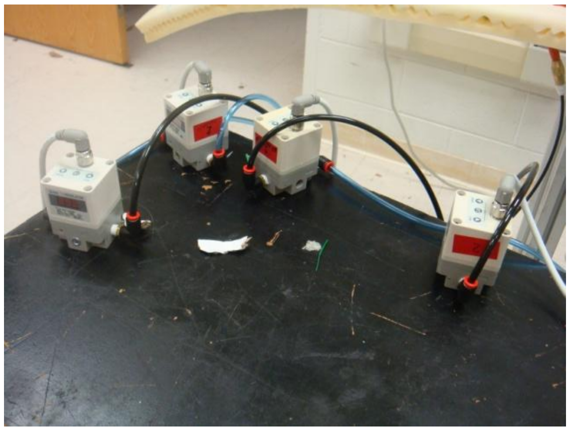
5.3 ComforTable Implementation

5.3.1 Physical Construction: Pneumatic Muscles

In order to meet the design constraint of user acceptance and comfort with the technology, pneumatic motors were chosen to actuate the continuum table. This also resulted in a number of different configurations and positions of the table to best serve the user. In order to reach these configurations, pneumatic muscles were constructed according to the procedure outlined in Appendix A. The table was then built in the method described in Appendix B.

5.3.2 Pneumatic, Electrical, and Software Connections

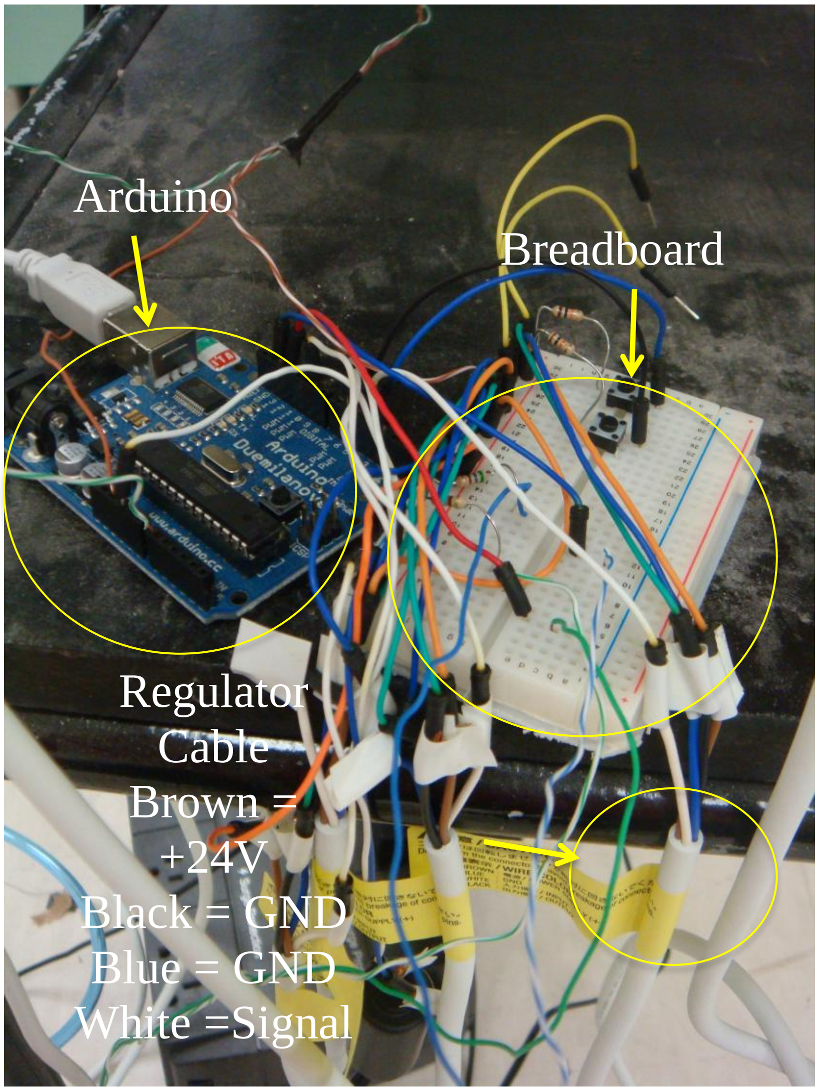
The air pressure in each of the pneumatic muscles (a.k.a McKibben actuators) is regulated by a pressure regulator (ITV3010-01N11L4). An air compressor is used as a pressurized air source for all the actuators. The pressure regulators are connected to a +24V DC power supply and the control signal is provided from an Arduino microcontroller (Arduino Duemilanove with ATMega328). The digital output pins (that can be Pulse Width Modulated to get an equivalent analog voltage) are directly connected to the control inputs of the pressure regulators. The grounds of the Arduino board and the pressure regulator electrical power supply source are tied together.



Pressure regulators and pneumatic lines from the air muscles

The basic table movements are obtained by changing the air pressures of the four pneumatic muscles. The pressure regulators used for this system have a linear voltage-pressure relationship. With every 1V increase in the control input, the output pressure increases by 13 psi. The control input can have a maximum voltage of 10V, which corresponds to an air pressure of 130 psi. The pulse width modulated output from the Arduino can range from 0-5V. This corresponds to values from 0-255 that has to be coded in the AnalogWrite function. Hence the value to be coded to set a pressure of x psi in an air muscle is 3.923*x. Since the muscles are not pressurized beyond 40 psi, the output voltage from the Arduino board is sufficient to control muscle pressure levels using pressure regulators.

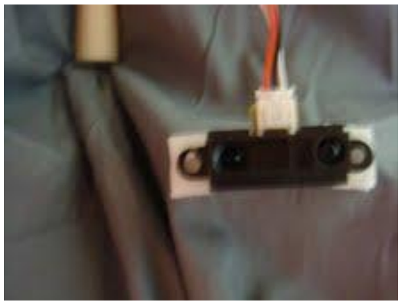
Two IR sensors (Sharp GP2D120A) are connected to the input pins of the Arduino – one is used to detect the person's movements below the table so that the table can move upward abruptly if the person wants to get up quickly from the bed; the other sensor is used to detect if the person has grabbed an object that has been delivered to the table from the nightstand. The supply and GND lines of the sensor are connected to the +5V and GND pins of the Arduino board respectively. The multiplication factor used to convert the values read from the input pin of the Arduino to voltage is 0.0048828125. This voltage value is converted to distance by the following conversion: distance = 65*pow(volt1,-1.1).

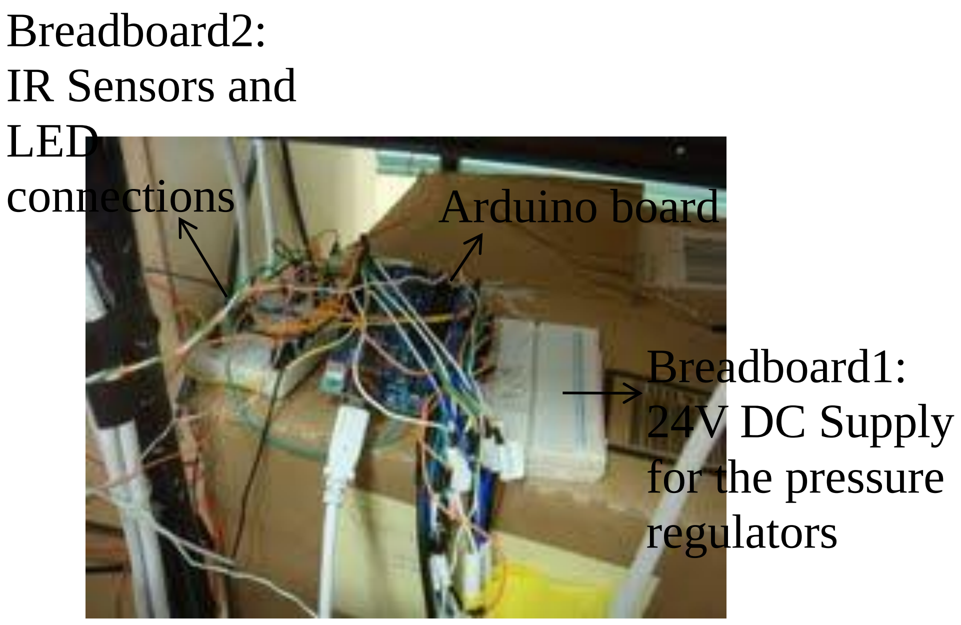


Breadboard connections for the pressure regulators

A panel consisting of three rows of LEDs is attached to the table on the user's side. The RED lights in the panel indicate that the table is morphing to change its current configuration. The GREEN lights indicate that the table is at rest. The BLUE lights are an indication of the table's happy mood when the user gives a reward for its timely actions by patting on a press button. The press button and the rows of LEDs are interfaced to the Arduino.

LEDs, IR sensor and push button at the side of table


IR sensor at the bottom of the table


Breadboard2: IR Sensors and LED connections
Arduino board
Breadboard1: 24V DC Supply for the pressure regulators

Electrical connections for sensors and pressure regulators

5.3.3 Table actuation and Scenarios

The table can bend, rotate and curve by pressurizing the four pneumatic muscles to appropriate pressure levels. Air muscle 1 refers to the muscle present on the top right portion of the table, air muscle 2 refers to the one on the top left, air muscle 3 refers to the one on the bottom right and air muscle 4 refers to the one on the bottom left. When the system is powered ON, air muscles 1 and 2 are pressurized to 25 psi so that the table remains horizontal balancing gravity. The table movements for two different scenarios are explained here along with a brief note on the code that is used to monitor the sensors to control the table's movement.

5.3.3.1 First scenario

The nightstand delivers an item that is frequently used by the person as soon as he wakes up in the morning (eg. Glasses). If the table detects that the person has not retrieved the glasses from the table in a short period of time, it curves and moves towards the person so that the person can easily reach for the item. The red lights turn on to indicate that the table is in motion. For the table to curve and reach the person, the air pressure values in different muscles are tabulated below,

Air Muscle	Pressure value (in psi)
1	0
2	30
3	0
4	30

The pressure values are increased in steps to get smoother table movements. After the person reaches for the item, the table retracts back to its normal position. The red lights turn off and the green lights turn on again.

5.3.3.2 *Second scenario*

The table detects movement of the person underneath it and realizes that the person wants to get up immediately from the bed. In this scenario, the table bends upward quickly in a single step. The air pressure values in the air muscles are tabulated below,

Air Muscle	Pressure value (in psi)
1	40
2	40
3	0
4	0

The table then comes to its normal position after a delay (which is programmed to be 10 seconds). The red lights are turned on until the table comes back to its original position.

At any point of time, the user can reward the table for its intelligent actions by patting on the push button on the table. In response to this, the blue LEDs in the panel blink twice.

The pressure values coded are slightly different from the values in the table to compensate for the slight variation in length of the muscles that are being used. Input from a push button is used to indicate the presence of an item on the table. The reward/feedback is communicated to the central intelligence serially. The occurrence of the two scenarios and the reward given to the table can be in any order and can be executed any number of times.

5.4 Nightstand Drawer System Implementation

5.4.1 Components

3 Servo motors (1 standard, 2 continuous)
Breadboard
Arduino ATmega 328 micro controller
Jumper cables
10 gears (5 for each side)
2 long chains
2 dowels in the shape of a boot for the kick-out mechanism

5.4.2 Physical Implementation

The drawer system within the nightstand is designed to deliver small compartments filled with miscellaneous items used in everyday life. This nightstand is designed to sit beside the bed in a home or hospital environment. Some of the items in the nightstand will consist of brush, tissues, glasses, loose change, notepad, etc. When an item is selected to be used, then drawer system will deliver the entire drawer to the opening on the side and push out a small compartment to the user. Each drawer consists of 1 to 3 compartments ranging in size and scale and can be used for any type of storage.

The component within the nightstand that makes up the drawer system consists of 5 gears, 1 long chain and 1 motor on each side of the nightstand. It also includes an Arduino 328 microcontroller that controls the left motor and right motor. For each side, 4 of the gears are placed in each corner of the nightstand to allow for full rotation. The long chain is then moving along all 4 gears as it is being powered by the remaining 5th gear. The 5th gear is powered by a single continuous servo motor that drives the single chain holding all of the drawers.

Each drawer is suspended above the chain as to allow free range of motion and to not obstruct any other drawers or other parts of the moving mechanism. Each drawer contains a wooden dowel so that it can connect to the moving chain and allow the drawer to stay upright when it is rotating within the nightstand. A single piece of cardboard is connected to the chain at one end and the other end contains a hole big enough to allow the ends of the drawer to fit in.

The final piece within the nightstand is the kick-out mechanism. This device is designed to push a specified compartment onto the table outside of the nightstand so that the user will be able to extract items from the compartment. The components that make up the kick-out mechanism consist of 1 motor and 1 thin slat. The thin slat is attached to the motor to resemble a fin or flipper. It is designed to push along the backside of the compartment without touching the drawer. This device is placed just above the opening of the nightstand.

5.5 Theoretical Design for Latching Mechanism

The idea for a supporting latching mechanism was a result of the need to support the opposite end of the table when in its extended position. This supporting device must be able to provide several functions:

- 360 degree of rotation – enables the table to move out of the way of the patient
- Telescoping device - adjusts the height of the table above the patient
- Scissoring Mechanism – helps with support of the table when detached from the nightstand
- Rail attached to the bed – aids in the positioning of the table.

Two types of locking mechanisms were proposed in order to determine the optimal device for this implementation:

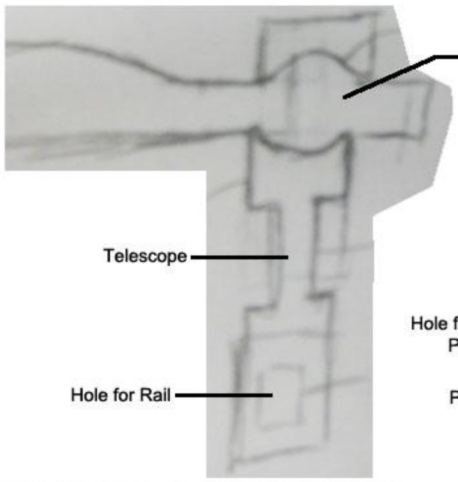
- Pin Mechanism – functions by a pin passively inserted into a slot on the underside of the table
- Ballooning Mechanism – the end of the table will expand once inserted into a slit.

The proposed materials to be used in constructing the latching mechanism include:

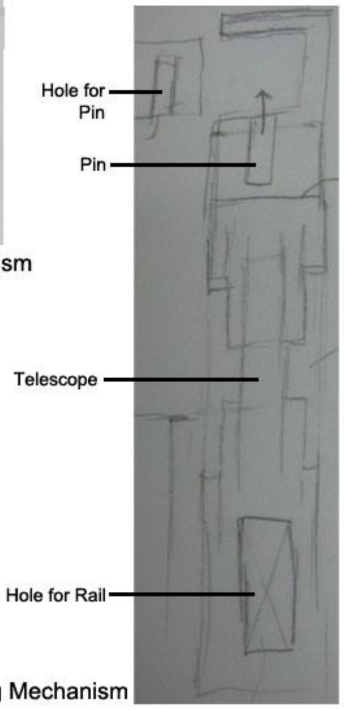
- Rail – selected to be light weight steel with matte finish for aesthetic appeal

- Receiver Piece – should be light weight steel with matte finish

The following image shows the initial sketches of the balloon locking mechanism and the pin locking mechanism.

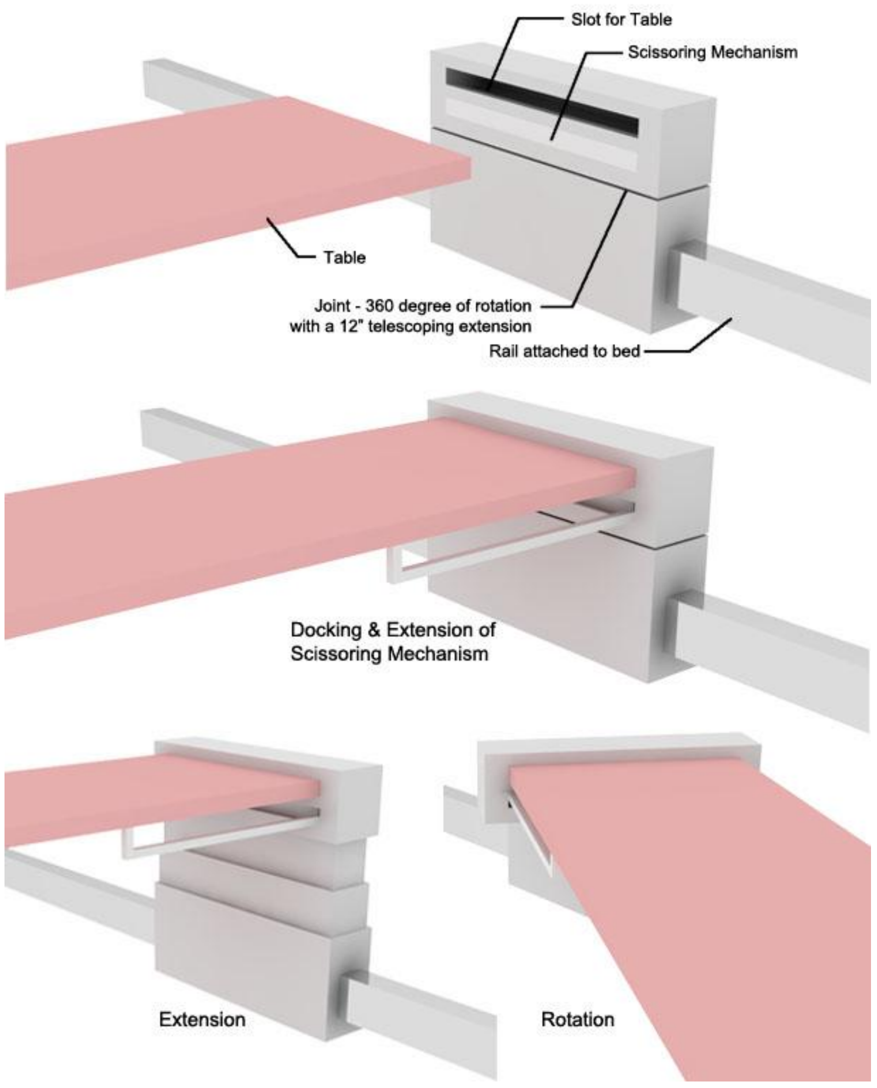


Idea for a Balloon Locking Mechanism



Idea for a Pin Locking Mechanism

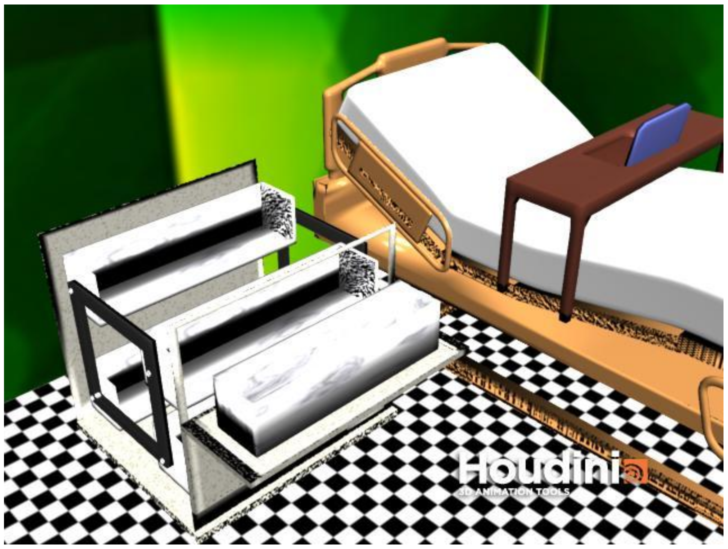
The image below illustrates the intended sequence for extending and locking the table in the opposite side thereafter.


Slot for Table
Scissoring Mechanism
Table
Joint - 360 degree of rotation with a 12" telescoping extension
Rail attached to bed
Docking & Extension of Scissoring Mechanism
Extension
Rotation

5.6 Previous Iterations of System Design

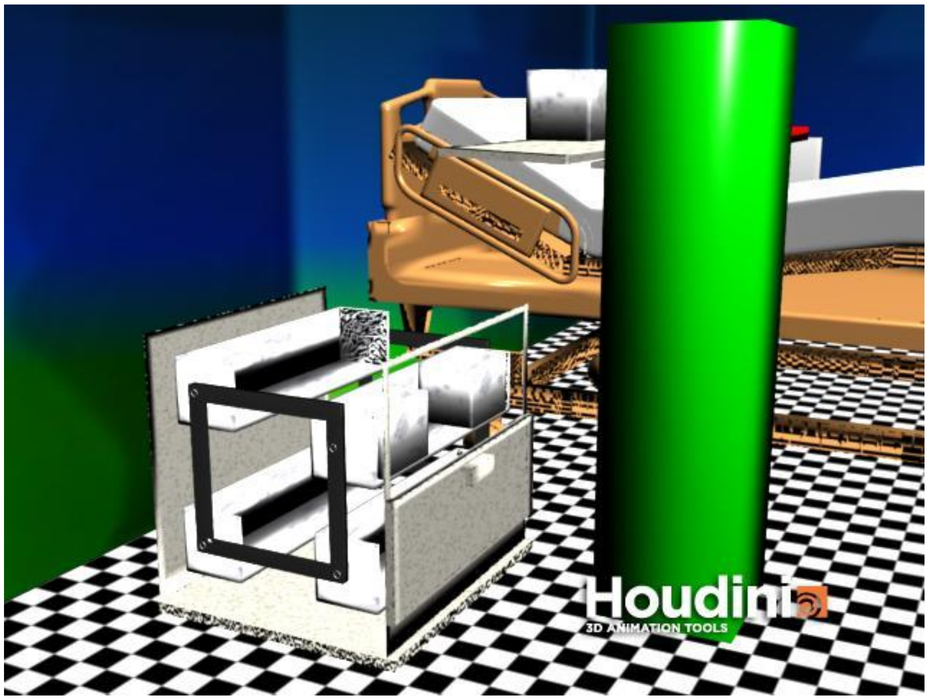
The nightstand and table configurations were refined several times before the final design was finalized. The images below will demonstrate the positive and negative aspects of each of the designs, and how each was refined to the final concept.

5.6.1 Table Sliding on Hospital Bed Rails



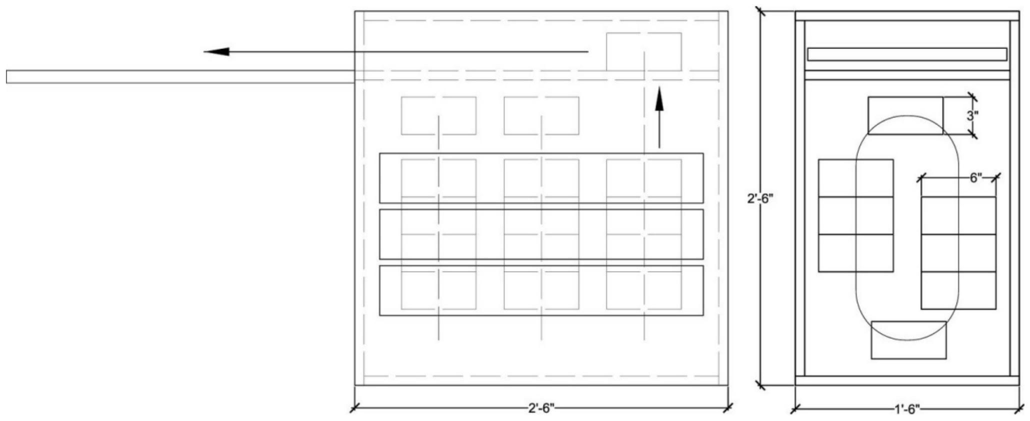
This first concept reflects the general concept of the current drawer system; it features the rotating compartments to be delivered to the user. However, each drawer must contain several items, and this could be very confusing to the user. Furthermore, this could result in a more complex method needed to index each item by the drawer system. This design of the table features a table sliding across the rails of a bedside table, but this idea was revoked due to its inability to be removed, caging the user in the bed.

5.6.2 Column Delivery of Nightstand Items



This idea iterated on the design in 5.4.3.2 by adding compartments to the drawers, enabling each item to be easily localized and indexed by the drawer system. An extra column structure, shown in green in the image above, was introduced to deliver items from the nightstand to the patient. However, this idea was completely ignored due to the android-like actions of the column.

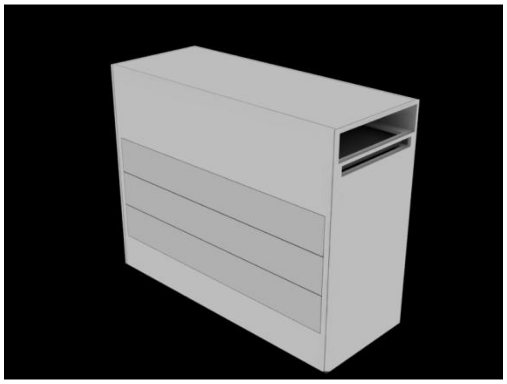
5.6.3 Design of Nightstand and Table Compartments

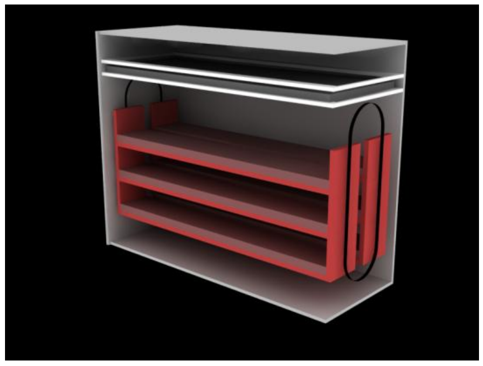


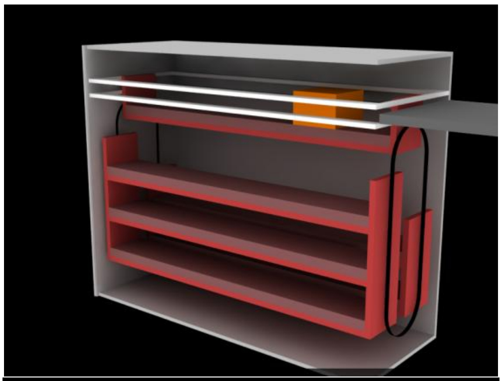
Above is a possible design for the table and nightstand. The table could be hidden inside the nightstand and could be deployed out. The boxes would be 6" x 6" x 3" (h x w x d), and there would be 24 of these boxes.

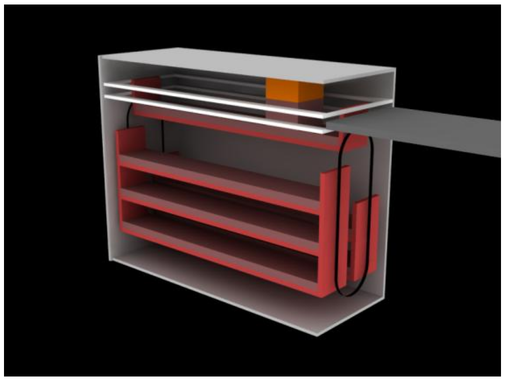
5.6.4 Ideal Nightstand Design

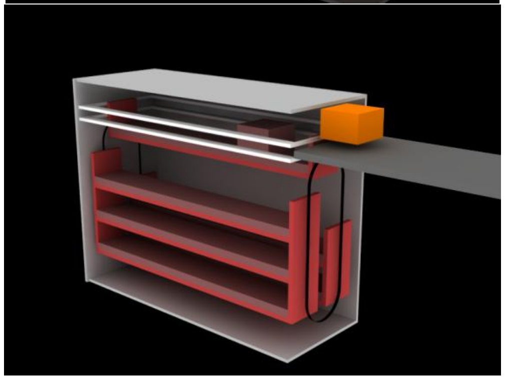
The following images outline a sequence of how the drawer system should work. There should be a row of 6 drawers, each approximately the size of a medical drawer that would be found in a hospital. Each drawer has an option to have several different size boxes. There would be a cut out in the 2 middle panels so the drawers could pop up through it. Then, a mechanism would pop the box up and then a mechanism will push the box off the drawer and down toward the table that is extended.





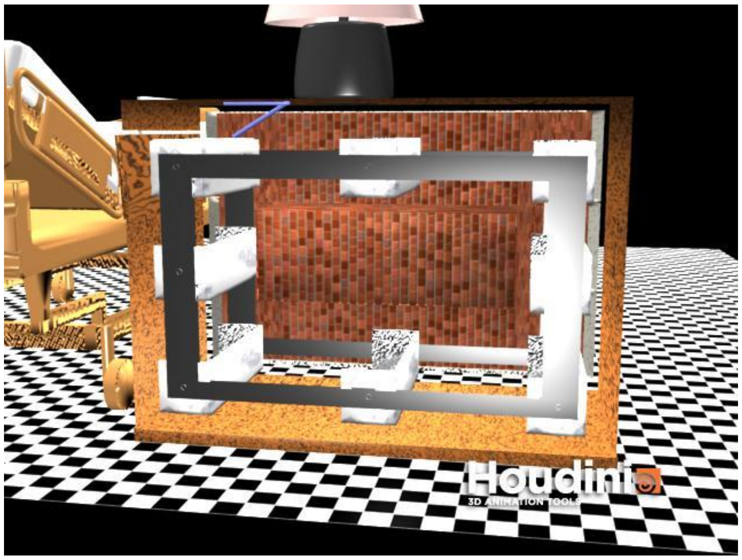


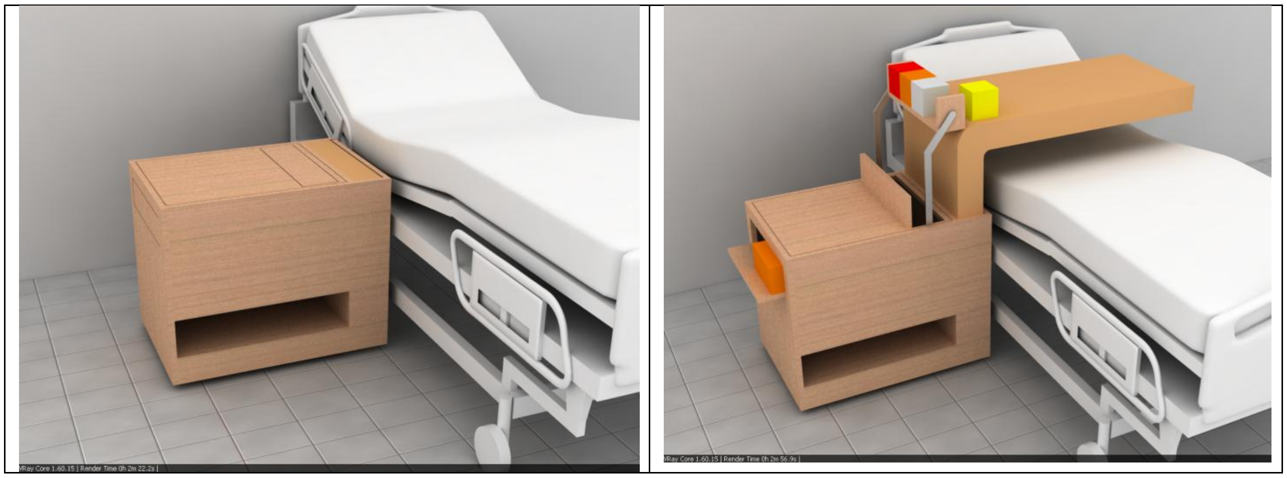




5.6.5 Alternate Physical Appearances of Ideal Nightstand

The following images are renderings of possible topological views of the nightstand. The latter two reflect the most likely method of implementation, although the table will most likely be much thinner than displayed in the image.





6 Future Work

6.1 Control Software and Artificial Intelligence

Several improvements could be made in order to maximize the benefits of the artificial intelligence and CTCS. The first is that the graphical user interface (GUI) should be made more attractive to users – something they'll be comfortable looking at as well as increasing usability. Furthermore, the system should have more inputs from the table or other sensing of the user such that more, richer responses can be gained from the system. With more inputs, it may be possible for the system to learn sequences of actions and predict possible actions in that manner. Even further, integrated with medical sensors such as heart rate or blood pressure monitors, a system aware of the physical state of the user can be established. This can lead to more responsive reactions to emergency situations, quickly offering the user counteractive pills such as nitroglycerine, offering water to the patient if their stats suggest dehydration, or even summoning nurses or doctors in the case of an urgent emergency.

In addition, in order to appropriately decide whether to deliver or not deliver an item, another mechanism needs to be created, which would be a no-operation (nop) operator. One can see the necessity of this in the system; all the rewards garnered from the system are rewards, nonetheless. Even if a production has a low reward score, the system will fire it if it has no alternatives. Thus, the nop operator, which suggests to do nothing should be proposed simultaneously. This would enable the system to truly choose to *not* deliver a certain item based on the garnered reward.

Lastly, the system should have more productions (rules) written for its operation. This would enable a richer set of template rules for the system to begin with to be able to carry out more complex reasoning.

6.2 Continuum Table

Apart from the conclusion that pneumatic muscles have proven to be very versatile, manageable, and controllable devices that are simple to make, the ability to use said muscles to control a continuum surface table has opened many doors for future work in the control and development of continuum surfaces. To further develop the effects that pneumatic muscles can have on continuum surfaces, more muscles with different mesh sleevings will be built and tested, for the key to changing the effect of the muscles lies with the size and pattern of the mesh sleeving. The effects of changing the diameter of the sleeving has been observed, so the main focus will be on seeing how different sleeving patterns change how the muscles move. The next step is to build muscles with mesh sleeving that allows for muscle expansion instead of contraction. It may also be desirable to explore the effect that different tube sizes have and also explore if there are lighter parts to make the muscles.

Once different muscles have been built, these muscles can then be placed onto the continuum surface table; then, one can observe the affect that different muscles have on table movement. Since there are many different muscles that can be built, there is the potential for the table to take many different configurations. So, there is plenty of potential work in trying all the different combinations of muscles and observing the effects they have on table movement. Furthermore, the placement of the muscles is also key in the table's movement. So, another area of future work lies in testing different muscle patterns on the table. As of now, the muscles are lying parallel to each other on the table. It may be desired to observe what happens when the muscles are snaked or lie in different directions on the table; a bat wing configuration is one which may also be tested.

The control of the table is another area in which future work can be performed. The table's actuation is currently controlled by two push buttons and two IR sensors, and the table's movement is pre-programmed. A more user-activated actuation of the table could be developed. So, instead of the table moving in a predetermined manner, the table would be able to respond to the subtle movements of the user; it would be able to "mold" to the person's current position as well as perform the predetermined motion sequences for certain scenarios. Therefore, more coding and an increase in the number of sensors would be necessary; the exploration of and use of different sensors is also possible.

6.3 Nightstand

The drawer and delivery system can be improved by incorporating different types of mechanisms other than the gear and cable. A robotic arm could be used to extract each drawer/compartment instead of using a kick-out mechanism. The compartments could be extracted on the top of the nightstand instead of the table. Currently, the continuous servo motors are able to pull the weight of the cardboard drawers. In the future, more powerful motors will need to be incorporated. The drawers could be placed in a stacked position instead of moving around a ferris wheel style gear system. With stacking compartments, each of the drawers could be moved within the nightstand by using a grid type map that directs the movement of each drawer.

Each of the compartments could be attached to the side of the nightstand, somewhere on the grid, and the compartments would be able to maneuver and rearrange in various combinations. This would incorporate more of an intelligent nightstand that handles navigation within and outside view. A mobile base would allow the nightstand to perform more tasks instead of staying in a constant position. The nightstand could be doubled as a “gopher” robot that can grab easy to manipulate objects that the user needs. The grasping part would be handled by the manipulator location within the nightstand. These are fruitful areas of research that can be explored further.

Appendix A: Instructions on Construction of Pneumatic Muscles

Supplies:

(1) High-Temperature Silicone Rubber Tubing Soft, 3/8" ID, 1/2" OD, 1/16" Wall
cut to desired length
http://www.mcmaster.com/#5236k15/=9ga04y

PAM Figure 1

(1) Expandable Mesh sleeving cut to desired length
http://www.mcmaster.com/#9142k34/=9ga0bm OR
(1) Heavy Duty Polyester Expandable Mesh Sleeving 5/8’’ ID, 5/16” to 1” Bundle Dia, cut to desired length
http://www.mcmaster.com/#9142k2/=9o8kcn OR
(1) Choose-A-Color Expandable Mesh Sleeving 1/2" ID, 1/4" to 3/4" Bundle Dia, cut to desired length
http://www.mcmaster.com/#9284k144/=9o8h55

PAM Figure 2

(1) Compression Tube Fitting Adapter – Female Pipe Adapter
http://www.mcmaster.com/#50915k226/=9ga0h4

PAM Figure 3

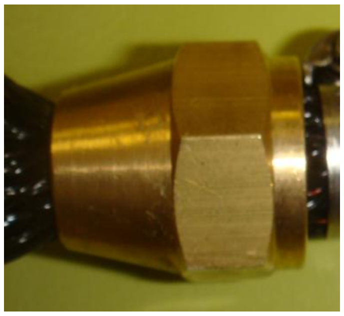
(2) Quick Connect – Male Adapter
Ace Hardware – PL-3005 (1/4OD(1/8CTS)x1/4MIP)

PAM Figure 4

(1) Compression Tube Fitting Long Nut
http://www.mcmaster.com/#50915k106/=9ga0mh

PAM Figure 5
(1) ½ Flare Plug
Ace Hardware – Part # 41211 (ABP2-8)

PAM Figure 6
(1) ½ Short Rod Nut
Ace Hardware – Part # 41154 (ABN1-8)



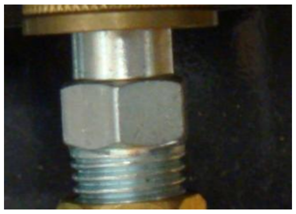
PAM Figure 7
(2) Compression Tube Fitting Tube Support – Brass
http://www.mcmaster.com/#50915k246/=9ga0wo

PAM Figure 8

(1) Flexible Polyerethane Tubing cut to necessary length
From lab – ¼' O.D.

PAM Figure 9

(1) Tru-Flate Plug: ¼” male NPT
Ace Hardware - #12-225


PAM Figure 10

(2) ¼ HB x ¼ FPT BARB
Ace Hardware - #4014247

PAM Figure 11

(6) PRT #23
Ace Hardware - #41915

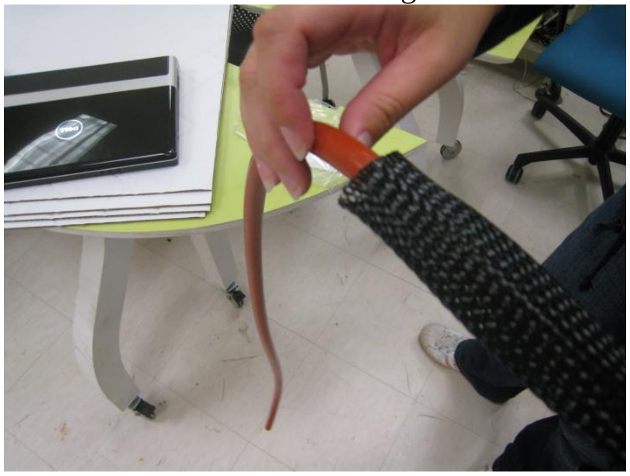
PAM Figure 12

Instructions:

1) Cut the silicone rubber tubing to 40 inches.

PAM Figure 13

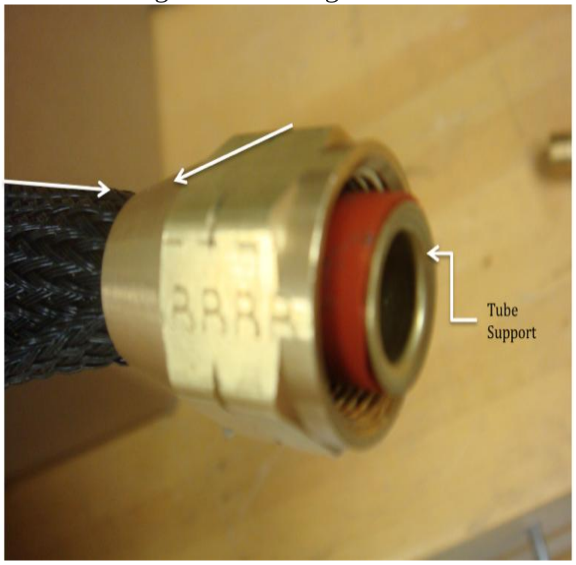
2) Slide the tubing into the mesh (the third mesh listed above was used for the muscles that appear in the final project for ECE 868); one must bunch the mesh section by section in order to move the tubing within the mesh.


PAM Figure 14

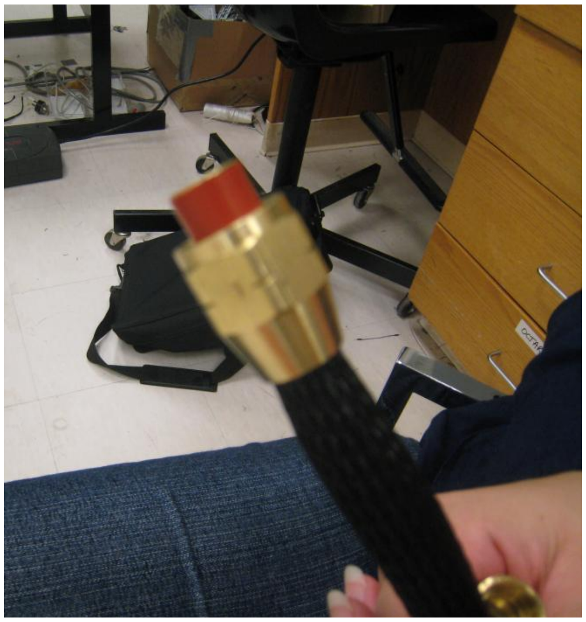
PAM Figure 15

3) Pull the tubing and the mesh sleeving at one end of the muscle through the small opening of the long nut.

4) Place a tube support into the tubing on that same end. It is important to leave a little bit of the tubing above the edge of the mesh so that a fitting cap can be placed on that end.

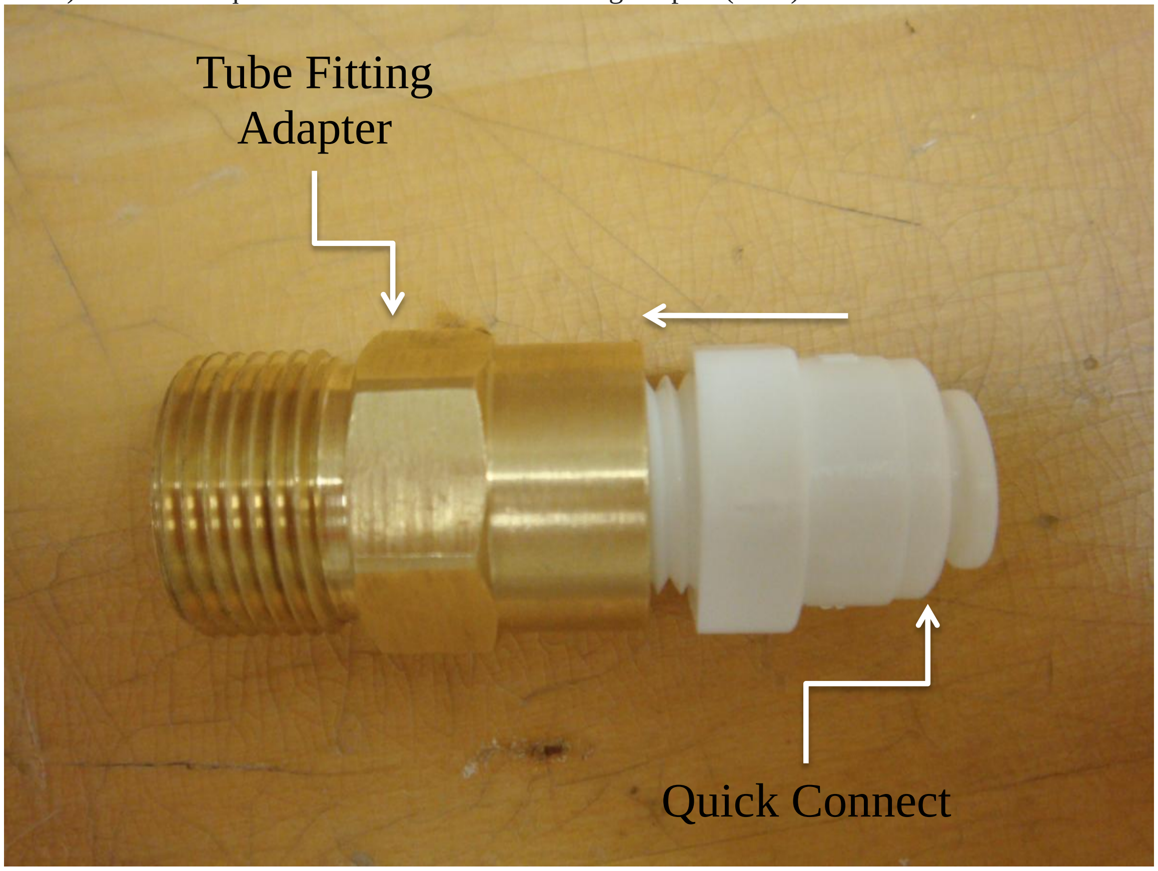


PAM Figure 16



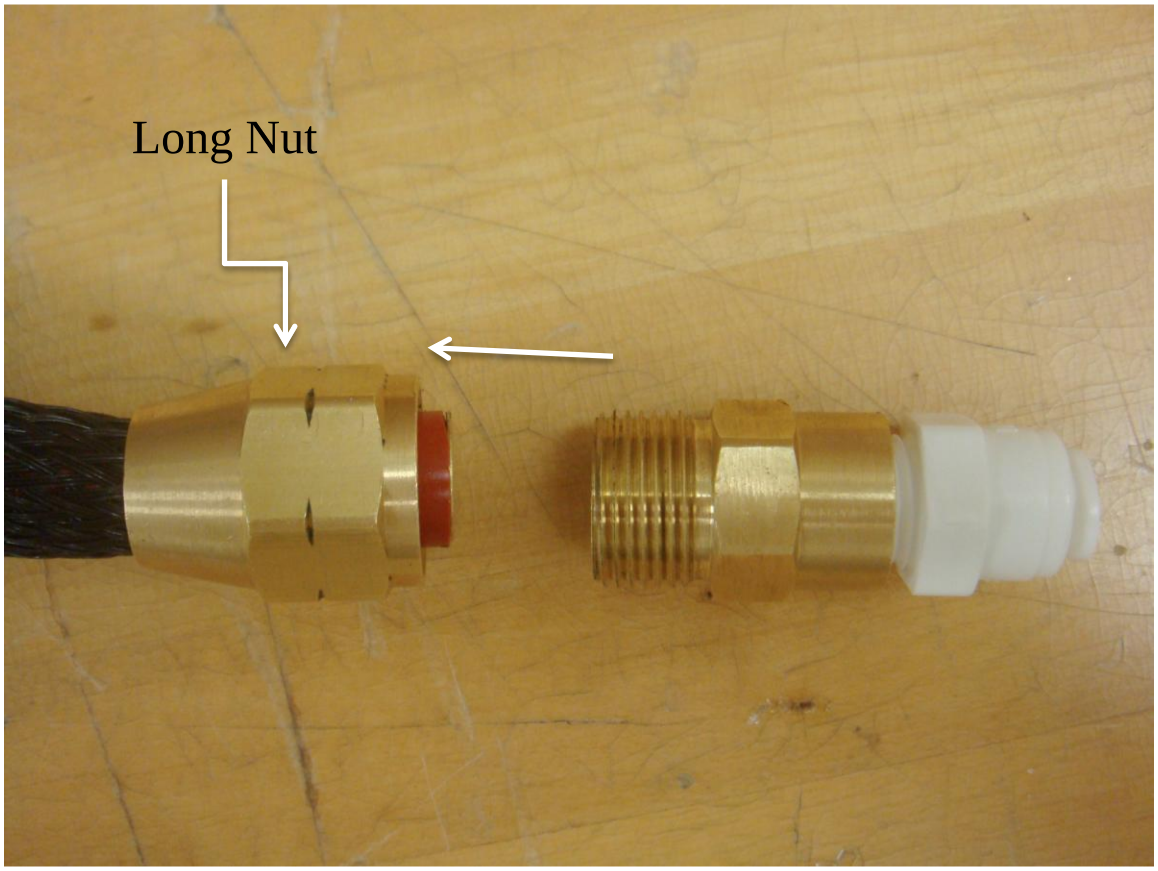
PAM Figure 17

5) Screw the quick-connect into the tube fitting adapter (male).

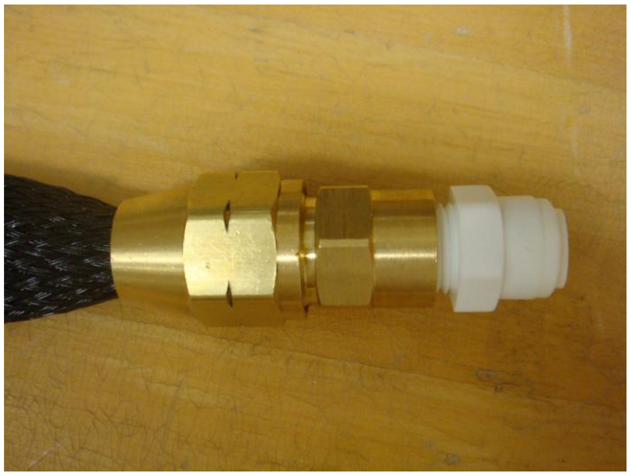


PAM Figure 18

6) Screw the other end of the tube fitting adapter into the long nut (female).

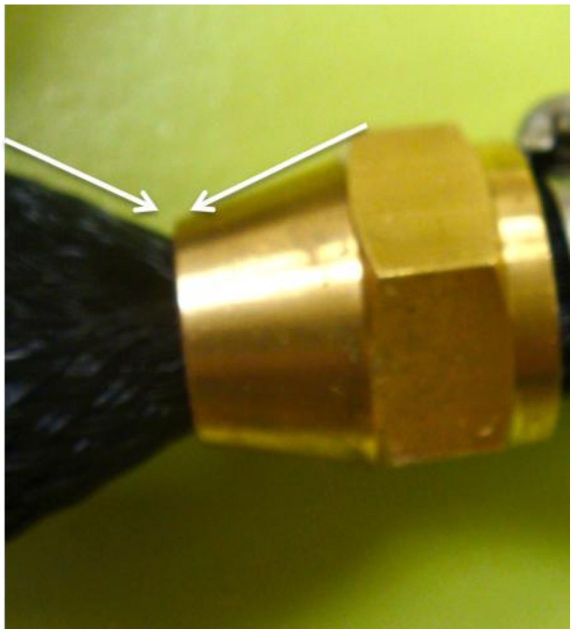


PAM Figure 19



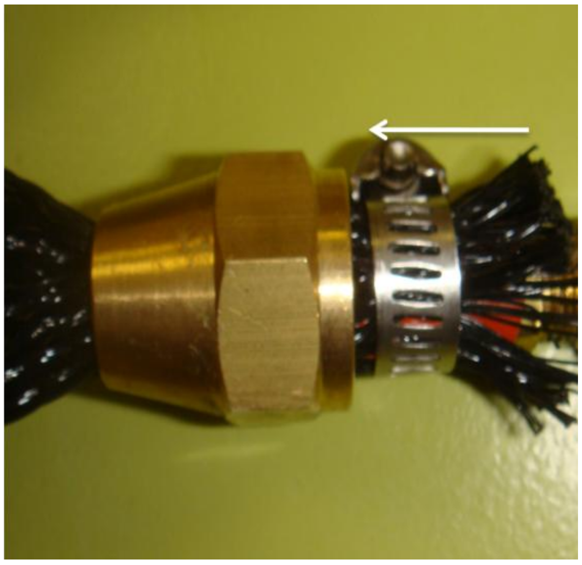
PAM Figure 20

7) Pull the tubing and mesh sleeving at the other end of the muscles through the ½ short rod nut; pull the tubing and sleeving through the rod so that they extend about an inch beyond the rod nut.



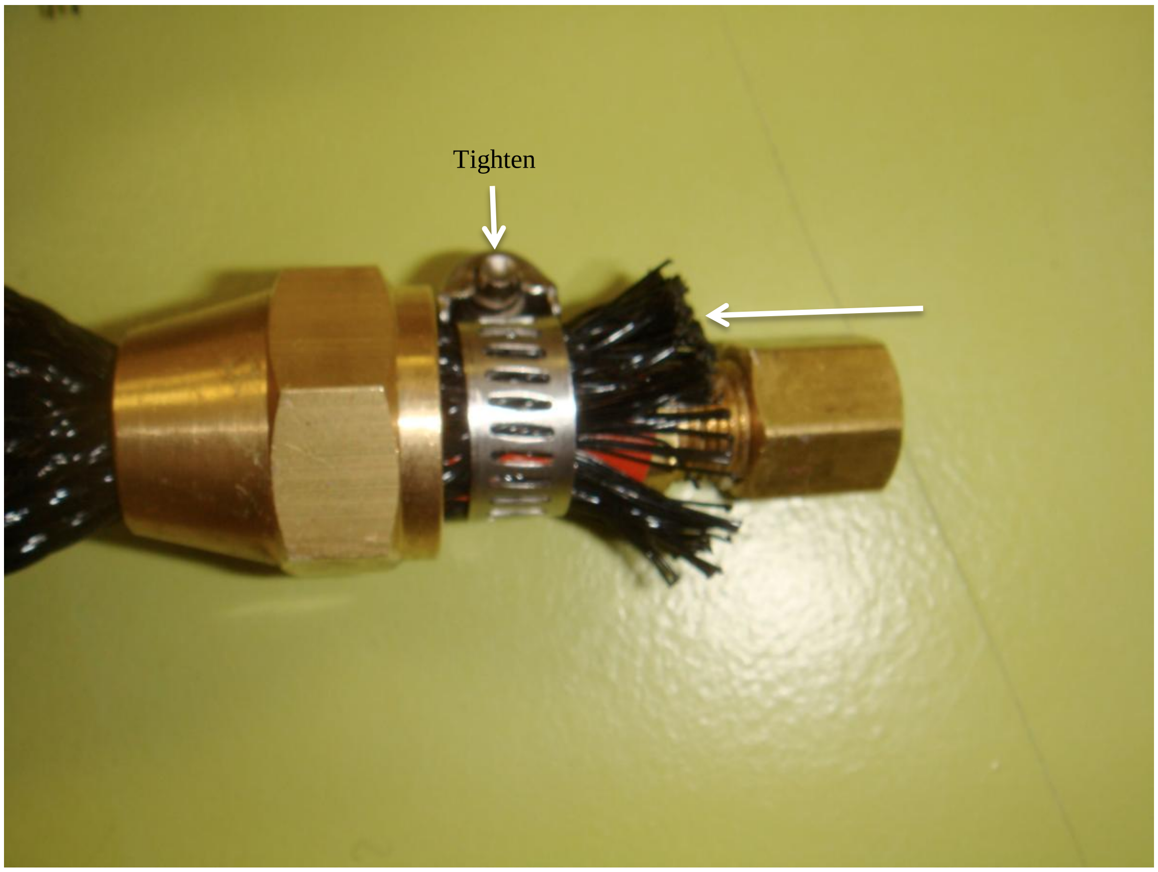
PAM Figure 21

8) Slide the PRT #23 over the mesh sleeving (which is covering the tubing) that is extended beyond the short rod nut.



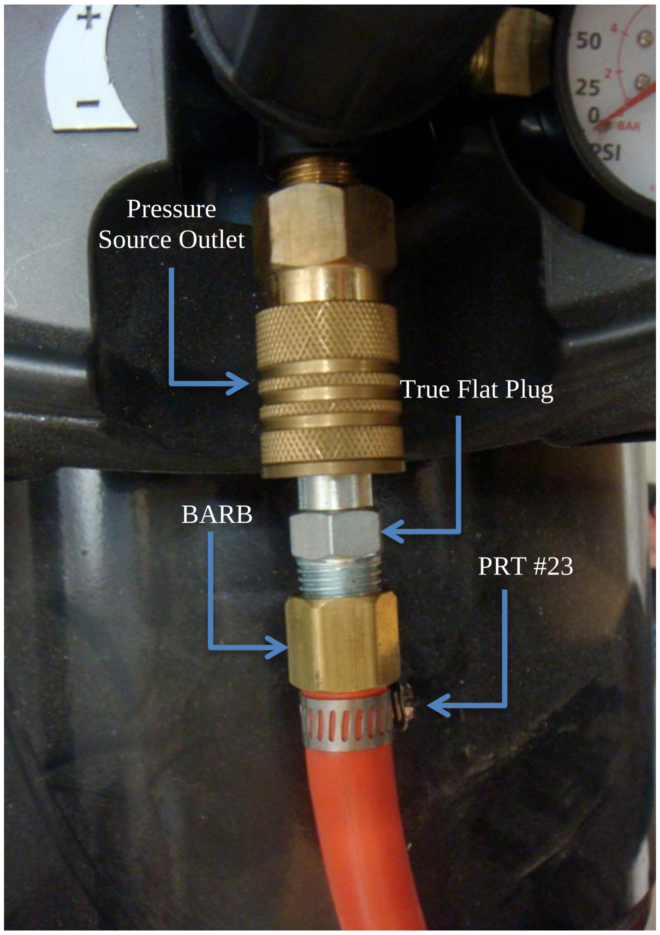
PAM Figure 22

9) Place the ½ flare plug into the end of the tubing, and then tighten the PRT #23 with a screwdriver around the mesh; tighten the PRT #23 so that it is tightly around the base of the flare plug.



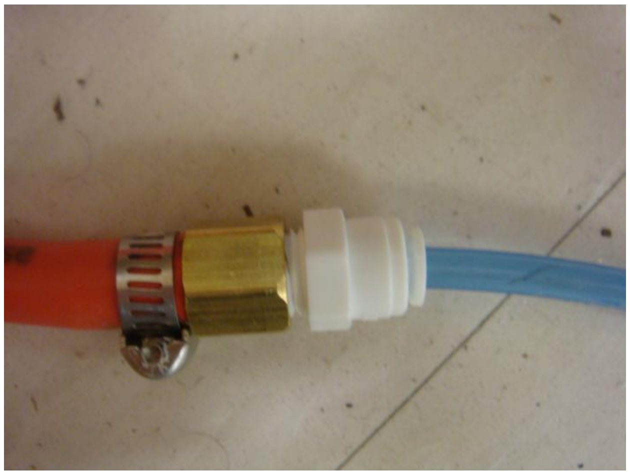
PAM Figure 23

10) Find the outlet of the pressure source and screw the True-Flat plug into the outlet; then, screw the BARB into the plug; lastly, attach the hose to the BARB and tighten the PRT #23 around it.



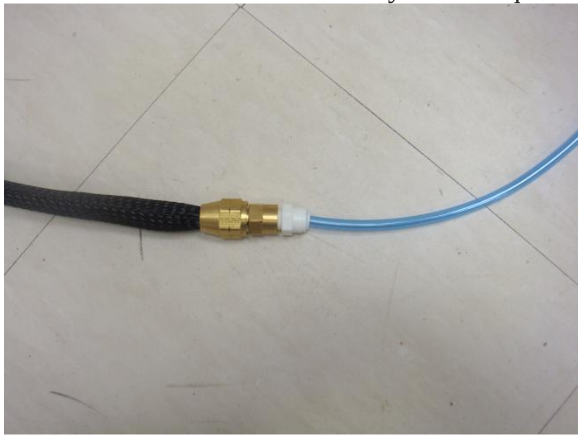
PAM Figure 24

11) Place a PRT #23 over the other end of the compressor hose, and place a BARB onto the end of the hose.
12) Place a quick-connect into the BARB.
13) Feed the Polyurethane tubing into the quick connect.



PAM Figure 25

14) Feed the other end of the Polyurethane tubing into the pressure regulators that are utilized for muscle control or directly into the quick connects of the muscles.



PAM Figure 26

Notes:

- The inner and outer diameters of the tubing are chosen based on the desired size of the muscles.
- The mesh sleeving inner diameter should match the outer diameter of the tubing for maximum muscle contraction.
- If the mesh sleeving is cut, the ends must be burnt with a lighter.

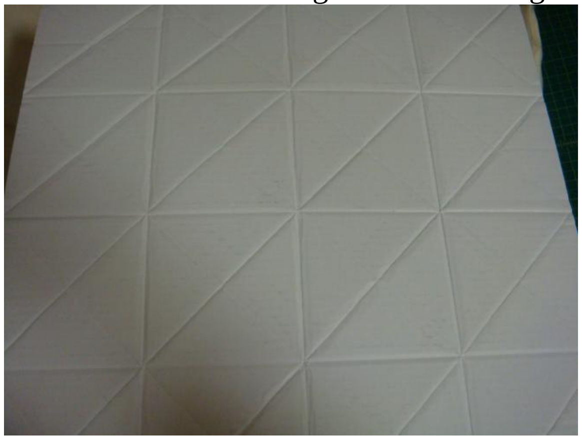
Appendix B: Instructions on Construction of Continuum Surface Table

Supplies:

- 1 ½” foam (Wal-Mart)
- 1” foam (Target)
- 10 ½ ”x32 ½ ” piece of cardboard (lab supplies)
- 20 large and 40 small zip ties (Wal-Mart)
- hot glue (Wal-Mart)
- 4 pneumatic muscles (see “How to Make Pneumatic Air Muscles” document)

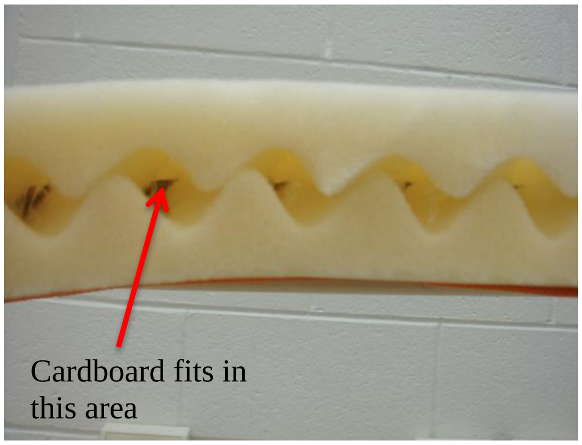
Procedure:

1) Tessellate the cardboard with triangles; half of the cardboard should be tessellated with larger triangles and the other half with triangles half the size of the larger ones; the tessellations shown in figure 1 are the larger ones.



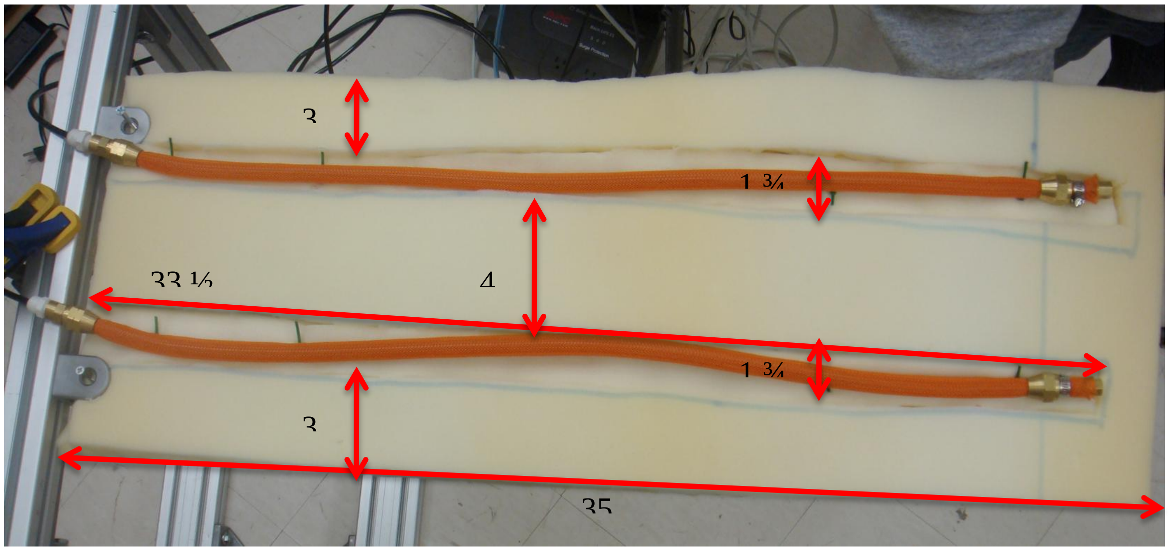
CST Figure 27

2) Cut two 13 ½ ”x35” pieces of the 1 ½” foam.
3) Align the cardboard in the center of one of the foam pieces (make sure that the cardboard is placed on the side with the egg crate texture), and then place the second piece of foam on top of the cardboard (also make sure that the egg crate textured side is touching the cardboard).



CST Figure 28

4) Use hot glue to secure the foam to the outside edges of the cardboard.
5) Place the pneumatic muscles on top of the foam in their desired positions and outline the muscles; an example of the outlines and the measurements is shown below in figure 3. Make sure to align the end of the muscle that connects to the air tubes with the edge of the foam.

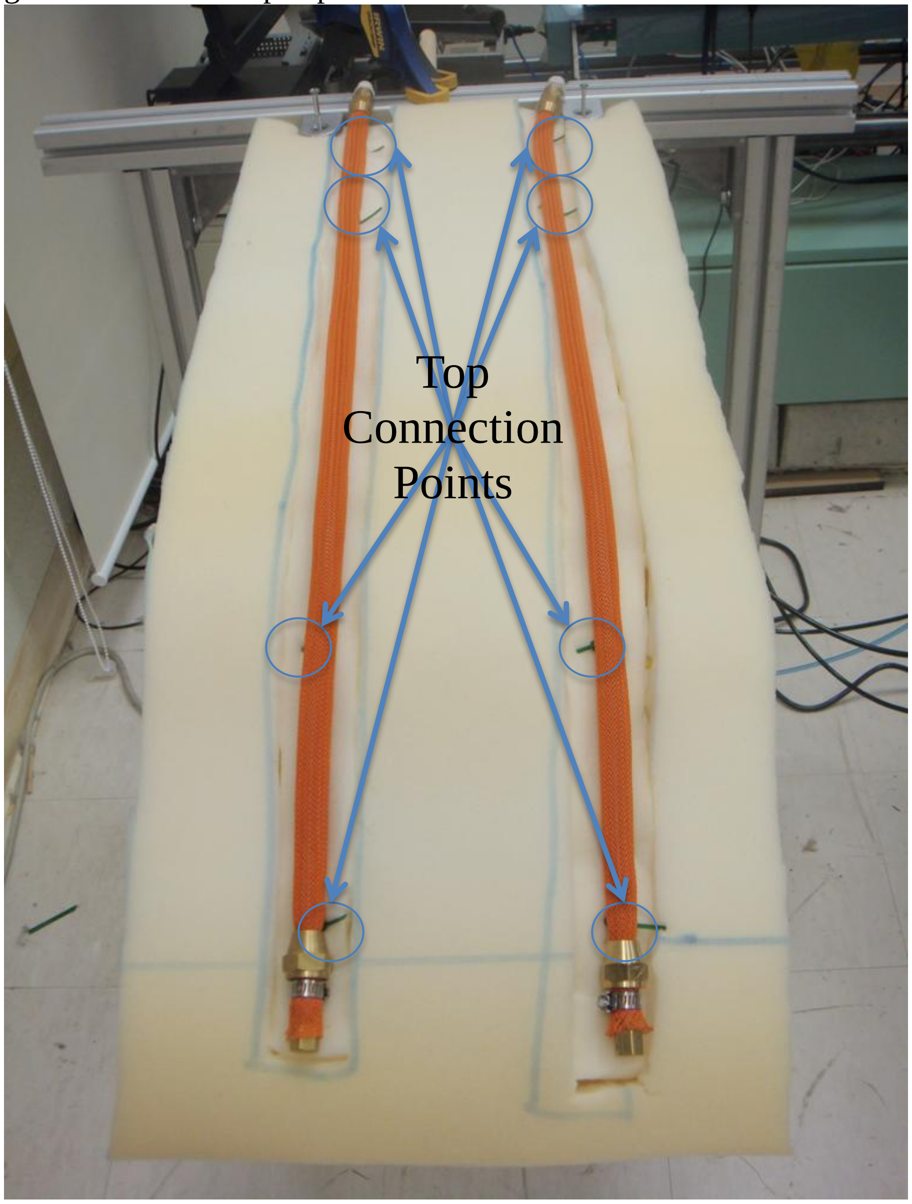


CST Figure 29

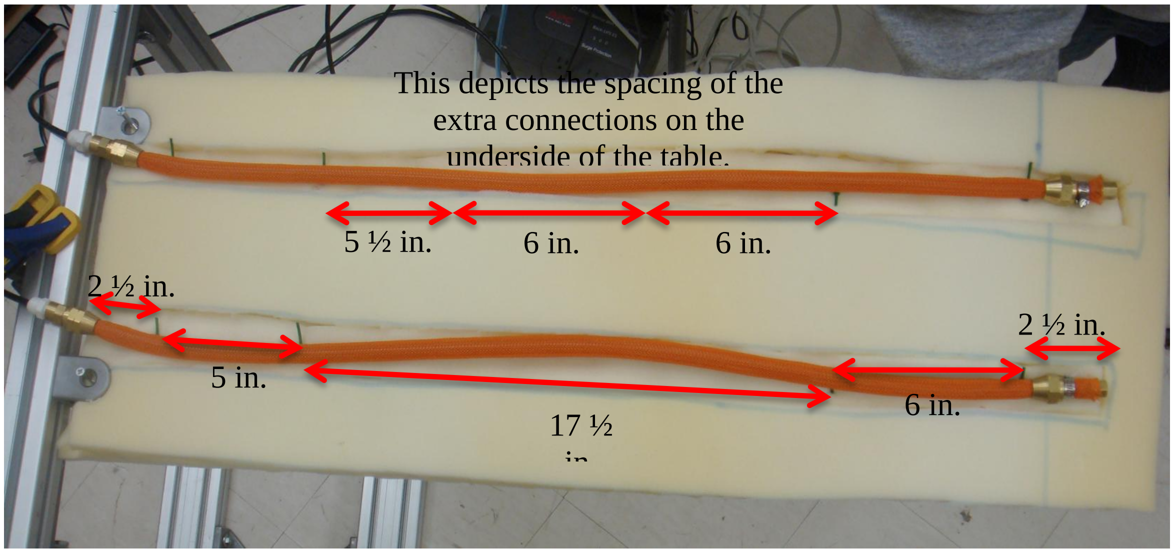
6) Once the outlines have been drawn, cut that section of the foam out; steps 5 and 6 are performed for the foam pieces on both the top and bottom sides of the table.
7) Once the area for the muscles has been cut, secure the remaining foam to the cardboard with hot glue.
8) Proceed to gather 20 small zip ties, and pull each one through so that a small loop is formed.

CST Figure 30

9) On the top side of the table, glue four of these zip ties to the points of the tessellated triangles in the cardboard in one of the areas where the foam was cut out in step 6; glue two ties at the top end of the table and two at the bottom end of the table. Measurments for the zip ties are displayed below in figure 6. Important: make sure that the zip ties are glued so that the loop is parallel with the side of the table that has the shortest width.

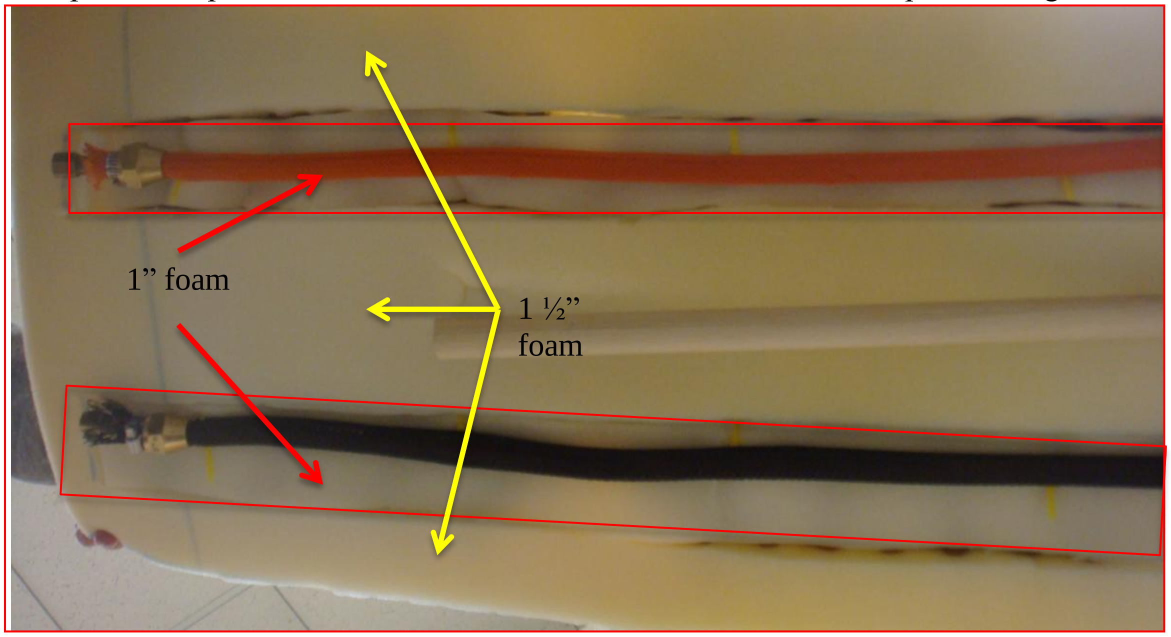


CST Figure 31


This depicts the spacing of the extra connections on the underside of the table.
5 ½ in.
6 in.
6 in.
2 ½ in.
5 in.
17 ½ in.
2 ½ in.
6 in.

CST Figure 32

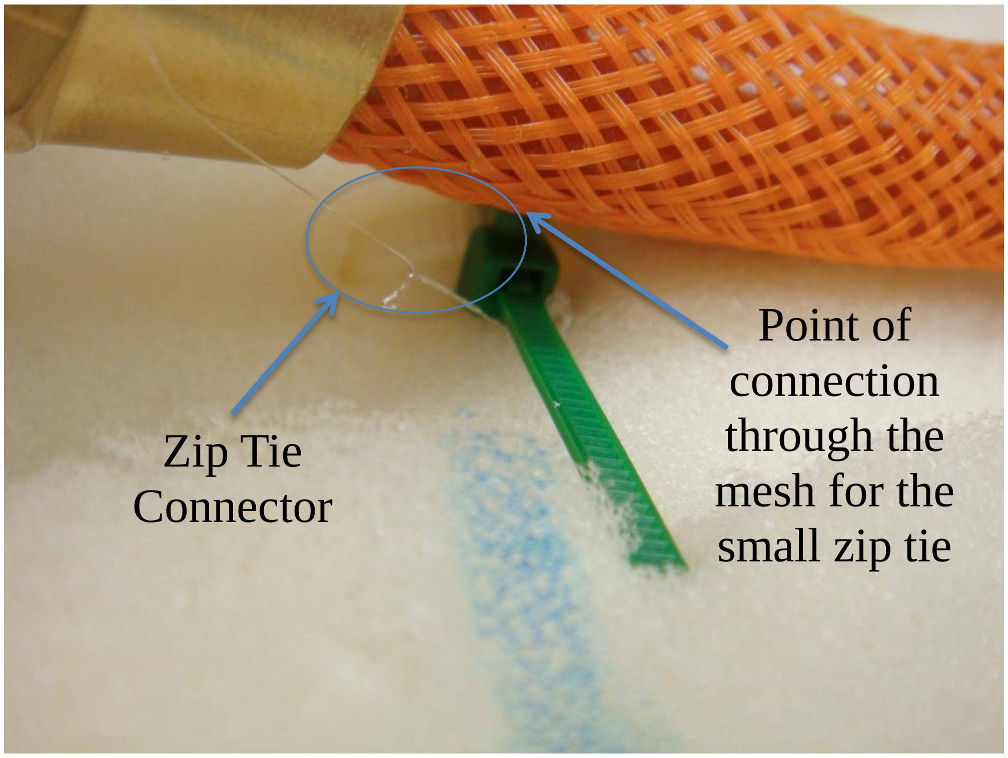
10) On the bottom side of the table, repeat step 9. Then, glue two additional zip ties spaced in the middle section of the table as shown above; do this for both muscles areas (this adds a total of 4 zip ties to the bottom of the table).
11) Cut sections of the 1" foam that are the same size as the foam areas that had been cut out in step 6. Then, place the 1" foam sections in the cut out sections as depicted in figure 7.



CST Figure 33

12) Gather 20 large zip ties.
13) For each zip tie that was glued in steps 9 and 10, feed the leading end of a large zip tie through the loop. Then, feed that leading end up through the 1" foam piece that corresponds to that set of zip ties. Next, feed that point back down through the foam. Lastly, tie the zip tie tightly.
14) Glue the foam to the cardboard.

15) Place the pneumatic muscles on top of the 1” foam areas that now have the zip ties fasteners; again, ensure that the end of the muscles that have the quick connects are at the edge of the table.
16) Gather 20 small zip ties.
17) For each zip tie connector in the foam, feed one end of a small zip tie through the braiding on the mesh sleeve parallel to the end pieces of the muscles. Then, feed the end of the zip tie through the zip tie connector in the foam. Lastly, pull the zip tie tightly. Figure 8 shows the end result of this step.



CST Figure 34

Appendix C: ComforTable and Nightstand Control Software – Relevant Code

Example production proposing and apply operator to deliver an item to the user

```
#propose*deliver-items proposes that a certain item should be delivered out of
the nightstand.  It first tests that there is some item and records the time
of day for later user in the rule.  It also tests that the user requested the
item, and the item hasn’t already been delivered.  It lastly tests that the
Table is already present, because if it isn’t the system will not deliver the
item before it.  If all these conditions are met, then the system suggests
that a certain item should be delivered.

sp {nightstand-productions*propose*deliver-items
   (state <s> ^name nightstand-productions ^io.input-link <il>)
   (<il> ^item <reqitem> ^time-of-day <tod>)
   (<reqitem> ^item-location requested ^name <itemname>)
   (<s> -^items-present <itemname>)
   (<s> ^items-present Table)
-->
   (<s> ^operator <op> +)
   (<op> ^name deliver-items
         ^req-item <reqitem>
         ^at-time <tod>)
}

#The action of apply*deliver-items is to deliver the items that have been
proposed by the above rule.  As can be seen beneathe the arrow, it changes
Soar’s current knowledge of the situation by adding the item to its internal
list of “items-present,” and then writes the relevant information back to the
C++ program controlling the GUI.  As the information is cycled in, it is
processed and the appropriate steps are taken.

sp {apply*deliver-items
   (state <s> ^operator <op> ^io.output-link <ol>)
   (<op> ^name deliver-items
         ^req-item <reqitem>
         ^at-time <tod>)
   (<reqitem> ^compartment <comp> ^drawer <myd> ^name <itemname>)
-->
   (<s> ^items-present <itemname>)
   (<reqitem> ^called-at <tod>)
   (<ol> ^item <reqitem>)
   (<reqitem> ^my-d <myd> ^my-comp <comp> ^name <itemname>)
   (write <itemname> | is being delivered. | (crlf))
}
```

Nightstand Brain.cpp – Code interfacing with Soar Kernel and Table for Reward

```
#include "NightstandBrain.h"
#include "resource.h"
#include "utility_functions.h"
#include "Serial.h"

sml::Agent* pAgent;
//Serial TableComm;
//Serial NightstandComm;
std::string rewardStatus;
sml::StringElement *pWME1e, *pWME2e, *pWME3e, *pWME4e, *pWME5e, *pWME6e, *pWME7e, *pWME8e,
*pWME9e, *pWME10e, *pWME11e, *pWME12e;
Serial TablePort(tstring(L"COM9"), 9600);

NightstandBrain::NightstandBrain(void):lastWMEno(0),outputID(0),TimeCounter(0)
{
	rewardStatus = "not_app";

	pKernel = Kernel::CreateKernelInNewThread("SoarKernelSML");

	if (pKernel->HadError())
	{
		//throw error
	}

	pAgent = pKernel->CreateAgent("Nightstand Helper");

	if (pKernel->HadError())
	{
		//throw error
	}

	std::string cmd = "learn --enable";
	char const* pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	cmd = "set-stop-phase --before --input";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	cmd = "rl --set learning on";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	cmd = "epmem --set learning on";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());
	cmd = "epmem --set database file";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	cmd = "epmem --set path C:/Documents and Settings/Charreau/My Documents/ECE 868 - Architectural
Robotics/Project 2 Files/Nightstand Intelligence/SMARTStand/SMARTStand Productions/Episodes";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	cmd = "epmem --set trigger dc";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	cmd = "epmem --set phase selection";
	pResult = pKernel->ExecuteCommandLine(cmd.c_str(), pAgent->GetAgentName());

	bool result = pAgent->Agent::SpawnDebugger(12121, "C:\\Soar-Suite-9.3.0-win-x86\\");

	pAgent->LoadProductions("C://Documents and Settings//Charreau//My Documents//ECE 868 - Architectural
Robotics//Project 2 Files//Nightstand Intelligence//SMARTStand//SMARTStand Productions//nightstand-
productions.soar");

	if (pAgent->HadError())
	{
		TRACE("\n\n\nLoad Productions Failed.\n\n\n");
	}

	pInputLink = pAgent->GetInputLink() ;

	pID1     = pAgent->CreateIdWME(pInputLink, "item");
	pWME1a = pAgent->CreateStringWME(pID1, "name", "Table");
	pWME1b = pAgent->CreateIntWME(pID1, "drawer", 1);
	pWME1c = pAgent->CreateIntWME(pID1, "compartment", 1);
	pWME1d = pAgent->CreateStringWME(pID1, "time-to-deliver", "nil");
	pWME1e = pAgent->CreateStringWME(pID1, "item-location", "in");

	pID2     = pAgent->CreateIdWME(pInputLink, "item");
	pWME2a = pAgent->CreateStringWME(pID2, "name", "Jewelry");
	pWME2b = pAgent->CreateIntWME(pID2, "drawer", 1);
	pWME2c = pAgent->CreateIntWME(pID2, "compartment", 2);
	pWME2d = pAgent->CreateStringWME(pID2, "time-to-deliver", "nil");
	pWME2e = pAgent->CreateStringWME(pID2, "item-location", "in");

	pID3     = pAgent->CreateIdWME(pInputLink, "item");
	pWME3a = pAgent->CreateStringWME(pID3, "name", "Work-Items");
	pWME3b = pAgent->CreateIntWME(pID3, "drawer", 1);
	pWME3c = pAgent->CreateIntWME(pID3, "compartment", 3);
	pWME3d = pAgent->CreateStringWME(pID3, "time-to-deliver", "nil");
	pWME3e = pAgent->CreateStringWME(pID3, "item-location", "in");
```

```
pID4     = pAgent->CreateIdWME(pInputLink, "item");
pWME4a = pAgent->CreateStringWME(pID4, "name", "Pens");
pWME4b = pAgent->CreateIntWME(pID4, "drawer", 2);
pWME4c = pAgent->CreateIntWME(pID4, "compartment", 1);
pWME4d = pAgent->CreateStringWME(pID4, "time-to-deliver", "nil");
pWME4e = pAgent->CreateStringWME(pID4, "item-location", "in");

pID5     = pAgent->CreateIdWME(pInputLink, "item");
pWME5a = pAgent->CreateStringWME(pID5, "name", "Book");
pWME5b = pAgent->CreateIntWME(pID5, "drawer", 2);
pWME5c = pAgent->CreateIntWME(pID5, "compartment", 2);
pWME5d = pAgent->CreateStringWME(pID5, "time-to-deliver", "nil");
pWME5e = pAgent->CreateStringWME(pID5, "item-location", "in");

pID6     = pAgent->CreateIdWME(pInputLink, "item");
pWME6a = pAgent->CreateStringWME(pID6, "name", "Glasses");
pWME6b = pAgent->CreateIntWME(pID6, "drawer", 2);
pWME6c = pAgent->CreateIntWME(pID6, "compartment", 3);
pWME6d = pAgent->CreateStringWME(pID6, "time-to-deliver", "nil");
pWME6e = pAgent->CreateStringWME(pID6, "item-location", "in");

pID7     = pAgent->CreateIdWME(pInputLink, "item");
pWME7a = pAgent->CreateStringWME(pID7, "name", "Family-Pictures");
pWME7b = pAgent->CreateIntWME(pID7, "drawer", 3);
pWME7c = pAgent->CreateIntWME(pID7, "compartment", 1);
pWME7d = pAgent->CreateStringWME(pID7, "time-to-deliver", "nil");
pWME7e = pAgent->CreateStringWME(pID7, "item-location", "in");

pID8     = pAgent->CreateIdWME(pInputLink, "item");
pWME8a = pAgent->CreateStringWME(pID8, "name", "Bible");
pWME8b = pAgent->CreateIntWME(pID8, "drawer", 3);
pWME8c = pAgent->CreateIntWME(pID8, "compartment", 2);
pWME8d = pAgent->CreateStringWME(pID8, "time-to-deliver", "nil");
pWME8e = pAgent->CreateStringWME(pID8, "item-location", "in");

pID9     = pAgent->CreateIdWME(pInputLink, "item");
pWME9a = pAgent->CreateStringWME(pID9, "name", "Journal");
pWME9b = pAgent->CreateIntWME(pID9, "drawer", 3);
pWME9c = pAgent->CreateIntWME(pID9, "compartment", 3);
pWME9d = pAgent->CreateStringWME(pID9, "time-to-deliver", "nil");
pWME9e = pAgent->CreateStringWME(pID9, "item-location", "in");

pID10   = pAgent->CreateIdWME(pInputLink, "item");
pWME10a = pAgent->CreateStringWME(pID10, "name", "Pills");
pWME10b = pAgent->CreateIntWME(pID10, "drawer", 4);
pWME10c = pAgent->CreateIntWME(pID10, "compartment", 1);
```

```
	pWME10d = pAgent->CreateStringWME(pID10, "time-to-deliver", "nil");
	pWME10e = pAgent->CreateStringWME(pID10, "item-location", "in");

	pID11   = pAgent->CreateIdWME(pInputLink, "item");
	pWME11a = pAgent->CreateStringWME(pID11, "name", "Money");
	pWME11b = pAgent->CreateIntWME(pID11, "drawer", 4);
	pWME11c = pAgent->CreateIntWME(pID11, "compartment", 2);
	pWME11d = pAgent->CreateStringWME(pID11, "time-to-deliver", "nil");
	pWME11e = pAgent->CreateStringWME(pID11, "item-location", "in");

	pID12   = pAgent->CreateIdWME(pInputLink, "item");
	pWME12a = pAgent->CreateStringWME(pID12, "name", "Checkbook");
	pWME12b = pAgent->CreateIntWME(pID12, "drawer", 4);
	pWME12c = pAgent->CreateIntWME(pID12, "compartment", 3);
	pWME12d = pAgent->CreateStringWME(pID12, "time-to-deliver", "nil");
	pWME12e = pAgent->CreateStringWME(pID12, "item-location", "in");
	//pID5    = pAgent->CreateIdWME(pInputLink, "time");
	//pWME17	= pAgent->CreateIntWME(pInputLink, "time-of-day", GetTimeInMinutes());
	pWMEA = pAgent->CreateIntWME(pInputLink, "time-of-day", TimeCounter*30+420);
	pWMER = pAgent->CreateIntWME(pInputLink, "reward", 0);

	pAgent->Commit();

	int outputNotifications=0;
	outputID = pAgent->RegisterForOutputNotification(HandleOutputEvent, &outputNotifications);

}

NightstandBrain::~NightstandBrain(void)
{
	pKernel->Shutdown();
	delete pKernel;
}

void NightstandBrain::ResetLocations(void)
{
	pAgent->Update(pWME1e, "in");
	pAgent->Update(pWME2e, "in");
	pAgent->Update(pWME3e, "in");
	pAgent->Update(pWME4e, "in");
	pAgent->Update(pWME5e, "in");
	pAgent->Update(pWME6e, "in");
	pAgent->Update(pWME7e, "in");
	pAgent->Update(pWME8e, "in");
	pAgent->Update(pWME9e, "in");
	pAgent->Update(pWME10e, "in");
	pAgent->Update(pWME11e, "in");
	pAgent->Update(pWME12e, "in");

	pAgent->Commit();
	pAgent->RunSelf(4);

}

void NightstandBrain::UpdateTime(void)
{
	TimeCounter++;
	pAgent->Update(pWMEA, TimeCounter*30+420);
	pAgent->RunSelf(3);

	if(TimeCounter==24)
		TimeCounter=-1;
}

void NightstandBrain::RequestItemtoWM(UINT nID)
{
	if(nID==Table_Button)
	{
		pAgent->Update(pWME1e, "requested");
	}
	if(nID==Jewelry_Button)
	{
		pAgent->Update(pWME2e, "requested");
	}
	if(nID==Bible_Button)
	{
		pAgent->Update(pWME8e, "requested");
	}
	if(nID==Books_Button)
	{
		pAgent->Update(pWME5e, "requested");
	}
	if(nID==Checkbook_Button)
	{
		pAgent->Update(pWME12e, "requested");
	}
	if(nID==Glasses_Button)
	{
		pAgent->Update(pWME6e, "requested");
	}
	if(nID==Pics_Button)
	{
		pAgent->Update(pWME7e, "requested");
	}
```

```
        if(nID==WorkDocs_Button)
        {
                pAgent->Update(pWME3e, "requested");
        }
        if(nID==Pens_Button)
        {
                pAgent->Update(pWME4e, "requested");
        }
        if(nID==Journal_Button)
        {
                pAgent->Update(pWME9e, "requested");
        }
        if(nID==Pills_Button)
        {
                pAgent->Update(pWME10e, "requested");
        }
        if(nID==Money_Button)
        {
                pAgent->Update(pWME11e, "requested");
        }

        pAgent->RunSelf(3);
}

void NightstandBrain::FixItemLocations(std::string name)
{
                if(name == "Table")
                        pAgent->Update(pWME1e, "out");
                else if(name =="Jewelry")
                        pAgent->Update(pWME2e, "out");
                else if(name =="Work-Items")
                        pAgent->Update(pWME3e, "out");
                else if(name =="Pens")
                        pAgent->Update(pWME4e, "out");
                else if(name =="Book")
                        pAgent->Update(pWME5e, "out");
                else if(name =="Glasses")
                        pAgent->Update(pWME6e, "out");
                else if(name =="Family-Pictures")
                        pAgent->Update(pWME7e, "out");
                else if(name =="Bible")
                        pAgent->Update(pWME8e, "out");
                else if(name =="Journal")
                        pAgent->Update(pWME9e, "out");
                else if(name =="Pills")
                        pAgent->Update(pWME10e, "out");
                else if(name =="Money")
                pAgent->Update(pWME11e, "out");
            else if(name =="Checkbook")
                pAgent->Update(pWME12e, "out");
}

void NightstandBrain::HandleOutputEvent(void* pUserData, Agent* pAgent)
{
    rewardStatus = "waiting_on";
    TRACE("\n IT WORKED!!!\n");
    int numberCommands = pAgent->GetNumberCommands();
    Identifier* pCommand;
    int i, j, numchars =0;
    WMElement* pID;
    for(i=0; i<numberCommands; i++)
    {

        /* Command so we can add status complete */
        pCommand = pAgent->GetCommand(i);
        const char* name = pCommand->GetParameterValue("name");
        pCommand->AddStatusComplete();

        /* Fix "item-location" WMEs to OUT */
        NightstandBrain::FixItemLocations(name);

        /* Write to the nightstand the drawers that need to be delivered. */
        char buffer[10];
        int numBytes=0, reward=0;

        int start_time=GetTimeInSeconds();
        int fin_time = start_time;
        int time_to_wait = 4;

        while(fin_time - start_time < time_to_wait)
        {
            numBytes = TablePort.read(buffer, 10, false);

            while(numBytes<=0 && fin_time-start_time < time_to_wait)
            {
                numBytes = TablePort.read(buffer, 10, false);
                fin_time=GetTimeInSeconds();
                if(numBytes>0)
                    break;
            }

        }
```

```
                    reward = atoi(buffer);
                    if(reward>0)
                    {
                              rewardStatus = "received";
                    }

                    pID = (pAgent->GetInputLink())->FindByAttribute("reward", 0);
                    pAgent->Update(pID->ConvertToIntElement(), reward);

                    pAgent->ClearOutputLinkChanges();
                    pAgent->Commit();

                    /* Run self for 5 decision cycles */
                    pAgent->RunSelf(5);

                    /* Cycle a few times to get the status complete and output-link clearing in order */
                    //pAgent->RunSelf(3);

          }

                              //rewardStatus = "received";
}

std::string NightstandBrain::GetRewardStatus(void)
{
          return rewardStatus;
}

void NightstandBrain::SetRewardStatus(std::string inc)
{
          rewardStatus = inc;
}
```

Appendix D: ComforTable Delivery and Movement Source Code

```
//Pressure regulator inputs
int outPin1 = 3;
int outPin2 = 6;
int outPin3 = 9;
int outPin4 = 10;

//LEDs
int red = 4;
int green = 5;
int blue = 8;

//IR sensors
int sense1 = 0;
int sense2 = 2;

//button  press
int Press = 2;
int feedback = 0;
//muscle pressures
int muscle1=98;
int muscle2=80;
int muscle3=0;
int muscle4=0;

float volt1=0,volt2=0, dist1=0,dist2=0;
int fl=0;
int pressval=0,feed=0;
float count1,count2,count3,count4;
int j;

void setup()
{
  Serial.begin(9600);
  analogWrite(outPin1,muscle1);
  analogWrite(outPin2,muscle2);
  analogWrite(outPin3,muscle3);
  analogWrite(outPin4,muscle4);
  delay(15000);

  count1=count2=count3=count4=0.0;
  pinMode(red,OUTPUT);
  pinMode(green,OUTPUT);
  pinMode(blue,OUTPUT);
  pinMode(Press,INPUT);
  pinMode(feedback,INPUT);
  digitalWrite(green,HIGH);

}
```

```
void loop()
{
  int i;


  // first event monitoring
  pressval=digitalRead(Press);
  if(pressval==LOW)
  {
  int senseflag=0;

  //waiting for the user to pick the glasses
  for(i=0;i<5000;i++)
  {
     volt1 = analogRead(sense1)*0.0048828125;
  dist1 = 65*pow(volt1,-1.1);
  if(dist1<40)
  {
    senseflag=1;
    break;
  }
  }
  //curving movement
  if(senseflag!=1)
  {

        digitalWrite(green,LOW);
    digitalWrite(red,HIGH);


    for(j=1;j<=10;j++)
    {
      count1=count1-9.8;
      count2=count2+4;
      count4=count4+12;

    analogWrite(outPin1,muscle1+(int)count1);
  analogWrite(outPin2,muscle2+(int)count2);
  analogWrite(outPin3,muscle3);
  analogWrite(outPin4,muscle4+(int)count4);

   delay(500);
    }

  //waiting for the user to pick the glasses
  while(senseflag!=1)
  {
     volt1 = analogRead(sense1)*0.0048828125;
  dist1 = 65*pow(volt1,-1.1);
  if(dist1<40)
{
    senseflag=1;

}
}



for(j=1;j<=10;j++)
  {
    count1=count1+9.8;
    count2=count2-4;
    count4=count4-12;
     analogWrite(outPin1,muscle1+(int)count1);
analogWrite(outPin2,muscle2+(int)count2);
analogWrite(outPin3,muscle3);
analogWrite(outPin4,muscle4+(int)count4);
delay(500);

}
digitalWrite(red,LOW);
delay(2000);
digitalWrite(green,HIGH);
delay(2000);
}


  count1=count2=count3=count4=0.0;
}

//sensing for upward movement
volt2 = analogRead(sense2)*0.0048828125;
dist2 = 65*pow(volt2,-1.1);
if(dist2<90)
{


  digitalWrite(green,LOW);
  digitalWrite(red,HIGH);
  for(j=1;j<=10;j++)
  {
    count1+=6.2;
    count2+=7;
      analogWrite(outPin1,muscle1+(int)count1);
analogWrite(outPin2,muscle2+(int)count2);
analogWrite(outPin3,muscle3);
analogWrite(outPin4,muscle4);

 delay(50);
  }
  delay(10000);
  for(j=1;j<=10;j++)
  {
    count1-=6.2;
    count2-=7;
      analogWrite(outPin1,muscle1+(int)count1);
analogWrite(outPin2,muscle2+(int)count2);
analogWrite(outPin3,muscle3);
analogWrite(outPin4,muscle4);
 delay(1000);
   }
  fl=0;

  digitalWrite(red,LOW);
digitalWrite(green,HIGH);
}

count1=count2=count3=count4=0.0;


//reward and response
feed=digitalRead(feedback);
if(feed==LOW)
{
  Serial.write(255);  // communicating the reward to central intelligence

  digitalWrite(green,LOW);
  digitalWrite(blue,HIGH);
  delay(1000);
  digitalWrite(blue,LOW);
  delay(1000);
  digitalWrite(blue,HIGH);
  delay(1000);
  digitalWrite(blue,LOW);
  delay(1000);
  digitalWrite(green,HIGH);


}


}
```

Appendix E: Nightstand Drawer System Source Code

```
#include <Servo.h>

int servoPin1 = 7;
int servoPin2 = 6;
Servo servoPin3;
int buttonPin1 = 2;
int buttonPin2 = 3;
int buttonPin3 = 4;
char startSignal;
int numBytes=0;

void setup()
{
  pinMode(servoPin1,OUTPUT);
  pinMode(servoPin2,OUTPUT);
  pinMode(buttonPin1,INPUT);
  pinMode(buttonPin2,INPUT);
  pinMode(buttonPin3,INPUT);
  servoPin3.attach(10);
  servoPin3.write(10);
  Serial.begin(9600);
  Serial.flush();
  pinMode(13, OUTPUT);
}

void blink()
{
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}

int state1=0;
int state2=0;
int state3=0;
int temp;

void Move();

void loop()
{
  state1=digitalRead(buttonPin1);
  state2=digitalRead(buttonPin2);
  state3=digitalRead(buttonPin3);
  numBytes = Serial.available();
  if(numBytes>0)
  {
    //received signal from AI
    startSignal = Serial.read();
    if(int(startSignal)>-1)
    {
      Move();
    }
  }

  if(state1==HIGH)
  {
    for (temp = 0; temp <= 4; temp++)
    {
      digitalWrite(servoPin1,HIGH);
      delayMicroseconds(2000); // 1.8ms
      digitalWrite(servoPin1,LOW);
      digitalWrite(servoPin2,HIGH);
      delayMicroseconds(1000); // 1.8ms
      digitalWrite(servoPin2,LOW);
      delay(20);
    }
  }
  else if(state2==HIGH)
  {
    for (temp = 0; temp <= 4; temp++)
    {
      digitalWrite(servoPin1,HIGH);
      delayMicroseconds(1000); // 1.2ms
      digitalWrite(servoPin1,LOW);
      digitalWrite(servoPin2,HIGH);
      delayMicroseconds(2000); // 1.2ms
      digitalWrite(servoPin2,LOW);
      delay(20);
    }
  }
  else
  {
    digitalWrite(servoPin1,HIGH);
    digitalWrite(servoPin2,HIGH);
    delayMicroseconds(1500); // 1.5ms
    digitalWrite(servoPin1,LOW);
    digitalWrite(servoPin2,LOW);
    delay(20); // 20ms
  }

  if(state3==HIGH)
  {
    Move();
  }
}
```

```
void Move()
{
      //move drawer forward
      for (temp = 0; temp <= 30; temp++)
      {
        digitalWrite(servoPin1,HIGH);
        delayMicroseconds(2000); // 1.8ms
        digitalWrite(servoPin1,LOW);
        digitalWrite(servoPin2,HIGH);
        delayMicroseconds(1000); // 1.8ms
        digitalWrite(servoPin2,LOW);
        delay(20);
      }

      delay(1000);

      //kick compartment out
      for(temp = 10; temp < 110; temp += 2)
      {
        servoPin3.write(temp);
        delay(15);
      }
      delay(500);
      for(temp = 110; temp >= 10; temp -= 2)
      {
        servoPin3.write(temp);
        delay(15);
      }

      delay(1000);

      //move drawer backward
      for (temp = 0; temp <= 30; temp++)
      {
        digitalWrite(servoPin1,HIGH);
        delayMicroseconds(1000); // 1.8ms
        digitalWrite(servoPin1,LOW);
        digitalWrite(servoPin2,HIGH);
        delayMicroseconds(2000); // 1.8ms
        digitalWrite(servoPin2,LOW);
        delay(20);
      }
}
```