MITTS (Motion Interface Thermal Touch Sensitive)

Cristopher Britt, David Simoneau, Anthony Hinnant, Francisco Tirado Perez

Dept. of Electrical Engineering and Computer
Science, University of Central Florida,
Orlando, Florida, 32816-2450

Abstract — The MITTS system is a virtual reality interface glove that allows not only interaction with a virtual environment, but also allows for haptic and thermal feedback from that environment. It was designed and tested in four discrete sections: The hardware, the embedded software, the configuration utility, and the virtual environment. Each of these systems work together to create a superior user experience that can pave the way for a more seamless virtual experience.

Index Terms — Virtual reality, haptic feedback, consumer device, thermal response, Peltier device.

I. Introduction

This product is a glove controller that has the capabilities of interacting with a virtual 3D object and receiving both haptic and thermal feedback. The design is focused on taking accelerometer inputs from the glove, transforming

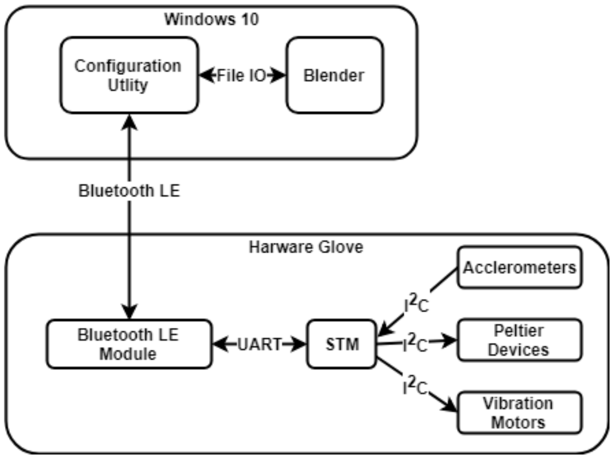


Fig. 1. System Block Diagram

them into positional data, transmitting data through a processor, then sending that data through our interface to the computer running the simulation, and finally receiving haptic feedback from that simulation. The project is designed in such a way to allow us to gain experience in PCB design as well as data communication, ARM development, and 3D modeling software API.

II. Hardware Design and Implementation

The driving philosophy behind the design of this hardware was that complex problems should be broken down into simple subsystems. As such this discussion will be focused on each of those subsystems, the part selection, and how that subsystem was designed to be as simple as possible. The overall hardware was specified to not exceed 2 kilograms and support a frame rate of at least 30 frames of data per second.

A. Accelerometer

The MMA8451Q accelerometer was selected for this application because of it is relatively inexpensive ($3.34), is I2C capable, has 14 bits of resolution, and a QFN package size. Its max sampling rate of 800 Hz is also sufficient for this application. Unfortunately, the MMA8451 only has two I2C addresses, but to address this an I2C multiplexer was also implemented. The TCA9548A has 8 possible I2C lines that are controlled via I2C. Not only does this allow for all 16 accelerometers to be addressed, but it also minimized line capacitances. Each accelerometer was implemented on its own separate board which was designed with its overall position in the glove in mind. The central board on the back of the hand required all 8 I2C lines running through it so it is larger by necessity. The boards on the first knuckle require 2 I2C lines running through them, while the boards on the second and third knuckles of each finger will only ever require the same I2C line. This is as well as the need for two selectable addresses is reflected in the designs. The two selectable addresses are implemented via a solder jumper so the team member writing the code may select addresses as needed. The design was successfully tested by both posting an I2C address to a Raspberry Pi and by sending data to an STM breakout board.

B. LED Signals

This design implements a 3-color common anode RGB LED to demonstrate states and signals. The red led activates as soon as the power goes on while the green and blue LEDs are controlled by BSS138 NFETs linked to GPIO pins. The common anode design allows for the

LED to be controlled by a single NFET since source is linked directly to ground. This design was tested on a breadboard before implementation.

C. Signal Switching

There are many instances where a GPIO pin needs to toggle a FET where the gate-source voltage would not be stable in an open configuration. To address this a signal switch was designed using a dual N and P FET. For this purpose, the NX3008CBKS was selected since it has robust gate voltages, sufficient current specs, and the hardware designer was familiar with its use. This switch uses the PFET as the main switch with the gate held high with a pullup resistor. The GPIO toggles an NFET which pulls the gate of the PFET to ground. This opens the PFET channel fully and acts as a reliable two state switch.

D. Motor Controller

The haptic feedback of the glove is implemented with several coin type eccentric rotating mass motors. It was determined that the feeling of vibration when tightly bound to the finger tips simulates the feeling of pressure. Increasing the intensity and frequency of this vibration can simulate increasing pressure. The motor selected was the C0720B015F, which was selected for its small size, its max current consumption of 80 mA, and its maximum voltage of 3.3 volts. The motor is controlled by a previously discussed signal switch, a digital potentiometer-controlled voltage divider, and a non-inverting operation amplifier. The signal switch opens the 3.3 rail to the voltage divider. The voltage divider consists of a 100 kΩ resistor and a 100 kΩ digital potentiometer (AD524). The digital potentiometer was selected for its small size and I2C capability. The resistance of the digital potentiometer is set via I2C. At maximum resistance the 3.3 rail voltage is divided by a factor of 2 so this signal is fed into a non-inverting amplifier with a gain of 2 based on a general-purpose operation amplified, the TSV321. This amplifier was selected because of its inexpensive nature and its current output of 80 mA. This ampler ensures that the voltage across the motor is within operational bounds and provides enough current to operate the motor.

E. Power Systems

The glove design utilizes a single 3.3 rail to power all devices and subsystems. As a result, the regulator was selected to be able to output a large amount current. Specifically, the calculated current requirements called for a maximum current of approximately 3 amps. Originally a high efficiency switching power regulator was selected, the TPS62823. It was selected based on low cost, high efficiency, and small footprint. Unfortunately, the design did not work in testing and was immediately scrapped for a simpler design to meet a team internal deadline on hardware delivery. The simplified design implemented a large linear regulator that, while less efficient, could still meet the required current demands. Specifically, the TPS75701KTTT was selected. It has a 150-mV dropout at 3 amps, but this level of dropout will not significantly impact the design. Two ICR18650 lithium ion batteries were selected to power the glove. These batteries both hold 6.6-amp hours at full charge which is sufficient to power the glove for longer than the required 30 minutes of use. Unfortunately, the maximum discharge rate of both batteries is only 1.25 amps. This means that the max current the batteries can provide is only 2.5 amps, which limits the number of Peltier devices that can be implemented on battery power. The maximum charging rate on each batter is 1.65 amps. The CH-L373 lithium ion charger was selected and purchased since it has a charging limit of 3 amps. This allows relatively quick charging. The power design also incorporated an auxiliary 3.3 port and a main rail jumper. When the jumper is removed the glove can be powered from an external power supply without damaging the regulator or the batteries.

F. Peltier Device

The glove design originally called for full thermal feedback using Peltier devices. This meant the user would be able to experience both heat and cold variations based on the current polarity and intensity. To this end the CM23-1.9 was selected. It is a small Peltier device that could fit on the second knuckle of the hand with sufficient temperature characteristics. Unfortunately, based on how the human hand experience heat and cold this design would not deliver the required experience. It was found during testing that it was not possible to create a cooling effect that was noticeable without using extreme amounts of current. This created unsafe temperatures on the potion of the Peltier facing away from the hand and it was decided that having a glove that created the equivalent of hot coals on the hand while the user was immersed in virtual reality was unsafe. The Peltier also did not perform as expected when it came to heating. The range that the Peltier device could heat the hand where it was noticeable and not painful to the user was exceedingly small and changes in it were not readily noticeable. It was decided that trying to control temperature variations would not enhance the experience and in certain cases be a burn risk for the user. The Peltier's were then implemented with an on/off architecture where when they are activated by GPIO the Peltiers generate a preset temperature. Due to the legacy hardware from the pretesting the DRV8836 H-Bridge is used as a current source for the Peltiers. It was originally implemented to allow for temperature control of the Peltier but was rendered redundant and now serves as switched current supply.

III. Embedded Programming

The MITTS system depends on a system of inputs and outputs. Each accelerometer needs to be communicating with the STM32 as inputs and each Peltier device as well as the vibrational motors needs to be communicating with the STM32 as outputs. The system depends on the I2C and UART protocols for the means of communication. While programming the MITTS board, it's important to keep in mind the sequence of events that needs to occur and how each event influences the other. The the next couple of section informs the reader of how the board is programmed, how to debug the system, and how challenges affected the whole process.

A. Programming the chip

Before the PCB was developed, coding the STM32 needed to be started, thus the chip was programmed by using the development board for a similar STM32. In this way, the team pipelined the process of developing the system and each individual could asynchronously work on different, major parts. With the development board, STM32 supplies a wide variety of tools to aid with programming the chip. One such tool is the STM32CubeMX. This tool essentially generates the code for the configurations of the chip. These configurations can include anything from clock configurations, peripheral configurations, and even power configurations. Through the user interface provided, the peripherals were set for each pin as seen in Figure 2:

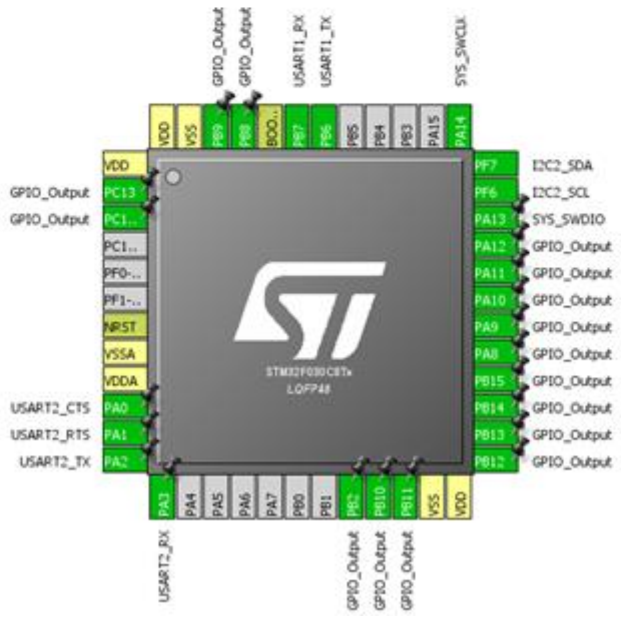
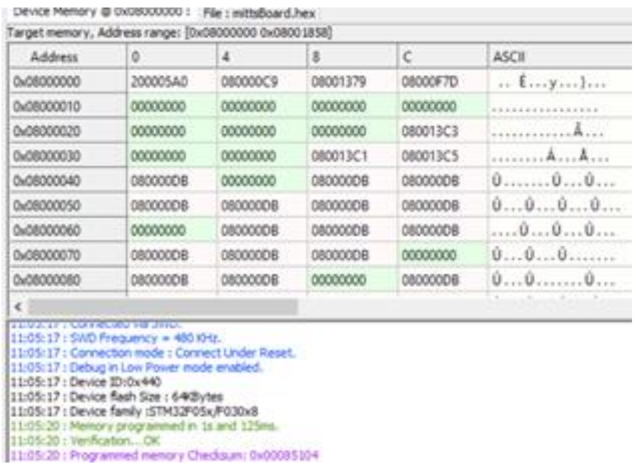


Fig. 2. Pinout of STM32 for MITTS Board

Now that the peripherals have been generated and the configuration has been set up, the STM32 ST-LINK Utility can be employed. This tool is the method used to program the chip. Essentially, the tool takes a hex file, loads up memory information from the chipset chosen, and then allows for the hex file to be uploaded into the chip's memory. After the memory has been flashed, the tool can be used to verify the bytes are correct and the tool has been correctly programmed. It is also possible to use physical debugging, where the on-board LED is flashed to see if the program was loaded; however, even simple code such as this functionality takes debugging and the problems that arise from a self-created PCB can be many.

Once the chip has been flashed the code can be verified. Figure 3 shows that the device memory, which can only be seen if the device is recognized, and the hex file alongside for a visual check that the bytes are the same as well as a console log of the checksum being produced, and that the verification of the code was "OK."


Device Memory @ 0x08000000 : File : mittsBoard.hex

Target memory, Address range: [0x08000000 0x08001858]

Address	0	4	8	C	ASCII
0x08000000	200005A0	080000C9	08001379	08000F7D	.. É...y...}...
0x08000010	00000000	00000000	00000000	00000000	....
0x08000020	00000000	00000000	00000000	080013C3	............Ã...
0x08000030	00000000	00000000	080013C1	080013C5	........Á...Å...
0x08000040	080000DB	00000000	080000DB	080000DB	Û.......Û...Û...
0x08000050	080000DB	080000DB	080000DB	080000DB	Û...Û...Û...Û...
0x08000060	00000000	080000DB	080000DB	080000DB	....Û...Û...Û...
0x08000070	080000DB	080000DB	080000DB	00000000	Û...Û...Û.......
0x08000080	080000DB	080000DB	00000000	080000DB	Û...Û.......Û...

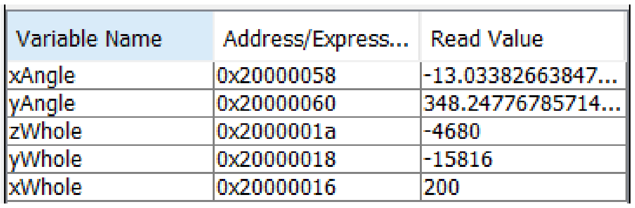
```
11:05:17 : SWD Frequency = 480 KHz.
11:05:17 : Connection mode : Connect Under Reset.
11:05:17 : Debug in Low Power mode enabled.
11:05:17 : Device ID:0x440
11:05:17 : Device flash Size : 64KBytes
11:05:17 : Device family :STM32F05x/F030x8
11:05:20 : Memory programmed in 1s and 125ms.
11:05:20 : Verification...OK
11:05:20 : Programmed memory Checksum: 0x00085104
```

Fig. 3. Program verification as seen by the STM32 ST-LINK software.

B. Debugging the System (software)

Debugging the software is an important step to verifying that the code is working. Through the STM32 tools, the project employed a tool called STM Studio. With the STM studio, monitoring variables and their respective values as they change in real-time becomes easier. This is essential at least for getting accelerometer data and transforming it to positional data. We will talk about this further in detail in a later section.

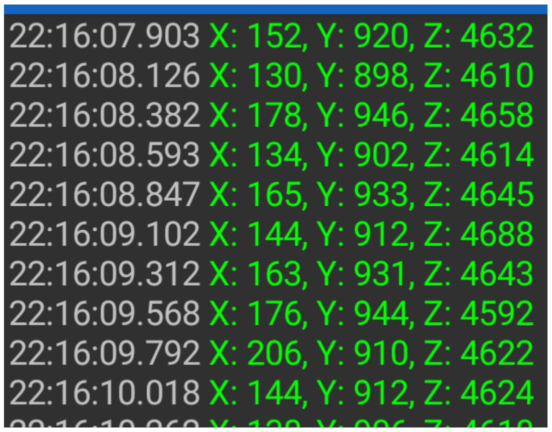
As we can see in Fig. 4, STM provides a graph interface to see the variables change as time elapses. The points are collected about every 100 ms and change in value from -16,384 to 16,384.



Variable Name	Address/Express...	Read Value
xAngle	0x20000058	-13.03382663847...
yAngle	0x20000060	348.24776785714...
zWhole	0x2000001a	-4680
yWhole	0x20000018	-15816
xWhole	0x20000016	200

Fig. 4. Readout of STM Studio of x, y, and z calibrated values with calculated x and y angles.

This is a good way to debug the I2C and any variables being changed, but the other side of the equation is the Bluetooth UART communication. Now, STM Studio does verify that the program runs on the MITTS board, but other issues need debugging. For Bluetooth communication, it's a lot of trial and error. Bluetooth adapters certainly help, but what really helps is going through logical thinking when trying to debug the reasons why data may not be sending via Bluetooth. First and foremost, it was important to make use of a Bluetooth terminal to scan for devices being used. In this first step, through the terminal, the "Laird BL652," which is the Bluetooth module used by this project, will become visible. Before connecting, it's imperative to set the correct service, read, and write characteristic UUIDs to ensure the correct read and write properties are configured. Without these, reading and/or writing from the Bluetooth module is impossible. Once the settings are correct, connect and verify the results are correct. The results are driven by the STM32 code, which have been programmed before-hand. If everything works out, it starts to look like Figure 5.



```
22:16:07.903 X: 152, Y: 920, Z: 4632
22:16:08.126 X: 130, Y: 898, Z: 4610
22:16:08.382 X: 178, Y: 946, Z: 4658
22:16:08.593 X: 134, Y: 902, Z: 4614
22:16:08.847 X: 165, Y: 933, Z: 4645
22:16:09.102 X: 144, Y: 912, Z: 4688
22:16:09.312 X: 163, Y: 931, Z: 4643
22:16:09.568 X: 176, Y: 944, Z: 4592
22:16:09.792 X: 206, Y: 910, Z: 4622
22:16:10.018 X: 144, Y: 912, Z: 4624
```

Fig. 5 Serial Bluetooth Terminal showing the X, Y, and Z values being sent via USART from the development board.

C. Challenges (software)

The challenges faced during the software part of the project include getting the I2C to work, getting the Bluetooth to send data and receive data, and verifying the code is being programmed onto the MITTS board.

The first challenge of getting the I2C to work was to scour through the data sheets to get the addresses for the accelerometers and to figure out the standard to coding these using the HAL library, which is used by the STM32 to program the I2C and UART protocols, for example. This was conquered by a lot of verification from datasheets to the datasheets created for the project of how the PCB was created. The pins need to first be verified that they were configured to the right functionality and that the pins themselves were soldered correctly.

The final challenge is getting the data from the accelerometers to positional data. The challenge faced was that with each differentiation of the accelerometer data, the results get less and less accurate, since the accelerometers have a plus/minus 5% accuracy. With this problem, it seemed to be too much of a challenge so another way of getting position data, which was easy to do with the current design, is to calculate the angles by the values received from the accelerometers.

IV. Configuration Utility

The Configuration Utility was designed to satisfy three primary goals of the completed system; isolation, scalability, and maintenance.

The utility provides isolation by operating between the two endpoint subsystems; the virtual environment software, and the hardware glove. This allowed for all three subsystems to be developed in parallel. Only the interface specifications between the utility and each endpoint subsystem was coordinated during development and testing. The isolation occurs at the two natural interface boundaries of the system; the Bluetooth connection from the hardware glove to the host operating system, and the data transmission between the utility and the virtual environment.

The utility provides scalability by supporting and coordinating the connection of multiple hardware gloves. Each glove is assigned a profile that tracks the parameters necessary to establish a Bluetooth connection to the glove, and a connection to its virtual counterpart in the virtual environment. Profiles can be created, edited, and stored for later use.

Maintenance is realized in the form of a debugging interface. The interface allows for a direct connection to a hardware glove. This connection is then used to inject feedback values directly to the gloves electronic feedback devices and receive positional data from the glove's motion tracking system. The debugging interface provides a real-time display for visualizing the data and is utilized to identify failures in specific electronic devices on the glove itself.

The Configuration Utility was developed for the Microsoft Windows 10 Operating System. Windows 10 was chosen due to its widespread use, the availability of development tools, and because the utility is deployed on the same machine that virtual environment is on which utilizes Windows 10.

The utility was developed in Microsoft Visual Studio Community Edition 2015 using the Visual C# programing language. Visual Studio was chosen for its debugging capabilities, and the stability and documentation of the 2015 release. C# was chosen for being object-oriented, fully supported by Visual Studio, direct compatibility with the Windows Bluetooth APIs, and the ability to utilize WinForms and Windows Presentation Foundation for future Graphical User Interface development.

A. Use Case Diagram

The use case diagram for the configuration utility is shown in the figure below. It describes the functionality that the utility exposes to the external system components and users. There are three actors that interact with the utility. They are the user, the virtual environment, and the hardware glove.

The diagram shows that from the glove and the environment's perspective, the utility exists only as a gateway for transmitting and receiving data between them. This provides the isolation effect of the system. In addition, all interaction between the user and the hardware glove and virtual environment, always passes through the utility.

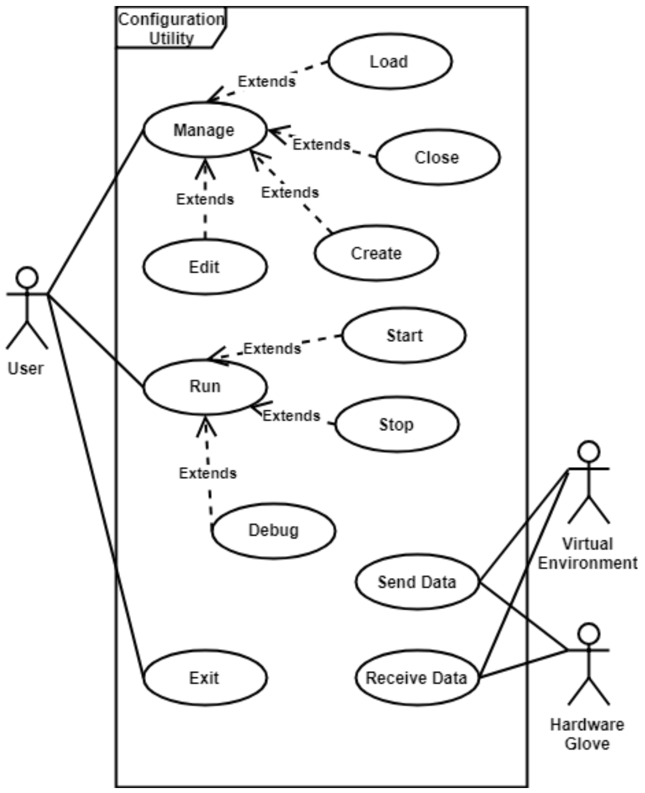


Fig. 6. Use Case Diagram.

The primacy actor that interacts with the configuration utility is the user whom uses the utility to perform three major tasks, each of which are supported by minor tasks. The major tasks for the utility are managing profiles, running gloves, and closing the utility.

The Managing Profile use case, is supported by the actions of creating, editing, loading, and closing deleting profiles. These actions are how the user manages the hardware glove profiles of the system. Profiles are maintained in a profiles directory within the utility directory; and are stored in plaintext. When loaded, the profiles preserve settings, so the user does not have to go through the process of configuring the Bluetooth connection for the same glove every time the utility is started.

The action of running a glove, is how gloves that have been loaded are subsequently started, stopped, and debugged. Starting a glove is the action of initiating normal operation. A started glove provides updated positional data to the virtual environment and receives updated feedback values from the virtual environment. Stopping a glove simply terminates normal operation and returns the glove and virtual environment to a waiting state. The debugging action immediately initiates a debugging interface for a selected glove. Any other currently connect and loaded gloves continue in their current operating state. Only single glove can be debugged at one time.

Exiting the utility terminates the operation of all connected gloves. For normal operation to occur, the utility must be active, the glove connected, a profile for the glove loaded, and the glove started in the normal operation mode.

B. Class Diagram

The class diagram for the configuration utility describes the logical organization and structure of the software as developed following the object-oriented programing paradigm.

The UtilCore is entry point of the software. The role of the UtilCore is to instantiate a Storage, Menu, and GloveRack object. The software is then initiated and controlled passed to the menu object for interaction with the user. Finally, the UtilCore waits for the exit command to be given, so that the software can be successfully terminated.

The Storage class process all manipulation of the file system for the loading and saving of profiles. The software uses profiles stored in plain text with a file extension of the form ".mitts" to facilitate ease of identify profiles in the working directory. The storage object provides methods for listing all profiles in the working directory, reading for and returning the contents of a profile, and creation and overwriting of files when saving new profiles or edits to existing ones.

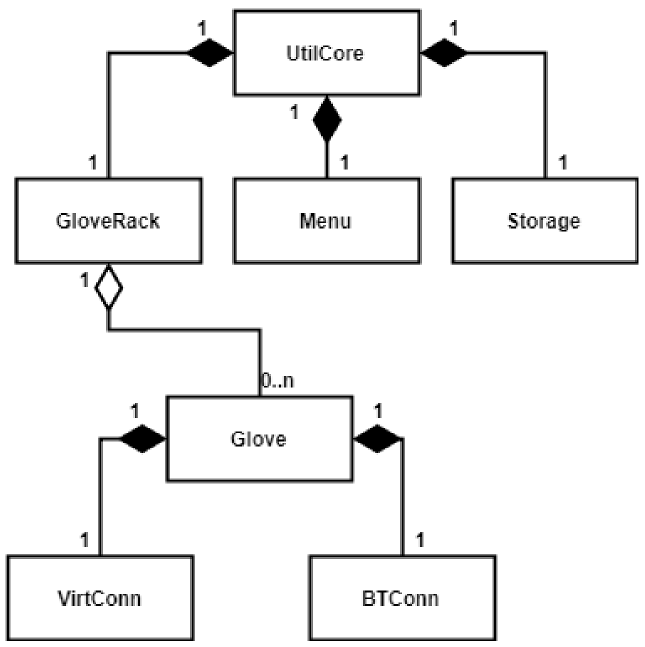


Fig. 7. Class Diagram.

The Menu class provides the user interface for the software. The interface consist of ASCII painted menu screens navigated by numerical indexes. The menu class is responsible for both painting the display, and reading in and processing the user's input. In total, there are fourteen menu screens containing both static and dynamic progression choices.

The GloveRack is the storage device for loaded gloves. A list of currently load gloves is maintained with methods provided for their manipulation. Methods include finding of gloves by name or list index, adding or removing gloves from the List, and activation of an individual glove in order to change its operating state.

The Glove class is the unit that represents a single connect glove. Within this class are methods for retrieving a glove's parameters, along with starting, stopping, and debugging the glove. This class is composed of two more classes; VirtConn handles the gloves connection to the virtual environment, while BTConn handles the glove's connection to the hardware glove itself.

VirConn coordinates the connection between the logical representation of the glove in the utility, and its virtual counterpart in the virtual environment. The connection is accomplished using low level filesystem commands. When the logical glove passes data tot eh virtual environment, a file containing the data is created in the working directory. When the virtual environment response with data, it also creates a file that the logical glove can open and read the data from. This approach was used do to its simplicity and ease of implementation.

The BTConn handles the Bluetooth connection between the logical glove and its physical representation. BTConn utilizes the Bluetooth low energy methods provided by the windows runtime library to facilitate the transfer of data. Connections are established using parameters loaded from a profile. Communication is performed synchronously in a receive then respond communication scheme. This ensures that when updated positional data is received from the physical glove and passed to the virtual environment, the feedback data that is returned from the virtual environment to the physical glove is synced and represents the effect of the previous user action.

V. Blender and Virtual Environment

A fundamental part of our project was creating an interactive 3D Environment to show the capabilities of our glove. Blender is a professional, open-source 3D graphics toolset. It's capable of animation, effects, printing 3D models, and most importantly for us, interactive 3D applications. Blender is used to demonstrate the abilities of our glove.

Blender serves its purpose in multiple steps, to model out assets we need to make, import other free assets that we need, rig the models to be animated, implement the physics system, and interface it to interact with the configuration utility.

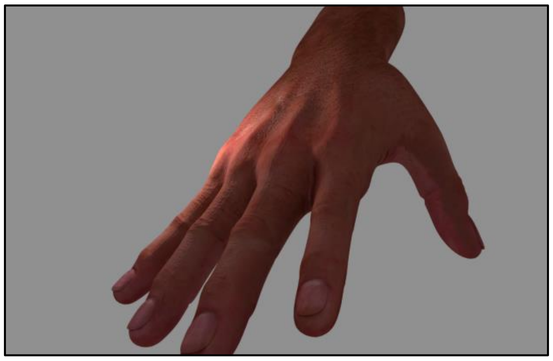


Fig. 8. Hand Model

A demonstration table with different objects was used to show the capabilities of our device, including motion tracking, the haptic feedback, and the Peltier devices heating up. The first of which is a sphere, which is added for two reasons. The first one is to be able to demonstrate the haptic feedback of the glove more thoroughly. The glove is designed to give haptic feedback at multiple "intensities," so the sphere can simulate a "soft" material while the block can simulate a "hard" material. The glove's haptic feedback will vibrate more intensely when meeting the block than coming in contact with the sphere. The second reason is to demonstrate the positional aspect of the haptic feedback. A sphere will be in contact at different points in your hand than a block when holding it. These different points will vibrate to more simulate the "feeling" part of the project in a more immersive way.

The other two objects added are thin cylinders on either side of the desk, one colored blue and one colored red. This is a simple way to make "stove tops" that can be used to test the Peltier portion of the glove's functionality. The cylinders are given a field of range, extending upward but not out to the side. When the hand enters these fields, the Peltier will heat or cool according to the color cylinder the hand is hovering above.

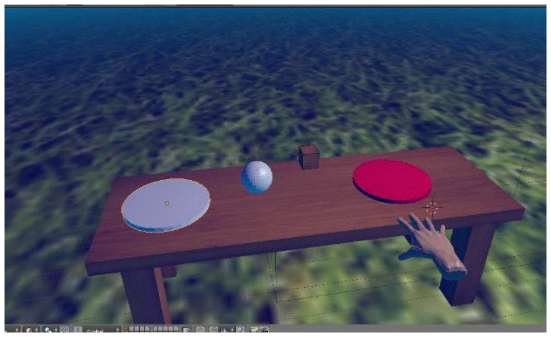


Fig. 9. Demonstration Table

The hand is capable of changing position, angle, and finger positions independently of one another. It implements file i/o with the configuration utility to get information on how it should change. This is all implemented with python scripts that are programmed through blender. When the glove change position in real life, the configuration utility writes to the file that blender reads, and when the hand comes in contact with something, it writes to a file that the configuration utility reads to activate the haptic feedback. The Peltier devices work in a very similarly.

VI. Virtual Reality

The device is used in conjunction with the HTC Vive. The glove doesn't necessitate use of any virtual reality headset, nor does it require use of virtual reality to function. That being said, the idea for this project was made to be implemented with virtual reality, as it can greatly enhance the current virtual reality experience. Because of this, we want to demonstrate our device's functionality in conjunction with virtual reality.

The Vive itself requires a little bit of setup to use. Its biggest feature is its room tracking technology, which lets you walk around a room in virtual reality. This requires two lighthouses on each corner of the room, which should be station 6 feet 6 inches off the floor, in a 6 feet 6 inch by 5 feet clear area. It also requires a wired connection to the PC because it needs a direct connection to the video card, as well as a USB connection and a power connection. The PC will run the software, and the glove will be connected via Bluetooth. The lighthouses require a power connection through the wall and communicate wirelessly with the headset. The user can explore a 3D space in a 5 feet by 6.5 feet area and interact with the environment using the glove.

VII. Conclusion

Interacting with a virtual environment was both an academically interesting and technically challenging task that could produce dividends for the gaming and hobbyist communities. The addition of both haptic and temperature feedback could greatly expand the immersions of virtual worlds. By adding this design to the open source community, we expect to advance our careers and highlight our technical skills to prospective employers.

From PCB design to programming interfaces to working with Blender, this design has brought a unique experience to the group. PCB design is complicated and comes with a lot of trial and error. Creating an interface is a very valid real-world experience that will benefit us when alike tasks are assigned to us. The STM32 main PCB design has posed a risk of bottle-necking the project because of it being the main source for data retrieval and data transmission.

Overall, gathering this information has been very beneficial because of the knowledge of the potential risks and the comparisons between the parts that were a potential for the design.

Acknowledgements

We would like to acknowledge a both Dr. Samuel Richie and Dr. Lei Wei as the Senior Design I & II coordinators.

References

[1] I2C Info – I2C Bus, Interface and Protocol. (2018). I2C Bus Specification. [online] Available at: http://i2c.info/i2c-bus-specification [Accessed 29 Jun. 2018].
[2] Neamen, "Microelectronics: Circuit Analysis and Design", McGraw Hill, 4th Edition
[3] Sharp, John. Microsoft Visual C♯ Step by Step. 8th ed., Microsoft Press, 2015.
[4] Johnson, Bruce. Professional Visual Studio 2017. Wrox, 2018.
[5] Townsend, Kevin, et al. Getting Started with Bluetooth Low Energy:1st ed., OReilly, 2014.

Biographies

Chrisopher B.T. Britt, is a second degree seeking senior electrical engineering major at the University of Central Florida. He works in a cube sat development lab as an integration and test engineer. He hopes to find employment with a major aerospace firm after graduation.

David J. Simoneau is a computer engineering major at the University of Central Florida. He is currently employed by the Disney company as a professional software engineering intern.

Anthony Hunter Hinnant is a computer engineering major at the University of Central Florida. He is currently employed by AVT Simulation as a Maintenance Software Engineer Intern.

Francisco Tirado Perez is a second degree seeking senior computer engineering major at the University of Central Florida. His career includes work, experience program participation with Lockheed Martin Rotary and Mission Systems, employment as an engineer with Constant Velocity of Ocala Inc., and service as an Infantryman in the United States Army. Francisco has also received a Bachelor of Science in aerospace engineering from UCF.