Northwestern University

McCormick School of Engineering
Department of Electrical Engineering and Computer Science

ECE 347 – Microprocessor System Projects
Winter 2021

Computer Input Team
Maintenance Manual

Alex Kolar, alexanderkolar2021@u.northwestern.edu
Eugene Choe, eugenechoe2022@u.northwestern.edu
Jean Selep, JeanSelep2021@u.northwestern.edu
Jiayue Bai, JiayueBai2022@u.northwestern.edu

Client: Professor Henschen
Date: June 12th 2021
Version: ONE

Overview

The wireless, gloved computer input device comes in two physical parts, the glove and the base station. All of the relevant hardware information for each component can be found below in the Hardware section, along with how to assemble the glove and the base station. In the software section, we show how each physical component connects/communicates with each other in code. This section also includes how the inputs are processed along with how the GUI works. Additional limitations and constraints for the device and the components have also been included for your reference. Potential steps forward have also been listed below under the Future Development section.

Safety Information

Special precaution needs to be taken with the battery pack. Although in our testing there weren't any problems, the battery did run warm at times and has the potential to overheat. The battery is placed away from the hand in its own pouch as a safety precaution, but be sure to inspect the power supply regularly for any cable, plug, and casing damage. If any overheating occurs, please disconnect the power and wait before using it again or replace it with a new power supply that is within the limits of the voltage regulator.

The electronic components are also very close to the skin, so make sure the connections are correct before turning it on or testing. Short circuiting components can be dangerous because the components are in close proximity with the hand.

Hardware

Glove Components

Microcontroller

The Expressif ESP32 Development Board from Adafruit was used in this product. This 3.3V board contains pints for interfacing with I2C, SPI, and analog signals. Additionally, the board includes built-in bluetooth hardware for transmission and reception of bluetooth signals. The board is programmed through a microUSB port, and may be powered through this port as well (via an on-board voltage regulator). There is also a 3.3V pin on the board that may be used to supply power, but this is not regulated. The pinout diagram for the microcontroller is shown in Figure 1.

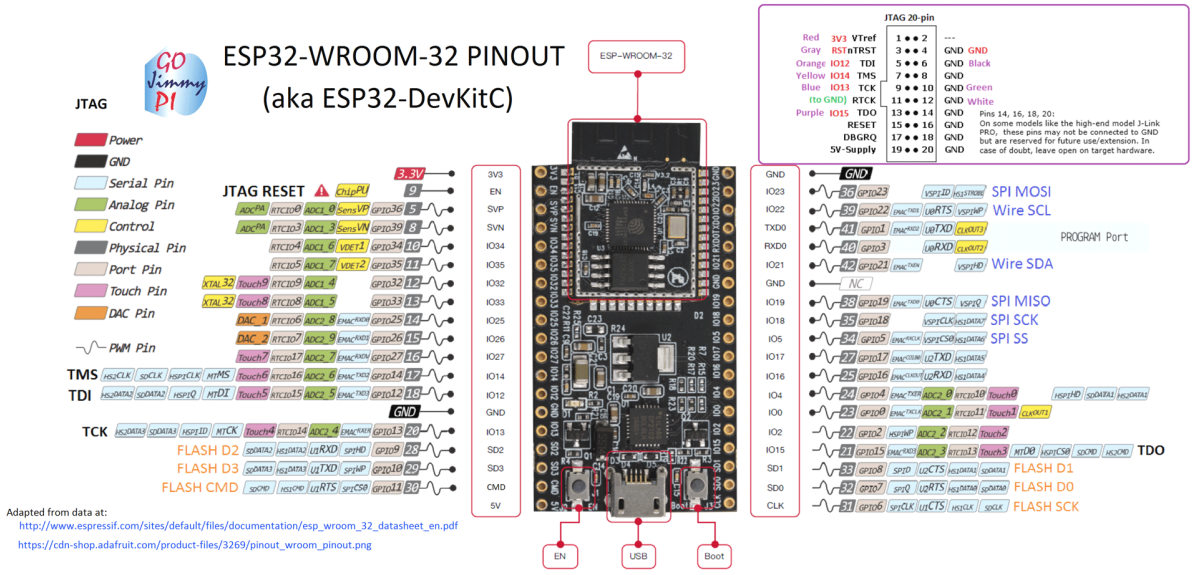


Figure 1: Pinout diagram for the esp32 dev board used.

Flex Sensors

The Spectra Symbol Flex Sensor was used in this product. The flex sensor acts as a variable resistor whose resistance changes depending on the angle of bend in the sensor. The flex sensor used in this product has a flat resistance (resistance when there is no bending) of 25kΩ ±30% and a max resistance of at least two times the flat resistance at the 180° pinch bend.

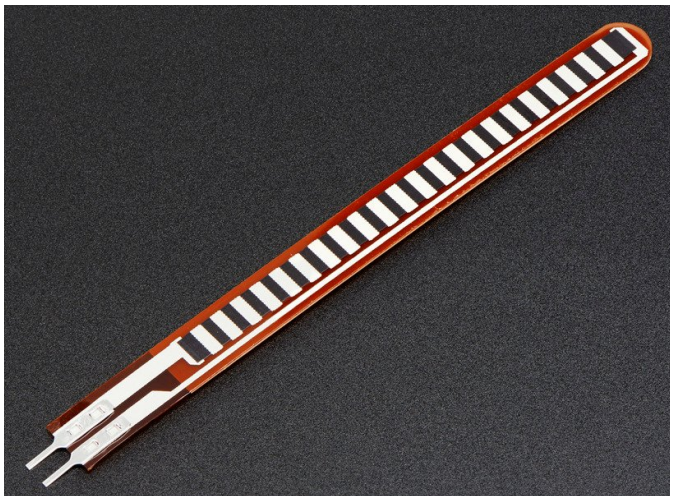


Figure 2: Flex sensor from Spectra Symbol

While using the flex sensor, ensure that the power supplied to the flex sensor is at least 0.5W and does not exceed 1W.

Infrared Camera

The IR camera selected is the IR Positioning Camera found on dfrobot.com. This camera interfaces with the microcontroller via 5V and 3.3V tolerant I2C.

Figure 3: IR camera.

Battery

The product uses a single-cell, 3.7V LiPo battery. The output voltage ranges from 4.2V when fully charged to 3.0V, at which point the safety circuitry built into the battery will cut off all output power. For details on the safety and usage concerns associated with the battery, please refer to the overview section.

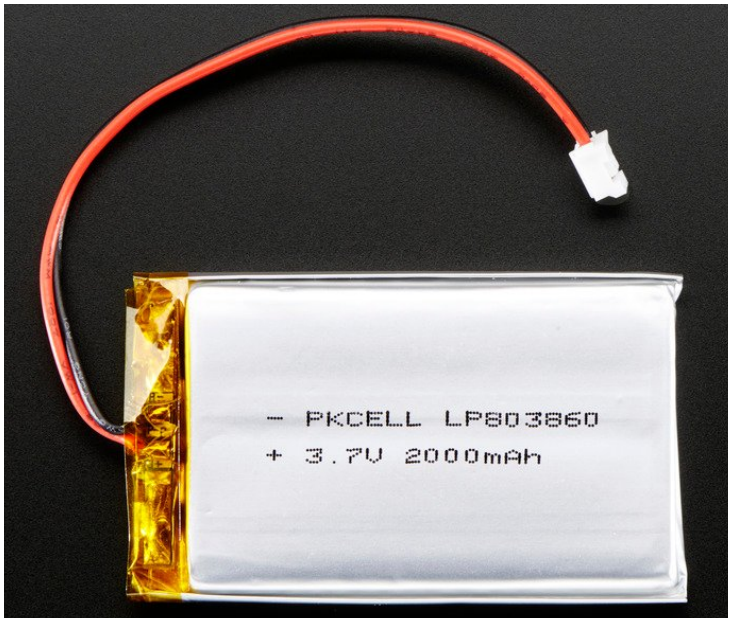


Figure 4: Lithium polymer battery used.

Voltage Regulator

For regulating voltage to glove components, the Pololu 3.3V Step-Up/Step-Down Voltage Regulator S7V8F3 is used in this product. This small chip can accept voltages ranging from 2.7V to 11V, and outputs the 3.3V required by the microcontroller.

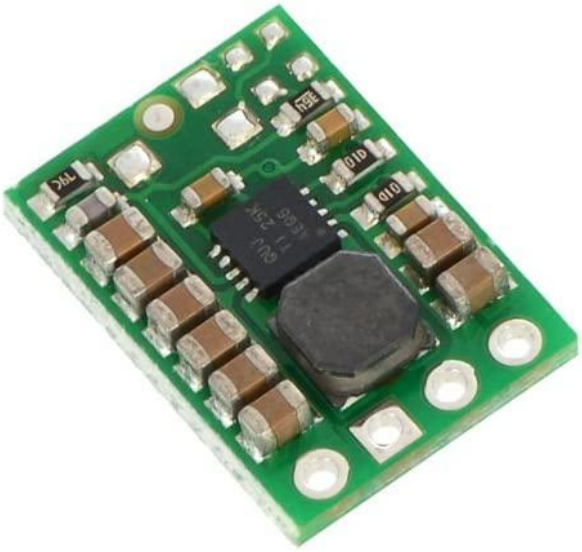


Figure 5: Voltage Regulator

Base Station Components

Wall Power Adapter

To power the base station components, an adafruit wall outlet adapter was used. This component plugs into a conventional wall outlet and provides power output via a Micro-USB cable at 5V and 2.5A.

Figure 6: The wall power adapter.

LiPo Battery Charger

For charging the battery, a pre-made Adafruit charger is used. This connects the battery's JST connector to the Micro-USB cable which will provide power from the wall power adapter. The charger comes equipped with an LED to indicate battery charging, along with an LED to indicate that the battery has finished charging.

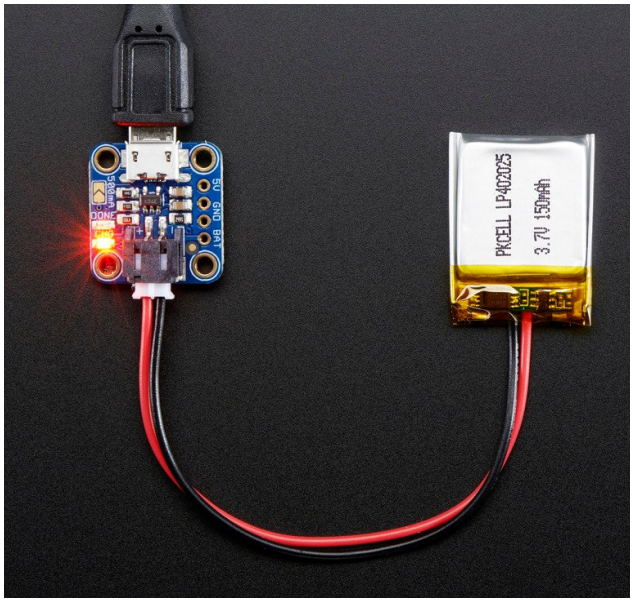


Figure 7: LiPo battery charger.

Breadboard Power Adapter

In this product, the [Micro-B breakout board] from adafruit was used. This board provides a 5V output from the Micro-USB cable to power components on the base station breadboard.

Assembly

Glove Component Assembly

The wiring diagram for all glove components is shown in Figure 8. A breadboard was used to connect wires for most components. For the flex sensors, and the buttons located on the fingertips, wires were soldered on to the component leads and connected to the breadboard. Heatshrink was also used to prevent damage to the solder joints. Finally, for the IR camera, individual wires were used to connect the female headers of the camera to those on the microcontroller.

In addition to the components discussed previously in this section, two pushbuttons were used with 10kΩ pull-down resistors, and two 25kΩ resistors were used in conjunction with the flex sensors to create a simple voltage divider (the output of which was read by the Microcontroller ADCs).

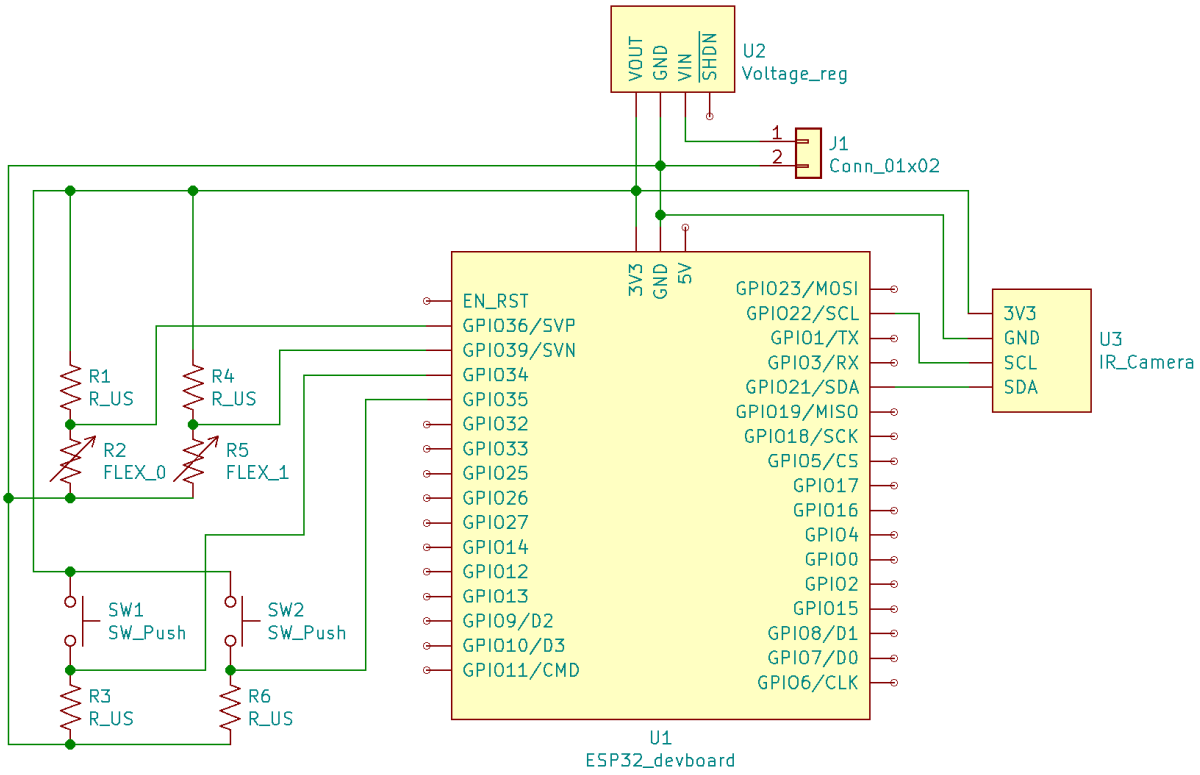


Figure 8: Wiring diagram for the glove. R1 and R4 should be 10kΩ resistors, while resistors R3 and R6 should be 20kΩ.

Base Station Component Assembly

The wiring diagram for the base station components is shown in Figure 9. The breadboard power adapter is used to supply power to the breadboard, where four 100Ω resistors were used with four IR LEDs. The LEDs should always be positioned in clusters of at least 2, such that there is ample light to trigger detection by the IR camera on the glove.

Not shown in the wiring diagram is the wall adapter and LiPo Charger. The MicroUSB end of the power supply is plugged into the breadboard power adapter when the glove is in use, and into the separate LiPo charger while the battery is being charged.

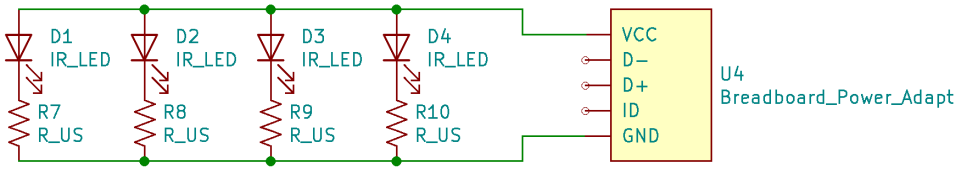


Figure 9: Wiring diagram for the base station. All resistors (R7 through R8) should be 100Ω.

Software

The device software is written in Python on the computer and the microcontroller is programmed using the Arduino IDE and its programming language.

All source code can be found here: https://github.com/JYB78/CE347_GUI

The provided files are only officially supported with the software and software library versions listed below.

- Python 3.8.5
- Pyserial 3.5
- PyAutoGUI 0.9.52
- Pillow 8.0.1

Microcontroller

The code on the microcontroller serves to transfer information from the glove sensors to the input processing code on the computer. This is achieved through a three-step looping process:

1. Poll all sensors (two buttons, two flex sensors, and the camera) to collect input data
2. Compile all sensor inputs into a packet that may be sent wirelessly to the computer
3. Use the built-in bluetooth hardware to wirelessly transmit the data packet.

This is shown in Figure 10.

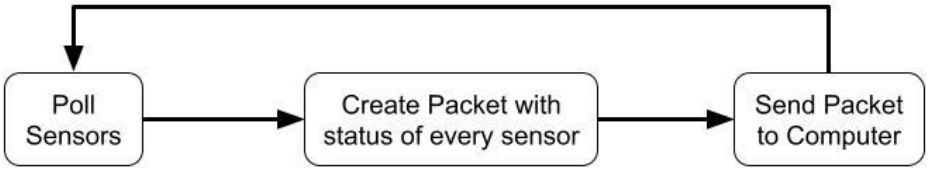


Figure 10: Diagram of microcontroller software

The format of the data packet is shown in Figure 11. This packet contains 20 bytes of data. The first 16 bytes are read from the camera using I2C and contain position data for every point. Byte 16 contains the button data. Bytes 17 through 19 carry the flex sensor data, which is formatted as described in figure due to the 12-bit resolution of the microcontroller ADC.

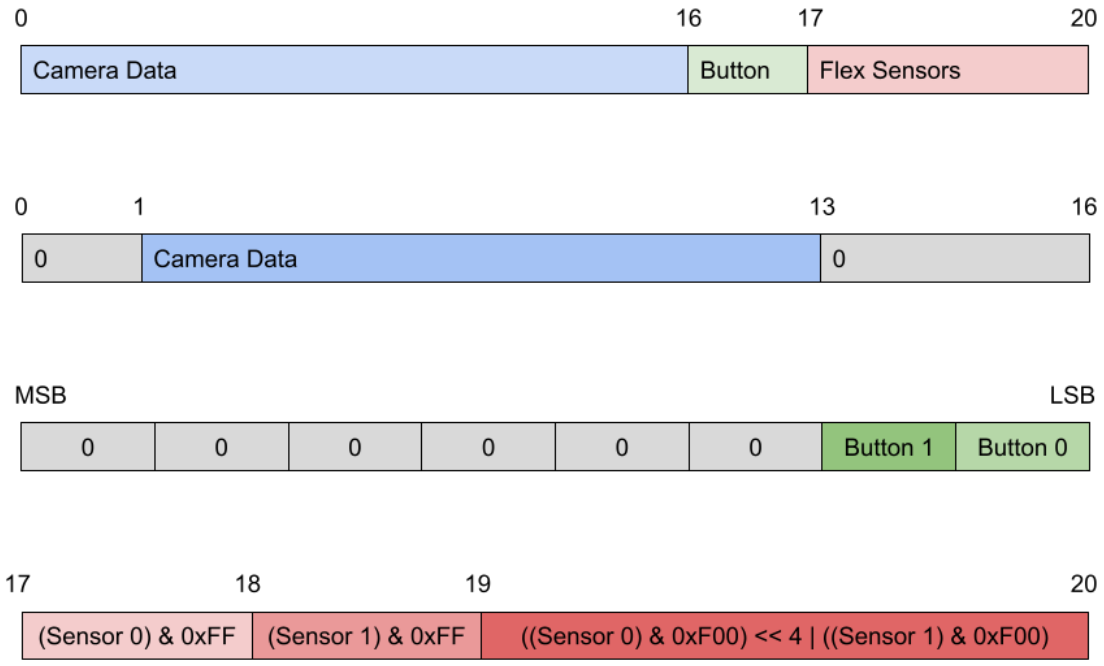


Figure 11: Data packet format

The 20-byte data packet is sent to the computer using bluetooth serial. To prevent a backlog of data on the computer, the microcontroller software first waits for a signal from the computer indicating it has already processed the previous packet. This may be sent at any time from the computer and stored in the microcontroller serial buffer. Once this packet is received, the microcontroller sends the 20 bytes of data and restarts the loop shown in Figure 11.

The Controller Program

The main controller program has a few different functionality, which includes communicating with the microcontroller via Bluetooth, communicating with the GUI via polling, processing raw input data, and lastly interfacing with the OS for mouse functions.

Communicate with GUI

The controller program communicates with the GUI by polling a file named “currentMode.pkl”. This file is updated whenever the user switches to a different mode or applies any changes to the button mapping. The controller polls the last modified time of the file and if the time is different compared to the previous recorded time, the controller reads the file and obtains the new sets of function mappings.

Control Mouse Functions

Communicate with Microcontroller

The ESP32 microprocessor communicates with the controller program via Bluetooth. The python module “pyserial” is used to establish serial communication between the microcontroller and the computer. With the format of the data packet explained in the previous section, the controller decodes the message and turns them into the raw variables: button0, button1, sensor0, sensor1, Ix, Iy. The two button inputs are further processed to distinguish whether it is a short or a long press. And the Ix Iy coordinates from the IR camera are also processed relative to the user’s screen size to get the exact x and y coordinates of the cursor position.

Interfacing with the OS

Interfacing with the OS is done using the “PyAutoGUI” module, where it has functions built-in to apply cursor movements, clicks, and keyboard presses. The detailed usage of the module can be found in the link here: https://pyautogui.readthedocs.io/en/latest/index.html

The GUI

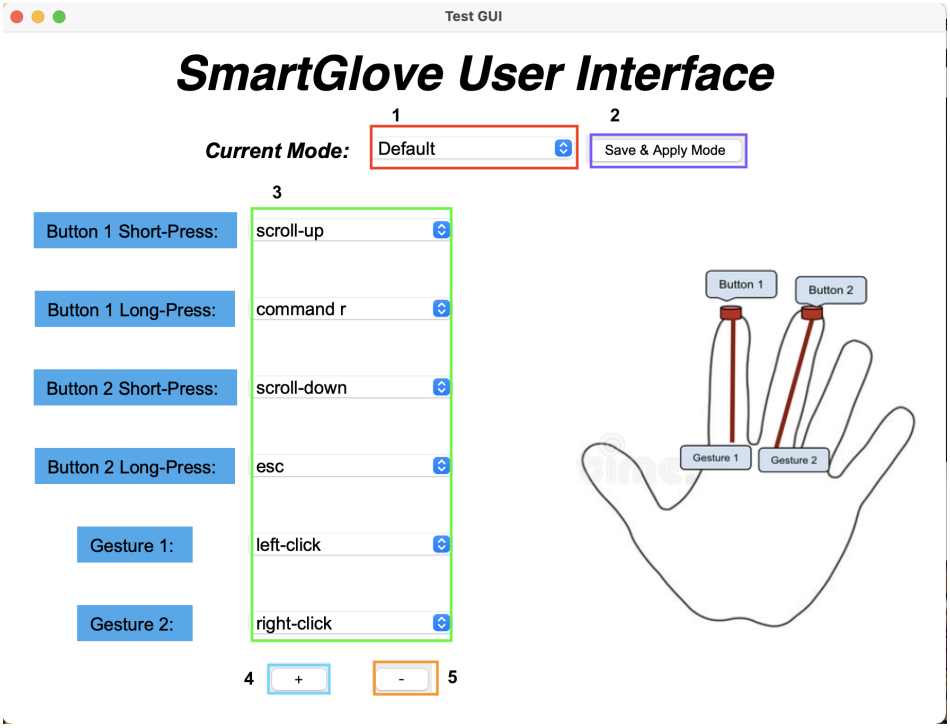


Figure 12: Overview of the GUI

The GUI is developed using Tkinter, which is a simple GUI module that comes built-in in Python 3. As shown in figure 12, the GUI can be divided into 5 different parts.

Part 1 is the mode dropdown menu, which comes with three modes to choose from: “Default”, “Accessibility”, and “Custom”. The button mapping will change to saved mapping of the selected mode automatically upon user selection. If no saved mapping is found, the button mapping will become the default “ese”, “ese”, “ese”, “ese”, “left-click”, “right-click” mapping.

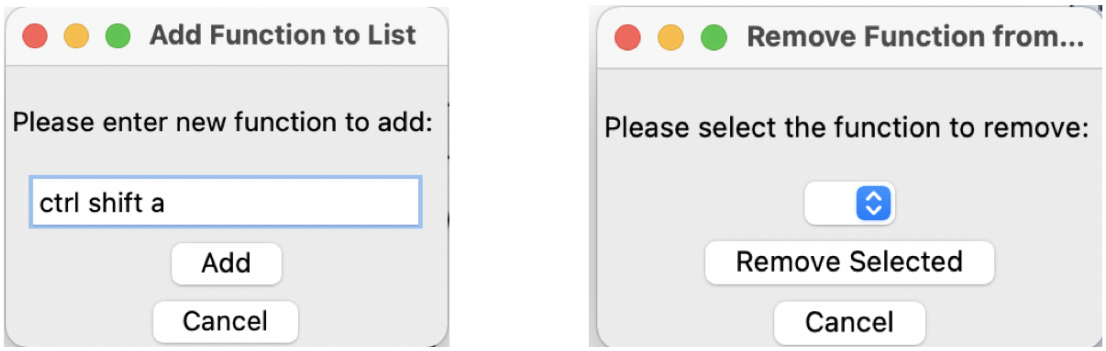
Part 2 is the “Save & Apply Mode” button, which should be used when the user makes any change to the button mapping of the current mode and would like to save and apply these changes. Once clicked, the GUI will write the current mapping to a corresponding .pkl file, which will be read whenever the designated mode is selected.

Part 3 is the function list dropdown for button mapping. All six dropdowns shares the same list of functions which by default includes: "esc", "delete", "enter", "space", "command c", "command v","command r","scroll-up","scroll-down", "left-click", and "right-click". The list can be modified freely using buttons in Part 4 and 5. Once the user makes any changes to the mapping, the user has to click on the “Save & Apply Mode” button to actually apply the changes.

Part 4 and 5 are buttons used to add and remove items from the function list used in Part 3. As shown in figure 13, the “+” button in Part 4 will open up a new pop-up window where you can enter any hotkey combinations ***up to three keys at max***, each key separated by ***space***. The full list of keys are listed here:

- 'left-click', 'right-click', 'scroll-up', 'scroll-down', 'space', 'enter', 'ctrl', 'shift', 'option', 'esc', 'alt', 'command', 'del', 'tab', 'backspace', 'capslock', 'up', 'down', 'left', 'right',
- '!', ' " ', '#', '$', '%', '&', " ' ", '(', ')', '*', '+', ',' , '-', '.', '/',
- '0', '1', '2', '3', '4', '5', '6', '7','8', '9',
- ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~',
- 'a', 'b', 'c', 'd', 'e','f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'

On the other hand, the “-” button in Part 5 will open up another pop-up window, where you can remove any item in the function list using the dropdown as shown in figure 13.



Part 4 *Part 5*

Figure 13: Pop-up Windows of Part 4 and 5

Limitations and Constraints

Some limitations and constraints need to be kept in mind in order for the system to work. General constraints include using an Arduino IDE compatible microcontroller as the code uses Arduino libraries. Additionally, all of the glove components must be able to fit on a glove.

In regards to the IR tracking camera, the microcontroller must support IIC in order to be able to communicate with the camera. This specific camera model was also chosen due to its ability to track multiple IR light sources, so any replacement IR tracking cameras must support tracking at least two IR light sources in order for the tracking algorithm to work properly. It would also need a tracking range of at least one meter in order to be used comfortably by the user.

The IR LEDs were placed on the base station instead of on the glove because it is very easy for a LED to be covered by the hand with any hand movements. It is also easier for the camera to track the LEDs if they are stationary on the base, being able to make head on contact with the camera lens.

There is also a requirement for the speed of polling/feedback in order for the glove to communicate properly with the computer. If the polling is not fast enough, then the user could experience delays with cursor movement and other inputs. This needs to be a minimum of 30 /s, which is the limit of human perception.

Future Development

There are several points of improvement for this prototype to move it towards full product readiness. Though the current prototype meets most of our specifications, there are some areas of improvement that would help with marketability and usage.

Improving The Glove’s Look

Currently, the glove has much of the electronics exposed which poses issues with pieces becoming disconnected and potential safety issues if the electronics get wet. Additionally, the lack of structure and neatness on the glove makes it difficult for a user to put it on without help. For future development, we would want to look into ways to house the electronics out of sight on the glove while still maintaining the glove’s wearability.

Battery

The current prototype uses a Lithium Polymer battery which can create high temperatures while in use. This heating issue can become hazardous to users after a long period of use. Additionally, a lipo battery can damage other electronic components if it is overcharged and become a safety hazard to users.

Further, the battery had to have a pocket sewn onto the glove that hanged off the glove. With a smaller battery, the electronic footprint on the glove could be smaller and make it easier to get the glove on. Research into different batteries would be useful in making the glove safer for the user and perhaps allow for a more compact glove.

Cursor Improvement

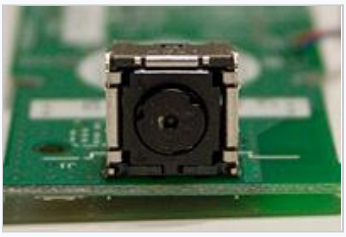
The cursor movement at the moment is not as responsive as a normal mouse, which prevents the glove from being a good alternative mouse. Further improvements could be done by using a better camera with better resolution or by optimizing the code to make it run faster or poll the glove more.

Personalization

The current prototype does not have the ability to change the gesture detection easily, making it difficult for users with hand-related disabilities to meet the requirements for using the gestures. Having the ability to change when a flex sensor is detected as “flexed” would allow for more people to be able to use the glove and help those who may not be able to use a mouse.

Another point of personalization is having more than 3 modes for the user to customize. While three is a lot for some, there may be people who like having many preset configurations of the glove. In addition, the glove may be used by several people who all have different preset preferences and having the ability to have as many modes as a user wants would allow for more flexibility and may be more attractive to potential users.

IR Camera



Wii remote camera

The Wii Remote includes a 128x96 monochrome camera with built-in image processing. The camera looks through an infrared pass filter in the remote's plastic casing. The camera's built-in image processing is capable of tracking up to 4 moving objects, and these data are the only data available to the host. Raw pixel data is not available to the host, so the camera cannot be used to take a conventional picture. The built-in processor uses 8x subpixel analysis to provide 1024x768 resolution for the tracked points. The Sensor Bar that comes with the Wii includes two IR LED clusters at each end, which are tracked by the Wii Remote to provide pointing information. The distance between the centers of the LED clusters is 20 cm (as measured on one unit).
The IR Camera is enabled by setting bit 2 on output reports 0x13 and 0x1a:

```
(a2) 13 04
(a2) 1a 04
```

The first enables a 24MHz pixel clock on pin 7 of the camera. The second pulls pin 4 low - probably an active-low enable.

Mechanical Characteristics

The camera component is mounted on the bottom surface of the circuit board. The camera module itself is mounted in a socket perpendicular to the circuit board; to remove just the camera module, no desoldering is required. The process is as follows:
First, orient the camera so that you are looking into the lens with the PCB horizontal and below the lens.. There are four metal clips, two on each of the vertical sides of the socket. Use something tiny to slide between each metal clip and the camera module: maybe wire wrap wire? Then look at the back of the camera module, opposite the lens. There is a small rectangular hole in the middle of each vertical side of the socket. Use a pin or something to pry/press the camera module out.
Once the camera module is free of its socket, it may be further disassembled by gently prising up the tiny PCB with gold contacts; this is gently glued to the module's structure, but will come loose without damage. At this point you have three pieces: the camera socket, still attached to the Wiimote PCB, the camera module housing, complete with lens and dichroic filter (of unknown optical properties), and a tiny PCB with the camera chip and eight gold contacts on the bottom.

Optical Characteristics

The IR camera has an effective field of view is about 33 degrees horizontally and 23 degrees vertically (as measured on one unit). With the IR-pass filter intact, 940nm sources are detected with approximately twice the intensity of equivalent 850nm sources, but are not resolved as well at close distances. If the filter is removed, it can track any bright object. However, the IR filter referred to here is not only the dark plastic window of the wiimote but also a teensy slab of dichroic-coated glass inside the camera module. One may operate the wiimote having installed neither, one or the other, or both filters.

Initialization

Reminder

Remember to set bit 2 (0x04) on the first byte of the Output Reports to write to registers!

The following procedure should be followed to turn on the IR Camera:

1. Enable IR Camera (Send 0x04 to Output Report 0x13)
2. Enable IR Camera 2 (Send 0x04 to Output Report 0x1a)
3. Write 0x08 to register 0xb00030
4. Write Sensitivity Block 1 to registers at 0xb00000
5. Write Sensitivity Block 2 to registers at 0xb0001a
6. Write Mode Number to register 0xb00033
7. Write 0x08 to register 0xb00030 (again)

After these steps, the Wii Remote will be in one of 3 states: IR camera on but not taking data, IR camera on and taking data and half sensitivity, IR camera on and taking data at full sensitivity. Which state you end up in appears to be pretty much random. Repeat the steps until you're in the desired state. To avoid the random state put a delay of at least 50ms between every single byte transmission.

The Wii preforms these steps slightly different, differences in bold:

1. Enable IR Pixel Clock (send 0x06 to Output Report 0x13)
2. Enable IR Logic (send 0x06 to Output Report 0x1A)
3. Write 0x01 to register 0xb00030
4. Write Sensitivity Block 1 to registers at 0xb00000
5. Write Sensitivity Block 2 to registers at 0xb0001a
6. Write Mode Number to register 0xb00033
7. Write 0x08 to register 0xb00030 (again)

Adding bit 0x02 to reports 0x13 and 0x1a is a request for acknowledgement (if set, wiimote will respond with report 0x22).

Sensitivity Settings

Sensitivity is controlled by two configuration blocks, 9 bytes and 2 bytes long. The following settings are known to work:

Block 1	Block 2	Notes
`00 00 00 00 00 00 90 00 C0`	`40 00`	Suggested by Marcan

00 00 00 00 00 00 FF 00 0C	00 00	Suggested by Kestrel (max sensitivity)
00 00 00 00 00 00 90 00 41	40 00	Suggested by inio (high sensitivity)
02 00 00 71 01 00 64 00 fe	fd 05	Wii level 1
02 00 00 71 01 00 96 00 b4	b3 04	Wii level 2
02 00 00 71 01 00 aa 00 64	63 03	Wii level 3 (Suggested by Cliff)
02 00 00 71 01 00 c8 00 36	35 03	Wii level 4
07 00 00 71 01 00 72 00 20	1f 03	Wii level 5

The last byte of Block 1 determines the intensity sensitivity, with increasing values reducing the sensitivity. Both bytes of Block 2 must be zero for the full sensitivity range to be available. Setting the sensitivity as high as possible, without unwanted light being tracked, is recommended to achieve the highest subpixel resolution. As the sensitivity is reduced, the subpixel resolution also reduces, approaching the true sensor resolution of 128x96.

Data Formats

The IR Camera can return different sets of data describing the objects it is tracking. When the IR camera identifies an object, it assigns it to the first available object slot. If an object moves out of view, its slot is marked as empty (returns 0xFF data), but other objects retain their slots. For example, if the camera is tracking two objects and the first moves out of view, the data returned will be [empty, second object, empty, empty]. With more than four objects visible, the camera is prone to rapidly switching between some of them. This could allow perception of more than four objects, at a reduced response speed and reliability.

Mode	Mode Number
Basic	1
Extended	3
Full	5

The data format MUST match the number of bytes available in the Reporting Mode selected. Even choosing a mode with space for more bytes than necessary will not work, it has to be an exact match.

Basic Mode

In Basic Mode, the IR Camera returns 10 bytes of data corresponding to the X and Y locations of each of the four dots. Each location is encoded in 10 bits and has a range of 0-1023 for the X dimension, and 0-767 for the Y dimension. Each pair of dots is packed into 5 bytes, and two of these are transmitted for a total of 4 dots and 10 bytes.

This is the data format for a pair of objects:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

0	**X1**<7:0>			
1	**Y1**<7:0>			
2	**Y1**<9:8>	**X1**<9:8>	**Y2**<9:8>	**X2**<9:8>
3	**X2**<7:0>			
4	**Y2**<7:0>			

Extended Mode

In Extended Mode, the IR Camera returns the same data as it does in Basic Mode, plus a rough size value for each object. The data is returned as 12 bytes, three bytes per object. Size has a range of 0-15.
This is the data format for each object:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**X**<7:0>							
1	**Y**<7:0>							
2	**Y**<9:8>		**X**<9:8>		**S**<3:0>			

Full Mode

In Full Mode, the IR Camera returns even more data, 9 bytes per object for a total of 36 bytes for all four. The data is split up between two input reports of 18 bytes each (see Data Reporting Mode 0x3e/0x3f). The first three bytes of each object are the same as the extended mode, and are followed by the bounding box of the pixels included in the blob along with a deeper intensity value. The data format of each object is:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**X**<7:0>							
1	**Y**<7:0>							
2	**Y**<9:8>		**X**<9:8>		**S**<3:0>			
3	0	**X min**<6:0>						
4	0	**Y min**<6:0>						
5	0	**X max**<6:0>						
6	0	**Y max**<6:0>						
7	0							
8	**Intensity**<7:0>							