HIGH SCHOOL EDITION
Journal of Student Research
Volume 12 Issue 4 (2023)

Hands Free Computer Interaction - The Intellipointer

James Li, Aaron Pan and Matthijs de Lange

ESF West Island School, Hong Kong

ABSTRACT

The intention of this project was to create an alternative peripheral device to fit the needs of those with dexterity issues or other similar limitations. Using a computer is an essential part of many people's lives, especially in this day and age where computer use is essential in almost every aspect. However, for those with physical or neurological limitations, this process becomes much more difficult, as conventional peripherals such as mice or keyboards do not take these intricacies into consideration. Currently, 13% of the United States has some sort of disability, and by 2050, the world's population of people aged 60 or older will be double of the current 1.1 billion people. With the market lacking a pragmatic and practical solution, his project aims to solve this problem by creating an affordable peripheral device with an alternative method of input to aid those who find it difficult to use traditional peripheral devices.

Introduction

Typically, a person will use traditional peripheral devices to access a computer - in particular, a mouse and keyboard. The importance of computers is also constantly increasing, as the proportion of workers using a computer at their main job has risen from 33% in 1989 to 57% in 2000. This dependence is further shown as in 2022, 92.9% of United States households own at least one computer - a 41.2% increase compared to 2000 ("IBISWorld - Industry Market Research, Reports, and Statistics"). However, these devices do not accommodate people who find it difficult to use them and as the ageing population continues to grow, the problem of finding a suitable alternative becomes more and more prevalent. Various aspects of mouse control like clicking, moving, fine positioning, etc. can be difficult for older people (Smith et al., 1999) and people with disabilities which prohibit them from using a mouse properly. A study done in 1996 showed that measures such as movement speed, reaction time, and coordination index derived from phase plane analysis, showed a significant difference between the normal population and Parkinson's population (Riviere and Thakor 7), as the Parkinson's population performed much worse compared to the normal population.

Unfortunately, there is no pragmatic solution in the market. For alternatives, trackerballs or touchpads are common options, but they may require a level of dexterity which some may not possess. Footmouse, joysticks are also popular choices, but it may require the users to have enough strength to lift their legs or yet again, enough dexterity to move the joystick accurately. There are also some more modern alternatives, such as eye-tracking hardware which can be quite expensive, ranging from just under $1000 to over $70,000. This project aims to fix this problem by providing both an affordable and practical tool that can be used by virtually anybody. It is a head-mounted device called the Intellipointer where users can control a mouse cursor with their head movement and direction. For example, if the user turns their head to the right, the cursor would move right; Fig 1 and Fig 2 below illustrate in more detail the basic principles of how the Intellipointer works.

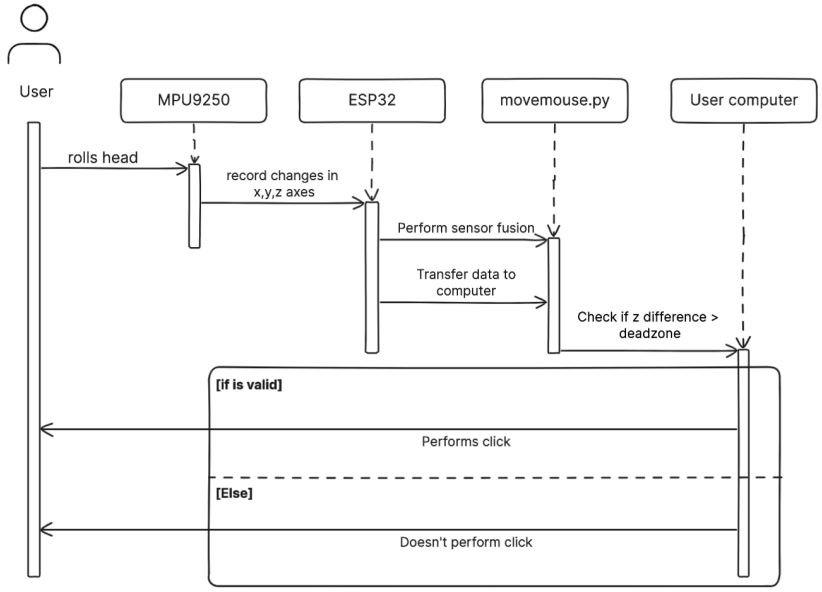

Figure 1. Sequential Diagram of Intellipointer Clicks

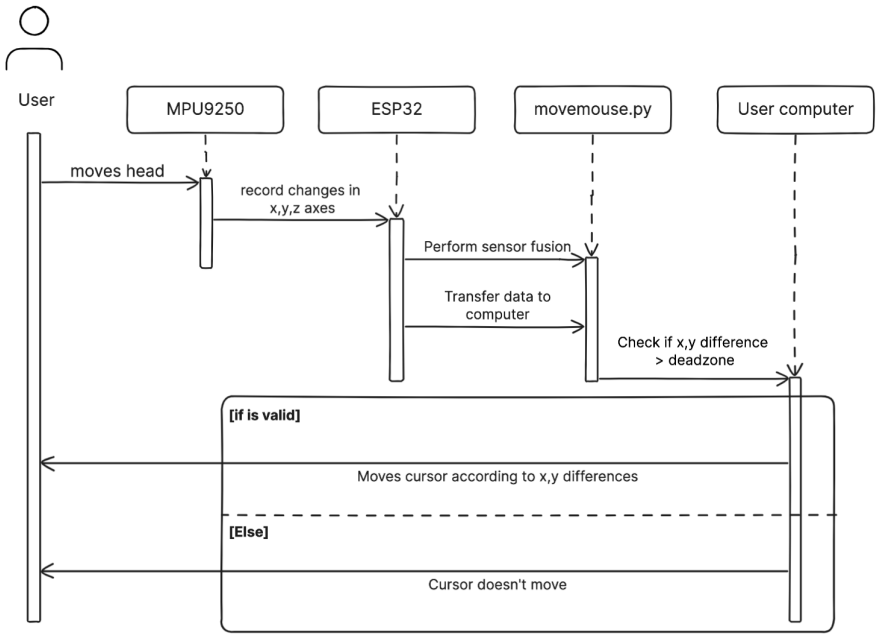

Figure 2. Sequential Diagram of Intellipointer Clicks

Methods and Results

We did several tests to measure the accuracy, drift, and user experience of the Intellipointer for both version 1 and version 2. The methods will be listed below:

Accuracy Test

1. First, place the IMU at a 0 degree angle
2. Use a protractor next to it to measure angle
3. Rotate the gyroscope by increments of 5 degrees (0-90deg) and calculate the change in x, y, z values
4. Repeat for all orientations, and compare recorded values to expected values.

For V1 of the Intellipointer, we'd discover a big accuracy issue upon running the test. For both the roll and pitch, the deviation was $0 \leq \theta \leq 20$ which was far too large. We discovered that this is mostly caused by using acceleration values from a single axis rather than from all 3. After changing the way we calculated pitch and roll by using all three x, y, z acceleration values, this issue was mostly solved.

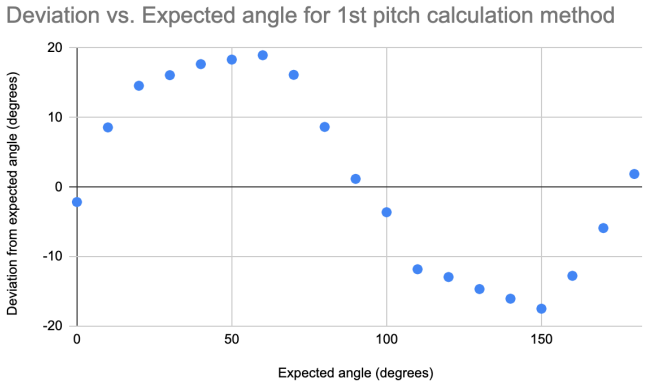

Figure 3. Deviation vs Expected Angle for Pitch

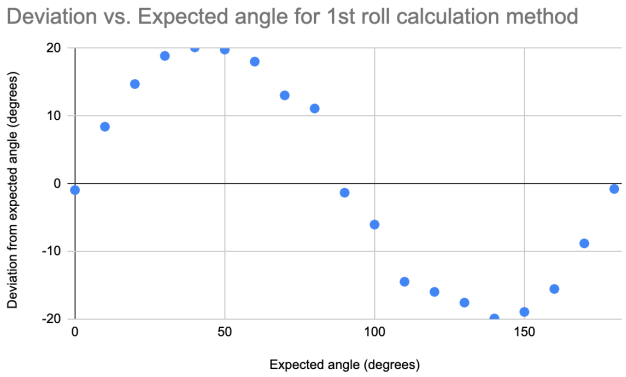

Figure 4. Deviation vs Expected Angle on Roll

Drift/Jitter Test

1. Place the device directly towards the front of the computer
2. Change the device (x, y, z) in different directions for a set angle.
3. Measure the amount of drift or jitter in the mouse pointer movements for new units and old units. This can be done by plotting the physical movement and displayed movement and measuring how far it differs from the expected value.

4. Repeat the test multiple times and calculate the average amount of drift or jitter.

For V1 of the Intellipointer, we discovered a severe drift issue upon running the test. We placed the Intellipointer on a flat surface - the graph on the left in Fig 5 shows the expected yaw. When there is no movement, yaw doesn't change. However, the graph on the right in Fig 5 shows the yaw drifting around 0.5 °/s as the gradient is not constant. After further research, we discovered that this is an inherent flaw with IMUs which we'll go into more detail in the Hardware section. This is fixed in version 2 of the Intellipointer, where we'll cover the improvements in both the Hardware and Software section.

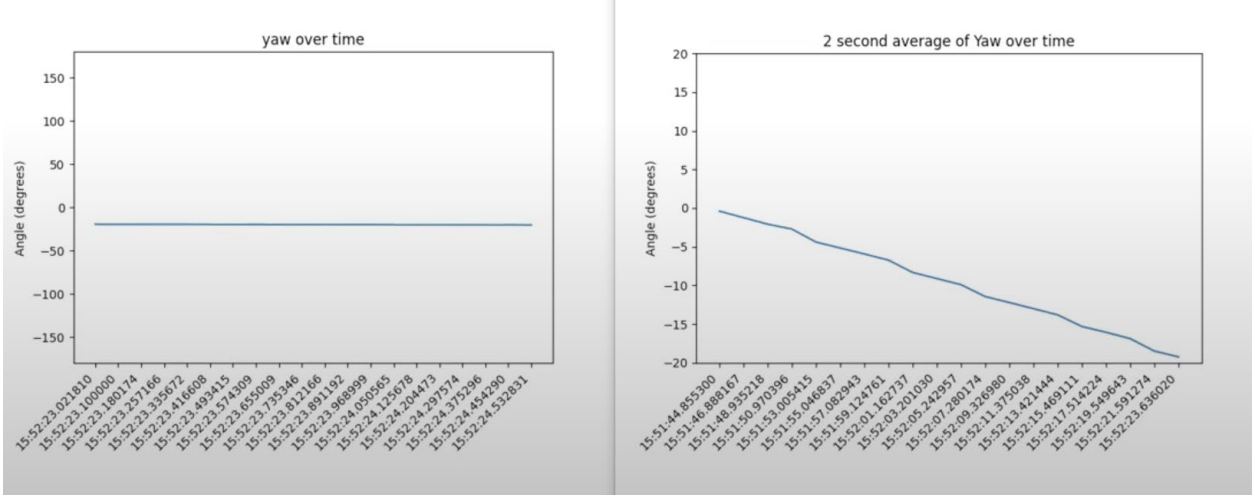

Figure 5. Expected yaw (left) vs. Recorded yaw (right)

User Experience Investigation

1. The test is set to a sample of 20 people to use the Intellipointer
2. Have 4 preset settings for users to test:
    a. Acceleration, low sensitivity, big roll deadzone
    b. Acceleration, low sensitivity, small roll deadzone
    c. No acceleration, high sensitivity, big roll deadzone
    d. No acceleration, high sensitivity, small roll deadzone
3. Participants will engage in a timed (1 minute) dot-clicking task on a screen using each setting
4. The score for each setting by each person is recorded to determine the optimal setting, and users will also give feedback for each setting

From this investigation, we discovered clear pros and cons as stated by users for version 1. For one, many find the Intellipointer intuitive and easy to use as cursor movement is only dictated by head movement and direction. However, there were also several weaknesses that users identified. There was quite a noticeable delay in movement which made it difficult to control - paired with the jitter in mouse movement, it was not very user-friendly. The substandard click detection and regular calibration needed also didn't help, as it further interfered with the usability of the device. In addition, many users said the hat and device weren't secure and could be a potential safety hazard.

We also found that Setting C, the setting with no acceleration, high sensitivity, and big roll deadzone, was the optimal setting as almost half the participants found it most preferable due to its linear speed, controllability, and precision despite slower cursor speeds. With all this feedback, we aimed to fix these issues in V2 and further versions.

Material

- Inertial Measurement Unit (MPU9250, MPU6050)
- Microprocessor (ESP32)
- Lithium Ion / Polymer Battery Pack
- USB Cable
- Scotch Tape / Velcro White Tape
- No longer used: Raspberry Pi 4

Hardware

For our hardware, we are using a setup of an MPU9250, ESP32, and a Lithium Polymer Battery.

An MPU9250 is an IMU, or an Inertial Measurement Unit. An IMU is an electronic device that measures an object's specific force (acceleration), angular rate, and orientation. In our case, the MPU9250 is a 9-axis IMU, as there are 3 different sensors measuring changes in 3 different axes - the gyroscope, accelerometer, and magnetometer in the X, Y, and Z direction.

A gyroscope, or in an MPU9250, a 3-axis MEMS (Micro-Electro-Mechanical-System) gyroscope measures the angular velocity around a fixed axis with respect to inertial space. It measures the angular rate by using a small, vibrating mass that moves in response to the rotation, and the movement generates a force called the Corialis acceleration. This force is sensed by the gyroscope through a method called capacitive transduction, which produces a signal that can be used to measure the rotation rate (Watson).

An accelerometer, or a 3-axis capacitive based MEMS accelerometer, works by measuring changes in capacitance. Capacitance is a measure of the capacity of storing electric charge for a given voltage ("Capacitance and Dielectrics" 3) - these changes in capacitance correspond to acceleration in a particular axis (Baliveau et al. 1999). This change in capacitance happens due to how an accelerometer works off the principle of a mass on a spring. When the mass is accelerated, it has a tendency to resist changes in its motion due to inertia and as a result, the spring experiences a force. This force changes the distance between two capacitor plates and causes a change in capacitance. This change is detected by modulation/demodulation circuits, to which the output is proportional to the change in acceleration (Zohra et al. 2014). As this accelerometer is 3-axis, the acceleration for the X, Y, and Z axis is measured, so it consists of three sensing elements, each oriented along a different axis.

A magnetometer measures magnetic fields. The magnetometer in question is a 3-axis MEMS which functions based on the Hall effect method. When current is passed through a conductor in the presence of a magnetic field, the magnetic field causes the current carriers to be deflected, creating a voltage difference, or an electromotive force perpendicular to the current. From this, Edwin Hall concluded that this electromotive force is proportional to the product of the intensity of the magnetic field and drift velocity (Popovic 2), and therefore, the magnetic field strength can be derived from this formula.

By using the data from all three sensors, we can gain accurate readings which can help determine changes in orientation, direction, magnetic fields, and velocity. This is a big improvement compared to our first prototype, as we'd used an MPU6050, which is a 6-axis IMU consisting of only a gyroscope and accelerometer. There was lots of drift in cursor movement with the initial prototype, and this is partially due to the lack of a

magnetometer. With a magnetometer in our second prototype (V2), we gain two reference data points in the north and south poles. As they remain constant, these points can be taken into account - combined with the software improvements that will be discussed later, the issue of drift is effectively mitigated.

The ESP32 microcontroller is also used in this device. A microcontroller is a processor that is simplified or stripped-down, containing memory, timers, parallel I/O pins, and other built-in peripherals (Gridling and Weiss) - essentially a mini computer that performs tasks like executing program instructions, controlling input and output operations, managing memory, etc. The ESP32 is a powerful SoC (System on Chip) microcontroller that combines features such as integrated Wi-Fi 802.11 b/g/n, dual-mode Bluetooth (version 4.2), 4MB of flash and a range of peripherals (Babiuch et al. 3). An SoC is a sophisticated integrated circuit that puts the essential, functional components of a completed product (i.e. smartphone, tablets, etc.) into a single chip. Typically, an SoC design integrates a programmable processor, on-chip memory, and specialized hardware units for accelerating specific functions. In addition, it also facilitates communication with peripheral devices and interfaces with the external world (i.e. sensors, motors) (Martin and Chang). The main advantage of using an ESP32 is cost efficiency, as consolidating multiple components onto a single chip reduces space requirements, lowers manufacturing costs, and speeds up development timelines. Compared to our V1 where we used a Raspberry Pi 4, using an ESP32 greatly reduced weight without sacrificing performance, something which greatly helps with ergonomics and user experience.

The combination of an ESP32 and MPU9250 is one that allows for a good balance between compactness and performance. With a combined weight of around 20 grams, it is lightweight and ergonomic for practical use to virtually anybody and is also 100 grams lighter than V1 of the Intellipointer. Of course, weight isn't the only factor in determining what is ergonomic, but it certainly is a good indicator of whether it is. Moreover, the ESP32 and MPU9250 integration has a much more compact footprint. This aspect is particularly crucial for physically disabled individuals, as it enhances usability by reducing physical strain and facilitating easy handling. By incorporating and embracing ergonomic principles, this solution empowers disabled individuals to effortlessly engage with the device, enabling them to seamlessly access and interact with the digital realm.

The Intellipointer as a whole is mounted on a cap due to reasons that impact practicality and user experience. By mounting the electronic components on a cap, it guarantees complete hands-free interaction between the user and computer, all while still maintaining ease of use and comfort. Compared to other options, like helmets or headbands, this cap-mounted design offers a more discrete and conventional solution by not deviating from daily attire. In addition, our goal is to create a device that can be used by anyone regardless of age, condition, or income. Caps are accessories that can be commonly found in households and if not, can be purchased for very low prices - this allows for greater scalability and accessibility in a different sense.

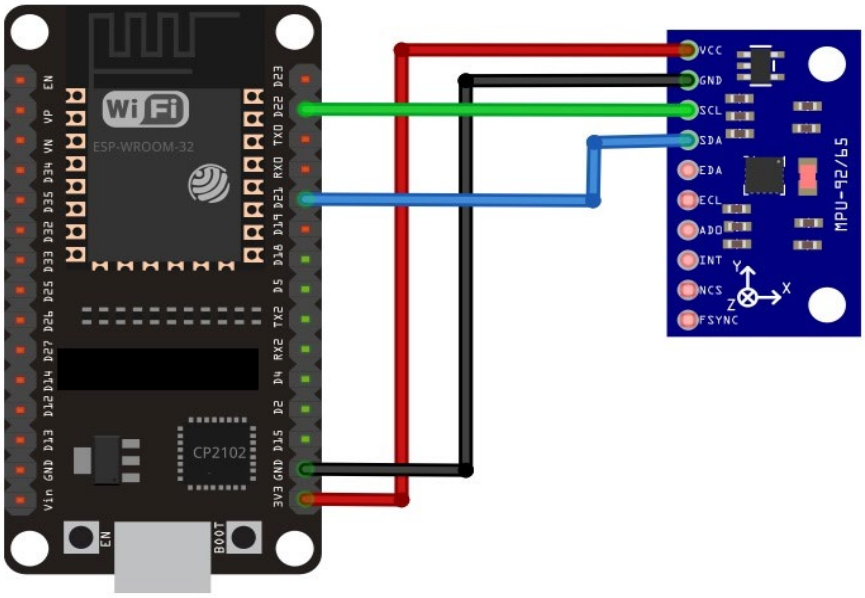

Figure 6. Hardware diagram of Intellipointer (source: microcontrollerslab)

Figure 7. Image of the Intellipointer, first prototype

Software

The software of the Intellipointer can be split into two parts: the code stored on the device itself, and the code stored on the user's computer. The code on each device can be further split into 2 functions; sensor reading and data transmission on the Intellipointer, and data receiving and decoding on the user's computer.

As mentioned, the code on the Intellipointer can be split into 2 halves; one half is responsible for decoding and fusing sensor data, while the other is responsible for transmitting this data to the user's computer via a Wi-Fi connection. Using source code adapted from ymtlab on github(ymtlab), we are able to convert the raw angular rate, acceleration, and electromotive force into degrees per second. These values are assigned to 3 different variables, each corresponding to the sensor of origin("accel", "gyro" and "mag"). These values are then passed through a sensor fusion algorithm to increase the precision of the data. The algorithm (Winer 2018) is based on the original program written by Sebastian Madgwick (Madgwick) and adapted to Python. The scope and depth of sensor fusion algorithms are far beyond the high school level; however, in its simplest form, sensor fusion is the combination and processing of data from different sensors that result in a more accurate value in

comparison to that of an individual sensor. In this instance, the sensor fusion helps to minimize gyroscopic yaw by utilising the fixed magnetic north as an additional data point, eliminating the accumulation of gyroscopic drift over longer periods of time. Finally, all the data is concatenated into a string, with x, y, and z values from each interaction being merged into 1 line, ready for transmission.

Once all the data has been processed on the Intellipointer, it is then transmitted to the user's computer via a Wi-Fi connection. By establishing an endpoint on the user's computer in the form of a server, as long as both devices are on the same network, data can be transmitted from the Intellipointer to the computer via a socket connection. Every fifth set of data is transmitted to the server; this is done in order to minimise false negative values generated as a result of the sensor fusion algorithm. During testing, it was discovered that upon a sudden change in direction, the algorithm would initially overcorrect and thus output values of the opposite direction, much to the user's confusion. Once received by the user's computer, the data is written to a text file in string form, before then being split into the respective x, y, and z values.

Finally, these x, y, and z float values are fed into a final Python program on the user's computer. This program utilises the PyAutoGUI module to control the user's mouse. By calculating the difference between x, y, and z values between iterations, the corresponding direction of movement based on these components can be determined. Additionally, a deadzone, not too dissimilar to one found on a game console controller, has been implemented into the code. This deadzone is created using an if condition; if the change in directional values is above a certain threshold, only then will the cursor move; otherwise, it remains stationary. This is done to make the device easier to use; it was found that if every little head movement were to translate into cursor movement, the user would have to strain their neck to an extra extent, rendering the device difficult to use over long periods of time.

Mathematics

The mathematics to convert raw values into more useful data will be briefly shown below.

For the conversion of accelerometer and gyroscope data, the values need to be converted from angular velocity to degrees per second. To do so, the sensitivity (gres and ares) needs to be determined based on the full-scale range (gfs and afs) - this can be found on the sensor's datasheet. From Fig 8, the method of calculating both sensitivities is shown. Divide every the gfs and afs by 32768 in an if-else statement - this is because all raw data registers have a range of -2^{15} to 2^{15} (16 bit, or 32767 to 32767), so dividing it by 32768 gives us the smallest possible change in angular velocity or acceleration which the sensors can detect.

Hence, from Fig 9, it can be seen that the data of the x, y, and z axis from the gyroscope and accelerometer is read and extracted through an I2C (Inter-Integrated Circuit) Interface. The subsequent lines (lines with functions *self.ares* and *self.gres*) apply a scaling factor that converts the recorded values into meaningful data to 3 decimal places with the *round* function.

```
def setting(self, gfs, afs):
    if gfs == self.GFS_250:
        self.gres = 250.0/32768.0
    elif gfs == self.GFS_500:
        self.gres = 500.0/32768.0
    elif gfs == self.GFS_1000:
        self.gres = 1000.0/32768.0
    else:  # gfs == GFS_2000
        self.gres = 2000.0/32768.0

    if afs == self.AFS_2G:
        self.ares = 2.0/32768.0
    elif afs == self.AFS_4G:
        self.ares = 4.0/32768.0
    elif afs == self.AFS_8G:
        self.ares = 8.0/32768.0
    else: # afs == AFS_16G:
        self.ares = 16.0/32768.0
```

Figure 8. Calculation of the sensitivity of gyroscope and accelerometer

```
def read_accel(self):
    data = self.i2c.readfrom_mem(self.address, self.ACCEL_OUT, 6)
    x = self.data_convert(data[1], data[0])
    y = self.data_convert(data[3], data[2])
    z = self.data_convert(data[5], data[4])

    x = round(x*self.ares, 3)
    y = round(y*self.ares, 3)
    z = round(z*self.ares, 3)

    return {"x":x, "y":y, "z":z}

def read_gyro(self):
    data = self.i2c.readfrom_mem(self.address, self.GYRO_OUT, 6)

    x = self.data_convert(data[1], data[0])
    y = self.data_convert(data[3], data[2])
    z = self.data_convert(data[5], data[4])

    x = round(x*self.gres, 3)
    y = round(y*self.gres, 3)
    z = round(z*self.gres, 3)

    return {"x":x, "y":y, "z":z}
```

Figure 9. Conversion to rounded and scaled gyroscope and accelerometer values

Therefore, this allows us to calculate the changes in the gyroscope or accelerometer, where it can be converted to mouse movement.

Next, the way of which the data from the magnetometer is processed will be described below. From Fig 10 below, the first three lines calculate the magnetometer coefficients for the x, y, and z axis by first subtracting the raw values by 128 and then dividing it by 256, and ultimately adding 1 to the quotient - this is done in order to convert the values into workable code that is more useful. Next, akin to the methods of conversion and procession undergone by the gyroscope and accelerometer, the raw values are multiplied by the magne-

tometer's sensitivity to convert the data into degrees per second. In addition, it is also multiplied by a magnetometer coefficient for each axis which allows us to scale movement appropriately according to the sensitivity of the magnetometer.

```
        self.magXcoef = (data[0] - 128) / 256.0 + 1.0
        self.magYcoef = (data[1] - 128) / 256.0 + 1.0
        self.magZcoef = (data[2] - 128) / 256.0 + 1.0

        self.i2c.writeto_mem(self.AK8963_SLAVE_ADDRESS, self.AK8963_CNTL1, b'\x0
        utime.sleep_ms(10)
        buffer = bytearray(1)
        buffer[0] = (mfs << 4 | mode)
        self.i2c.writeto_mem(self.AK8963_SLAVE_ADDRESS, self.AK8963_CNTL1, buffe
        utime.sleep_ms(10)

    def read_magnet(self):
        x, y, z=0, 0, 0

        data = self.i2c.readfrom_mem(self.AK8963_SLAVE_ADDRESS, self.AK8963_MAGN

        # check overflow
        if (data[6] & 0x08)!=0x08:
            x = self.data_convert(data[0], data[1])
            y = self.data_convert(data[2], data[3])
            z = self.data_convert(data[4], data[5])

            x = round(x * self.mres * self.magXcoef, 3)
            y = round(y * self.mres * self.magYcoef, 3)
            z = round(z * self.mres * self.magZcoef, 3)

        return {"x":x, "y":y, "z":z}

    def data_convert(self, data1, data2):
        value = data1 | (data2 << 8)
        if(value & (1 << 16 - 1)):
            value -= (1<<16)
        return value
```

Figure 10. Calculation and procession of raw magnetometer values

From this point on, the aforementioned sensor fusion algorithm essentially combines all values of each sensor from each axis to generate a more accurate reading and, thus more accurate movement.

Limitations

As with every project or device, the Intellipointer has its limitations.

The movement of the cursor is still not perfectly smooth with frequent jitters and pauses. This is likely caused by the fact that the programs are written in Python rather than C++, which is the natural programming language for an input peripheral like a mouse. Our original plan was to use an Arduino Pro Mini, which requires C++. In addition, our main program, main.py, is almost 1000 lines long, meaning it'd take more time to iterate through each line than a program with 100 lines. This inefficiency leaves us with lots of room to improve by making the program more compact and concise with methods like removing redundancies.

One other problem may be the fact that a Wi-Fi connection is used rather than a Bluetooth connection like most traditional wireless peripheral devices. Due to the library constraints of MicroPython, a socket had to be used over Bluetooth protocol; however, Bluetooth can be used if written with C++. The main advantage Bluetooth has over wifi is the fact it is more user-friendly; there is no need to look for IP addresses and constantly restart servers, which may be a daunting task to the average user. Additionally, the use of a server requires the deactivation of the Windows Defender firewall or any other security software. Most users would not be comfortable with doing so, and even then, it was discovered that during testing, third-party antivirus such as Norton or Avast would interfere with data transmission, further complicating the device setup and compromising the security of the device.

References

Babiuch, Marek, et al. “Using the ESP32 Microcontroller for Data Processing.” 2019 20th International Carpathian Control Conference (ICCC), May 2019, https://doi.org/10.1109/carpathiancc.2019.8765944.

Beliveau, A., Spencer, G.T., Thomas, K.A., and Roberson, S.L. "Evaluation of MEMS capacitive accelerometers." IEEE Design & Test of Computers, vol. 16, no. 4, Oct.-Dec. 1999, pp. 48-56. doi: 10.1109/54.808209.

Capacitance and Dielectrics. MIT, n.d., https://web.mit.edu/8.02t/www/802TEAL3D/visualizations/coursenotes/modules/guide05.pdf.

Gridling, Günther, and Bettina Weiss. Introduction to Microcontrollers. 2007, www.skylineuniversity.ac.ae/pdf/software-engineering/Microcontroller.pdf.

“IBISWorld - Industry Market Research, Reports, and Statistics.” Www.ibisworld.com, 26 Aug. 2022, www.ibisworld.com/us/bed/percentage-of-households-with-at-least-one-computer/4068/.

Madgwick, Sebastian. An Efficient Orientation Filter for Inertial and Inertial/Magnetic Sensor Arrays. 30 Apr. 2010.

Manjiyani, Zohra Aziz Ali, et al. "Development of MEMS Based 3-Axis Accelerometer for Hand Movement Monitoring." International Journal of Scientific and Research Publications, vol. 4, no. 2, Feb. 2014, pp. 326, ISSN 2250-3153, www.ijsrp.org.

Marshall, Katherine. “Statistics Canada: Working with Computers Vol.2, No.5.” Www150.Statcan.gc.ca, May 2001, www150.statcan.gc.ca/n1/pub/75-001-x/00501/5724-eng.html#:~:text=In%20a%20mere%20decade%2C%20the. Accessed 1 Aug. 2023.

Martin, G., and H. Chang. “System-On-Chip Design.” IEEE Xplore, 1 Oct. 2001, pp. 12–17, https://doi.org/10.1109/ICASIC.2001.982487.

Popovic, R. S. Hall Effect Devices, Second Edition. CRC Press, 2003.

Riviere, C N, and N V Thakor. “Effects of age and disability on tracking tasks with a computer mouse: accuracy and linearity.” Journal of rehabilitation research and development vol. 33,1 (1996): 6-15.

Smith, Michael & Sharit, Joseph & Czaja, Sara. (1999). Aging, Motor Control, and the Performance of Computer Mouse Tasks. Human factors. 41. 389-96. 10.1518/001872099779611102.

Watson, Jeff. “MEMS Gyroscope Provides Precision Inertial Sensing in Harsh, High Temperature Environments.” Analog.com, 2016, www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html.

Winer, Kris. “MPU-9250.” GitHub, 12 Nov. 2018, github.com/kriswiner/MPU9250. Accessed 14 Aug. 2023.

ymtlab. “Ymtlab - Overview.” GitHub, github.com/ymtlab.