HEAD GESTURED CONTROLLED MOUSE

A project report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION

PROJECT MENTOR:
Ms. VANDANA NATH
ASSISTANT PROFESSOR

SUBMITTED BY:
RAHUL SINGH THAKUR
08316412812
B.TECH (ECE), 8TH SEM

UNIVERSITY SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
SECTOR-16C, DWARKA, DELHI-110078

CERTIFICATE

This is to certify that report entitled "**HEAD GESTURED CONTROLLED MOUSE**" submitted by **RAHUL SINGH THAKUR** carried at USICT, GGSIPU, Dwarka, under the supervision of **Dr. Vandana Nath**. The dissertation work done is not derived from any source (existing reports, project works, internet related articles, books, etc), directly or indirectly. I, the student of B.tech (ECE) take full responsibility for the content of the report and its relevant source code modules, and hence accountable for the project work done by me.

Date:

Rahul Singh Thakur
83716412812 USICT,
GGSIPU

Date:

Dr. Vandana Nath
Assistant Professor
USICT, GGSIPU

ACKNOWLEDGEMENT

It would be my utmost pleasure to express my sincere thanks to my guide **Dr. Vandana Nath, Assistant Professor, USICT, GGSIP University** in providing a helping hand in this report. Her unflagging patience, creativity and immense knowledge that she shared with me have proved highly beneficial to me and have made this Project both possible and successful.

Date:

Rahul Singh Thakur
08316412812, USICT,
GGSIPU

ABSTRACT

The electronic devices has taken an important position in our lives where one performs every single task with the help of these devices. One of the Input device is mouse which helps a person to interact with the device in a way which is enabled by the motion. The project was inspired by the idea of usage of these devices by the people who are disabled and are not able to use them. Our final project moves and clicks a mouse cursor on a computer screen by tracking the user's head movements using an accelerometer. The motivation for this project came from thinking about applications of accelerometer technology. We narrowed our ideas down to head-tracking because of its potential benefit to the disabled and the lack of accurate, inexpensive head-tracking devices out there.

The other option available is of using infrared LEDs. However, there is a health risk involved there. The infra radiations have not been cleared by the health standards since there adverse effect on our eyes has not been resolved.

The project focuses on the working of the mouse using the head movement and enabling the accelerometer to provide us with the values of the two dimensional plane. The hardware includes arduino Leonardo, accelerometer and connectors. The main idea is to transmit the data of the accelerometer which is processed by arduino Leonardo which converts the accelerometer readings into cursor movement on the screen.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO EMBEDDED SYSTEMS

1. INTRODUCTION:

An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured as components of embedded systems.

Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more-complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic

lights, factory controllers, and largely complex systems like hybrid vehicles, MRI, and avionics. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

2. USES:

Embedded systems are commonly found in consumer, cooking, industrial, automotive, medical, commercial and military applications.

Telecommunications systems employ numerous embedded systems from telephone switches for the network to cell phones at the end user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics include personal digital assistants (PDAs), mp3 players, mobile phones, videogame consoles, digital cameras, DVD players, GPS receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, include embedded systems to provide flexibility, efficiency and features. Advanced HVAC systems use networked thermostats to more accurately and efficiently control temperature that can change by time of day and season. Home automation uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors — brushless DC motors, induction motors and DC motors — use electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles increasingly use embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment uses embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (PET, SPECT, CT, MRI) for non-invasive internal inspections. Embedded systems within medical equipment are often powered by industrial computers.

Embedded systems are used in transportation, fire safety, safety and security, medical applications and life critical systems, as these systems can be isolated

from hacking and thus, be more reliable. For fire safety, the systems can be designed to have greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self-sufficient and be able to deal with cut electrical and communication systems.

A new class of miniature wireless devices called motes are networked wireless sensors. Wireless sensor networking, WSN, makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensors, enabling people and companies to measure a myriad of things in the physical world and act on this information through IT monitoring and control systems. These motes are completely self-contained, and will typically run off a battery source for years before the batteries need to be changed or charged.

Embedded Wi-Fi modules provide a simple means of wirelessly enabling any device which communicates via a serial port.

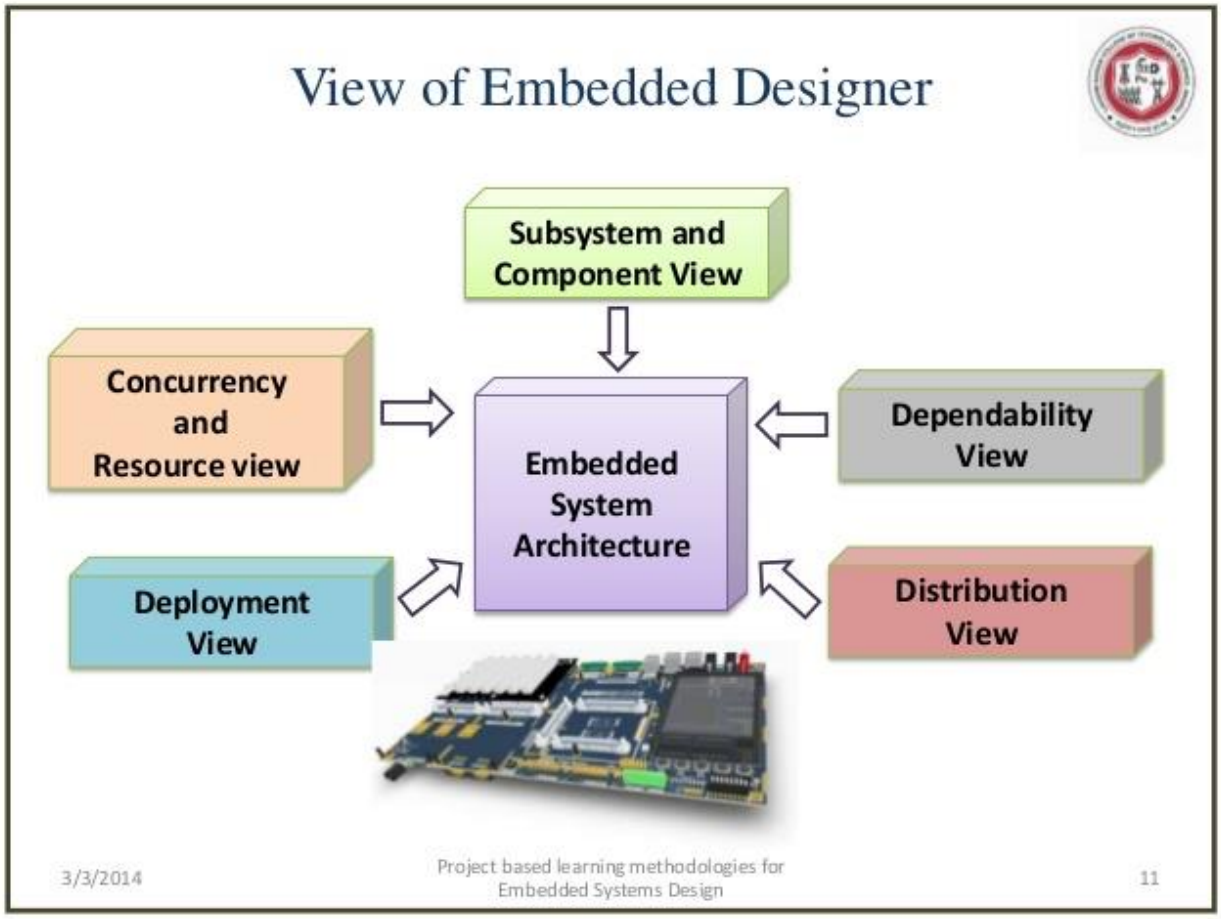


Figure 1.1: Embedded systems basic architecture

Figure 1.2: Embedded systems

CHAPTER 2

ARDUINO BOARDS

1. INTRODUCTION:

An Arduino board historically consists of an Atmel 8-, 16- or 32-bit AVR microcontroller (although since 2015 other makers' microcontrollers have been used) with complementary components that facilitate programming and incorporation into other circuits. An important aspect of the Arduino is its standard connectors, which let users connect the CPU board to a variety of interchangeable add-on modules termed *shields*. Some shields communicate with the Arduino board directly over various pins, but many shields are individually addressable via an I^2C serial bus—so many shields can be stacked and used in parallel. Before 2015, Official Arduinos had used the Atmel megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560. In 2015, units by other producers were added. A handful of other processors have also been used by Arduino compatible devices.

Most boards include a 5 V linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external programmer. This makes using an Arduino more straightforward by allowing the use of an ordinary computer

as the programmer. Currently, optiboot bootloader is the default bootloader installed on Arduino UNO.

At a conceptual level, when using the Arduino integrated development environment, all boards are programmed over a serial connection. Its implementation varies with the hardware version. Some serial Arduino boards contain a level shifter circuit to convert between RS-232logic levels and transistor–transistor logic (TTL) level signals. Current Arduino boards are programmed via Universal Serial Bus (USB), implemented using USB-to-serial adapter chips such as the FTDI FT232. Some boards, such as later-model Uno boards, substitute the FTDI chip with a separate AVR chip containing USB-to-serial firmware, which is reprogrammable via its own ICSP header. Other variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods, when used with traditional microcontroller tools instead of the Arduino IDE, standard AVR in-system programming (ISP) programming is used.

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, Duemilanove, and current Uno provide 14 digital I/O pins, six of which can produce pulse-width modulated signals, and six analog inputs, which can also be used as six digital I/O pins. These pins are on the top of the board, via female 0.1-inch (2.54 mm) headers. Several plug-in application shields are also commercially available. The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards may provide male header pins on the underside of the board that can plug into solderless breadboards.

Many Arduino-compatible and Arduino-derived boards exist. Some are functionally equivalent to an Arduino and can be used interchangeably. Many enhance the basic Arduino by adding output drivers, often for use in school-level education, to simplify making buggies and small robots. Others are electrically equivalent but change the form factor, sometimes retaining compatibility with shields, sometimes not. Some variants use different processors, of varying compatibility.

Arduino programs may be written in any programming language with a compiler that produces binary machine code. Atmel provides a development environment for their microcontrollers, AVR Studio and the newer Atmel Studio.

2. PROGRAMMING SOFTWARE:

The Arduino project provides the Arduino integrated development environment (IDE), which is a cross-platform application written in the programming language Java. It originated from the IDE for the languages *Processing* and *Wiring*. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and provides simple one-click mechanism to compile and load programs to an Arduino board. A program written with the IDE for Arduino is called a "sketch".

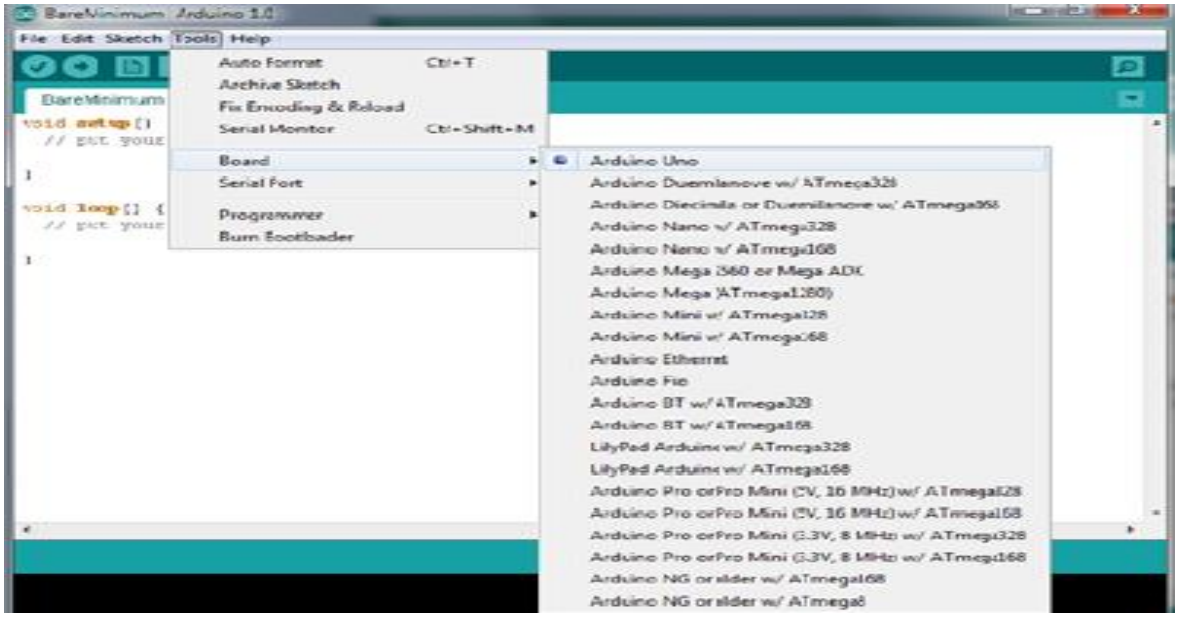
The Arduino IDE supports the languages C and C++ using special rules to organize code. The Arduino IDE supplies a software library called Wiring from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consist of two functions that are compiled and linked with a program stub *main()* into an executable cyclic executive program:

- *setup()*: a function that runs once at the start of a program and that can initialize settings.
- *loop()*: a function called repeatedly until the board powers off.

After compiling and linking with the GNU tool chain, also included with the IDE distribution, the Arduino IDE employs the program *avrdude* to convert the executable code into a text file in hexadecimal coding that is loaded into the Arduino board by a loader program in the board's firmware.

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (e.g. Flash, Processing, MaxMSP).The boards can be built by hand or purchased preassembled; the software can be downloaded for free. The hardware reference designs (CAD files) are available under an open-source license, you are free to adapt them to your needs. Arduino received an Honorary Mention in the Digital Communities section of the 2006 Ars Electronica Prix. The Arduino team is: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis.

STEP 1: Selecting a board:



STEP 2: Selecting a serial port:

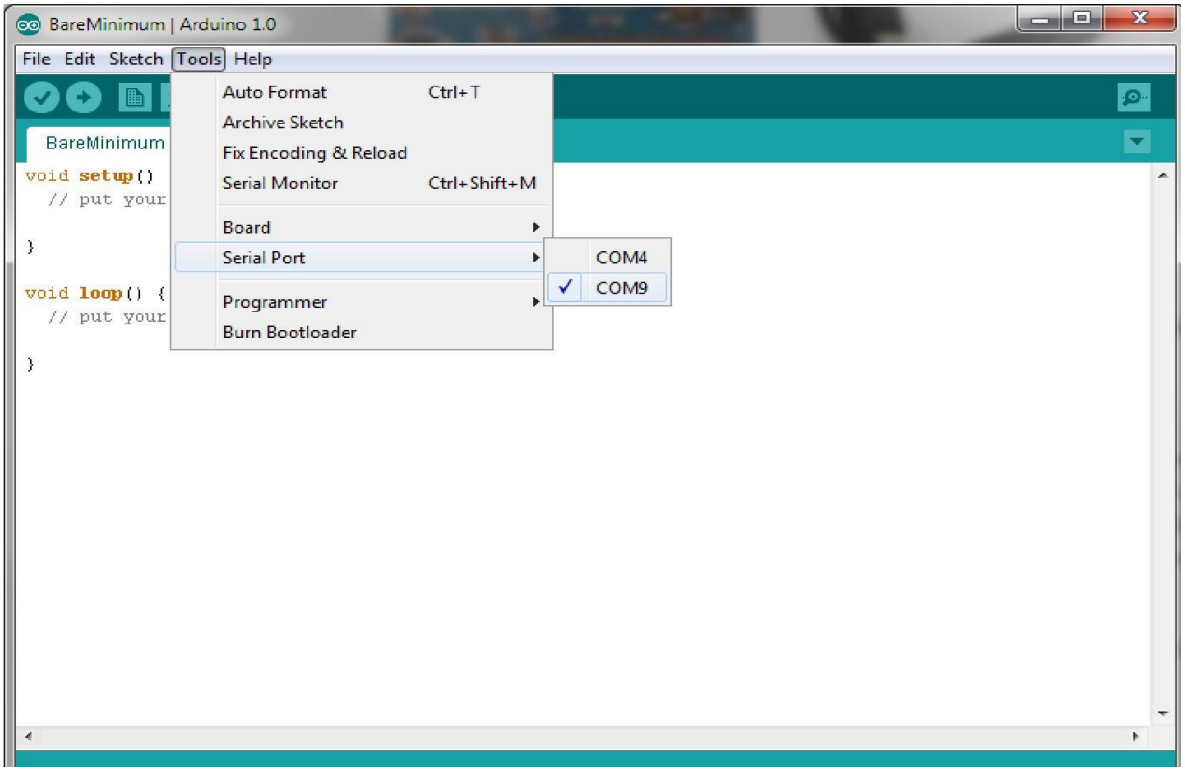


Figure 2.1: Arduino software

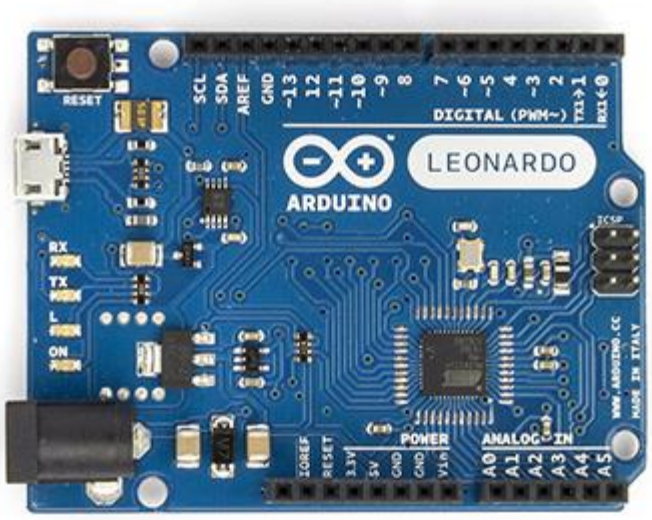


Figure 2.2: Arduino Leonardo Board front side

Figure 2.3: Arduino Leonardo Board back side

3. ARDUINO LEONARDO:

The **Arduino Leonardo** is a microcontroller board based on the ATmega32u4 (datasheet). It has 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Leonardo differs from all preceding boards in that the ATmega32u4 has built-in USB communication, eliminating the need for a secondary processor. This allows the Leonardo to appear to a connected computer as a mouse and keyboard, in addition to a virtual (CDC) serial / COM port.

3.1 Arduino Leonardo Specifications:

Microcontroller	ATmega32u4
Operating Voltage	5V
InputVoltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	20
PWM Channels	7
Analog Input Channels	12
DC Current per I/O Pin	40 mA

DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega32u4) of which 4 KB used by bootloader
SRAM	2.5 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.3 mm
Weight	20g

3.2 Power:

The Arduino Leonardo can be powered via the micro USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V. The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.
- IOREF. The voltage at which the i/o pins of the board are operating (i.e. VCC for the board). This is 5V on the Leonardo.

3.3 Memory:

The ATmega32u4 has 32 KB (with 4 KB used for the bootloader). It also has 2.5 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

3.4 Input and Output:

Each of the 20 digital i/o pins on the Leonardo can be used as an input or output, using pinMode(), digitalWrite(), anddigitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data using the ATmega32U4 hardware serial capability. Note that on

the Leonardo, the Serial class refers to USB (CDC) communication; for TTL serial on pins 0 and 1, use the Serial1 class.

- TWI: 2 (SDA) and 3 (SCL). Support TWI communication using the Wire library.
- External Interrupts: 3 (interrupt 0), 2 (interrupt 1), 0 (interrupt 2), 1 (interrupt 3) and 7 (interrupt 4). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attach Interrupt() function for details.
- PWM: 3, 5, 6, 9, 10, 11, and 13. Provide 8-bit PWM output with the analogWrite() function.
- SPI: on the ICSP header. These pins support SPI communication using the SPI library. Note that the SPI pins are not connected to any of the digital I/O pins as they are on the Uno, They are only available on the ICSP connector. This means that if you have a shield that uses SPI, but does NOT have a 6-pin ICSP connector that connects to the Leonardo's 6-pin ICSP header, the shield will not work.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- Analog Inputs: A0-A5, A6 - A11 (on digital pins 4, 6, 8, 9, 10, and 12). The Leonardo has 12 analog inputs, labeled A0 through A11, all of which can also be used as digital i/o. Pins A0-A5 appear in the same locations as on the Uno; inputs A6-A11 are on digital i/o pins 4, 6, 8, 9, 10, and 12 respectively. Each analog input provide 10 bits of resolution (i.e. 1024 different values). By default the analog inputs measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analog Reference() function.

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analog Reference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega32u4 ports.

Communication.

The Leonardo has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers.

The ATmega32U4 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). The 32U4 also allows for serial (CDC) communication over USB and appears as a virtual com port to software on the computer. The chip also acts as a full speed USB 2.0 device, using standard USB COM drivers. On Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Leonardo's digital pins.

The ATmega32U4 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

The Leonardo appears as a generic keyboard and mouse, and can be programmed to control these input devices using the Keyboard and Mouse classes.

3.5 Programming:

The Leonardo can be programmed with the Arduino software (download). Select "Arduino Leonardo from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega32U4 on the Arduino Leonardo comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the AVR109 protocol.

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header using Arduino ISP or similar; see these instructions for details.

Automatic (Software) Reset and Bootloader Initiation

Rather than requiring a physical press of the reset button before an upload, the Leonardo is designed in a way that allows it to be reset by software running on a connected computer. The reset is triggered when the Leonardo's virtual (CDC) serial / COM port is opened at 1200 baud and then closed. When this happens, the processor will reset, breaking the USB connection to the computer (meaning that the virtual serial / COM port will disappear). After the processor resets, the bootloader starts, remaining active for about 8 seconds. The bootloader can also be initiated by pressing the reset button on the Leonardo. Note that when the board first powers up, it will jump straight to the user sketch, if present, rather than initiating the bootloader.

Because of the way the Leonardo handles reset it's best to let the Arduino software try to initiate the reset before uploading, especially if you are in the habit of pressing the reset button before uploading on other boards. If the software can't reset the board you can always start the bootloader by pressing the reset button on the board.

3.6 USB Overcurrent Protection:

The Leonardo has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

3.7 Physical Characteristics:

The maximum length and width of the Leonardo PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

CHAPTER 3

ACCELEROMETER

1. INTRODUCTION:

An **accelerometer** is a device that measures proper acceleration ("g-force"). Proper acceleration is not the same as coordinate acceleration (rate of change of velocity). For example, an accelerometer at rest on the surface of the Earth will measure an acceleration g= 9.81 m/s^2 straight upwards. By contrast, accelerometers in free fall (falling toward the center of the Earth at a rate of about 9.81 m/s^2) will measure zero.

Accelerometers have multiple applications in industry and science. Highly sensitive accelerometers are components of inertial navigation systems for aircraft and missiles. Accelerometers are used to detect and monitor vibration in rotating machinery. Accelerometers are used in tablet computers and digital cameras so that images on screens are always displayed upright. Accelerometers are used in drones for flight stabilisation. Pairs of accelerometers extended over a region of space can be used to detect differences (gradients) in the proper accelerations of frames of references associated with those points. These devices are called gravity gradiometers, as they measure gradients in the gravitational field. Such pairs of accelerometers in theory may also be able to detect gravitational waves.

Single- and multi-axis models of accelerometer are available to detect magnitude and direction of the proper acceleration (or g-force), as a vector quantity, and can be used to sense orientation (because direction of weight changes), coordinate acceleration (so long as it produces g-force or a change in g-force), vibration, shock, and falling in a resistive medium (a

case where the proper acceleration changes, since it starts at zero, then increases). Micromachined accelerometers are increasingly present in portable electronic devices and video game controllers, to detect the position of the device or provide for game input.

2. PRINCIPLE:

An accelerometer measures proper acceleration, which is the acceleration it experiences relative to freefall and is the acceleration felt by people and objects. Put another way, at any point in spacetime the equivalence principle guarantees the existence of a local inertial frame, and an accelerometer measures the acceleration relative to that frame. Such accelerations are popularly measured in terms of g-force. An accelerometer at rest relative to the Earth's surface will indicate approximately 1 g *upwards*, because any point on the Earth's surface is accelerating upwards relative to the local inertial frame (the frame of a freely falling object near the surface). To obtain the acceleration due to motion with respect to the Earth, this "gravity offset" must be subtracted and corrections made for effects caused by the Earth's rotation relative to the inertial frame.

The reason for the appearance of a gravitational offset is Einstein's equivalence principle, which states that the effects of gravity on an object are indistinguishable from acceleration. When held fixed in a gravitational field by, for example, applying a ground reaction force or an equivalent upward thrust, the reference frame for an accelerometer (its own casing) accelerates upwards with respect to a free-falling reference frame. The effects of this acceleration are indistinguishable from any other acceleration experienced by the instrument, so that an accelerometer cannot detect the difference between sitting in a rocket on the launch pad, and being

in the same rocket in deep space while it uses its engines to accelerate at 1 g. For similar reasons, an accelerometer will read *zero* during any type of free fall. This includes use in a coasting spaceship in deep space far from any mass, a spaceship orbiting the Earth, an airplane in a parabolic "zero-g" arc, or any free-fall in vacuum. Another example is free-fall at a sufficiently high altitude that atmospheric effects can be neglected.

However this does not include a (non-free) fall in which air resistance produces drag forces that reduce the acceleration, until constant terminal velocity is reached. At terminal velocity the accelerometer will indicate 1 g acceleration upwards. For the same reason a skydiver, upon reaching terminal velocity, does not feel as though he or she were in "free-fall", but rather experiences a feeling similar to being supported (at 1 g) on a "bed" of up rushing air.

Acceleration is quantified in the SI unit metres per second per second (m/s^2), in the cgs unit gal (Gal), or popularly in terms of g-force (*g*). For the practical purpose of finding the acceleration of objects with respect to the Earth, such as for use in an inertial navigation system, a knowledge of local gravity is required. This can be obtained either by calibrating the device at rest, or from a known model of gravity at the approximate current position.

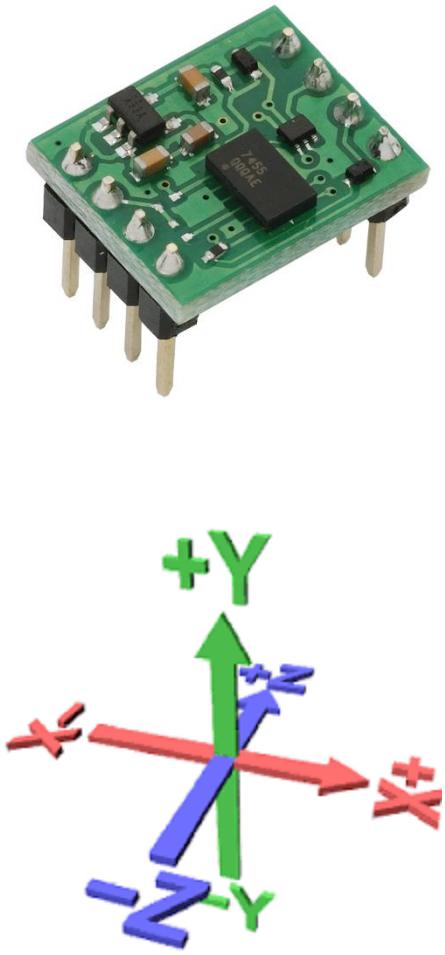


Figure 3.1: Accelerometer boards and axis indicated

CHAPTER 4

STAGES OF THE PROJECT

The project included from basic to advanced steps and contained all the different stages. The initial stage was to define the method and brainstorming which included hours of research and finding a way to start the problem. The next stage included learning about the software Avrdude and about microcontrollers which led us towards a particular type of microcontroller named as Arduino Leonardo. This microcontroller has unique functionalities and due to which the way was found for moving towards the project and the structure was studied for using the same microcontroller. The platform was also studied for the interfacing of the integrated circuit and the use of it with the external devices for the establishment of the circuit which could further help to solve the problem.

1. STAGE ONE: BRAINSTORMING

The main task of any project is to find the project and implement the idea towards a working model with knowing about the groups abilities and expansion sphere. We came up with the idea when we were searching and got fascinated by the mouse which could be made able to work without touching and moving it.

2. STAGE TWO: DEFINING PROBLEM

By searching more about the topic we found out that it is possible for engineers like us to implement that kind of thing and it will only expand our sphere and will take it to the new level. So by taking the challenge we finalized the project and started getting oriented thoughts with the idea of defining the problem and making an official step towards the success of the to project.

3. STAGE THREE: PROVIDING ALGORITHM

The next task was to get an algorithm and learn the new components involved in the project and by getting online materials and studying from the books we came to start the algorithm work and it helped us getting a concrete landmark towards the project.

4. STAGE FOUR: COMBINING CIRCUIT

The circuit includes the pin diagram of the Leonardo and making those connections on the board with the help of circuit diagrams the connections were made in that.

4.1 Frame

The very first step in this project is to find a frame which could be used for testing again and again. It should be flexible enough so that everyone can wear it without facing any difficulty. For this we used our old mechanics game connectors to make a frame.

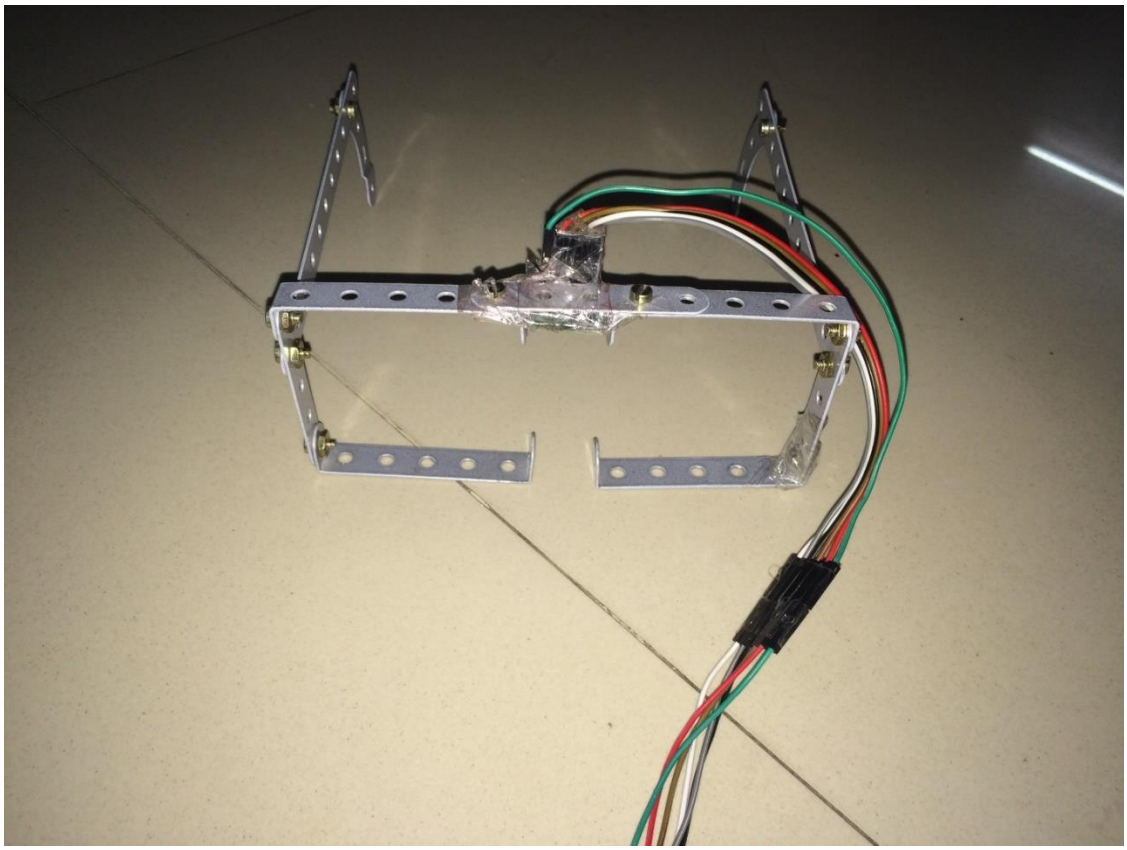


Figure 4.1: Frame with accelerometer attached at the centre

4.2 Connections with the board

The connections of the accelerometer here is as follows

Grey wire: VCC
White wire: Ground
Green wire: A0 (sensitivity pic, can / cannot be used)
Black wire: A1 (X axis)
Brown wire: A2 (Y axis)
Red wire: A3 (Z axis)

Figure 4.2: Arduino Leonardo with accelerometer connections

CHAPTER 5

PROGRESS

The progress of the project was seen from the output which was obtained as desired and the project took several turns for the fulfilment of the objective. The different stages were made from the last ones which took us to the failure but from the continuous appraisal and the motivation of the team members and the oriented approach led us to the way where we once thought we would be.

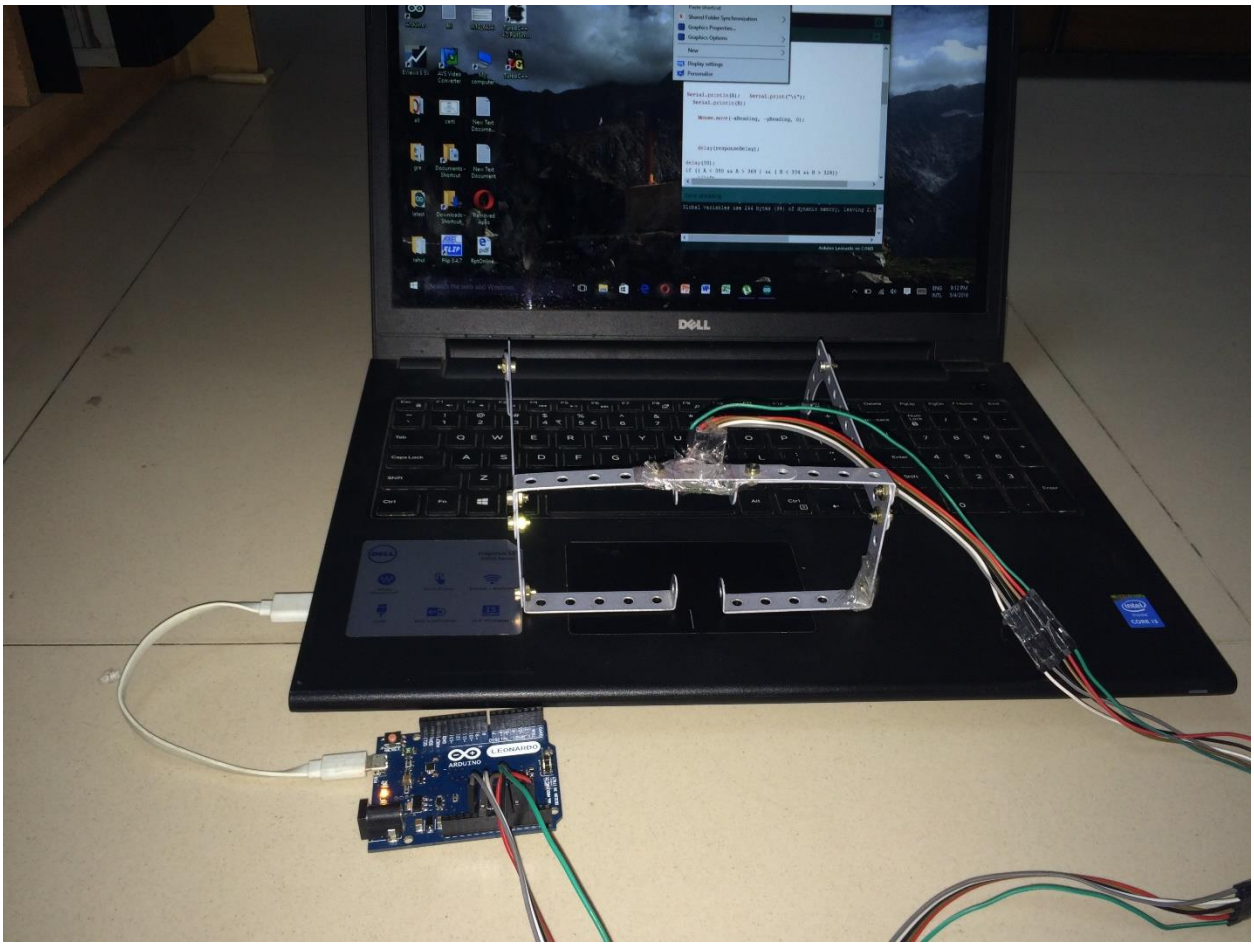


Figure 5.1: Frame attached to Arduino Leonardo
and board attached to laptop

CHAPTER 6

CONCLUSION

It was seen that by tilting our head in left direction the mouse on the laptop is moving in the left direction, same as the case with up, down and right directions. For the left and right clicks the user head to look in the left and direction of the screen. By looking at the extreme left direction of the screen, left click is operated and same as the case with the right click.

With this project the differently challenged people can easily operate a laptop just by tilting their head. Left and right clicks can be further made easier with switches for individual clicks.

“Calibration is the key of this project” and we have made a program which will automatically calibrate the position of user’s head and accordingly the left and right tilt will work. It is easier to use and everyone can use this as the frame is so flexible.

CHAPTER 7

REFERENCES

- https://blogs.msdn.microsoft.com/msgulfcommunity/2013/05/16/hand-gestures-to-control-windows-mouse-cursor-kinect-for-windows/
- http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6272668&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6272668
- http://technofall.com/control-mouse-pointer-using-gestures/
- https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2008/xh33_jdm55_ww239/xh33_jdm55_ww239/
- http://kinesicmouse.xcessity.at

```
//CODE

/*
 //Details
 Accelerometer_Mouse
 This code reads the Memsic 2125 two-axis accelerometer, converts the pulses output by the
2125 into milli-g's (1/1000 of earth's gravity) and prints a mapped version of them over the
serial connection to the computer.output. It then takes these mapped values and uses them for
virtual mouse movement, clicking, and scrolling.
 The circuit:
 X output of accelerometer to digital pin 2
 Y output of accelerometer to digital pin 3
 +V of accelerometer to +5V
 GND of accelerometer to ground
 created by David Kerns of The Athenian School on Dec 3 2012, with a large portion of code
taken from David A. Mellis' and Tom Igoe's found here:
http://www.arduino.cc/en/Tutorial/Memsic2125
*/
#include<Mouse.h>
const int xPin = A1;   // const for X output of the accelerometer
const int yPin = A2;  // const for Y output of the accelerometer
void setup()
 {
 Serial.begin(9600); // initialize serial communications:
 pinMode(xPin, INPUT); // xPin connected to the accelerometer is input pinMode(yPin, INPUT);
 // yPin connected to the accelerometer is input
 Mouse.begin();
 // Serial.begin(9600); // Set up serial communication at 9600bps
 Serial.println("ready"); //Print ready once setup loop is finished
}
void loop() {
 int pulseX, pulseY; // Variables to read the pulse widths from accelerometer:
 int accelerationX, accelerationY; // Variables to contain the resulting accelerations
 //  pulseX = pulseIn(xPin, HIGH); // read pulse from x-axes
 //  pulseY = pulseIn(yPin, HIGH); // read pulse from y-axes
 // convert the pulse width into acceleration accelerationX and accelerationY are in milli-g's:
earth's gravity is 1000 milli-g's, or 1g.
 //  accelerationX = ((pulseX / 10) - 500)/2 ;
 //  accelerationY = ((pulseY / 10) - 500)/2 ;
 accelerationX = analogRead(xPin);
 accelerationY = analogRead(yPin);
 accelerationX = map(accelerationX, 240, 440, -20, 20); //map the acceleration output to more
manageable values
 //            Average of min and max value of xAxis should be threshold or value at reference
 //            select the mapping value according to the sensitivity. Higher range => High sensitivity
 //             Use an external input reference (around 2v) as the values are between 200-400. It
will increase the sensitivity.
 accelerationY = map(accelerationY, 170, 370, -20, 20); //map the acceleration output to more
manageable values
//             Average of min and max value of yAxis should be threshold or value at reference
 Serial.print(accelerationX); // print the x acceleration result
```

```
  Serial.print("\t"); // print a tab character:
  Serial.print(analogRead(xPin));
  Serial.print("\t");
  Serial.print(accelerationY); // print the y acceleration result
  Serial.print("\t");
  Serial.print(analogRead(yPin));
  Serial.println(); //print a new line
  delay(500);
  if (accelerationX < -17) { // override loop for scrolling, if accelerationX < -17
    Mouse.move(0, 0, accelerationY); // mouse scroll function at the rate of accelerationY
  }
  else { // when accelerationX is > -17 (most of the time), do this loop
    if (accelerationY > 4 || accelerationY < -4) /*if accelerationY > 4 or < -4 || accelerationX < -4))
// or if accelerationY > 4 and < -4(accelerationX > 4)*/
    {
      Mouse.move(-accelerationX, -accelerationY, 0); // move the mouse the negative
accelerations(X and Y), do not scroll
    }
  }
  if (accelerationY > 20) { //if the accelerationY value reaches a value of 20,
    Mouse.click(); //left click the computer's mouse
  }
  if (accelerationY > 35) { //if the accelerationY value reaches a value of 35,
    Mouse.click(); //left click once
    Mouse.click(); //left click again (resulting in a double click)
  }
  if (accelerationX > 20) { //if the accelerationX value reaches a value of 20,
    Mouse.click(MOUSE_RIGHT); //right click the computer's mouse
  }
}
```