بسم الله الرحمن الرحيم

Sudan University of Science and Technology

Faculty of Engineering

Aeronautical Engineering Department

Synthesize of Design of Inertial navigation system (INS) using MEMS sensor

—Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science. (BSc Honor)

By:

1. Joireya Balla Abd-elfattah
2. Khidir Ahmed Khidir
3. Mohammed Hashim Hassan
4. Mohammed Talaat TagElsir

Supervised by:

Dr. Omer Abd-Elrazag

الاية

قال تعالى:

{أَفَمَنْ أَسَّسَ بُنْيَانَهُ عَلَى تَقْوَى مِنَ اللَّهِ وَرِضْوَانٍ خَيْرٌ أَمْ مَنْ أَسَّسَ بُنْيَانَهُ عَلَى شَفَا جُرُفٍ هَارٍ فَانْهَارَ بِهِ فِي نَارِ جَهَنَّمَ وَاللَّهُ لَا يَهْدِي الْقَوْمَ الظَّالِمِينَ}

صدق الله العظيم

سورة التوبة الاية (109)

Abstract

The purpose of this project is mainly to convey an elementary inertial navigation system, and take advantage of the capabilities that the Micro-Electromechanical system (MEMS) affords. The main aspects cover the simplicity, intuition, and fairness in the design of inertial navigation system. A noticeable tendency of cutting-off the cost of materials and facilities that are used to design any system in general is in demand which stands no-distance from the intention behind designating this system. Thus, using MEMS sensor and Programmable microcontroller to emulate the inertial navigation system is quite affordable and stands for the expected performance. The information yielded from the sensor as real-time evaluation of the momentarily motion and deviation of the airplane is illustrated on a Liquid crystal display (LCD), as a fair and neat technique of showing upshot digitally. The outcome of the work achieved turns to stand up as simplified system for the sake of understanding and building of affordable and yet reliable system, although main components are not fully supported and a lot of restriction is applied, the final project was carried down but in longer time. Furthermore, the need of well understanding and studying of the inertial navigation system is considerable in order to get more familiarize with the Inertial navigation system.

التجريد

الغرض الاساسي من هذا المشروع هو التعبير عن نظام الملاحة بالقصور الذاتي البسيط والاستفادة من المميزات التي يقدمها نظام الكهروميكانيكيات الدقيقة. الجوانب التي يغطيها المشروع شملت البساطة والبديهية في تصميم نظام الملاحة بالقصور الذاتي. من الملاحظ ان هنالك ميول تجاه تقليل تكلفة المواد والتصنيع التي تستخدم لتصميم أي نظام بشكل عام ونظام الملاحة لا يستثني هذا الميول لذلك استخدام تقنيات الكهروميكيانيات الدقيقة و متحكم دقيق قابل للبرمجة من أجل محاكاة حركة والتفاف الطائرة،ولقد توافقت هذه المكونات من حيث السعر المناسب والداء المتوقع منها. البيانات التي تم التحصل عليها من الحساس هي قياس لحظي لحركة الطائرة، وتم عرضها في شاشة عرض كريستالي ،بالرغم من ان المكونات لم تتوفر بسهولة وأيضا عدم الدعم الكافي من الجهة المصنعة مما ادى الى استغراق المزيد من الوقت ، لكن النتيجة الأخيرة للمشروع كانت مرضية ،بالاضافة الى انه من اجل فهم نظام الملاحة بصورة أدق ، يجب دراسة الملاحة الجوية المعتمدة على القصور الذاتي بتفصيل أعمق.

ACKNOWLEDGMENT

First and above all, we praise Allah, the almighty for providing us this opportunity and granting us the capability to proceed successfully. This thesis appears in its current form due to the assistance and guidance of several people. I would therefore like to offer my sincere thanks to all of them.

Dedication

In the memory of Ahmed Al-Bukhari

You left fingerprints in our life, must not be forgotten...

Contents

الاية ____ I

Abstract ____ II

التجريد ____ III

ACKNOWLEDGMENT ____ IV

Dedication ____ V

List of Figures ____ IX

List of Table ____ XI

Abbreviation ____ XII

Symbols ____ XIII

Chapter One: Introduction ____ 1

1.1 Overview 1

1.2 Aim and objectives 2

1.2.1 Aim 2

1.2.2 Objectives 2

1.3 Problem statement 2

1.4 Proposed solution 2

1.5 Methodology 3

1.6 Thesis Outline 3

Chapter Two: Inertial Navigation System and MEMS sensor ____ 4

2.1 Introduction 4

2.2 Navigation methods 8

2.2.1 Pilotage 9

2.2.2 Dead reckoning 9

2.2.3 Celestial navigation 9

2.2.4 Radio navigation 10

2.2.5 Inertial navigation 11
2.3 Inertial navigation system INS 12
2.4 Basic principles of inertial navigation 13
2.4.1 An accelerometer 15
2.4.2 A gyroscope 16
2.4.3 Gimballed Systems 17
2.4.3.1Gimballed INS work.................... 19
2.4.4 Strap down systems 19
2.5 Disadvantages of INS 21
2.6 Micro-Electro-Mechanical Systems (MEMS) 22
Chapter Three: Proposed Inertial Navigation System using MPU6050 ____________24
3.1 Introduction 24
3.2 Arduino Uno 24
3.2.1 Programming 26
3.2.2 Warnings 26
3.2.3 Power 27
3.2.4 Memory 27
3.2.5 Input and Output 27
3.2.6 Interrupts 28
3.2.7 MPU (Microprocessor Unit) 28
3.3 I^2C 29
3.4 GY521 30
3.5 Processing 32
3.5.1 Toxic library 33
3.6 Step 1: Wire circuit 35
3.7 Step 2: Install I2Cdev & MPU6050 Libraries 35
3.8 Step 3: Install Latest Version of Processing & ToxicLibs Library 37

3.9 Step 4: Run the simulation 38
3.10 LCD (16 x2): 39
3.11 Variable Resistor 40
3.12 Another simulation by using LCD 41
3.12.1 Step 1: Connection Pin LCD (16*2) with Arduino Uno 41
3.12.2 Step 2: Download the Liquid Crystal Library 42
Chapter Four: Results and Discussion ____________________46
4.1 Results 46
Chapter Five: Conclusion and Recommendation ____________________51
5.1 Conclusion 51
5.2 Recommendation and Future work 51
Reference ____________________52

List of Figures

Figure (1): old Chinese kites 5
Figure (2): Ornithopter 6
Figure (3): a sextant...10
Figure (4): GPS satellite 11
Figure (5): inertial navigation system...13
Figure(6):abasic inertial navigation system 13
Figure (7): The XYZ frame is the inertial frame ECEF NWU frame is the local navigational frame, where the axes are north (N), west (W), and up (U). 14
Figure (8): aircraft reference frame 14
Figure 9 Accelerometer 15
Figure (10): Gyroscope...16
Figure (11): (a) (b) Gimballed inertial system parts 18
Figure (12): Strap-down system...20
Figure (13): MEMS components 23
Figure (14):Arduino Pins 25
Figure (15):Arduino IDE 26
Figure (16): MPU6050 29
Figure (17): I2C Bus works 30
Figure (18): the schematic of the GY-521 break-out board for the MPU6050 chip....31
Figure (19): GY-521...32
Figure (20): Sketch of the Processing software 33
Figure (21): Teapot in levelled position 34
Figure (22):Arduino Uno connected with GY-521 35
Figure (23): Total Pitch, Roll, and Yaw angles readings 37
Figure (24): Simulation of the Teapot 38
Figure (25):LCD (Liquid Crystal Display) 39
Figure (26): Variable resistor..40
Figure (27): LCD (16*2) and Arduino Uno Connection.................................41
Figure (28):Calibration of MPU6050 GY-521 46
Figure(29): Fetching the motion in serial monitor 47
Figure (30):Parallel Display of Both LCD & Serial monitor 48

Figure(31):calibration of GY-521 motion 49
Figure(32):   Simulation of GY-521 Through Teapot..49
Figure (33): rotation motion about lateral axis 50

List of Table

Table 1 Description of GY-521 pins 31
Table 2 Pins connection between Arduino & GY-521...32
Table 3 Pins connection between Arduino & LCD..41

Abbreviation

ADF	Automatic Direction Finder
DMP	Digital Motion Processor
DOF	Degrees of Freedom
GPS	Global positioning system
IC	Integrated circuit
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IFR	Instrument Flight Rule
ILS	Instrument landing systems
IMU	Inertial Measurement Unit
INS	Inertial Navigation system.
LCD	Liquid Crystal Display
MEMS	Micro Electronic Mechanical System.
MPU	Microprocessor Unit
NAVAIDS	Navigation Aids
NDB	Non-Directional Radio Beacon
VFR	Visual Flight Rule
VOR	Very High Frequency Omi-Directional Range

Symbols

$\emptyset$	Roll angle
ρ	Pitch angle
θ	Yaw angle

Chapter One: Introduction

1.1 Overview

Finding the way from one place to another is called (NAVIGATION), moving of an aircraft from one point to another is the most important part for any kind of mission, there was five basic forms of navigation are Pilotage, which essentially relies on recognizing landmarks to know where you are. It is older than human kind, Dead reckoning, which relies on knowing where you started from, plus some form of heading information and some estimation of speed, Celestial navigation using time and the angles between local vertical and known celestial objects (e.g., sun, moon, or stars), Radio navigation which relies on radio-frequency sources with known locations (including Global Positioning System satellites), finally Inertial navigation which relies on knowing your initial position, velocity, and attitude and thereafter measuring your attitude rates and accelerations, it is the only form of navigation that does not rely on external references.

Inertial navigation is used in a wide range of applications including the navigation of aircraft, recent advances in the construction of MEMS devices have made it possible to manufacture small and light inertial navigation systems.

An inertial measurement unit (IMU), this sensor, coupled with the proper mathematical background is capable of detecting accelerations and angular velocities and then transforming those into the current position and orientation of the system.

The first type of INS developed was a gimballed system. the accelerometers are mounted on a motorized gimballed platform which was always kept aligned with the navigation frame, Pickups are located on the outer and inner gimbals which keep track of the attitude of the stabilized platform relative to the vehicle on which the INS is mounted, this setup has several detractors which make it undesirable, as Bearings are not frictionless, Motors are not perfect.

Consumers power to keep the platform aligned with the navigational frame which is not always good on an embedded system, cost is high due to the need for high quality motors slip rings, bearings and other mechanical parts.

Recalibration is difficult, and requires regular maintenance by certified personnel which could be difficult on an autonomous vehicle.

A strap-down system is a major hardware simplification of the old gimballed systems. The accelerometers and gyros are mounted in body coordinates and are not mechanically moved, this method overcomes the problems encountered with the gimballed system, and most importantly reduces the size, cost, power consumption, and complexity of the system, the major disadvantage is a substantial increase in computing complexity.

1.2 Aim and objectives

1.2.1 Aim

To augment the safety of the aircraft by interface microelectromechanical system chip with inertial navigation system and simulate the motion of aircraft.

1.2.2 Objectives

- Study of microelectromechanical system.
- Select a suitable method for simulating the motion of the aircraft.
- Display the outputs of system into liquid crystal display (LCD).

1.3 Problem statement

The existence of moving parts of the gyroscopes, contain friction between these parts, produce errors, also cost is high due to the need for high quality motors slip rings, bearings and other mechanical parts.

In addition to the complexity of conventional inertial navigation system in terms of construction and operations, these different elements result considerable problems, complexity and difficult of computing operations.

1.4 Proposed solution

Avoiding the use of moving parts and reducing the cost by using the microprocessor unit (MPU6050) sensor and Arduino microcontroller, and using certain codes by computers to carry out operations of computing to reduce the complexity.

1.5 Methodology

The methodology will have used in the project contain of theoretical, analytical, and simulation methods.

study of the inertial navigation system and MEMS were considered first to be able to dive in the understanding of inertial navigation system operation

The design procedures for the microelectromechanical system MEMS with inertial navigation system INS will be applied through the process:

1. Adoption of the Arduino Uno and MPU6050 as microcontroller and sensing device respectively.
2. Uploading the programming code using the Arduino IDE.
3. Manipulation and illustration of the retrieved evaluations in an LCD.
4. Connection Arduino Uno and MPU6050 and calibration of motion of aircraft about three axes in chapter three and four.
5. Programmed Arduino Uno with the (Arduino Software (IDE)).
6. Processing MPU6050 with Arduino Uno used Processing sketch (MPUTeapot.pde), install Latest Version of Processing & ToxicLibs Library.
7. Install I2C and MPU6050 Libraries
8. Run the simulation, and at last Connection Pin LCD with Arduino Uno.
9. Evaluate, Analysis and discussion of the results.

1.6 Thesis Outline

Chapter one: introduction

Chapter two: Literature Review.

Chapter three: proposed inertial navigation system using MPU6050.

Chapter four: Result and discussion.

Chapter five: Recommendation and conclusion.

Appendices

Chapter Two: Inertial Navigation System and MEMS sensor

2.1 Introduction

From the earliest times, people have moved from one place to another by finding or 'knowing' their way.

They crossed parts and rivers generally using landmarks, that is, navigation by observation also used their understanding of celestial bodies, used sun and moon, these techniques can only be used in clear weather conditions, this method of navigation relies on the observation and recognition of known features or fixed objects in our surroundings and moving between them. In technical narratives, the locations of these features are often referred to as 'way-points'. an extension of this process is navigation by following a map in this case, the navigator will determine his or her position by observation of geographical features such as roads, rivers, hills and valleys which are shown on the map. these features may be defined on the map with respect to a grid system or 'reference frame' the use of reference frames is fundamental to the process of navigation.

An ancient and well-established technique is to take sightings of certain of the fixed stars to which the navigator can relate his or her position. The fixed stars effectively define a reference frame which is fixed in space is commonly referred to as an 'inertial' reference frame. Navigation systems of this type, which rely upon observation of the outside world, are known as 'position fixing' systems.

finding the way from one place to another is called NAVIGATION, moving of an aircraft from one point to another is the most important part for any kind of mission. Plotting on the paper or on the map a course towards a specific area of the earth, in the past, used to be a task assigned to a specialized member of the aircraft's crew such a navigator, such a task was quite complicated and not always accurate. Since it depended on the observation, using simple maps and geometrical instruments for calculations, today aerial navigation has become an art which nears to perfection.

Both external Navaids (Navigational Aids) and on-board systems help navigate any aircraft over thousands of miles with such accuracy that could only be imagined a few decades ago.

The first reference to the air traffic was during the hieroglyphic inscriptions in ancient Egypt, where they found fees describe the flight of birds, there are also signs of ancient China, where people are using kites for thousands of years.

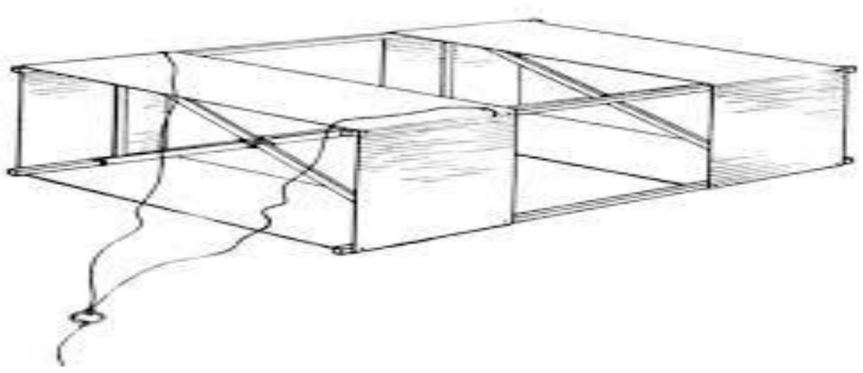


Figure (1): old Chinese kites

It was not far from the Middle Ages that of Muslim Scholars, which absorbed the mechanical correct the flight of birds. Before the start of scientific research Aeronautics, people had already begun to think of ways to fly.

In the Greek era, making Icarus and his father Daadalus wings of feathers and glue and flew them away from their imprisonment, when people began to learn how to fly Air began to understand the basics and dynamic.

One of the oldest scientists who have studied the foundations of air navigation was Abbas ibn Firnas where he learned the dynamics of flight and work on it some experiments, and that was in Cordoba in Andalusia in the eighth century, the early Europeans scientists who have studied aeronautics was Roger Bacon and Leonardo da Vinci. Fdavenci discuss how to fly when birds and drawing geometric diagrams of what is known (ornithopter) as shown in figure (2). is the oldest flying machine in the late fifteenth century, it was a failed design practice, either because the tippers were small compared to the size flying machine to create enough lift force or a heavy person can agitate, those Aloornotpetr have followed with interest the amateur until they set up the so-called plane sailing the late 19th century. [1]

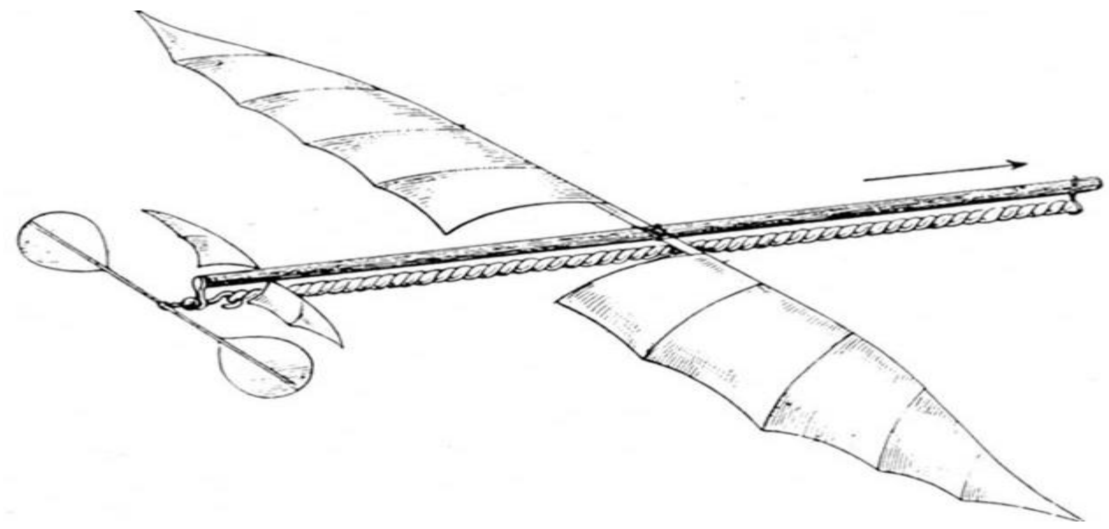


Figure (2): Ornithopter

the earliest applications were on land, then as the desire developed to explore farther afield, instruments were developed for marine applications, more recently, there have been significant developments in inertial sensors, and systems for inertial navigation on land, in the air, on or under the oceans as well as in space to the planets and beyond.

During the thirteenth century, the Chinese discovered the properties of lodestone and applied the principles of magnetism to fabricate a compass.

They used this instrument to navigate successfully across the south China Sea, this device could be used irrespective of visibility but was difficult to use in rough weather. The other significant development to help the long distance traveller was the sextant, which enabled position fixes to be made accurately on land.

In the seventeenth century, Sir Isaac Newton defined the laws of mechanics and gravitation, which are the fundamental principles on which inertial navigation is based.

Despite this, it was to be about another two centuries before the inertial sensors were developed that would enable the demonstration of inertial navigation techniques. However, in the early eighteenth century, there were several significant developments; Serson demonstrating a stabilized sextant and Harrison devising an accurate chronometer, the former development enabling sightings to be taken of celestial objects without reference to the horizon and the latter enabling an accurate determination of longitude.

These instruments, when used with charts and reference tables of location of celestial bodies, enabled accurate navigation to be achieved, provided the objects were visible. Foucault is generally credited with the discovery of the gyroscopic effect in 1852. He was certainly the first to use the word, there were others, such as Bohneberger, Johnson and Lemarle, developing similar instruments.

All of these people were investigating the rotational motion of the Earth and the demonstration of rotational dynamics, they were using the ability of the spin axis of a rotating disc to remain fixed in space, later in the nineteenth century, many fine gyroscopic instruments were made, in addition there were various ingenious applications of the gyroscopic principle in heavy equipment such as the grinding mill.

A significant discovery was made in 1890 by Professor G.H. Bryan concerning the ringing of hollow cylinders, a phenomenon later applied to solid-state gyroscopes.

The early years of the twentieth century saw the development of the gyrocompass for the provision of a directional reference, the basic principle of this instrument is the indication of true north by establishing the equilibrium between the effect of its pendulous and the angular momentum of the rotating base carrying the compass. Initially, this instrument was sensitive to acceleration

Elmer and Lawrence Sperry improved the design of the gyrocompass with further refinements by Brown and Perry. these instruments provided the first steps towards all-weather, autonomous navigation, the Sperry brothers were also at the forefront of the application of the gyroscopic effect to control and guidance in the early twentieth century, they produced navigation and autopilot equipment for use in aircraft and gyroscopes for use in torpedoes, rate of turn indicators, artificial horizons and directional gyroscopes for aircraft were being produced in the 1920s.

There was significant progress during the early part of the twentieth century with the development of stable platforms for fire control systems for guns on ships and the identification of the concept for an inertial navigation system.

However, at this stage, the quality of the inertial sensors was not suitable for the production and demonstration of such a system, there was much activity in various pans of the world devising new types of inertial sensors, improving their accuracy and, in 1949, the first publication suggesting the concept of the strap down technique for navigation.

The pace of development and innovation quickened in the 1950s with many significant developments for seaborne and airborne applications. more accurate sensors were produced, with the accuracy of the gyroscope being increased substantially.

It was also during the 1950s that the principle of force-feedback was applied to the proof mass in an accelerometer to produce an accurate acceleration sensing instrument. The early part of the 1950s saw the fabrication of a stabilized platform inertial navigation system followed by the first crossing of the United States of America by an aircraft using full inertial navigation. Inertial navigation systems became standard equipment in military aircraft, ships and submarines during the 1960s, all of these applications using the so-called stable platform technology.

Major projects of this period in which inertial system technology was applied were the ballistic missile programmers and the exploration of space, similar progress has taken place in the last two decades; one major advance being the application of the micro-computer and development of gyroscopes with large dynamic ranges enabling the strap down principle to be realized, this has enabled the size and complexity of the inertial navigation system to be reduced significantly for very many applications, the use of novel methods has enabled small, reliable, rugged and accurate inertial sensors to be produced that are relatively inexpensive, thus enabling a very wide range of diverse applications as discussed below, this period has also seen significant advances in the development of solid-state sensors such as optical fibre gyroscopes and silicon accelerometers.[2]

2.2 Navigation methods

Air navigation is accomplished by various methods, the method or system that a pilot uses for navigating through today's airspace system will depend on the type of flight that will occur (VFR or IFR), which navigation systems are installed on the aircraft, and which navigation systems are available in a certain area.

- **VFR** (visual flight rule) a pilot will navigate using "dead rocking" combined with visual observation (known as pilotage) with reference to appropriate maps, this may supplement using radio navigation aids.
- **IFR** (instrumental flight rule), the pilot relies only on the airborne instruments, that due to (weather, night, altitude etc.)

According to this method, there were five basic forms of navigation are as follows:

2.2.1 Pilotage

which essentially relies on recognizing landmarks to know where you are, it is older than human kind, is a term that refers to the sole use of visual ground references.

the pilot identifies landmarks, such as rivers, towns, airports, and buildings and navigates among them, the trouble with pilotage is that often times.

references aren't easily seen and can't be easily identified in low visibility conditions or if the pilot gets off track even slightly.

2.2.2 Dead reckoning

which relies on knowing where you started from, plus some form of heading information and some estimate of speed, involves the use of visual checkpoints along with time and distance calculations. the pilot chooses checkpoints that are easily seen from the air and also identified on the map, and then calculates the time it will take to fly from one point to the next based on distance, airspeed, and wind calculations.

A flight computer aids pilots in computing the time and distance calculations, and the pilot typically uses a flight planning log to keep track of the calculations during flight.

2.2.3 Celestial navigation

It's one of the oldest forms of navigation, and one of the first navigation aids used by transport aircraft. celestial navigators use a device called a sextant to determine the angle between a known star and the horizon, by using the angle, plus the time it was measured, using time and the angles between local vertical and known celestial objects (e.g., sun, moon, or stars).

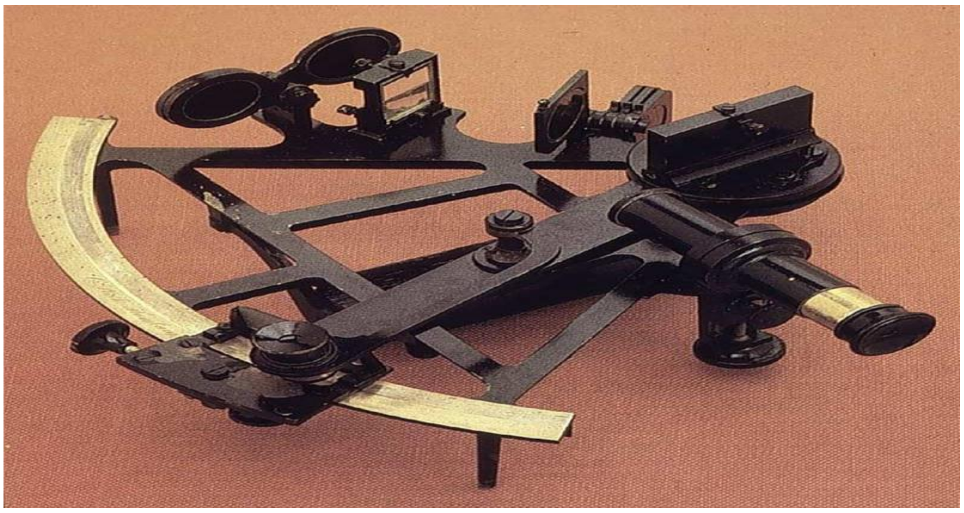


Figure (3): a sextant

2.2.4 Radio navigation

which relies on radio-frequency sources with known locations (including Global Positioning System satellites), aircraft equipped with radio navigation aids (NAVAIDS), pilots can navigate more accurately than with dead reckoning alone, radio NAVAIDS come in handy in low visibility conditions and act as a suitable back-up method for general aviation pilots that prefer dead reckoning, they are also more precise.

There are different types of radio NAVAIDS used in aviation:

- **ADF/NDB**

An **NDB** is a no directional radio beacon is medium frequency navigational aid which transmits non-directional signals, superimposed with a Morse code identifier and received by an aircraft's ADF that is stationed on the ground and emits an electrical signal in all directions.

- **ADF** Automatic Direction Finder is an aircraft radio navigation which senses and indicates the direction to a Low/Medium Frequency non-directional radio beacon (NDB) ground transmitter.
- **GPS** (Global Positioning System)**,** navigation system based on the transmission of signals from satellites provided and maintained by the United States of America and available to civil aviation users.

The global positioning system has become the most valuable method of navigation in the modern aviation world. GPS is probably the most common NAVAID in use today, it provides precise location data, such as aircraft position, track, speed, and to pilots, it has become a preferred method of navigating due to the accuracy and ease of use.

Figure (4): GPS satellite

- **VOR** (VHF Omnidirectional Range), VORs were first used in the 1940s, and they're still one of the most common radio navigation system in the US.

is a radio-based NAVAID that operates in the very-high-frequency range.

- **ILS** an instrument landing system (ILS) is an instrument approach system used to guide aircraft down to the runway from the approach phase of flight. It uses both horizontal and vertical radio signals emitted from a point along the runway. These signals intercept to give the pilot precise location information in the form of a glideslope -- a constant-angle, stabilized descent path all the way down to the approach end of the runway. ILS systems are widely in use today as one of the most accurate approach systems available.

2.2.5 Inertial navigation

which relies on knowing your initial position, velocity, and attitude and thereafter measuring your attitude rates and accelerations, it is the only form of navigation that does not rely on external references.

2.3 Inertial navigation system INS

Inertial navigation has had a relatively short but intense history of development, much of it during the half-century of the Cold War, with contributions from thousands of engineers and scientists.

INS systems use a series of accelerometers and gyroscopes to determine their position. In the 1960s, INS reached widespread usage in civilian and military aircraft for worldwide navigation.

use of inertial sensors and inertial navigation has developed rapidly in the recent past, owing to a number of very significant technological advances, the rapid development of micro electromechanical sensors and superior computer performance has provided the stimulus for many new applications.

Methods of measuring position and velocity are just as numerous, they include fixed land references such as beacons, devices measuring motion relative to a fixed medium such as the ground, atmosphere, or earth’s magnetic field, and satellite navigation systems like GPS (global positioning system).

However, varied all these methods may be, they all resemble each other in that they involve measurement with respect to a reference with known position and velocity. On one hand, this is part of the nature of the problem, since position is by definition relative.

However, there are numerous cases where a moving object’s initial position is known, but its subsequent motions cannot be conveniently tracked with respect to a reference. In these cases, an inertial navigation system is used.

Aircraft are perhaps the most important application today, because they often travel in conditions of low visibility and must be able to maintain level flight without ground references, even in cases where a reference is normally available, it might be necessary to use an inertial navigation system in case of a momentary outage.

The operation of inertial navigation systems depends upon the laws of classical mechanics as formulated by Sir Isaac Newton. Newton's laws tell us that the motion of a body will continue uniformly in a straight line unless disturbed by an external force acting on the body.

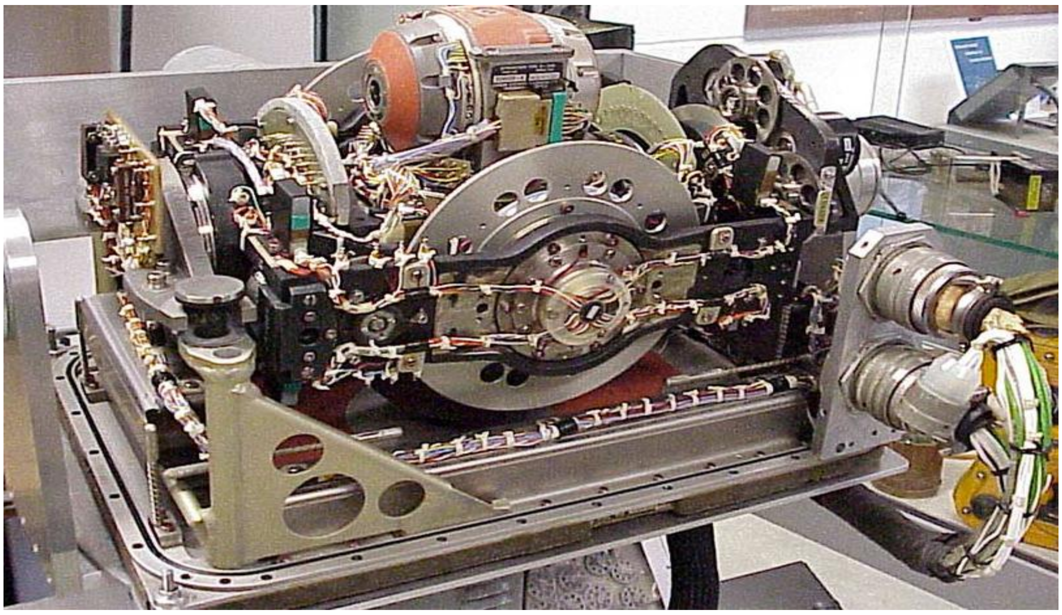
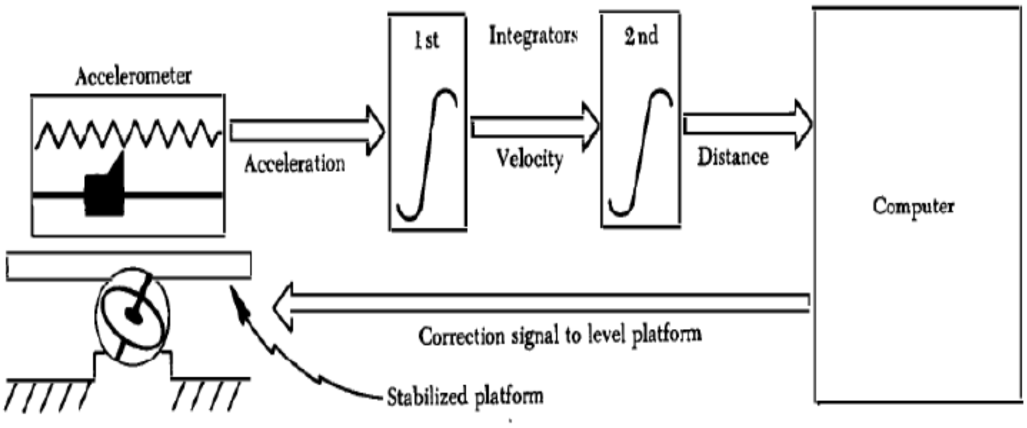


Figure (5): inertial navigation system

2.4 Basic principles of inertial navigation

Inertia is the propensity of bodies to maintain constant translational and rotational velocity, unless disturbed by forces or torques, respectively (Newton's first law of motion).



Figure(6):abasic inertial navigation system

An inertial reference frame is a coordinate frame in which Newton's laws of motion is valid, inertial reference frames are neither rotating nor accelerating.

Inertial navigation uses several reference frames, which are shown in Fig (7) and (8).

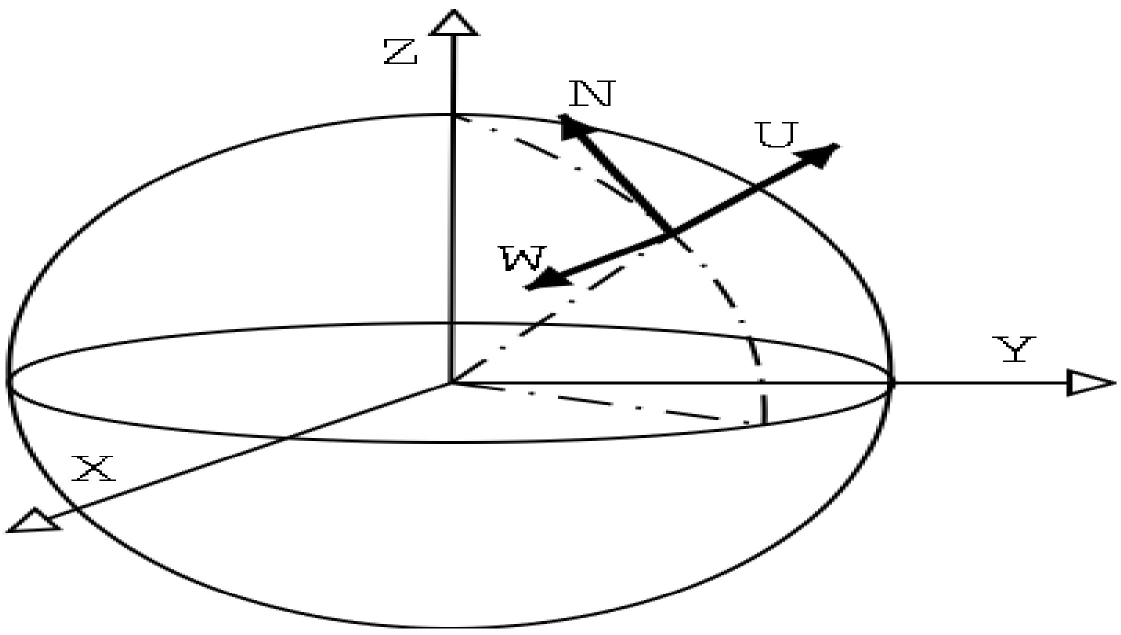


Figure (7): The XYZ frame is the inertial frame ECEF NWU frame is the local navigational frame, where the axes are north (N), west (W), and up (U).

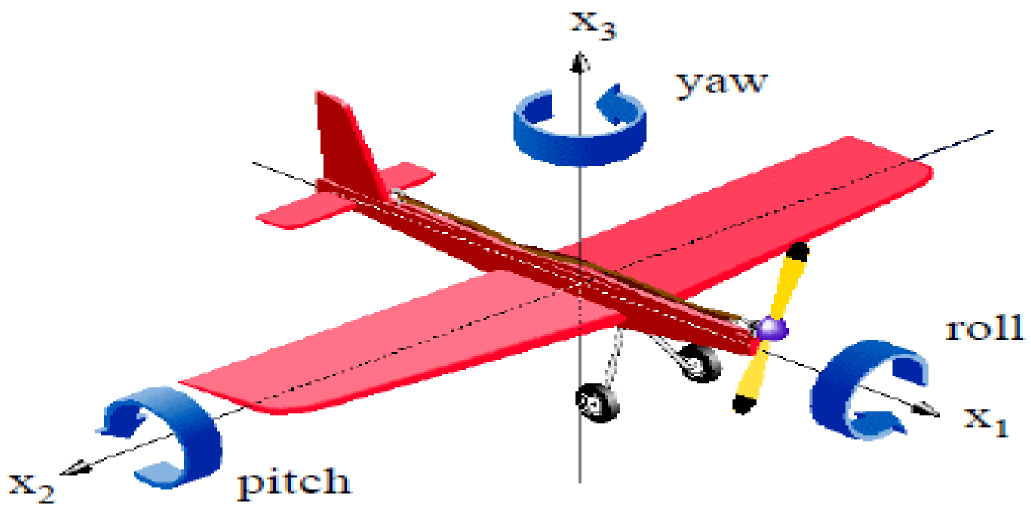


Figure (8): aircraft reference frame

Which: X1axis: oriented in the motion direction.

X3axis: perpendicular to the vehicle plane and in the up direction.

X2axis: to complete the right-handed triad.

Inertial sensors measure rotation rate and acceleration, both of which are vector valued

Variables, inertial navigation system (INS) uses two types of sensors called accelerometers and gyros to measure its motion parameters.

2.4.1 An accelerometer

an electromechanical device that measures acceleration forces, these forces may be static, there are many types of accelerometers developed, the vast majority is based on piezoelectric crystals, but they are too big and to clumsy, a prototypical accelerometer contains a mass suspended on a spring, with some way of measuring the extent to which the spring is compressed.

People tried to develop something smaller, that could increase applicability and started searching in the field of microelectronics.

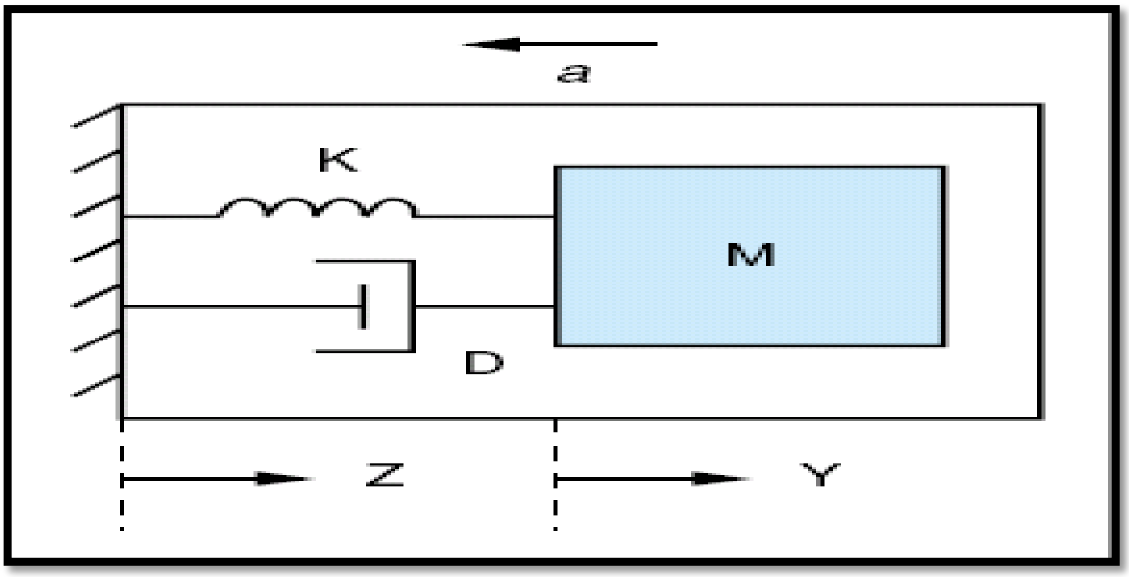


Figure 9 Accelerometer

When the accelerometer's body is accelerated, a force is transmitted to the mass through the spring, causing the spring to stretch or contract, this can then be measured, and results in a value proportional to the accelerometer's acceleration.

2.4.2 A gyroscope

is a device that measures the rate of rotation around the gyro axis, the earliest gyros were actual spinning gyroscopes which, when rotated perpendicularly to their spin axis, will produce a force which can be measured.

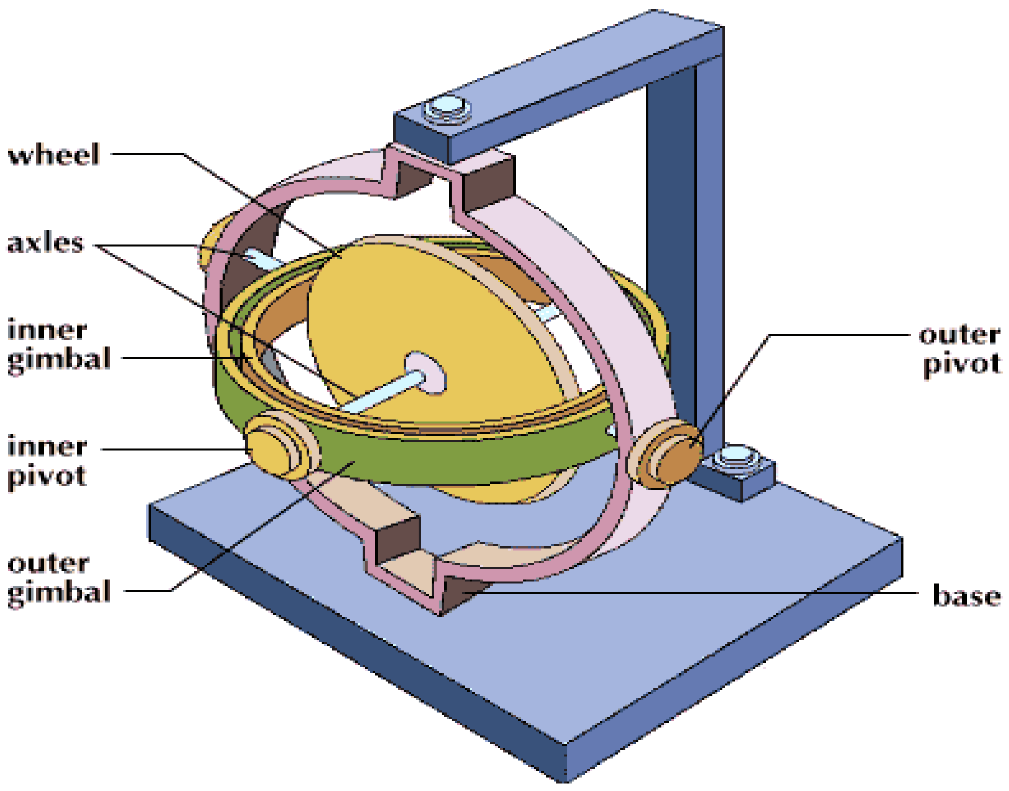


Figure (10): Gyroscope

Given the ability to measure the acceleration of vehicle it would be possible to calculate the change in velocity and position by performing successive mathematical integrations of the acceleration with respect to time, in order to navigate with respect to our inertial reference frame, it is necessary to keep track of the direction in which the accelerometers are pointing.

Rotational motion of the body with respect to inertial reference frame may be sensed using gyroscopic sensors that are used to determine the orientation of the accelerometers at all times, given this information it is possible to resolve the accelerations into the reference frame before the integration process takes place.

An **INS** consists of the following:

- An IMU (containing a cluster of sensors: accelerometers (two or more, but usually three) and gyroscopes (three or more, but usually three).
- Instrument support electronics
- Navigation computers (one or more) calculate the gravitational acceleration (not measured by accelerometers) and doubly integrate the net acceleration to maintain an estimate of the position of the host vehicle.

There are many different designs of INS with different performance characteristics, but they fall generally into two categories:

- gimbaled or stabilized platform techniques.
- strap down

The original applications of INS technology used stable platform techniques.

In such systems, the inertial sensors are mounted on a stable platform and mechanically isolated from the rotational motion of the vehicle. Platform systems are still in use, particularly for those applications requiring very accurate estimates of navigation data, such as ships and submarines.

Modern systems have removed most of the mechanical complexity of platform systems by having the sensors attached rigidly, or "strapped down", to the body of the host vehicle, the potential benefits of this approach are lower cost, reduced size, and greater reliability compared with equivalent platform systems, the major disadvantage is a substantial increase in computing complexity.

2.4.3 Gimballed Systems

Gimbal is a rigid frame with rotation bearings for isolating the inside of the frame from external rotations about the bearing axes.

At least three gimbals are required to isolate a subsystem from host vehicle rotations about three axes, typically labeled roll, pitch, and yaw axes.

The gimbals in an INS are mounted inside one another, gimbals and torque servos are used to null out the rotation of stable platform on which the inertial sensors are mounted.

Gyros were initially located on a rotating platform connected to an outer housing via low friction gimbals, accelerometers were attached to each gimballed gyro axis and thus were held in a fixed orientation, any angular motion was sensed by the rotating platform; this maintains the platform's original orientation, pickoffs on the gimbals measure the movement of the outer body around the steady platform and the accelerometers measure the body's acceleration in the fixed inertial axes.

The gimballed systems primary advantage is its inherently lower error. Since its three orthogonal accelerometers are held in a fixed inertial orientation, only the vertically oriented one will be measuring gravity (and therefore experiencing gravity-related errors).

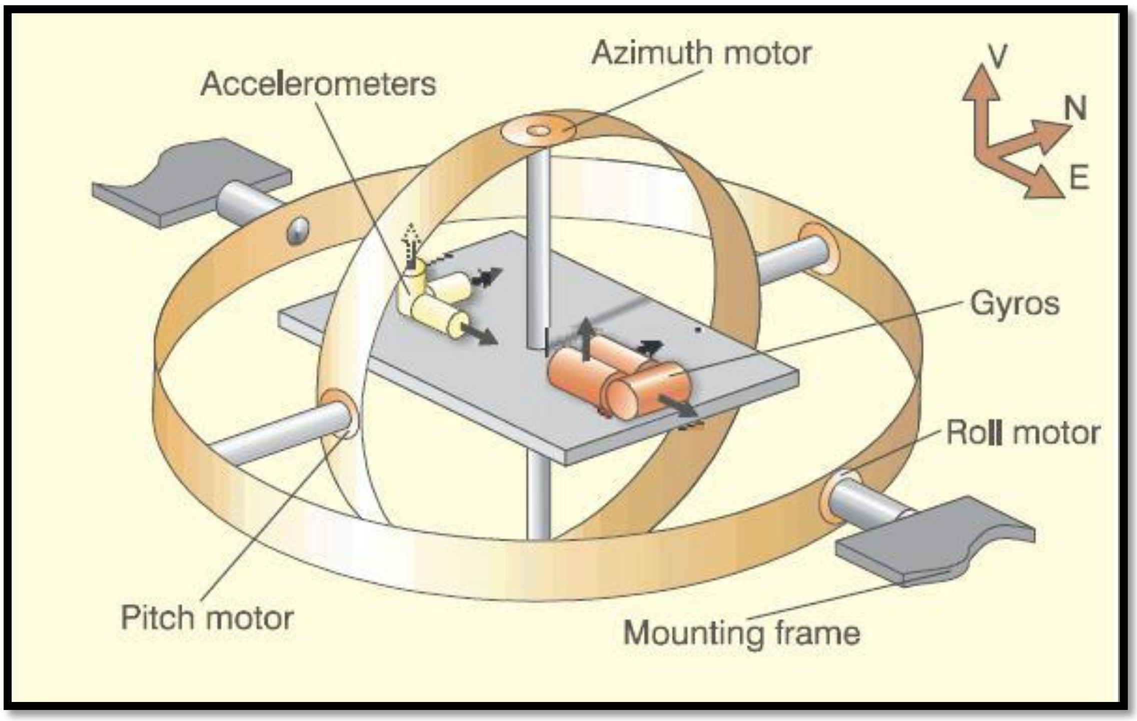


Figure (11): (a) Gimballed inertial system parts

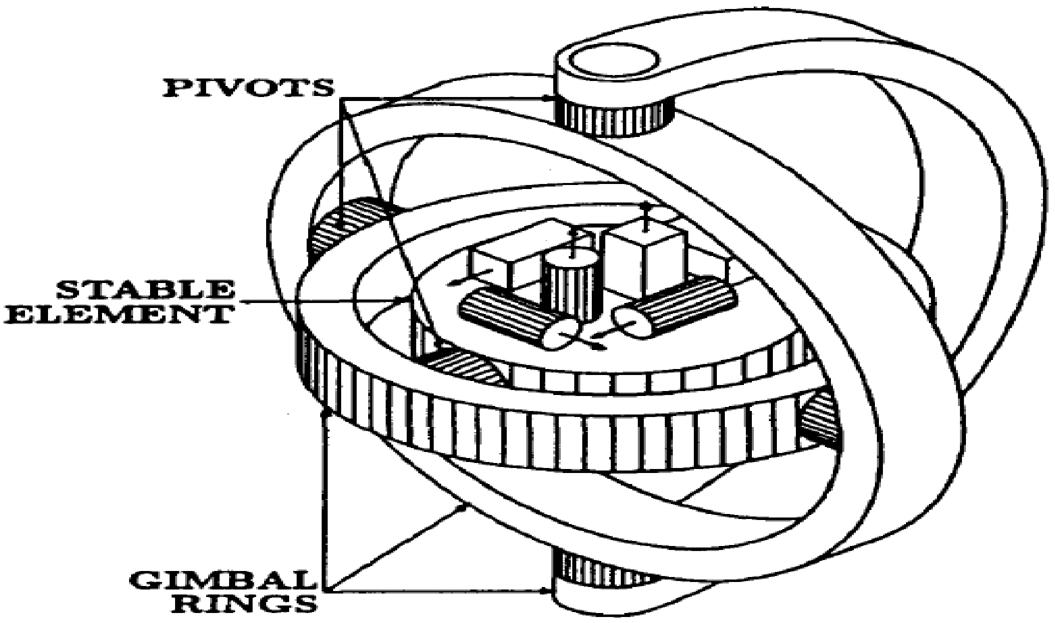


Figure (11) : (b) gimbeld system

2.4.3.1 Gimballed INS work

The gyros of a type known as “integrating gyros” give an output proportional to the angle through which they have been rotated, output of each gyro connected to a servo-motor driving the appropriate gimbal, thus keeping the gimbal in a constant orientation in inertial space.

Gimballed systems have the advantage of simplicity of operation, the primary function of the gyro in a gimballed system is to spin and maintain a high moment of inertia, whereas strap down gyros need to actually measure the subtended angles of motion.

2.4.4 Strap down systems

With fewer moving parts, strap down systems were developed using advanced computer technologies.

Progress in electronics, optics, and solid state technology have enabled very accurate reliable systems to be developed. Modern commercially available equipment's take advantage of integrated circuit technologies. Strap down systems are fixed to the aircraft structure.

The gyros detecting changes in angular rate and the accelerometers detecting changes in linear rate, both with respect to the fixed axes.

These three axes are a moving frame of reference as opposed to the constant inertial frame of reference in the gimballed system, the system computer uses this data to calculate the motion with respect to an inertial frame of reference in three dimensions. The strap down system's main advantage is the simplicity of its mechanical design. Gimballed systems require complex and expensive design for its gimbals, pickoffs, and low-friction platform connections; strap down systems are entirely fixed to the body in motion and are largely solid-state in design.

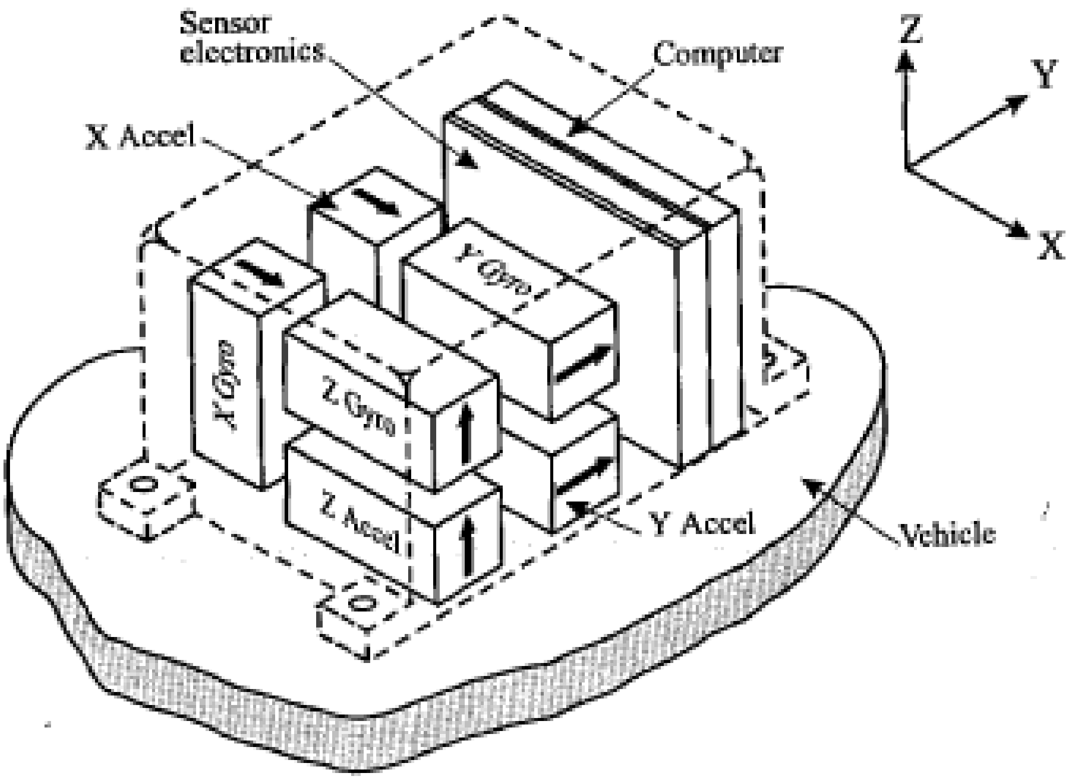


Figure (12): Strap-down system

2.5 Disadvantages of INS

- Mean-squared navigation errors increase with time.
- Cost, including:
  1. Acquisition cost, which can be an order of magnitude (or more) higher than GPS receivers.
  2. Operations cost, including the crew actions and time required for initializing position and attitude.
  3. Maintenance cost. Electromechanical avionics systems (e.g., INS) tend to have higher failure rates and repair cost than purely electronic avionics systems (e.g., GPS).
- Size and weight, which have been shrinking
- Power requirements, which have been shrinking along with size and weight but are still higher than those for GPS receivers.
- Heat dissipation, which is proportional to and shrinking with power requirements.

2.6 Micro-Electro-Mechanical Systems (MEMS)

Micro-Electro-Mechanical Systems (MEMS) is then integration of mechanical elements, sensors, actuators, and electronics on a common substrate through the utilization of microfabrication technology or “micro technology”.

Microelectromechanical systems (MEMS) refer to a collection of microseconds and actuators that can sense its environment and have the ability to react to changes in that environment with the use of a microcircuit control.

They include, in addition to the conventional microelectronics packaging, integrating antenna structures for command signals into microelectromechanical structures for desired sensing and actuating functions. Micro components make the system faster, more reliable, cheaper, and capable of incorporating more complex functions. In the beginning of 1990s, MEMS emerged with the aid of the development of integrated circuit (IC) fabrication processes, in which sensors, actuators, and control functions are fabricated in silicon.

Since then, remarkable research progresses have been achieved in MEMS under the strong capital promotions from both government and industries. In addition to the commercialization of some less integrated MEMS devices, such as micro accelerometers, inkjet printer head, micro mirrors for projection, etc., the concepts and feasibility of more complex MEMS devices have been proposed and demonstrated for the applications in such varied fields as microfluidics, aerospace, biomedical, chemical analysis, wireless communications data storage, display, optics, etc.

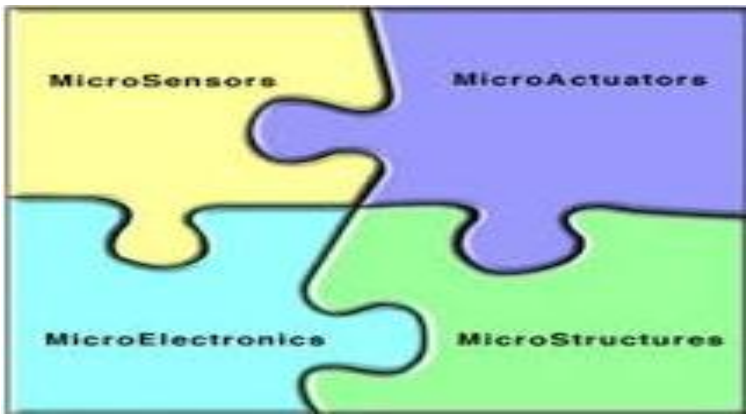


Figure (13): MEMS components

The physical mechanisms underlying MEMS accelerometers include capacitive, piezo resistive, electromagnetic, piezoelectric, ferroelectric, optical, and tunneling.

The most successful types are based on capacitive transduction due to the simplicity of the sensor element, small size, low power consumption, and stability over a wide temperature range.

The first micro machined accelerometer was designed in 1979 at Stanford University, but it took over 15 years before such devices became accepted mainstream products for large volume applications.

In the 1990s MEMS accelerometers revolutionized the automotive-airbag system industry, since then they have enabled unique features and applications ranging from hard-disk protection on laptops to game controllers. More recently, the same sensor-core technology has become available in fully integrated, full-featured devices suitable for industrial applications, micro machined accelerometers are a highly enabling technology with a huge commercial potential. They provide lower power, compact and robust sensing. Multiple sensors are often combined to provide multi-axis sensing and more accurate data. Accelerometers are being incorporated into more and more personal electronic devices such as media players and gaming devices, also found real-time applications in controlling and monitoring military and aerospace systems. Advantage that MEMS can bring relates with the system integration. Instead of having a series of external components (sensor, inductor...) connected by wire or soldered to a printed circuit board

Chapter Three: Proposed Inertial Navigation System using MPU6050

3.1 Introduction

In this chapter, a basic IMU (Inertia Measurement Unit) sensors principles, Arduino Uno, and a short brief for interfacing Arduino with the best IMU sensor available i.e. **MPU 6050** is introduced.

3.2 Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analogue inputs, a 16 MHz quartz crystal, a 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. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards can read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux, most microcontroller systems are limited to Windows.

Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community. Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms.

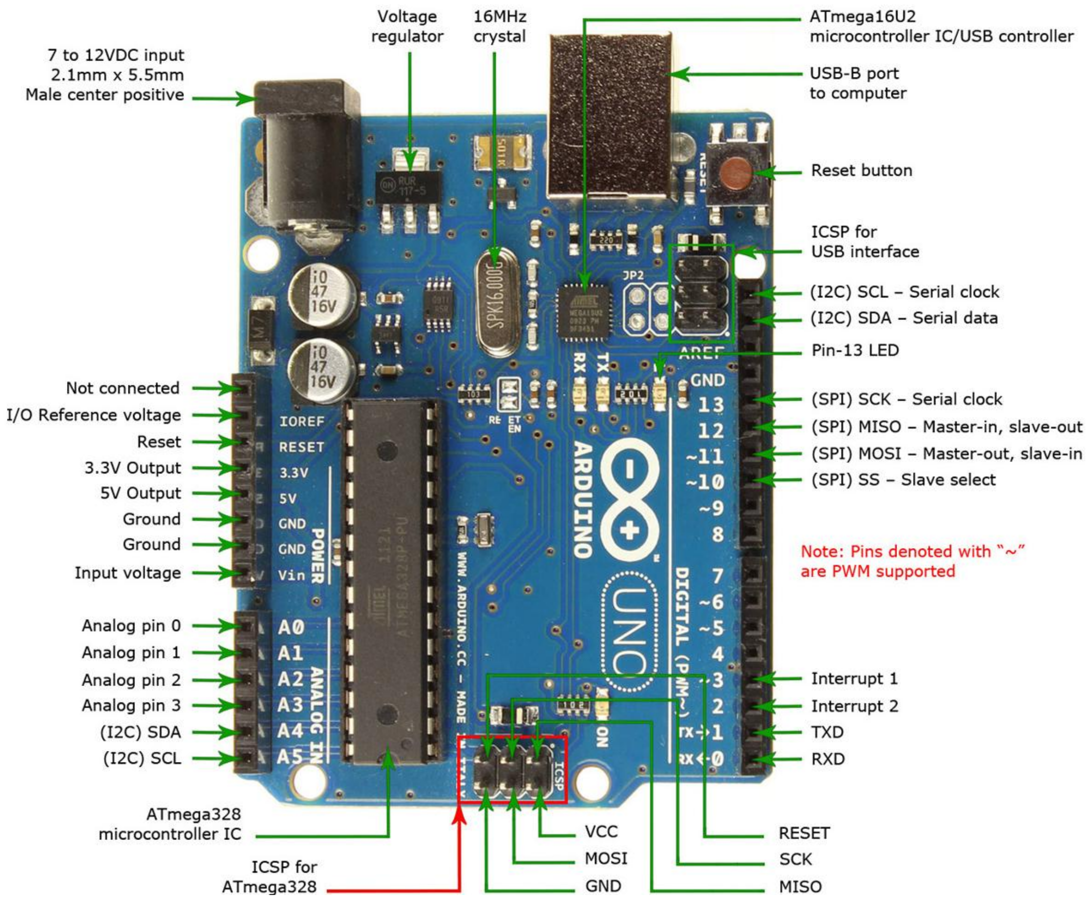
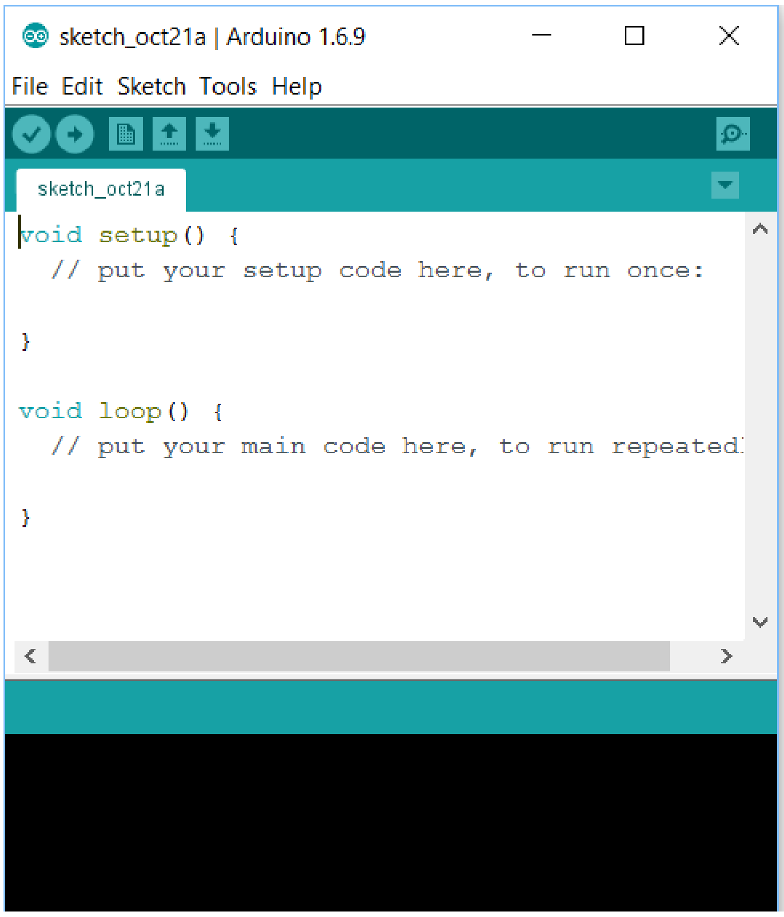


Figure (14):Arduino Pins

3.2.1 Programming

The Arduino Uno can be programmed with the (Arduino Software (IDE)). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). The ATmega328 on the Arduino Uno comes preprogramed with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol. You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header using Arduino ISP or similar.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeated

}
```

Figure (15):Arduino IDE

3.2.2 Warnings

The Arduino Uno 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.

3.2.3 Power

The Arduino Uno board can be powered via the USB connection or with an external power supply, 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 from 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become 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: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3.3V: A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.

3.2.4 Memory

The ATmega328 has 32 KB (with 0.5 KB occupied by the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM.

3.2.5 Input and Output

See the mapping between Arduino pins and ATmega328P ports. The mapping for the Atmega8, 168, and 328 is identical.

3.2.6 Interrupts

The processor at the heart of any Arduino has two different kinds of interrupts: “external”, and “pin change”. There are only two external interrupt pins on the ATmega168/328 (i.e., in the Arduino Uno/Nano/Due), INT0 and INT1, and they are mapped to Arduino pins (2) and (3). These interrupts can be set to trigger on RISING or FALLING signal edges, or on low level. The triggers are interpreted by hardware, and the interrupt is very fast.

3.2.7 MPU (Microprocessor Unit)

The MPU-60X0 is the world’s first integrated 6-axis Motion Tracking device designed for the low power, low cost, and high-performance requirements of smartphones, tablets and wearable sensors. It combines a 3-axis gyroscope, 3-axis accelerometer on the same silicon die together, This combination of sensors is frequently referred to as an IMU, or “Inertial Measurement Unit”, and it is used in airplanes, spacecraft, GPS navigators (for use when GPS signals are unavailable) and other devices. The number of sensor inputs in an IMU are referred to as “DOF” (Degrees of Freedom), so a chip with a 3-axis gyroscope and a 3-axis accelerometer would be a 6-DOF IMU, and a ("Digital Motion Processor") DMP all in a small 4x4x0.9mm package. The DMP can do fast calculations directly on the chip. This reduces the load for the microcontroller (like the Arduino). The DMP is even able to do calculations with the sensor values of another chip, for example a magnetometer connected to the second (sub)-I2C-bus.

The MPU-60X0 Motion Tracking device, with its 6-axis integration, on-board Motion Fusion™, and run-time calibration firmware, enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices, guaranteeing optimal motion performance for consumers.

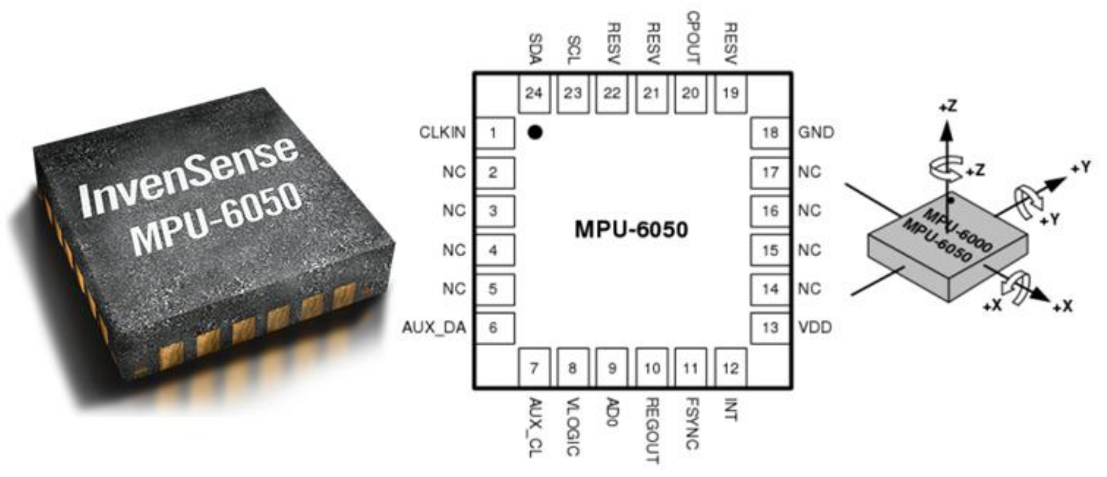


Figure (16): MPU6050

The MPU-60X0 features three 16-bit analogue-to-digital converters (ADCs) for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps) and a user-programmable accelerometer full-scale range of ±2*g*, ±4*g*, ±8*g*, and ±16*g*. Communication with all registers of the device is performed using either I2C at 400kHz or SPI at 1MHz (MPU-6000 only). For applications requiring faster communications, the sensor and interrupt registers may be read using SPI at 20MHz (MPU-6000 only). Additional features include an embedded.

3.3 I^{2}C

The I^{2}C communication bus is very popular and broadly used by many electronic devices because it can be easily implemented in many electronic designs which require communication between a master and multiple slave devices or even multiple master devices, well each device has a present ID or a unique device address so the master can choose with which devices will be communicating.

The two wires, or lines are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL line is the clock signal which synchronize the data transfer between the devices on the I2C bus and it’s generated by the master device. The other line is the SDA line which carries the data.

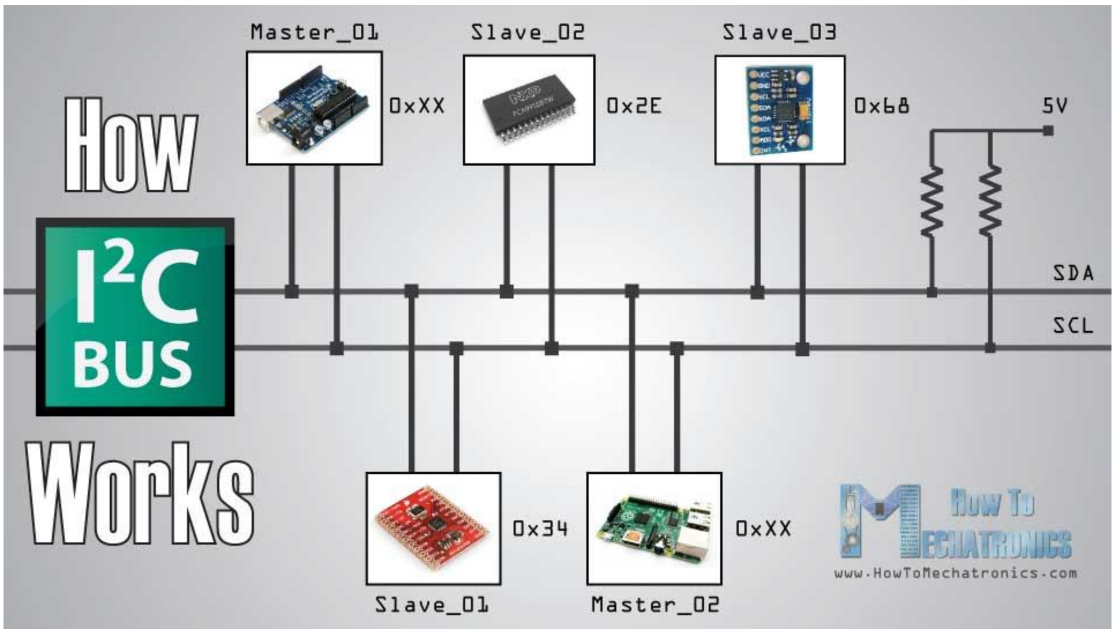


Figure (17): I2C Bus works

The two lines are "open-drain" which means that pull up resistors needs to be attached to them so that the lines are high because the devices on the I^{2}C bus are active low. Commonly used values for the resistors are from 2K for higher speeds at about 400 kbps, to 10K for lower speed at about 100 kbps.

Lastly, the MPU-6050 has a FIFO buffer, together with a built-in interrupt signal. It can be instructed to place the sensor data in the buffer and the interrupt pin will tell the Arduino, when data is ready to be read.

3.4 GY521

The InvenSense MPU-6050 sensor contains a MEMS accelerometer and a MEMS gyro in a single chip. It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefor it captures the x, y, and z channel at the same time.

- Chip: MPU-6050.
- Power supply: 3.5V (But as there is a voltage regulator on the breakout board, you can use 5V directly).
- Communication mode: standard IIC communication protocol.
- Chip built-in 16bit AD converter, 16bit data output.
- Gyroscopes range: +/- 250 500 1000 2000 degree/sec.

- Acceleration range: +/- 2g, +/- 4g, +/- 8g, +/- 16g.

Table 1 Description of GY-521 pins

Pins	Description
VCC	Power
GND	Ground
SCL	Serial CLock
SDA	Serial DAta
XCL	Auxiliary CLock
XDA	Auxiliary DAta
INT	INTerrupt

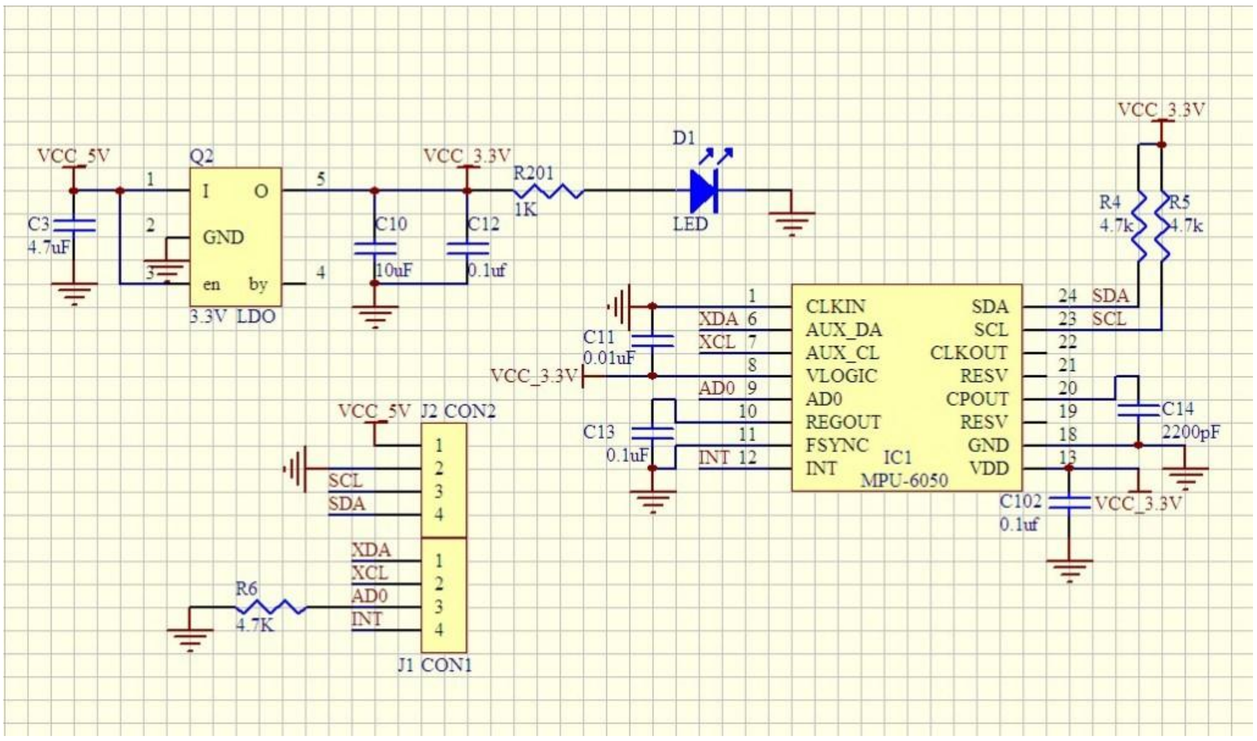


Figure (18): the schematic of the GY-521 break-out board for the MPU6050 chip

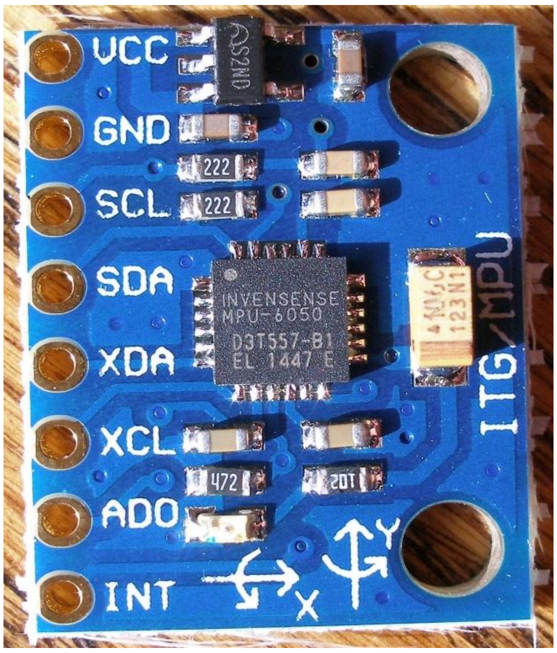


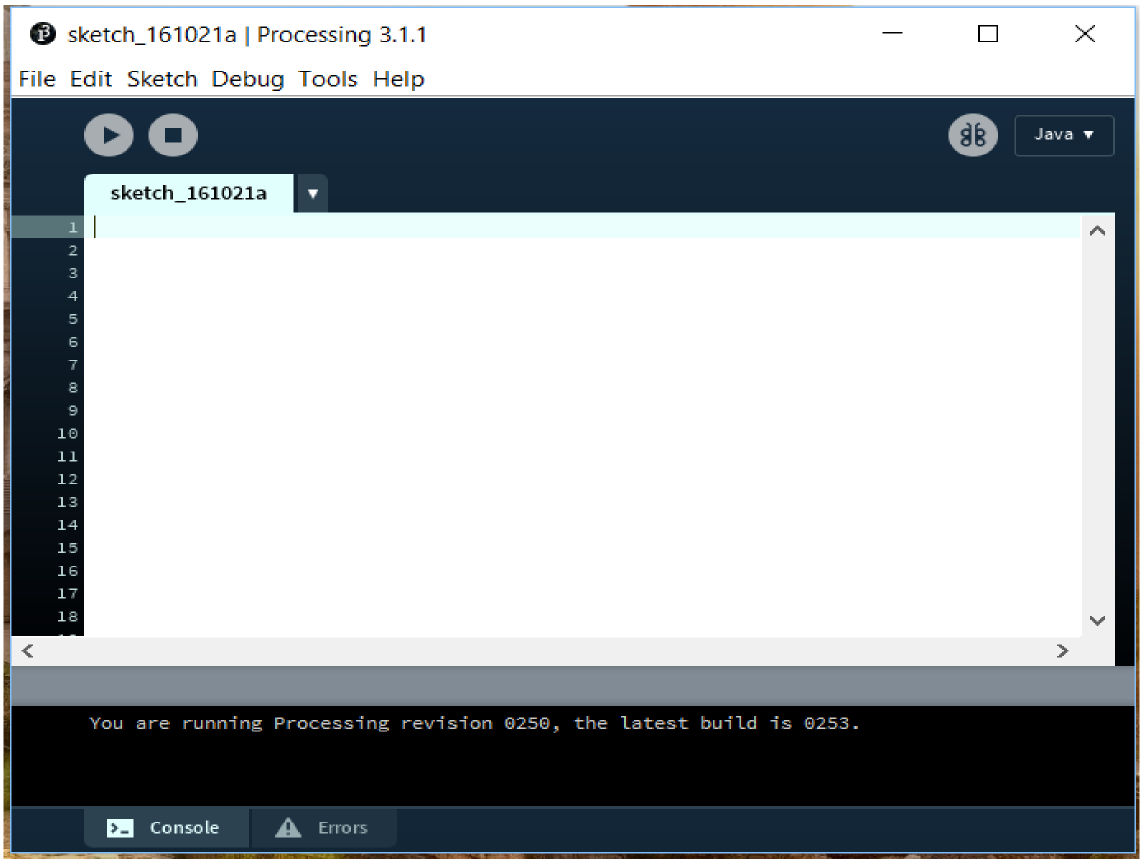
Figure (19): GY-521

Table 2 Pins connection between Arduino & GY-521

GY-521	Arduino Uno
VCC	3.3V
GND	GND
SCL	A5
SDA	A4
D2	INT

3.5 Processing

Is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. There are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning and prototyping.


sketch_161021a | Processing 3.1.1
File Edit Sketch Debug Tools Help
Java
sketch_161021a
Console
Errors

```
You are running Processing revision 0250, the latest build is 0253.
```

Figure (20): Sketch of the Processing software

3.5.1 Toxic library

Is an independent, open source library collection for computational design tasks with Java & Processing developed by Karsten “toxi” Schmidt (thus far). The classes are purposefully kept fairly generic in order to maximize re-use in different contexts ranging from generative design, animation, interaction/interface design, data visualization to architecture and digital fabrication, use as teaching tool and more.

The Processing sketch is called MPUTeapot.pde, even though the figure it displays is a 3-D arrow, not a teapot.

The 3-D airplane/arrow in the Processing sketch follows the rotation of the IMU without significant jitter or lag. In addition, the demo easily performs rotations of any angle, even those greater than 180 degrees.

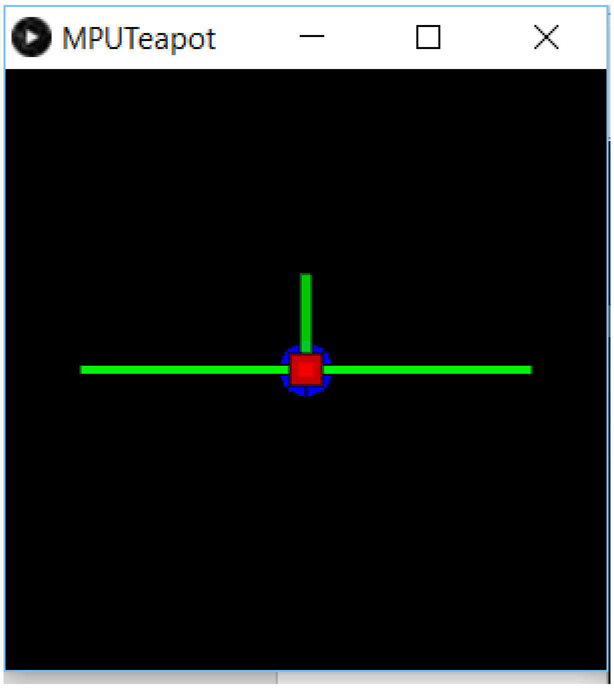


Figure (21): Teapot in levelled position

learn to wire a simple circuit to test MPU6050 with an Arduino and simulate the YAW, PITCH and ROLL on a 3D model plane on the screen, this is intended as a learning tool to get familiar with gyro modules, breakout boards and installing the necessary libraries to Arduino IDE to allow you to make the best use of MEMS gyro and save time instead of writing complex code from scratch.

Here's what you need:

- 1 x Arduino UNO + USB cable
- 1 x mini prototyping breadboard
- 1 x GY-521 breakout board
- Some male-to-male or male-to-female jumper cables
- Soldering iron + solder

3.6 Step 1: Wire circuit

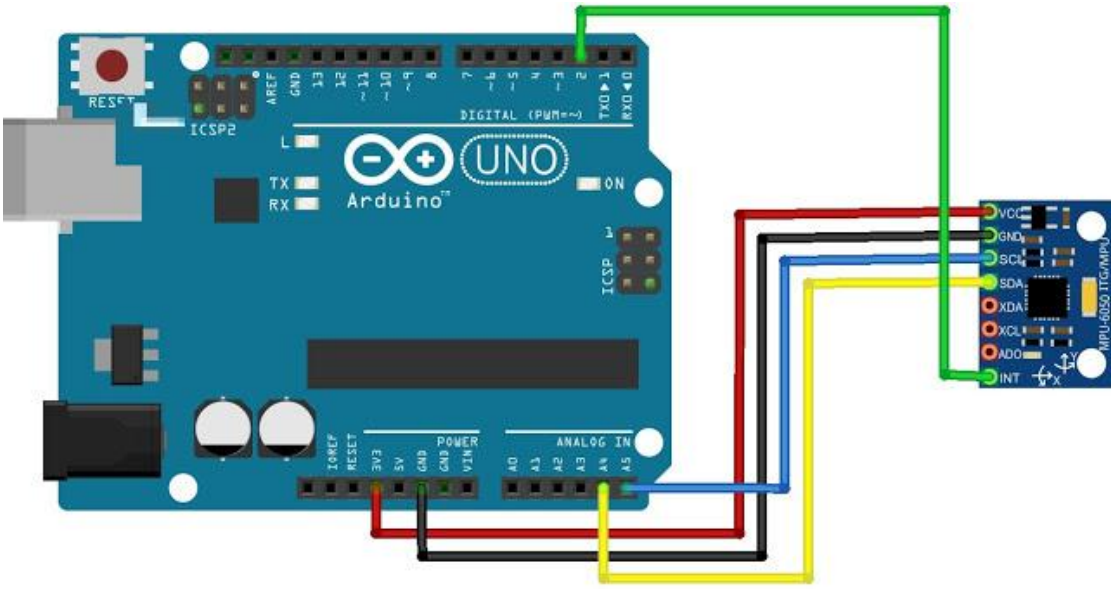


Figure (22):Arduino Uno connected with GY-521

3.7 Step 2: Install I2Cdev & MPU6050 Libraries

Iwe were to write the code from scratch, it would take ages and there would be a lot of reverse engineering required to make good use of the module's proprietary Digital Motion Processing (DMP) engine because Invensense intentionally released minimal data on its MPU6050. Good thing someone has already done the hard work for us; Jeff Rowberg wrote some Arduino libraries to obtain the accelerometer / gyro data and handle all the calculations. They are available as a zip file from here:

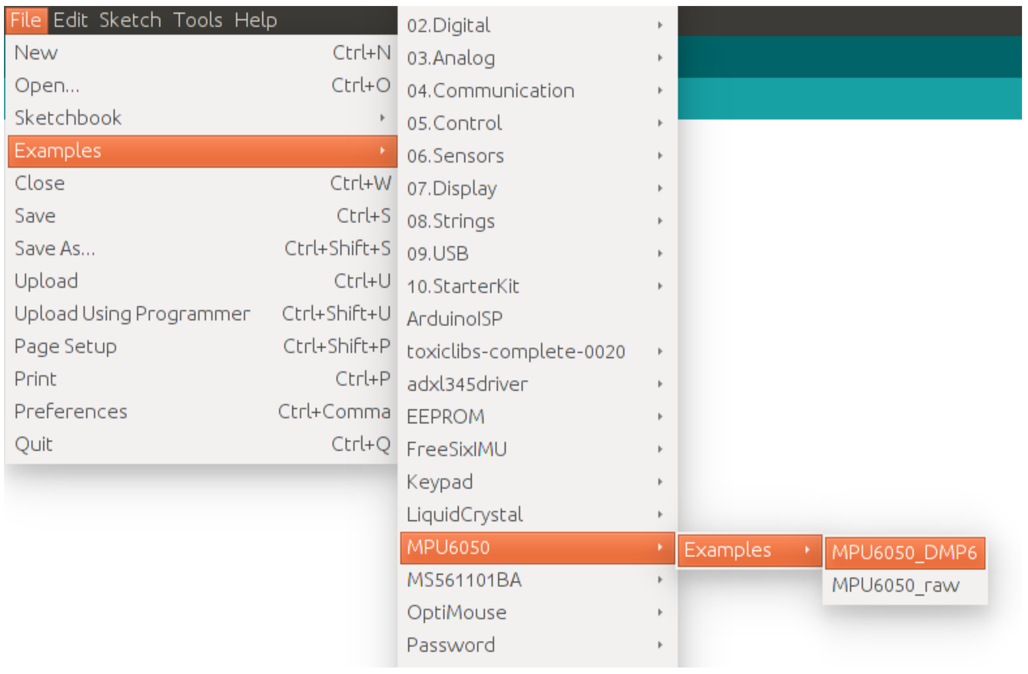
https://github.com/jrowberg/i2cdevlib/zipball/master

Once unzipped, find the Arduino folder within it and copy the two folders "I2Cdev" and "MPU6050" over to your Arduino "libraries" folder in the following directory:

C:\Program Files (x86)\Arduino\libraries

Then open the Arduino IDE and in the examples section, you should find MPU6050_DMP6 within MPU6050.

Directory of the MPU6050 code is :

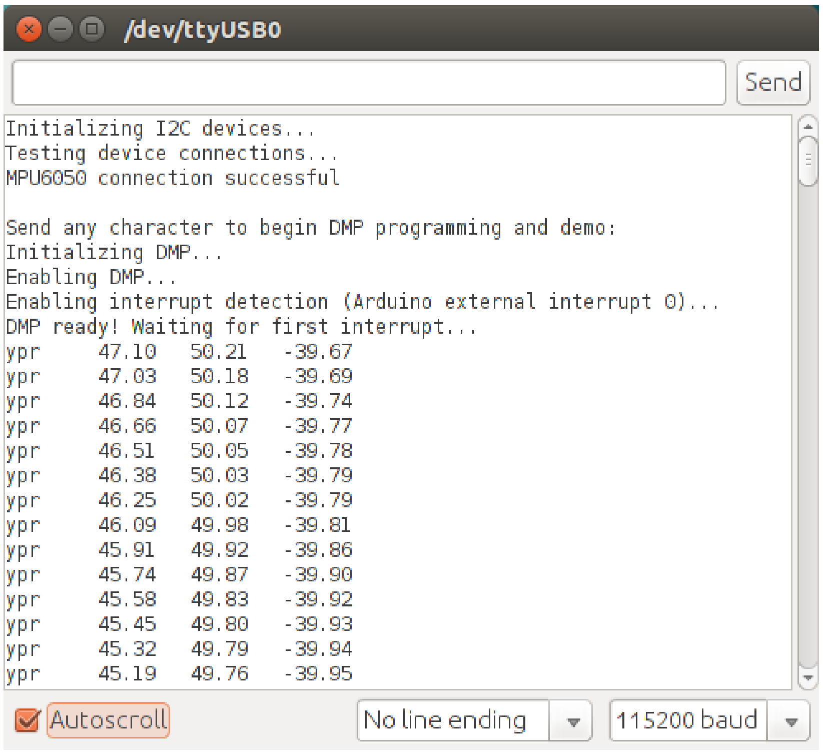


.

Open it, plug Arduino in, select the appropriate COM Port and upload the sketch.

In the Serial Window, select a baud rate of 115200.

should be prompted that the MPU6050 connection was successful, test the data collection by typing anything in the text bar and pressing enter, the data should start showing up.



```
Initializing I2C devices...
Testing device connections...
MPU6050 connection successful

Send any character to begin DMP programming and demo:
Initializing DMP...
Enabling DMP...
Enabling interrupt detection (Arduino external interrupt 0)...
DMP ready! Waiting for first interrupt...
ypr    47.10   50.21   -39.67
ypr    47.03   50.18   -39.69
ypr    46.84   50.12   -39.74
ypr    46.66   50.07   -39.77
ypr    46.51   50.05   -39.78
ypr    46.38   50.03   -39.79
ypr    46.25   50.02   -39.79
ypr    46.09   49.98   -39.81
ypr    45.91   49.92   -39.86
ypr    45.74   49.87   -39.90
ypr    45.58   49.83   -39.92
ypr    45.45   49.80   -39.93
ypr    45.32   49.79   -39.94
ypr    45.19   49.76   -39.95
```

Figure (23): Total Pitch, Roll, and Yaw angles readings

Now we want to set the code to run the teapot demo to show the 3D simulation. Close the serial window, then find and comment out the line #define OUTPUT_READABLE_YAWPITCHROLL and uncomment the line //#define OUTPUT_TEAPOT. Select "save as" and choose where you want to save the modified code. Upload again but don't open the serial window this time.

3.8 Step 3: Install Latest Version of Processing & ToxicLibs Library

To run a 3D simulation of the yaw / pitch / roll values on an airplane on the screen, we'll be running the teapot demo from the MPU6050_DMP6 example from Jeff Rowberg's MPU6050 library. However, the Arduino IDE will only be acquiring the data, to display the 3D simulation we'll need additional software: Processing.
Download Processing from here, then unzip to wherever you like:

https://processing.org/download/?processing

We'll need one final library to get things running: ToxicLib.

This library will be going into Processing's libraries folder instead of Arduino's. The latest version of the ToxicLibs library is here:

https://bitbucket.org/postspectacular/toxiclibs/downloads/

The "libraries" folder of Processing can be found by following (starting from within the processing folder): modes -> java -> libraries. Unzip ToxicLibs and place ALL the contents there.

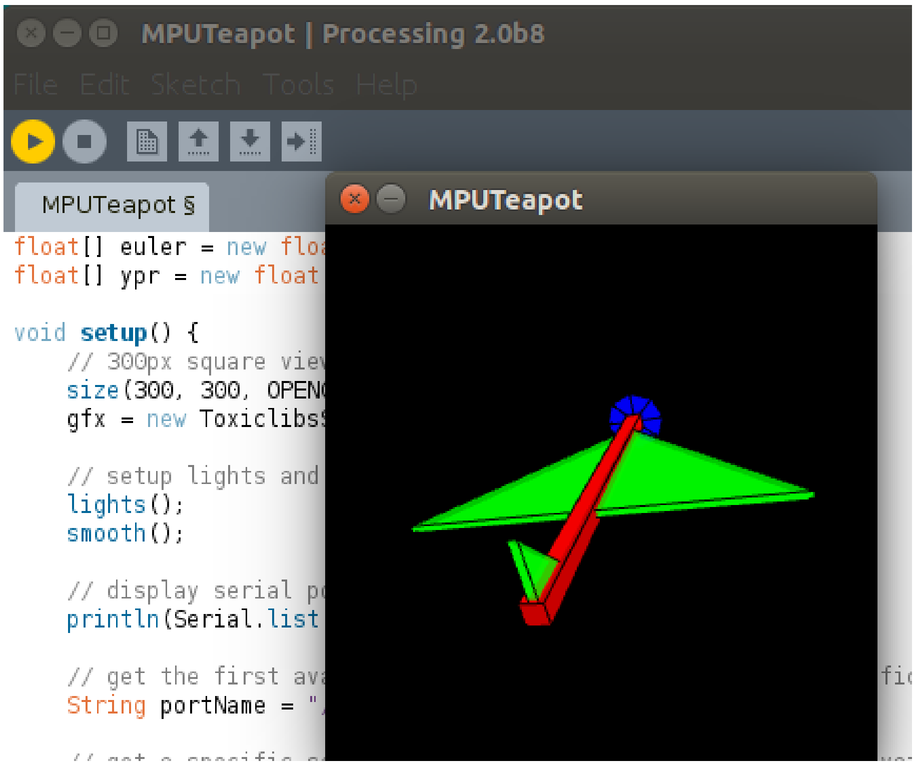
3.9 Step 4: Run the simulation

Last of all, open the Processing application file and then

File -> Open -> follow this directory C:\Program Files (x86)\Arduino\libraries\MPU6050\Examples\MPU6050_DMP6\Processing\MPUTeapot)

and open the MPUTeapot file.

Click the play button and the system should calibrate for about 20-30 seconds, leave the gyro stationary during that period



```
float[] euler = new floa
float[] ypr = new float

void setup() {
    // 300px square vie
    size(300, 300, OPEN
    gfx = new Toxiclibs

    // setup lights and
    lights();
    smooth();

    // display serial p
    println(Serial.list

    // get the first ava                                    fic
    String portName = "/
```

Figure (24): Simulation of the Teapot

3.10 LCD (16 x2):

LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A **16x2 LCD** means it can display 16 characters per line and there are 2 such lines. In this LCD, each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data

A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over seven segments and other multi segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on.

.

The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD. Click to learn more about internal structure of a LCD.

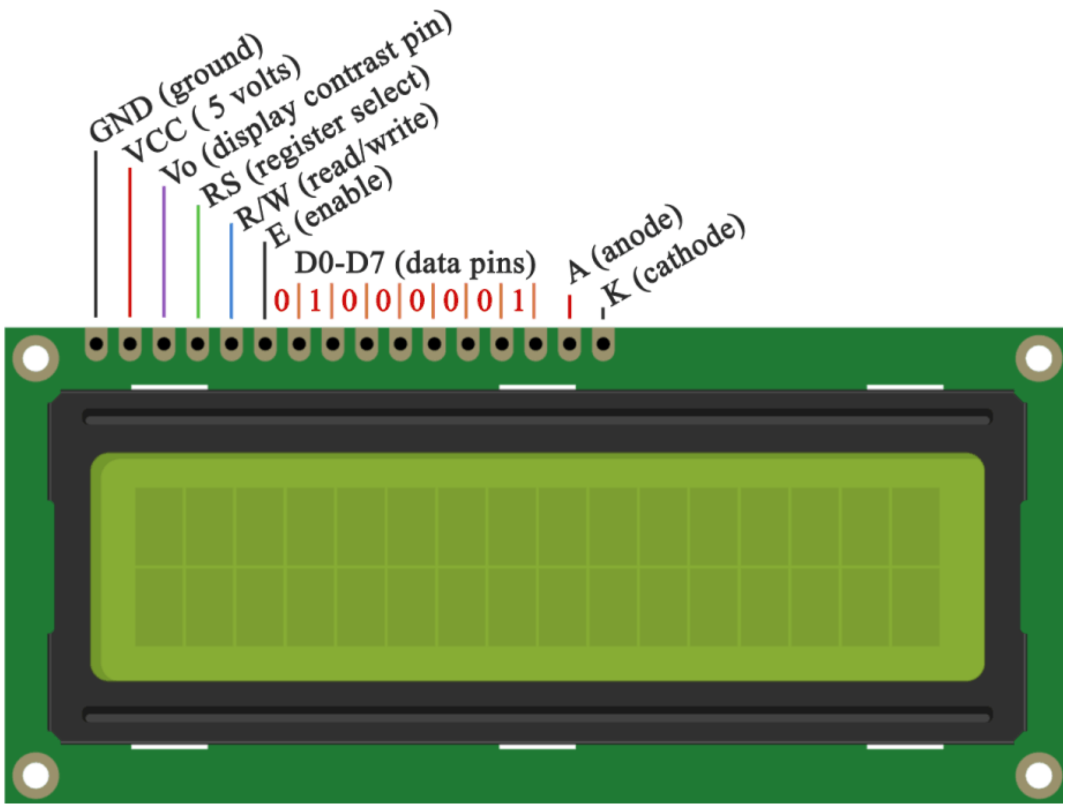


Figure (25):LCD (Liquid Crystal Display)

3.11 Variable Resistor

A variable resistor is a device that is used to change the resistance according to our needs in an electronic circuit. It can be used as a three terminal as well as a two-terminal device. Mostly they are used as a three-terminal device. Variable resistors are mostly used for device calibration.

As shown in the diagram below, a variable resistor consists of a track which provides the resistance path. Two terminals of the device are connected to both the ends of the track. The third terminal is connected to a wiper that decides the motion of the track. The motion of the wiper through the track helps in increasing and decreasing the resistance.

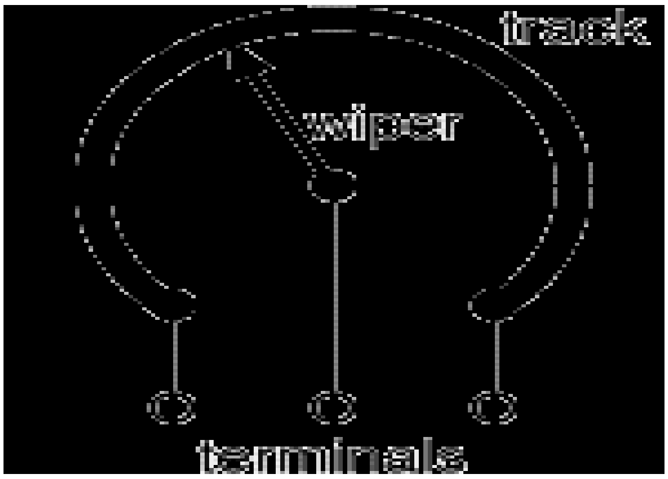


Figure (26): Variable resistor

3.12 Another simulation by using LCD

3.12.1 Step 1: Connection Pin LCD (16*2) with Arduino Uno

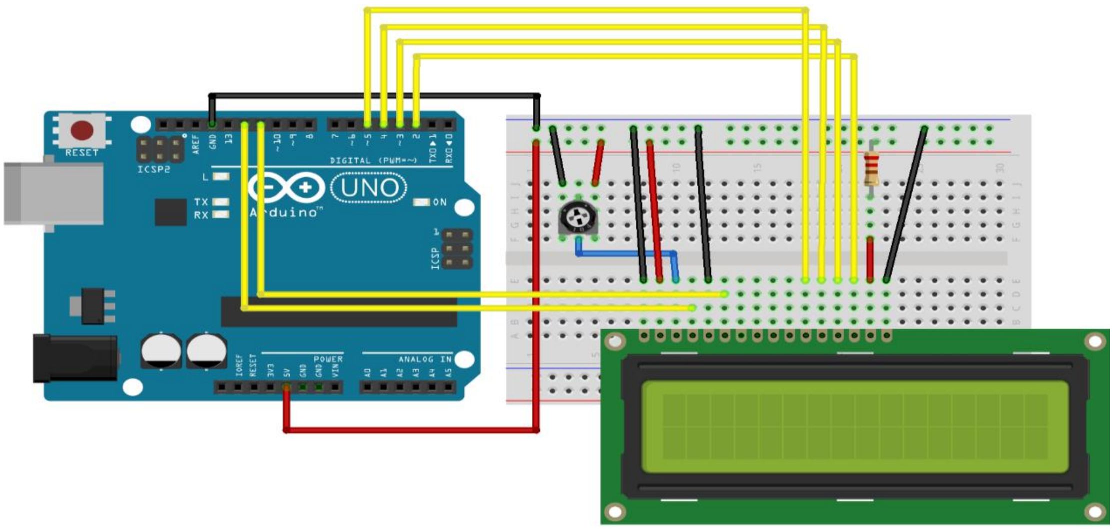


Figure (27): LCD (16*2) and Arduino Uno Connection

Table 3 Pins connection between Arduino & LCD

LCD	**Arduino Uno**
RS	Pin (12)
EN(enable)	(11)
D4	(7)
D5	(6)
D6	(5)
D7	(4)
R/W & Led-	GND
Ed+	5V
VEE	Variable resistance (10K ohms), Which connected to 5V & GND

3.12.2 Step 2: Download the Liquid Crystal Library

The Liquid Crystal Library is a core library for Arduino - there should be no need to install it. If you need to install it for some reason, visit the Arduino site.

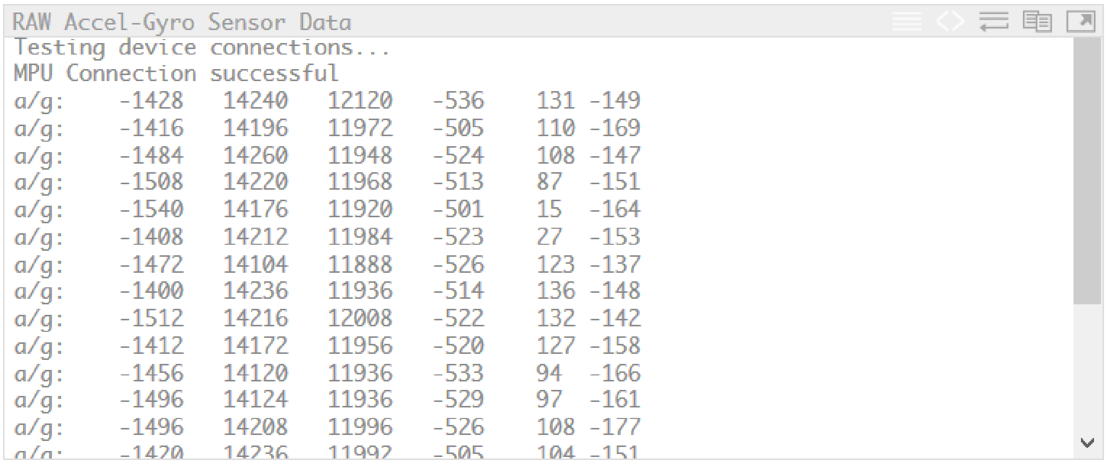
http://arduino.cc/en/Reference/Libraries

Now it is time to get some data from the accel-gyro module! To do this I simply used the sample code which came with the documentation of the MPU6050 in order to read the raw sensor data. For this sample to work, the **I2Cdev** and the **MPU6050** libraries need to be installed. Here is the code:

```
// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files

// for both classes must be in the include path of your project

#include "I2Cdev.h"

#include "MPU6050.h"

 // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation

// is used in I2Cdev.h

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

#include "Wire.h"

#endif

// class default I2C address is 0x68

// specific I2C addresses may be passed as a parameter here

// AD0 low = 0x68 (default for InvenSense evaluation board)

// AD0 high = 0x69

MPU6050 accelgyro;

//MPU6050 accelgyro(0x69); // <-- use for AD0 high
```

```
int16_t ax, ay, az;

int16_t gx, gy, gz;

 // uncomment "OUTPUT_READABLE_ACCELGYRO" if you want to
see a tab-separated

// list of the accel X/Y/Z and then gyro X/Y/Z values in
decimal. Easy to read,

// not so easy to parse, and slow(er) over UART.

#define OUTPUT_READABLE_ACCELGYRO

 #define LED_PIN 13

bool blinkState = false;

 void setup() {

//  join  I2C  bus  (I2Cdev  library  doesn't  do  this
automatically)

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

Wire.begin();

#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE

Fastwire::setup(400, true);

#endif

 // initialize serial communication

// (38400 chosen because it works as well at 8MHz as it
does at 16MHz, but

// it's really up to you depending on your project)

Serial.begin(38400);

 // initialize device

Serial.println("Initializing I2C devices...");

accelgyro.initialize();


// verify connection

Serial.println("Testing device connections...");
Serial.print("MPU Connection ");

Serial.println(accelgyro.testConnection()  ?  "successful"
: "failed");

// configure Arduino LED for

pinMode(LED_PIN, OUTPUT);

}

void loop() {

// read raw accel/gyro measurements from device

accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

 #ifdef OUTPUT_READABLE_ACCELGYRO

// display tab-separated accel/gyro x/y/z values

Serial.print("a/g:\t");

Serial.print(ax); Serial.print("\t");

Serial.print(ay); Serial.print("\t");

Serial.print(az); Serial.print("\t");

Serial.print(gx); Serial.print("\t");

Serial.print(gy); Serial.print("\t");

Serial.println(gz);

#endif

 // blink LED to indicate activity

blinkState = !blinkState;

digitalWrite(LED_PIN, blinkState);

}
```

The result I got on the Serial Monitor looked like this:



```
RAW Accel-Gyro Sensor Data
Testing device connections...
MPU Connection successful
a/g:    -1428   14240   12120   -536    131 -149
a/g:    -1416   14196   11972   -505    110 -169
a/g:    -1484   14260   11948   -524    108 -147
a/g:    -1508   14220   11968   -513    87  -151
a/g:    -1540   14176   11920   -501    15  -164
a/g:    -1408   14212   11984   -523    27  -153
a/g:    -1472   14104   11888   -526    123 -137
a/g:    -1400   14236   11936   -514    136 -148
a/g:    -1512   14216   12008   -522    132 -142
a/g:    -1412   14172   11956   -520    127 -158
a/g:    -1456   14120   11936   -533    94  -166
a/g:    -1496   14124   11936   -529    97  -161
a/g:    -1496   14208   11996   -526    108 -177
a/g:    -1420   14236   11992   -505    104 -151
```

the readings from the accelerometer are divided into the x/y/z values, and the readings from the gyroscope are divided into its x/y/z components. But unfortunately, this data wasn't very usable in its current form. We still had to fuse the accelerometer and gyroscope data together. The roll, pitch, and yaw angles are evaluated due to the following equations:

$$\emptyset = \arctan\left(\frac{Ax}{\sqrt{A_y^2+A_z^2}}\right) \quad \text{.. } (\textit{equation 3.1})$$

$$\rho = Arctan\left(\frac{A_y}{\sqrt{A_x^2+A_z^2}}\right) \quad \text{.. (equation 3.2)}$$

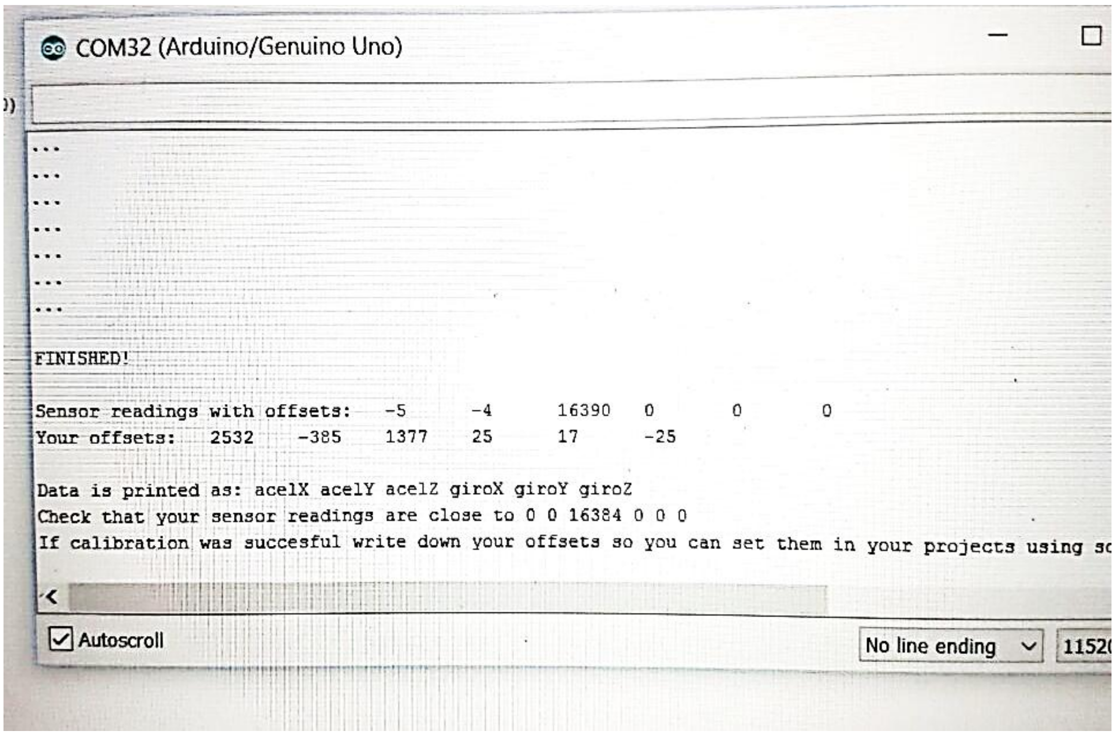
$$\theta = Arctan\left(\frac{\sqrt{{A_x}^2+{A_y}^2}}{A_z}\right) \quad \text{.. (equation 3.3)}$$

Chapter Four: Results and Discussion

4.1 Results

The First calibration was carried out for the GY-521 to set the reference adjustment which represents the airplane at its levelled situation.

The next evaluation was to try to tilt the GY-521 to sense the deviation as digital information.



```
...
...
...
...
...
...
...

FINISHED!

Sensor readings with offsets:   -5      -4      16390   0       0       0
Your offsets:   2532    -385    1377    25      17      -25

Data is printed as: acelX acelY acelZ giroX giroY giroZ
Check that your sensor readings are close to 0 0 16384 0 0 0
If calibration was succesful write down your offsets so you can set them in your projects using sc
```

Figure (28):Calibration of MPU6050 GY-521

For above figure, the calibration offsets are being achieved for the 6 degree of freedom; the first 3 offsets were for (X, Y, Z) accelerometer, and the last three for gyroscope.

The output scale for any setting is [-32768, +32767] for each of the six axes. The default setting in the I2Cdevlib class is +/- 2g for the accel and +/- 250 deg/sec for the gyro. If the device is perfectly level and not moving, then:

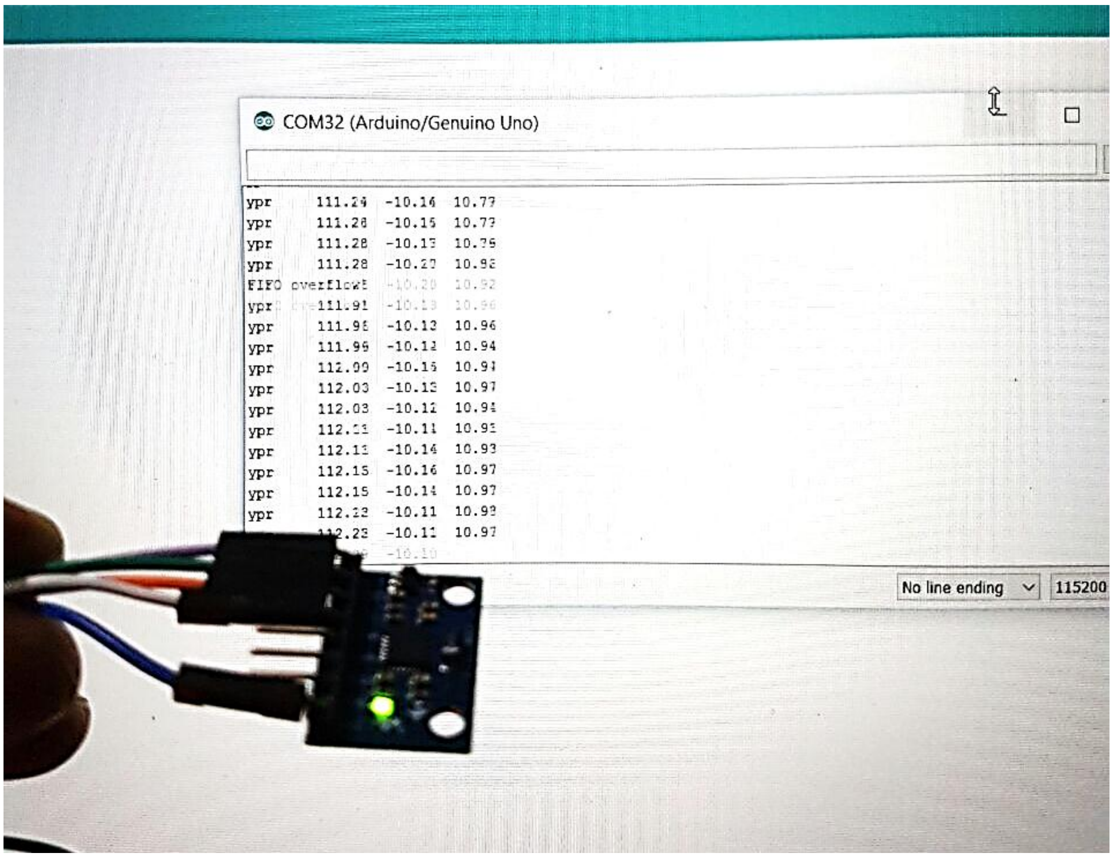
- X/Y accel axes should read 0
- Z accel axis should read 1g, which is +16384 at a sensitivity of 2g
- X/Y/Z gyro axes should read 0

Then, these resultant offsets were taken and replaced in the MPU6050 code,

As showed below to get accurate result and simulation.

So the calibration offsets in Code :

```
// supply your own gyro offsets here, scaled for min sensitivity
mpu.setXGyroOffset(25);
mpu.setYGyroOffset(17);
mpu.setZGyroOffset(-25);
mpu.setXAccelOffset(2532);
mpu.setYAccelOffset(-385);
mpu.setZAccelOffset(1377); // 1688 factory default for my test chip
```

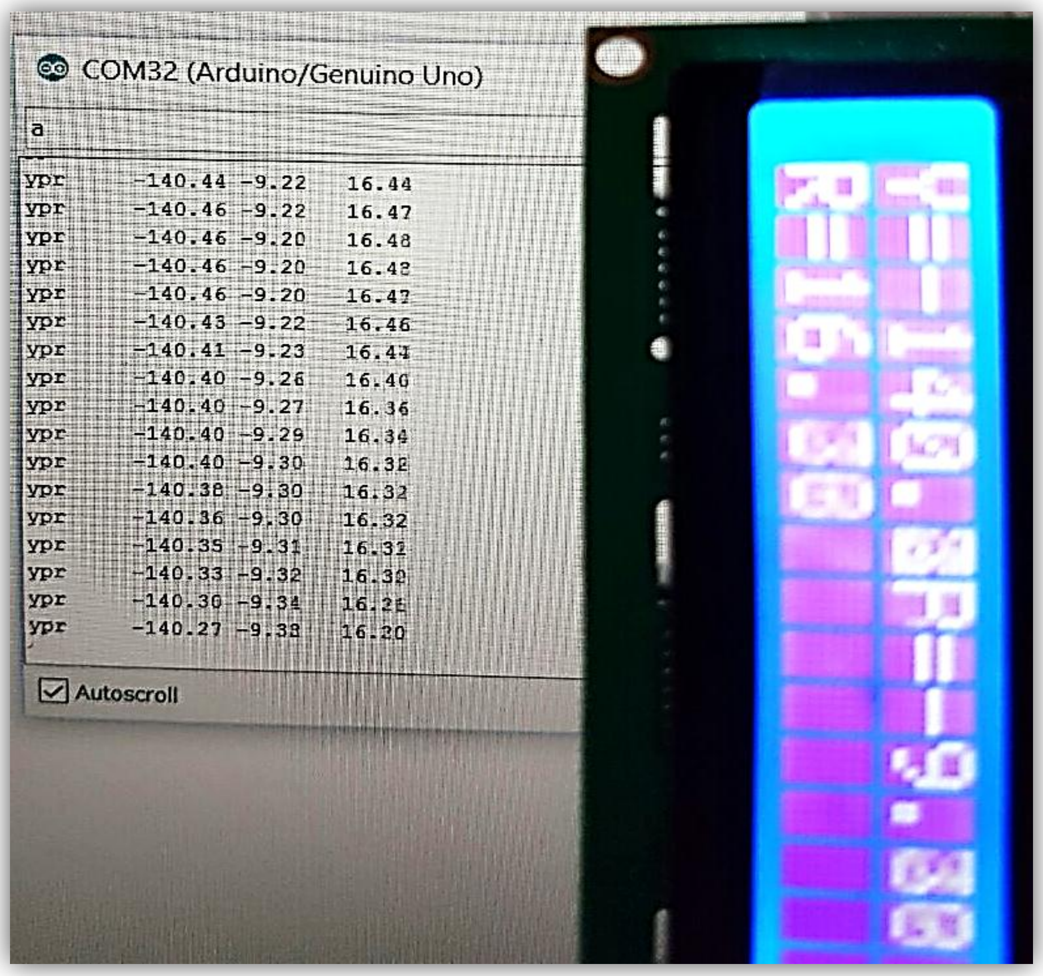


```
ypr     111.24  -10.16  10.77
ypr     111.28  -10.15  10.77
ypr     111.28  -10.13  10.75
ypr     111.28  -10.27  10.92
FIFO overflow!  -10.20  10.92
ypr     111.91  -10.13  10.96
ypr     111.95  -10.13  10.96
ypr     111.99  -10.12  10.94
ypr     112.99  -10.15  10.97
ypr     112.03  -10.13  10.97
ypr     112.03  -10.12  10.95
ypr     112.11  -10.11  10.95
ypr     112.11  -10.14  10.93
ypr     112.15  -10.16  10.97
ypr     112.15  -10.14  10.97
ypr     112.22  -10.11  10.99
        112.23  -10.11  10.97
                -10.10
```

Figure(29): Fetching the motion in serial monitor

At this figure, the code was uploaded and the serial monitor was activated by adding any characteristic, so as shown in the figure, when the sensor (GY-521) was oscillated, the results were presented in the display in 3 columns (Yaw, Pitch and Roll respectively).

In addition to that, there is no line ending, which means it will keep giving results until the serial monitor was closed or unplugging the Arduino from the PC.


COM32 (Arduino/Genuino Uno)

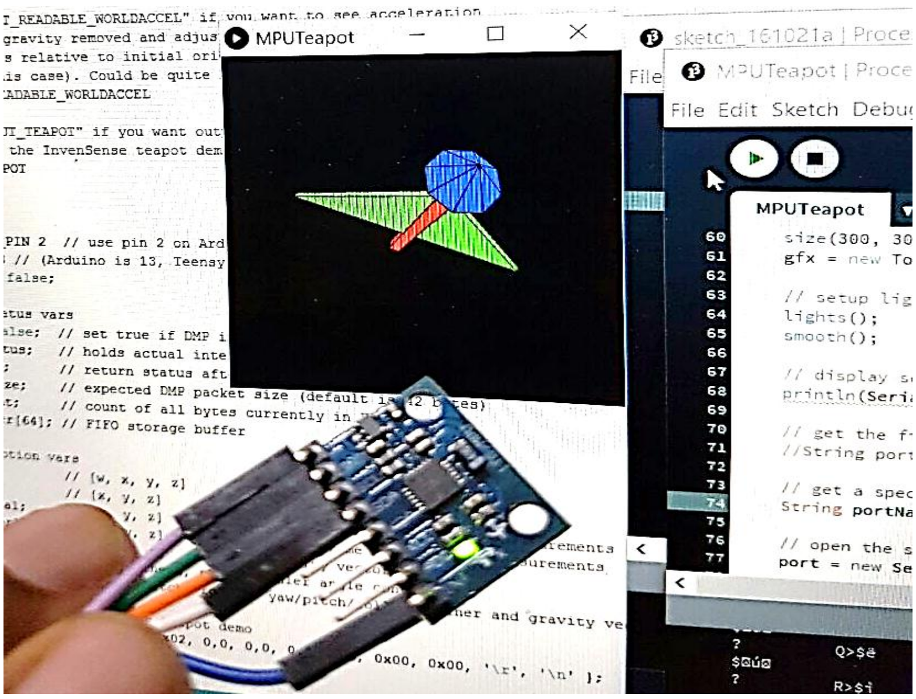
```
a
```

```
ypr     -140.44 -9.22   16.44
ypr     -140.46 -9.22   16.47
ypr     -140.46 -9.20   16.48
ypr     -140.46 -9.20   16.48
ypr     -140.46 -9.20   16.47
ypr     -140.43 -9.22   16.46
ypr     -140.41 -9.23   16.44
ypr     -140.40 -9.26   16.40
ypr     -140.40 -9.27   16.36
ypr     -140.40 -9.29   16.34
ypr     -140.40 -9.30   16.32
ypr     -140.38 -9.30   16.32
ypr     -140.36 -9.30   16.32
ypr     -140.35 -9.31   16.31
ypr     -140.33 -9.32   16.30
ypr     -140.30 -9.34   16.25
ypr     -140.27 -9.38   16.20
```

☑ Autoscroll

Figure (30):Parallel Display of Both LCD & Serial monitor

Now, the LCD was added for displaying with serial monitor. As showed in figure, the results were presented parallel in both in LCD and Arduino.

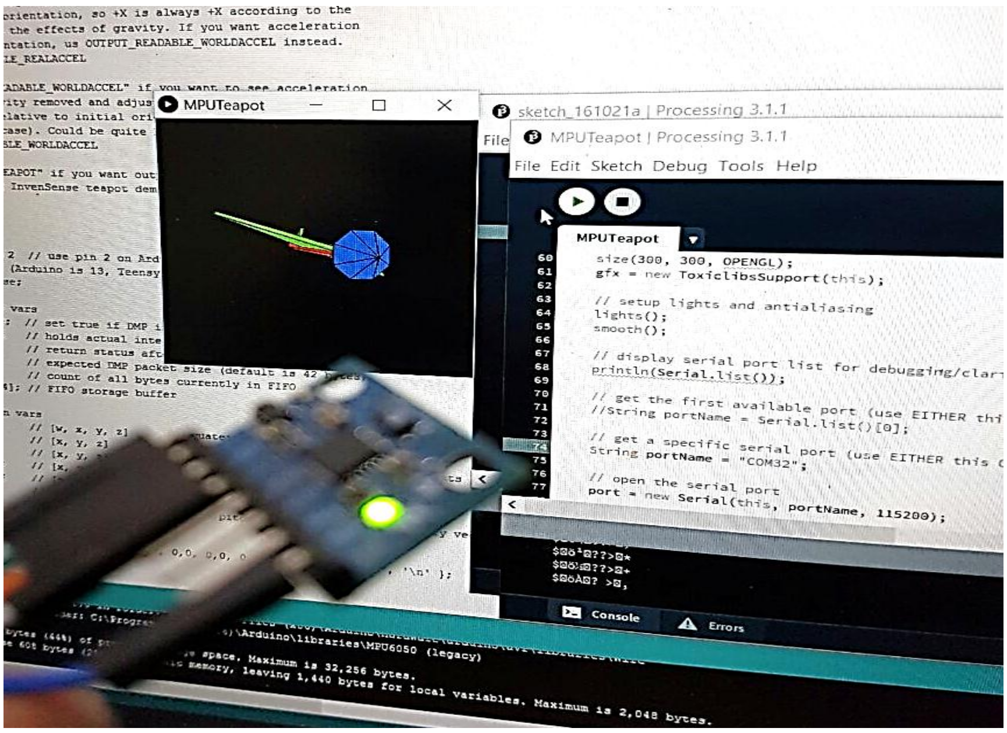


```
I_READABLE_WORLDACCEL" if you want to see acceleration
gravity removed and adjus
s relative to initial ori
is case). Could be quite
ADABLE_WORLDACCEL

JT_TEAPOT" if you want out
 the InvenSense teapot dem
POT

PIN 2  // use pin 2 on Ard
 // (Arduino is 13, Teensy
false;

tus vars
lse;  // set true if DMP i
tus;   // holds actual inte
;      // return status aft
ze;    // expected DMP packet size (default is 42 bytes)
t;     // count of all bytes currently in
r[64]; // FIFO storage buffer

tion vars
        // [w, x, y, z]
        // [x, y, z]
```

```
size(300, 30
gfx = new To

// setup lig
lights();
smooth();

// display s
println(Seri

// get the f
//String port

// get a spec
String portNa

// open the s
port = new Se
```

Figure(31):calibration of GY-521 motion

The Simulation of the motion of the GY-521 was illustrated using the MPUTeapot add-on of the Processing Software.it shows momentarily motion and deviation of the GY-521.


MPUTeapot
sketch_161021a | Processing 3.1.1
MPUTeapot | Processing 3.1.1
File Edit Sketch Debug Tools Help
MPUTeapot
Console
Errors

```
orientation, so +X is always +X according to the
 the effects of gravity. If you want acceleration
ntation, us OUTPUT_READABLE_WORLDACCEL instead.
LE_REALACCEL

ADABLE_WORLDACCEL" if you want to see acceleration
ity removed and adjus
lative to initial ori
case). Could be quite
BLE_WORLDACCEL

EAPOT" if you want out
 InvenSense teapot dem

2  // use pin 2 on Ard
 (Arduino is 13, Teensy
se;

vars
;  // set true if DMP i
   // holds actual inte
   // return status aft
   // expected DMP packet size (default is 42 bytes)
   // count of all bytes currently in FIFO
]; // FIFO storage buffer
```

```
size(300, 300, OPENGL);
gfx = new ToxiclibsSupport(this);

// setup lights and antialiasing
lights();
smooth();

// display serial port list for debugging/clar
println(Serial.list());

// get the first available port (use EITHER thi
//String portName = Serial.list()[0];

// get a specific serial port (use EITHER this 
String portName = "COM32";

// open the serial port
port = new Serial(this, portName, 115200);
```

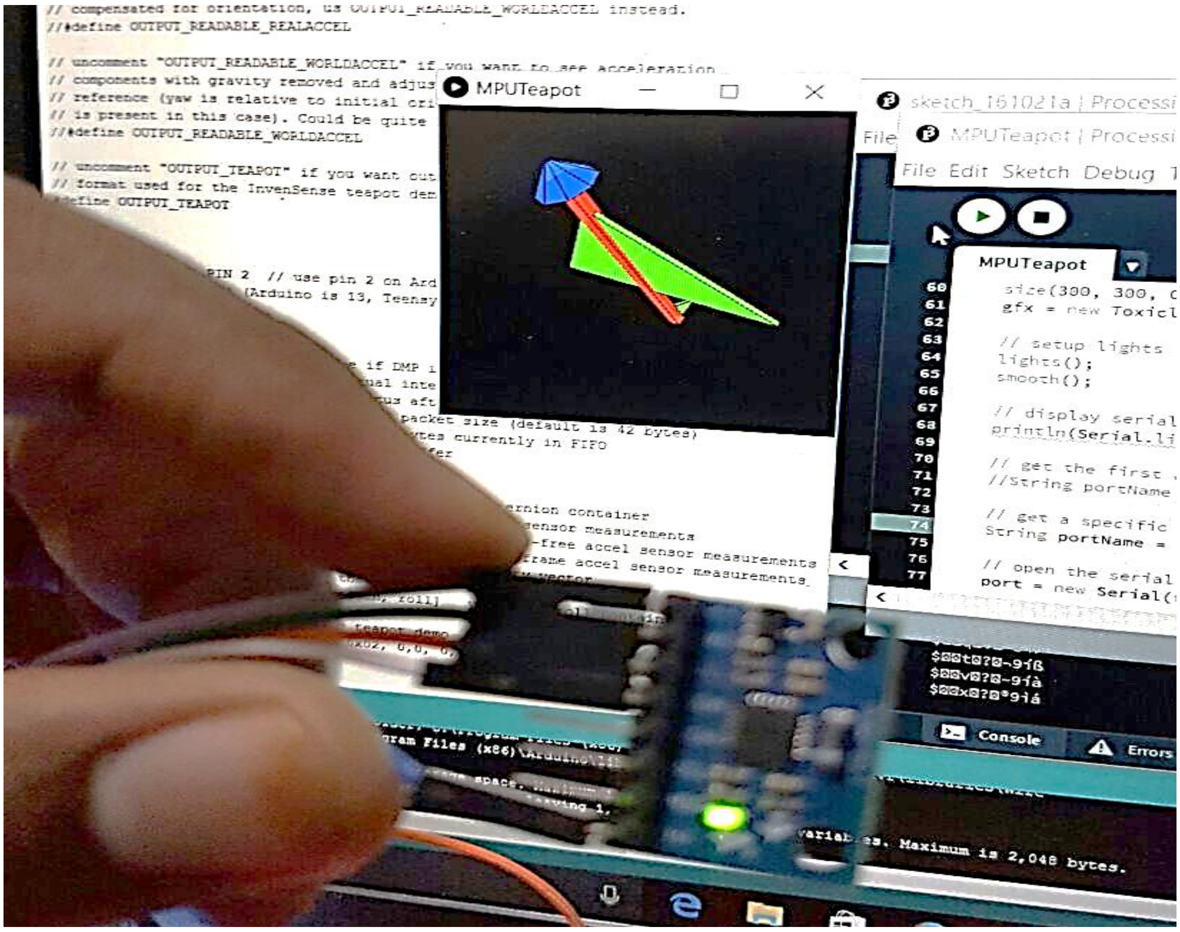
```
\Arduino\libraries\MPU6050 (legacy)
e space. Maximum is 32,256 bytes.
c memory, leaving 1,440 bytes for local variables. Maximum is 2,048 bytes.
```

Figure(32):   Simulation of GY-521 Through Teapot

Processing software was used, an open source programming language and environment (very similar to the Arduino IDE) that is particularly well suited for data visualization.

 modified the Arduino sketch to send the processed sensor and filtered data through the serial port, and wrote a Processing sketch to show the sensor and filter output as applied to three 3-D rectangles.

This figure shows the momentarily rolling motion which is in this case is rolling towards the left with accordance to the leading part which is pointing forwards.



```
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjus
// reference (yaw is relative to initial ori
// is present in this case). Could be quite
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want out
// format used for the InvenSense teapot dem
#define OUTPUT_TEAPOT
```

```
size(300, 300, 
gfx = new Toxicl

// setup lights
lights();
smooth();

// display serial
println(Serial.li

// get the first 
//String portName

// get a specific
String portName =

// open the serial
port = new Serial(
```

Figure (33): rotation motion about lateral axis

In this case the rotation was mostly about the Lateral axis creating pitching motion (pitch-up in this figure) which is successfully mimicked by the GY-521

Chapter Five: Conclusion and Recommendation

5.1 Conclusion

The Arduino IDE and Arduino microcontroller were quite sufficient to represent a simple and intuitive navigation system, to understand the way that a real inertial navigation system operates, the Simulation was handled using the Teapot add-on of the Processing Software, and the result was displayed in LCD, and the Whole system delivered a good chance of experiencing the Inertial navigation System starting from the calibration of sensors till reaching the final step which is Taking decision upon the end user display.

5.2 Recommendation and Future work

As a result of viewing and experiencing the inertial navigation system, considering that the system represents the motion around six axises of freedom, adding the Compass will cover more three axes showing the course of a certain vehicle which help in creating a comprehensive standalone system.

Furthermore, comparing the results obtained from this simplified inertial navigation system with results fetched from a real implemented system to calculate the reliability of the system upon the error rate of adjusting and calibrating the sensors.

Also for augmenting the ability of monitoring the results, the use of graphical LCD is considerable, to help deliver a clear way to build decisions upon eye monitoring or digital recognition.

Reference

[1] Titterton, David, and John L. Weston. *Strap down inertial navigation technology*. Vol. 17. IET, 2004.

[2] A Design Project Report Presented to the Engineering Division of the Graduate School of Cornell University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering (Electrical) by Maksim Eskin Project Advisor: Bruce Land Degree Date: January 2006

[3] Aircraft Digital Electronic and Computer Systems: Principles, Operation and Maintenance First edition 2007

[4] MEMS ACCELEROMETERS Author: Matej Andrejašiˇc Mentor: doc. dr. Igor Poberaj, Marec 2008

APPENDIX I:

MPU6050 Calibration:

```
// I2Cdev and MPU6050 must be installed as libraries, or else
the .cpp/.h files

// for both classes must be in the include path of your project

#include "I2Cdev.h"


#include "MPU6050_6Axis_MotionApps20.h"


// Arduino Wire library is required if I2Cdev
I2CDEV_ARDUINO_WIRE implementation

// is used in I2Cdev.h

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

    #include "Wire.h"

#endif


// class default I2C address is 0x68

// specific I2C addresses may be passed as a parameter here

// AD0 low = 0x68 (default for SparkFun breakout and InvenSense
evaluation board)

// AD0 high = 0x69

MPU6050 mpu;

//MPU6050 mpu(0x69); // <-- use for AD0
```

```
/*
=====================================================================
===========

   NOTE: In addition to connection 3.3v, GND, SDA, and SCL,
this sketch

   depends on the MPU-6050's INT pin being connected to the
Arduino's

   external interrupt #0 pin. On the Arduino Uno and Mega
2560, this is

   digital I/O pin 2.



   For the Galileo Gen1/2 Boards, there is no INT pin support.
Therefore

   the INT pin does not need to be connected, but you should
work on getting

   the timing of the program right, so that there is no buffer
overflow.

*
=====================================================================
=========== */



/*
=====================================================================
===========

   NOTE: Arduino v1.0.1 with the Leonardo board generates a
compile error

   when using Serial.write(buf, len). The Teapot output uses
this method.

   The solution requires a modification to the Arduino
USBAPI.h file, which

   is fortunately simple, but annoying. This will be fixed in
the next IDE

   release. For more info, see these links:
```

```
   http://arduino.cc/forum/index.php/topic,109987.0.html

   http://code.google.com/p/arduino/issues/detail?id=958

*
=====================================================================
====== */



#define OUTPUT_READABLE_YAWPITCHROLL


// Unccomment if you are using an Arduino-Style Board

// #define ARDUINO_BOARD


// Uncomment if you are using a Galileo Gen1 / 2 Board

#define GALILEO_BOARD


#define LED_PIN 13      // (Galileo/Arduino is 13)

bool blinkState = false;


// MPU control/status vars

bool dmpReady = false;  // set true if DMP init was successful

uint8_t mpuIntStatus;   // holds actual interrupt status byte from
MPU

uint8_t devStatus;      // return status after each device
operation (0 = success, !0 = error)

uint16_t packetSize;    // expected DMP packet size (default is 42
bytes)

uint16_t fifoCount;     // count of all bytes currently in FIFO

uint8_t fifoBuffer[64]; // FIFO storage buffer
```

```
// orientation/motion vars

VectorFloat gravity;    // [x, y, z]            gravity
vector

Quaternion q;           // [w, x, y,
z]         quaternion container

float euler[3];         // [psi, theta, phi]    Euler
angle container

float ypr[3];           // [yaw, pitch,
roll]   yaw/pitch/roll container and gravity vector





//
=======================================================
=========

// ===               INTERRUPT DETECTION
ROUTINE                 ===

//
=======================================================
=========



// This function is not required when using the Galileo

volatile bool mpuInterrupt = false;     // indicates
whether MPU interrupt pin has gone high

void dmpDataReady() {

    mpuInterrupt = true;

}



 //
```

```
================================================================
// ===                      INITIAL SETUP                      ===
// ================================================================


void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        int TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    Serial.begin(115200);
    while (!Serial);

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(F("MPU6050 connection "));
    Serial.print(mpu.testConnection() ? F("successful") :
F("failed"));
```

```
    // wait for ready

    Serial.println(F("\nSend any character to begin DMP
programming and demo: "));

    while (Serial.available() && Serial.read()); //
empty buffer

    while (!Serial.available());                 //
wait for data

    while (Serial.available() && Serial.read()); //
empty buffer again



    // load and configure the DMP

    Serial.println(F("Initializing DMP..."));

    devStatus = mpu.dmpInitialize();



    // supply your own gyro offsets here, scaled for
min sensitivity

    mpu.setXGyroOffset(220);

    mpu.setYGyroOffset(76);

    mpu.setZGyroOffset(-85);

    mpu.setZAccelOffset(1788); // 1688 factory default
for my test chip



    // make sure it worked (returns 0 if so)

    if (devStatus == 0) {

        // turn on the DMP, now that it's ready

        Serial.println(F("Enabling DMP..."));

        mpu.setDMPEnabled(true);

  // enable Arduino interrupt detection
Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));

        attachInterrupt(0, dmpDataReady, RISING);

        mpuIntStatus = mpu.getIntStatus();



        // set our DMP Ready flag so the main loop()
function knows it's okay to use it

        Serial.println(F("DMP ready! Waiting for first
interrupt..."));

        dmpReady = true;



        // get expected DMP packet size for later
comparison

        packetSize = mpu.dmpGetFIFOPacketSize();

    } else {

        // ERROR!

        // 1 = initial memory load failed

        // 2 = DMP configuration updates failed

        // (if it's going to break, usually the code
will be 1)

        Serial.print(F("DMP Initialization failed (code
"));

        Serial.print(devStatus);

        Serial.println(F(")"));

    }

   // configure LED for output

    pinMode(LED_PIN, OUTPUT);

}
```

```
//
================================================================
=======

// ===                      MAIN PROGRAM
LOOP                       ===

//
================================================================
=======


void loop() {

    // if programming failed, don't try to do anything

    if (!dmpReady) return;


    // wait for MPU interrupt or extra packet(s)
available


    #ifdef ARDUINO_BOARD

        while (!mpuInterrupt && fifoCount < packetSize) {

        }

    #endif


    #ifdef GALILEO_BOARD

        delay(10);

    #endif


    // reset interrupt flag and get INT_STATUS byte

    mpuInterrupt = false;
```

```
mpuIntStatus = mpu.getIntStatus();


    // get current FIFO count

    fifoCount = mpu.getFIFOCount();


    // check for overflow (this should never happen
unless our code is too inefficient)

    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {

        // reset so we can continue cleanly

        mpu.resetFIFO();

        Serial.println(F("FIFO overflow!"));


    // otherwise, check for DMP data ready interrupt
(this should happen frequently)

    } else if (mpuIntStatus & 0x02) {

        // wait for correct available data length,
should be a VERY short wait

        while (fifoCount < packetSize) fifoCount =
mpu.getFIFOCount();


        // read a packet from FIFO

        mpu.getFIFOBytes(fifoBuffer, packetSize);


        // track FIFO count here in case there is > 1
packet available

        // (this lets us immediately read more without
waiting for an interrupt)

        fifoCount -= packetSize;
        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```