ACKNOWLEDGEMENT

We are greatly indebted to the authorities of SR International Institute of Technology, Rampally(V), Keesara(M), R.R. Dist, for providing us the necessary facilities to successfully carry out this major project work titled "**TELEOPERATION OF A 4-DOF HUMANOID ARM USING HUMAN ARM ACCELERATIONS**".

We express our deep sense of gratitude to our principal **Dr. Sravan Kumar**, for granting us permission to work with necessary infrastructure to complete our major project.

We express our deep sense of gratitude to **Dr. Sravan Kumar** Associate Professor &HOD, Engineering department, SR International Institute of Technology for her invaluable help and support which helped us a lot in successfully completing our major project.

We express our solicit gratitude to Mrs. **Ujwala**, Associate professor, and other faculty members of the department of Computer Science Engineering for being a constant source of encouragement and inspiration throughout our major project work.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set of goals.

ANURAG AKELLA(18VF1M3303)

ANIREDDY SUJIT REDDY(18VF1M3306)

ABSTRACT

The paper concentrates on the development and control of the humanoid robot arm named ‘Jarvis’, intended for operation in a dynamic unstructured environment. The main scope of the project is to carry out research on control and operation of a humanoid 4-DOF robot using human arm accelerations, wirelessly and using it as a slave, for a master slave system, controlled by a wearable master device.

Humanoid robotics is a challenging field. To achieve human like behavior, humanoid robots not only have to feature human-like physical appearance but, more importantly, they must possess human-like characteristics regarding motion, communication and intelligence. In this project, we try and recreate a humanoid-arm skeleton, up to the wrist, and achieve wireless master-slave control using a various sensors.

Telerobotics usually refers to the remote control of robotic systems over a wireless link. This project also aims at achieving tele-operation, where the master in a master-slave system would be wireless and wearable, communicating over Bluetooth, with a host computer that could deocde and send information to the actual humanoid arm.

A robot, of any kind can be modified and/or upgraded with no limits and Jarvis, is no exception. Currently, it can follow arm movements, but in the future, additional degrees of freedom could be added, including an end effector which can possibly carry out complex tasks, similar to how a human would.

1. INTRODUCTION

1.1 Overview

More and more robots are being sent to distant environments to perform a variety of tasks, often also to places where a human cannot survive. From space exploration to nuclear spill clean-up, robots are being used. In most of these cases the robots are controlled remotely. Although these systems are able to perform interesting tasks, depending on the situation, the tasks might become increasingly complex. To tele-operate intricate systems requires extensive training. Even then, efficient operation is not possible in most cases and it is not clear how to operate these complex, remote robots robustly and safely. Furthermore these control interface systems should be intuitive and easy to use. In situations like these, a Master-Slave robotic system would be quite useful. Since the controls are based on human movements, the controls are intuitive and would eliminate the need of having to learn controls. The only limitation in such cases would be reliability of the mechanical system and the latency.

Telerobotics, usually refers to the control of a robotic system over a wireless connection. First use of telerobotics dates back to the first robotic space probes used during the 20-th century. Remote manipulators are nowadays commonly used to handle radioactive materials in nuclear power plants and waste management. In the space exploration scenario tele-operated mobile robots, such as, the Russian lunar and NASA's Martian rovers, are routinely used. More complex robotic systems have only recently seen use in tele-operation settings. One of these systems, NASA's Robonaut, was the first humanoid robot in space. It is currently on-board the International Space Station and used for tests on how tele-operation, as well as, semiautonomous operation, can assist the astronauts with certain tasks within the station. For the robot to safely interact with the environment sensory feedback is of critical importance. Even when tele-operated robots require the ability to perceive their surroundings, as it is generally not feasible or possible to transfer all the information back to the operator. Furthermore tests of controlling robots on the ground from the space station have been conducted to investigate how intermittant data transmission and the time-delay effect the operations. Recently quite a few

telepresence robot have entered the market, these though provide only a very limited, easy-to-control mobile robot. In addition tele-operated robots have been proposed to support medical staff. They could allow doctors to perform various medical check-ups in remote locations on Earth or space.

This project aims at using a gyroscope – accelerometer combination , along with a potentiometer and a low-cost wearable housing to create a transmitter with an Arduino NANO, that would transmit arm-joint angle information to a windows computer, through a wireless COM port, and the computer would then receive and send the data to an Arduino UNO, connected via USB, which would then decode the data structure and send appropriate pulses to the servo motors.

2. BACKGROUND AND RELATED WORK

2.1 The Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to 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. Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount.

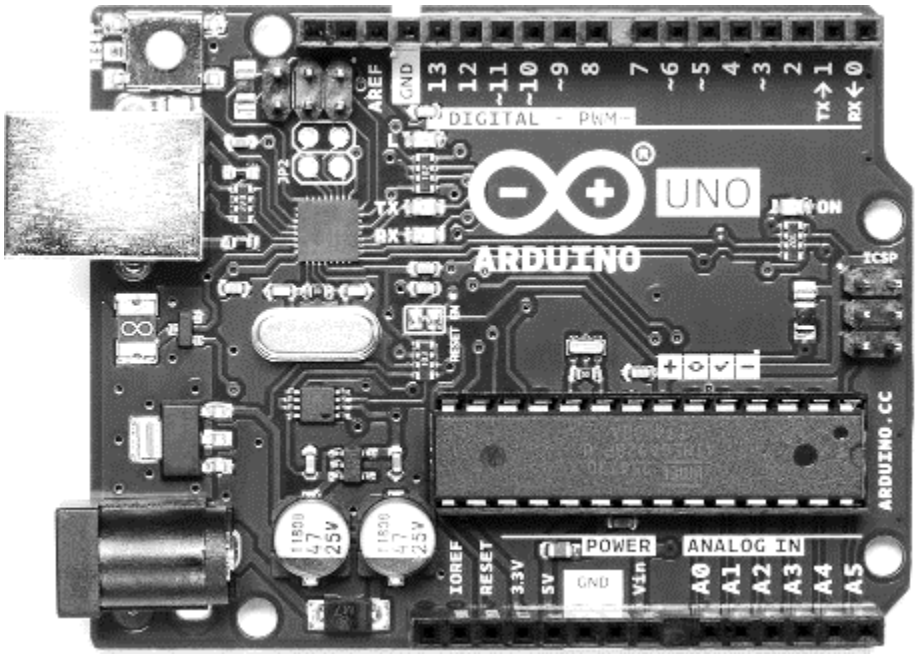


Figure 2.1 the Arduino UNO

For this project, we used the Arduino UNO R3 and an Arduino NANO, because of the ease of use and simplicity. The Arduinos helped us concentrate more on the arm and its design, rather than tinkering around, getting it to work with the sensors and peripherals.

Moreover, the Arduino supports a large array of sensors and peripherals, all supported and developed by a huge community around the world.

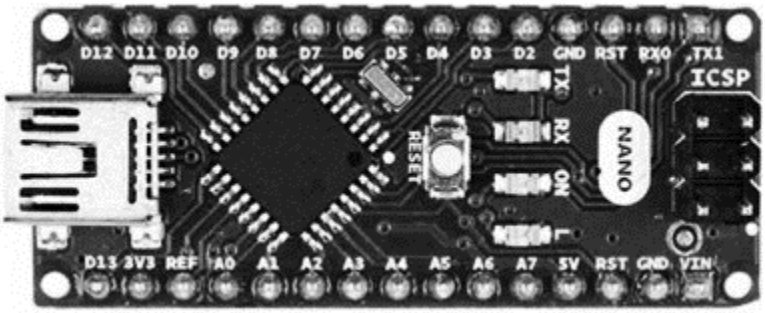


Figure 2.2 the Arduino NANO

2.1.1 Pin Description

The Arduino UNO consists of the Atmel ATmega328p, a 28 pin microcontroller. It consists of 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

Digital Inputs: It consists of 14 digital inputs/output pins, each of which provide or take up 40mA current. Some of them have special functions like pins 0 and 1, which act as Rx and Tx respectively, for serial communication, pins 2 and 3-which are external interrupts, pins 3,5,6,9,11 which provides PWM output and pin 13 where LED is connected.

Analog inputs: It has 6 analog input/output pins, each providing a resolution of 10 bits. The input values can range from 0 to 1023, depending on the sensor reading. Common

peropherals like potentiometers and slider potentiometers use the analog inputs to send values to the Arduino.

Reset: It resets the microcontroller when low.

The pin description of the Arduino is as shown figure 2.1.1:

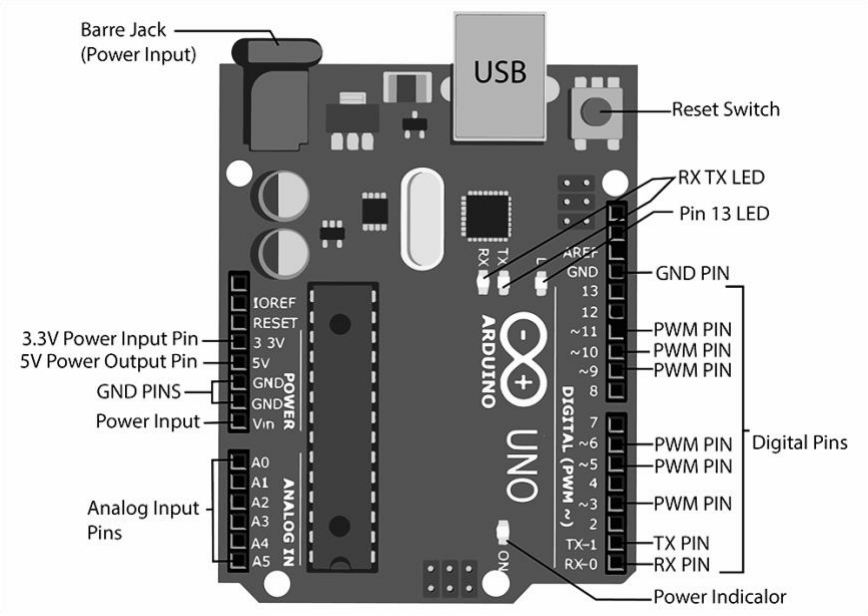


Figure 2.1.1 the Arduino UNO's pinout

The Nano is inbuilt with the ATmega328P microcontroller, same as the Arduino UNO. The main difference between them is that the UNO board is presented in PDIP (Plastic Dual-In-line Package) form with 30 pins and Nano is available in TQFP (plastic quad flat pack) with 32 pins. The extra 2 pins of Arduino Nano serve for the ADC functionalities, while UNO has 6 ADC ports but Nano has 8 ADC ports. The Nano board doesn't have a DC power jack as other Arduino boards, but instead has a mini-USB port. This port is used for both programming and serial monitoring. The fascinating feature in Nano is that it will

choose the strongest power source with its potential difference, and the power source selecting jumper is invalid.

The pin description of the Arduino is as shown in figure 2.1.2.

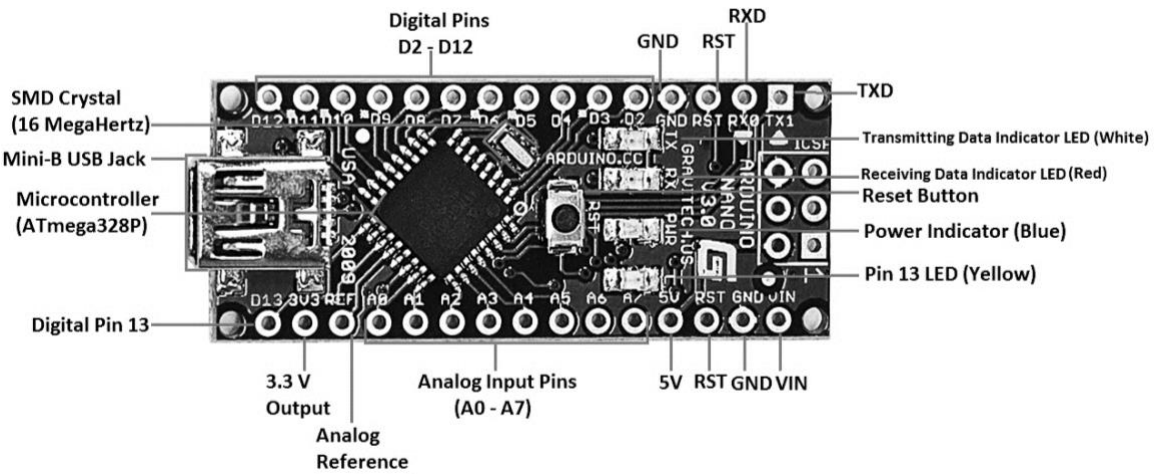


Figure 2.1.2 the Arduino UNO's pinout

2.1.2 Architecture

The Arduino UNO's The Atmel ATmega328/P is a low-power CMOS 8-bit microcontroller based on the enhanced **RISC** (reduced instruction set computer) architecture. The same chip is used on the Arduino NANO, but is presented in a different package, the UNO in PDIP and the NANO in TQFP. In Order to maximize performance and parallelism, the AVR uses **Harvard** architecture – with separate memories and buses for program and data.

Instructions placed in the program memory are executed with a single level of pipelining,

and the clock is controlled by an external **16MHz Crystal Oscillator**.

The architecture is shown in figure 2.1.3

The basic working of CPU of ATmega328

1. The data is uploaded in serial via the port (being uploaded from the computer's Arduino IDE). The data is decoded and then the instructions are sent to **instruction register** and it decodes the instructions on the same clock pulse.
2. On the next clock pulse the next set of instructions are loaded in instruction register.
3. In general purpose registers the registers are of 8-bit but there are 3 16-bit registers also.

a. 8-bit registers are used to store data for normal calculations and results.
b. 16-bit registers are used to store data of timer counter in 2 different register. Eg. X-low & X-high. They are fast, and are used to store specific hardware functions.

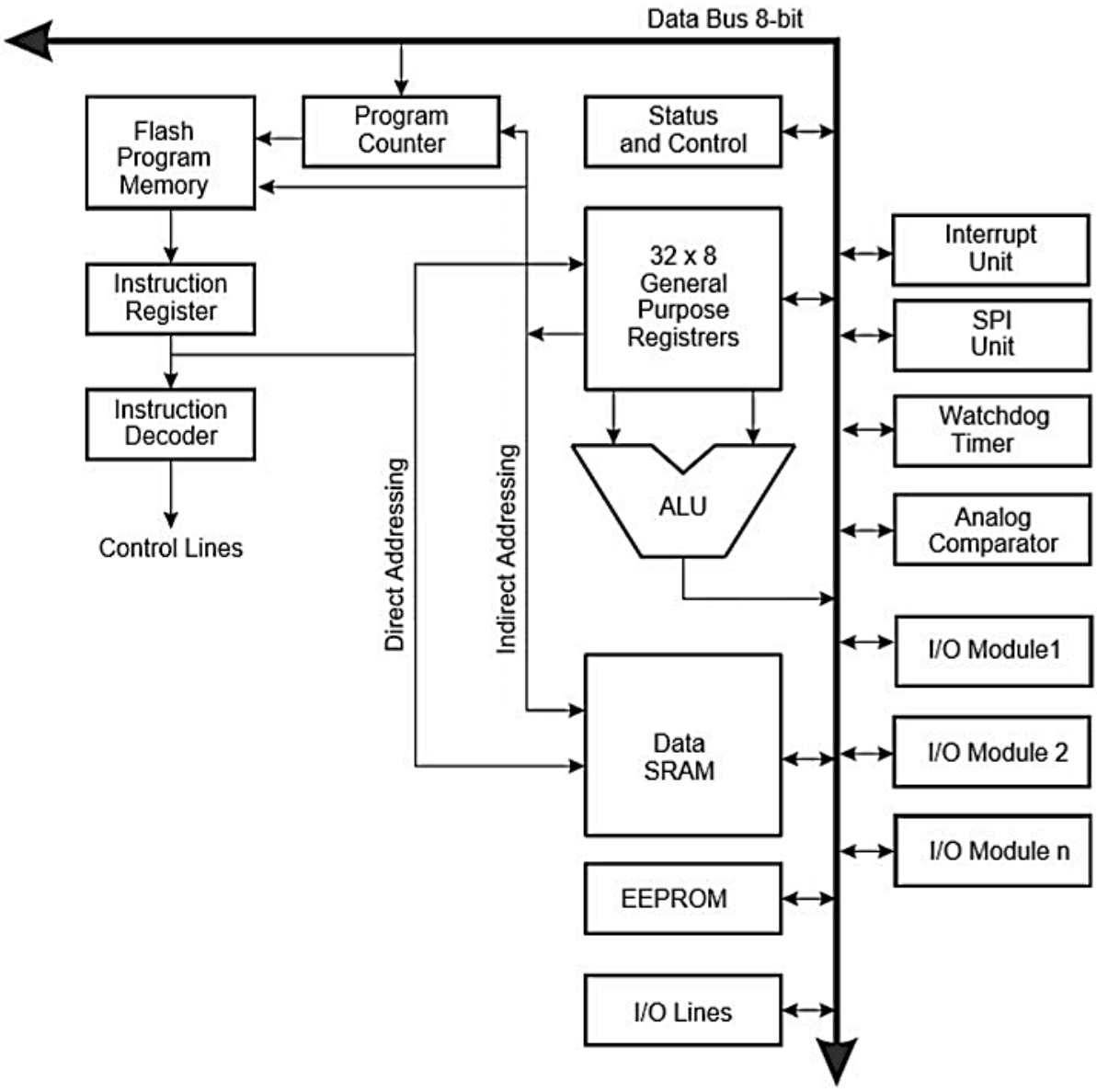


Figure 2.1.3 Atmel ATMega328P's architecture

2.2 Servos

Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds. Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board.

Servos are controlled by sending an electrical pulse of variable width, or **pulse width modulation** (PWM), through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. A servo motor can usually only turn 90° in either direction for a total of 180° movement. The motor's neutral position is defined as the position where the servo has the same amount of potential rotation in the both the clockwise or counter-clockwise direction. The PWM sent to the **motor** determines position of the shaft, and based on the duration of the pulse sent via the control wire; the **rotor** will turn to the desired position.

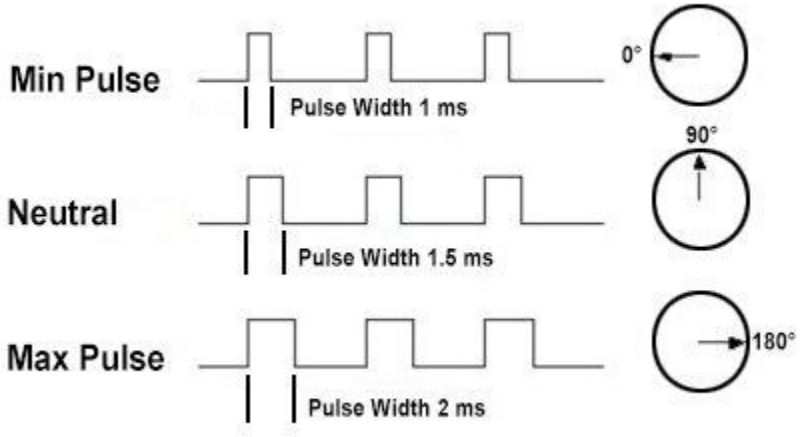


Figure 2.2.1 Variable Pulse width control servo position

When these servos are commanded to move, they will move to the position and hold that position. If an external force pushes against the servo while the servo is holding a position,

the servo will resist from moving out of that position. The maximum amount of force the servo can exert is called the **torque rating** of the servo. Servos will not hold their position forever though; the position pulse must be repeated to instruct the servo to stay in position.

2.3 The InvenSense MPU-6050

The MPU-6050 is an integrated 6-axis Motion tracking device that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor™ (DMP) all in a small 4x4x0.9mm package. With its dedicated I2C sensor bus, it directly accepts inputs from an external 3-axis compass to provide a complete 9-axis output.

The MPU-6050 is also designed to interface with multiple noninertial digital sensors, such as pressure sensors, on its auxiliary I^2C port. The MPU-60X0 features three 16-bit analog-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 ±2g, ±4g, ±8g, and ±16g.

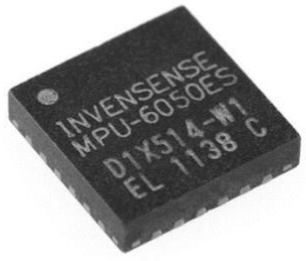


Figure 2.3.1 the MPU-6050 (QFN package)

2.3.1 MEMS Sensors

MEMS stands for micro electro mechanical system and applies to any sensor manufactured using microelectronic fabrication techniques. These techniques create

mechanical sensing structures of microscopic size, typically on silicon. When coupled with microelectronic circuits, MEMS sensors can be used to measure physical parameters such as acceleration.

The Accelerometer: The sensing element in MEMS VC accelerometers is comprised of a micro-machined proof mass that is suspended between two parallel plates. The mass is suspended on flexures that are attached to a ring frame. This configuration forms two air gap capacitors between the proof mass and upper and lower plates. As the proof mass moves when acceleration is applied, one air gap decreases and the other gap increases creating a change in capacitance proportional to acceleration.

The upper and lower plates are laminated to the proof mass sensing element with a glass bond. This creates a hermetic enclosure for the proof mass and provides mechanical isolation and protection.

A selection of full scale measurement ranges are attained by modifying the stiffness of the suspension system of the proof mass. A high natural frequency is accomplished through the combination of a lightweight proof mass and suspension stiffness. Ruggedness is enhanced through the use of mechanical stops on the two outer wafers to restrict the travel of the proof mass.

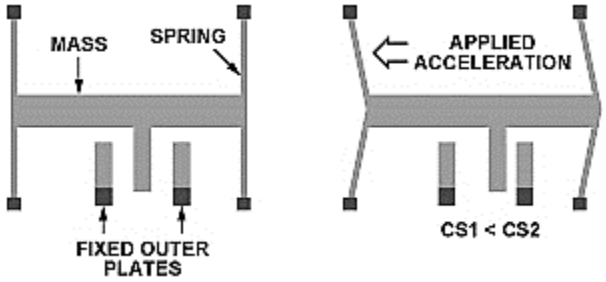


Figure 2.3.2 Internal structure of a MEMS accelerometer

The Gyroscope: MEMS gyroscopes measure angular rate by means of Coriolis acceleration. The Coriolis Effect can be explained as follows, starting with Figure 1. Consider yourself standing on a rotating platform, near the center. Your speed relative to the ground is shown as the blue arrow lengths. If you were to move to a point near the outer edge of the platform your speed would increase relative to the ground, as indicated by the longer blue arrow. The rate of increase of your tangential speed, caused by your radial velocity, is the Coriolis acceleration.

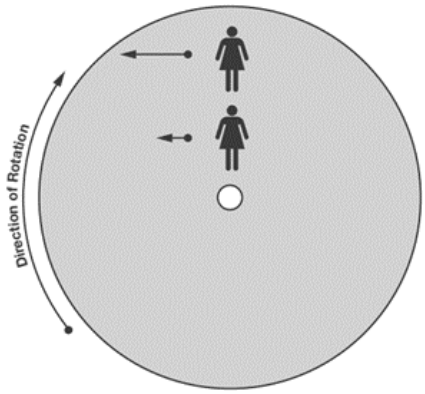


Figure 2.3.3 Coriolis acceleration example: A person moving northward toward the outer edge of a rotating platform must increase the westward speed component (blue arrows) to maintain a northbound course. The acceleration required is the Coriolis acceleration.

To measure the Coriolis acceleration, the frame containing the resonating mass is tethered to the substrate by springs at 90° relative to the resonating motion, as shown in Figure 3. This figure also shows the Coriolis sense fingers that are used to sense displacement of the frame through capacitive transduction in response to the force exerted by the mass.

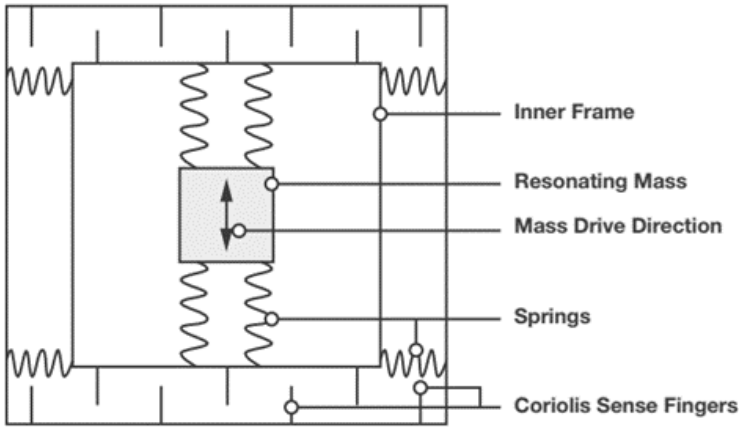


Figure 2.3.4. Schematic of the gyroscope's mechanical structure.

2.3.2 The GY-521 Breakout Board

The GY-521 module is a breakout board for the MPU-6050 MEMS (Microelectromechanical systems) that features a 3-axis gyroscope, a 3-axis accelerometer, a digital motion processor (DMP), and a temperature sensor. The digital motion processor can be used to process complex algorithms directly on the board. Usually, the DMP processes algorithms that turn the raw values from the sensors into stable position data.

Figure 2.3.5 Shows the pinout of the GY-521.

The GY-521 uses the popular I^2C (Inter Integrated Communication) protocol to communicate with the Arduino. The SCA and SDL lines are connected to the I^2C lines A4 and A5 pins of the Arduino. The Wire Library is used to communicate with the IMU.

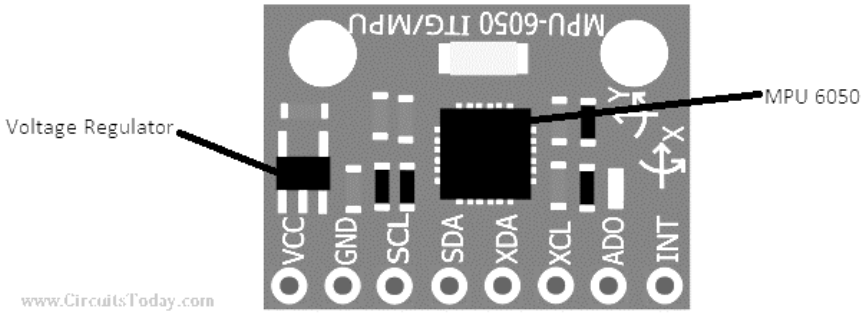


Figure 2.3.5. Pinout for GY-521

2.4 The HC-06 Bluetooth Module

The HC-06 is a Bluetooth module designed for establishing short range wireless data communication between two microcontrollers or systems. The module works on Bluetooth 2.0 communication protocol and it can only act as a slave device. This is cheapest method for wireless data transmission and more flexible compared to other methods and it even can transmit files at speed up to 2.1Mb/s.

HC-06 uses frequency hopping spread spectrum technique (FHSS) to avoid interference with other devices and to have full duplex transmission. The device works on the frequency range from 2.402 GHz to 2.480GHz.

The communication with this HC-06 module is done through UART interface. The data is sent to the module or received from the module though this interface. So we can connect

the module to any microcontroller or directly to PC which has RS232 port (UART interface). A typical interface circuit of the module to an Arduino is shown in figure 2.4.1

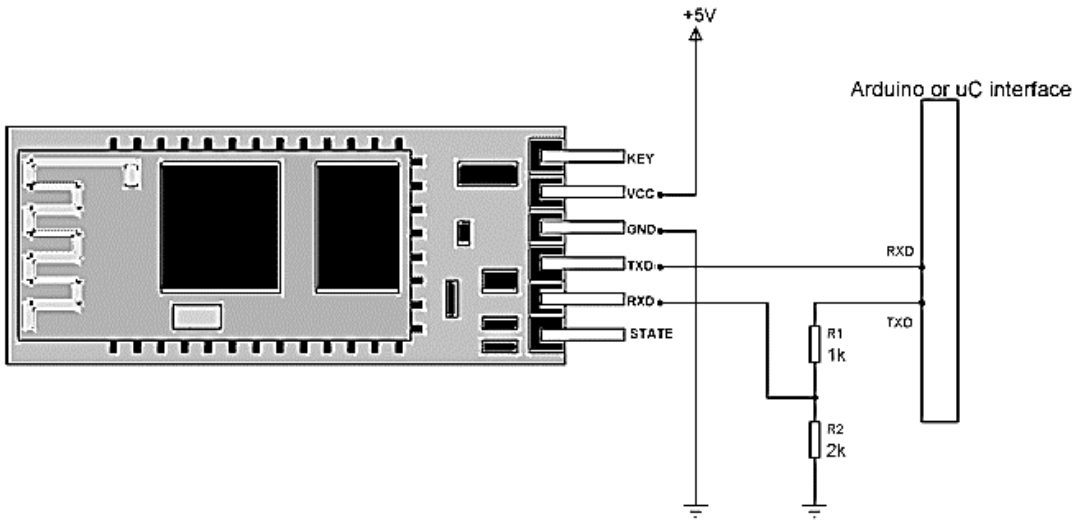


Figure 2.4.1 The HC-06, and its pin configuration

2.5 Potentiometers

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

The measuring instrument called a potentiometer is essentially a voltage divider used for measuring electric potential (voltage); the component is an implementation of the same principle, hence its name.

Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment. Potentiometers operated by a mechanism can be used as position transducers, for example, in a joystick. Potentiometers are rarely used to directly control significant power (more than a watt), since the power dissipated in the potentiometer would be comparable to the power in the controlled load.

Potentiometers consist of a resistive element, a sliding contact (wiper) that moves along the element, making good electrical contact with one part of it, electrical terminals at each end of the element, a mechanism that moves the wiper from one end to the other, and a housing containing the element and wiper.

See drawing. Many inexpensive potentiometers are constructed with a resistive element *(B)* formed into an arc of a circle usually a little less than a full turn and a wiper *(C)* sliding on this element when rotated, making electrical contact. The resistive element can be flat or angled. Each end of the resistive element is connected to a terminal *(E, G)* on the case. The wiper is connected to a third terminal *(F)*, usually between the other two. On panel potentiometers, the wiper is usually the center terminal of three. For single-turn potentiometers, this wiper typically travels just under one revolution around the contact. The only point of ingress for contamination is the narrow space between the shaft and the housing it rotates in.

Figure 2.5.1 Shows the insides of a common potentiometer.

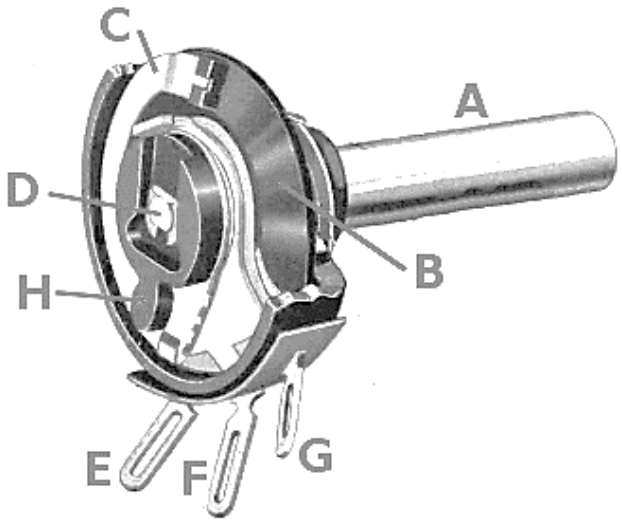


Figure 2.5.1 A potentiometer

3. SETUP AND DESIGN

The whole Master-Slave system can be broken down into a Transmitter (Master) and a Receiver (Slave). The Transmitter communicates with the slave, via a webapp, written in python using the Django framework. After multiple iterations of designing, testing and execution, we finally stopped at the final design, which includes a simple wooden skeleton for the transmitter. The transmitter is fitted with all the necessary sensors at estimated positions.

The receiver on the other hand, uses low-cost PVC parts, which mimic a human arm and is connected to the Arduino. The Arduino uses USB serial communication to communicate with a windows PC, which provides it with a data structure that contains the servo angles.

Figure 3.1 Shows a block diagram of the project

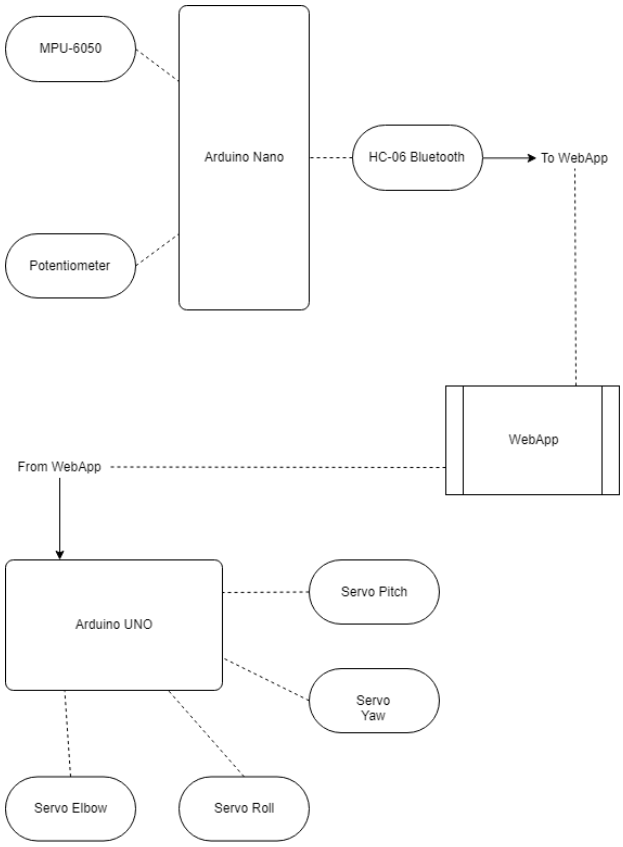


Figure 3.1 The Block Diagram

3.1 The Transmitter

The Transmitter is made of two wooden planks, pivoted at one end with a potentiometer. This acts as the elbow joint and the Arduino reads the potentiometer values through an analog pin.

The accelerometer, records and outputs data in a 4 Dimensional unit, commonly known as a quaternion. An MPU6050 Library converts the data into usable Euler angles, which then would be mapped into an angle and packed into a data structure.

Figure 3.1.1 The Transmitter

3.2 The Receiver

The Receiver is an Arduino UNO, that takes in the data structure from the receiver, and writes the angle to the Servo. The connections are fairly simple. The arduino communicates with the web app using a serial COM port via the USB.

The arm was designed with the human arm in mind. The Pitch axis affects the rest of the axes, with yaw directly related to the roll. The shoulder setup when simplified, forms

a gimbal like structure. A simple extension to the shoulder forms the upper arm and a pivot servo attaches the lower arm to the upper arm, forming an elbow joint. To prevent over-complication, we chose to make an ‘arm’ and not the complete upper limb, which includes the wrist and fingers too. That could be implemented in a future version of the project.

Figure 3.2.1 shows the humanoid slave arm, detached from its base.

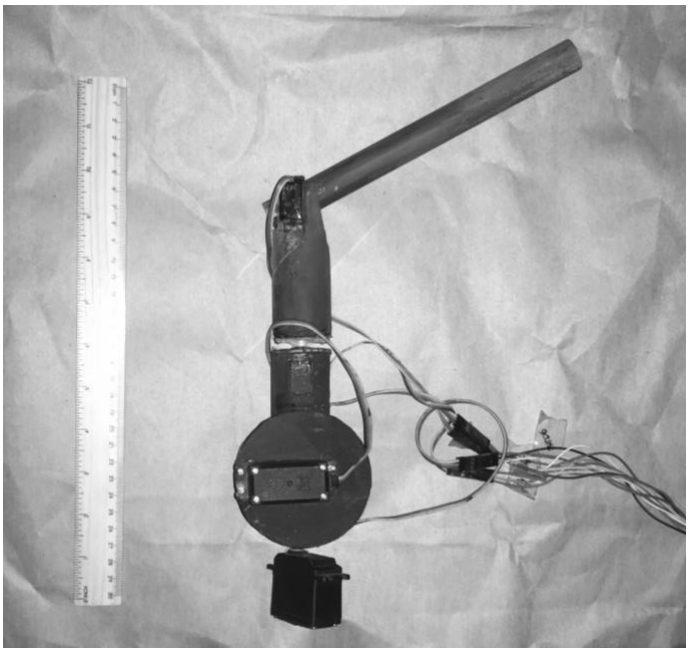


Figure 3.1.1 The Receiver

3.3 The Web App

The python webapp, uses the Django framework and runs a website that receives values from the transmitter and forwards it to the arduino. the arduino then uses a decoding algorithm to decode the string data to find X, Y, Z and E values, and writes them to the yaw, pitch, roll and elbow servos respectively. This happens in the loop, until the power is disconnected so the device continuously tracks arm movements.

In addition to the transmitter, the webapp also has local remote controls that can be used to control the arm's position. A simple web page with sliders sends servo data to the Arduino, using the same data structure, as discussed in section 3.4.

3.4 The Data Structure

The data structure is a simple string that contains the X, Y, Z and E values from the transmitter. The transmitter uses the '+' operator to concatenate the string, and sends it as serial data, via Bluetooth.

The receiver Arduino decodes the string using the default string library, finding the indices of X, Y, Z and E and extracting integer values from the string. Figure 3.4.1 shows the data structure, with explanation.

X{0}Y{1}Z{2}E{3}

Figure 3.4.1 The Data Structure

The {0}, {1}, {2}, and {3} are sensor readings, that have been constrained in code. Functions like charAt(), indexOf(), etc. were used to decode the string. The {} are replaced with sensor readings, while encoding.

4. SOFTWARE SPECIFICATIONS

4.1 Software Tools

Software tools helps in building the project as per the aim. Following are the software tools used for the project-

1. Arduino IDE
2. Django Web Framework

4.2 Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

Writing Sketches

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.

Verify

Checks your code for errors compiling it.

Upload

Compiles your code and uploads it to the configured board. See uploading below for details.

Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"

New Creates a new sketch.

Open Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the File | Sketchbookmenu instead.

Save Saves your sketch.

Serial Monitor Opens the serial monitor.

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

File

New Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.

1. Open Allows to load a sketch file browsing through the computer drives and folders.
2. Open Recent Provides a short list of the most recent sketches, ready to be opened.
3. Sketchbook shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
4. Examples Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
5. Close Closes the instance of the Arduino Software from which it is clicked.
6. Save saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
7. Saveas... Allows to save the current sketch with a different name.
8. Page Setup It shows the Page Setup window for printing.
9. Print Sends the current sketch to the printer according to the settings defined in Page Setup.
10. Preferences Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
11. Quit Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

1. Undo/Redo

   Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
2. Cut

   Removes the selected text from the editor and places it into the clipboard.
3. Copy

   Duplicates the selected text in the editor and places it into the clipboard.
4. Copy for Forum Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax colouring.

5. Copy as HTML Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
6. Paste

   Puts the contents of the clipboard at the cursor position, in the editor.
7. Select All Selects and highlights the whole content of the editor.
8. Comment/Uncomment

   Puts or removes the // comment marker at the beginning of each selected line.
9. Increase/Decrease Indent :Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
10. Find

    Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
11. Find Next: Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
12. Find Previous: Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

Sketch

1. Verify/Compile

   Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
2. Upload

   Compiles and loads the binary file onto the configured board through the configured Port.
3. Upload Using Programmer

   This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a Tools -> Burn Bootloader command must be executed.

4. Export Compiled Binary
   Saves a .hex file that may be kept as archive or sent to the board using other tools.
5. Show Sketch Folder
   Opens the current sketch folder.
6. Include Library
   Adds a library to your sketch by inserting #include statements at the start of your code. For more details, see libraries below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
7. Add File.
   Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right side o the toolbar.

Tools

1. Auto Format
   This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
2. Archive Sketch
   Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
3. Fix Encoding & Reload
   Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
4. Serial Monitor
   Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.
5. Board
   Select the board that you're using. See below for descriptions of the various boards.

6. Port

   This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.

7. Programmer

   For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're burning a bootloader to a new microcontroller, you will use this.

8. Burn Bootloader

   The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino or Genuino board but is useful if you purchase a new AT mega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the Boards menu before burning the bootloader on the target board. This command also set the right fuses.

Help

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

Find in Reference

This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

Sketchbook

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the File > Sketchbook menu or from the Open button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the Preferences dialog.

Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

Uploading

Before uploading your sketch, you need to select the correct items from the Tools > Board and Tools > Port menus. The boards are described below. On the Mac, the serial port is probably something like /dev/tty.usbmodem241 (for an Uno or Mega2560 or Leonardo) or /dev/tty.usbserial-1B1 (for a Duemilanove or earlier USB board), or/dev/tty.USA19QW1b1P1.1 (for a serial board connected with a Key span USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be /dev/ttyACMx ,/dev/ttyUSBx or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the Upload item from the Sketch menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the Sketch > Import Library menu. This will insert one or more #include statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its #include statements from the top of your code.

There is a list of libraries in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these instructions for installing a third-party library.

To write your own library.

Third-Party Hardware

Support for third-party hardware can be added to the hardware directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the hardware directory, then unzip the third-party platform into its own sub-directory. (Don't use "Arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

For details on creating packages for third-party hardware, see the Arduino IDE 1.5 3rd party Hardware specification.

Serial Monitor

Displays serial data being sent from the Arduino or Genuino board (USB or serial board). To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down that matches the rate passed to Serial.begin in your sketch. Note that on Windows, Mac or Linux, the Arduino or Genuino board will reset (rerun your sketch execution to the beginning) when you connect with the serial monitor.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the interfacing page for details).

Preferences

Some preferences can be set in the preferences dialog (found under the Arduino menu on the Mac, or File on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

Boards

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader.

4.3 The Django Web Framework

Django is a Python-based free and open-source web framework, which follows the model-template-view (MTV) architectural pattern.[5][6] It is maintained by the Django Software Foundation (DSF), an independent organization established as a 501(c)(3) non-profit.

Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself.[7] Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Some well-known sites that use Django include the Public Broadcasting Service, Instagram, Mozilla, *The Washington Times*, Disqus, Bit bucket, and Next-door. It was used on Pinterest, but later the site moved to a framework built over Flask.

Components

Despite having its own nomenclature, such as naming the callable objects generating the HTTP responses "views",[5] the core Django framework can be seen as an MVC architecture.[6] It consists of an object-relational mapper (ORM) that mediates between data models (defined as Python classes) and a relational database ("**M**odel"), a system for processing HTTP requests with a web templating system ("**V**iew"), and a regular-expression-based URL dispatcher ("**C**ontroller").

Also included in the core framework are:

- a lightweight and standalone web server for development and testing
- a form serialization and validation system that can translate between HTML forms and values suitable for storage in the database
- a template system that utilizes the concept of inheritance borrowed from object-oriented programming
- a caching framework that can use any of several cache methods

- support for middleware classes that can intervene at various stages of request processing and carry out custom functions
- an internal dispatcher system that allows components of an application to communicate events to each other via pre-defined signals
- an internationalization system, including translations of Django's own components into a variety of languages
- a serialization system that can produce and read XML and/or JSON representations of Django model instances
- a system for extending the capabilities of the template engine
- an interface to Python's built-in unit test framework
- Django REST framework is a powerful and flexible toolkit for building Web APIs.

Bundled applications

The main Django distribution also bundles a number of applications in its "contrib" package, including:

- an extensible authentication system
- the dynamic administrative interface
- tools for generating RSS and Atom syndication feeds
- a "Sites" framework that allows one Django installation to run multiple websites, each with their own content and applications
- tools for generating Google Sitemaps
- built-in mitigation for cross-site request forgery, cross-site scripting, SQL injection, password cracking and other typical web attacks, most of them turned on by default[18][19]
- a framework for creating GIS applications

Extensibility

Django's configuration system allows third party code to be plugged into a regular project, provided that it follows the reusable app conventions. More than 2500 packages are available to extend the framework's original behavior, providing solutions to issues the original tool didn't tackle: registration, search, API provision and consumption, CMS, etc.

This extensibility is, however, mitigated by internal components' dependencies. While the Django philosophy implies loose coupling,[22] the template filters and tags assume one engine implementation, and both the auth and admin bundled applications require the use of the internal ORM. None of these filters or bundled apps are mandatory to run a Django project, but reusable apps tend to depend on them, encouraging developers to keep using the official stack in order to benefit fully from the apps ecosystem.

Server arrangements

Django can be run in conjunction with Apache, Nginx using WSGI, Gunicorn, or Cherokee using flup (a Python module). Django also includes the ability to launch a FastCGI server, enabling use behind any web server which supports FastCGI, such as Lighttpd or Hiawatha. It is also possible to use other WSGI-compliant web servers. Django officially supports four database backends: PostgreSQL, MySQL, SQLite, and Oracle. Microsoft SQL Server can be used with django-mssql on Microsoft operating systems, while similarly external backends exist for IBM Db2, SQL Anywhereand Firebird. There is a fork named django-nonrel, which supports NoSQL databases, such as MongoDB and Google App Engine's Datastore.[30]

Django may also be run in conjunction with Jython on any Java EE application server such as GlassFish or JBoss. In this case django-jython must be installed in order to provide JDBC drivers for database connectivity, which also can provide functionality to compile Django in to a .war suitable for deployment.

Google App Engine includes support for Django version 1.x.xas one of the bundled frameworks.

.

4.4 Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

5. CODE

5.1 The Transmitter

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define INTERRUPT_PIN 2
#define LED_PIN 13
bool blinkState = false;
int buttonPin = 9;
int LED2 = 10;
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity ve
ctor
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone
 high
void dmpDataReady() {
    mpuInterrupt = true;
}
void setup() {
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000);
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif
    Serial.begin(115200);
    while (!Serial);
    mpu.initialize();
```

```
    pinMode(INTERRUPT_PIN, INPUT);
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU60
50 connection failed"));
    devStatus = mpu.dmpInitialize();
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788);
    if (devStatus == 0) {
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        mpu.setDMPEnabled(true);
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;
        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);
    pinMode(LED2, OUTPUT);
    pinMode(buttonPin, INPUT);
}
void loop() {
    if (!dmpReady) return;
    while (!mpuInterrupt && fifoCount < packetSize) {
        if (mpuInterrupt && fifoCount < packetSize) {
          // try to get out of the infinite loop
          fifoCount = mpu.getFIFOCount();
        }
    }
    mpuInterrupt = false;
```

```
    mpuIntStatus = mpu.getIntStatus();
    fifoCount = mpu.getFIFOCount();
    if(fifoCount < packetSize){
    }
    else if ((mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT)) || fifoCount >= 102
4) {
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));
    } else if (mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {
    while(fifoCount >= packetSize){
        mpu.getFIFOBytes(fifoBuffer, packetSize);
        fifoCount -= packetSize;
    }
        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            if(digitalRead(buttonPin) == HIGH){
              Serial.print("1");
              digitalWrite(LED_PIN, HIGH);
            }
            else {
              Serial.print("0");
              digitalWrite(LED_PIN, LOW);
            }
            Serial.print("X");
            Serial.print(round(constrain(map(ypr[0] * 180/M_PI, 0, 90, 5, 85), 5  , 85)
));
            //Serial.print(ypr[0] * 180/M_PI);
            //sl.write(round(constrain(map(ypr[0] * 180/M_PI, 0, 90, 0, 85), 0, 85)));
            Serial.print("Y");
            Serial.print(round(constrain(map(ypr[1] * 180/M_PI, 90, 0, 100, 180), 100,
180)));
            //Serial.print(ypr[1] * 180/M_PI);
            //sh.write(round(constrain(map(ypr[1] * 180/M_PI, -
90, 0, 100, 180), 100, 180)));
            Serial.print("Z");
            Serial.print(round(constrain(map(ypr[2] * 180/M_PI, 90, 0, 0, 100), 0, 100)
));
            //Serial.print(ypr[1] * 180/M_PI);
            //su.write(round(constrain(map(ypr[2] * 180/M_PI, -
90, 0, 0, 100), 0, 100)));
            Serial.print("E");
```

```
            Serial.println(round(constrain(map(analogRead(A0), 1024, 920, 0, 115), 0, 1
15)));
            //el.write(round(constrain(map(analogRead(A0), 1024, 920, 0, 115), 0, 115))
);
            delay(15);
        #endif
    }
}
```

5.2 The Receiver

```
       #include<Servo.h>

Servo sh, sl, su, elb;
String ds;
int xl, yl, zl ,el;
int X, Y, Z, E;
void setup() {
  Serial.begin(115200);
  sh.attach(9);
  sl.attach(10);
  su.attach(11);
  elb.attach(7);
  Serial.setTimeout(10);
}
void loop() {
    //b'X23Y180Z100E115\r\n'
    if(Serial.available()){
    ds = Serial.readString();
    xl = ds.i ndexOf('X');
    yl = ds.indexOf('Y');
    zl = ds.indexOf('Z');
    el = ds.indexOf('E');
    X = ds.substring(xl+1, yl).toInt();
    Y = ds.substring(yl+1, zl).toInt();
    Z = ds.substring(zl+1, el).toInt();
    E = ds.substring(el+1).toInt();
    sl.write(X);
    sh.write(Y);
    su.write(Z);
    elb.write(E);
    Serial.print("X: ");
    Serial.print(X);
    Serial.print(" Y: ");
    Serial.print(Y);
```

```
    Serial.print(" Z: ");
    Serial.print(Z);
    Serial.print(" E: ");
    Serial.println(E);
    delay(15);
    }
 }
```

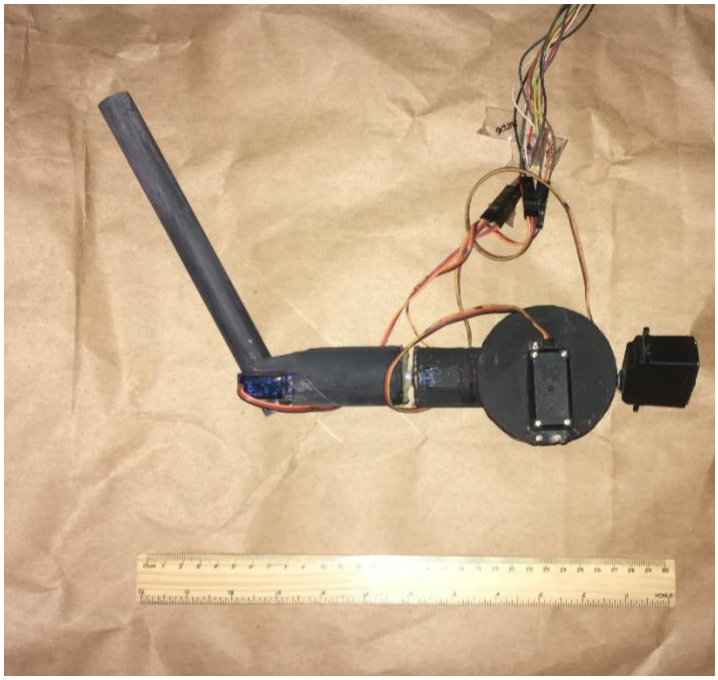
5.3 Python Web App

```
       import serial

import queue
import threading
import time
from django.shortcuts import render, HttpResponse

queue1 = queue.Queue(1000)
ports_open = False

def serial_read(s):
    while True:
        line = s.readline()
        queue1.put(line)

def serial_write(s1):
    while True:
        line = queue1.get(True, 1)
        s1.write(line)



def serial_read_c3(s):
    while True:
        line = s.readline()
        print(line.decode())

def runfriday():
    try:
        serial0 = serial.Serial('COM11', baudrate=115200, timeout=1);
        serial1 = serial.Serial('COM3', baudrate=115200, timeout=1);
        ports_open = True
    except:
        print('Could not open Ports. Please connect the arduino and Retry.' )
        ports_open = False
```

```
    if(ports_open):
        thread1 = threading.Thread(target=serial_read, args=(serial0,),)
        thread2 = threading.Thread(target=serial_write, args=(serial1,),)
        thread3 = threading.Thread(target=serial_read_c3, args=(serial1,),)
        thread1.setDaemon(True)
        thread2.setDaemon(True)
        thread3.setDaemon(True)
        thread1.start()
        thread2.start()
        thread3.start()
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            pass

if __name__ == '__main__':
    try:
        arg = sys.argv[1]
    except IndexError:
        arg = None
    runfriday()
```

6. RESULT

5.1 The Prototype



7. CONCLUSION AND FUTURE SCOPE

We've learnt a lot of things while working on the project. Things like quaternions, the Bluetooth technology, how MEMS sensors work and a lot more. A lot of effort was put into designing the arm mechanics, which was quite challenging. Especially replication of the ball and socket joint of a human arm, with no outside help of any kind. A humanoid robot of this kind can be used in practically every industry out there. The arm could help package and build goods in the manufacturing industry. It can help assemble electronic devices and appliances. It can help us work in places where it would generally be hard for humans to work. It can be used in dangerous places like nuclear power plants. It can also be used for education, in the medical field, by surgeons, etc.

7.1 Future Scope

Although we've worked hard on the project, a lot can be done, in the future. Based on where it is being used. An end effector of any kind can be attached to this robot, and a simple change in the source code can make this robot fully functional. For example, an electromagnetic attachment would make it capable of working in a factory, for example to lift things. A gripper can be attached, and it could be used in labs, small stores, etc. Similarly, after swapping the end effector for appropriate tools, the robot can be used in the medical field as well. In the future, the MPU-6050 can also be swapped with a more accurate and efficient sensor, and the servos can also be replaced with stepper motors for higher accuracy and torque. The arm can also be modified to help people with muscular/ nervous disabilities.

8. REFERENCES

Dzmitry Tsetserukou, Naoki Kawakami and Susumu Tachi Department of Information Physics and Computing, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan.

J. Leitner, M. Luciw, A. Forster, J. Schmidhuber ¨ Dalle Molle Institute for Artificial Intelligence (IDSIA) & Scuola universitaria professionale della Svizzera Italiana (SUPSI) & Universita della Svizzera italiana (USI), Switzerland

https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

https://www.djangoproject.com/

https://arduino.cc

https://ieeexplore.ieee.org/document/5358267

Sunatthra Jariyawattanakul, Thavida Maneewarn, Gesture based master-slave controller for a social humanoid robot, IEEE.

Wikipedia