International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 11 Nov 2021, pp: 180-189www.ijaem.net ISSN: 2395-5252

Hand Gestures Controller for Drone

Prakaas M, Pavithra N, Sunil Saravanan J, Suriya K, Veeramanikandan R

Hindusthan college of engineering and technology, Coimbatore.
Hindusthan college of engineering and technology, Coimbatore.
Hindusthan college of engineering and technology, Coimbatore.
Hindusthan college of engineering and technology, Coimbatore.
Corresponding author: Veeramanikandan R

Submitted: 01-11-2021 Revised: 06-11-2021 Accepted: 09-11-2021

INTRODUCTION:

Due to its adaptability in performing any type of airborne duty, a drone is one of the most complex flying devices. The gesture-based control interface has a lot of potential for more natural, intuitively comprehensible, and customizable human-machine interaction, and it can expand the capabilities of common graphical and command line interfaces that we use today with the mouse and keyboard.As a result, advanced hardware and software approaches to hand-gesture recognition are critical for a wide range of 3D applications, including computer and robot control, interaction with a computer-generated environment (virtual or augmented reality), sign language comprehension, gesture visualization, game control, and the enhancement of disabled people's communication abilities, among others. Hand Gesture Controlled Drone's primary goal is to eliminate the weighted Radio-Controller.The Radio-Controller, which can be worn in the hand and is used to control the drone's movement, flight, and other functions, is usually replaced by a glove (Transmitting Circuit). For peripheral memory allocation or, to put it another way, for the storage and installation of the interface, we used an Arduino uno in the transmitter. The interface was created in such a way that the drone may flip, accelerate, and change direction in microseconds. After then, the uno is directed to the Sensor (MPU 6050) and finally to the flight controller (APM).

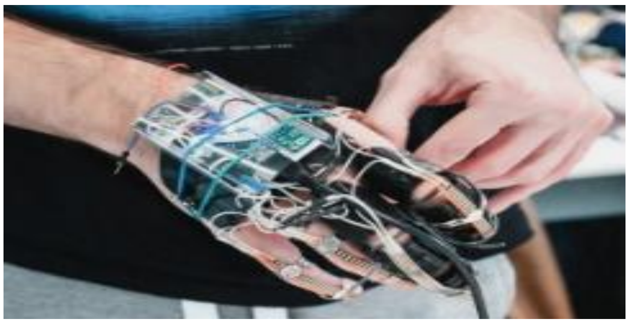
HAND CONTROLLER:

The hand controller is the feature that distinguishes our system from other quadcopter systems that use joysticks. This controller features a highly straightforward, basic technique of piloting, allowing even individuals with no prior expertise with UAVs to fly the system within an hour of acquiring it. To avoid using a big battery, the hand controller must be able to run for at least 10 minutes without needing to be recharged.
The hand controller will be able to accommodate most adult human hands and will be durable enough to survive a fall of 1.5 meters. Because speed is crucial in an emergency situation, any setup for the hand controller,
including calibrating the motion/position sensor, will take less than 5 minutes. To keep the pilot from being fatigued while piloting the quadcopter, the controller will be under 250 grammes in weight.

There will be specialized command such as,
- Takeoff
- Stop
- Hover

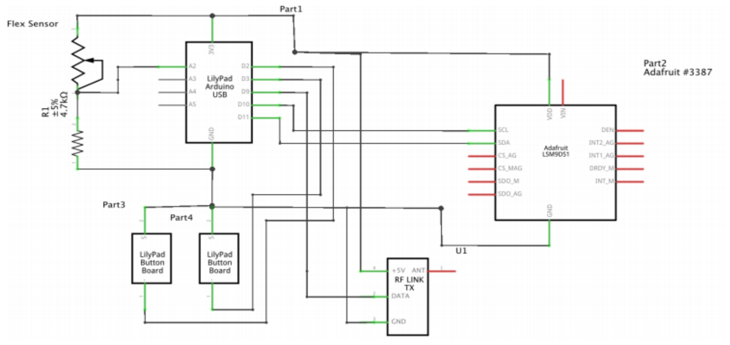
which will make complex quadcopter functions easier to understand. Specialized commands represented by buttons and a flex sensor should take precedence over movement controls while the pilot is piloting the quadcopter.

When the quadcopter is hovering, for example, the pilot can move their hand without sending the action to the quadcopter. Most crucially, the hand controller must be able to translate data from the movement/position sensor, buttons, and flex sensor into strings that meet the drone's chosen format.



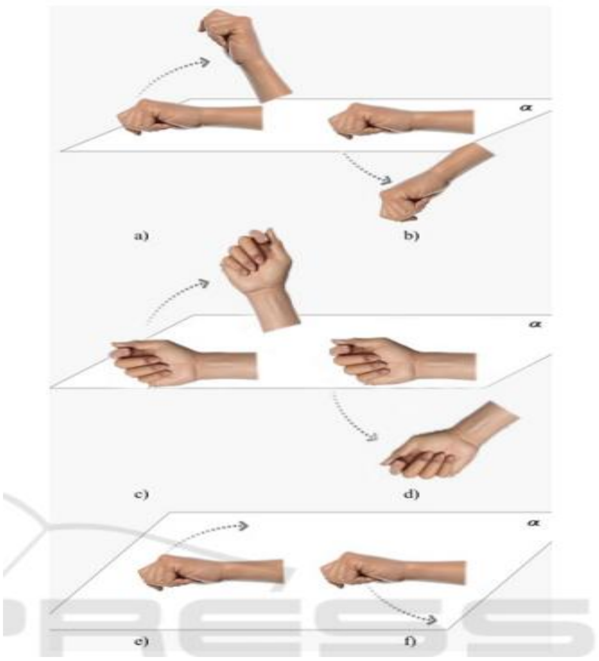
GESTURE DESIGN:

The throttle would be controlled by pressing a button and repeating the forward and backward commands, as seen in the previous design. The throttle was supposed to be controlled by the vertical movement of the hand, but we changed our minds. Because we couldn't acquire a constant signal from the gyroscope on the 9-DOF board, we decided against using the original design. Because the flex sensor's main duty is to initialize the takeoff function, we opted to use it to fix this problem.



SCHEMATIC OF HAND CONTROLLER:

If the flex sensor is engaged while the hand controller is activated and reading positional data, the throttle will increase until the flex sensor is unengaged or the hand is flipped, at which point the throttle will decrease. This will extend the range of the quadcopter without requiring the use of the hover special command.

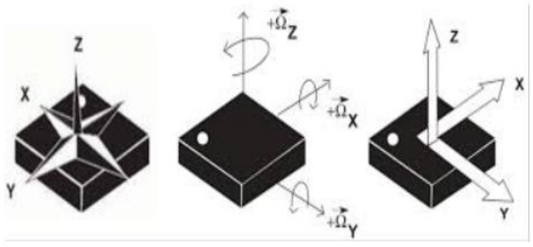


Gestures to control a quadcopter,
a) moves forward.
b) moves backward.
c) moves up.
d) moves down.
e) turn right.

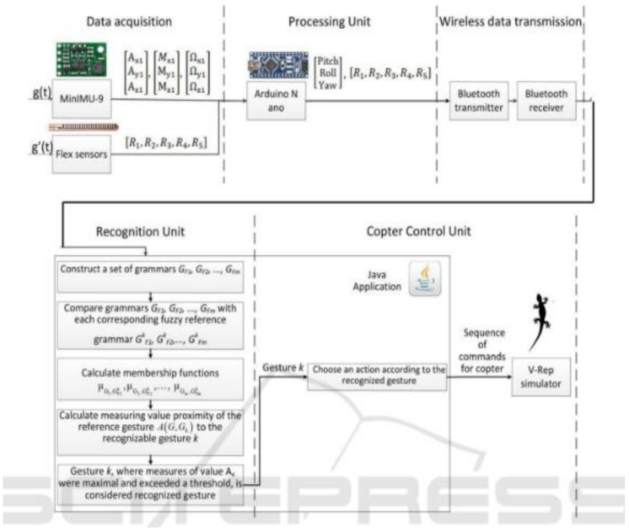
DEVELOPMENT OF A WEARABLE GLOVE SYSTEM FOR RECOGNITION:

The creation of a wearable gesture recognition system that includes sensor-integrated glove hardware and hand gesture detection software for human control of computer-based objects and devices.

Our gesture recognition method consists of three steps:
(1) tracking hand movement trajectories along the coordinate axes x(t), y(t), and z(t);
(2) creating a recognizable model for the gesture G[x(t), y(t), z(t)] using the tracked trajectories; and
(3) comparing the recognizable gesture model with the reference gesture patterns Ei [x(t), y(t), z(t)] by computing the similarity



These wearable glove systems and gesture-based software were utilized to create a control interface for manipulating a quadcopter model in the V-Rep simulator, exhibiting successful real-time gesture-based control of the quadcopter position and orientation with six different dynamic gestures.


Data acquisition
Processing Unit
Wireless data transmission
g(t)
MinIMU-9
g'(t)
Flex sensors
$[R_1, R_2, R_3, R_4, R_5]$
Arduino Nano
[Pitch Roll Yaw]
Bluetooth transmitter
Bluetooth receiver
Recognition Unit
Copter Control Unit
Construct a set of grammars
Compare grammars with each corresponding fuzzy reference grammar
Calculate membership functions
Calculate measuring value proximity of the reference gesture to the recognizable gesture k
Gesture k, where measures of value A, were maximal and exceeded a threshold, is considered recognized gesture
Java Application
Gesture k
Choose an action according to the recognized gesture
Sequence of commands for copter
V-Rep simulator

The functional diagram of wearable glove-system for gesture-based control of UAV.

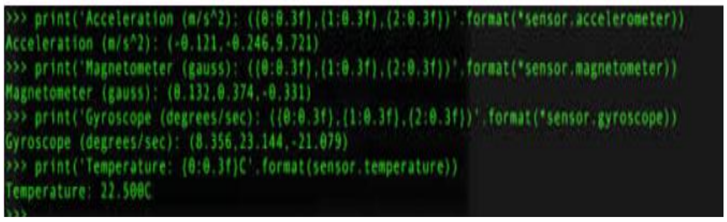
In order for our system to recognize an unfamiliar gesture, we need the following information:

1) Pitch, roll, and yaw hand rotations relative to the surface;

2) Acceleration projections on each coordinate axis;

3) The numerical value of each finger's bending

The integrated sensor MinIMU-9 v2, which consists of an accelerometer, magnetometer, and gyroscope, and measures projections to determine pitch, roll, and yaw, can be used to solve objectives 1 and 2.The objective 3 is reached using the flex sensors.



```
>>> print('Acceleration (m/s^2): ({0:0.3f},{1:0.3f},{2:0.3f})'.format(*sensor.accelerometer))
Acceleration (m/s^2): (-0.121,-0.246,9.721)
>>> print('Magnetometer (gauss): ({0:0.3f},{1:0.3f},{2:0.3f})'.format(*sensor.magnetometer))
Magnetometer (gauss): (0.132,0.374,-0.331)
>>> print('Gyroscope (degrees/sec): ({0:0.3f},{1:0.3f},{2:0.3f})'.format(*sensor.gyroscope))
Gyroscope (degrees/sec): (8.356,23.144,-21.079)
>>> print('Temperature: {0:0.3f}C'.format(sensor.temperature))
Temperature: 22.500C
>>>
```

We designed a gesture-based control interface for the wearable glove-based system, combining sensors with the Arduino Nano controller, Bluetooth (BT) wireless data transmission, and a Java application.

SOFTWARE DESIGN:
PERIODICALLY SENDS THE MOTION

- Pin Type: I2C protocol (SDA and SCL) 19Gesture Controlled Quadcopter System Receives input from the 9-DOF sensor.
- The data from the sensor is obtained using the Arduino library.
- Converts data to a String that may be sent.
- This is a cyclic function that executes every few milliseconds.
- An enable variable is used to control it.

HOVER BUTTON:

- Pin Type: Digital I/O, Interrupt message
- If the quadcopter is currently in the air, a particular message is sent to the base telling it to hover.
- The microcontroller's signal LED should be lit to indicate that the controller is off.
- Until the hand is pulled out of Hover mode, no movement will be relayed to the controller.
- This mode allows you to command the quadcopter to land.

STOP BUTTON:

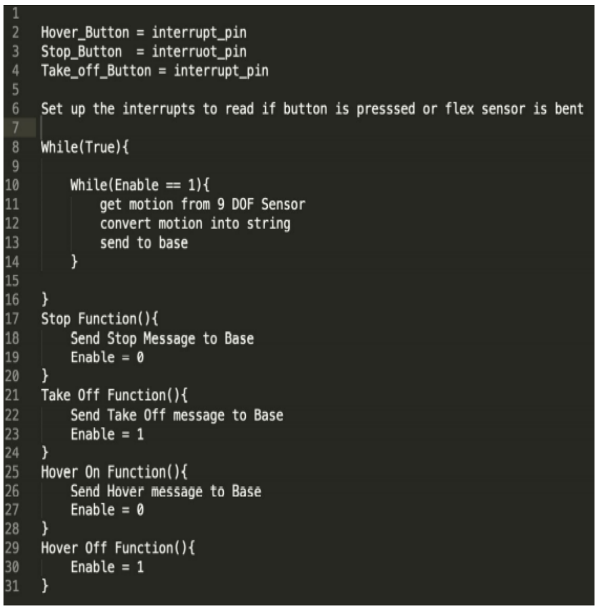
- Pin Type: Digital I/O, Interrupt message
- If the quadcopter is currently in the air, it receives a special message instructing it to land.
- The controller will no longer be able to move.

TAKE OFF FLEX SENSOR:

- Interrupt message, Analog Pin Type

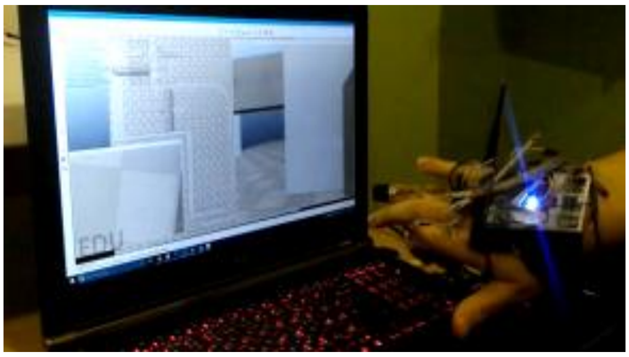
- Because the data is analogue, there will be a "bentness" threshold that indicates the start function.
- After the quadcopter has completed the task, this function allows the hand controller to read the motion of the hand.
- This will begin the quadcopter's takeoff function, which will take it to a height of roughly a meter before hovering.

The pseudo code that implements the functions described above is listed below:

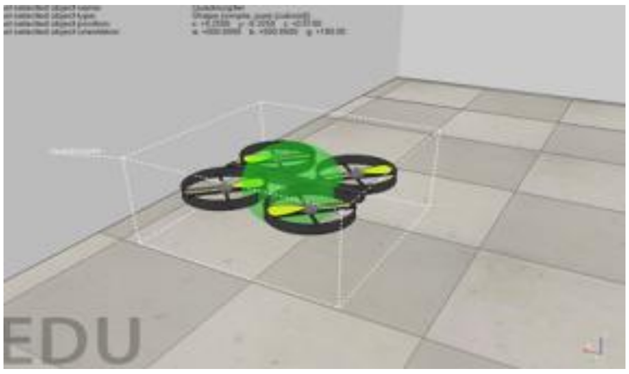


```

Hover_Button = interrupt_pin
Stop_Button  = interruot_pin
Take_off_Button = interrupt_pin

Set up the interrupts to read if button is presssed or flex sensor is bent

While(True){

    While(Enable == 1){
        get motion from 9 DOF Sensor
        convert motion into string
        send to base
    }

}
Stop Function(){
    Send Stop Message to Base
    Enable = 0
}
Take Off Function(){
    Send Take Off message to Base
    Enable = 1
}
Hover On Function(){
    Send Hover message to Base
    Enable = 0
}
Hover Off Function(){
    Enable = 1
}
```

INPUT DATABASE:
The first requires the employment of a third-party device that can reliably recognize non-verbal gestures beforemapping them into appropriate digital commands.
The Leap Motion Controller and Microsoft Kinect are two examples of such devices.



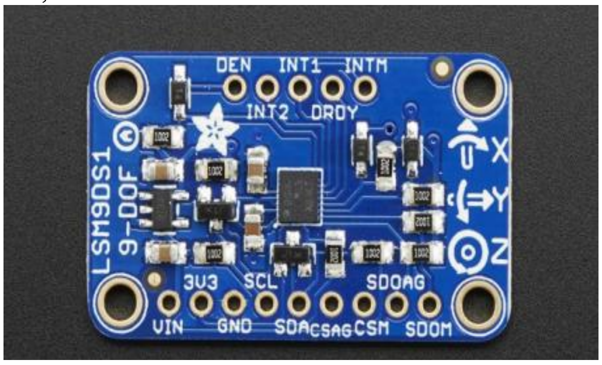
While the Leap Motion Controller is meant to catch hand motions only, the Kinect is capable of accurately capturing complete body motion. While this method provides excellent accuracy in gesture or body motion recognition, it requires a computer to function, hence portability is a limitation.



Body movement is recognized in real-time using machine vision in the second direction, allowing the drone to be controlled without the use of any additional equipment. Eye gazes, face expressions, hand gestures, and combinations of these have all been studied by researchers.

COMPONENTS:
LSM9DS1 Accelerometer + Gyro + Magnetometer

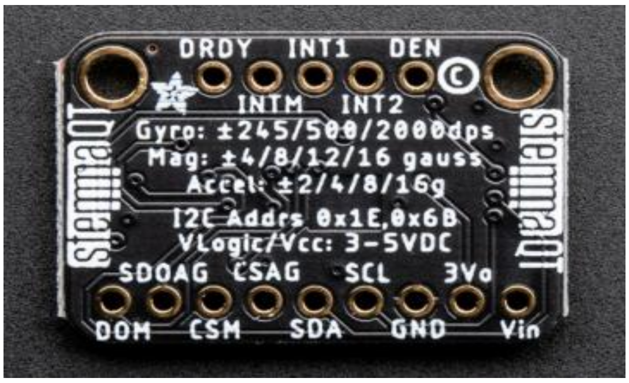
With this all-in-one 9-DOF sensor, you can detect movement, direction, and orientation in any Arduino project. Three sensors are housed within the chip, one of which is a traditional 3-axis accelerometer that can tell you which way is down towards the Earth (by sensing gravity) or how fast the board is accelerating in 3D space. The other is a three-axis magnetometer, which can determine magnetic north by detecting where the strongest magnetic force is coming from. A 3-axisgyroscope that can measure spin and twist is the third component.



<u>The LSM9DS1 is not the same set of sensors as the LSM9DS0.</u>

- The LSM9DS0 accelerometer has ranges of 2/4/6/8/16 g. The LSM9DS1 has a weight range of 2/4/8/16 g (no 6 g range).
- The LSM9DS0 magnetometer has gauss ranges of 2/4/8/12 gauss. The LSM9DS1 has ranges of 4/8/12/16 gauss.
- As a result, the LSM9DS0 has a 2-gauss low range and a 16-gauss high range.
- The gyros LSM9DS0 and LSM9DS1 have the same 245/500/2000 dps ranges.

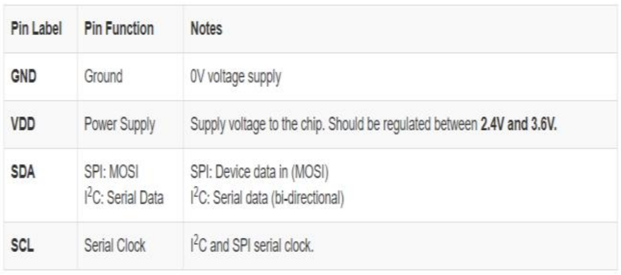
BREAKOUT BOARD:



The breakout board comes completely constructed and tested, as well as some extra header for usage on a breadboard. We put the popular power data pins on one side and the interrupt pins on the other side for a neat & compact breakout, and we used four mounting holes to provide a solid connection.

Power Pins:

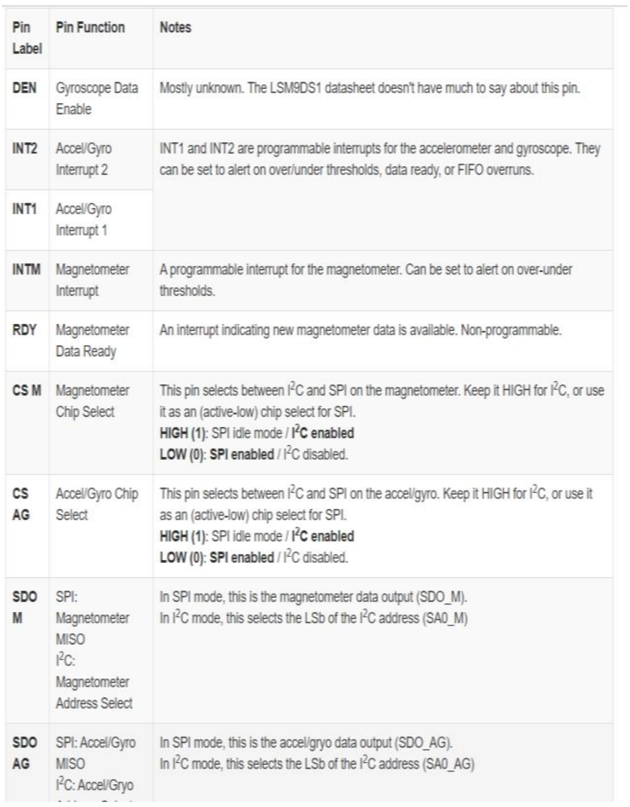
The breakout requires 3V electricity to power the sensor. We placed a 3.3V regulator on the board because many clients use 5V microcontrollers like Arduino. Because of its extremely low dropout, it can be powered from 3.3V to 5V. The power pin is Vin. We added a voltage regulator on board to safely convert 3-5VDC to 3 VDC because the chip uses 3 VDC. To power the board, use the same voltage as your microcontroller's logic level - for example, for a 5V microcontroller like Arduino, use 5V 3V3 - this is the 3.3V output from the voltage regulator, and you can pull up to 100mA from it if you like.
GND - a place where logic and power meet. Pins for I2C communication
SCL - I2C clock pin, which should be connected to the I2C clock line on your microcontroller. This pin has a 10K pullup and is level adjusted, allowing it to be used with 3-5V logic.
SDA - I2C data pin, which should be connected to the I2C data line on your microcontroller. This pin has a 10K pullup and is level adjusted to allow for 3-5V logic.



Pin Label	Pin Function	Notes
GND	Ground	0V voltage supply
VDD	Power Supply	Supply voltage to the chip. Should be regulated between **2.4V and 3.6V.**
SDA	SPI: MOSI I^{2}C: Serial Data	SPI: Device data in (MOSI) I^{2}C: Serial data (bi-directional)
SCL	Serial Clock	I^{2}C and SPI serial clock.

SPI Pins:

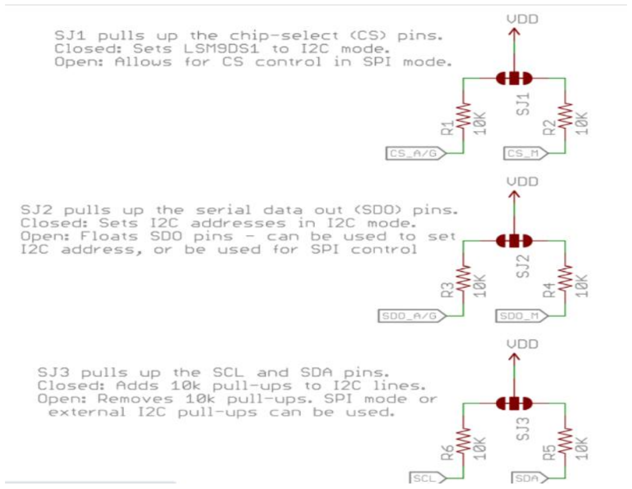
• SCL - commonly known as the SPI clock pin, this pin has been level shifted to allow 3-5V logic input.
• SDA - commonly known as the SPI MOSI pin, this is a level-shifted SPI MOSI pin that allows you to use a 3-5V logic input.
• CSAG - this is the Accelerometer Gyro Sub-Chip Select, which has been level-shifted to allow for 3-5V logic input.
• CSM - this is the Magnetometer sub-chip Select, which has been level-shifted to allow 3-5V logic input.
• SDOAG - this is the Accelerometer Gyro sub chip's MISO pin; it's 3V logic out, but 5V logic chips can read it well.
• SDOM/DOM- This is the MISO pin on the Magnetometer sub chip; it's 3V logic out, but 5V logic chips can read it well.



Pin Label	Pin Function	Notes
DEN	Gyroscope Data Enable	Mostly unknown. The LSM9DS1 datasheet doesn't have much to say about this pin.
INT2	Accel/Gyro Interrupt 2	INT1 and INT2 are programmable interrupts for the accelerometer and gyroscope. They can be set to alert on over/under thresholds, data ready, or FIFO overruns.
INT1	Accel/Gyro Interrupt 1	
INTM	Magnetometer Interrupt	A programmable interrupt for the magnetometer. Can be set to alert on over-under thresholds.
RDY	Magnetometer Data Ready	An interrupt indicating new magnetometer data is available. Non-programmable.
CS M	Magnetometer Chip Select	This pin selects between I^{2}C and SPI on the magnetometer. Keep it HIGH for I^{2}C, or use it as an (active-low) chip select for SPI. **HIGH (1)**: SPI idle mode / **I^{2}C enabled** **LOW (0)**: **SPI enabled** / I^{2}C disabled.
CS AG	Accel/Gyro Chip Select	This pin selects between I^{2}C and SPI on the accel/gyro. Keep it HIGH for I^{2}C, or use it as an (active-low) chip select for SPI. **HIGH (1)**: SPI idle mode / **I^{2}C enabled** **LOW (0)**: **SPI enabled** / I^{2}C disabled.
SDO M	SPI: Magnetometer MISO I^{2}C: Magnetometer Address Select	In SPI mode, this is the magnetometer data output (SDO_M). In I^{2}C mode, this selects the LSb of the I^{2}C address (SA0_M)
SDO AG	SPI: Accel/Gyro MISO I^{2}C: Accel/Gryo	In SPI mode, this is the accel/gryo data output (SDO_AG). In I^{2}C mode, this selects the LSb of the I^{2}C address (SA0_AG)

Interrupt &Misc Pins:
• DEN - this is a pin that is designed to be used to enable/disable the Gyro dynamically. We don't have any documentation on it, but we'll break it down for you nonetheless.
• INT1 & INT2 - The accelerometer/gyro sub chip interrupts. We don't have any library support for these, so look at the datasheet to see what you can do with them. They're logic outputs with a voltage of 3V.
• DRDY - this is the data ready output from the accelerometer/gyro sub chip. We don't have particular library support for them, therefore consult the datasheet for information on how to enable this pin using the registers. It's a logic output with a voltage of 3V.
• INTM - This is the magnetometer subchip's interrupt. We don't have any library support for it, so look at the datasheet to see what you can do with it. It's a logic output with a voltage of 3V.

The Jumpers:
A trio of two-way surface mount jumpers are revealed when the LSM9DS1 breakout is flipped over. Each of these jumpers is finished with a zipper closure. They're there to put the LSM9DS1 in I2C mode automatically.


SJ1 pulls up the chip-select (CS) pins.
Closed: Sets LSM9DS1 to I2C mode.
Open: Allows for CS control in SPI mode.
VDD
SJ1
R1 10K
R2 10K
CS_A/G
CS_M
SJ2 pulls up the serial data out (SDO) pins.
Closed: Sets I2C addresses in I2C mode.
Open: Floats SDO pins - can be used to set I2C address, or be used for SPI control
VDD
SJ2
R3 10K
R4 10K
SDO_A/G
SDO_M
SJ3 pulls up the SCL and SDA pins.
Closed: Adds 10k pull-ups to I2C lines.
Open: Removes 10k pull-ups. SPI mode or external I2C pull-ups can be used.
VDD
SJ3
R6 10K
R5 10K
SCL
SDA

Through a 10k resistor, each of these jumpers pushes a pair of pins up to VDD. The resistor is connected to the central pad of the jumper, while the edge pads are connected to a pin. The top jumper links CS AG and CS M to a pull-up, which activates I2C mode on the LSM9DS1. The center jumper activates SDO AG and SDO M, which sets the chip's I2C address. Finally, the far-left jumper connects the I2C communication pins SDA and SCL using pull-up resistors.
These jumpers are designed to make utilizing the board as simple as possible, with the least number of wires possible. The four SDO and CS pins can be ignored if you're utilizing I2C with the breakout.

FUNCTIONALITY:
Using the Arduino Library
Setup Stuff:
<span class="meta preprocessor">#<span class="keyword include">include</span><span class="string"><SPI.h> // SPI library included for SparkFunLSM9DS1</span></span>
<span class="meta preprocessor">#<spanclass="keyword include">include</span><span class="string"><Wire.h> // I2C library included for SparkFunLSM9DS1</span></span>
<span class="meta preprocessor">#<span class="keyword include">include</span><span class="string"><SparkFunLSM9DS1.h> // Spark Fun LSM9DS1 library</span></span>
Make sure the SPI and Wire includes are above the "SparkFunLSM9DS1"

Constructor:
A new instance of the LSM9DS1 class is created by calling the function Object () [native code]. You'll use the instance to control the breakout from there on once you've built it. This one line of code is normally placed in your sketch's global section.
constructor should be left without any parameters:
<span class="comment">// Use the LSM9DS1 class to create an object. [imu] can be</span>
<span class="comment">// named anything, we'll refer to that through the sketch. </span>
LSM9DS1 imu;

Setting Up the Interface:
There are numerous settings available on the LSM9DS1. Some are insignificant, while others are vital. The communication interface and the device addresses are the three most important things we'll need to specify.
<span class="comment">// SDO_XM and SDO_G are both pulled high, so our addresses are:</span>
<span class="meta preprocessor">#<span class="keyword define">define</span><span class="entity name">LSM9DS1_M</span><span class="constant numeric">0</span>x<span class="constant numeric">1</span>E // <span class="keyword define">Would</span><span class="entity name">be</span><span class="keyword define">0x1C</span><span class="entity name">if</span><span class="keyword define">SDO_M</span><span class="entity name">is</span> LOW</span>
<span class="meta preprocessor">#<span class="keyword define">define</span><span class="entity name">LSM9DS1_AG</span><span class="constant numeric">0</span>x<span class="constant numeric">6</span>B // <span class="keyword define">Would</span><span class="entity name">be</span><span class="keyword define">0x6A</span><span class="entity name">if</span><span class="keyword define">SDO_AG</span><span class="entity name">is</span> LOW</span>

...
imu. Settings. device. COMM Interface<span class="keyword operator">=</span> IMU_MODE_I2C; <span class="comment">// Set mode to I2C</span>
imu. Settings. device. mAddress <span class="keyword operator">=</span> LSM9DS1_M; <span class="comment">// Set mag address to 0x1E</span>
imu. Settings. device. agAddress <span class="keyword operator">=</span> LSM9DS1_AG; <span class="comment">// Set ag address to 0x6B</span>
Alternatively, if you're using SPI mode, the imu. Settings. device. mAddress and imu. Settings. device. agAddress values become the chip select pins.
imu. Settings. device. COMM Interface<span class="keyword operator">=</span> IMU_MODE_SPI; <span class="comment">// Set mode to SPI</span>
imu. Settings. device. mAddress <span class="keyword operator">=</span><span class="constant numeric">9</span>; <span class="comment">// Mag CS pin connected to D9</span>
imu. Settings. device. agAddress <span class="keyword operator">=</span><span class="constant numeric">10</span>; <span class="comment">// AG CS pin connected to D10</span>
Configuring any value from the imu. Settings. device can't take place in the global are of a sketch. If you get a compilation error, like 'imu' does not name a type, you may have those in the wrong place -- put them in setup ().

begin ()-ing and Setting Sensor Ranges:

To initialize the IMU, call the begin () member function once you've created the LSM9DS1 object and set its interface.
The begin () function attempts to interact with and initialize the sensors using the settings you adjusted in the previous phase. To see if the setup was successful, look at the return value of begin (); if something goes wrong, it will return 0.
<span class="keyword">if</span> (<span class="keyword operator">! </span>imu. <span class="function call">begin</span> ())
{Serial. <span class="function call">println</span> (<span class="string">"Failed to communicate with LSM9DS1."</span>);
Serial. <span class="function call">println</span> (<span class="string">"Looping to infinity."</span>);
<span class="keyword">while</span> (<span class="constant numeric">1</span>);}
Once begin () has returned a success, you can start reading some sensor values!

Reading and Interpreting the Sensors: readAccel (), readGyro (), and readMag ()

The readAccel (), readGyro (), and readMag () functions poll the LSM9DS1 for the most recent measurements from each of the three sensors.
When the function completes, it will update a set of three class variables with the sensor data you want. readAccel () will update the values for axe, ay, and az, readGyro () will update the values for gx, gy, and gz, and readMag () will update the values for mx, my, and mz.
imu. <span class="function call">readAccel</span> (); <span class="comment">// Update the accelerometer data</span>
Serial. <span class="keyword">print</span>(imu.ax); <span class="comment">// Print x-axis data</span>
Serial. <span class="keyword">print</span> (<span class="string">", "</span>);
Serial. <span class="keyword">print</span> (imu. ay); <span class="comment">// print y-axis data</span>
Serial. <span class="keyword">print</span> (<span class="string">", "</span>);
Serial. <span class="function call">println</span>(imu.az); <span class="comment">// print z-axis data</span>

An example of reading and printing all three axes of accelerometer data.

Those values are all signed 16-bit integers, meaning they'll range from -32,768 to 32,767.
calcAccel (), calcGyro (), and calcMag ()
The library keeps track of each sensor's scale and provides some helper methods to make converting raw ADC readings to real units as simple as possible.
The functions calcAccel (), calcGyro (), and calcMag () all take a single parameter, a signed 16-bit integer, and convert it to the appropriate unit. All of them return a float value.
imu. <span class="function call">readGyro</span> (); <span class="comment">// Update gyroscope data</span>
Serial. <span class="keyword">print</span> (imu. <span class="function call">calcGyro</span> (imu. gx)); <span class="comment">// Print x-axis rotation in DPS</span>
Serial. <span class="keyword">print</span> (<span class="string">", "</span>);
Serial. <span class="keyword">print</span> (imu. <span class="function

call">calcGyro</span>(imu.gy)); <span class="comment">// Print y-axis rotation in DPS</span>
Serial. <span class="keyword">print</span> (<span class="string">", "</span>);
Serial. <span class="function call">println</span> (imu. <span class="function call">calcGyro</span>(imu.gz)); <span class="comment">// Print z-axis rotation in DPS</span>

Setting Sensor Ranges and Sample Rates:
The sensor ranges and output data rates are two of the most regularly modified properties in the IMU. Again, by setting a value in the settings struct, these variables can be customized.
For example, to set the IMU's accelerometer range to ±16g, gyroscope to ±2000 °/s, and magnetometer to ±8 Gs.
imu.settings. accel. scale <span class="keyword operator">=</span><span class="constant numeric">16</span>; <span class="comment">// Set accel range to +/-16g</span>
imu.settings. gyro. Scale <span class="keyword operator">=</span><span class="constant numeric">2000</span>; <span class="comment">// Set gyro range to +/-2000dps</span>
imu.settings.mag. scale. scale <span class="keyword operator">=</span><span class="constant numeric">8</span>; <span class="comment">// Set mag range to +/-8Gs</span>
imu. <span class="function call">begin</span> (); <span class="comment">// Call begin to update the sensor's new settings</span>
The data rates for the output data are a little more ethereal. The update rate can be anywhere between 1 and 6, with 1 being the slowest and 6 being the fastest.

REFERNCES:

[1]. Quadcopter control using Arduino microcontroller, Ghosh, Arijit in 2018.
[2]. The Hand gesture-based controller interface with wearable glove system, Berezhnoy, Vladislav; Popov, Dmitry; Afanasvey, Ilya; Mavridis, Nikolaos in 2018.
[3]. Implementation of Arduino based hand motion control drone, Dubey, Abhishek Kumar in 2020.
[4]. Gesture controlled quadcopter system, Xu, Anthony yang; Crawford, kendra; Yang, Wenhao in 2018.
[5]. Vision based control by hand-directional gestures converting to voice, Malallah, fahad layth; Khaled, Khaled N. Yasen Mustafa; Abdulameer, Sadeer in 2018.
[6]. AdafruitLSM9DS0 Accelerometer+Gyro+Magnetometer 9-DOF Breakouts, Edt, PM in 2014.
[7]. Hand gesture controller drones; An open library, Natarajan, kathiravan; Nguyen, Truong Huy D; Mete, mutlu in 2018.