Università degli Studi "Magna Græcia" di Catanzaro

Dipartimento di Medicina sperimentale e clinica

Corso di Laurea in

Ingegneria Informatica e Biomedica

Tesi di laurea in

Fondamenti di Automatica

"Dispositivo low-cost per il monitoraggio dei parametri vitali dei neonati"

Relatore

Chiar.mo Prof.

Carlo Cosentino

Candidato

Alessandro Pellegrino

Matricola n. 114918

Anno Accademico 2015/2016

...a Mamma.
...a Papà.
...a Valerio.
...a Olga.

INDICE

INTRODUZIONE

L’oggetto di questa tesi sperimentale è incentrato sulla progettazione, implementazione e sperimentazione di un dispositivo a basso costo che si presti al monitoraggio delle funzioni vitali di chi lo indossa. L’idea di questo dispositivo è nata per consentire un maggior controllo da parte dei genitori nei bambini nel loro primo anno di età, col fine di evitare tutti quei casi imprevisti di decesso: soffocamento accidentale, asfissia accidentale da posizione, soffocamento per inalazione o per rigurgito o anche casi che la comunità scientifica non ha ancora spiegato con chiarezza come la SIDS (*Sudden Infant Death Syndrome),* ovvero sindrome della morte improvvisa infantile. In tutti in casi sopraelencati è di vitale importanza accorgersi immediatamente del pericolo, sia per salvare la vita all’infante, sia per evitargli danni celebrali permanenti.

Sebbene le percentuali dei suddetti decessi siano per fortuna veramente basse, è quantomeno necessario che i genitori abbiamo a disposizione un sistema *embedded* economicamente accessibile e molto affidabile.

All’interno di questo lavoro di tesi, si andranno ad analizzare le varie fasi che hanno costituito lo sviluppo del dispositivo. Dall’ideazione alla progettazione, dalla scelta dei sensori alla loro applicazione, dalle prove preliminari fino alla sperimentazione finale, avendo sempre come scopo quello di descrivere il tutto da un punto di vista ingegneristico.

CAPITOLO 1

Presentazione del problema

1.1 Descrizione del problema

Saper rilevare, analizzare e interpretare i biosegnali col fine di avere un quadro ampliamente dettagliato di quello che accade nel corpo umano è uno dei principali obiettivi che si pone l'Ingegneria Biomedica.

Col presente lavoro ci si preoccuperà di come monitorare alcuni semplici parametri vitali concentrandosi in particolar modo sulla respirazione, avendo come target principale i bambini sotto i cinque anni di età.

Tutti i genitori sanno quanta attenzione e responsabilità comporti l'avere un figlio. Nel suo primo anno di vita, il neonato necessita di un controllo costante sia di giorno che di notte.

Avere dunque a disposizione un dispositivo che monitori il neonato e che avverta i genitori in caso di pericolo durante le ore di sonno, può rappresentare un grosso vantaggio se si pensa che solo la sindrome della morte improvvisa del lattante (SIDS) rappresenta l'1% delle morti in culla di bambini sotto l'anno di vita[1].

1.2 Dispositivi in commercio

Diverse aziende si sono già occupate del problema producendo dispositivi capaci di rilevare la respirazione e la frequenza respiratoria. La *Jablotron* produce *Nanny*, certificato come dispositivo medico. Questo apparecchio riesce a monitorare la respirazione grazie ad una tavoletta-sensore che si posiziona sotto il materasso della culla del neonato e comunica all'unità di controllo se l'atto respiratorio si sta espletando correttamente; se negli ultimi venti secondi non si è rilevato alcun movimento oppure se la frequenza respiratoria dovesse scendere al di sotto di 8 respiri/min, verrà emanato un allarme acustico e visivo. Il prezzo di *Nanny* è di 114 €. Altre aziende che producono dispositivi simili a *Nanny* sono: *Foppapedretti* con *AngelCare* (95€), *Hisense* con *Babysense* (120€) e *SafeToSleep* con

[1] http://www.istat.it/it/files/2014/01/Mortalita_-sotto_i_5_anni-.pdf

l’omonimo dispositivo (100€); quest’ultime sono anche provviste di applicazione smartphone per visualizzare i dati provenienti dai sensori in tempo reale.

Tra i dispositivi che utilizzano diversi principi di funzionamento figurano: *Oma+* e *Zoyobaby*, piccole unità che vanno applicate sul pannolino e *Owlet*, uno *smart socket* che invia allo smartphone via bluetooth diversi parametri vitali (battito cardiaco, frequenza respiratoria, temperatura).

Fig.1 Da sinistra a destra: Owlet, Nanny, Angelcare.

1.3 Approccio al problema

Ci si pone dunque il problema di come fare per monitorare respirazione e temperatura del neonato. Per riuscire a monitorare quest’ultima nel tempo ci si è affidati ad un sensore di temperatura, la cui scelta del tipo e del modello verrà discussa in seguito. Per quanto riguarda la respirazione il discorso è più ampio, poiché bisogna prima decidere quale principio fisico si intende sfruttare e interpretare. Si potrebbe optare di scegliere una tecnologia simile ai dispositivi visti in precedenza, ma l’ obiettivo è sia progettare qualcosa di nuovo sia quello di mantenere i costi decisamente ridotti, dunque si dovrà procederà su strade differenti da quelle adottate dai dispositivi già visti.

Come è ben noto, l’atto respiratorio si divide in due fasi: inspirazione ed espirazione. Durante la prima fase la zona toracica subisce un’espansione mentre durante la seconda subisce una recessione, dovuti rispettivamente

all’aumento e alla diminuzione di volume di aria nei polmoni. Si possono dunque analizzare questi segnali per verificare se chi indossa il dispositivo respiri o meno, e per farlo ci si affiderà ad una particolare categoria di sensori: gli accelerometri.

Negli ultimi anni, l’uso degli accelerometri è cresciuto esponenzialmente in moltissimi settori partendo da quello industriale fino ad arrivare a quello hobbistico. Il successo di questa versatile tipologia di sensori, che ha trovato i sui primi utilizzi nel settore aereospaziale, è dovuta al fatto che il loro costo ha subito un prepotente calo durante gli ultimi anni, dovuto soprattutto al rapido sviluppo della microelettronica.

Una volta scelta la tipologia di sensori che si andrà ad utilizzare per lo sviluppo dell’apparecchio bisogna decidere in che modo ci si potrà interfacciare, in modo da manipolare e controllare i dati in modo utile. Per farlo si è deciso di utilizzare la scheda elettronica Arduino, le cui caratteristiche principali verranno discusse nel prossimo capitolo.

CAPITOLO II

Descrizione generale di Arduino.

Scelta dei sensori e specifiche tecniche.

2.1 Arduino

2.1.1 Caratteristiche generali

La scheda elettronica Arduino è stata sviluppata da alcuni membri della Interaction Design institute di Ivrea[1], con lo scopo di rendere possibile la realizzazione di semplici progetti anche a persone che non hanno competenze di programmazione e di elettronica. Arduino è un progetto *open source*, ovvero i produttori hanno deciso di rendere pubblico lo schema circuitale (solo di Arduino Uno, per le altre schede resta un progetto *closed source*) e del codice sorgente dell'IDE (*Integrated Development Environment*) in modo tale da consentire a programmatori/elettronici esperti o anche a semplici appassionati, la possibilità di apportare modifiche e sviluppare espansioni e librerie di supporto. Grazie a questa scheda, è possibile creare progetti interattivi che fanno uso di LED, servomotori, motori, e altre tipologie di sensori e attuatori. La possibilità di interfacciarsi con sensori e altri elementi elettronici, è data dal microcontrollore, che rappresenta il cuore della scheda Arduino. In Fig.1 è riportato Arduino UNO:

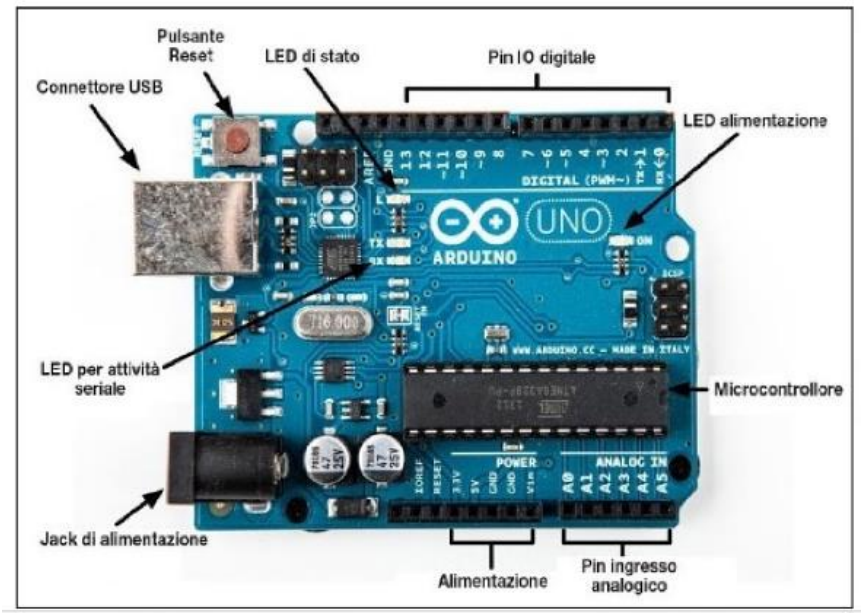

Fig.1 Arduino UNO.

[1] https://it.wikipedia.org/wiki/Arduino_(hardware)

- Connettore USB: il suo scopo è quello di collegare tramite un cavo USB la scheda al computer per fornire l'alimentazione e per poter caricare nella memoria del microcontrollore il programma che si vuole far eseguire ad Arduino.
- Pulsante reset: se premuto fa ripartire dall'inizio un programma caricato precedentemente sulla scheda.
- LED di stato: è l'unico attuatore presente sulla scheda, è collegato al pin digitale n° 13 e ha diversi scopi; i programmatori esperti fanno in modo che si accenda quando durante l'esecuzione di un programma insorge un errore.
- Pin Input/Output Digitali: sono i pin dedicati ai segnali digitali. I segnali digitali sono quelli che possono assumere solo due valori, 1 oppure 0. In questo caso sono espressi in termini di tensione, 5 e 0 V. Attraverso l'IDE, si può decidere se un determinato pin riceve un segnale digitale (input), oppure se manda un segnale digitale (output). Tra i pin digitali, alcuni fanno uso della "Modulazione a larghezza d'impulso" (PWM, *pulse width modulation*): si tratta essenzialmente di un segnale digitale periodico caratterizzato da alcuni parametri fondamentali quali:
  -il periodo e la frequenza;
  -il ciclo utile (duty cycle).
- LED di alimentazione: si accende quando Arduino è collegato ad una fonte di alimentazione.
- Microcontrollore: come già detto, rappresenta il cuore di Arduino. Qui risiede la memoria in cui vengono caricati i programmi. Nel caso di Arduino Uno, viene utilizzato l'ATmega328, un microcontrollore a 8 bit con una memoria di 32kb.
- Pin ingresso analogico: questi pin sono dedicati alla lettura di segnali analogici, ovvero segnali che assumono valori all'interno di un range noto. Anche per essi c'è la possibilità di impostarli come input o come

output. Il segnale analogico per essere elaborato da Arduino deve essere opportunamente campionato, cioè deve essere convertito in una sequenza di bit che ne esprime l'ampiezza.

- Alimentazione: grazie a questi pin si può fornire l'alimentazione al nostro circuito. Le tensioni erogabili sono: messa a terra, pin indicato col nome GND,3.3 V e 5 V.
- Jack di alimentazione: qui è possibile alimentare Arduino con una batteria esterna, la cui tensione consigliata da utilizzare va dai 6 ai 12 V, ma la scheda supporta fino a 20 V.
- LED per attività seriale: i led *tx* e *rx* comunicano se è in corso un'attività di comunicazione seriale. Il LED *tx* indica quando Arduino sta trasmettendo, mentre il LED *rx* quando sta ricevendo.

Le specifiche tecniche della scheda sono riportate in Fig.2, mentre lo schema circuitale di Arduino UNO è riportato nell'appendice A.

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Fig.2 Specifiche tecniche Arduino Uno.

2.1.2 *Integrated development environment* (IDE)

Dopo aver descritto brevemente le parti hardware, è il momento di parlare del software dove è possibile scrivere il codice che poi verrà caricato sull'MCU. I linguaggi che si usano per programmare sono C o C++.

La struttura dei programmi è molto semplice, infatti necessita solo di due funzioni principali: il *setup()* e il *loop()*. Il primo serve per le impostazioni preliminari, come per esempio scrivere se un pin leggerà un input o un output, mentre il secondo conterrà il corpo del programma e verrà eseguito ripetutamente finché la scheda è alimentata. In fig.5 è riportato un esempio di codice.

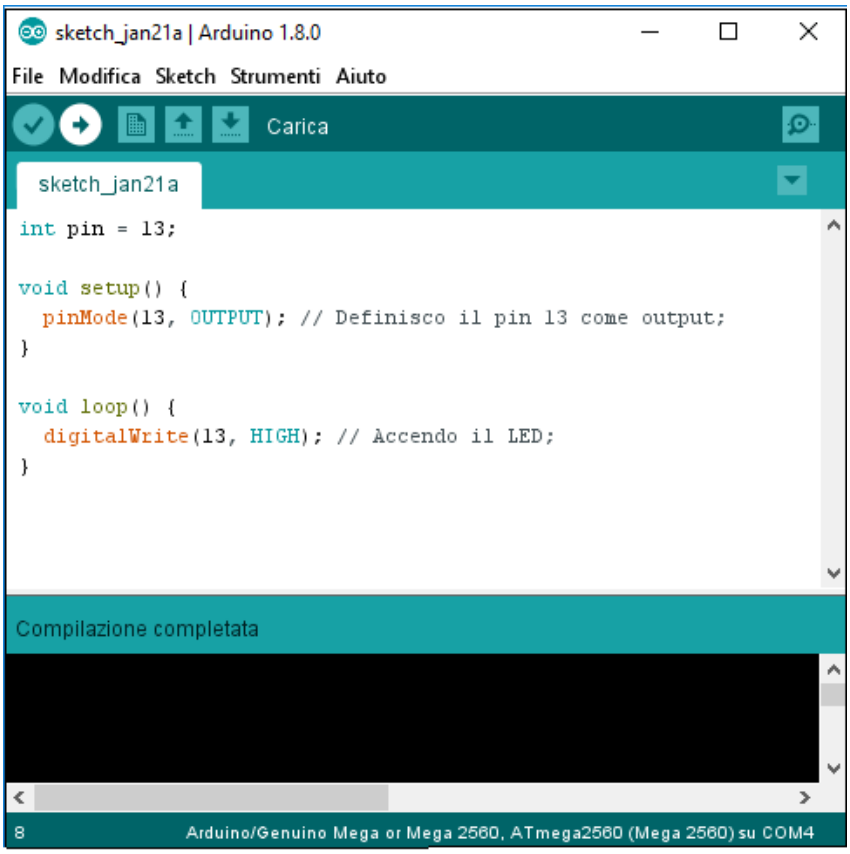

Fig.5 Esempio di codice per accendere il LED presente sulla scheda Arduino.

2.2 Sensori: descrizione e aspetti generali.

Il sensore è un dispositivo che ha il compito di misurare una determinata grandezza fisica. Esso può essere di natura meccanica, chimica o elettronica. Il principio di funzionamento cambia da sensore a sensore in base a cosa si vuole misurare, anche se in linea di massima quello che fanno è ricevere in input un

segnale e dare in output un altro segnale di diversa natura, anche se in questo caso sarebbe più opportuno chiamarli con nome di trasduttori.

Un sensore può essere classificato in base a diverse caratteristiche, tra cui:

- Accuratezza;
- Precisione;
- *Rangeability;*
- Sensibilità;
- Risoluzione.

2.2.1 Accuratezza.

Accuratezza è il termine usato per esprimere quanto la misura di avvicina al valore "vero" del misurando. Il fatto che il valore vero sia, in generale, sconosciuto e non conoscibile, evidenzia la difficoltà concettuale che si incontra nel definire quantitativamente l'accuratezza di uno strumento. Il "valore vero" sarà in realtà un valore di riferimento, ottenibile anch'esso per misurazione, ma utilizzando un metodo considerato esemplare. Difficoltà di carattere pratico sono dovute al fatto che la misura, in generale, non è influenzata solo dal misurando, ma anche da altre variabili, come la temperatura, l'umidità, la pressione, la presenza di vibrazioni e accelerazioni.

L'accuratezza di uno strumento è solitamente specificata da un unico valore, in cui significato è quello di massimo scostamento tra le misure fornite dal sensore e la curva di taratura sull'intero campo di misura. Può essere espressa come percentuale del campo di misura, o del fondo scala, o della misura. Nel primo caso si avrà:

$$\varepsilon_f = 100 \cdot \frac{X_m - X_v}{X_{FS}}$$

Dove X_v è il valore vero; X_m è il valore misurato più distante da X_v; X_{FS} il valore di fondo scala.

Nel secondo caso (percentuale della misura), l'accuratezza sarà espressa come:

$$\varepsilon_a = 100 \cdot \frac{X_m - X_v}{X_v}$$

In generale sia ε_f che ε_a sono funzioni di X_v.

La mancanza di accuratezza deriva da errori sistematici, che dipendono principalmente dalla calibrazione del sensore.

2.2.2 Precisione.

Il termine precisione si riferisce alla ripetitività della misura, ossia esprime la dispersione delle successive misure dello stesso misurando nelle medesime condizioni. La precisione di un sensore può essere calcolata come il massimo scostamento tra una generica lettura e la miglior stima della misura ottenuta con il sensore e poiché deriva da errori casuali, può essere aumentata con medie di successive letture. Nel controllo di processo la precisione è molto più importante dell'accuratezza.

2.2.3 Rangeability.

Il termine *rangeability* esprime di solito il rapporto tra l'estremo superiore (fondo scala) e quello inferiore, normalizzato all'unità. Per esempio, un sensore di portata con rangeability di 20:1, con fondo scala di 300 kg/s e accuratezza del 1% (della misura), registra, con tale accuratezza, portate comprese tra 15 e 300 kg/s.

2.2.4 Sensibilità.

La sensibilità di un sensore S, è definita come il rapporto tra una variazione dell'uscita Δy del sensore e la corrispondente variazione della variabile misurata Δu:

$$S(y) = \frac{\Delta y}{\Delta u}$$

2.2.5 Risoluzione.

La risoluzione è generalmente espressa come percentuale del campo di misura. Per esempio, se un sensore di temperatura produce un incremento ΔV nell'uscita in tensione in seguito ad una variazione di temperatura ΔT, la risoluzione R è la più piccola variazione di temperatura ΔT che produce una variazione ΔV rilevabile:

$$R = 100 \cdot \frac{\Delta T}{T_{max} - T_{min}}$$

La risoluzione attorno al valore 0 è detta anche soglia di sensibilità dello strumento.

2.3 Accelerometri e sensori di temperatura: principio di funzionamento.

2.3.1 Principio di funzionamento di un accelerometro.

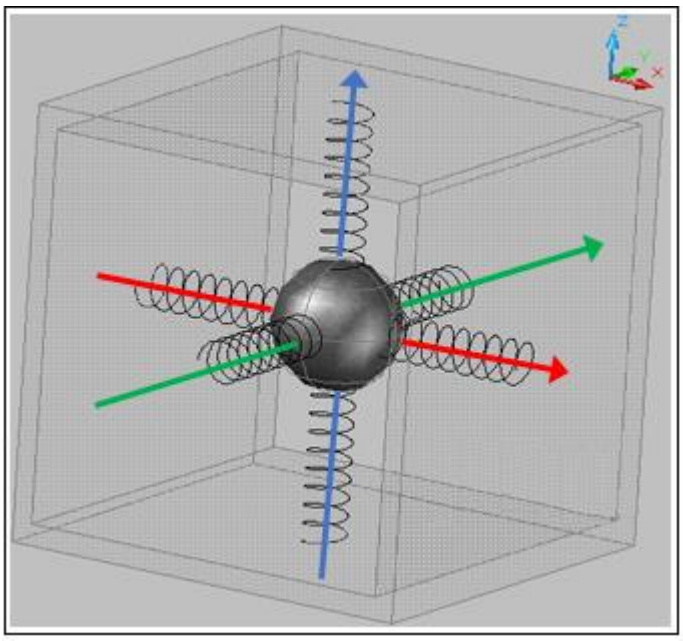

Fig.6 Interno di un accelerometro.

In fig.6 è rappresentato il principio di funzionamento di un generico accelerometro. È composto da una sfera di metallo collegata a tre molle, una per ogni asse. Quando l'accelerometro si muove, la sfera al suo interno si sposta di conseguenza comprimendo le relative molle. Quello che l'accelerometro fa, è trasdurre la variazione di compressione della molla in una variazione di accelerazione sfruttando la legge di Hooke:

$$F_e = k \cdot \Delta x$$

Dove F_e rappresenta la forza elastica, k la costante elastica della molla e Δx la compressione della molla, espressa come differenza tra posizione iniziale $x_0 - x$.

Se per esempio si lascia l'accelerometro solo in balia della forza di gravita g, il risultato sarà il seguente:

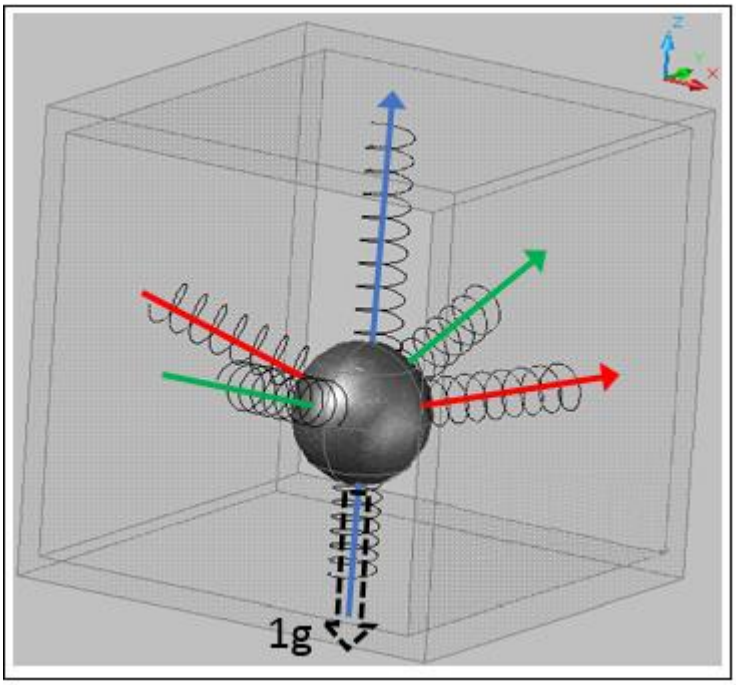

Fig.7 Accelerometro che subisce l'accelerazione di gravita g.

Come si può notare sarà compressa solo la molla relativa all'asse z.

2.3.2 Principio di funzionamento dell'LM35

Per quanto riguarda i sensori di temperatura, esistono modelli diversi che sfruttano altrettanti diversi principi di funzionamento, come la termocoppia, i termistori o le termoresistenze.

Nel caso dell'LM35 si parla di sensori di temperatura integrati. Questa tipologia di sensori fornisce un valore che è direttamente proporzionale alla temperatura. Per esempio il sensore AS 590 fornisce una corrente in uscita che segue la seguente legge:

$$I = K \cdot T$$

Dove I è la corrente, K è una costante ($1\ {}^{\mu A}/_{\circ K}$) e T la temperatura. Quindi I e T sono direttamente proporzionali.

Lo stesso accade per LM35 solo che, al contrario dell'AS 590, fornisce in uscita una tensione e non una corrente.

$$V = K \cdot T$$

2.4 L'accelerometro

Nel precedente capitolo si è visto come la scelta del sensore per il monitoraggio della respirazione sia ricaduta sugli accelerometri. In questo capitolo si esamineranno i vari modelli che il mercato offre, cercando quello con il rapporto qualità/prezzo migliore.

2.4.1 ADXL335

L'accelerometro modello ADXL335 prodotto da *Analog Devices* fornisce l'accelerazione sugli assi x, y e z, con un valore di fondo scala pari a ±3g. Per comunicare con il microcontrollore il sensore adotta il protocollo I^2C. Il chip misura 2.1 cm x 1.6 cm x 1.1 cm 3 pesa circa 3 grammi.

In Fig.8 è riportata la scheda GY-61 che monta questo accelerometro:

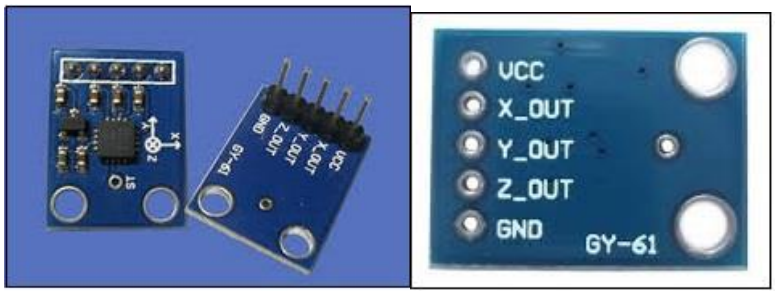

Fig.8 ADXL335

I vantaggi di questo sensore sono il ridotto consumo energetico e la sensibilità. Inoltre le sue prestazioni sono garantite anche in presenza di forti sbalzi di temperatura.

Gli svantaggi sono il rumore di cui soffrono i dati in uscita e il prezzo relativamente alto (8-10 €) se confrontato con altri sensori della stessa tipologia.

SPECIFICATIONS

T_A = 25°C, V_S = 3 V, C_X = C_Y = C_Z = 0.1 µF, acceleration = 0 *g*, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range		±3	±3.6		*g*
Nonlinearity	% of full scale		±0.3		%
Package Alignment Error			±1		Degrees
Interaxis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[1]			±1		%
SENSITIVITY (RATIOMETRIC)[2]	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 3 V	270	300	330	mV/*g*
Sensitivity Change Due to Temperature[3]	V_S = 3 V		±0.01		%/°C

Fig.9 Estratto del datasheet dell'AXL335

2.4.2 ADXL362

Prodotto sempre dalla *Analog Devices*, è una versione "potenziata" del sensore visto in precedenza. Ha diverse funzionalità in più, tra cui la possibilità di scegliere il valore di fondo scala (±2g, ±4g, ±8g), risoluzione a 12 bit e valori di rumore relativamente bassi ($550\mu g/\sqrt{Hz}$).

Il costo di questo pezzo varia tra i 10 e i 15 €.

Parameter	Test Conditions/Comments	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range	User selectable		±2, ±4, ±8		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Sensor Resonant Frequency			3500		Hz
Cross Axis Sensitivity[2]			±1.5		%
OUTPUT RESOLUTION	Each axis				
All *g* Ranges			12		Bits
SENSITIVITY	Each axis				
Sensitivity Calibration Error				±10	%
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	2 *g* range		1		mg/LSB
	4 *g* range		2		mg/LSB
	8 *g* range		4		mg/LSB
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	2 *g* range		1000		LSB/*g*
	4 *g* range		500		LSB/*g*
	8 *g* range		250		LSB/*g*
Sensitivity Change Due to Temperature[3]	−40°C to +85°C		0.05		%/°C
0 *g* OFFSET	Each axis				
0 *g* Output	X_{OUT}, Y_{OUT}	−150	±35	+150	mg
	Z_{OUT}	−250	±50	+250	mg
0 *g* Offset vs. Temperature[3]					
Normal Operation	X_{OUT}, Y_{OUT}		±0.5		mg/°C
	Z_{OUT}		±0.6		mg/°C
Low Noise Mode and Ultralow Noise Mode	X_{OUT}, Y_{OUT}, Z_{OUT}		±0.35		mg/°C
NOISE PERFORMANCE					
Noise Density					
Normal Operation	X_{OUT}, Y_{OUT}		550		µg/√Hz
	Z_{OUT}		920		µg/√Hz
Low Noise Mode	X_{OUT}, Y_{OUT}		400		µg/√Hz
	Z_{OUT}		550		µg/√Hz
Ultralow Noise Mode	X_{OUT}, Y_{OUT}		250		µg/√Hz
	Z_{OUT}		350		µg/√Hz
	V_S = 3.5 V; X_{OUT}, Y_{OUT}		175		µg/√Hz
	V_S = 3.5 V; Z_{OUT}		250		µg/√Hz

Fig.10 Estratto del datasheet dell'ADXL362.

2.4.3 MMA8452Q

Accelerometro triassiale prodotto dalla *NPX* molto simile ai precedenti. La tensione di alimentazione 3.3 V e anch'esso per comunicare con l'MCU usa il protocollo I^2C.

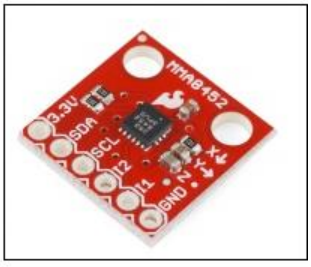

Fig.11 MMA8452Q.

2.4.4 MPU6050

Prodotto dall'*Invensense*, è un accelerometro-giroscopio a sei assi. Fornisce sei output: tre per gli assi dell'accelerazione e tre per quelli del giroscopio. Come i precedenti fa uso del protocollo I^2C per la comunicazione. Di seguito sono elencate le specifiche tecniche:

- Alimentazione richiesta: 3.3-5 V;
- Basso consumo energetico;
- Peso 9 grammi;
- Dimensioni: 2.1 cm x 1.6 cm x 0.3 cm;
- I^2C come protocollo di comunicazione;
- Accelerometro: valori di fondo scala impostabili a ±2g, ±4g, ±8g, ±16g;
- Giroscopio: valori di fondo scala impostabili a ±250 °/sec, ±500 °/s, ±1000 °/s
  ±2000 °/s;
- Alta risoluzione (16 bit);

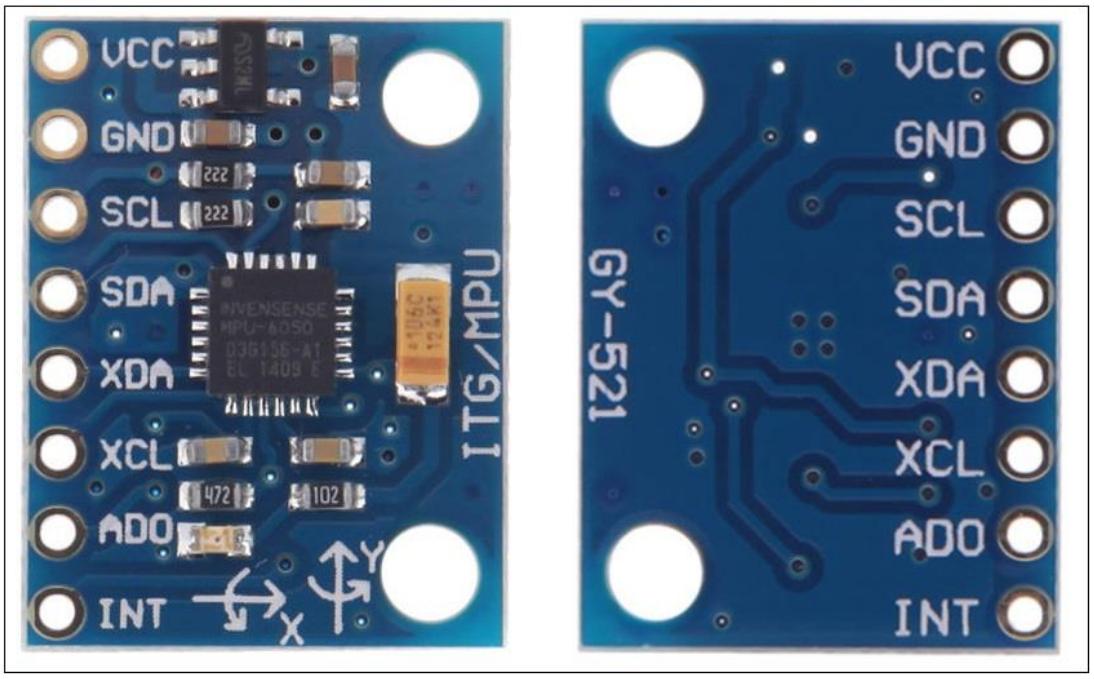

Fig.12 Scheda GY-521 che monta l'MPU6050.

Vantaggi:

Il vantaggio di questo sensore rispetto ai precedenti è il DMP (*Digital Motion Processor)*, un'unità di calcolo dedicata appositamente per il processo di *sensor fusion* degli output dell'accelerometro e del giroscopio ed inoltre svolge il processo di attenuazione del rumore. Inoltre, la scheda monta un sensore di temperatura. Il costo è di circa 5-8 €.

In Fig.14 è riportato un estratto del *datasheet* per informazioni più dettagliate.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	
	FS_SEL=1		±500		°/s	
	FS_SEL=2		±1000		°/s	
	FS_SEL=3		±2000		°/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	
	FS_SEL=1		65.5		LSB/(°/s)	
	FS_SEL=2		32.8		LSB/(°/s)	
	FS_SEL=3		16.4		LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			±2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		±20		°/s	
ZRO Variation Over Temperature	-40°C to +85°C		±20		°/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		°/s	
Linear Acceleration Sensitivity	Static		0.1		°/s/*g*	
SELF-TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	1
GYROSCOPE NOISE PERFORMANCE	**FS_SEL=0**					
Total RMS Noise	DLPFCFG=2 (100Hz)		0.05		°/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to10Hz		0.033		°/s-rms	
Rate Noise Spectral Density	At 10Hz		0.005		°/s/ √ Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		256	Hz	
OUTPUT DATA RATE						
	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME	**DLPFCFG=0**					
ZRO Settling (from power-on)	to ±1°/s of Final		30		ms	

Fig.13 Estratto del datasheet dell'MPU6050.

2.4.5 Scelta del sensore

Indubbiamente il sensore che offre il miglior rapporto qualità/prezzo è l'MPU6050, in termini di sensibilità, dati che fornisce e funzionalità.

2.5 Il sensore di temperatura

Per quanto riguarda i sensori di temperatura la scelta è risultata essere molto più semplice, in quanto il mercato offre sensori con prestazioni simili. Le differenze che si presentano riguardano la sensibilità, la risoluzione e il campo di misura; quest'ultimo poco rilevante in quanto molto ristretto quando si parla della temperatura del corpo umano. Ai fini della progettazione, si è scelto il sensore LM35 prodotto dalla *National Semiconductors*.

2.5.1 Caratteristiche tecniche:

- Range di misura: [-55°; 150°];
- Sensibilità: 10mv/°C;
- Risoluzione: 4.88mV;
- Dimensioni: 5.2mm x 4.19mm x 5.2mm;
- Adatto per applicazioni in remoto;
- Basso costo;
- Buone prestazioni garantite con appena 60 µA (basso consumo energetico).

Electrical Characteristics
(Notes 1, 6)

Parameter	Conditions	LM35			LM35C, LM35D			Units (Max.)
		Typical	Tested Limit (Note 4)	Design Limit (Note 5)	Typical	Tested Limit (Note 4)	Design Limit (Note 5)	
Accuracy, LM35, LM35C (Note 7)	T_A=+25°C	±0.4	±1.0		±0.4	±1.0		°C
	T_A=−10°C	±0.5			±0.5		±1.5	°C
	T_A=T_{MAX}	±0.8	±1.5		±0.8		±1.5	°C
	T_A=T_{MIN}	±0.8		±1.5	±0.8		±2.0	°C
Accuracy, LM35D (Note 7)	T_A=+25°C				±0.6	±1.5		°C
	T_A=T_{MAX}				±0.9		±2.0	°C
	T_A=T_{MIN}				±0.9		±2.0	°C
Nonlinearity (Note 8)	T_{MIN}≤T_A≤T_{MAX}	**±0.3**		**±0.5**	**±0.2**		**±0.5**	°C
Sensor Gain (Average Slope)	T_{MIN}≤T_A≤T_{MAX}	**+10.0**	**+9.8, +10.2**		**+10.0**		**+9.8, +10.2**	mV/°C
Load Regulation (Note 3) 0≤I_L≤1 mA	T_A=+25°C	±0.4	±2.0		±0.4	±2.0		mV/mA
	T_{MIN}≤T_A≤T_{MAX}	**±0.5**		**±5.0**	**±0.5**		**±5.0**	mV/mA
Line Regulation (Note 3)	T_A=+25°C	±0.01	±0.1		±0.01	±0.1		mV/V
	4V≤V_S≤30V	**±0.02**		**±0.2**	**±0.02**		**±0.2**	mV/V
Quiescent Current (Note 9)	V_S=+5V, +25°C	56	80		56	80		µA
	V_S=+5V	**105**		**158**	**91**		**138**	µA
	V_S=+30V, +25°C	56.2	82		56.2	82		µA
	V_S=+30V	**105.5**		**161**	**91.5**		**141**	µA
Change of Quiescent Current (Note 3)	4V≤V_S≤30V, +25°C	0.2	2.0		0.2	2.0		µA
	4V≤V_S≤30V	**0.5**		**3.0**	**0.5**		**3.0**	µA
Temperature Coefficient of Quiescent Current		**+0.39**		**+0.7**	**+0.39**		**+0.7**	µA/°C
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L=0	+1.5		+2.0	+1.5		+2.0	°C
Long Term Stability	T_J=T_{MAX}, for 1000 hours	±0.08			±0.08			°C

Fig.14 Estratto del datasheet dell'LM35

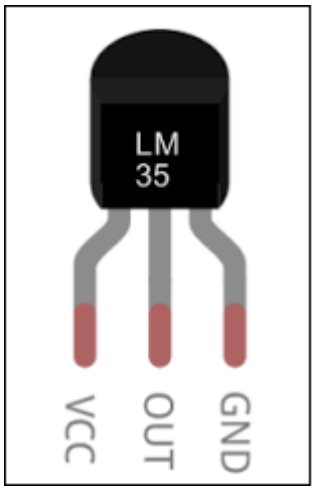

Fig.15 LM35

2.6 Sistema di allarme

Una volta che il dispositivo rileva qualche problema (bassa frequenza respiratoria, bassa temperatura ecc.), deve essere in grado di comunicalo. Per fare ciò, si è deciso di ricorrere ad allarmi sia acustici che visivi.

Per quanto riguarda l'allarme acustico si userà un *Buzzer*, o piezo, che emetterà un segnale sonoro evidente in caso di pericolo. Mentre per quanto riguarda il segnale visivo si utilizzeranno due LED: uno verde e uno rosso. Il LED verde sarà acceso quando tutti i parametri vitali rilevati sono nella norma, mentre quello rosso in caso contrario.

Si utilizzerà anche un terzo LED, giallo, che ci dirà quando il dispositivo è acceso.

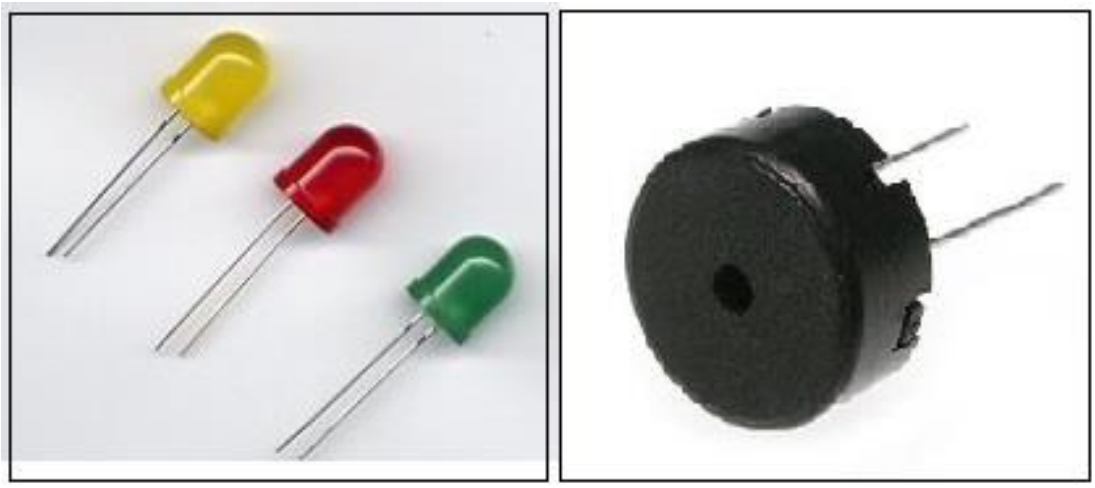

Fig.16: A sinistra LED rosso, giallo, verde; a destra: buzzer.

2.7 Schema a blocchi.

Infine, dopo aver scelto accuratamente i componenti che faranno parte del nostro sistema, si schematizza il sistema e si riassume nel seguente modo:

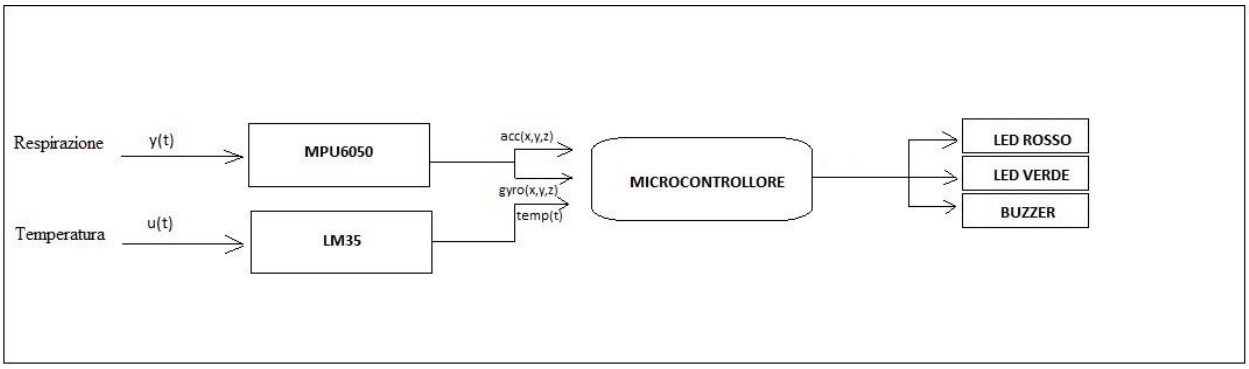

Fig.17 Schema a blocchi riassuntivo.

Dove:

- y(t) rappresenta il segnale che proviene dalla respirazione, ovvero il movimento a livello toracico o addominale provocato dallo stesso;
- u(t) rappresenta la temperatura corporea;
- acc(x,y,z) rappresenta l'accelerazione lungo i tre assi;
- gyro(x,y,z) rappresenta la velocità angolare lungo i tre assi;

2.8 Applicazione del dispositivo sul paziente

Una volta realizzato, il dispositivo dovrà essere applicato come una clip sul pannolino del neonato. Si è scelta questa particolare posizione, perché si vuole sfruttare l'inclinazione che viene provocata sul dispositivo a seguito delle fasi della respirazione, come riportato in fig.19 e in fig.20:

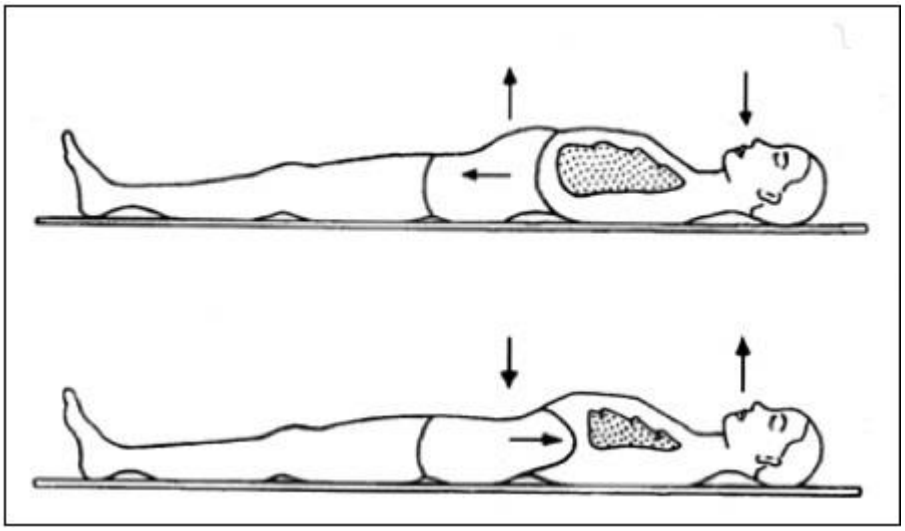

Fig.18 Figura che schematizza il le due fasi della respirazione.

Così facendo, la parte inferiore del dispositivo sarà ancorata sul pannolino del neonato, mentre la parte superiore subirà l'inclinazione di un certo angolo θ dal movimento generato dall'addome durante la respirazione. Questa variazione di angolo andrà a inclinare il dispositivo e di conseguenza solleciterà l'accelerometro che registrerà i dati che successivamente verranno elaborati e monitorati:

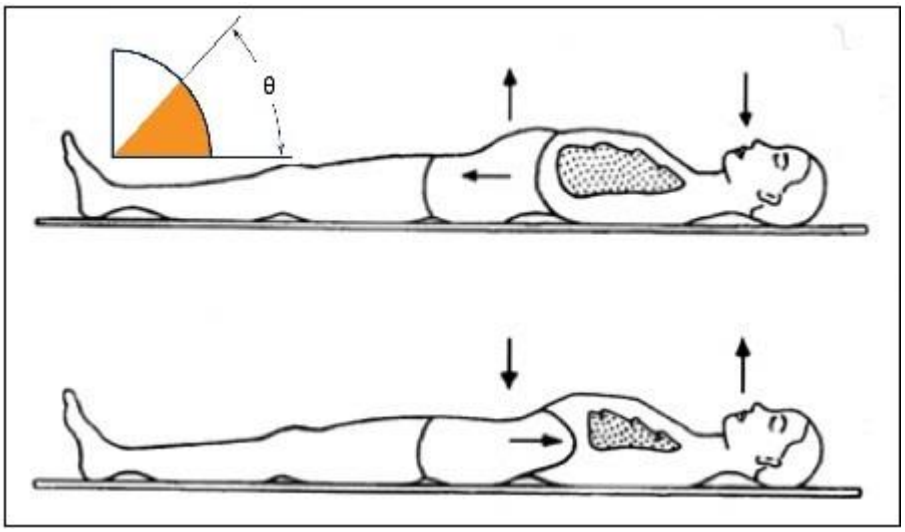

Fig.19

CAPITOLO III

Montaggio hardware. Prove preliminari e calibrazione dei sensori. Sketch Arduino.

3.1 Montaggio dei componenti hardware e cablaggio.

Nel precedente capitolo si è discusso della scelta dei componenti che andranno a far parte del dispositivo, mentre adesso si vedrà come montarli sulla breadboard per effettuare le prime prove preliminari e successivamente effettuare la calibrazione dei sensori.

3.1.1 Montaggio MPU6050

Nel paragrafo 2.4.4 figura 13, è rappresentato il *pinout* dell'accelerometro. Esso è dotato di:

- Pin Vcc dedicato all'alimentazione, che andrà collegato ai 3.3v che fornisce la scheda; lo stesso si farà per il pin GND (ground) a cui si collegherà la messa a terra;
- Pin SCL e SDA: questi servono per far comunicare l'accelerometro e la scheda tramite il protocollo I^2C. La scheda Arduino Mega dispone di due pin dedicati per questa operazione, quindi si collegheranno i pin dell'accelerometro con i rispettivi pin della scheda;
- Pin XDA e XCL: questi servono per effettuare la *sensor fusion* nel caso in cui si volessero combinare i dati provenienti dall'accelerometro con i dati provenienti da un altro sensore, in genere un magnetometro. Saranno gli unici due pin non utilizzati;
- Pin INT che serve per attivare l'unità di calcolo che effettua il *Digital Motion Processor* (DMP). Si collega al pin digitale n. 2.

Nella successiva figura sono rappresentati tutti i collegamenti precedentemente elencati.

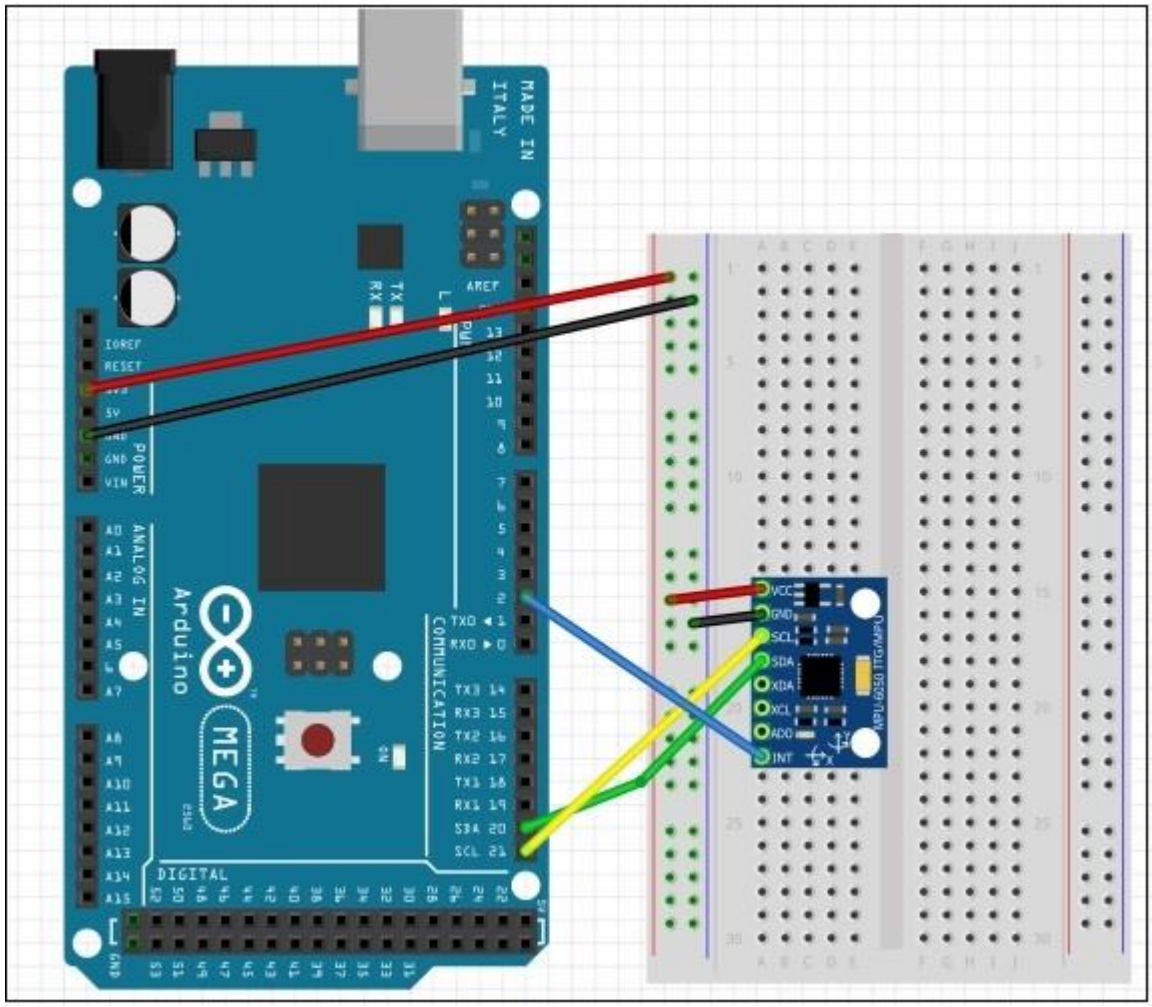

Fig.1 Collegamento dell'MPU6050.

3.1.2 Montaggio LM35

In Fig.16 del paragrafo 2.5.1 è rappresentato il *pinout* del sensore di temperatura LM35. Esso è dotato di due pin per l'alimentazione, Vcc e GND, che si andrà a collegare rispettivamente ai 3.3v e alla messa a terra forniti dalla scheda e da un pin dedicato all'uscita analogica, che si collegherà al pin A0 della scheda Arduino. Di seguito è riportata la rappresentazione grafica.

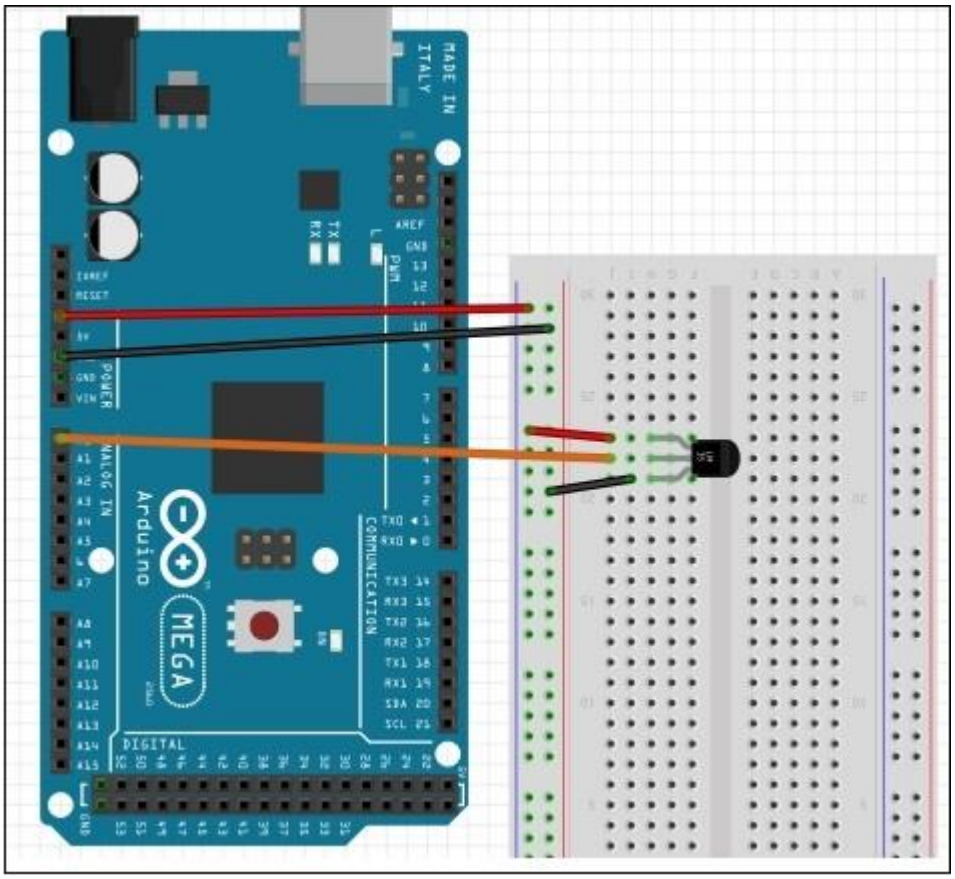

Fig.2 Collegamento dell'LM35.

3.1.3 Montaggio dei LED

Per quanto riguarda i LED si è deciso di utilizzare uno verde e uno rosso. L'accensione del primo servirà a comunicare che la situazione rilevata dal dispositivo è normale, ovvero non viene rilevato alcun pericolo; il secondo risulterà acceso in caso di pericolo.

I LED (*Light Emitting Diode*) sono dei diodi che emettono luce quando sono attraversati da corrente, che in questi elementi può scorrere solo in una direzione. I diodi sono dei bipoli, ovvero degli elementi circuitali che possiedono due terminali che prendono il nome di anodo e catodo. L'anodo è il piedino più lungo del LED ed è quello che riceve l'alimentazione; il catodo è il piedino più corto e va collegato alla messa a terra. Poiché si è preferito controllare i LED attraverso i pin digitali di Arduino, si collega l'anodo dei LED ai pin digitali, nel nostro caso 3 e 4,

e il catodo alla messa a terra. Il *datasheet* riporta che la corrente massima che il LED può sopportare affinché non venga bruciato è di circa 23 mA. I pin digitali forniscono 5 v se sono nello stato HIGH, mentre 0 se sono nello stato LOW. Quindi, nel caso della tensione massima, per garantire che la corrente sia quella sopracitata, si dovrà usare una resistenza che si andrà a ricavare con la legge di Ohm.

$$V = RI \rightarrow R = \frac{V}{I} = \frac{5}{0.023} = 217.39\ \Omega$$

Si sceglie dunque R=220Ω sia per cautelarsi da eventuali disturbi sia perché in commercio è la resistenza che più si avvicina al valore desiderato. Di seguito è riportato lo schema circuitale.

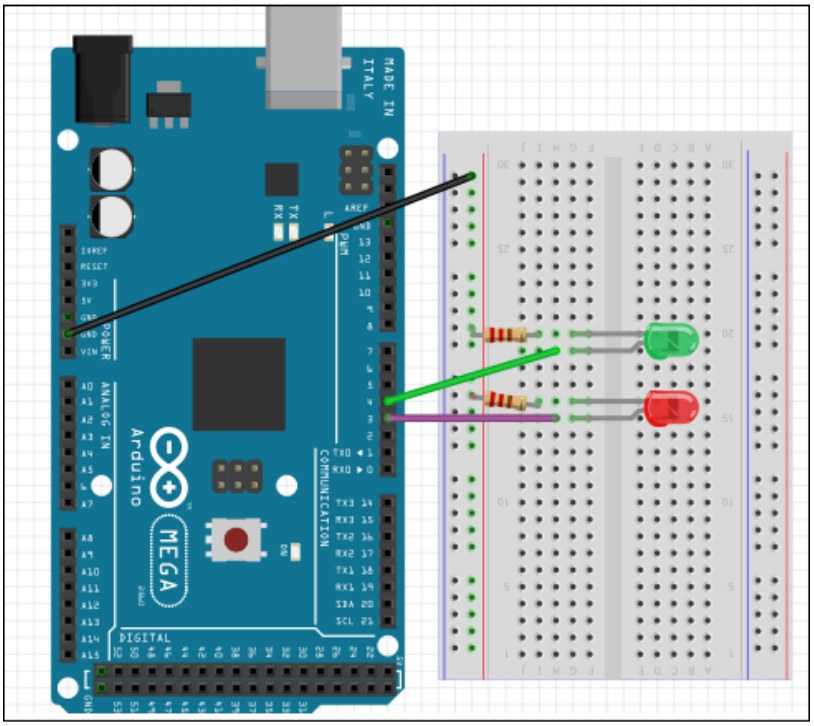

Fig.3 Collegamento dei LED.

3.1.4 Montaggio del buzzer

Resta in fine da vedere come montare il *buzzer*, l'allarme sonoro. Un *buzzer* è un bipolo, dove un'uscita va collegata a terra mentre l'altra ad un pin digitale, in questo caso al pin 5 della scheda Arduino.

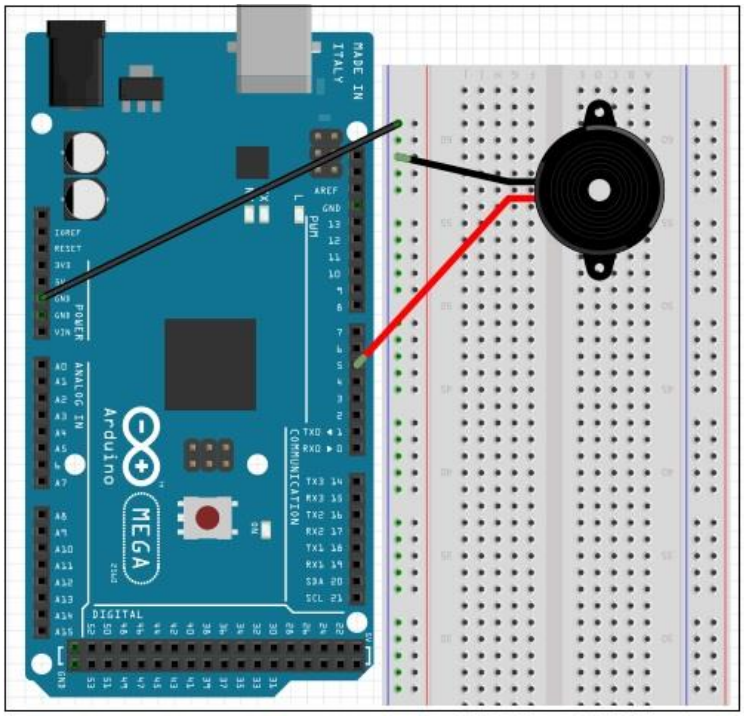

Fig.4 Collegamento del buzzer.

Nella seguente figura è mostrato lo schema circuitale completo:

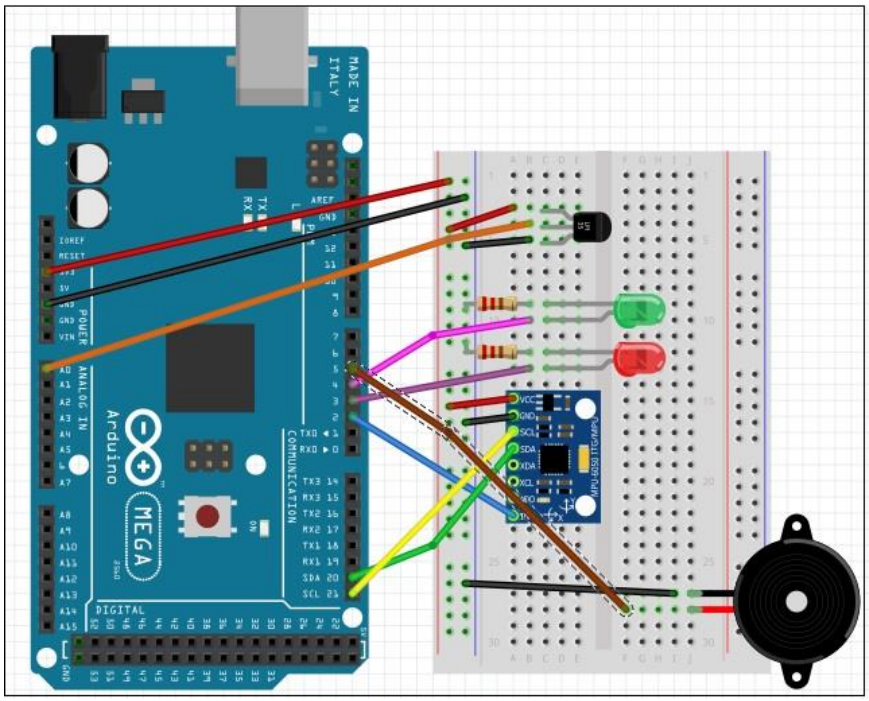

Fig.5 Collegamenti schema circuitale completo

3.2 Prove preliminari.

Dopo aver visto come collegare i vari componenti si possono effettuare le prime prove, volte a testare l'efficacia di ognuno dei pezzi.

3.2.1 Prova LM35 e visualizzazione output.

Si è visto che il sensore di temperatura sfrutta un'uscita analogica per comunicare al microcontrollore i dati che rileva. Questo valore oscillerà tra 0 e 1024 in base alla differenza di potenziale che vi è ai capi del sensore. Quest'ultima a sua volta dipende dalla temperatura; il *datasheet* riporta che per ogni 10mV corrisponde un grado Celsius. Si procederà in tal modo:

- Leggere il valore analogico tramite la funzione *analogRead();*
- Dividere tale valore per 1024 e poi moltiplicarlo per il valore di tensione massima che può esserci ai capi del sensore;
- Poiché ogni 10mV corrispondono ad 1 C° divideremo il valore di cui sopra per 10.

```
float Valore_analogico;
float Volt;
float Temperatura;

void setup() {
 Serial.begin(9600); // Avvio comunicazione seriale
}

void loop() {
  Valore_analogico = analogRead(A0); // Lettura del valore analogico
  Volt = (Valore_analogico/1024)*5000; // Conversione in volt
  Temperatura = Volt/10; // Conversione in C°

  Serial.print(Temperatura);Serial.println(" C"); // Stampa sul monitor la temperatura espressa in gradi Celsius
}
```

Fig.6 Sketch per visualizzare la temperatura sul monitor seriale.

3.2.2 Prove preliminari MPU6050.

Per quanto riguarda l'accelerometro il discorso è più complesso. Prima di vedere direttamente come ottenere i dati che fornisce questo sensore, si vedrà più nel dettaglio come funziona il protocollo I^2C.

3.2.3 Il protocollo I^2C.

Nel capitolo precedente si è visto che per comunicare con la scheda il sensore utilizza il protocollo I^2C (*Inter Integrated Circuit*), un pratico sistema di comunicazione che utilizza soltanto due fili. Questo protocollo è stato ideato nel 1982 dalla Philips per far comunicare tra loro diversi circuiti integrati. L'I^2C prevede un master e uno o più slave, o anche più master nel caso di sistemi più complessi, che condividono due linee di chiamata: SDA (Serial DAta) e SCL (Serial CLock).

Due fili sono sufficienti perché ogni slave ha un suo indirizzo e, quando il master vuole comunicare con un circuito periferico, prima di tutto annuncia con quale dispositivo vuole dialogare, in modo che questo si prepari alla comunicazione. In tal modo, si risparmiano i collegamenti CS o SS richiesti, per esempio, nel protocollo SPI. La velocità di trasmissione è tipicamente 100 Kb/s.

Come già visto in precedenza, Arduino Mega dispone di due pin dedicati per questo tipo di comunicazione.

Ci sono quattro possibili modi di comunicazione:

- Un master trasmette – controlla il clock e invia i dati agli slave;
- Un master riceve – controlla il clock ma riceve i dati dallo slave;
- Lo slave trasmette – il dispositivo non controlla il clock ma invia i dati al master;
- Lo slave riceve – il dispositivo non controlla il clock e riceve i dati dal master.

Il master inizia la comunicazione inviando lo *start bit* seguito dall'indirizzo dello slave con cui vuole comunicare. Segue un bit che indica se il master vuole scrivere informazioni (non sempre è consentito) o leggere informazioni. Nel primo caso il bit è tenuto nello stato basso

(LOW) altrimenti sarà nello stato alto (HIGH). In Fig.7 è riportato uno schema di comunicazione.

Per gestire al meglio questa comunicazione nell'IDE si farà uso della libreria "Wire.h" che implementa tutti i metodi necessari per ottenere i dati dal sensore.

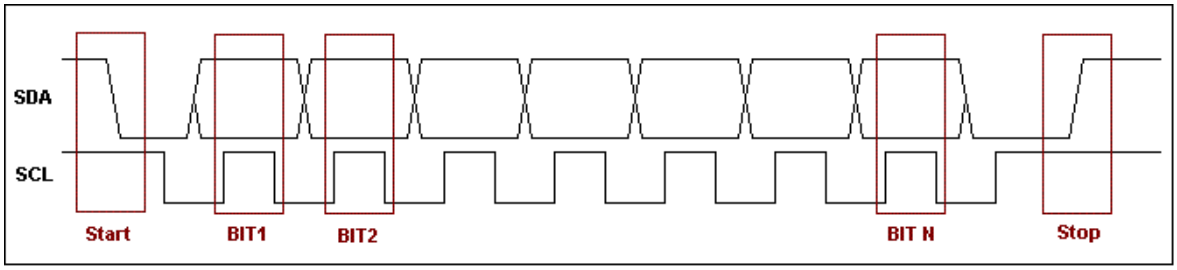

Fig.7 Schema protocollo I^2C

3.2.4 Sketch per MPU6050

Prima di iniziare con lo sketch vero e proprio, si andrà a verificare che tutti i collegamenti sono stati effettuati in maniera corretta implementando un codice di verifica disponibile sul sito ufficiale di Arduino:

```
#include <Wire.h>


void setup()
{
  Wire.begin();

  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan

}
```

Fig.8 Codice di verifica dell'MPU6050.

Se tutti i collegamenti e il sensore dovessero funzionare correttamente, si avrà questo output sul monitor seriale:

Fig.9 Messaggio che conferma il corretto funzionamento dell'MPU6050.

Adesso si può procedere con lo sketch vero e proprio per visualizzare gli output del sensore. Gli step da seguire saranno i seguenti:

- Consultare il DS per vedere quali registri contengono quali dati (Fig.10);
- Nel metodo *setup()* verrà inizializzata la connessione grazie ai metodi forniti dalla libreria Wire.h; verrà settato l'indirizzo del registro di partenza e verrà inviato un bit per avviare la comunicazione (Fig.11a);
- Nel metodo *loop()* si richiederanno allo slave tutti i dati desiderati (Fig.11b);
- Si stamperanno i vari output sul monitor seriale (Fig.11c).

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							

Fig.10 Registri d'interesse dell'MPU6050.

```ino
#include<Wire.h>
const int MPU=0x68;  // I2C indirizzo del registro dell'MPU6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ; // Inizializzo le variabili che conterranno gli output del sensore

void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);  // Indirizzo del registro che contiene il bit che avvia la comunicazione
  Wire.write(0);     // Inviamo un bit per avviare la comunicazione
  Wire.endTransmission(true); // Chiudiamo la comunicazione
  Serial.begin(9600); // Avviamo in monitor seriale con velocità di 9600 baud/s
}
```

a)

```ino
void loop(){

  Wire.beginTransmission(MPU);
  Wire.write(0x3B);  //Partiamo dal registro 0x3B che contiene i dati relativi all'asse x dell'accelerometro
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true);  // richiediamo la lettura di 14 registri
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
```

b)

```ino
  //Stampiamo i dati relativi ai tre assi dell'accelerometro:
  Serial.print("Accelerometer: ");
  Serial.print("X = "); Serial.print(AcX);
  Serial.print(" | Y = "); Serial.print(AcY);
  Serial.print(" | Z = "); Serial.println(AcZ);
  //Dal DS convertiamo la temperatura in C°
  Serial.print("Temperature: "); Serial.print(Tmp/340.00+36.53); Serial.println(" C ");

  //Stampiamo i dati relativi ai tre assi del giroscopio:
  Serial.print("Gyroscope: ");
  Serial.print("X = "); Serial.print(GyX);
  Serial.print(" | Y = "); Serial.print(GyY);
  Serial.print(" | Z = "); Serial.println(GyZ);
  Serial.println(" ");
```

c)

Fig.11: a) Inizializzazione della connessione con l'MPU6050; b) Richiesta dati del sensore; c) Comandi per stampare su monitor seriale i dati richiesti;

Nella fig.11b e 11c è riportato il codice che viene eseguito continuamente finché è presente alimentazione. Per prima cosa si avvia la comunicazione partendo dal registro “0x3B”, quello dove vengono contenuti i dati relativi all’asse x dell’accelerometro, successivamente si imposta come falso il metodo “Wire.endTrasmission()” altrimenti la connessione verrebbe chiusa alla fine del primo ciclo di loop. Il metodo “Wire.requestFrom(MPU,14,true)” ha in entrata l’indirizzo di start dell’MPU e il numero delle richieste che si intende fare. Ogni registro contiene il byte sia nello stato LOW che nello stato HIGH del valore contenuto; poiché si vuole visualizzare un totale di 7 dati (3 per l’accelerazione, 3 per la velocità angolare, 1 per la temperatura) il numero di richieste totali da effettuare sarà di 14. Successivamente si salvano i valori nelle rispettive variabili. Si noti bene che poiché il sensore è di 16 bit, non si possono salvare i dati in una variabile *integer* normale di Arduino, che ha una dimensione di 2 byte, ma si dovrà utilizzare una variabile intera di 16 byte, il cui comando è “int16_t”, come riportato nella terza linea di codice della fig.11 a).

L’output sarà il seguente:

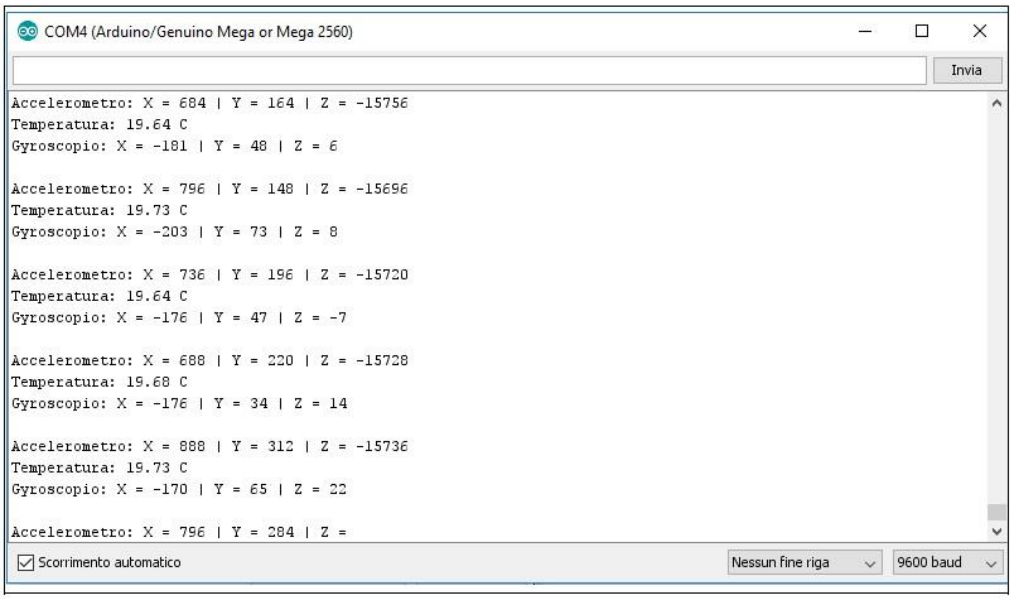

Fig.12 Monitor seriale che riporta i dati dell’accelerometro, del giroscopio e del sensore di temperatura.

Si noti che questi non sono altro che i valori grezzi che fornisce l'MPU6050, nel prossimo paragrafo verrà discusso come ottenere i dati con la relativa unità di misura e la calibrazione dello stesso.

3.3 Calibrazione MPU6050

Come si è visto dall'ultima immagine i valori dati in output dal sensore non sembrano avere un senso preciso, ci si aspetterebbero dei valori con le rispettive unità di misura: g per l'accelerazione e °/s per la velocità angolare.

I valori riportati, infatti, non sono altro che i valori *raw* del sensore (grezzi) e quindi occorre eseguire una giusta conversione. Il ragionamento da seguire sarà il medesimo di quello usato per ottenere il valore °C per il sensore di temperatura, ma con qualche piccola differenza. Infatti l'MPU6050, come già esposto, è un sensore a 16 bit e quindi il suo *range* sarà di 2^{16} = 65536 possibili valori. Poiché il sensore restituisce anche i valori negativi che possono assumere giroscopio e accelerometro, il *range* sarà [-32768; +32768].

Una volta definito il *range* di misura, bisogna stabilire il valore di fondo scala ovvero il valore massimo misurabile dal sensore, che di default corrisponde a ±2g per l'accelerometro, mentre ±250 °/s per il giroscopio. Quindi se per esempio sull'asse x dell'accelerometro agisce un'accelerazione di +2g in output si avrà +32768, di conseguenza si imposterà la proporzione per ottenere il valore con la relativa unità di misura.

Si considerino a titolo esemplificativo, i seguenti valori:

```
Accelerometro: X = 736 | Y = 196 | Z = -15720
Temperatura: 19.64 C
Gyroscopio: X = -176 | Y = 47 | Z = -7
```

Fig.13 Raw output MPU6050.

Si ha l'asse x dell'accelerometro che risulta $x_A = 736$, mentre l'asse x del giroscopio $x_G = -176$. Per convertire tali valori i dati con le relative unità di misura basterà impostare la seguenti proporzioni:

$$2 : 32768 = x_A : y \ ;$$

$$250 : 32768 = x_G : y \ ;$$

Che rappresentano rispettivamente le proporzioni per convertire i dati dell'accelerometro e del giroscopio. Al primo membro si avrà il rapporto tra il valore di fondo scala e il relativo estremo del range di misura; al secondo membro, x rappresenta il valore convertito mentre y il valore dato in output sul monitor seriale.

In base a queste informazioni, considerando i valori evidenziati in rosso in fig.13, si avrà:

$$2 : 32768 = x_A : 736 \ ;$$

$$250 : 32768 = x_G : (-176) \ ;$$

Pertanto:

$$x_A = \frac{2 \cdot 736}{32768} = 0.0449 \ g;$$

$$x_G = \frac{250 \cdot (-176)}{32768} = -1.324 \ \frac{^\circ}{s};$$

Di seguito sono riportati i metodi che effettuano la conversione dei valori dell'MPU6050:

```
float M1(int16_t x){
  float y = x;
  return (y*2)/(32768);
}
float M2(int16_t x){
  float y = x;
  return (y*250)/(32768);
}
```

Fig.14 Metodi per la conversione.

Il metodo M1 serve per convertire i dati forniti dall'accelerometro, mentre M2 per quelli forniti dal giroscopio. Sono metodi "float" perché restituiscono valori non interi ma con la virgola. In entrata hanno una variabile intera a 16 bit che viene subito salvata in una variabile temporale di tipo float altrimenti, se si effettuassero le operazioni successive su variabili intere, si perderebbe l'informazione relativa ai numeri decimali. Infine viene effettuata la conversione.

La Fig.15 e mostra dove vengono richiamati i metodi di cui sopra.

```
//Stampiamo i dati relativi ai tre assi dell'accelerometro:
Serial.print("Accelerometro: ");
Serial.print("X = "); Serial.print(M1(AcX));
Serial.print(" | Y = "); Serial.print(M1(AcY));
Serial.print(" | Z = "); Serial.print(M1(AcZ));Serial.println("   [g]");
//Dal DS convertiamo la temperatura in C°
Serial.print("Temperatura: "); Serial.print(Tmp/340.00+36.53); Serial.println(" C ");

//Stampiamo i dati relativi ai tre assi del giroscopio:
Serial.print("Gyroscopio: ");
Serial.print("X = "); Serial.print(M2(GyX));
Serial.print(" | Y = "); Serial.print(M2(GyY));
Serial.print(" | Z = "); Serial.print(M2(GyZ));Serial.println("  [gradi/s]");
Serial.println(" ");
```

Fig.15 Codice con i metodi di conversione implementati.

Finalmente:

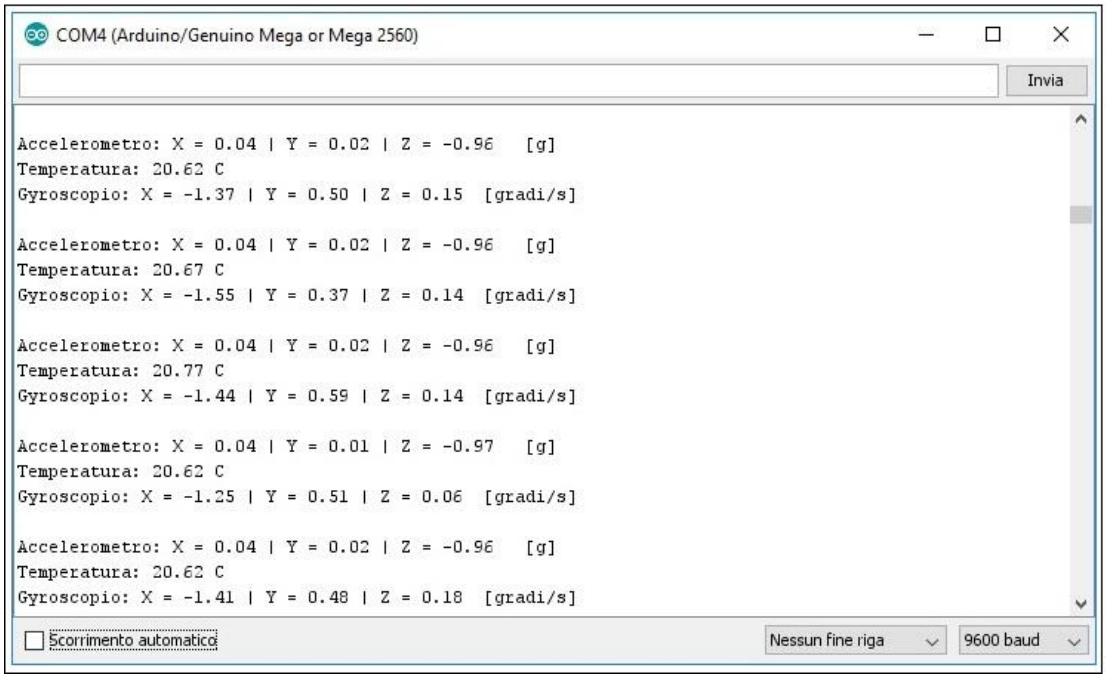

Fig.16 Monitor seriale che riporta i dati convertiti nelle rispettive unità di misura.

Osservando questi valori ottenuti tenendo fermo il sensore ci si accorge che oltre ad avere un leggero offset, essi soffrono di un leggero rumore. Lo studio e di quest'ultimo sarà oggetto di discussione nel prossimo capitolo.

3.4 Sketch Arduino per il monitoraggio della respirazione.

3.4.1 Librerie utilizzate.

A questo punto si è pronti per implementare il tutto sotto forma di programma che verrà successivamente caricato sulla scheda Arduino. Nel corso della programmazione si è fatto utilizzo di diverse librerie ufficiali, tra cui:

- "Wire.h" per gestire la comunicazione tramite il protocollo I^2C;
- "I2Cdev.h" e "MPU6050.h" per gestire il *Digital Motion Processor*.

```
// Include la libreria I2C
#include "I2Cdev.h"

//Include la libreria "MPU6050.h"
#include "MPU6050_6Axis_MotionApps20.h"


//Include la libreria "Wire.h"
    #include "Wire.h"
```

Fig.17

3.4.2 Inizializzazioni variabili utilizzate.

Successivamente si dichiareranno le variabili che si intendono utilizzare:

```
// VARIABILI DATI OUTPUT
float off_ax , off_ay , off_az , off_gx , off_gy , off_gz ;
int16_t ax, ay, az;
int16_t gx, gy, gz;
float A_x,A_y,A_z,G_x,G_y,G_z;

//VARIABILI UTILI PER IL MONITORAGGIO
unsigned long tempo, t_t_mov, t_u_mov = 0;  // Inizializzo la variabili unsigned di tipo long a 32 bit
int t_ins ,t_ins_real,t_esp_real, t_esp , t_resp , timer_i , timer_e = 0 ;
int frequenza_resp;
int Deadline = 20000;
```

Fig.18 Inizializzazioni variabili.

Si noti che le variabili dedicate per i dati dell'MPU6050 sono dello tipo degli output dello stesso, ovvero variabili intere a 16 bit. Mentre le variabili che dovranno tener traccia del tempo saranno di tipo *unsigned*, poiché dovranno essere solo positive, e di tipo *long* a 32 bit poiché altrimenti non potrebbero tenere traccia di più di 32768 millisecondi.

3.4.3 Metodo setup(). *Digital Motion Processor (DMP).*

Come già spiegato nel capitolo 2, si farà utilizzo di un'unità di calcolo interna all'MPU6050 che servirà per eliminare il problema del rumore. Il DMP processa i dati *raw* del sensore attraverso un filtro di Kalman, un oggetto matematico molto complesso in grado di stimare quale sia il valore reale del dato che si sta analizzando. La casa produttrice dell'MPU6050 non dà indicazioni chiare sul *datasheet* su come utilizzarlo; per questo motivo si è utilizzata la libreria ufficiale e il codice standard di attivazione scritte da Jeff Rowberg.

3.4.4 Calcolo degli offset.

Una volta inizializzato il DMP, il prossimo passo da effettuare sarà quello di calcolare gli offset presenti su ogni asse del sensore; offset perlopiù dovuti ad artefatti di corretto posizionamento del sensore. Per calcolarli è stato sufficiente acquisire un campione di n valori a sensore fermo e fare successivamente la media:

```
// Calcolo degli off_set
            for ( int i = 0 ; i < 30 ; i++ ){
            mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
            off_ax = off_ax + ax;
            off_ay = off_ay + ay;
            off_az = off_az + az;
            off_gx = off_gx + gx;
            off_gy = off_gy + gy;
            off_gz = off_gz + gz;
            }
            off_ax = off_ax/491520;
            off_ay = off_ay/491520;
            off_az = off_az/491520;
            off_gx = off_gx/3932.16;
            off_gy = off_gy/3932.16;
            off_gz = off_gz/3932.16;
```

Fig.19 Calcolo offset.

Come si può vedere nella figura 19, con un ciclo for si è richiamata trenta volte la funzione “mpu.getMotion6()” e si è aggiunto ad ogni ciclo il valore di ogni asse nella variabile off_. Successivamente si è calcolata la media e si sono convertiti i valori. Infatti, ogni valore relativo all’accelerometro è stato diviso per 491520 = 30·16384, dove il primo termine è relativo al calcolo della media, infatti la media è stata calcolata su un campione di 30 valori; il secondo, come già spiegato, serve per convertire i valori in g.

La medesima cosa è stata fatta per gli offset dei giroscopi, ovvero si è diviso per 3932,16 = 30·131,072, dove appunto l’ultimo termine serve per convertire l’output del giroscopio in gradi/s.

Infine si stampano sul monitor seriale gli offset calcolati:

```
// Stampo su monitor serial gli offset
          Serial.print("Offset: ");
           Serial.print(off_ax);
           Serial.print("\t");
           Serial.print(off_ay);
           Serial.print("\t");
           Serial.print(off_az);
           Serial.print("\t");
           Serial.print(off_gx);
           Serial.print("\t");
           Serial.print(off_gy);
           Serial.print("\t");
           Serial.println(off_gz);
```

Fig.20 Stampa su monitor seriale gli offset.

3.4.5 Configurazione dei pin digitali.

Come ultima cosa del metodo *setup()*, si sono configurati i pin d’interesse, che in questo caso ci serviranno per controllare gli attuatori, ovvero i LED e il buzzer, che sono stati configurati tutti come output. Per fare ciò basta richiamare la funzione “pinMode()” che in entrata ha il numero del pin digitale di Arduino che si sta utilizzando e il tipo, che può essere INPUT o OUTPUT:

```
// SETTO I PIN D'INTERESE COME OUTPUT.
pinMode(3, OUTPUT); //LED BLU
pinMode(4, OUTPUT); //LED ROSSO
pinMode(5, OUTPUT); //PIEZO
```

Fig.21

3.4.6 Metodo loop(). Acquisizione dati accelerometro e compensazione.

Una volta finito con le impostazioni generali effettuate nel metodo setup(), si comincia a programmare il metodo loop() che, come già spiegato nel capitolo 2, è quel metodo che viene fatto eseguire costantemente dalla scheda Aduino fin quando è presente alimentazione. Per prima cosa si acquisiscono i dati dell'MPU6050, si convertono nelle corrispettive unità di misura e infine si effettua la compensazione degli offset precedentemente calcolati:

```
//------COMPENSAZIONE VALORI------
// Utilizzando il DMP(Digital Motion Processor) interno dell'MPU6050 ottengo già i valori filtrati con il filtro di Kalman

            mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); //Comando per richiedere i valori dell'accelerometro e del giroscopio processati dal DMP
              A_x = ax;
              A_y = ay;
              A_z = az;
              G_x = gx;
              G_y = gy;
              G_z = gz;
```

```
// Divido il valore ottenuto per 16384. Questo perchè il sensore è di 16 bit e quindi il range dei valori è  [-32768;+32768]. Essendo i valori di fondo scala "+/-2g"
//per l'accelerometro e "+/-250 gradi/s" per il giroscopio, la più piccola variazione o risoluzione si calcola come range di misura diviso il valore di fondo scala.
//Per l'accelerometro avrò 32768/2 = 16384;
//Per il giroscopio invece 32768/250 = 131.072;
//Dividendo quindi i valori forniti dai sensori per i valori appena ottenuti avremo i valori nelle loro rispettive unità di misura.
//Successivamente sottraggo l'offset precedentemente calcolato
            A_x = (A_x/16384)- off_ax;
            A_y = (A_y/16384) - off_ay;
            A_z = (A_z/16384) - off_az;
            G_x = (G_x/131.072) - off_gx;
            G_y = (G_y/131.072) - off_gy;
            G_z = (G_z/131.072) - off_gz;
//------FINE COMPENSAZIONE VALORI------
```

Fig.22 Acquisizione dati dell'MPU6050 processati dal DMP e successiva compensazione.

I valori così trattati appariranno in questo modo sul monitor seriale:

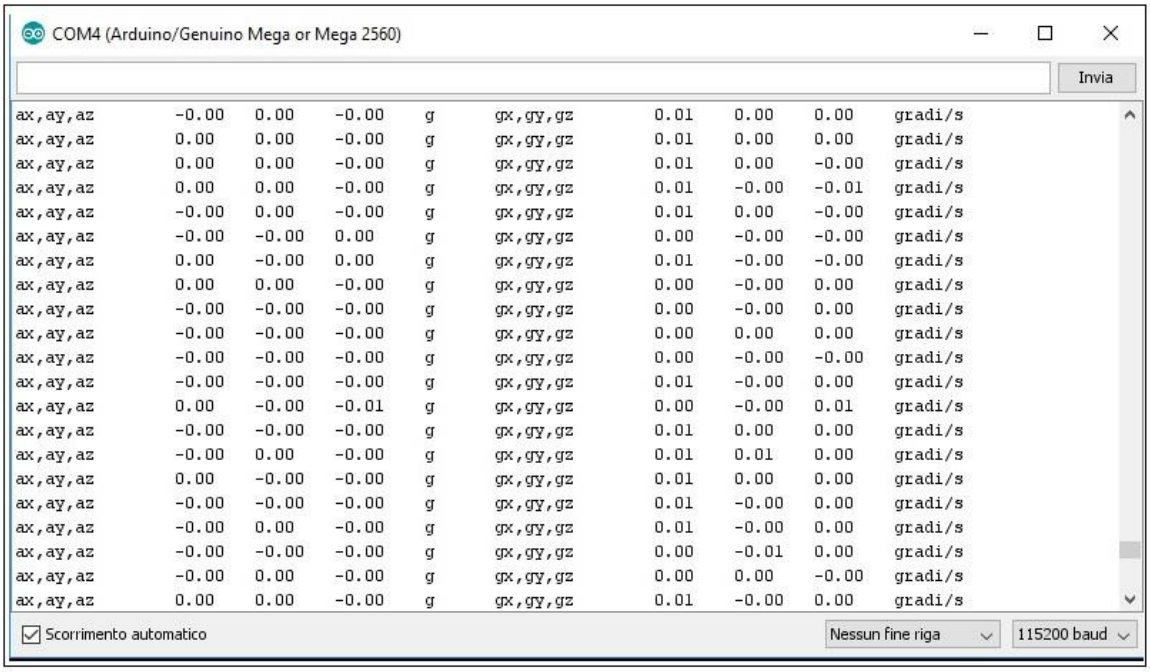

Fig.23 Output su monitor seriale.

3.4.7 Monitoraggio respirazione.

Una volta finito il processo di calibrazione, tutti i valori saranno pari a 0 qualunque sia la posizione iniziale del sensore:

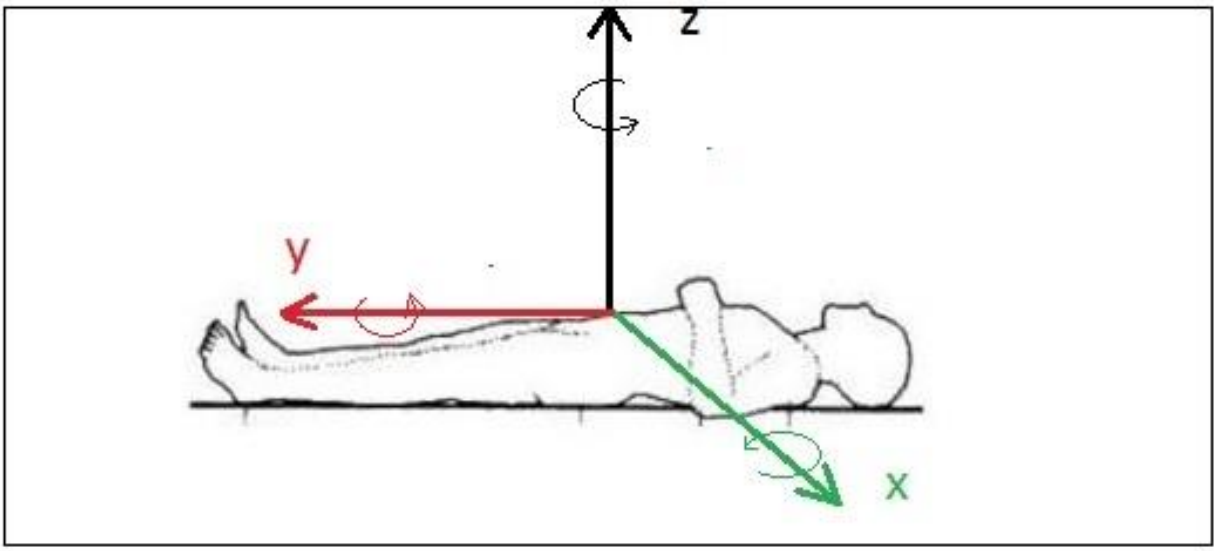

Fig.24 Assi di rotazione dell'MPU6050.

Il dispositivo sarà in grado dunque in grado di:

- a) Discriminare se si sta trattando della fase inspiratoria o espiratoria;

- b) Azionare il sistema d'allarme nel caso in cui non venisse rilevato alcun movimento negli ultimi venti secondi;
- c) Calcolare il tempo relativo all'inspirazione e all'espirazione, e quindi il tempo totale di ogni respiro;
- d) Calcolare la frequenza respiratoria.

3.4.8 Discriminazione fase inspiratoria e fase espiratoria.

Per quanto riguarda il punto a) riportato precedentemente, le due distinte fasi della respirazione verranno discriminate come segue:

- Se la velocità angolare lungo l'asse x risultasse positiva, per come sono stati impostati gli assi cartesiani, allora il dispositivo sta ruotando in verso concorde con l'asse, quindi si è in fase di inspirazione;
- Al contrario, se dovesse risultare negativa, allora il dispositivo si troverà in recessione e quindi la fase sarà quella espiratoria.

```
//------INIZIO MONITORAGGIO------
    tempo = millis ();

if( G_x > 1 ){
timer_i = tempo;
t_ins = tempo - timer_e;
t_esp_real = t_esp;
Serial.print("Fase inspiratoria ");Serial.print(" ");Serial.println(t_ins);
}
if(G_x < -1 ){
timer_e = tempo;
t_esp = tempo - timer_i;
t_ins_real = t_ins;
Serial.print("Fase espiratoria ");Serial.print(" ");Serial.println(t_esp);
}
```

Fig.25 Monitoraggio della velocità angolare rispetto all'asse x.

In figura 25 è riportato il controllo sulla variabile "G_x" che è quella che riporta l'output della velocità angolare rispetto all'asse x: se è maggiore di 1°/s, allora il programma stamperà sul monitor seriale il messaggio

“Fase inspiratoria”, altrimenti se sarà minore di -1°/s stamperà “Fase espiratoria”. Si noti bene che queste velocità non sono state ottenute sperimentalmente, ma sono state scelte in base alla sensibilità del sensore per garantire un corretto funzionamento effettuando delle semplici prove manuali. Nel capitolo 4 il dispositivo verrà testato su dei soggetti e successivamente si determineranno empiricamente velocità tali da consentire il corretto funzionamento su chiunque indossi il dispositivo senza incombere in falsi allarmi e/o malfunzionamenti.

3.4.9 Azionamento del sistema d’allarme.

Il sistema di allarme si aziona quando il dispositivo non rileva nessun movimento per più di venti secondi. Esso è composto da:

- Un allarme visivo, che si manifesta con il lampeggiamento del LED rosso;
- Un allarme sonoro, provocato da un buzzer.

Per tenere traccia del tempo trascorso dall’ultimo movimento si è fatto uso di tre variabili:

- Variabile “tempo” che inizia a scorrere quando il dispositivo è alimentato. Per tenere traccia del tempo si è utilizzato la funzione *millis()* che comunica il tempo in millisecondi;
- Variabile “t_u_mov” che sta per “tempo ultimo movimento”. In questa variabile viene registrato il tempo in cui il dispositivo ha rilevato un movimento;
- Variabile “t_t_mov” che sta per “tempo trascorso dall’ultimo movimento”. E’la differenza tra la variabile “tempo” e “t_u_mov” e dice quanti secondi sono trascorsi dall’ultima respirazione registrata.
- Variabile “Deadline” impostata a 20000 millisecondi come tempo limite che può trascorrere prima che scatti il sistema d’allarme.

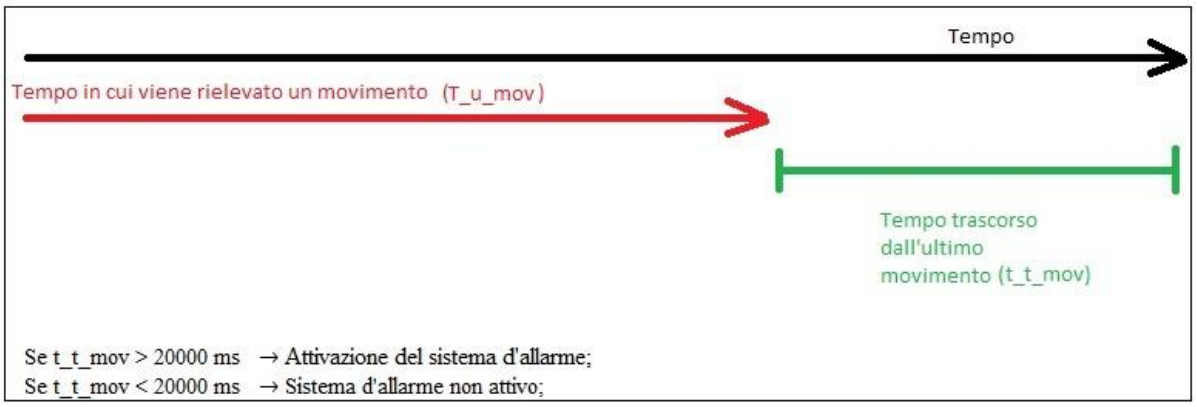

Fig.26 Schema rappresentativo del monitoraggio del tempo trascorso dall'ultimo movimento.

Nella figura successiva è riportato il relativo codice:

```
if ( G_x < -0.3 || G_x > 0.3) {
  t_u_mov = tempo;
}

if ( G_x > -0.03 && G_x < 0.03 ){
  Serial.print("Nessun movimento rilevato da ");Serial.print(t_t_mov/1000);Serial.println(" s ");
  t_t_mov = tempo - t_u_mov;
}

if ( t_t_mov < Deadline ) {
  digitalWrite(4,HIGH);
  digitalWrite(3,LOW);
}
if ( t_t_mov > Deadline ) {
  digitalWrite(3,HIGH);
  digitalWrite(4,LOW);
  tone(5,150,100);
}
```

Fig.27 La figura rappresenta i quattro blocchi *if* responsabili del monitoraggio della respirazione.

Nel primo riquadro rosso è presente l'*if* che effettua il primo controllo: se la velocità è considerevole, allora si aggiorna la variabile "t_u_mov" tenendo traccia del tempo. Di conseguenza, come si può vedere nel primo riquadro blu, il LED verde resta acceso comunicando che il dispositivo sta rilevando movimento; altrimenti, se la velocità si trova nell'intorno dello zero, la variabile "t_u_mov" non viene più aggiornata e quindi la

differenza tra “tempo” e “t_u_mov”, ovvero “t_t_mov”, diventa sempre più grande e una volta che diventa maggiore della variabile “*Deadline*”, si attiva il blocco *if* responsabile dell’attivazione dell’allarme, disattivando dapprima il LED verde, successivamente attivando il LED rosso, e infine facendo suonare il buzzer (secondo riquadro blu).

3.4.10 Misurazione dei tempi di respirazione e calcolo della frequenza respiratoria.

Per il calcolo del tempo di inspirazione e espirazione si è operato come segue:

- Appena inizia la fase inspiratoria, la variabile “timer_i” inizia a scorrere prendendo il valore corrente dalla variabile “tempo”;
- Analogamente si fa per la fase espiratoria, utilizzando la variabile “timer_e”;
- Successivamente si fa la differenza tra “tempo” e “timer_e” affinché la variabile “t_ins” (tempo inspirazione) possa partire da 0 millisecondi e fermarsi una volta che la fase inspiratoria è finita;
- Analogamente al punto precedente, si calcola “t_esp” ovvero il tempo di espirazione.

```
//------INIZIO MONITORAGGIO------
    tempo = millis ();

 if( G_x > 1 ){
timer_i = tempo;
t_ins = tempo - timer_e;
t_esp_real = t_esp;
Serial.print("Fase inspiratoria ");Serial.print(" ");Serial.println(t_ins);
 }
 if(G_x < -1 ){
timer_e = tempo;
t_esp = tempo - timer_i;
t_ins_real = t_ins;
Serial.print("Fase espiratoria ");Serial.print(" ");Serial.println(t_esp);
 }
```

Fig.28 Codice relativo al calcolo dei tempi di inspirazione/espirazione.

Si noti bene che le variabili “t_ins” e “t_esp” si azzerano ogni qualvolta inizia la corrispettiva fase respiratoria e che quindi non ha significato sommarle per ottenere il tempo di respiro, proprio perché queste variano nel tempo. Per calcolare il tempo effettivo di ciascun respiro, abbiamo bisogno di due variabili che registrino l’ultimo valore assunto da “t_ins” e “t_esp”, ovvero quanto effettivamente sono durate le due fasi respiratorie. Per registrare l’ultimo valore assunto dalle variabili sopracitate è sufficiente salvare il valore di queste in altre variabili ma ciò va fatto nella fase opposta. Le variabili in questione sono state dichiarate col nome “t_ins_real” e “t_esp_real”. Quindi, per esempio, se si vuole ottenere la durata della fase inspiratoria sarà sufficiente imporre “t_ins_real = t_ins”, ma ciò va inserito nel ciclo *if* che monitora la fase espiratoria, in modo da garantire che la variabile “t_ins” abbia smesso di scorrere. Analogamente si ottiene “t_esp_real”.

Ad ogni giro di loop, le variabili “t_ins_real” e “t_esp_real” vengono sommate e il valore salvato nella variabile “t_resp” che sta per tempo di respiro. Alla fine di ogni ciclo di loop dunque, avremo il tempo effettivo di ogni respiro.

Finalmente, una volta ottenuta la durata di un respiro per calcolare la frequenza respiratoria sarà sufficiente eseguire il seguente calcolo:

$$f_R = 60000/T_{Resp}$$

Che restituisce il numero di respiri al minuto.

```
t_resp = t_ins_real + t_esp_real;
frequenza_resp= 60000/(t_resp);

Serial.print("Frequenza Respiratoria ");Serial.print(frequenza_resp);Serial.print(" respiri/min ");Serial.print(" ");
Serial.print("Durata ultimo respiro ");Serial.print(t_resp);Serial.println(" s ");
```

Fig.29 Codice relativo al calcolo e alla stampa del tempo di respiro e della frequenza respiratoria.

CAPITOLO 4

Prove sperimentali.

4.1 Analisi del rumore tramite Matlab.

Quando si trattano i sensori il problema del rumore è abbastanza frequente. Questo fenomeno è dovuto ad alcune proprietà della materia ed essendo frutto di variabili aleatorie non è possibile eliminarlo alla radice, ma è possibile comunque ovviare a tale problema facendo uso di diverse tecniche.

4.1.1 Definizione di rumore elettronico.

Per rumore si intendono tutti quei segnali di origine aleatoria che si vanno a sovrapporre al segnale che si vuole studiare e che quindi vanno ad alterare, in maniera poco o molto significativa, il segnale stesso. Il rumore non va confuso con il disturbo che trova origine in cause estrinseche allo strumento di misura, quindi fisicamente eliminabile. L'origine del rumore invece è dovuta a eventi intrinsechi allo strumento di misura che si sta utilizzando. Per esempio, se una persona provasse a tracciare una linea retta su di un foglio senza l'ausilio di un righello, questa in ogni caso risulterebbe non perfettamente retta a causa di diversi fattori, come per esempio una mano non tenuta perfettamente ferma ecc. Si parla quindi di fattori non eliminabili. Lo stesso accade con i sensori, ed è riconoscibile come una fluttuazione casuale attorno al segnale utile. Matematicamente lo si può rappresentare come:

$$f(t) = s(t) + n(t)$$

Dove f(t) rappresenta la somma tra il segnale utile s(t) e il rumore n(t) che per l'appunto assume valori casuali nel tempo. Data la sua natura stocastica, il rumore potrà essere descritto solamente attraverso metodi statistici.

4.1.2 Varianza e deviazione standard.

In statistica la deviazione standard, o scarto quadratico medio, è un parametro che indica la variabilità di un insieme di dati. In altre parole indica quanto mediamente si discosta un elemento N, preso da un insieme

I, dalla media dei valori di questo insieme. Analiticamente parlando si calcola come:

$$\sigma_X = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}}$$

Dove:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

e indica la media aritmetica di degli elementi dell'insieme di dati.
Mentre la varianza non è altro che ${\sigma_X}^2$.
Questi valori, che verranno calcolati in seguito, serviranno per classificare il tipo di rumore di cui è afflitto l'MPU6050. Nella maggior parte dei casi, la distribuzione del rumore tende ad avere una forma approssimabile ad una curva gaussiana; mentre per rumore bianco si intende un rumore caratterizzato dall'assenza di periodicità nel tempo e da un'ampiezza costante su tutto lo spettro di frequenze.

4.1.3 Matlab e Simulink.

Matlab, che sta per MATrix LABoratory, è un programma molto versatile che fornisce svariati strumenti di calcolo, sviluppato da *MathWorks*.
L'interfaccia di Matlab si suddivide in:

- Command Window: è la parte dove scrivere le varie istruzioni. Il linguaggio di programmazione è un derivato del C.
- *Current Folder*: permette di importare script e progetti da un file esterno oppure per indicare la cartella di destinazione di un file da salvare.
- *Workspace*: sezione che indica tutte le variabili attualmente salvate in memoria.

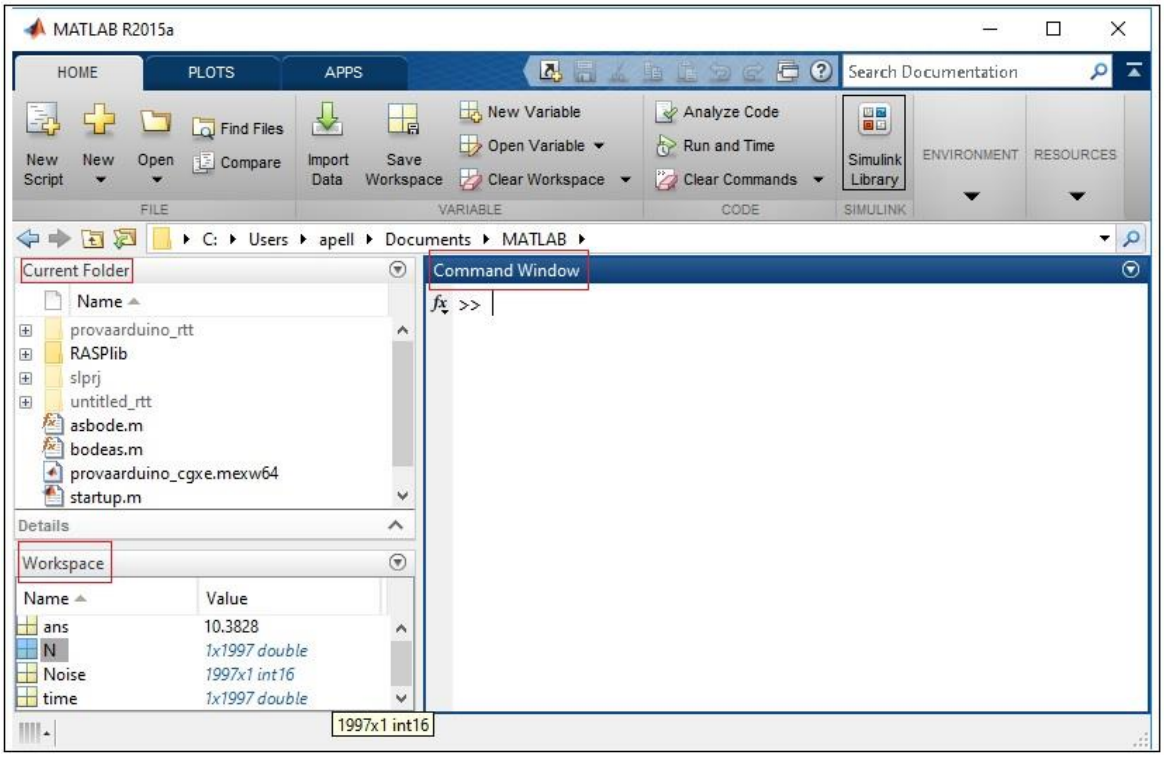

Fig.1 Interfaccia Matlab.

Simulink, che è un'estensione di Matlab, fornisce gli strumenti per permettere la simulazione di qualunque sistema si intenda implementare facendo uso di schemi a blocchi. Esempio:

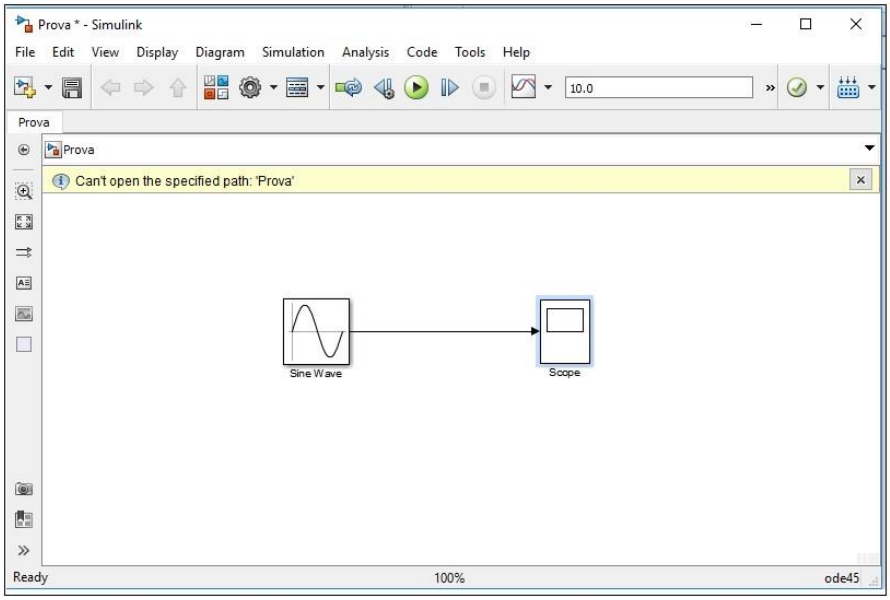

Fig.2 Interfaccia Simulink.

Questi software saranno utili per lo studio del segnale che fornisce l'MPU6050.

4.1.4 Librerie di supporto.

Per rendere possibile l'interfacciamento tra Matlab è Arduino, si è fatto uso della libreria *Simulink Support Package for Arduino Hardware*, un'estensione fornita direttamente dalla *MathWorks*. Questa libreria permette di far interagire Matlab e Arduino: si può usare il primo come generatore di segnali e visualizzare l'output, per esempio su un LED, oppure può fungere da oscilloscopio, come si è fatto nel corso di questo progetto.

Infine, permette di effettuare simulazioni in tempo reale. Questa opzione è disponibile solo per le versioni più potenti di Arduino, come per esempio l'Arduino Mega. Questo spiega perché ai fini progettuali si è fatto uso di Mega e non di Uno.

Un'altra utile libreria di cui si è fatto utilizzo è "Rensellear Arduino Support Package" che implementa come blocchi utili per la simulazione diversi sensori, tra cui l'MPU6050.

4.1.5 Real-Time Simulation.

Dopo aver installato correttamente le librerie utili precedentemente descritte, si può avviare la simulazione in tempo reale. Questo tipo di simulazione permette di visualizzare grazie ad un oscilloscopio l'andamento degli output forniti dall'MPU6050.

Grazie a ciò si potrà analizzare molto più dettagliatamente il rumore di cui è affetto il sensore.

4.1.6 Costruzione schema a blocchi. Conversione. Compensazione offset.

Per prima cosa, si costruisce lo schema a blocchi (Fig.3). Esso sarà composto da:

- Il blocco che rappresenta l'accelerometro o il giroscopio dell'MPU6050, costituito da tre output: uno per ogni asse;
- Un *Multiplex* per raggruppare i tre output dato che l'oscilloscopio può avere un solo input;
- Uno *Scope*, o oscilloscopio, che servirà per visualizzare i dati su un diagramma temporale.

Successivamente, come mostrato in fig.4, bisognerà impostare i parametri della simulazione. In fig.4a sono mostrate le impostazioni della voce "*Run on Target Hardware*":

- *Target hardware*: serve ad indicare quale microcontrollore si sta utilizzando.
- *Set host COM port*: serve per specificare quale porta USB si intende utilizzare;
- *Serial Baud*: serve per impostare la velocità di comunicazione seriale.

In fig.4b sono riportate le impostazioni del *Solver*:

- *Start-time e Stop-time*: qui si imposteranno il tempo di inizio e il tempo di fine simulazione;
- *Fixed-Step-Size*: serve per indicare il "passo", ovvero quanto disterà a livello temporale un punto del grafico dall'altro.

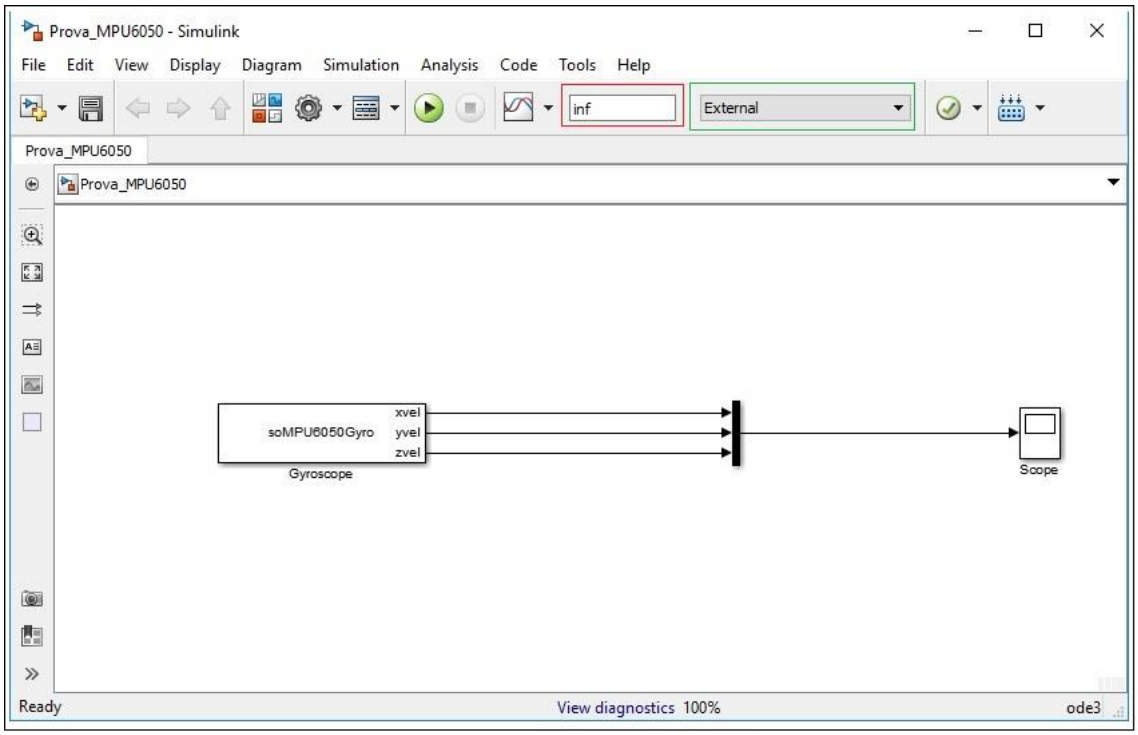

Fig.3 Prova giroscopio in tempo reale

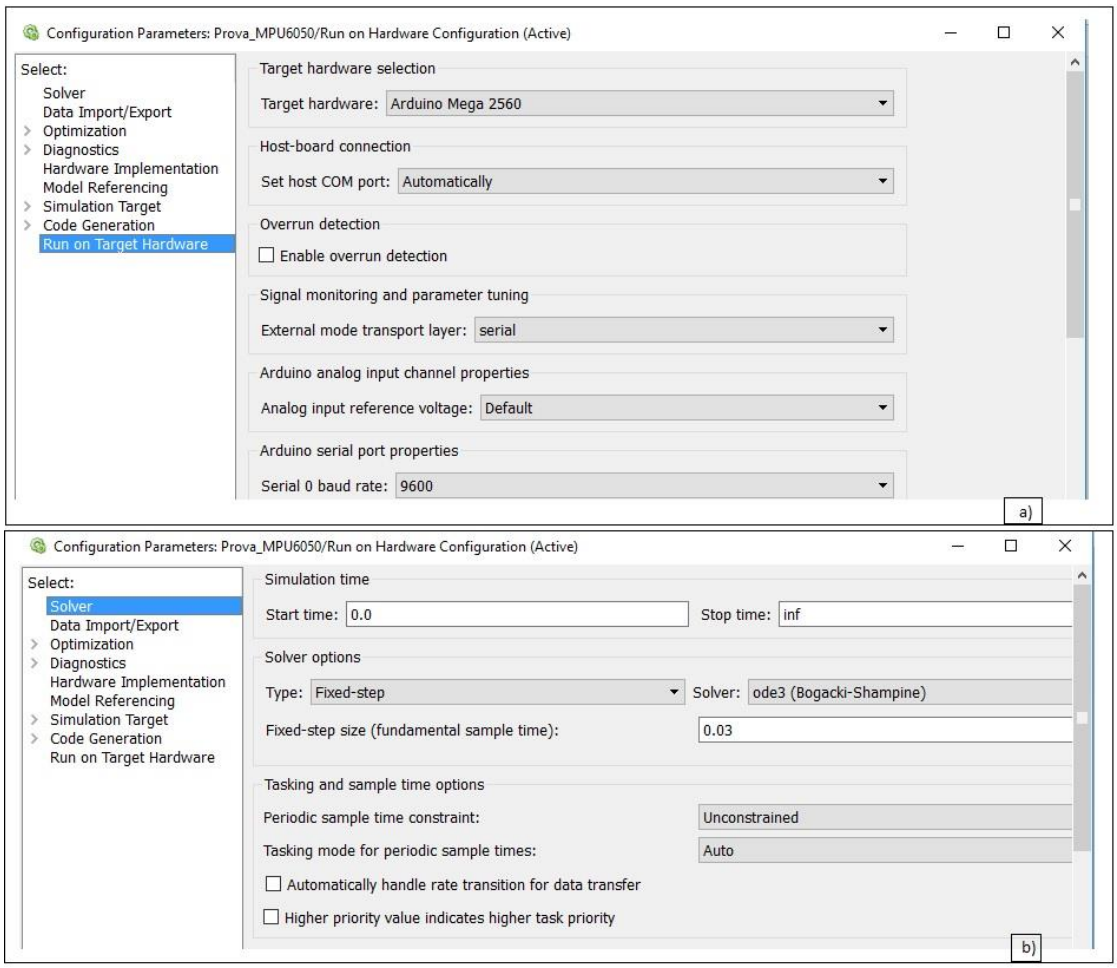

Fig.4: a) Impostazioni “Run on Target Hardware”; b) Impostazioni Solver.

Infine, come riportato in alto a destra in fig.3 nei riguardi verde e rosso, si è scelto un tempo di simulazione senza limiti di tempo (inf) e si è indicato che un dispositivo esterno fornirà i dati da elaborare (external).

In fig.5 e 6 sono riportate le uscite dell'accelerometro e del giroscopio sull'oscilloscopio tenendo il sensore fermo.

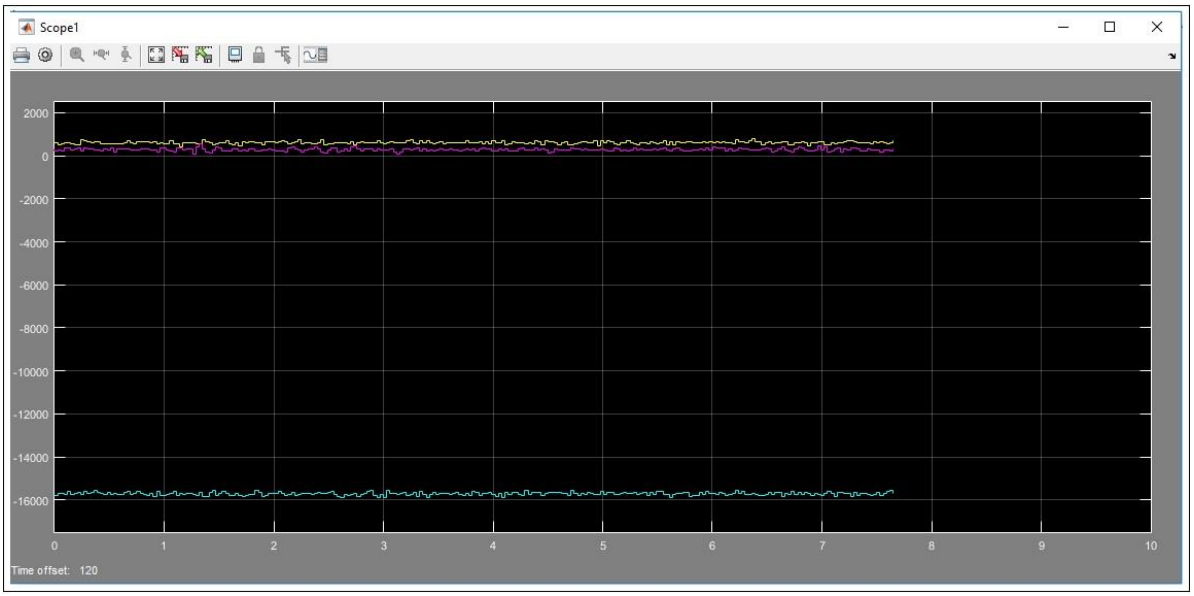

Fig.5: le linee di color giallo, viola e azzurro rappresentano rispettivamente gli assi x,y,z.

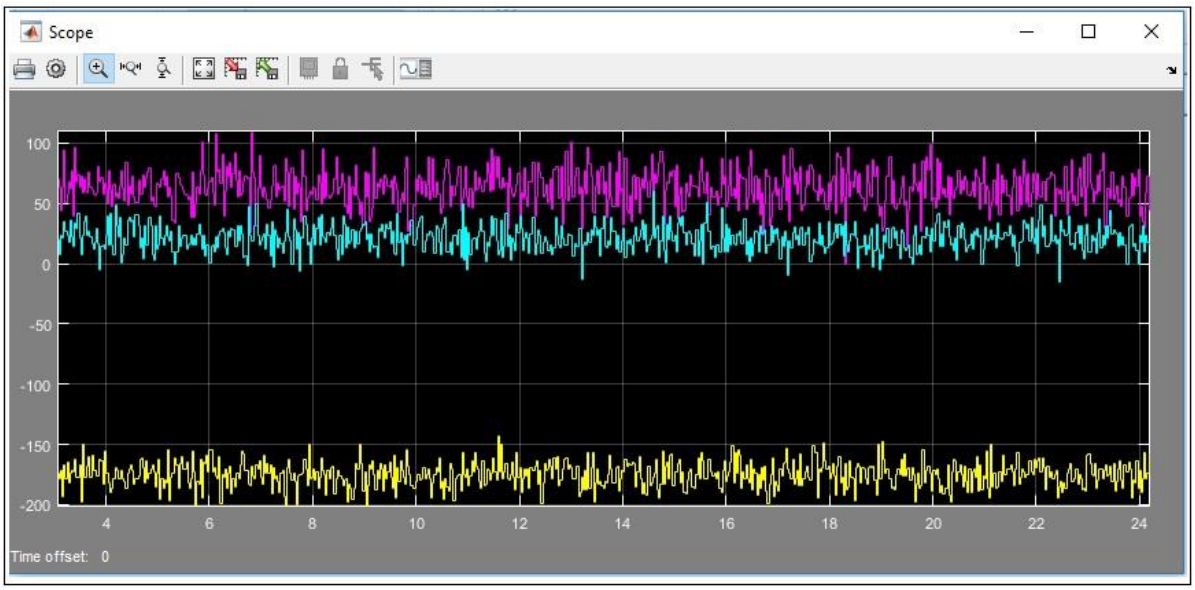

Fig.6: output assi del giroscopio.

Già prendendo visione della fig.6 è possibile vedere in modo più marcato gli effetti del rumore, inoltre sono presenti degli offset che

successivamente verranno rimossi. Si noti bene che i valori raffigurati in precedenza non sono ancora stati convertiti. Per fare ciò, come già spiegato nel paragrafo 3.3, bisogna semplicemente dividere i valori dell'accelerometro per 16.384 e quelli del giroscopio per 131,072. Per fare ciò si dovrà semplicemente aggiungere un blocco "Gain", ovvero di guadagno.

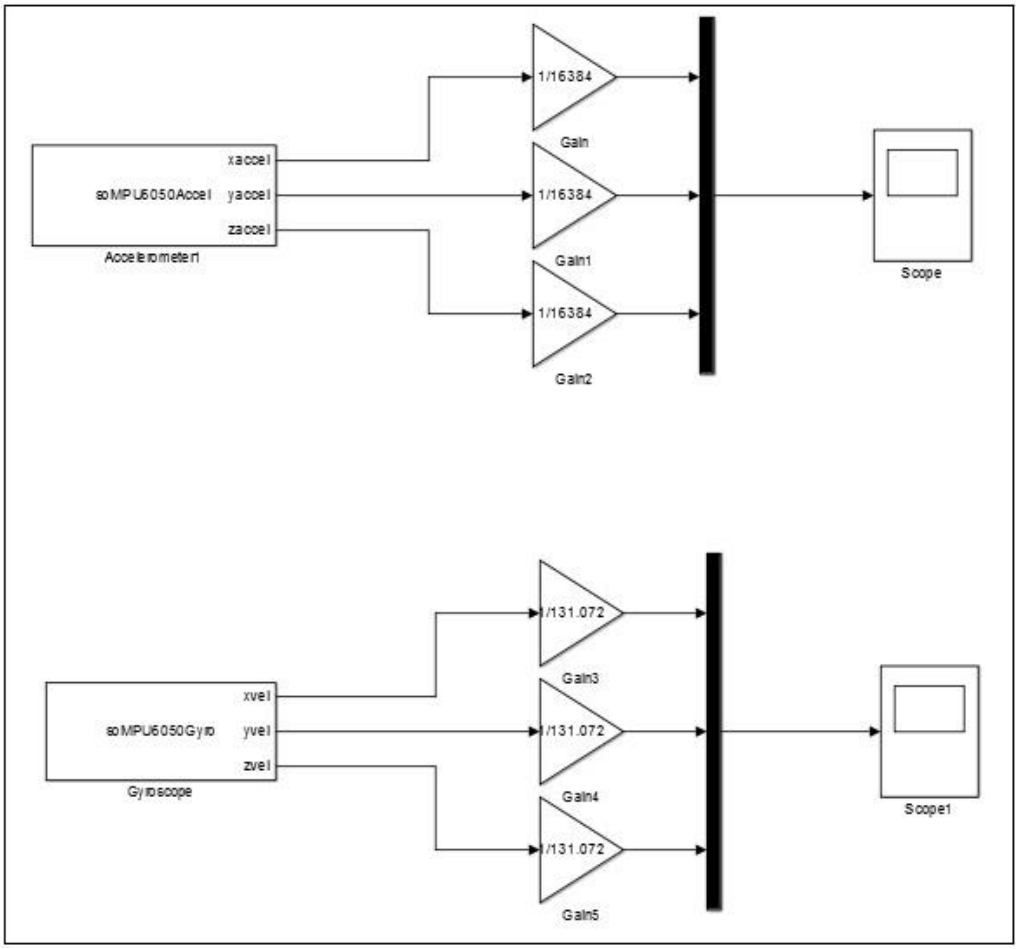

Fig.7 Schema a blocchi per avere i dati convertiti nelle rispettive unità di misura.

4.1.7 Calcolo media, varianza e deviazione standard del rumore

Utilizzando il blocco "To Workspace" è possibile estrapolare i valori dei singoli assi del sensore e salvarli in una variabile di tipo array direttamente nel Workspace di Matlab per eseguire i calcoli desiderati.

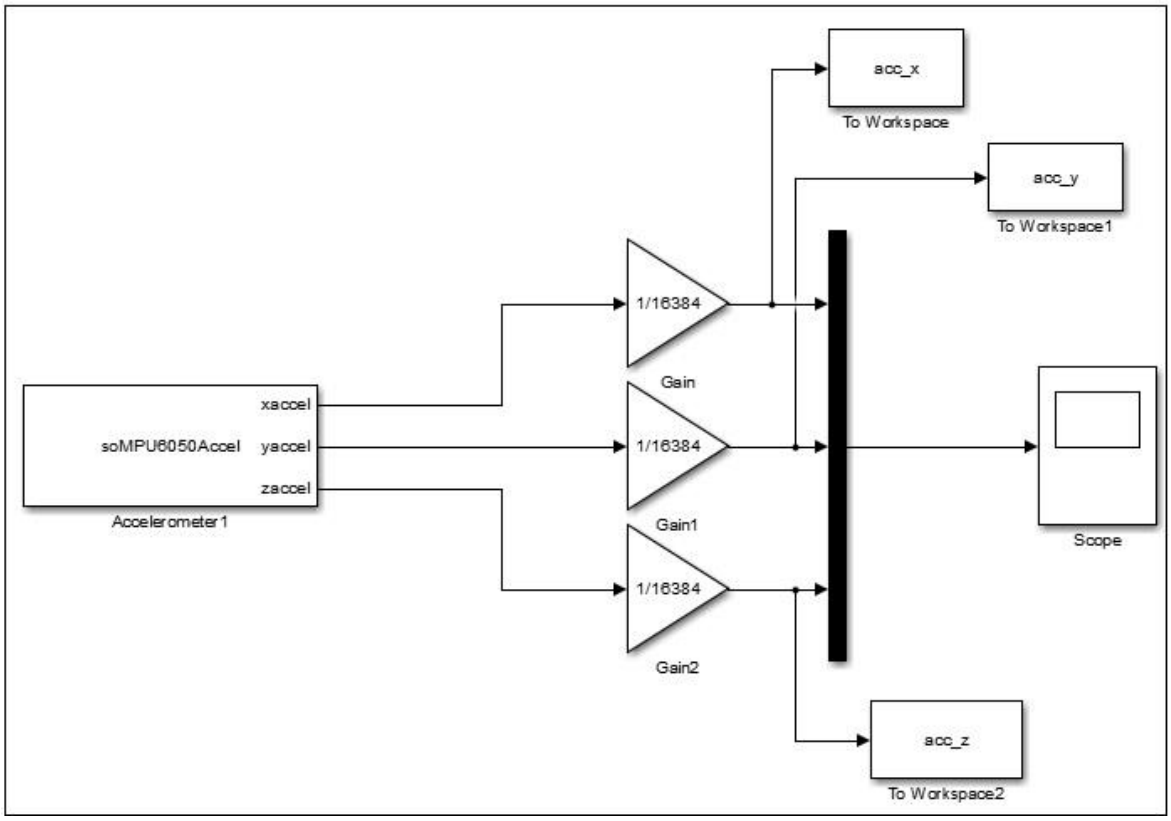

Fig.8 Salvataggio valori accelerometro nel Workspace.

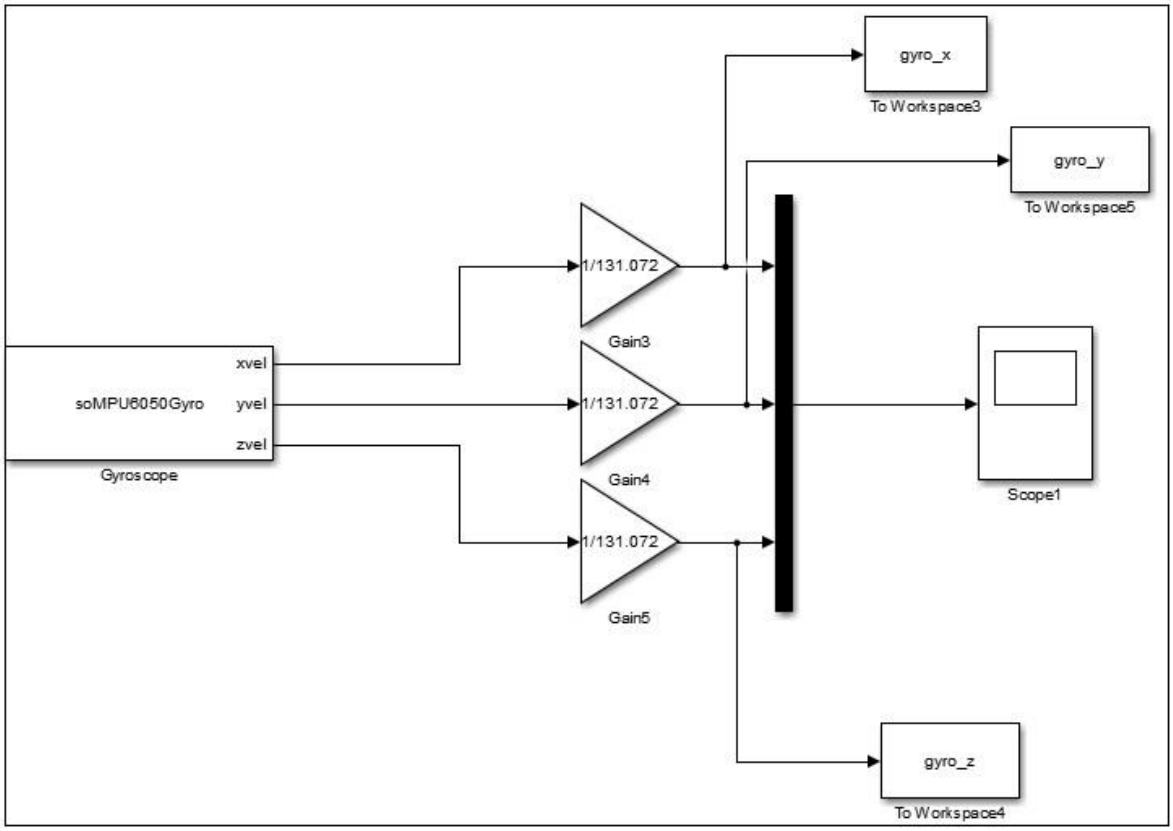

Fig.9 Salvataggio valori giroscopio nel Workspace.

Successivamente si avvia la simulazione e i valori verranno direttamente salvati nelle variabili temporali.

Una volta fatto ciò, si calcoleranno i seguenti parametri:

- Medi aritmetica, utilizzando il comando *mean()*;
- Varianza, utilizzando il comando *var()*;
- Deviazione standard, utilizzando il comando *std()*;

Questi andranno calcolati per i dati relativi ad ogni asse, sia dell'accelerometro che del giroscopio, salvandoli in un apposito array:

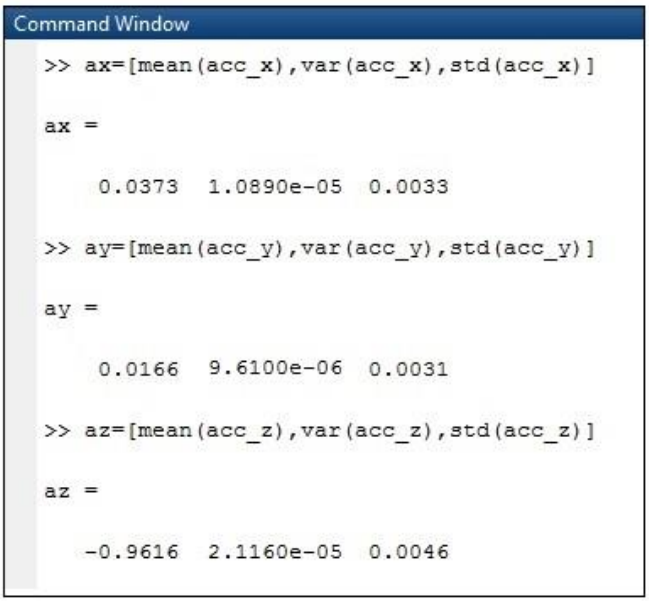

Fig.10 Calcolo dei parametri del rumore in Matlab, relativi all'accelerometro.

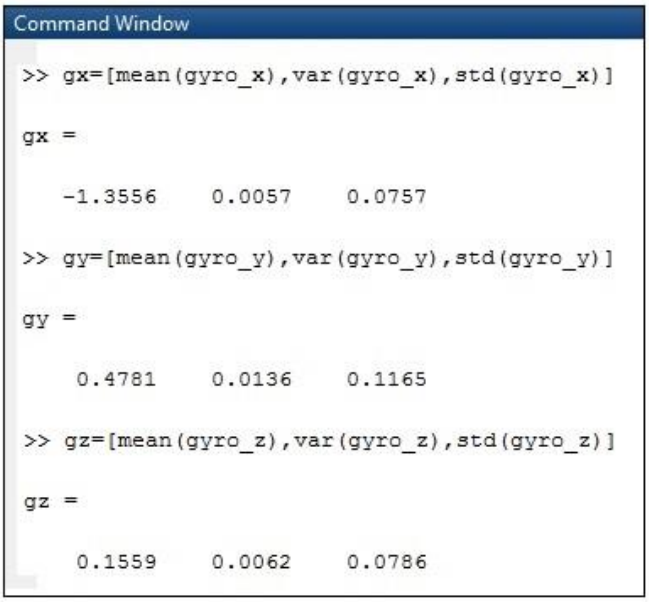

Fig.11 Calcolo dei parametri del rumore in Matlab, relativi al giroscopio.

I parametri appena calcolati rappresentano rispettivamente:

- l'offset, dovuto per la maggior parte ad artefatti di corretto posizionamento del sensore (media aritmetica);
- la media dei quadrati degli scarti dei singoli valori dalla media aritmetica (varianza);
- la variabilità del rumore stesso (deviazione standard).

Una volta calcolati gli offset, si può effettuare una semplice compensazione per calibrare correttamente il sensore:

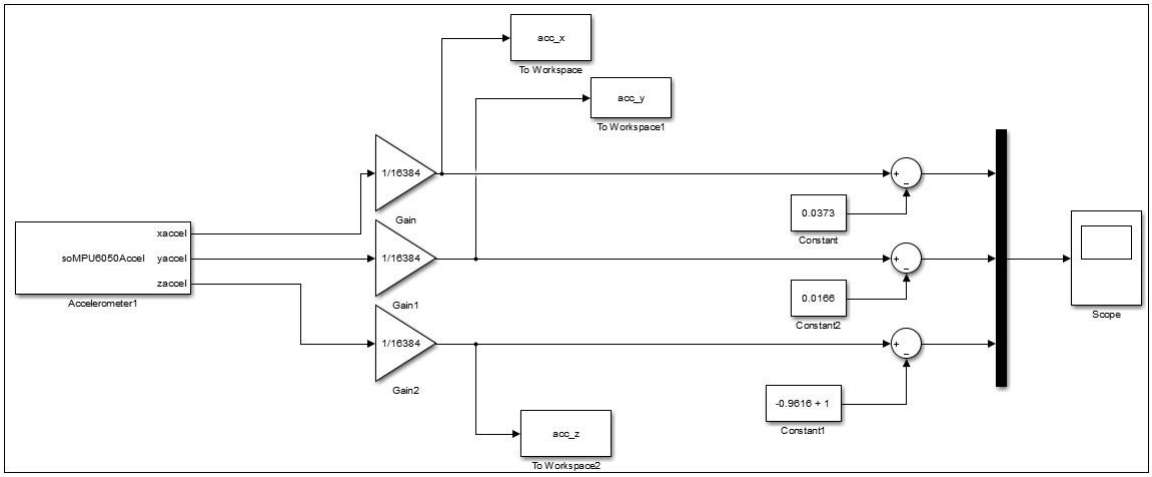

Fig.12 Schema a blocchi relativo all'accelerometro comprendente la compensazione degli offset.

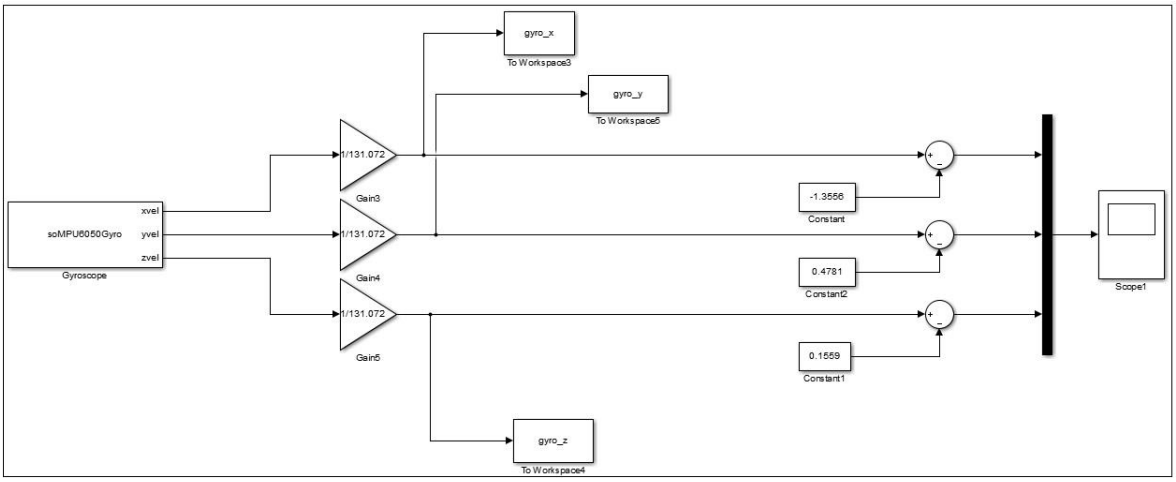

Fig.13 Schema a blocchi relativo al giroscopio comprendente la compensazione degli offset.

Si noti che in figura 13 per quanto riguarda la compensazione dell'asse z, si è dapprima compensato l'offset e successivamente si è aggiunto 1. Si è fatto ciò poiché perché l'asse z misura proprio l'accelerazione di gravità e quindi l'asse z, a sensore fermo, misurerà 1g.

A seguito della compensazione, il risultato sarà il seguente:

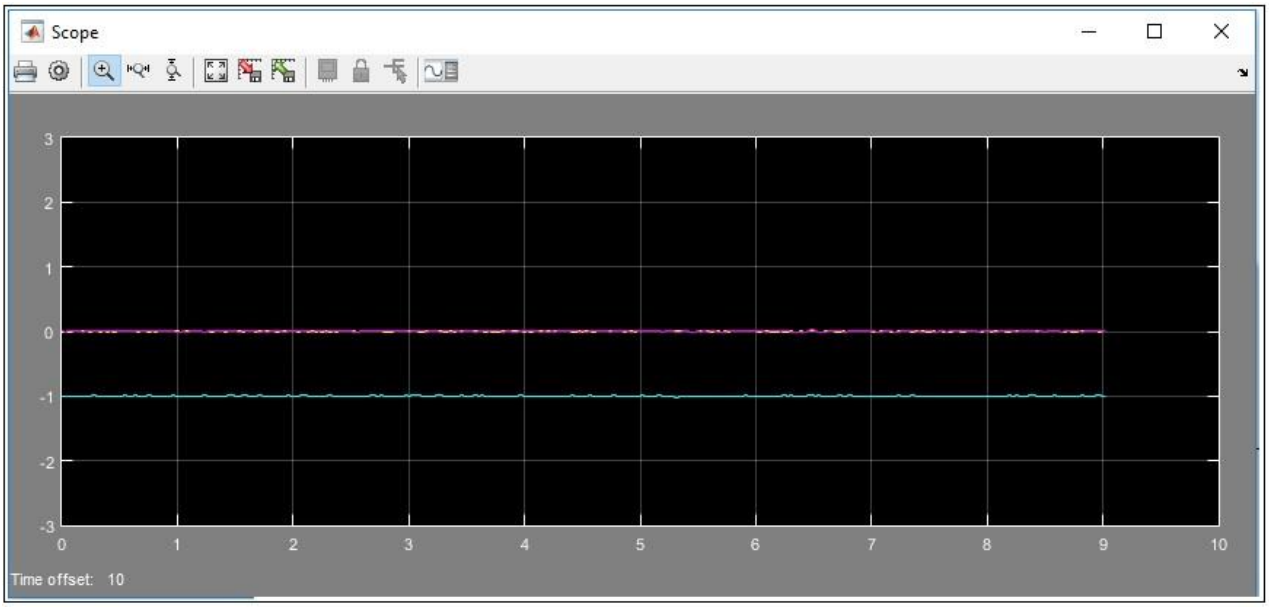

Fig.14 Output accelerometro su oscilloscopio con compensazione.

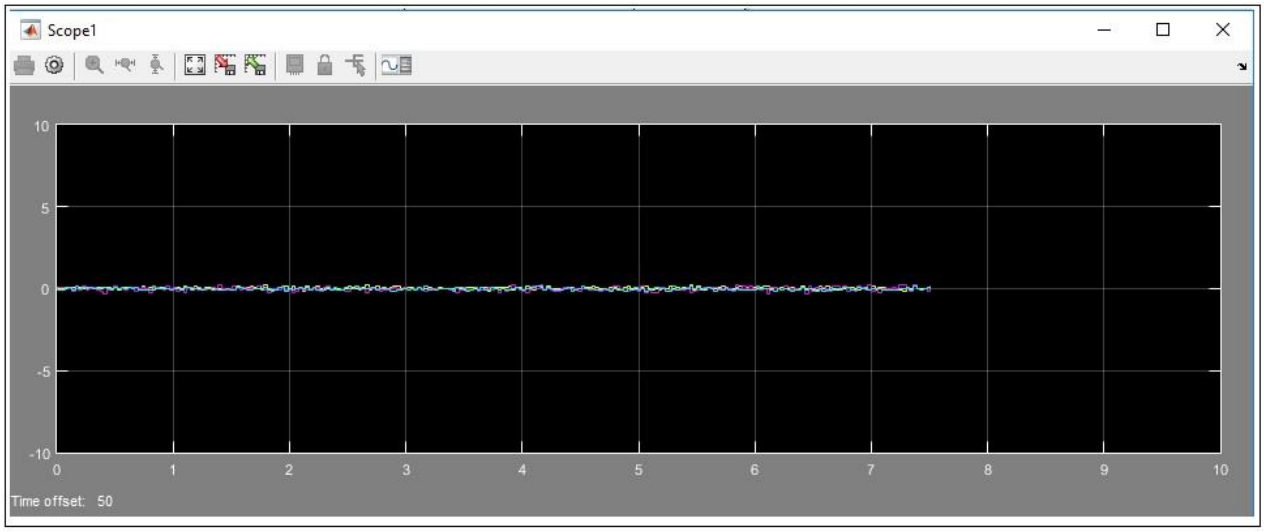

Fig.15 Output giroscopio su oscilloscopio con compensazione.

In figura 14 si può notare che l'asse z misura -1g e non 1g come ci si aspetterebbe. Questo è dovuto al fatto che l'asse z del sensore è stato posizionato in modo opposto rispetto all'asse z che misura l'MPU6050.

4.1.8 Misura dell'angolo di inclinazione. Considerazioni finali sull'MPU6050.

Una volta effettuata la calibrazione è possibile misurare, oltre la velocità angolare e l'accelerazione, anche l'inclinazione rispetto agli assi. Per fare ciò, sarà sufficiente semplicemente integrare la velocità angolare rispetto al tempo utilizzando l'apposito blocco che fornisce la libreria Simulink:

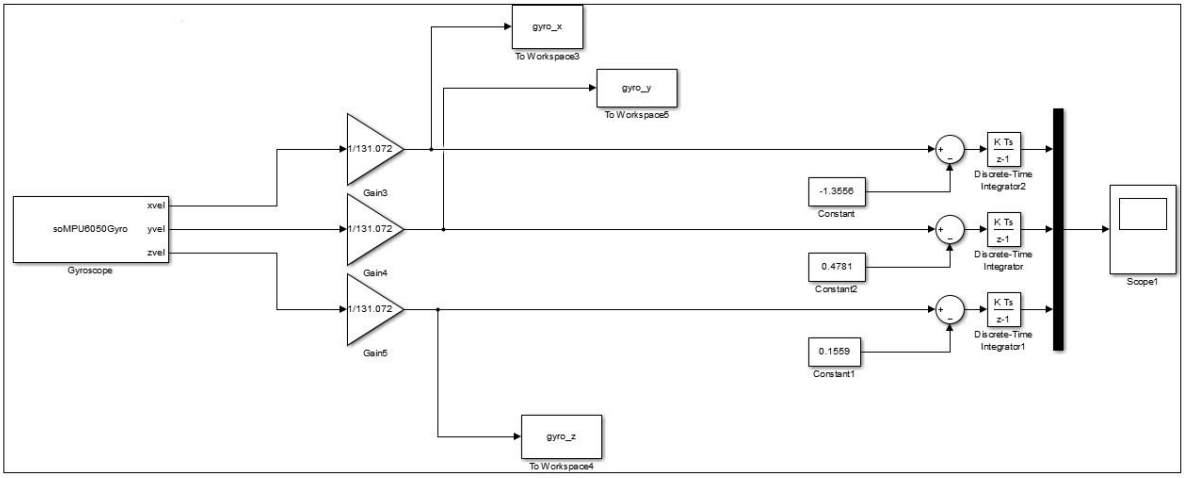

Fig.16 Schema a blocchi per misurare l'angolo di inclinazione di ogni singolo asse.

I blocchi "Discrete-Time Integrator" integrano i valori in input del giroscopio rispetto al tempo fornendo così l'angolo misurato. Di seguito sono riportate figure esemplificative:

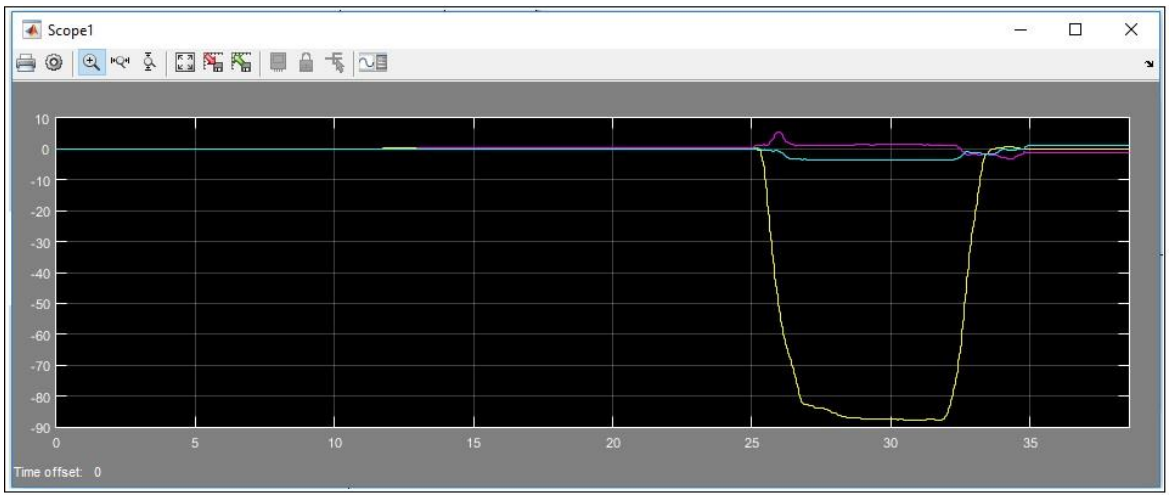

Fig.17 Output (linea gialla) ottenuto inclinando il sensore rispetto all'asse x di -90°.

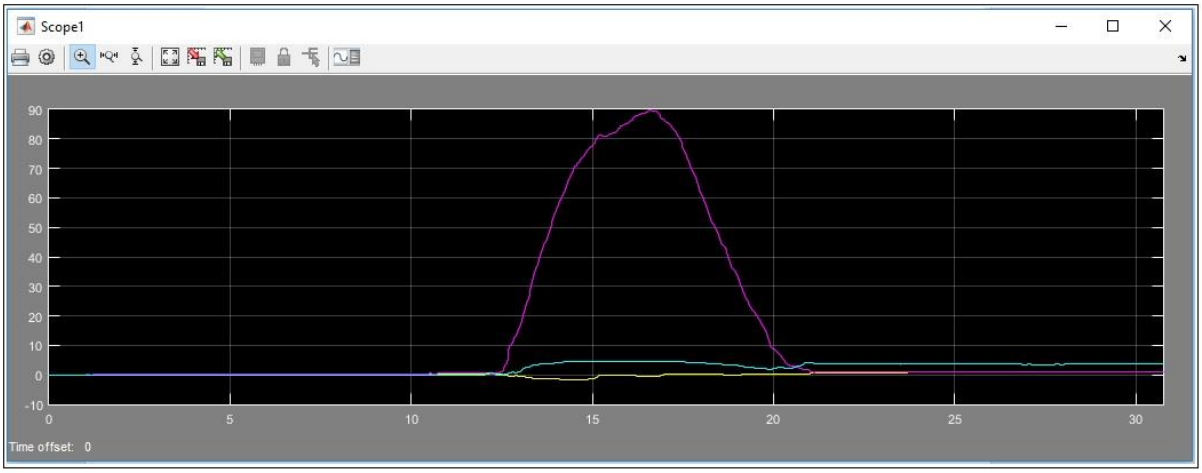

Fig.18 Output (linea viola) ottenuto inclinando il sensore rispetto all'asse y di +90°.

Dopo che il sensore è stato mosso più volte, se riportato alla posizione iniziale si può notare che i valori non ritornano esattamente a zero:

Fig.19 Offset dovuto all'accumulo dell'errore nell'integrale.

Come si può notare, man mano che si usa, il sensore accumula una certa quantità di offset nel tempo. Questo è dovuto all'integrale che si aggiunto per calcolare l'inclinazione, che accumula una certa quantità di errore

(chiamato anche *drift*, ovvero deriva). Questo è uno dei difetti presenti se si decide di calcolare la posizione angolare utilizzando questa tecnica.
Per quanto riguarda l'accelerometro, altri fattori che sono emersi durante le prove sono:

- al contrario del giroscopio soffre di meno il problema del rumore e le sue misure sono leggermente più accurate;
- è sensibile alle piccole vibrazioni.

4.2 Prove sperimentali.

4.2.1 Premesse.

Si è arrivati dunque alle prove volte a testare l'efficacia e il funzionamento del dispositivo. Le seguenti prove sono state effettuate su due soggetti maschi di diversa corporatura. Durante le simulazioni, i soggetti sono stati fatti sistemare su di un letto in posizione supina con le braccia distese lungo i fianchi.
Le prove sono state effettuate in modo tale da testare una ad una le diverse funzionalità del dispositivo: discriminazione delle fasi della respirazione; efficacia del sistema d'allarme; calcolo dei tempi di inspirazione, espirazione, respiro completo; calcolo frequenza respiratoria.

4.2.2 Soggetto 1.

Per prima cosa si è fatta una prova preliminare per verificare la giusta calibrazione del sensore. Subito ci si è accorti che i valori scelti per il controllo sulla variabile "G_x" non erano idonei per garantire un funzionamento adeguato del dispositivo. Di conseguenza si sono verificati i valori assunti dalla variabile "G_x" durante una normale respirazione e successivamente si fatte più prove fino a che non fosse garantito il corretto funzionamento del dispositivo.

```
if( G_x > 0.6 ){
timer_i = tempo;
t_ins = tempo - timer_e;
t_esp_real = t_esp;
Serial.print("Fase inspiratoria ");Serial.print(" ");Serial.println(t_ins);
}
if(G_x < -0.6 ){
timer_e = tempo;
t_esp = tempo - timer_i;
t_ins_real = t_ins;
Serial.print("Fase espiratoria ");Serial.print(" ");Serial.println(t_esp);
}

if ( G_x < -0.6 || G_x > 0.6) {
 t_u_mov = tempo;
}

if ( G_x > -0.55 && G_x < 0.55 ){
  t_t_mov = tempo - t_u_mov;
}

if(t_t_mov > 5000){
  Serial.print("Nessun movimento rilevato da ");Serial.print(t_t_mov/1000);Serial.println(" s ");
}
```

Fig.20 Calibrazione a seguito delle prime prove.

Fatto ciò si è passati alla prova vera e propria. Il risultato è stato il seguente:

Fig.21 Monitor seriale che riporta la fase inspiratoria.

Fig.22 Monitor seriale che riporta la fase espiratoria.

Come si può appurare dalle precedenti figure, il dispositivo riesce a discriminare se si sta trattando di fase inspiratoria o espiratoria, riportando anche le rispettive durate.

Successivamente si è chiesto al soggetto 1 di trattenere il respiro il più a lungo possibile, con lo scopo di testare il sistema di allarme del dispositivo. Anche questa prova è stata soddisfacente:

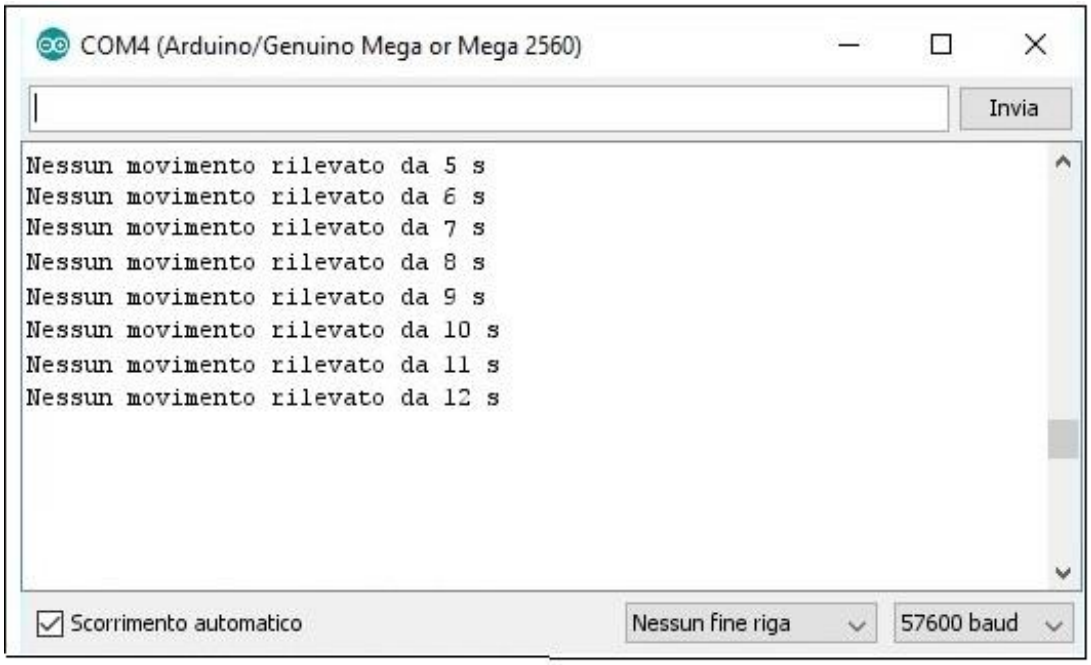

Fig.23 Risultati su monitor seriale con sistema di allarme attivo.

Una volta superati i sette secondi (si è scelto questo valore limite per comodità pratica), i sistemi d’allarme visivo e acustico si sono attivati comunicando che nessun respiro è stato rilevato dal dispositivo.

Infine si è testato se il dispositivo riuscisse a calcolare con buona approssimazione la frequenza respiratoria:

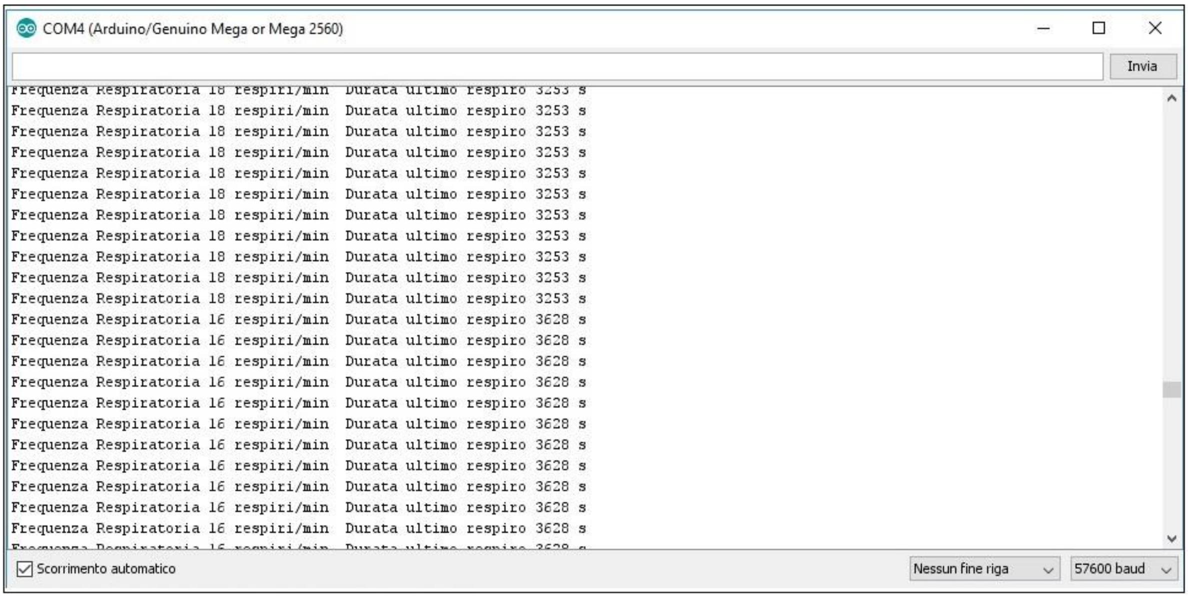

Fig.24 Risultati frequenza respiratoria in condizioni normali.

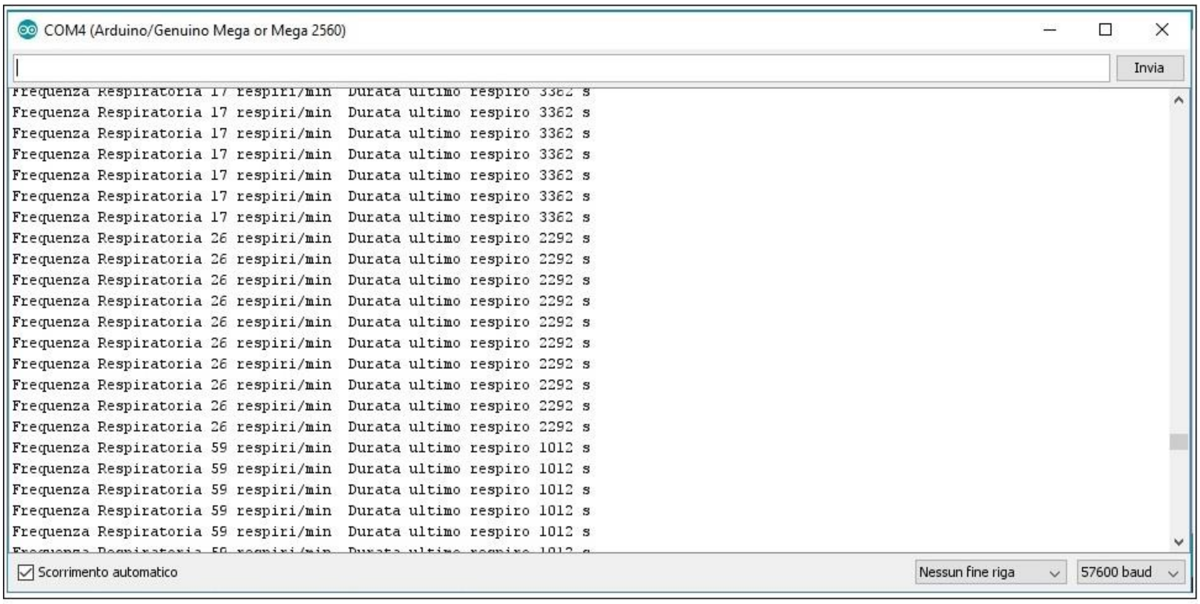

Fig.25 Risultati frequenza respiratoria con progressivo aumento della velocità di respirazione.

Come ultima prova, grazie all'ausilio di Matlab e della simulazione in tempo reale, si è visualizzato l'angolo di inclinazione durante il respiro:

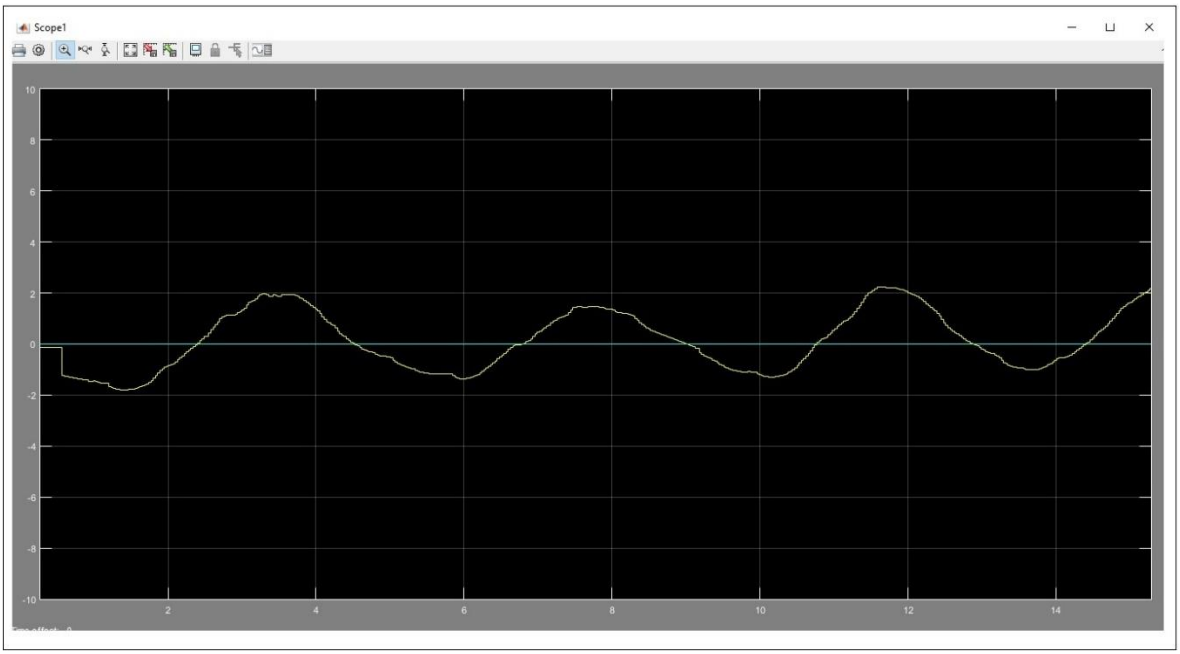

Fig.26 Inclinazione del dispositivo riportata sull'oscilloscopio di simulink in condizioni di respiro normali.

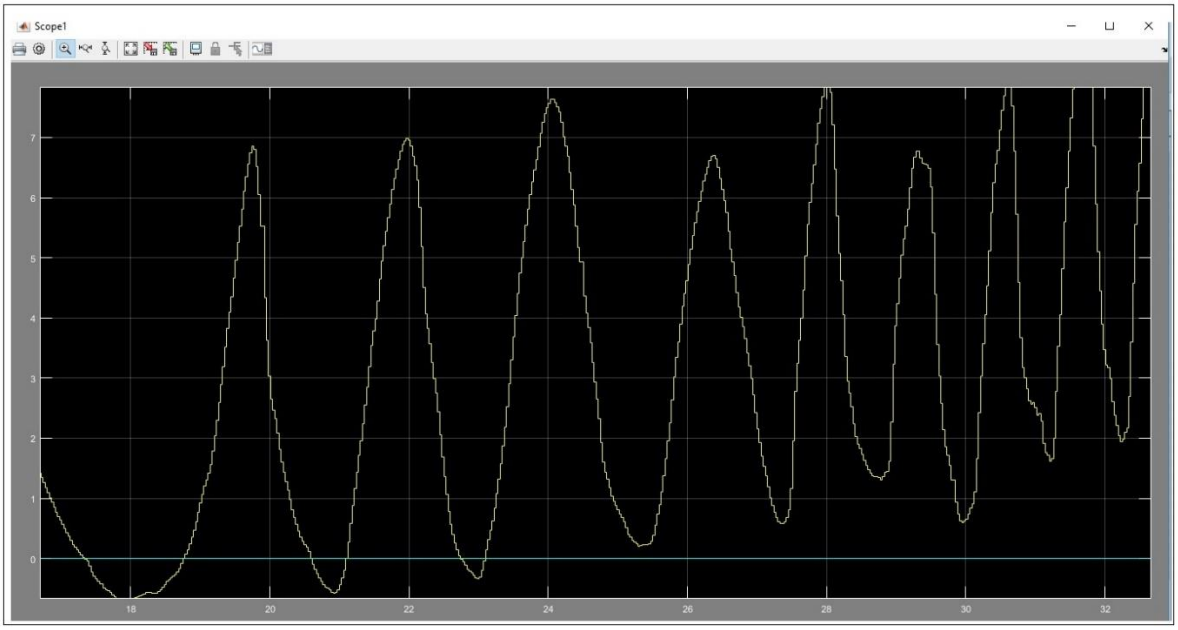

Fig.27 Inclinazione del dispositivo in condizioni di respiro profondo.

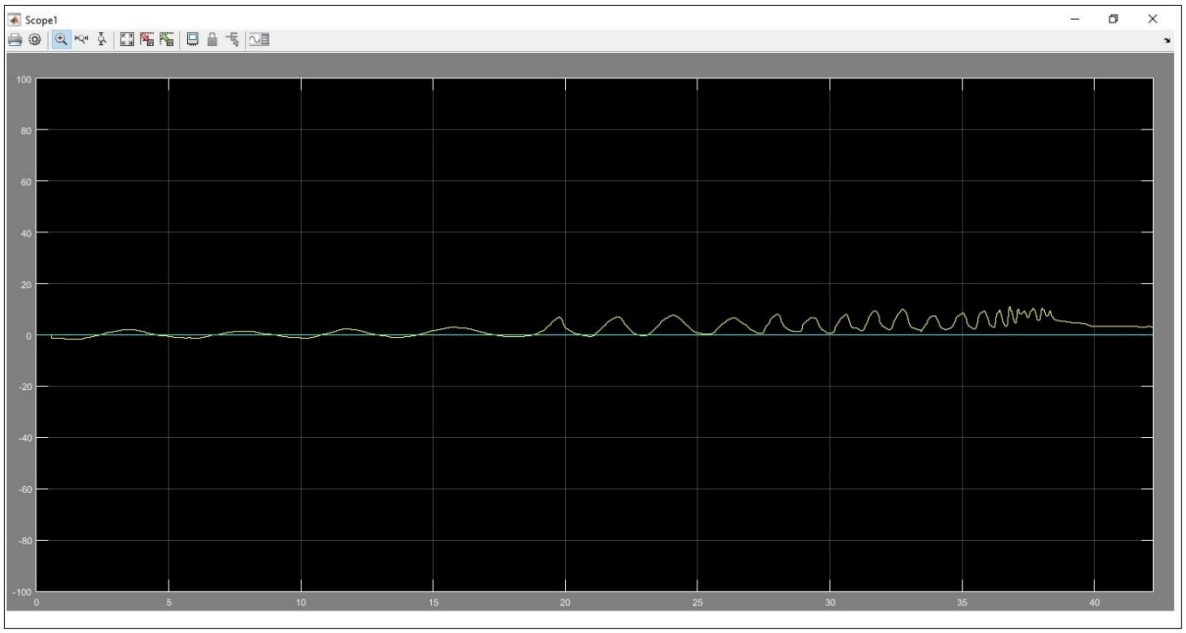

Fig.28 Simulazione completa.

4.2.3 Soggetto 2.

Per il soggetto 2 sono state effettuate lo stesso tipo di prove nelle stesse condizioni del soggetto 1. Anche in questo caso le prove sono state soddisfacenti. Di seguito sono riportati i risultati ottenuti:

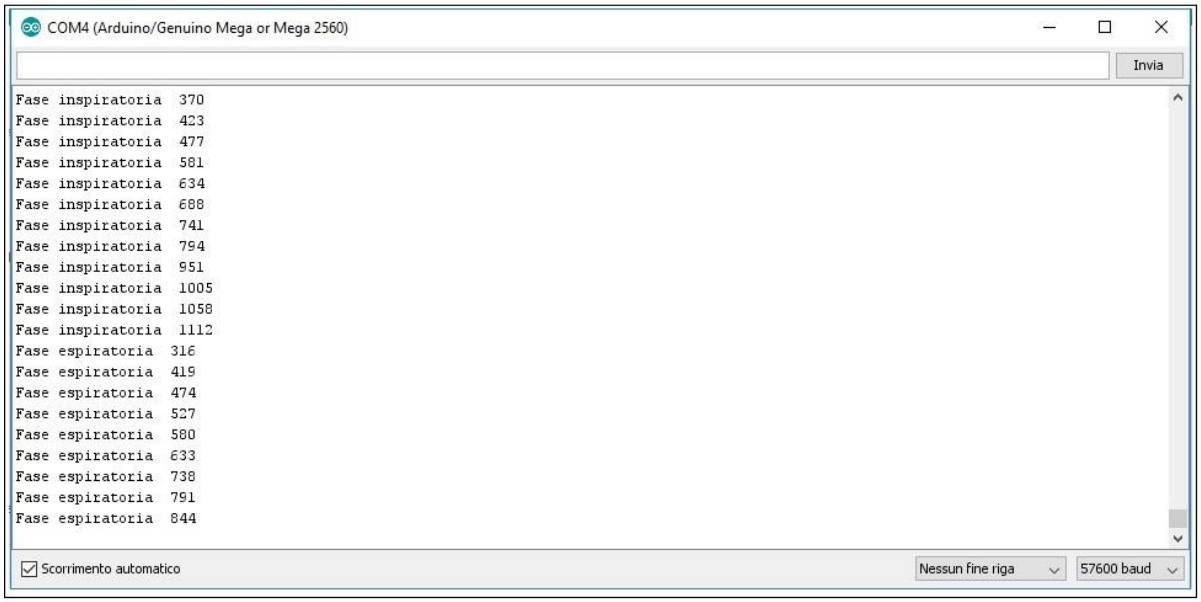

Fig.29 Risultati soggetto 2. Fase di inspirazione/espirazione.

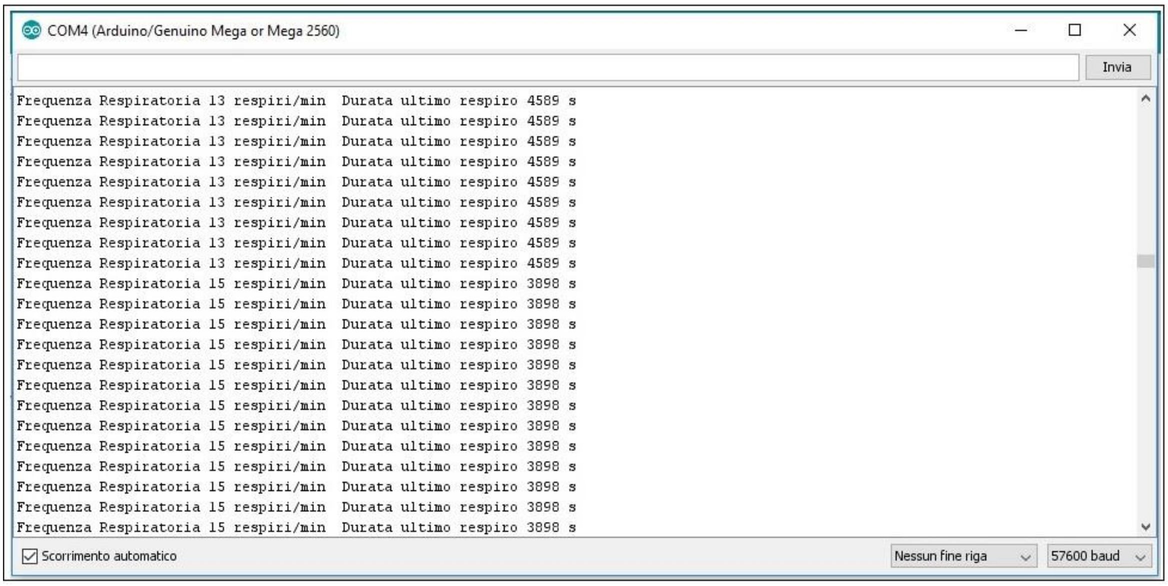

Fig.30 Risultati soggetto 2. Frequenza respiratoria in condizioni normali.

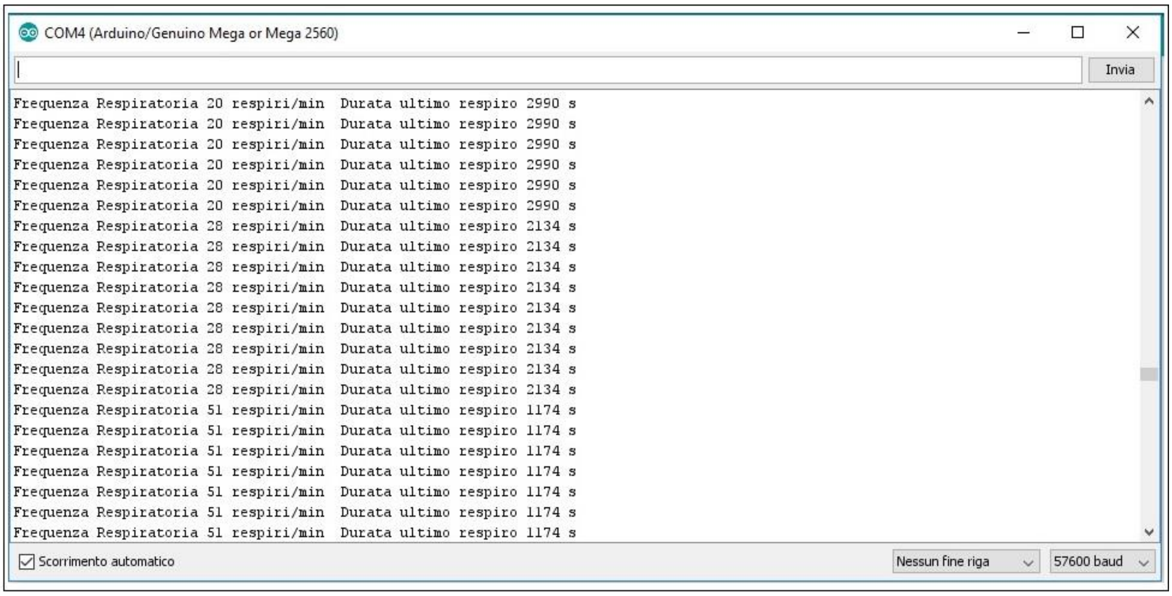

Fig.31 Frequenza respiratoria relativa ad un respiro affannoso.

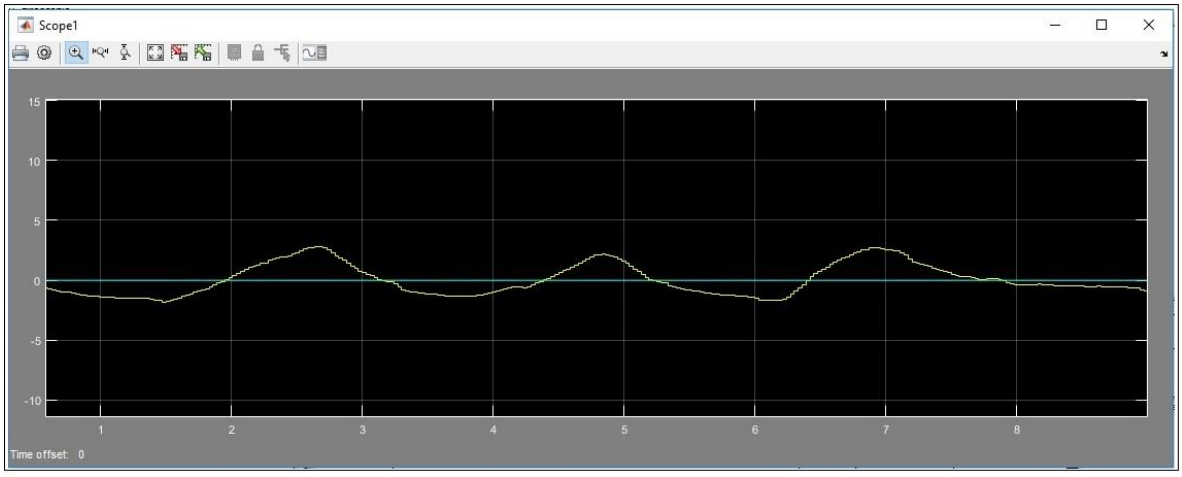

Fig.32 Risultati soggetto 2. Grafico che riporta l'inclinazione del dispositivo dovuta ad una respirazione normale.

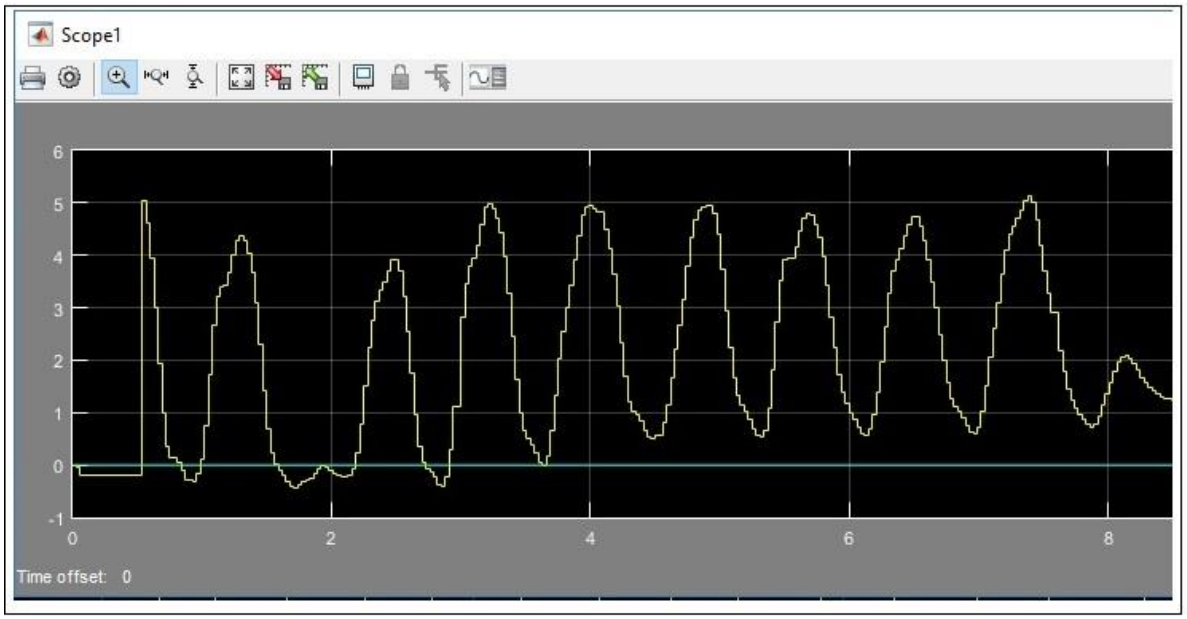

Fig.33 Risultati soggetto 2. Grafico che riporta l'inclinazione del dispositivo dovuta ad una respirazione affannosa.

CONCLUSIONI

Il lavoro presentato in questa tesi, soprattutto il capitolo relativo alle prove sperimentali, hanno avuto come principale obiettivo quello di provare a dimostrare che è possibile produrre un'alternativa non troppo costosa per il monitoraggio del respiro dei neonati. Non solo per quest'ultimi ma anche per persone di tutte le età, come hanno dimostrato le prove sperimentali.

Il progetto contenuto in questo lavoro ovviamente si presenta come un prototipo che ha grandi margini di miglioramento. Per esempio si potrebbe progettare un'applicazione smartphone che sia collegata direttamente al dispositivo e quindi consenta ai genitori di vedere in qualsiasi momento quali sono i parametri vitali del bambino, oppure implementare ulteriori controlli, come per esempio dare allarme in caso la frequenza respiratoria dovesse essere troppo bassa oppure in caso di respirazione affannosa.

Inoltre, la possibilità di ottenere diversi parametri vitali legati sempre alla respirazione apre la strada anche ad usi ospedalieri e quindi dare ai medici uno strumento in più per il controllo dei pazienti.

Alla fine di questo progetto e soprattutto alla luce dei risultati ottenuti nelle prove sperimentali, si può concludere che il sistema utilizzato per monitorare la respirazione può rappresentare una soluzione economica ma efficace a problemi di diversa natura.

BIBLIOGRAFIA

Schmidt M., Il manuale di Arduino, **seconda edizione**, **s.l**., **Apogeo editore**, **2015;**

Aliverti P., Il manuale di Arduino: Guida Completa, **prima edizione, s.l, Edizioni LSWR, 2016;**

G. Magnani, G. Ferretti, P. Rocco, Tecnologie dei sistemi di controllo, **McGraw-Hill**.