

Università Politecnica delle Marche

Corso di Laurea in Ingegneria Informatica e  
dell'Automazione

Relazione per l'esame di Laboratorio di  
Automazione

*Docente: Prof. Bonci Andrea*

**SENSORE DI DISTANZA  
VL53L1X**

Del Rossi Lorenzo mat. 1080698  
Giannini Maria Cristina mat. 1077586  
Menotta Claudio mat. 1076921  
Saccuti Alessio mat. 1077339

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Il sensore VL53L1X</b>	<b>3</b>
2.1	Caratteristiche tecniche . . . . .	4
2.2	Possibili applicazioni . . . . .	4
2.3	Funzionamento . . . . .	5
<b>3</b>	<b>Sviluppo del progetto</b>	<b>7</b>
3.1	Porting C++/C . . . . .	7
3.2	Interfaccia di comunicazione I <sup>2</sup> C . . . . .	8
3.2.1	Protocollo I <sup>2</sup> C . . . . .	8
3.2.2	Interfaccia di controllo . . . . .	11
3.2.3	Adattamento I <sup>2</sup> C . . . . .	14
<b>4</b>	<b>Prove effettuate</b>	<b>18</b>
4.1	Impostazione . . . . .	18
4.2	Test . . . . .	19
4.3	Risultati . . . . .	19
<b>5</b>	<b>Il codice</b>	<b>24</b>
<b>6</b>	<b>Conclusioni</b>	<b>28</b>
<b>Appendici</b>		<b>30</b>
<b>Appendice A Collegamento Scheda-Sensore</b>		<b>31</b>
<b>Appendice B Correzione errori</b>		<b>32</b>
B.1	I <sup>2</sup> C . . . . .	32
B.2	Ottimizzazione . . . . .	32

# Capitolo 1

## Introduzione

Il progetto del Ducted Fan consiste in un drone con due eliche sovrapposte situate nel suo asse centrale e racchiuse all'interno di un corpo rigido. Esso è perciò in grado di volare vicino pareti ed oggetti senza rischio di rompere le eliche o danneggiare qualcosa ed è quindi utilizzabile per effettuare misurazioni o rilevamenti in ambienti dove potrebbe trovare diversi ostacoli.

La parte del progetto trattata riguarda la misurazione della quota alla quale si trova il drone. Tale aspetto veniva gestito negli scorsi anni attraverso un sonar, montato sul drone, che serviva per rilevare la distanza da terra. Esso era però soggetto a molti disturbi di riflessione con il terreno e incontrava diversi problemi nel caso la superficie non era perfettamente orizzontale. Quest'anno si è perciò optato per cambiare sensore, in particolare con un senore laser immune a disturbi di riflessione e poco soggetto ai cambiamenti di luce, ovvero il VL53L1X. Lo scopo è quello di interfacciare tale sensore con la scheda Renesas YRDKRX63N, con la quale era già stata sviluppata la vecchia versione del Ducted Fan. Attraverso il protocollo di comunicazione I<sup>2</sup>C, leggere il valore della distanza rilevata, dando una stima dell'errore che si commette in tutto il range di misurazione.

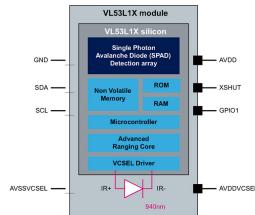
## Capitolo 2

# Il sensore VL53L1X

Il sensore di distanza VL53L1X STMicroelectronics (mostrato in Figura 2.1) utilizza la tecnologia "Time of Flight" (ToF) di ultima generazione. Tale tecnologia consente una misura della distanza indipendente dalla riflettanza dell'obiettivo, a differenza di quanto avviene con la classica tecnologia a Infrarosso (IR) che si limita a misurare l'intensità del segnale riflesso (si veda Capitolo 2.3). Il sensore è costituito da una matrice di ricevimento di 16x16 SPAD (Single-Photon Avalanche Diode), un laser a infrarosso di tipo VCSEL (Vertical Cavity Surface Emitting Laser) di Classe 1 e con lunghezza d'onda di 940 nm e da un microcontrollore embedded a bassa potenza. Il dispositivo inoltre integra dei filtri infrarosso e una lente in corrispondenza della matrice ricevente che contribuiscono al miglioramento delle prestazioni, soprattutto in termini di portata. Grazie a queste tecnologie il sensore riesce ad offrire una misura di distanza accurata da 4 cm fino a 4 m ad una frequenza massima di 50 Hz. Il dispositivo offre anche la possibilità di programmare le dimensioni e la posizione della Region of Interest (ROI) della matrice ricevente, consentendo quindi la riduzione e lo spostamento del Field of View (FoV) del sensore. Il suo schema a blocchi è mostrato in Figura 2.2.



**Figura 2.1:** Il sensore VL53L1X.



**Figura 2.2:** Schema a blocchi del sensore VL53L1X <sup>1</sup>.

<sup>1</sup> Fonte [Datasheet](#)

## 2.1 Caratteristiche tecniche

- Modulo (mostrato in Figura2.1) miniaturizzato completamente integrato in un package Optical Land Grid Array (OPLGA):
  - Misura: 4.9 mm · 2.5 mm · 1.56 mm.
  - Emettitore: laser invisibile 940 nm (Classe 1).
  - Ricevitore: matrice di SPAD (16x16) con lente integrata.
  - Microcontrollore a bassa potenza con firmware digitale avanzato.
  - Tensione operativa: da 2.6 V a 3.5 V (2.8 V valore raccomandato).
  - Temperatura operativa: da -20 °C a 85 °C.
- Carrier board (mostrata in Figura2.5):
  - Misura: 13 mm · 18 mm · 2 mm.
  - Regolatore di tensione: fornisce i 2.8 V richiesti dal sensore permettendo un'alimentazione da 2.6 V a 5.5 V .
  - Traslatore di livello: permette la comunicazione tra dispositivi con diverse tensioni di lavoro.
  - Tensione operativa: da 2.6 V a 5.5 V.
- FoV: programmabile da 15° a 27°
- Facile integrazione:
  - Singolo componente rifrangente.
  - Può essere coperto da diversi materiali (utile per evitare il danneggiamento del sensore).
  - Alimentazione singola (2v8).
  - Interfaccia I2C (fino a 1 MHz).

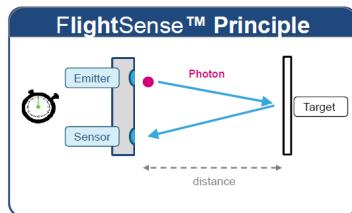
## 2.2 Possibili applicazioni

- Droni (assistenza per atterraggio, hovering, rilevamento soffitto).
- Rilevamento utente (modalità di risparmio energetico autonomo) per accendere/spegnere e bloccare/sbloccare dispositivi come personal computer, laptop e IoT.
- Smart Building e Smart Lighting (rilevamento degli utenti e controllo dei gesti).

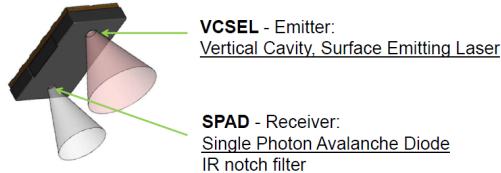
- Robot di servizio e aspirapolvere (veloce rilevamento di ostacoli a lunga distanza).
- Riconoscimento di gesti 1D.
- Auto-focus.

## 2.3 Funzionamento

La tecnologia ToF (Figura 2.3), su cui si basa il funzionamento del sensore, consiste nel misurare la distanza sensore-oggetto utilizzando il “tempo di volo”, cioè il tempo che l’impulso luminoso generato da un emettitore impiega per raggiungere l’ostacolo, essere riflesso e tornare indietro eccitando il ricevitore (Figura 2.4). Data la velocità della luce ( $c$ ) e misurato il tempo trascorso tra l’impulso inviato e quello ricevuto ( $\Delta t$ ), il valore della distanza ( $d$ ) viene calcolato semplicemente con la formula  $d = \frac{c\Delta t}{2}$ .



**Figura 2.3:** Tecnologia ToF <sup>2</sup>.



**Figura 2.4:** Struttura emettitore-ricevitore del sensore <sup>3</sup>.

In particolare il sensore VL53L1X utilizza come emettitore un laser VCSEL (laser a cavità verticale a emissione superficiale) a 940 nm e come ricevitore una matrice 16x16 di SPAD (diodo a valanga rilevatore di singoli fotoni). Il sensore integra inoltre una lente che, posta di fronte al ricevitore, contribuisce a migliorare le prestazioni del dispositivo sotto diversi aspetti e in particolare permette di aumentare la distanza massima rilevata (il VL53L1X ha una portata di 4m rispetto ai 2m raggiunti dal sensore VL53L0X che utilizza la stessa tecnologia ma è privo di lente). L’obiettivo permette inoltre di ridurre l’effetto interferente della luce ambientale agendo come filtro e consente anche di modificare il campo visivo del sensore. Il laser emette un fascio luminoso la cui potenza è concentrata in un cono con un’ampiezza di 27°, di conseguenza il campo visivo dell’emettitore non è modificabile. Il sensore permette invece di intervenire sul FoV del ricevitore, variando posizione e dimensioni della ROI sulla matrice ricevente e cioè selezionando una sottomatrice di SPADs. Di default il sensore utilizza l’intera superficie del ricevitore (16x16 SPADs), elaborando tutte le informazioni da

---

<sup>2</sup>Fonte [Presentazione ST](#)

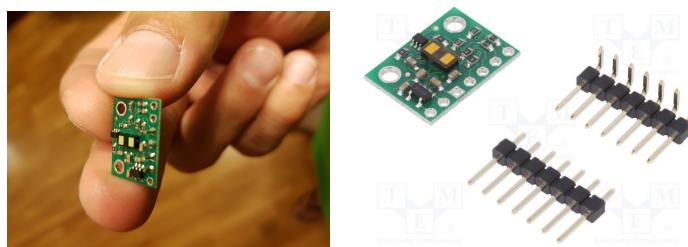
<sup>3</sup>Fonte [Presentazione ST](#)

essa catturate ed analizzando quindi tutto il campo visivo disponibile ( $27^\circ$ ). Tuttavia alcune applicazioni non hanno bisogno dell'intero FoV e il tentativo di catturarlo nella sua totalità costituisce soltanto uno spreco di risorse. In alcune situazioni inoltre potrebbe essere indispensabile ridurre la ROI per evitare riflessioni che comprometterebbero l'accuratezza della misura. Per questo motivo il sensore offre la possibilità di ridurre il numero di SPADs utilizzati fino ad un minimo di  $4 \times 4$  SPADs ( $15^\circ$ ).

Occorre però considerare che una riduzione della ROI comporta anche una riduzione del numero di fotoni ricevuti e quindi una diminuzione significativa della distanza massima rilevabile dal dispositivo (3,6m con la ROI massima rispetto a 1,7m con quella minima<sup>4</sup>). Inoltre la posizione della ROI va stabilita considerando la presenza della lente convergente e quindi la relazione tra le coordinate dell'oggetto nel campo visivo e le corrispondenti coordinate sulla matrice ricevente.

Il sensore mette inoltre a disposizione la possibilità di scegliere tra 3 diverse Ranging Options (o Distance Mode): Short, Medium e Long. La scelta della modalità determina una diversa configurazione del sensore (non sono fornite ulteriori informazioni sul datasheet) che consente di ottenere i migliori risultati in base alle distanze di interesse e alle condizioni di illuminazione dell'ambiente. In particolare la modalità Long permette di raggiungere la massima distanza di 4 m ma è fortemente influenzata dalla luce dell'ambiente; la modalità Short è la più immune ai disturbi luminosi ma copre un range massimo di 1,3 m.

Il sensore a cui dovranno essere saldati i pin è mostrato in Figura 2.5.



**Figura 2.5**

---

<sup>4</sup>Fonte [Datasheet](#)

## Capitolo 3

# Sviluppo del progetto

Il sensore acquistato non dispone di una documentazione che descrive i suoi registri interni e le procedure per configurarlo ed utilizzarlo, fornisce invece un'API per interfacciarsi con esso. È stato necessario adattare innanzitutto la libreria API per poter essere successivamente in grado di testarlo sulla scheda Renesas. Data la vasta gamma di funzionalità del sensore si è optato per un'API semplificata, sviluppata per un utilizzo su Arduino. All'interno dell'API non è comunque specificato l'uso dei vari registri, quindi molte funzioni non sono state completamente commentate. Dopo aver testato la correttezza della libreria con Arduino Mega si è passati alla sua riscrittura per poter usare il sensore con il microcontrollore utilizzato.

Per adattare la libreria bisogna trovare la soluzione a due problemi principali:

- La libreria API per Arduino è stata scritta in linguaggio C++, definendo il sensore come una classe con i relativi metodi, abbiamo effettuato un porting da C++ a C (linguaggio scelto all'interno del progetto).
- La scheda Arduino gestisce ogni interfaccia con il sensore tramite una sua libreria chiamata "Wire", quindi per poter usare il sensore con la scheda Renesas è stato necessario adattare l'interfaccia I<sup>2</sup>C già esistente.

Dopo aver risolto questi problemi si è passati alla fase di test.

### 3.1 Porting C++/C

Per portare la libreria API dal linguaggio C++ a C è stato riscritto l'header file della libreria modificando le classi presenti in strutture; nella libreria per la scheda Arduino era presente una classe "VL53L1X" che a sua volta conteneva elementi di tipo struttura "ResultBuffer" e "RangingData", mentre nella libreria riscritta per la scheda Renesas è stata creata una struttura "vl53l1x" che contiene elementi di tipo struttura "ResultBuffer" e "RangingData".

Per poter utilizzare i metodi della classi senza dover effettuare eccessive modifiche al codice sono stati lasciati inalterati i nomi delle funzioni, ma anzichè essere definiti nell'header file come metodi della classe "VL53L1X", sono state definite come funzioni che prendono in argomento (oltre agli argomenti che prendono le omonime funzioni della libreria API per Arduino) un elemento di tipo riferimento alla struttura "vl53l1x" (B.2).

In figura sono mostrate le differenze tra l'inizializzazione del sensore nella libreria per la scheda Arduino e quella adattata per la scheda Renesas.

```
//Inizializzazione della classe VL53L1X mediante
//costruttori
VL53L1X::VL53L1X()
: address(AddressDefault)
, io_timeout(0) // no timeout
, did_timeout(false)
, calibrated(false)
, saved_vhv_init(0)
, saved_vhv_timeout(0)
, distance_mode(Unknown)
{
```

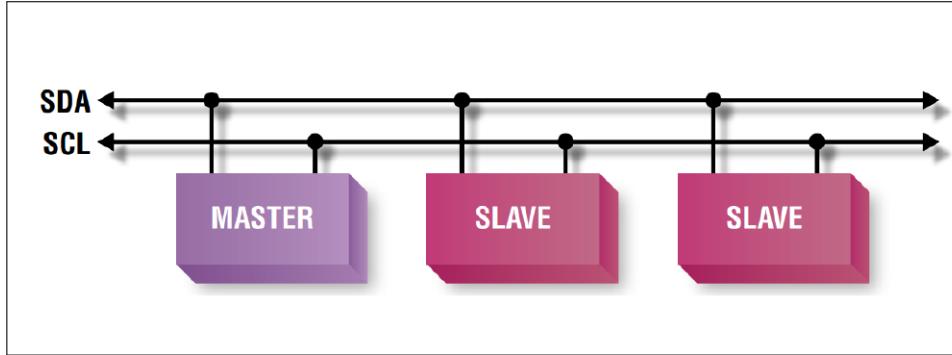
```
//Inizializzazione della struttura vl53l1x
void Sensor_Init(vl53l1x* sens)
{
    sens->address = AddressDefault;
    sens->io_timeout = 0; // no timeout
    sens->did_timeout = false;
    sens->calibrated = false;
    sens->saved_vhv_init = 0;
    sens->saved_vhv_timeout = 0;
    sens->distance_mode = Unknown;
}
```

## 3.2 Interfaccia di comunicazione I<sup>2</sup>C

### 3.2.1 Protocollo I<sup>2</sup>C

Il protocollo I<sup>2</sup>C sta per Inter-Integrated Circuit. Il protocollo permette la comunicazione di dati tra due o più dispositivi I<sup>2</sup>C utilizzando un bus a due fili, più uno per il riferimento comune di tensione (Figura3.1). In tale protocollo le informazioni sono inviate serialmente usando una linea per i dati (SDA: Serial Data line) ed una per il Clock (SCL: Serial Clock line).

Deve inoltre essere presente una terza linea: la massa, comune a tutti i dispositivi.



**Figura 3.1:** Bus I<sup>2</sup>C cui sono connessi un dispositivo master e due dispositivi slave.

In generale ci sono 4 distinti modi di operare:

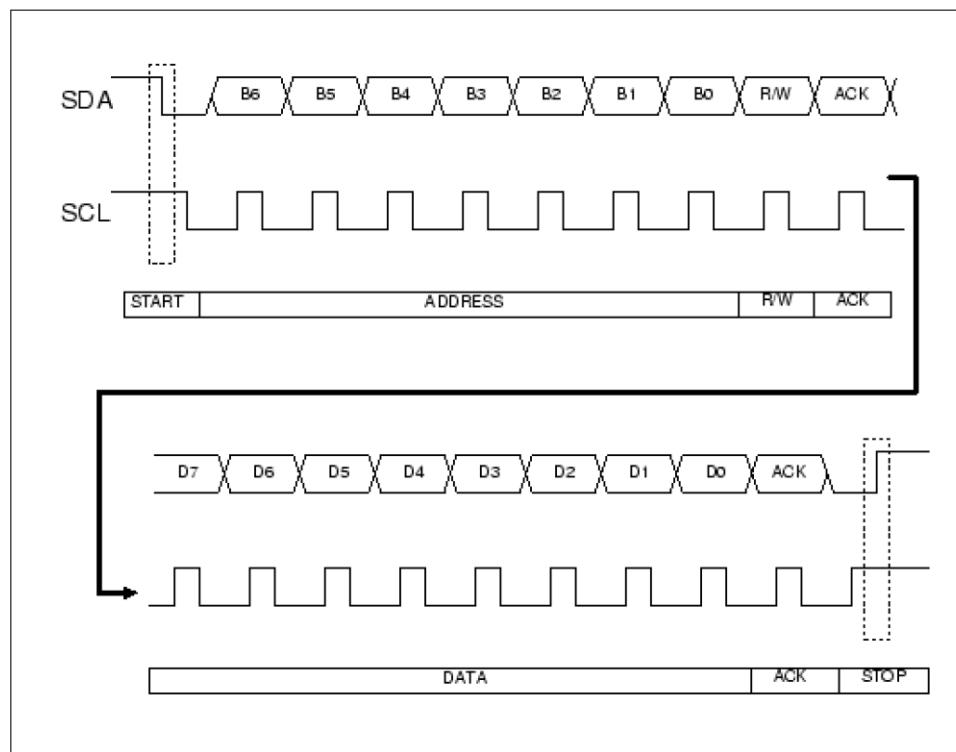
- Un master trasmette - genera il clock e invia dati agli slave.
- Un master riceve - genera il clock e riceve dati dallo slave.
- Uno slave trasmette - il dispositivo riceve il clock e invia dati al master.
- Uno slave riceve - il dispositivo riceve il clock e riceve dati dal master.

Il dispositivo master è il dispositivo che controlla il bus in un certo istante (in questo caso è la scheda Renesas); tale dispositivo controlla il segnale di Clock e genera i segnali di START e di STOP. I dispositivi slave (in questo caso è il sensore di distanza, a cui verranno successivamente collegate altre periferiche come ad esempio l'IMU) “ascoltano” il bus ricevendo dati dal master o inviandone qualora questo ne faccia loro richiesta.

In tutte le modalità i dati vengono trasmessi a pacchetti di 8 bit seguiti da un bit di Acknowledge (ACK). Il byte di dati è trasferito sulla linea SDA dal Most Significant Bit (MSB) al Less Significant Bit (LSB) ad opera del dispositivo che sta trasmettendo (master/slave), mentre l'ACK è generato dal ricevitore (slave/master) solo se questo riconosce il dato trasmesso (l'assenza del bit di ACK è interpretata come Not Acknowledge NACK). Ogni sequenza di dati trasmessa è preceduta da una condizione di START e terminata da una condizione di STOP, entrambe generate dal master. Il primo byte di dati è sempre trasmesso dal master e contiene l'indirizzo a 7 bit dello slave seguito da un bit che indica il verso della trasmissione (0: master trasmette e slave riceve; 1: slave trasmette e master riceve). Concretamente queste operazioni sono eseguite come segue:

- Dati: i singoli bit del dato trasmesso vengono trasferiti sulla linea SDA quando il clock è basso e devono essere stabili quando l'SCL è alto (vengono infatti letti dal ricevitore in queste condizioni).
- START / STOP: la linea SDA è commutata dal master mentre il clock è alto (si distingue per questo dai bit del dato), in particolare la transizione alto-basso corrisponde allo START (in assenza di trasmissione la linea SDA è infatti alta), la transizione basso-alto corrisponde allo STOP (dopo la trasmissione l'SDA deve essere riportato alto per ripristinare le condizioni iniziali).
- ACK / NACK: il ricevitore prende il controllo della linea SDA, lasciata alta dal trasmettitore, e la forza bassa; l'assenza di questa forzatura è interpretata come NACK ed è generalmente sintomo di un errore di trasmissione, viene però anche utilizzata in modalità lettura dal master per fermare la trasmissione dei dati da parte dello slave.

Una sequenza elementare di lettura o scrittura di dati tra master e slave segue il seguente ordine (Figura 3.2):



**Figura 3.2:** Tipica sequenza di trasferimento dati con il protocollo I<sup>2</sup>C

1. Invio del bit di START (S) da parte del master.

2. Invio dell'indirizzo dello slave (ADDRESS) ad opera del master.
3. Invio del bit di Read (R) o di Write (W), che valgono rispettivamente 1 e 0 (sempre ad opera del master).
4. Attesa/invio del bit di Acknowledge (ACK), il quale viene emesso in risposta alla ricezione di un'informazione completa.
5. Invio/ricezione del byte dei dati (DATA).
6. Attesa/invio del bit di Acknowledge (ACK).
7. Invio del bit di STOP (P) da parte del master.

I passi 5 e 6 possono essere ripetuti così da leggere o scrivere più byte.

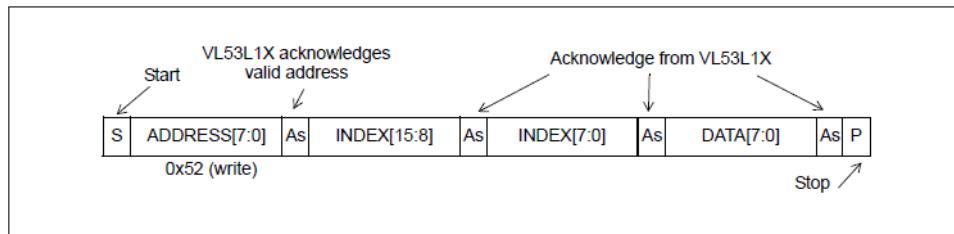
### 3.2.2 Interfaccia di controllo

Il sensore VL53L1X utilizza il protocollo di comunicazione I<sup>2</sup>C descritto dettagliatamente nel precedente paragrafo. L'indirizzo di default del dispositivo è 0x52 (si veda Appendice B.1). La velocità massima di comunicazione permessa dal sensore è di 400 Kbps (I<sup>2</sup>C fast mode) ma non è stata sfruttata nel progetto che utilizza invece una velocità di soli 100Kbps (I<sup>2</sup>C slow mode); per incrementarla occorre intervenire direttamente sulla funzione di libreria della Renesas *R\_RIIC\_Init()* ma non è stato necessario nel progetto sviluppato.

Entrando più nello specifico il sensore permette 4 modalità di trasmissione dati: Write, Read, Sequential Write e Sequential Read. In realtà queste 4 modalità descritte nel datasheet del dispositivo coincidono in pratica con le semplici modalità di lettura e scrittura proprie della comunicazione I<sup>2</sup>C e descritte in precedenza. Vengono distinte soltanto per evidenziare che il sensore supporta la lettura/scrittura sequenziale: è possibile cioè operare in lettura/scrittura su registri adiacenti del dispositivo specificando soltanto l'indirizzo del primo registro, l'indirizzo del registro dove si sta operando è infatti incrementato automaticamente di 1 dopo il trasferimento di ogni byte. Ciò appare evidente analizzando nel dettaglio le singole modalità. La modalità Write è mostrata in Figura3.3 e consiste nei seguenti passi (nel caso di riuscita trasmissione):

1. Generazione della condizione di Start da parte del master.
2. Trasmissione da parte del master dell'indirizzo a 7 bit del sensore seguito dal bit 0 che indica la modalità di scrittura: 0x52.
3. Bit di ACK generato dal sensore (As).

4. Trasmissione da parte del master del primo byte (il più significativo) dell'indirizzo a 16 bit del registro del VL53L1X dove effettuare l'operazione di scrittura.
5. Bit di ACK generato dal sensore (As).
6. Trasmissione da parte del master del secondo byte (il meno significativo) dell'indirizzo a 16 bit del registro del VL53L1X dove effettuare l'operazione di scrittura.
7. Bit di ACK generato dal sensore (As).
8. Trasmissione da parte del master del dato (1 byte) che deve essere scritto sul registro il cui indirizzo è stato precedentemente trasmesso.
9. Bit di ACK generato dal sensore (As).
10. Generazione della condizione di Stop da parte del master.



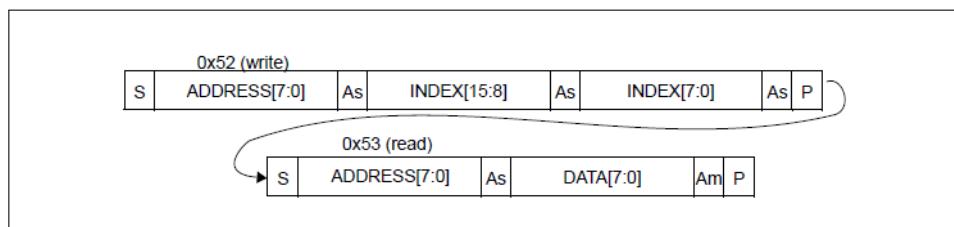
**Figura 3.3:** Schema della modalità Write.

La modalità di Read (mostrata in Figura 3.4) è analoga ma necessita di operazioni aggiuntive: i primi passi ricalcano esattamente quelli della modalità di Write (passi 1-7), prima della trasmissione dei dati da parte dello slave è però necessaria una nuova condizione di Start e il rinvio del registro dello slave (stavolta in modalità lettura).

Le operazioni (sempre nel caso di riuscita trasmissione) sono le seguenti:

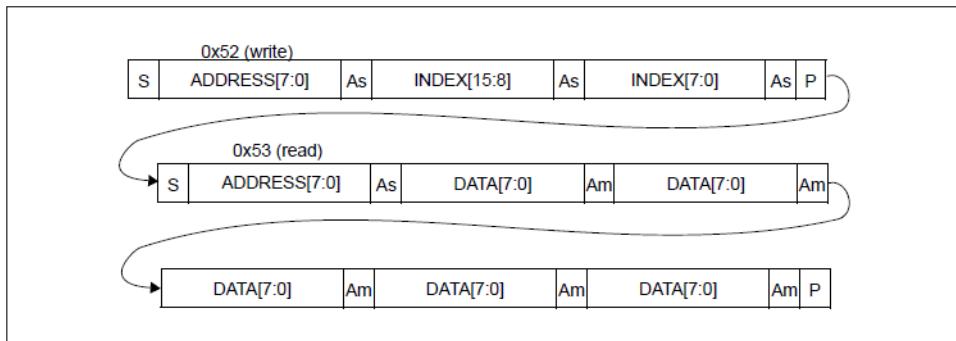
1. Generazione della condizione di Start da parte del master.
2. Trasmissione da parte del master dell'indirizzo a 7 bit del sensore seguito dal bit 0 che indica la modalità di scrittura: 0x52.
3. Bit di ACK generato dal sensore (As).
4. Trasmissione da parte del master del primo byte (il più significativo) dell'indirizzo a 16 bit del registro del VL53L1X dove effettuare l'operazione di lettura.
5. Bit di ACK generato dal sensore (As).

6. Trasmissione da parte del master del secondo byte (il meno significativo) dell'indirizzo a 16 bit del registro del VL53L1X dove effettuare l'operazione di lettura.
7. Bit di ACK generato dal sensore (As).
8. Generazione di un'ulteriore condizione di Start da parte del master.
9. Trasmissione da parte del master dell'indirizzo a 7 bit del sensore seguito dal bit 1 che indica la modalità di lettura: 0x53.
10. Bit di ACK generato dal sensore (As).
11. Trasmissione da parte dello slave del dato (1 byte) contenuto nel registro il cui indirizzo è stato precedentemente trasmesso.
12. Bit di ACK generato dal master (Am): nel caso di lettura di un solo byte è in realtà un bit di NACK necessario per sospendere la trasmissione dei dati da parte del sensore, lo slave continua infatti a trasmettere se riceve un ACK da parte del master (questo aspetto non è però ben specificato nel datasheet del sensore).
13. Generazione della condizione di Stop da parte del master.

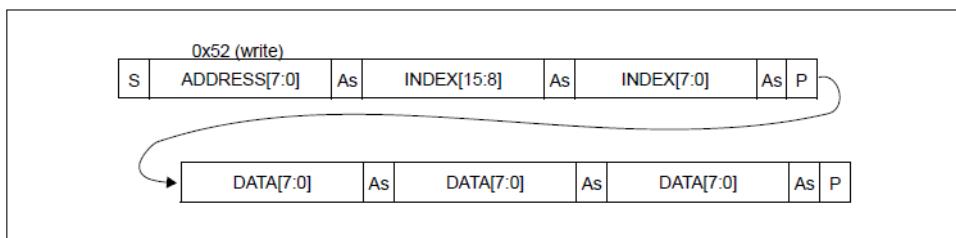


**Figura 3.4:** Schema della modalità Read.

Come anticipato precedentemente le modalità di Sequential Read e Sequential Write coincidono di fatto con le modalità di Read e Write ma permettono la lettura/scrittura di più byte su registri consecutivi. In particolare la modalità Sequential Read (mostrata in Figura 3.5) ripete i passi 11 e 12 della Read mentre la modalità Sequential Write (mostrata in Figura 3.6) ripete i passi 8 e 9 della Write.



**Figura 3.5:** Schema della modalità Sequential Read.



**Figura 3.6:** Schema della modalità Sequential Write.

### 3.2.3 Adattamento I<sup>2</sup>C

La gestione della comunicazione scheda-sensore tramite protocollo I<sup>2</sup>C si è rivelata il principale problema affrontato nel corso del progetto. Per riuscire ad ottenere delle letture e scritture corrette dei registri non è stato infatti sufficiente modificare la libreria fornita per l'utilizzo del lidar, sostituendo le funzioni Arduino per la comunicazione I<sup>2</sup>C con le analoghe per la scheda Renesas già implementate. Si è dovuto invece analizzare e studiare con un discreto grado di dettaglio non soltanto le funzioni della libreria *i2c.h* ereditata da gruppi precedenti ma anche le principali funzioni in essa richiamate implementate nel file *r\_riic\_rx600\_master.c* della libreria Renesas *r\_riic\_rx600*. Soltanto dopo questo è stato possibile individuare e risolvere una serie di problematiche e arrivare quindi alla successiva fase di test del sensore. Tutte le funzioni implementate nella libreria Arduino (ad eccezione della *ReadResult* che è poi stata modificata e resa come le altre) si basano su 6 funzioni fondamentali per il funzionamento del lidar che permettono la scrittura e lettura di 8/16/32 bit dai registri del sensore: *writeReg*, *writeReg16*, *writeReg32*, *readReg*, *readReg16*, *readReg32* (prototipi riportati in Figura 3.7). Le funzioni di write prendono in ingresso l'indirizzo a 16 bit del registro del sensore a partire dal quale effettuare l'operazione e il valore da andare a scrivere (8/16/32 bit); le funzioni di read hanno sempre come

input l'indirizzo ma restituiscono il valore letto (8/16/32 bit). Entrambe le funzioni prendono in ingresso anche un riferimento alla struttura sensore (si veda il Capitolo3.1) per memorizzare nel suo campo *last\_status* l'esito dell'ultima operazione di scrittura o di lettura effettuata.

```
void writeReg( uint16_t reg, uint8_t value,vl53l1x*);
void writeReg16Bit(uint16_t reg, uint16_t value,vl53l1x*);
void writeReg32Bit(uint16_t reg, uint32_t value,vl53l1x*);
uint8_t readReg(uint16_t reg,vl53l1x*);
uint16_t readReg16Bit(uint16_t reg,vl53l1x*);
uint32_t readReg32Bit(uint16_t reg,vl53l1x*);
```

**Figura 3.7:** Dichiarazione delle funzioni *WriteReg* e *ReadReg*.

Queste funzioni erano implementate utilizzando la libreria Arduino dedicata alla comunicazione I<sup>2</sup>C (*Wire.h*) come mostrato in Figura3.8 ed in Figura3.9.

Per modificare tali funzioni e renderle compatibili con la nostra scheda il

```
Wire.beginTransmission(address);
Wire.write((reg >> 8) & 0xFF); // reg high byte
Wire.write( reg      & 0xFF); // reg low byte
Wire.write((value >> 8) & 0xFF); // value high byte
Wire.write( value    & 0xFF); // value low byte
last_status = Wire.endTransmission();
```

**Figura 3.8:** Implementazione della funzione *WriteReg16Bit* per Arduino.

```
Wire.beginTransmission(address);
Wire.write((reg >> 8) & 0xFF); // reg high byte
Wire.write( reg      & 0xFF); // reg low byte
last_status = Wire.endTransmission();

Wire.requestFrom(address, (uint8_t)2);
value  = (uint16_t)Wire.read() << 8; // value high byte
value |=           Wire.read(); // value low byte
```

**Figura 3.9:** Implementazione della funzione *ReadReg16Bit* per Arduino.

riferimento che ci è stato fornito è stata la libreria i2c.h precedentemente

nominata ed in particolare le funzioni *i2c\_write* e *i2c\_read* in essa implementate (prototipi in Figura3.10). Per ridurre al minimo la riscrittura della

```
int i2c_read (uint8_t slave_addr, uint8_t register_number, uint8_t num_bytes, uint8_t *dest_buff);

int i2c_write(uint8_t slave_addr, uint8_t register_number, uint8_t num_bytes, uint8_t *source_buff);
```

**Figura 3.10:** Dichiarazione delle funzioni *i2c\_write* e *i2c\_read* per Renesas.

libreria Arduino non sono state utilizzate direttamente queste funzioni, ma si è preferito richiamarle all'interno delle funzioni di lettura e scrittura. È stato quindi necessario prima di tutto adattare i parametri delle diverse funzioni trasformando vettori di `uint8_t` in `uint16_t` e `uint32_t` e viceversa (per farlo sono stati utilizzati semplici shift) e gestendo opportunamente l'indirizzo dello slave come un `uint8_t` (B.1). Nel fare questo però ci si è accorti della necessità di modificare le funzioni *i2c\_read* e *i2c\_write*, pensate per la lettura e scrittura di registri con indirizzo a 8 bit, in modo da poter essere utilizzate con i registri del sensore aventi indirizzo a 16 bit, implementando gli schemi mostrati in Figura3.4 e Figura3.3. Questo è stato il problema che ci ha costretto ad approfondire, almeno in parte, il file *r\_riic\_rx600\_master.c* della Renesas contenente tutte le funzioni necessarie alla gestione del protocollo I<sup>2</sup>C. Analizzando l'implementazione delle principali funzioni in esso contenute (*R\_RIIC\_MasterTransmitHead*, *R\_RIIC\_MasterReceive*, *R\_RIIC\_MasterTransmit*) e prendendo come riferimento l'utilizzo della libreria *Wire.h* fatto nelle funzioni di read e write da modificare si è arrivati alla soluzione. Le funzioni sono quindi state modificate in modo da essere utilizzabili indipendentemente dalla dimensione degli indirizzi (8/16 bit). Questo risultato è stato ottenuto aggiungendo in entrambe le funzioni il parametro *head\_num\_byte* (Figura3.11) con il numero di byte del Trasmit Head ("testa" della comunicazione) e modificando i parametri della funzione *MasterTrasmitHead* in esse richiamata. È questa infatti la funzione che si occupa dell'invio da parte del master dell'intestazione della comunicazione con le informazioni sull' indirizzo dello slave e l'indirizzo del registro dove effettuare le successive operazioni di lettura/scrittura. In base alla dimensione degli indirizzi dello slave la Trasmit Head ha quindi la seguente forma:

- **Indirizzi a 8 bit:** 1 byte slave address + 1 byte register address.
- **Indirizzi a 16 bit:** 1 byte slave address + 2 byte register address.

```

int i2c_read (uint8_t slave_addr, uint16_t register_number,
               uint8_t num_bytes, uint8_t *dest_buff, uint8_t head_num_byte)

int i2c_write(uint8_t slave_addr, uint16_t register_number,
                uint8_t num_bytes, uint8_t *source_buff, uint8_t head_num_byte)

```

**Figura 3.11:** Dichiarazione delle nuove funzioni *i2c\_write* e *i2c\_read* per Renesas.

```

int i2c_read (uint8_t slave_addr, uint16_t register_number,
               uint8_t num_bytes, uint8_t *dest_buff, uint8_t head_num_byte) {

    uint8_t addr_and_register[3];
    riic_ret_t ret = RIIC_OK;

    if(head_num_byte == 2){ //SLAVE ADDR + 8bit REGISTER
        uint8_t register_number_adapted = (uint8_t)register_number; //CAST FROM 16bit TO 8bit
        addr_and_register[0] = slave_addr<<1;
        addr_and_register[1] = register_number_adapted & 0xFF;
    }

    else if (head_num_byte == 3){ //SLAVE ADDR + 8bit H-REG + 8bit L-REG
        addr_and_register[0] = slave_addr<<1;
        addr_and_register[1] = (register_number >> 8) & 0xFF;
        addr_and_register[2] = register_number & 0xFF;
    }

    ret |= R_RIIC_MasterTransmitHead(CHANNEL_0, addr_and_register, head_num_byte);

    /* Now read the data from the target register into the destination buffer. */
    ret |= R_RIIC_MasterReceive(CHANNEL_0, slave_addr<<1, dest_buff, num_bytes);

    return ret;
}

```

**Figura 3.12:** Implementazione della nuova funzione *i2c\_read* per Renesas.

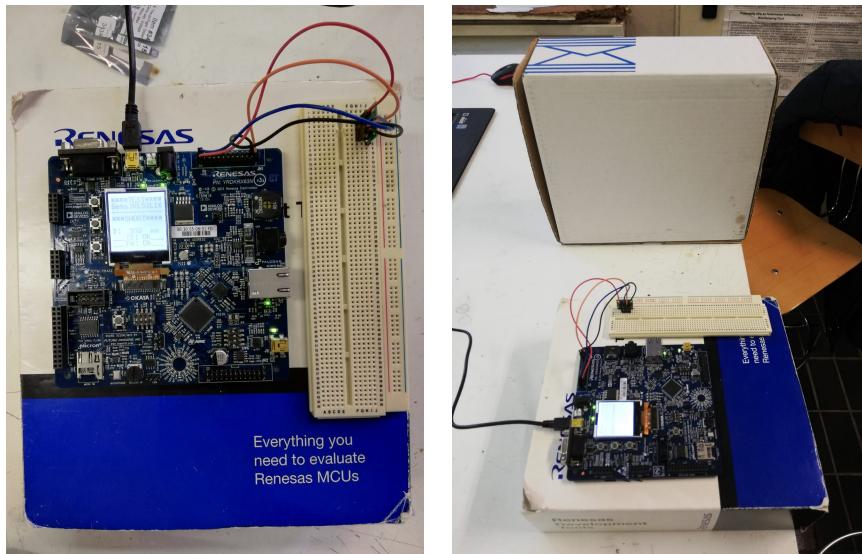
La funzione *MasterTrasmitHead* prende come parametri, oltre al canale di comunicazione utilizzato, il vettore *addr\_and\_register* contenente i byte della testa e la variabile *head\_num\_byte* con il numero effettivo di byte da inviare. In particolare *addr\_and\_register* è stato reso in ogni caso un vettore di **uint8\_t** di 3 elementi (non è infatti permessa l'allocazione dinamica di un vettore di dimensione *head\_num\_byte*); in base al valore di *head\_num\_byte* vengono quindi valorizzati gli elementi del vettore (2/3 elementi per indirizzi a 8/16 bit). L'eventuale byte rimasto non inizializzato non costituisce un problema in quanto la funzione *MasterTrasmitHead* ha come parametro il numero effettivo di byte inviati attraverso il canale, indipendentemente dalla dimensione della testa passata come argomento.

# Capitolo 4

## Prove effettuate

### 4.1 Impostazione

Per testare il funzionamento del sensore inizialmente sono state effettuate prove senza l'ausilio di basi di fissaggio. Successivamente per ottenerne una stima corretta dell'errore, per poterlo valutare, si è optato per la Breadboard<sup>1</sup>. La seconda configurazione è riportata in Figura 4.1. Per il collegamento del sensore alla scheda si rimanda all'Appendice A.



**Figura 4.1:** Impostazione utilizzata per testare il sensore di distanza VL53L1X.

---

<sup>1</sup> Una breadboard consiste in una base in plastica con numerosi fori nei quali inserire i cavi di collegamento (reofori). Le breadboard hanno generalmente una struttura simile composta da linee di trasmissione (strips) che consistono in collegamenti elettrici tra i fori (Fonte [Wikipedia](#)).

## 4.2 Test

I test effettuati sono di due tipologie:

1. Valutazione, per ogni Ranging Option, da 0 a 450 cm della distanza misurata dal sensore.
2. Valutazione, in modalità Ranging Option variabile, da 0 a 60 cm della distanza misurata dal sensore. Questo range analizzato è legato alla quota di volo del Ducted Fan.

Queste due valutazioni sono state effettuate spostando manualmente il sensore, misurando la distanza effettiva e prendendo la media di più misurazioni. La distanza effettiva, misurata manualmente con un metro, è stata soggetta a errori di precisione dovuti alla strumentazione usata. La superficie di appoggio del sensore è stata rialzata per evitare riflessioni da terra, poichè senza rialzo il sensore rilevava ostacoli inesistenti.

Per quanto riguarda il primo test, le misure sono state prese dal pavimento scuro. Nella secondo invece è stato utilizzato un tavolino bianco. E' importante notare come, nonostante la superficie altamente riflettente nel secondo test, i risultati ottenuti siano ottimi poichè la modalità Short è trascurabilmente soggetta a disturbi fino a 135 cm.

Tutti i test sono stati effettuati in un edificio con luce artificiale.

Il setting delle varie modalità è descritto nel Capitolo 5.

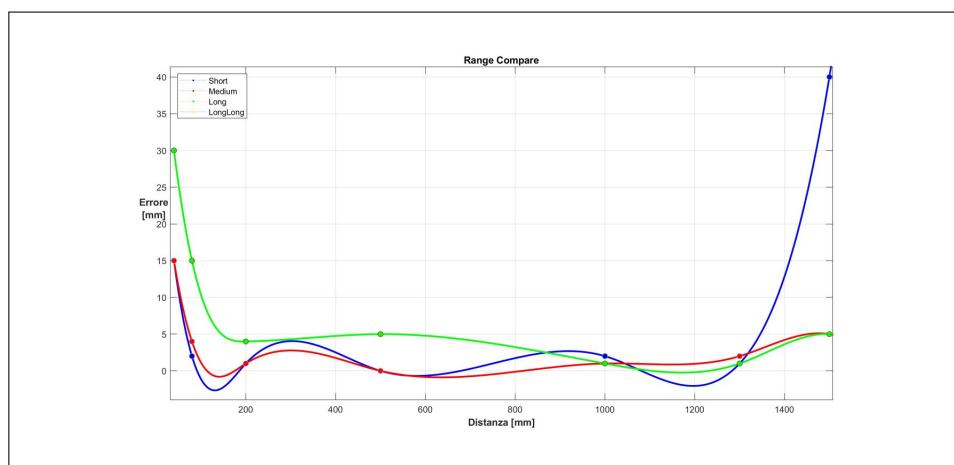
## 4.3 Risultati

I risultati ottenuti sono stati raccolti in file di testo, successivamente sono stati rielaborati e graficati tramite MATLAB. Le Figure 4.2, 4.3, 4.4, 4.5 e 4.6 mostrano il comportamento e l'affidabilità del sensore con l'aumentare della distanza ed al variare della Ranging Option. Sulla base di questi grafici sono state scelte gli intervalli di lavoro delle varie Ranging Option. Dato l'eccessivo errore dopo 3400 mm, è stata creata una quarta modalità LongLong, avente gli stessi settaggi della modalità Long ma con *Timing Budget* aumentato come da manuale.

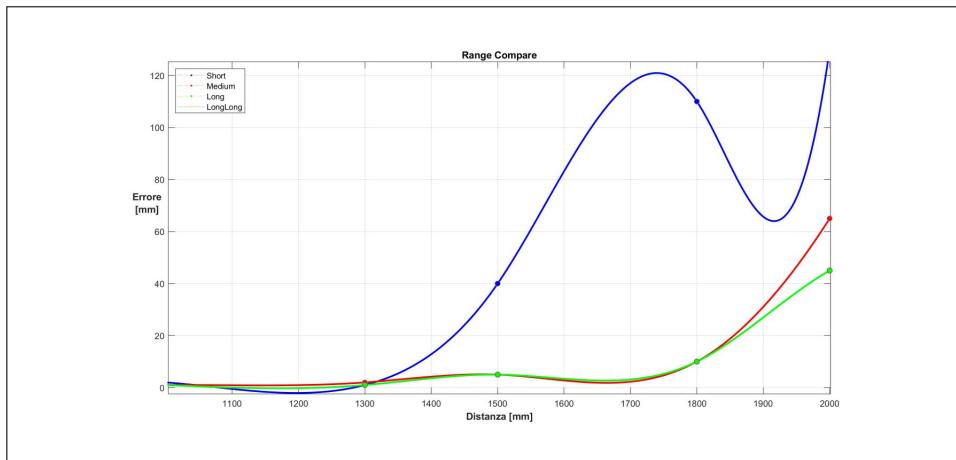
In Figura 4.7 è mostrato l'andamento dell'errore di misura in funzione della distanza nell'intervallo di distanze interessate dal volo del Ducted Fan (i punti rossi sul grafico rappresentano le misure effettuate in laboratorio, mentre la linea blu è l'interpolazione tra i punti). Nella Tabella 4.1 sono mostrate le distanze per cui è consigliato utilizzare le varie Ranging Options e che sono state adottate per la scrittura del codice.

Modalità	Range Minimo (mm)	Range Massimo (mm)	Tempo di misura (ms)	Timing Budget (ms)
<b>Short</b>	0	1300	50	20
<b>Medium</b>	1301	1800	110	50
<b>Long</b>	1801	3400	110	50
<b>LongLong</b>	3401	4500	290	140

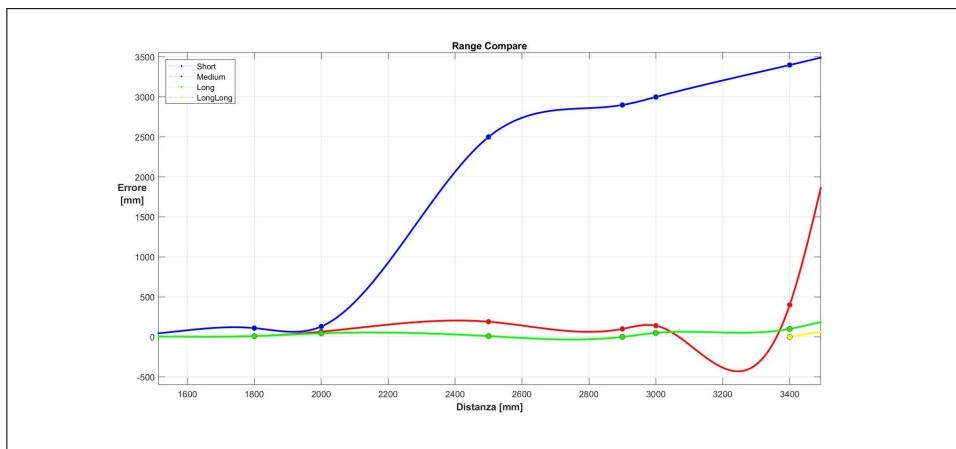
**Tabella 4.1:** tabella delle Ranging Options.



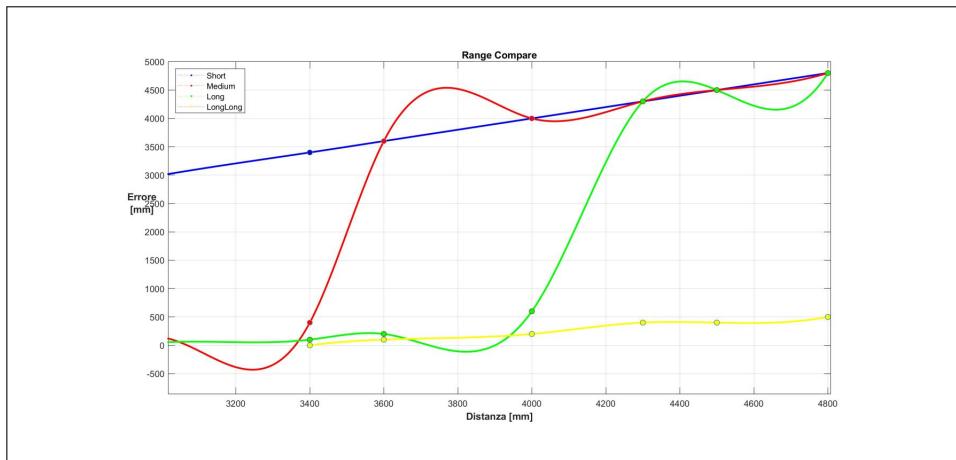
**Figura 4.2:** Dettaglio dell’andamento dell’errore di misura in funzione della distanza nell’intervallo 0/1400 mm.



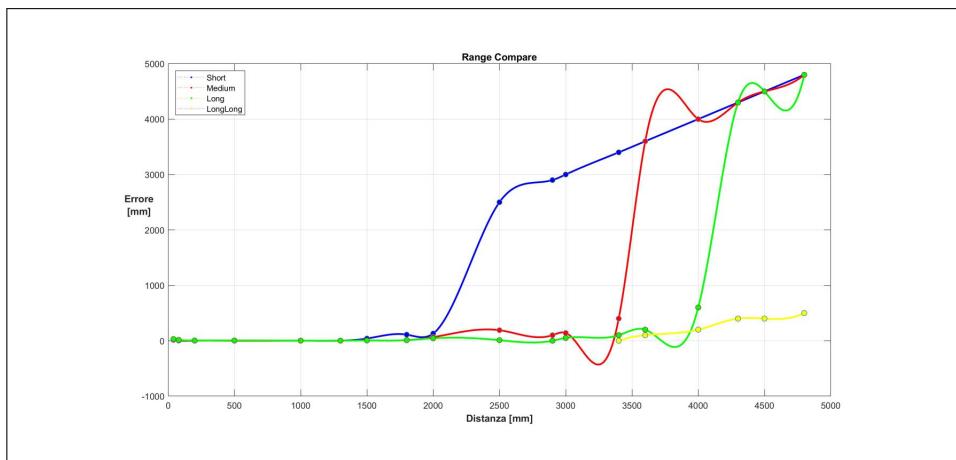
**Figura 4.3:** Dettaglio dell’andamento dell’errore di misura in funzione della distanza nell’intervallo 1000/2000 mm.



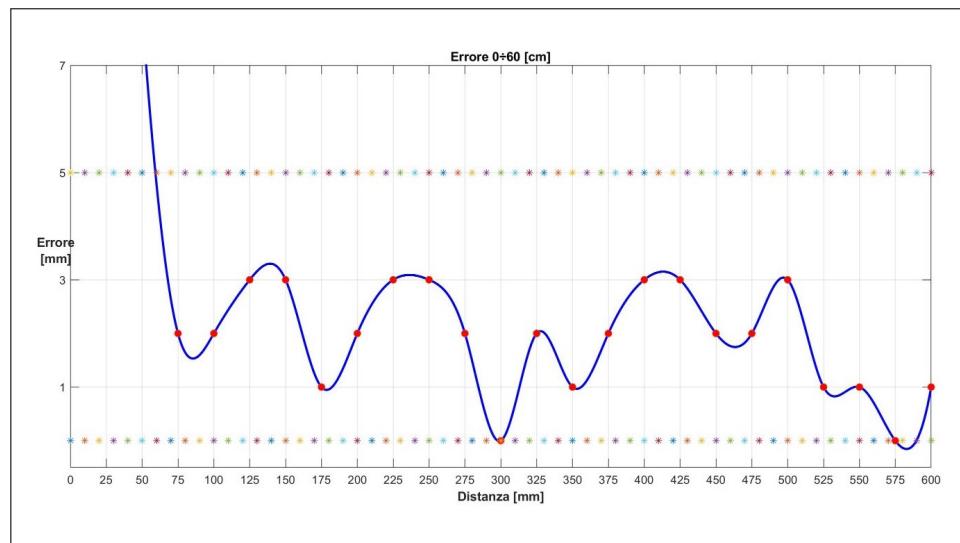
**Figura 4.4:** Dettaglio dell’andamento dell’errore di misura in funzione della distanza nell’intervallo 1600/3400 mm.



**Figura 4.5:** Dettaglio dell’andamento dell’errore di misura in funzione della distanza nell’intervallo 3100/4800 mm.



**Figura 4.6:** Andamento complessivo dell’errore di misura in funzione della distanza.



**Figura 4.7:** Andamento dell'errore di misura nel range di interesse.

# Capitolo 5

## Il codice

La realizzazione del Main è stata effettuata considerando che tale parte dovrà poi essere implementata in unico codice che comprenda tutti i diversi task del progetto, perciò si è cercato di rendere il tutto molto compatto. Vengono infatti per prima cosa effettuata un'inizializzazione del timer CMT, della periferica I<sup>2</sup>C e del display. Poi attraverso la funzione *initialize()*(Figura 5.1) si provvede in maniera sequenziale, di effettuare l'inizializzazione e l'assegnazione di valori di default alla struttura in cui verranno memorizzati tutti i parametri utili alla lettura della distanza attraverso la funzione *Sensor\_Init*. Vengono inoltre settati i parametri per effettuare la prima lettura dell'altezza in modalità Short, poiché si presuppone che all'accensione il drone si trovi a terra. Una volta effettuata la prima misurazione si procede in manie-

```
void initialize() {
    Sensor_Struct_Init(&sensor);
    setTimeout(500, &sensor);
    Sensor_Init(&sensor);
    setDistanceMode(SHORT, &sensor);
    setMeasurementTimingBudget(20000, &sensor);
    startContinuous(30, &sensor);
    temporization = 50; //TimingBudget + InterMeasurement = 50
}
```

**Figura 5.1:** Inizializzazione iniziale del sensore.

ra continua alla lettura. Come già spiegato il sensore può lavorare in diverse modalità di distanza in base al range in cui ci troviamo. Perciò effettuiamo un passaggio da una modalità all'altra in maniera automatica attraverso il confronto della distanza letta con i valori che definiscono gli intervalli di ciascuna DistanceMode, attraverso la funzione di *ChangeRange()*. Inoltre, nel caso che da una lettura all'altra la modalità non cambi, non vengono rieseguite tutte le istruzioni di settaggio ma viene semplicemente riletto il

valore dell'altezza a cui ci troviamo. Il passaggio in maniera automatica da una modalità all'altra permette di poter leggere sempre in maniera piuttosto precisa la giusta altezza, considerando comunque che l'errore aumenta in maniera proporzionale alla distanza misurata. Un altro aspetto molto importante da considerare è la temporizzazione. Ogni misura ha bisogno di un tempo ben preciso in base alla modalità che si sta usando. Precisamente i tempi da impostare sono due, ovvero il *Timing Budget* e *Inter Measurement*. Il primo indica il tempo fornito al sensore per effettuare la lettura, e tale tempo è maggiore tanto più grande è la distanza da leggere. Viene quindi settato in maniera diversa in ogni modalità, con valori che sono 20 ms in Short, 50 ms in Medium e Long, e 140 ms in LongLong. *Inter Measurement* indica invece il tempo tra l'inizio di due letture, e deve essere almeno 4 ms più grande del *Timing Budget*, perciò viene posto 10 ms più grande di quest'ultimo in ciascuna modalità. La funzione *Read*, ovvero quella che effettua la lettura della distanza, attende che il sensore abbia effettuato la misura e sia pronto a fornire tale valore, e ciò viene realizzato attraverso il controllo del risultato della funzione *dataReady*, che verifica in maniera continua il valore del registro `GPIO_TIO_HV_STATUS`, tenendo occupata la periferica I<sup>2</sup>C, utilizzata anche per l'IMU. Per risolvere il problema viene perciò gestita la lettura attraverso un timer, che genera un flag, utilizzato come abilitazione, con un tempo variabile a seconda della modalità (temporization), che è pari al tempo di *Inter\_Measurement*, tenendo così occupata la periferica solo per la lettura di alcuni registri utilizzati per il calcolo della distanza con un periodo di campionamento ben preciso. La relazione dei tempi è mostrata in Figura 5.2.

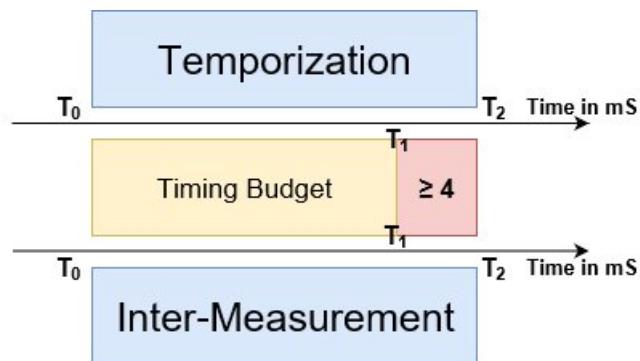
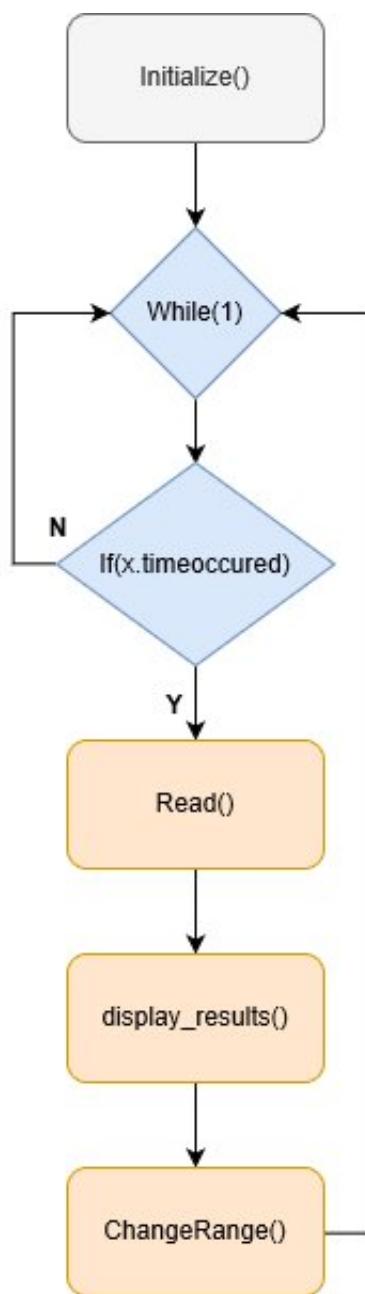
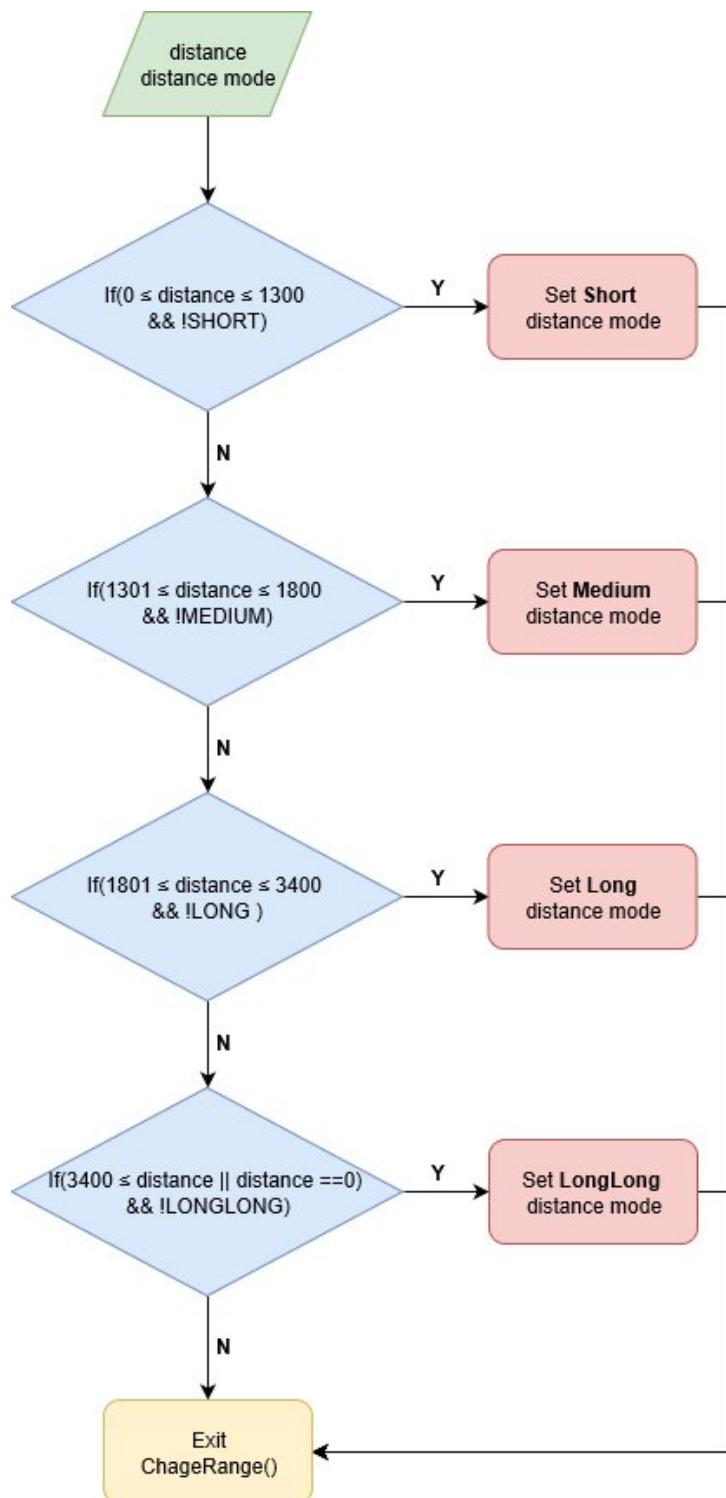


Figura 5.2



**Figura 5.3:** Diagramma di flusso del Main.



**Figura 5.4:** Diagramma di flusso della funzione *ChangeRange*.

# Capitolo 6

## Conclusioni

L'obiettivo del progetto sviluppato è stato quello di gestire un nuovo sensore di distanza, da sostituire al sonar precedentemente impiegato, per misurare l'altezza rispetto al terreno del Ducted Fan necessaria per controllare il volo del drone. Tale obiettivo è stato raggiunto dopo aver affrontato e risolto le diverse problematiche descritte nei paragrafi precedenti e riguardanti l'adattamento della libreria Arduino del VL53L1X. Sotto questo punto di vista il lavoro svolto può essere considerato completo in quanto la libreria adattata risulta adesso utilizzabile e funzionante con la scheda Renesas. Sono però necessarie alcune considerazioni conclusive sul progetto svolto. In primo luogo la libreria Arduino adattata è stata solo parzialmente commentata in quanto è priva di un'adeguata documentazione (vengono fornite informazioni solo sui metodi pubblici in essa implementati) e non si dispone neppure di una documentazione a basso livello del sensore che ne descriva i registri interni e la loro configurazione. Per queste ragioni è stato possibile comprendere approfonditamente e quindi commentare solo i metodi pubblici, che sono stati utilizzati poi all'interno del programma, e i metodi privati, che è stato necessario modificare. Le restanti funzioni sono state semplicemente adattate formalmente (porting C++/C) e utilizzate indirettamente. Altra considerazione da fare riguarda l'esistenza di un'altra libreria per la gestione del sensore. La libreria Arduino modificata è infatti una semplificazione specifica per Arduino dell'API vera e propria fornita dai produttori. Data la complessità di quest'ultima si è optato per l'utilizzo di una versione semplificata ma con meno funzionalità rispetto all'originale. La libreria adattata ad esempio non implementa le funzioni per la modifica della ROI e per la calibrazione del dispositivo e attua controlli sugli errori meno stringenti. Un possibile sviluppo futuro del progetto può quindi riguardare l'adattamento dell'API per poter sfruttare appieno le funzionalità del sensore. Infine è necessaria un'importante osservazione sui test effettuati e quindi sull'utilizzo della distanza rilevata per il controllo del drone. Le prove effettuate hanno messo in evidenza che il sensore fornisce misure sempre accurate ma non

molto precise: misure successive della medesima distanza e nelle medesime condizioni forniscono valori piuttosto diversi pur avendo un valor medio molto vicino alla distanza reale. Questo problema è stato riscontrato durante i test realizzati e, pur non essendo stato valutato dettagliatamente, è stato preso in considerazione nei risultati in quanto i valori riportati sono stati ottenuti effettuando una media tra 5 misurazioni successive (i test descritti nel datasheet del sensore utilizzano invece ben 23 misurazioni successive). Questo aspetto non è affrontato esplicitamente nel programma realizzato che si limita ad effettuare singole misure ad intervalli regolari che dipendono dalla Ranging Option utilizzata. Considerando però la rilevanza di questo problema anche in condizioni di misura relativamente favorevoli, almeno rispetto a quelle in cui si troverà il sensore sul drone, tale aspetto dovrà sicuramente essere considerato all'interno del progetto del Ducted Fan. Si dovrà quindi effettuare una media tra misurazioni successive, compatibilmente con i tempi necessari per la singola misura e con quelli richiesti per il controllo del drone.

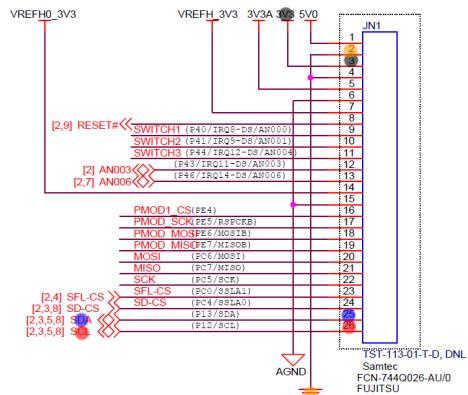
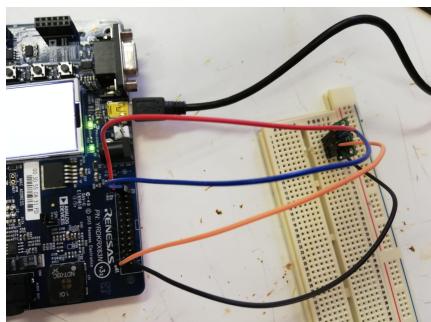
# **Appendici**

## Appendice A

# Collegamento Scheda-Sensore

Per poter passare al test abbiamo collegato il sensore alla scheda in questo modo (FiguraA.1):

- Pin V<sub>in</sub> → pin n°3 del settore JN1 (Pin collegato ad alimentazione a 3,3V).
- Pin GND → Pin n°2 del settore JN1 (Pin collegato a massa).
- Pin SDA → Pin n°25 del settore JN1 (Pin che gestisce l'SDA).
- Pin SCL → Pin n°26 del settore JN1 (Pin che gestisce l'SCL).



**Figura A.1:** A destra con gli stessi colori dei cavetti è evidenziato il pinout preso dallo Schematics della scheda.

# Appendice B

## Correzione errori

### B.1 I<sup>2</sup>C

Un errore frequente è quello di utilizzare un indirizzo errato. Gli indirizzi degli slave sono tutti a 7 bit. La nostra funzione necessita un indirizzo a 8 bit, dove l'ottavo bit (LSB) indica lo stato di scrittura/lettura. Da manuale il valore dell'indirizzo è 0x52 poichè si presuppone che LSB sia 0. Alla funzione forniamo invece 0x29 poichè è il valore dei 7 bit più significativi, con l'MSB posto a 0. Infatti sarà la funzione I<sup>2</sup>C fornитaci ad effettuare uno shift a sinistra per riottenere 0x52.

E' possibile che durante il funzionamento della scheda in modalità Debug (e non) si verifichino errori di lettura/scrittura mediante I<sup>2</sup>C. In tal caso conviene effettuare una pulizia del progetto, riavviando anche la scheda.

### B.2 Ottimizzazione

Un altro fondamentale problema è alla base dell'ambiente di sviluppo. Esso ammette la funzione di ottimizzazione automatica del codice su vari livelli. Questa ottimizzazione taglia il contenuto della struttura dati sensore non permettendo il corretto aggiornamento dei dati in essa. E' possibile risolvere il problema disattivando l'ottimizzazione o utilizzando una struttura di tipo extern. Nel nostro è stata adottata la prima soluzione.