Scanner Lidar

Documentazione

Sommario

[1 Introduzione 4](#_Toc72402180)

[1.1 Informazioni sul progetto 4](#_Toc72402181)

[1.2 Abstract 4](#_Toc72402182)

[1.3 Scopo 4](#_Toc72402183)

[2 Analisi 5](#_Toc72402184)

[2.1 Analisi del dominio 5](#_Toc72402185)

[2.2 Analisi e specifica dei requisiti 5](#_Toc72402186)

[2.3 Use case 7](#_Toc72402187)

[2.4 Pianificazione 8](#_Toc72402188)

[2.5 Analisi dei mezzi 9](#_Toc72402189)

[2.5.1 Software 9](#_Toc72402190)

[2.5.1.1 Tools 9](#_Toc72402191)

[2.5.1.2 Librerie 9](#_Toc72402192)

[2.5.1.2.1 Arduino 9](#_Toc72402193)

[2.5.1.2.2 Server 9](#_Toc72402194)

[2.5.2 Hardware 10](#_Toc72402195)

[2.5.2.1 Arduino e le sue componenti 10](#_Toc72402196)

[2.5.2.2 Macchina Virtuale 10](#_Toc72402197)

[2.5.2.3 Macchina fisica scolastica 10](#_Toc72402198)

[2.5.2.4 Portatile 10](#_Toc72402199)

[3 Progettazione 11](#_Toc72402200)

[3.1 Design dell’architettura del sistema 11](#_Toc72402201)

[3.2 Design dei dati 11](#_Toc72402202)

[3.3 Design delle interfacce 12](#_Toc72402203)

[3.4 Design procedurale 12](#_Toc72402204)

[3.4.1 Arduino 12](#_Toc72402205)

[3.5 Design Hardware 13](#_Toc72402206)

[4 Implementazione 15](#_Toc72402207)

[4.1 Server 15](#_Toc72402208)

[4.1.1 Comunicazione con Arduino tramite seriale 15](#_Toc72402209)

[4.1.2 Rilevamento porta seriale 15](#_Toc72402210)

[4.1.3 Gestione richieste 16](#_Toc72402211)

[4.1.3.1 MultiThreading 16](#_Toc72402212)

[4.1.4 Inizializzazione dati configurabili 17](#_Toc72402213)

[4.1.4.1 Configurazione porta 17](#_Toc72402214)

[4.1.5 Creazione vettori 17](#_Toc72402215)

[4.1.6 Invio Dati al client 18](#_Toc72402216)

[4.1.7 Log 19](#_Toc72402217)

[4.1.7.1 Struttura del messaggio 20](#_Toc72402218)

[4.1.7.2 Struttura delle cartelle 20](#_Toc72402219)

[4.2 Client 21](#_Toc72402220)

[4.2.1 Connessione al server 21](#_Toc72402221)

[4.2.2 Lettura dei messaggi dal server 22](#_Toc72402222)

[4.2.3 Invio dei messaggi al server 23](#_Toc72402223)

[4.2.4 Render dei punti 23](#_Toc72402224)

[4.2.5 Aggiunta degli assi nello spazio 3D 24](#_Toc72402225)

[4.2.6 Movimento della camera 24](#_Toc72402226)

[4.2.7 Movimento dello spettatore 25](#_Toc72402227)

[4.2.8 Interfaccia grafica 26](#_Toc72402228)

[4.3 Arduino 28](#_Toc72402229)

[4.3.1 Angolo di rotazione degli stepper motor 28](#_Toc72402230)

[4.3.2 Attesa comunicazione con server 30](#_Toc72402231)

[4.3.3 Movimento motore asse orizzontale 30](#_Toc72402232)

[4.3.4 Movimento motore asse verticale 32](#_Toc72402233)

[4.3.5 Invio dato 32](#_Toc72402234)

[4.3.6 Fine esecuzione e reset motori 32](#_Toc72402235)

[4.3.7 Struttura HW 34](#_Toc72402236)

[4.3.8 Corretta posizione dei cavi 37](#_Toc72402237)

[5 Test 38](#_Toc72402238)

[5.1 Protocollo di test 38](#_Toc72402239)

[5.2 Risultati test 40](#_Toc72402240)

[6 Consuntivo 41](#_Toc72402241)

[7 Conclusioni 42](#_Toc72402242)

[7.1 Sviluppi futuri 42](#_Toc72402243)

[7.2 Considerazioni personali 42](#_Toc72402244)

[8 Bibliografia 43](#_Toc72402245)

[8.1 Sitografia 43](#_Toc72402246)

[9 Allegati 43](#_Toc72402247)

Indice delle figure

[Figura 1 - Use Case 7](#_Toc72401720)

[Figura 2 - Gantt preventivo 8](#_Toc72401721)

[Figura 3 - Architettura sistema 11](#_Toc72401722)

[Figura 4 - Design dell'interfaccia SW 12](#_Toc72401723)

[Figura 5 – Arduino, design procedurale 12](#_Toc72401724)

[Figura 6 – Arduino, struttura HW 13](#_Toc72401725)

[Figura 7 – Arduino, schema elettrico 14](#_Toc72401726)

[Figura 8 - Schema di rete 15](#_Toc72401727)

[Figura 9 - Dimostrazione calcoli 18](#_Toc72401728)

[Figura 10 - Client, interfaccia di connessione 27](#_Toc72401729)

[Figura 11 - Client, esempio di errore nel Label 27](#_Toc72401730)

[Figura 12 - Client, interfaccia principale 28](#_Toc72401731)

[Figura 13 - Calcolo dell'interlacciato 31](#_Toc72401732)

[Figura 14 - Client, griglia di scansione 31](#_Toc72401733)

[Figura 15 - Disegno struttura HW, 1 34](#_Toc72401734)

[Figura 16 - Disegno struttura HW, 2 34](#_Toc72401735)

[Figura 17 - Disegno struttura HW, 3 35](#_Toc72401736)

[Figura 18 - Sostegno Lidar e sostegno motore verticale 35](#_Toc72401737)

[Figura 19 - Base HW 36](#_Toc72401738)

[Figura 20 - Base rotazione HW 36](#_Toc72401739)

[Figura 21 - Posizionamento cavi, 1 37](#_Toc72401740)

[Figura 22 - Posizionamento cavi, 2 37](#_Toc72401741)

[Figura 23 - Gantt consuntivo 41](#_Toc72401742)

# Introduzione

## Informazioni sul progetto

* Mandante: Geo Petrini.
* Partecipanti: Daniele Cereghetti, Isaac Gragasin, Veljko Markovic, Matteo Lupica.
* Tempo a disposizione: 14 gennaio 2021 - 20 maggio 2021
* Classe: I3AA-AC
* Progetto: Scanner Lidar

## Abstract

*Up until now, man strived to find various methods to acquire and analyse knowledge on many things, adapting and optimizing such procedures to be more efficient and effective as technological progress permitted so. This allowed the development of specialized scanning systems, fundamental for many professions, from modern doctors to simple shop clerks. Our project revolves around the use of a structure with a built-in LIDAR scanner and focuses on the extraction of data on surfaces in order to produce a graphic depiction of the current surroundings. A useful tool for jobs that are associated to architecture or that work heavily with planes, coordinates and metrics.*

## Scopo

Lo scopo del nostro progetto è quello di realizzare uno scanner LIDAR, utilizzando una combinazione tra una piattaforma Arduino e le capacità computazionali di un PC. Il quale dovrà visualizzare a schermo tutti i punti trovati dalla piattaforma.

# Analisi

## Analisi del dominio

È stato richiesto dal mandante di realizzare uno scanner LIDAR, utilizzando una combinazione tra una piattaforma Arduino e le capacità computazionali di un PC. Il prodotto sarà composto da due parti: una di Arduino, che si occupa di gestire l’aspetto meccanico e dell’inoltro dei dati al software; una, come accennato poco fa, di software, che si occupa invece di manipolare i dati ricevuti dalla parte fisica e di rappresentarli in uno spazio 3D. Il prodotto potrà girare solamente su sistemi operativi Windows 10 con architettura a 32 o 64 bit.

Potenzialmente, tutti potrebbero essere gli utenti di questo prodotto, ma è principalmente pensato per ambiti di architettura e misurazione. Le capacità minime richieste da parte degli utenti sono quelle di collegare l’Arduino ad un PC e di far partire il software.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-HW-1** | |
| **Nome** | Scansione a 360° sull’asse orizzontale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-HW-2** | |
| **Nome** | Scansione a 90° sull’asse verticale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-HW-3** | |
| **Nome** | Minor delay possibile tra scansione LIDAR e rotazione motore |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-HW-4** | |
| **Nome** | Invio dati al PC |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Inviare l’angolo della rotazione orizzontale, verticale e la distanza rilevata |

|  |  |
| --- | --- |
| **ID: REQ-SW-1** | |
| **Nome** | Acquisire i dati da Arduino tramite USB |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-SW-2** | |
| **Nome** | Interpretare i dati in ambiente 3D |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Trasformare i valori ottenuti dalla scansione in vettori |

|  |  |
| --- | --- |
| **ID: REQ-SW-3** | |
| **Nome** | Creazione dell’applicativo per un solo sistema operativo |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Sistema operativo scelto: Windows |

|  |  |
| --- | --- |
| **ID: REQ-SW-4** | |
| **Nome** | L’applicativo deve essere un file eseguibile |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** |  |

|  |  |
| --- | --- |
| **ID: REQ-SW-5** | |
| **Nome** | Linguaggio di programmazione da utilizzare |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Linguaggio scelto: C# |

## Use case

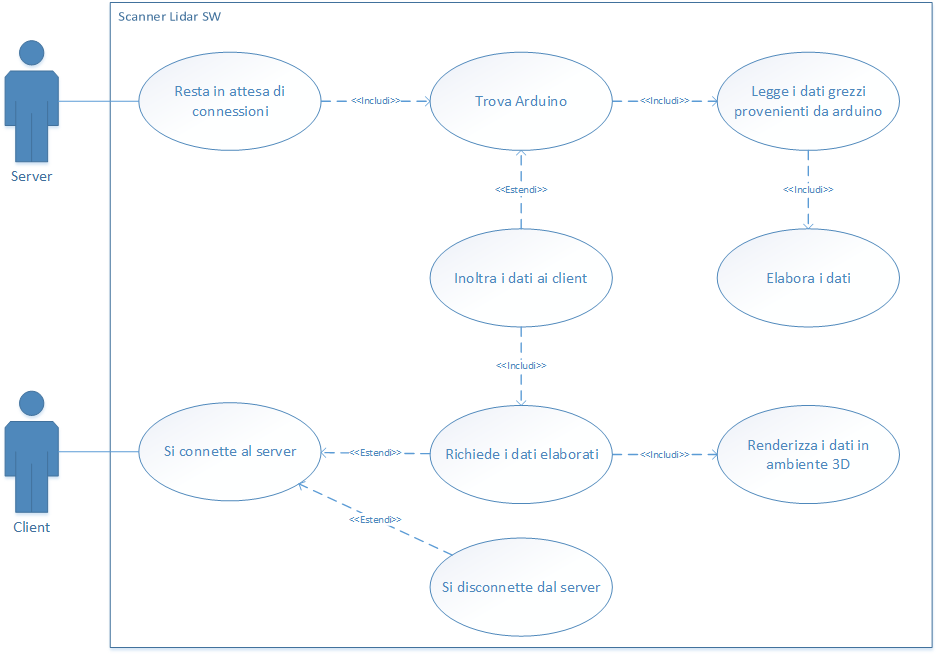


Figura - Use Case

## Pianificazione

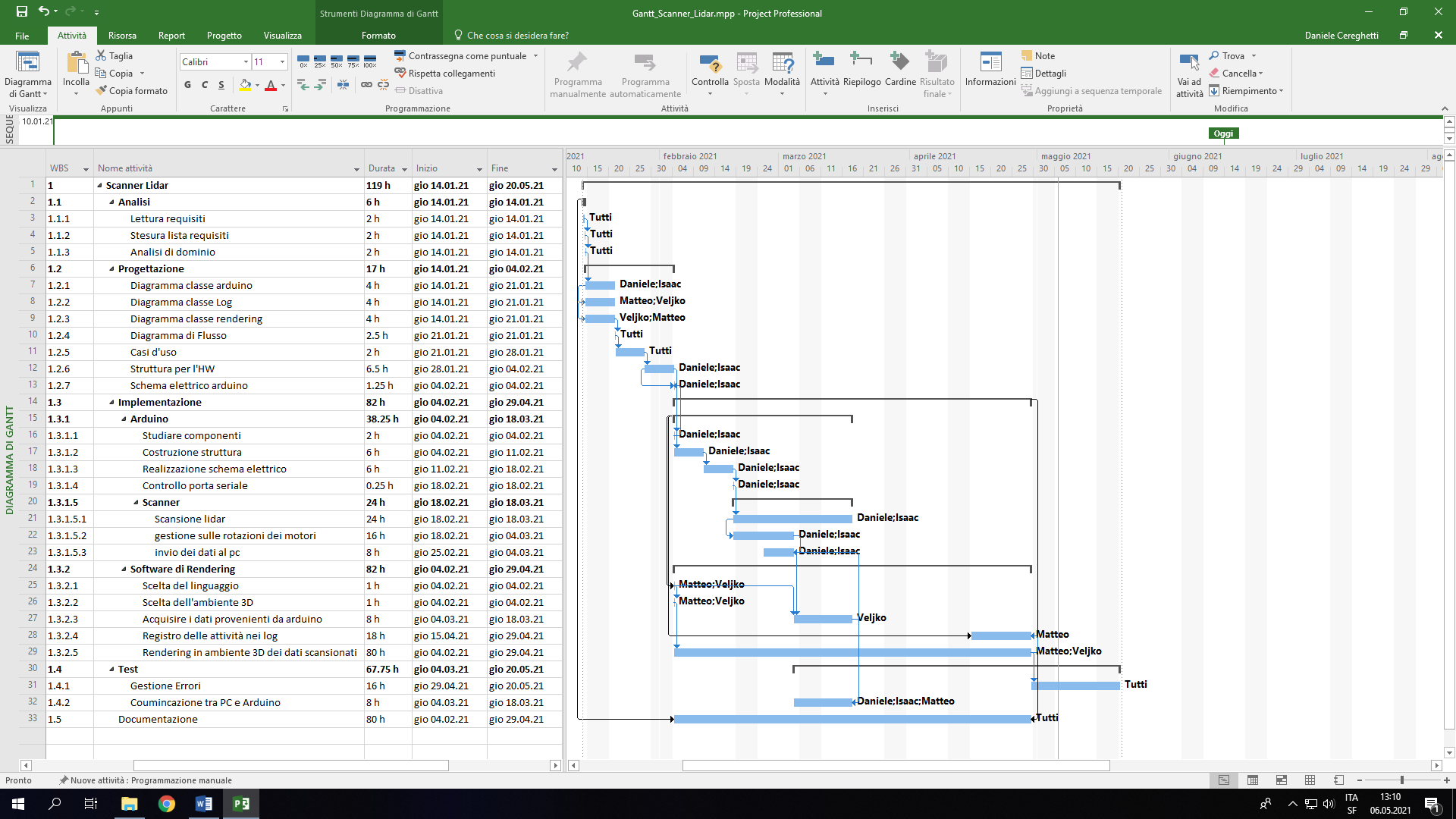


Figura - Gantt preventivo

## Analisi dei mezzi

### Software

### Tools

Per la realizzazione del server è stato utilizzato:

* Visual Studio 2019 (16.7.2)

Per la realizzazione del client sono stati utilizzati:

* Visual Studio 2019 (16.8.6)
* Unity (2020.2.3.f1 Personal <DX11>)

Per la realizzazione del codice di Arduino è stato utilizzato:

* Arduino IDE (1.8.13)

Per la realizzazione della struttura dello scanner Lidar è stato utilizzato:

* LeoCAD (21.01)
* Easel (WebApp)

Per la realizzazione dei diagrammi e use case sono stati utilizzati:

* Draw IO (WebApp)
* Microsoft Visio 2016

Per la realizzazione dello schema di rete è stato usato:

* Frizing

Per la realizzazione del Gantt è stato utilizzato:

* Microsoft Project 2016

Per la realizzazione della documentazione è stato utilizzato:

* Microsoft Word 2016

Per la creazione e l’utilizzo della macchina virtuale è stato utilizzato:

* Oracle VM VirtualBox (6.1.16 r140961 (Qt5.6.2))

### Librerie

### Arduino

Le librerie elencate di seguito sono state utilizzate per la realizzazione della parte di Arduino e sono disponibili sul nostro repository di GitHub:

* Arduino JSON (6.17.3)
* TFmini Arduino (0.1.1)

### Server

Le librerie utilizzate per la realizzazione del Server sono:

* Microsoft.EntityFrameworkCore (3.1.12)
* Microsoft.EntityFrameworkCore.Sqlite (3.1.12)
* Microsoft.EntityFrameworkCore.Tools (3.1.12)
* Microsoft.Extensions.Hosting (3.1.12)
* NLog (4.7.7)
* NLog.Config (4.7.7)
* System.Configuration.ConfigurationManager (5.0.0)
* System.IO.Ports (5.0.0)
* System.Management (5.0.0)

### Hardware

Il prodotto dovrà essere eseguito su una macchina Windows 10.

### Arduino e le sue componenti

Per la realizzazione della struttura fisica dello scanner Lidar, sono stati utilizzati:

* Arduino Mega 2560
* TFmini-S LiDAR Module
* Due STEP MOTOR 28BYJ-48 5VDC
* Driver ULN2003

### Macchina Virtuale

Per problemi di proxy, non abbiamo potuto installare Unity sui PC fisici di scuola. Pertanto, abbiamo optato di installare e utilizzare Unity su una macchina virtuale con le seguenti specifiche di sistema:

* Sistema operativo: Windows 10 Home (1909) con architettura a 64-bit
* Processore: 2 Core
* RAM: 8 GB
* Memoria: 50 GB

Per quanto concerne le impostazioni della macchina virtuale, sono le seguenti:

* Scheda di rete: NAT
* Audio:
  + Uscita: Sì
  + Entrata: No
* USB: Abilitato

### Macchina fisica scolastica

I PC che abbiamo utilizzato e che avevamo a disposizione a scuola hanno le seguenti specifiche:

* Sistema operativo: Windows 10 Education (1803) con architettura a 64-bit
* Processore: Intel® Core™ i7-7700 CPU @ 3.60 GHz
* RAM: 16 GB
* Memoria: 460 GB

### Portatile

Per la realizzazione della parte di Arduino, abbiamo utilizzato un portatile MacBook Pro (Retina, 15-inch, Mid 2015) con le seguenti specifiche:

* Sistema operativo: macOS Big Sur (11.0.1)
* Processore: Intel® Core™ i7 CPU @ 2.50 GHz Quad-Core
* RAM: 16 GB
* Memoria: 250 GB

# Progettazione

## Design dell’architettura del sistema

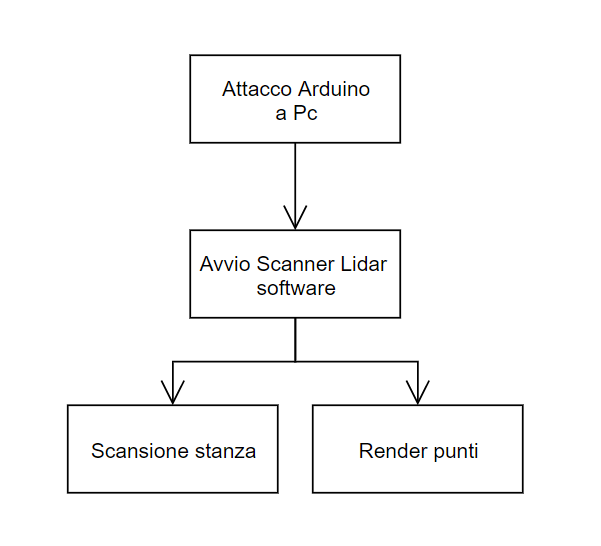


Figura - Architettura sistema

Quando si vuole effettuare una scansione di una stanza, come prima cosa bisogna attaccare l’Arduino al PC, successivamente avviare il software dedicato, il quale in contemporanea ordinerà di fare la scansione e il rendere dei punti ricevuti dalla parte HW.

## Design dei dati

Per rappresentare i punti su schermo ci siamo chiesti per prima cosa, che cosa ci poteva fornire la parte di Arduino e decidere quale di queste ci avrebbero aiutati. Perciò abbiamo deciso che la parte hardware deve mandare alla parte software l’angolo della rotazione orizzontale, l’angolo della rotazione verticale e la distanza rilevata dallo scanner Lidar. Lo scanner Lidar quando rileva una distanza, la rileva facendo un angolo di 2 gradi, perciò il dato che riceveremmo sarà una media di tutti i dati presi in quel momento.

Il protocollo per passare i dati tra le due parti è simile a quello del CSV, ovvero che passiamo 3 valori separati da una virgola: il primo è la rotazione orizzontale, il secondo è la rotazione verticale e il terzo è la distanza presa.

## Design delle interfacce

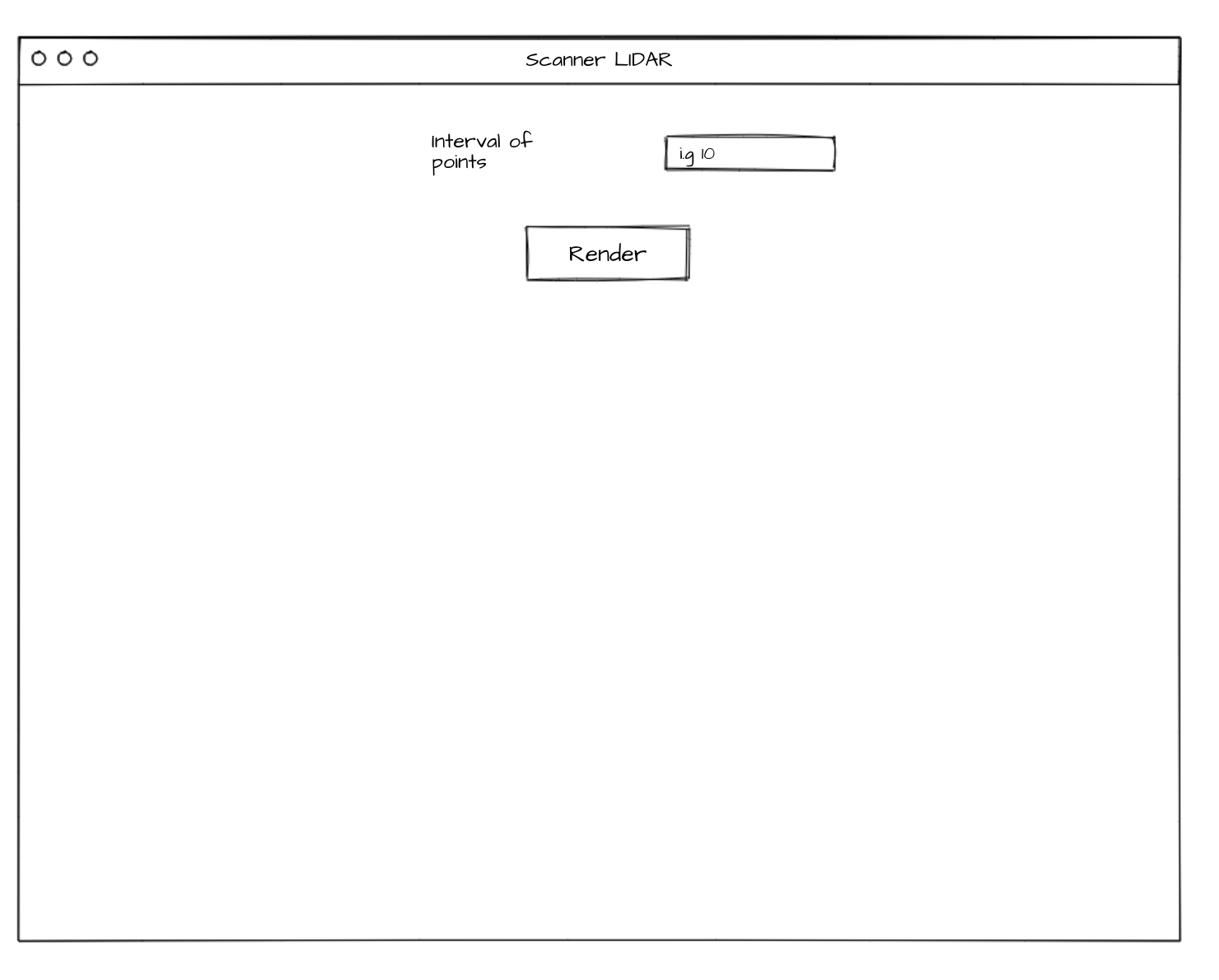


Figura - Design dell'interfaccia SW

## Design procedurale

## Arduino

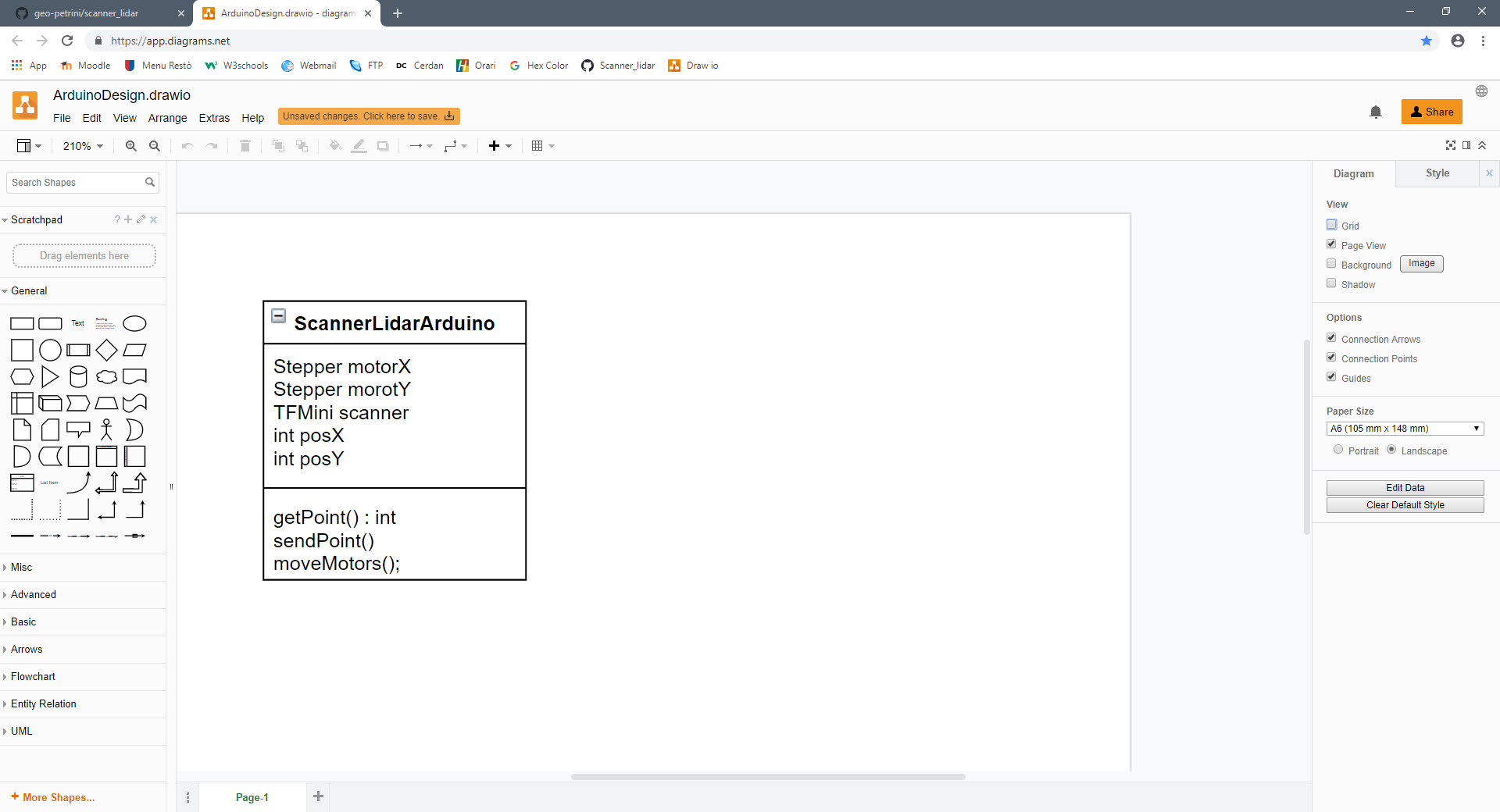


Figura – Arduino, design procedurale

## Design Hardware

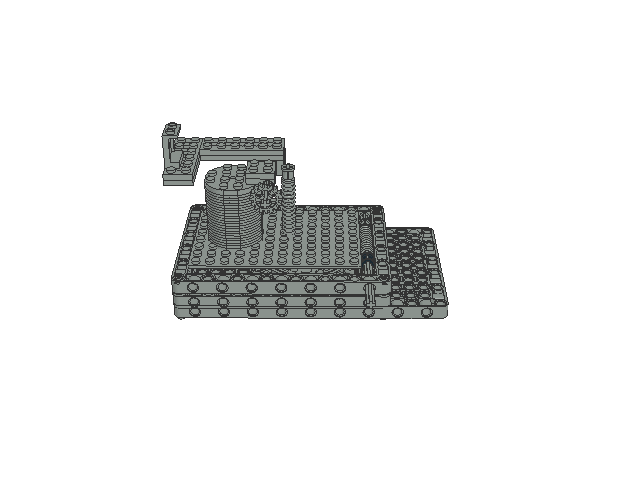


Figura – Arduino, struttura HW

Questa è la struttura sulla quale si appoggerà lo scanner Lidar per fare la scansione, ed è realizzata con pezzi Lego e Lego Tecnic. Per fare una rotazione orizzontale attacchiamo un motore al bastoncino (cerchio blu) che esce lateralmente, la quale muove una rotella, sulla quale è attaccata il pannello che si deve ruotare. Per fa eseguire la rotazione verticale, posizioniamo l’altro motore all’altro bastoncino (cerchio rosso), il quale tramite una serie di ingranaggi muove il scanner Lidar verso l’alto e verso il basso (cerchio giallo).

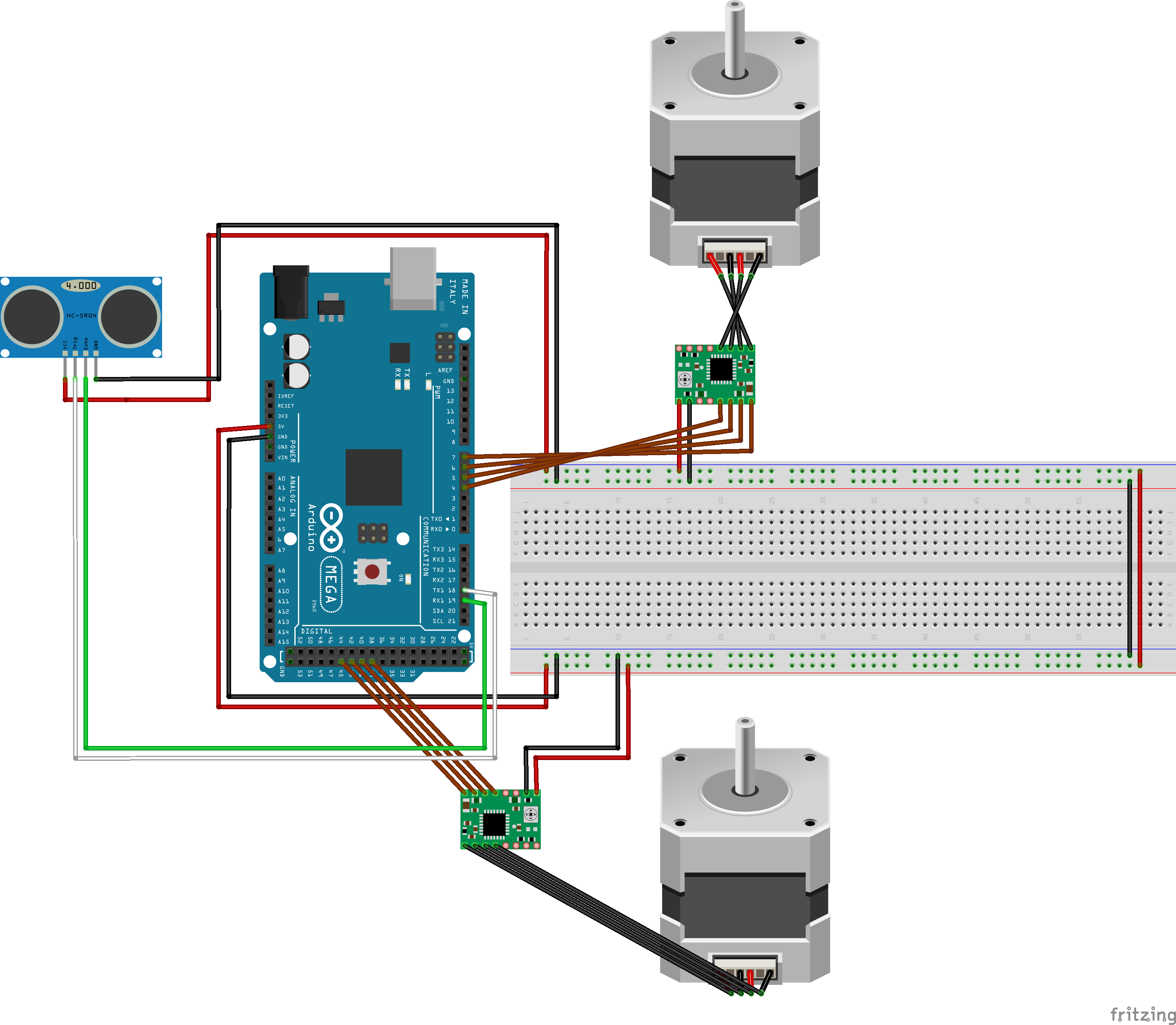


Figura – Arduino, schema elettrico

Questo è lo schema elettrico della parte di Arduino, infatti sono presenti i due motori che permettono alla struttura di ruotare, e lo scanner Lidar. I motori utilizzati hanno 4 pin, il che significa che sono in grado di girare in senso anti orario e orario, altrimenti sarebbero stati da due pin.

In questo schema è presente anche la breadboard, ma nella realtà colei non sarà presente per motivi di spazio.

# Implementazione

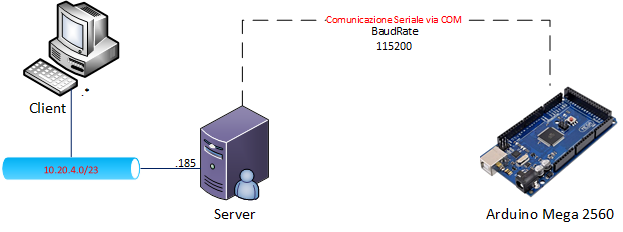


Figura - Schema di rete

La nostra implementazione si sviluppa tramite un Arduino che scansiona e invia i dati grezzi al server centrale che gli elabora e successivamente gli fornisce, sotto forma di componenti vettoriali, ai client.

## Server

Il ruolo del server all’interno del progetto, a parte essere un punto centrale, è quello di elaborare i dati provenienti da Arduino, gestire il log, gestire i client e distribuirgli i dati necessari.

## Comunicazione con Arduino tramite seriale

Per poter leggere i dati provenienti sulla seriale abbiamo bisogno di sapere su quale porta comunica Arduino. Dopo che sappiamo su che porta comunica, creiamo un buffer grande quanto i dati disponibili sulla seriale e poi convertiamo e leggiamo il suo valore.

byte[] buffer = new byte[arduino.BytesToRead];

string reader = "";

for (int i = 0; i < buffer.Length; i++){

reader += (char)arduino.ReadByte();

}

## Rilevamento porta seriale

Per poter rilevare la porta COM ho dovuto cercare nella documentazione se ci fosse un modo per poter ottenere tutte le porte seriali attualmente utilizzate, infatti nella documentazione ho trovato il metodo **.GetPortNames()** che ritorna un array di stringhe contenente il nome delle porte seriali.

string[] availblePort = SerialPort.GetPortNames();

List<string> ports = availblePort.OfType<string>().ToList();

List<string> disponibili = new List<string>();

Dopo aver ottenuto tutte le porte disponibili, andiamo ad aprire una connessione con esse, leggendo successivamente dallo stream. Se il contenuto della lettura dello stream, a cui è collegato l’Arduino, rispetta il protocollo di comunicazione, teniamo traccia di quale porta è stata appena effettuato il controllo.

**while** **(true** **&&** **!**found**)**

**{**

**try**

**{**

serial**.**PortName **=** ports**[**cnt **<** ports**.**Count **-** 1 **?** cnt**++** **:** cnt**];**

myLogger**.**Debug**(**"Porta attuale: " **+** serial**.**PortName**);**

serial**.**Open**();**

Thread**.**Sleep**(**500**);**

**if** **(**serial**.**BytesToRead **>** 0**)**

**{**

string reader **=** serial**.**ReadExisting**();**

**if** **(**reader**.**Contains**(**"CIAO"**))**

**{**

disponibili**.**Add**(**serial**.**PortName**);**

myLogger**.**Debug**(**"SONO ARDUINO"**);**

found **=** **true;**

**}**

**}**

**else**

**{**

serial**.**Close**();**

**}**

**}**

**catch** **(**Exception ioe**)**

**{**

**...**

**}**

**}**

## Gestione richieste

Siccome il nostro server centrale deve poter fornire sempre il servizio, esso deve rimanere in attesa di una qualsiasi connessione in arrivo.

**while** **(true)**

**{**

TcpClient connectedClient **=** tcpListener**.**AcceptTcpClient**();**

Thread t **=** **new** Thread**(new** ParameterizedThreadStart**(**ServerHandler**));**

t**.**Start**(**connectedClient**);**

**}**

## MultiThreading

Nel nostro caso il server può fornire il servizio a più dispositivi contemporaneamente, quindi utilizziamo una Thread per ogni client che si connette.

...

**while** **((**length **=** stream**.**Read**(**bytes**,** 0**,** bytes**.**Length**))** **!=** 0**)**

**{**

/\*

\* FORMATO --> Int:0,10

\* Int:{0},{1}

\* 0 -> dati di partenza

\* 1 -> quanti dati ricevere (\* => tutti i dati disponibili)

\*/

**...**

string[] message = clientMessage.Split(':');

string header = message[0];

...

**if** **(**canSend**)**

**{**

**object** args **=** **new** **object[**5**]** **{** myLogger**,** client**,** vector3s**,** min**,** max **};**

**(new** Thread**(new** ParameterizedThreadStart**(**mT**.**SendToUnity**))).**Start**(**args**);**

**}**

**...**

**}**

## Inizializzazione dati configurabili

Siccome il nostro server può essere ospitato su qualsiasi macchina, esso prende i dati dal file di configurazione (**Server\_Lidar.dll.config**) e se i valori sono accettati li imposta. **Attenzione:** bisogna modificare solo il valore della porta del server.

<?xml version=**"1.0"** encoding=**"utf-8"** ?>

<configuration>

<appSettings>

<add key=**"BaudRate"** value=**"115200"** />

<add key=**"ServerPort"** value=**"12345"** />

</appSettings>

</configuration>

## Configurazione porta

Siccome i sistemi operativi hanno un numero massimo di porte e delle porte riservate (soprattutto ambienti OSx e Lx) andiamo a effettuare un controllo se la porta immessa dall’utente sia nel range [1024, 65535] con gli estremi compresi.

**public** int ServerPort

**{**

**get** => serverPort;

**set**

**{**

**if(value** **>=** 1024 **&&** **value** **<=** 65535**)**

**{**

serverPort **=** **value;**

**}**

**else**

**{**

myLogger**.**Warn**(**"The port value in: Server\_Lidar.dll.config is not in the range of [1024,65535], default settings loaded -> port = 12345"**);**

serverPort **=** 12345**;**

**}**

**}**

**}**

## Creazione vettori

Visto che il nostro server, ottiene dei dati provenienti da Arduino (angolo orizzontale, verticale e lunghezza misurata) riguardo una scansione di un punto nello spazio, noi andiamo a creare il vettore che rappresenta quel punto.

X **=** **(**float**)(**intensity **\*** Math**.**Round**(**Math**.**Cos**(**Math**.**PI **\*** vertical **/** 180.0**),** 6**)** **\*** Math**.**Round**(**Math**.**Cos**(**Math**.**PI **\*** horizontal **/** 180.0**),**6**)),**

Y **=** **(**float**)(**intensity **\*** Math**.**Round**(**Math**.**Cos**(**Math**.**PI **\*** vertical **/** 180.0**),**6**)** **\*** Math**.**Round**(**Math**.**Sin**(**Math**.**PI **\*** horizontal **/** 180.0**),**6**)),**

Z **=** **(**float**)(**intensity **\*** Math**.**Round**(**Math**.**Sin**(**Math**.**PI **\*** vertical **/** 180.0**),** 6**)),**

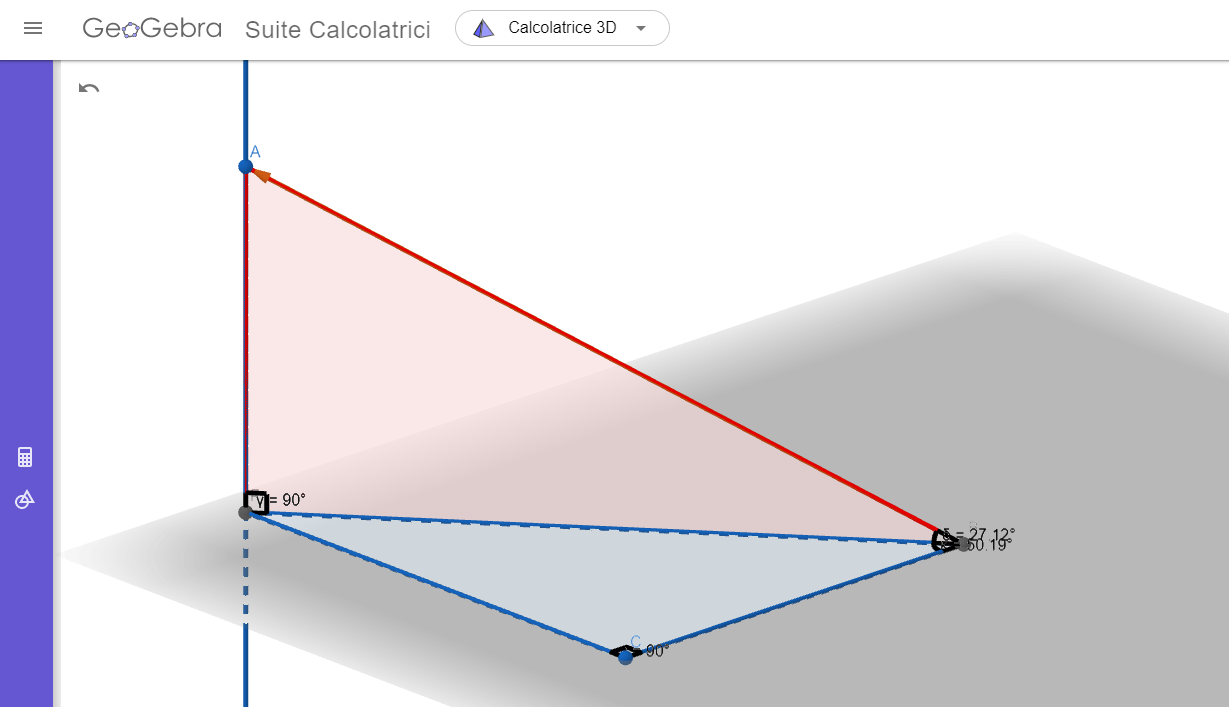


Figura - Dimostrazione calcoli

Per effettuare i calcoli per riuscire a trovare le componenti del vettore, abbiamo utilizzato GeoGebra, così da poter visualizzare il nostro punto nello spazio e successivamente calcolarne le coordinate.

## Invio Dati al client

Per l’invio dei dati ai client, si controlla se esso sia connesso

if (connectedClient == null)

{

return;

}

if (connectedClient.Connected)

{

try

{

// ottengo l'oggetto stream per scrivere.

NetworkStream stream = connectedClient.GetStream();

if (stream.CanWrite)

{

// Converto la stringa in un array di byte.

byte[] serverMessageAsByteArray = Encoding.ASCII.GetBytes(serverMesage);

// Scrivo l'array di byte sul socketConnection stream.

stream.Write(serverMessageAsByteArray, 0, serverMessageAsByteArray.Length);

}

else

{

stream.Close();

}

}

catch (Exception e)

{

IsSending = false;

}

}

## Log

Il log è una parte fondamentale per qualsiasi progetto, così da poter tenere traccia di ciò che avviene e poter controllare più facilmente gli eventuali errori. Nel nostro caso abbiamo utilizzato la libreria di **NLOG**, il quale si basa su un file di configurazione (json o xml).

<!--Modificare solo i valori nelle variabili-->

<variable name=**"file\_size"** value=**"500000"**/>

<variable name=**"logFile\_minLevel"** value=**"Debug"**/>

<variable name=**"logConsole\_minLevel"** value=**"Info"**/>

<variable name=**"file\_name\_path"**

value=**"${basedir}/logs/${shortdate}/${level}/${level}.log"**/>

<variable name=**"layout\_format\_file"** value=**"[${date:format=yyyy-MM-dd HH\:mm\:ss}] |**

**${uppercase:${level}} | ${message} | ${callsite} At line:${callsite-linenumber} "**/>

<variable name=**"layout\_format\_console"** value=**"[${date:format=yyyy-MM-dd HH\:mm\:ss}] | ${uppercase:${level}} | ${message}"**/>

<variable name=**"archive\_file\_name"**

value=**"${basedir}/logs/${shortdate}/${level}/${level}-{#}.log"**/>

<!--Non modificare-->

<targets>

<!--Log for File-->

<target xsi:type=**"File"**

name=**"logFile"**

fileName=**"${file\_name\_path}"**

layout=**"${layout\_format\_file}"**

createDirs=**"true"**

archiveFileName=**"${archive\_file\_name}"**

archiveNumbering=**"Sequence"**

archiveDateFormat=**"yyyy-MM-dd"**

archiveAboveSize=**"${file\_size}"**/>

<!--Log for console-->

<target xsi:type=**"ColoredConsole"**

name=**"logConsole"**

useDefaultRowHighlightingRules=**"false"**

layout=**"${layout\_format\_console}"**>

<highlight-word foregroundColor=**"White"** ignoreCase=**"true"** text=**"info"** wholeWords=**"true"** />

<highlight-word foregroundColor=**"Darkgray"** ignoreCase=**"true"** text=**"debug"** wholeWords=**"true"** />

<highlight-word foregroundColor=**"Red"** ignoreCase=**"true"** text=**"warn"** wholeWords=**"true"** />

<highlight-word backgroundColor=**"Red"** foregroundColor=**"White"** ignoreCase=**"true"** text=**"error"** wholeWords=**"true"** />

<highlight-row backgroundColor=**"DarkRed"** foregroundColor=**"Yellow"** condition=**"level == LogLevel.Fatal"** />

</target>

</targets>

<rules>

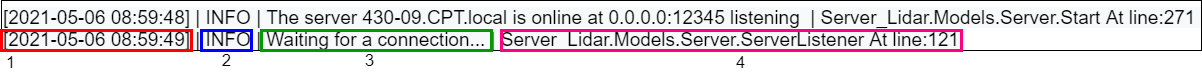
<logger name=**"\*"** minlevel=**"${logFile\_minLevel}"** writeTo=**"logFile"**/>

<logger name=**"\*"** minlevel=**"${logConsole\_minLevel}"** writeTo=**"logConsole"**/>

</rules>

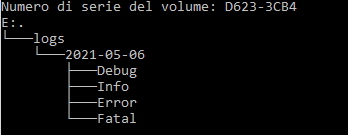
* ${**basedir**}: indica il percorso in cui si trova l’eseguibile dell’applicativo.
* ${**shortdate**}: indica il formato di data (yyyy-MM-dd).
* ${**level**}: indica il livello di log (**verbose**/**info**/**warning**/**error**/**fatal**).
* Target ColoredConsole cambia i colori del testo su console in base al livello.

## Struttura del messaggio



1. TimeStamp
2. Livello:
   * Debug
   * Info
   * Warn
   * Error
   * Fatal
3. Messaggio
4. Chi ha scaturito l’evento **(viene mostrato solo nel file)**

## Struttura delle cartelle



Le cartelle vengono generate automaticamente tramite un pattern definito, ovvero:

* Cartella logs: contente tutti i log relativi alle giornate.
* TimeStamp: il giorno in cui l’applicativo è stato avviato.
* Livello: suddivide i file in base al loro livello.
* File: file contente i messaggi suddivisi da un numero sequenziale.

## Client

Il ruolo del client in questo progetto è quello di mostrare, in uno spazio 3D, i punti scansionati dall’Arduino, richiedendoli al server.

## Connessione al server

Per potersi connettere al server, c’è innanzitutto bisogno di creare un file di nome **server.config** nella cartella “***C:/Users/<utente>/AppData/LocalLow/DefaultCompany/Scanner\_Lidar/***”. Nel file dovrà essere presente una stringa con il seguente protocollo: <hostname>:<port>

// Legge i valori dal file di configurazione e imposta le variabili su quei valori

private void SetServerInfo()

{

// Controllo se esiste il file di configurazione

if (File.Exists(destination))

{

// Leggo il contenuto del file

var file = File.ReadAllText(destination);

// Estraggo IP e porta

string[] serverInfo = file.Split(':');

// Controllo del formato

if (serverInfo.Length == 2)

{

// Controllo se la porta è un numero intero

if(int.TryParse(serverInfo[1], out serverPort))

{

if(serverPort < 0 || serverPort > 65535)

{

serverPort = DEFAULT\_PORT;

}

serverIp = serverInfo[0];

}else

{

throw new InvalidDataException("The port format is incorrect! Please insert an Integer value");

}

}else

{

throw new InvalidDataException("Configuration file incorrect! The format is: <server\_ip>:<server\_port>");

}

}

else

{

throw new InvalidDataException("Cannot find configuration file! " + destination);

}

}

Successivamente, per poter ricevere i punti scansionati, ho prima creato un metodo **Connect()** nel quale faccio partire una Thread che rimane in ascolto.

// Fa partire la Thread di lettura

private void Connect()

{

try

{

// Inizializzo la Thread e le passo una nuova ThreadStart (rappresenta il metodo che verrà eseguito quando parte la Thread) con il metodo GetPointsData

listener = new Thread(new ThreadStart(GetPointsData));

// Indico alla Thread che dovrà essere una Background Thread e non una Foreground Thread (se si vuole terminare l'esecuzione di tutto, la Thread lo permetterà)

listener.IsBackground = true;

// Faccio partire la Thread

listener.Start();

}

catch (Exception e)

{

Debug.Log(e);

}

}

## Lettura dei messaggi dal server

Il metodo **GetPointsData()** si occuperà di creare il socket e di leggere i messaggi in arrivo dal Server.

// Tento di effettuare la connessione con il Server

DisplayInfo("Attempting to connect to the Server, please wait...", Color.white);

// Imposto l'IP e la porta del Server

SetServerInfo();

connection = new TcpClient(serverIp, serverPort);

connected = true;

// Istanzio il buffer, dove manterrò i dati ricevuti dal Server

byte[] bytes = new byte[1024];

while (true)

{

// Ottengo un oggetto Stream per la lettura dei dati

stream = connection.GetStream();

// Variabile dove conserverò il numero di byte letti dal NetworkStream

int length;

while ((length = stream.Read(bytes, 0, bytes.Length)) != 0) // Se length è diverso da 0, quindi sono stati letti dei byte, continua il ciclo

{

// Istanzio l'array nel quale salevrò i dati contenuti nel Buffer

var incomingData = new byte[length];

// Copio i dati dal Buffer all'array incomingData

Array.Copy(bytes, 0, incomingData, 0, length);

// Converto l'array di byte in una stringa che rappresenta il messaggio ricevuto dal Server

serverMessage = Encoding.ASCII.GetString(incomingData);

if (serverMessage == "#,#,#")

{

receivedAllPoints = true;

}else

{

renderAPoint = true;

}

}

}

## Invio dei messaggi al server

Il metodo **SendMessage()** si occupa di inviare un messaggio al server. Il protocollo di invio dei messaggi è il seguente:

* Int: <punto\_iniziale>,<punto\_d’arrivo> (N.B.: se punto d’arrivo equivale a ‘\*’, verrano ricevuti tutti i punti.
* Msg:Stop: indica al server di smettere di inviare punti.

/// <summary>

/// Invia un messaggio al Server utilizzando una connessione Socket

/// </summary>

private void SendMessage()

{

// Se non c'è nessuna connessione con il Server non verrà inviato nessun messaggio e si uscirà dal metodo

if (connection == null || !connection.Connected)

{

return;

}

try

{

// Ottengo un oggetto Stream per la scrittura dei dati

stream = connection.GetStream();

// Se lo Stream può scrivere, si può procedere

if (stream.CanWrite)

{

// Converto il messaggio da stringa ad array di byte, in modo tale da poterlo inviare al Server

byte[] clientMessageAsByteArray = Encoding.ASCII.GetBytes(clientMessage);

// Scrive l'array di byte tramite lo stream di connection

stream.Write(clientMessageAsByteArray, 0, clientMessageAsByteArray.Length);

}

}

catch (IOException)

{

ConnectionInterrupted();

}

}

## Render dei punti

Per mostrare i punti nello spazio 3D, ho creato due metodi che si occupano di:

* Estrarre dalla stringa <x,y,z> le componenti del vettore.
* Creare l’oggetto primitivo sfera.
* Modificare la sua posizione in base alle componenti estrapolate prima.

// Prendo la stringa ricevuta dal Server e ne ricavo i valori che mi servono. Successivamente creo il punto

public void Render(string point)

{

string[] temp = point.Split(',');

float[] data = { float.Parse(temp[0]), float.Parse(temp[2]), float.Parse(temp[1]) };

CreatePoint(data);

}

// Crea un punto

private void CreatePoint(float[] data)

{

GameObject point = GameObject.CreatePrimitive(PrimitiveType.Sphere);

point.GetComponent<MeshRenderer>().shadowCastingMode = UnityEngine.Rendering.ShadowCastingMode.Off;

point.GetComponent<MeshRenderer>().receiveShadows = false;

Vector3 temp = new Vector3(data[0], data[1], data[2]);

point.transform.position = temp;

points.Add(point);

}

## Aggiunta degli assi nello spazio 3D

Per creare gli assi ho preso un codice online (vedi sitografia).

// Will be called after all regular rendering is done

public void OnRenderObject()

{

CreateLineMaterial();

// Apply the line material

lineMaterial.SetPass(0);

GL.PushMatrix();

// Set transformation matrix for drawing to

// match our transform

GL.MultMatrix(transform.localToWorldMatrix);

// Draw lines

GL.Begin(GL.LINES);

// Asse X

GL.Color(Color.red);

GL.Vertex3(-2500, 0, 0);

GL.Vertex3(2500, 0, 0);

// Asse Y

GL.Color(Color.green);

GL.Vertex3(0, -2500, 0);

GL.Vertex3(0, 2500, 0);

// Asse Z

GL.Color(Color.blue);

GL.Vertex3(0, 0, -2500);

GL.Vertex3(0, 0, 2500);

GL.End();

GL.PopMatrix();

}

## Movimento della camera

Per il movimento della camera, ci sono due metodi utilizzabili:

* Utilizzo del mouse
* Utilizzo delle freccette

Per cambiare tra questi due metodi, bisogna premere il tasto “**TAB**”. Il codice per il movimento della camera è stato in parte preso online (vedi sitografia).

// Se sono in "mouse look mode", ruoto la camera con il mouse, altrimenti con le freccette

if (useMouseLook)

{

currentRotation.x += Input.GetAxis("Mouse X") \* sensitivity \* (Time.deltaTime + 0.5f);

currentRotation.y -= Input.GetAxis("Mouse Y") \* sensitivity \* (Time.deltaTime + 0.5f);

currentRotation.x = Mathf.Repeat(currentRotation.x, 360);

currentRotation.y = Mathf.Clamp(currentRotation.y, -maxYAngle, maxYAngle);

transform.rotation = Quaternion.Euler(currentRotation.y, currentRotation.x, 0);

}

else

{

// Salvo la posizione iniziale della camera

rotation = transform.eulerAngles;

//Se il pulsante premuto è RightArrow, la camera ruota verso destra

if (Input.GetKey(KeyCode.RightArrow))

{

rotation.y += rotationSpeed;

transform.eulerAngles = rotation;

}

//Se il pulsante premuto è LeftArrow, la camera ruota verso sinistra

if (Input.GetKey(KeyCode.LeftArrow))

{

rotation.y -= rotationSpeed;

transform.eulerAngles = rotation;

}

//Se il pulsante premuto è UpArrow, la camera ruota verso l'alto

if (Input.GetKey(KeyCode.UpArrow))

{

rotation.x -= rotationSpeed;

transform.eulerAngles = rotation;

}

//Se il pulsante premuto è DownArrow, la camera ruota verso il basso

if (Input.GetKey(KeyCode.DownArrow))

{

rotation.x += rotationSpeed;

transform.eulerAngles = rotation;

}

}

## Movimento dello spettatore

Per il movimento dello spettatore sono stati utilizzati i tasti “**WASD**” (avanti, sinistra, indietro, destra), “**LeftShift**” (velocizza il movimento dello spettatore), “**LeftControl**” (movimento in basso) ed infine “**Space**” (movimento in alto).

// Se il pulsante premuto è Tab, si swithcerà tra mouse look o no

if (Input.GetKeyUp(KeyCode.Tab))

{

if(Cursor.lockState == CursorLockMode.Locked)

{

Cursor.lockState = CursorLockMode.None;

useMouseLook = false;

}else

{

Cursor.lockState = CursorLockMode.Locked;

useMouseLook = true;

}

}

// Se il pulsante premuto è LeftShift, il movimento della camera sarà raddoppiato

if (Input.GetKeyDown(KeyCode.LeftShift))

{

movementSpeed \*= 2;

}

// Se il pulsante premuto è LeftShift, il movimento della camera tornerà normale

if (Input.GetKeyUp(KeyCode.LeftShift))

{

movementSpeed /= 2;

}

// Se il pulsante premuto è W, la camera si muove avanti

if (Input.GetKey(KeyCode.W))

{

transform.position += transform.forward \* movementSpeed \* Time.deltaTime;

}

// Se il pulsante premuto è S, la camera si muove indietro

if (Input.GetKey(KeyCode.S))

{

transform.position -= transform.forward \* movementSpeed \* Time.deltaTime;

}

// Se il pulsante premuto è A, la camera si muove a sinistra

if (Input.GetKey(KeyCode.A))

{

transform.position -= transform.right \* movementSpeed \* Time.deltaTime;

}

// Se il pulsante premuto è D, la camera si muove a destra

if (Input.GetKey(KeyCode.D))

{

transform.position += transform.right \* movementSpeed \* Time.deltaTime;

}

// Se il pulsante premuto è Space, la camera salirà in alto

if (Input.GetKey(KeyCode.Space))

{

transform.position += new Vector3(0, transform.up.y \* movementSpeed \* Time.deltaTime, 0);

}

// Se il pulsante premuto è LeftControl, la camera scenderà in basso

if (Input.GetKey(KeyCode.LeftControl))

{

transform.position -= new Vector3(0, transform.up.y \* movementSpeed \* Time.deltaTime, 0);

}

## Interfaccia grafica

L’interfaccia grafica, una volta aperto l’applicativo, si mostrerà nel modo seguente:

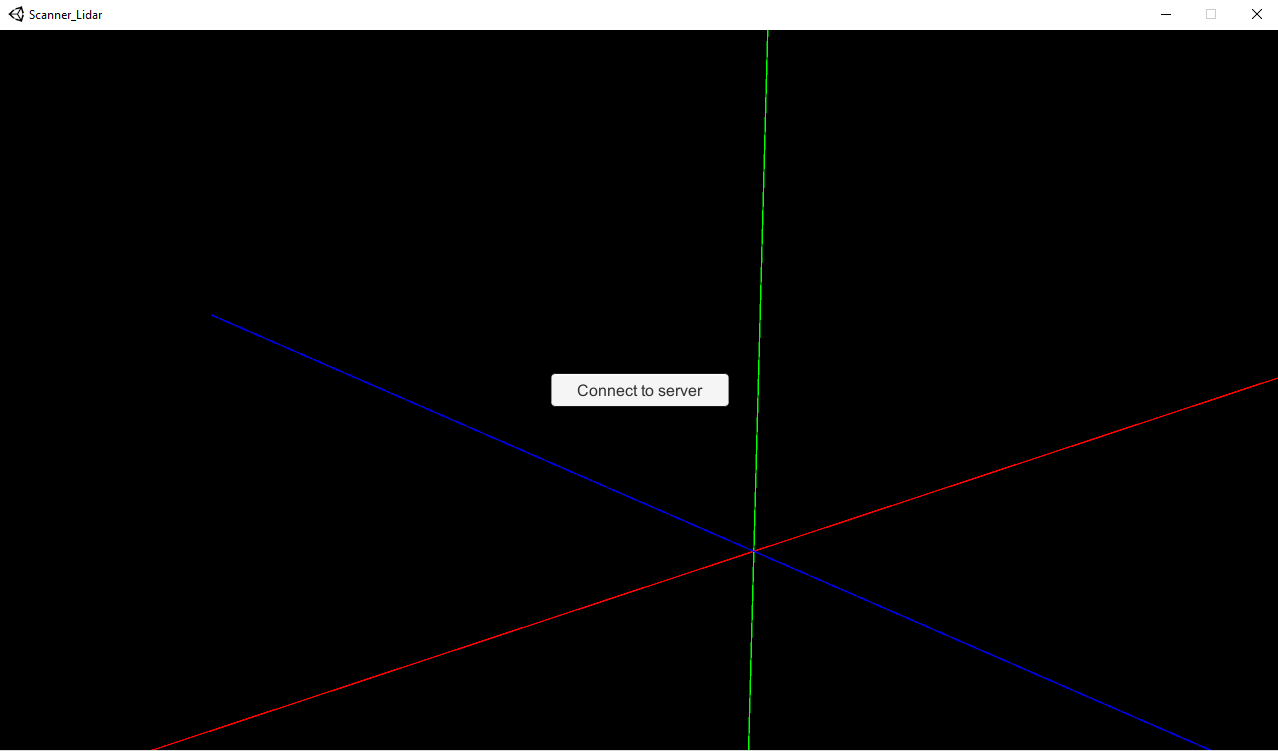


Figura - Client, interfaccia di connessione

Il bottone centrale permette all’utente di connettersi al server, così facendo, si procederà con un’interfaccia successiva (quella principale) dove si potranno richiedere i punti da renderizzare.

Nella parte in basso è presente un Label di log dove verranno notificate all’utente le operazioni eseguite o gli errori.



Figura - Client, esempio di errore nel Label

L’interfaccia principale, una volta eseguita la connessione con il server, si presenterà in questo modo:

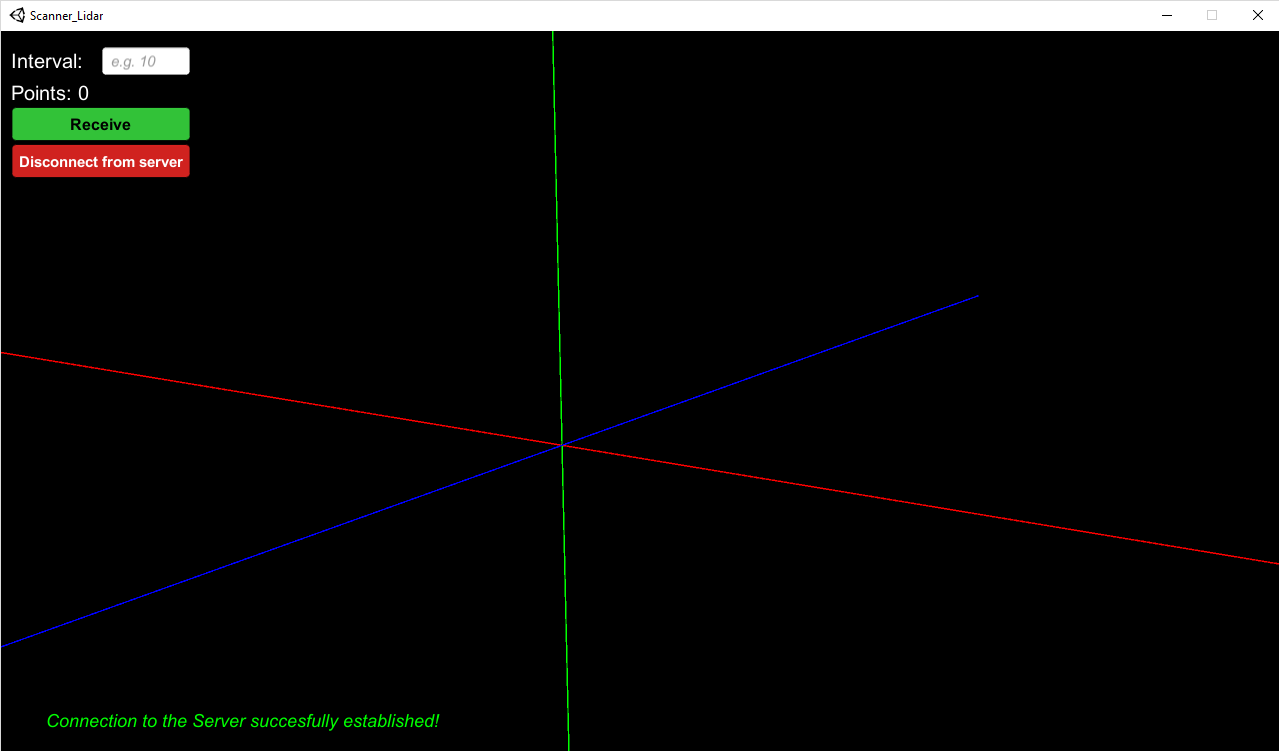


Figura - Client, interfaccia principale

La parte importante è il menu in alto a sinistra, che ci permette di:

* Inserire quanti punti si vogliono ricevere (se si lascia vuoto, vengono presi tutti i punti)
* Vedere il numero di punti renderizzati
* Iniziare/fermare la richiesta dei punti
* Disconnettersi dal server

## Arduino

Il ruolo dell’Arduino in questo progetto è quello di prelevare i dati dallo scanner Lidar e poi inviarli alla parte del server.

## Angolo di rotazione degli stepper motor

Per poter muovere il sensore con la giusta angolazione abbiamo dovuto cercare la documentazione degli stepper motor, infatti nel foglio c’era scritto che con 64 step per ogni rotazione, usciva il rapporto tra 360° e i 64 step, questo rapporto se viene moltiplicato per il numero di gradi che si vuole far eseguire al motore, uscirà un numero che permetterà di fare una rotazione della giusta angolazione.

const int stepsPerRevolution = 64 ;  // Il numero di step che i Stepper Motor eseguono per compiere una rivoluzione completa.

Stepper myStepperX(stepsPerRevolution, 11,9,10,8); // Definizione dello Stepper Motor orizzontale utilizzato dal programma.

Stepper myStepperY(stepsPerRevolution, 6,4,5,3); // Definizione dello Stepper Motor verticale utilizzato dal programma.

const int degreesX = 6; // Il numero di gradi eseguiti dal Stepper Motor dell'asse X, per step.

const int degreesY = 3; // Il numero di gradi eseguiti dal Stepper Motor dell'asse Y, per step.

const double ratio = 5.625\*1.0; // Il rapporto tra 360° e stepsPerRevolution (hardcoded per eccesso di approssimazione da parte di Arduino).

int stepX = ratio \* degreesX; // Il numero di gradi eseguiti dal Stepper Motor dell'asse X, per step, seguendo il rapporto associato al motore relativo.

int stepY = ratio \* degreesY; // Il numero di gradi eseguiti dal Stepper Motor dell'asse Y, per step, seguendo il rapporto associato al motore relativo.

## Attesa comunicazione con server

Come prima cosa L’Arduino deve assicurarsi che sia collegato al server prima di iniziare la scansione, infatti invia sulla porta seriale una serie di caratteri che consentiranno al server di riconoscere l’apparecchio, una volta inviato l’Arduino leggerà sulla seriale il messaggio di conferma così da poter iniziare la scansione, se questo messaggio però non viene ricevuto, l’Arduino continuerà a ripetere questa operazione.

if(send){

    ....

}else{

    if(Serial.readString() == "OK"){

        send = true;

    }else{

        Serial.write("CIAO");

    }

}

## Movimento motore asse orizzontale

Per la rotazione del motore orizzontale, inizialmente abbiamo deciso che si fermava ogni 6°.

while(intervalX \* degreesX <= 360 + degreesX){

    if(clockwise){

        myStepperX.step(stepX);

    }else{

        myStepperX.step(-stepX);

    }

    // Controlla se lo scanner LIDAR esiste ed è accessibile;

    // in seguito vengono inoltrati gli angoli di rotazioni sugli assi, e la

distanza ricavata dallo scanner.

…

    intervalX++; // Incremento numero rotazione orizzontale attuale.

}

Ma durante una discussione tra compagni di lavoro, ci siamo chiesti se quei 6° ero sufficienti tra un punto e latro, dopo averci sentito il professor Petrini ci ha detto che sulle piccole distanze (10-100 cm) non c’è molta differenza, ma sulle lunghe distanze (1 – X m) quei 6° gradi cominciavano un po’ ad essere tanti, per questo i nostri compagni che si sono occupati della parte hardware del progetto hanno trovato un modo per dimezzare i gradi.



Figura - Calcolo dell'interlacciato

Questo è il disegno che rappresenta l’idea che ci è venuta, ovvero che facciamo fare un mezzo passo falso (3), continuiamo a muoverci di un passo (6), però finiamo un passo prima, e facciamo ancora un altro mezzo passo falso. Questa tecnica ci permette di avere una grande flessibilità e di avere una campionatura più precisa.

// Mezzo step per sfasare righe pari sull'asse orizziontale

myStepperX.step(-ratio\*(degreesX/2));

    while(intervalX \* degreesX <= 360){ //  Deve terminare una rotazione prima.

        myStepperX.step(-stepX);

        ...

    }

// Mezzo step per ri-allineamento per righe dispari sull'asse orizzontale

myStepperX.step(-ratio\*(degreesX/2));

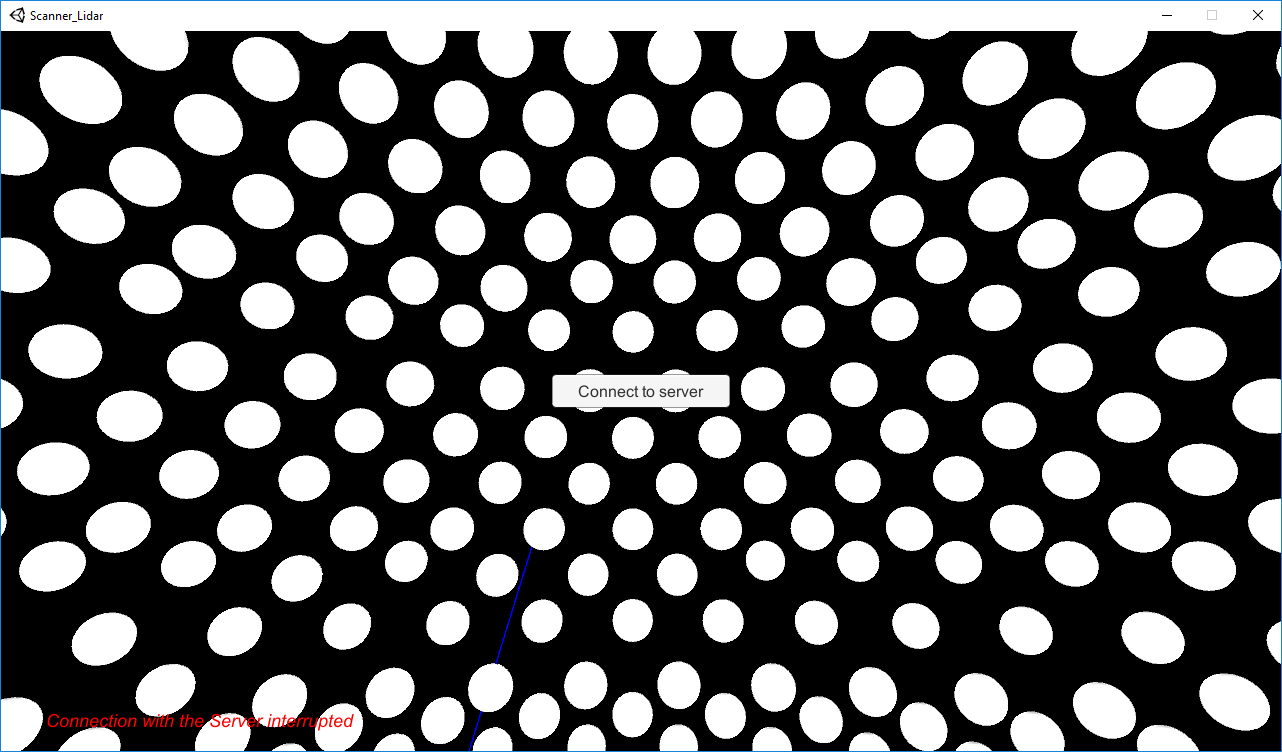


Figura - Client, griglia di scansione

## Movimento motore asse verticale

Per la rotazione del motore sull’asse verticale, abbiamo scelto di sfruttare il ciclo del metodo loop, infatti una volta che il motore sull’asse orizzontale ha effettuato i suoi 360° di rotazione, il motore verticale effettua il suo piccolo passo verso l’alto, aumentando così i gradi sulla rotazione verticale e fare in modo di invertire il senso di rotazione del motore orizzontale.

…

// Dopo aver completato un giro 360° sull'asse X, viene fatto uno step da parte

dello Stepper Motor dell'asse Y,

// incrementa il numero di rotazione verticale attuale, inverte il verso dello

Stepper Motor dell'asse X.

myStepperY.step(stepY);

intervalY++;

clockwise = !clockwise;

## Invio dato

Per il formato d’invio dei dati abbiamo due possibilità, all’inizio avevamo deciso di inviare il dato in formato CSV, successivamente abbiamo deciso di usare il formato JSON, ma per alcune complicazioni, abbiamo deciso di tornare al formato CSV. Nel codice c’è ancora la parte che invia in JSON, l’abbiamo lasciata nel caso che in sviluppi futuri si voglia utilizzare quel formato, sia già implementato nel codice.

JSON:

void sendJson(int dist) {

  DynamicJsonDocument data(200);

  data["point"]["x"] = intervalX \* degreesX;

  data["point"]["y"]   = intervalY \* degreesY;

  data["point"]["distance"] = dist;

  serializeJson(data, Serial);

}

CSV:

…

if (tfmini.available()){

    Serial.print(intervalX \* degreesX);

    Serial.print(",");

    Serial.print(intervalY \* degreesY);

    Serial.print(",");

    Serial.print(tfmini.getDistance());

}

…

## Fine esecuzione e reset motori

Per finire la scansione del motore, abbiamo deciso che sono i gradi del motore verticale a dircelo, ovvero che una volta raggiunta una certa angolazione, la raccolta dati viene interrotta, i motori vengono riportati alla loro posizione iniziale e quest’ultima viene inviata al server, con un messaggio da parte dell’Arduino che indica che ha terminato la scansione.

if(!finish){

    ...

    // Se il nuovo numero di rotazione causa l'eccesso dei 90°

    // da parte del prossimo step, viene fatto il reset dei Step Motor

    // e viene settato lo stato della scansione come terminata.

    if(intervalY \* degreesY > 90){

        myStepperY.step(-stepY \* intervalY);

        myStepperX.step(-stepX \* intervalX);

        finish = true;

        Serial.write("<EOF>");

    }

}

## Struttura HW

Dopo alcune discussioni con il nostro professore Geo Petrini ci siamo resi conto che la nostra progettazione iniziale della struttura dell’hardware non era completamente corretta, quindi abbiamo dovuto capire come migliorare e rendere performante la struttura.

Dopo una accurata progettazione su carta, ne abbiamo discusso ancora con il professore Petrini, il quale ci fa che andava bene.

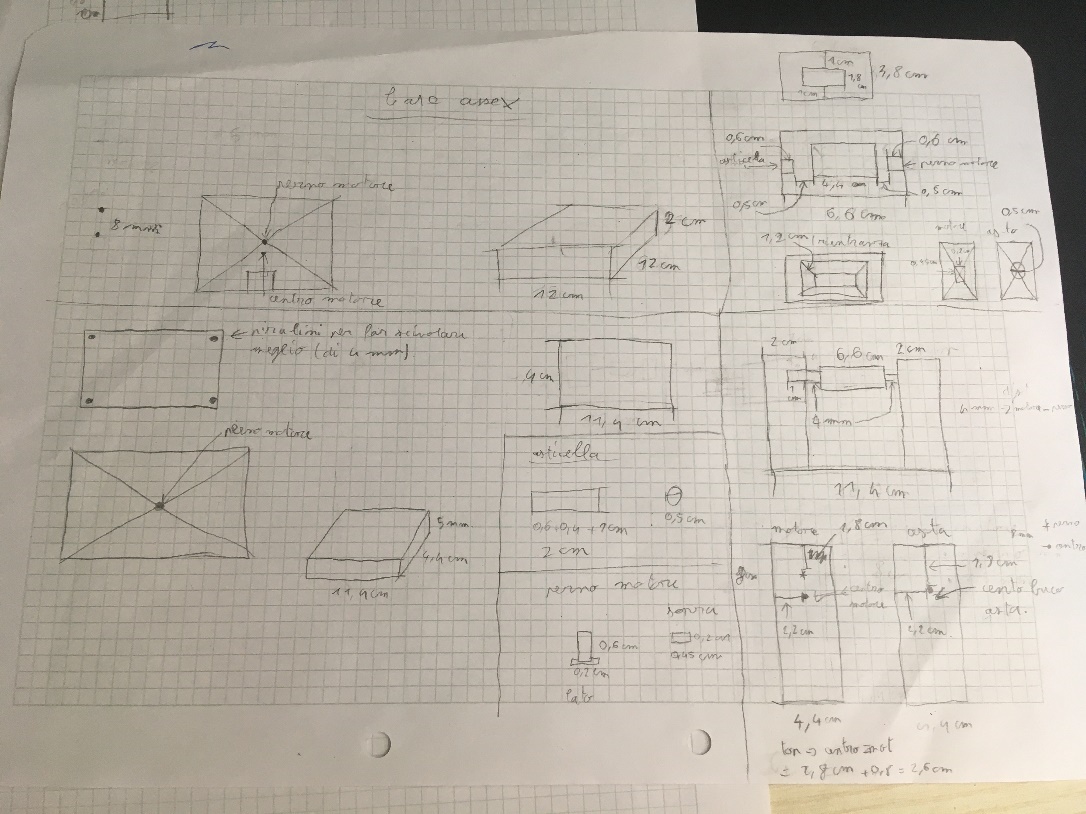


Figura - Disegno struttura HW, 1

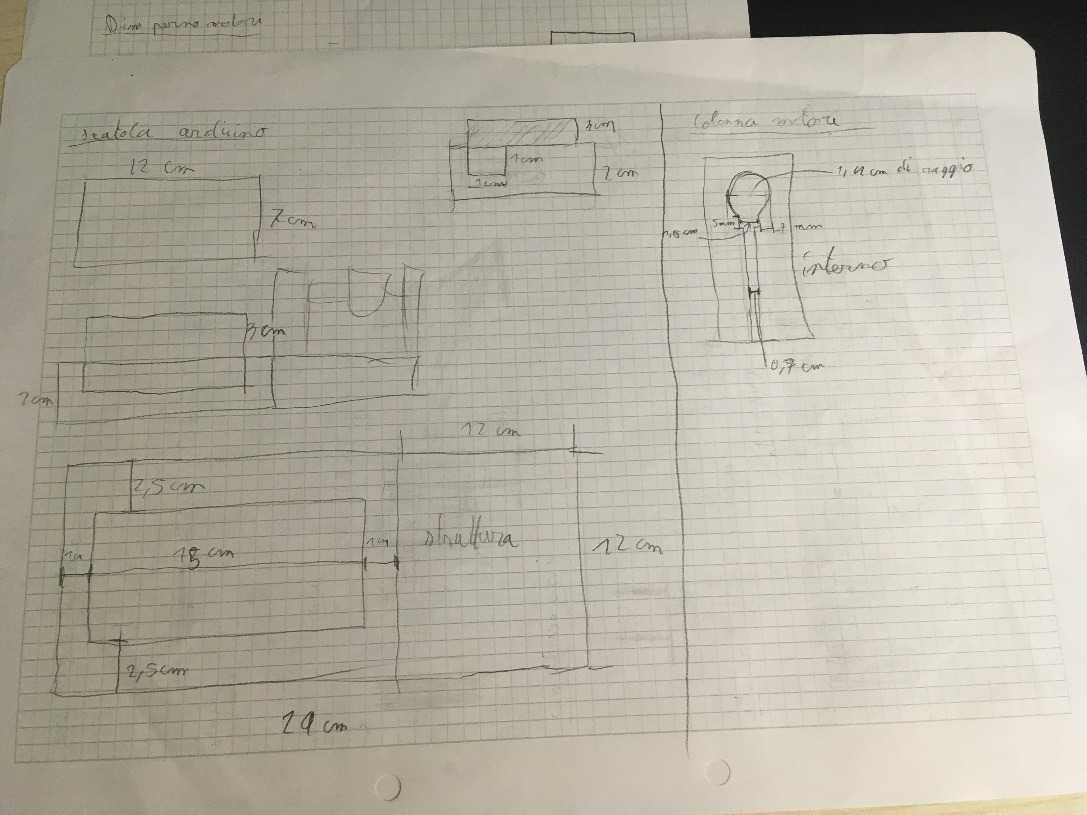


Figura - Disegno struttura HW, 2

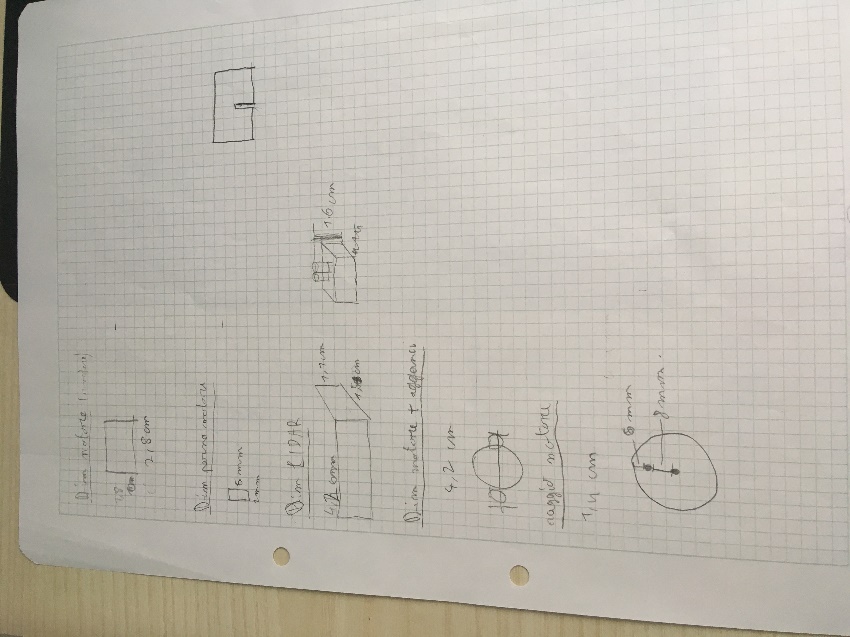


Figura - Disegno struttura HW, 3

Il professore Petrini si è gentilmente offerto per la realizzazione dei pezzi, lui ci ha consigliato di farli in legno perché è un materiale facilmente modellabile, ci ha chiesto solamente di fare i disegni con un Easel, così che possa tagliarli.

I disegni elettronici sono usciti nei seguenti modi:

* Parte di sostegno per lo scanner Lidar

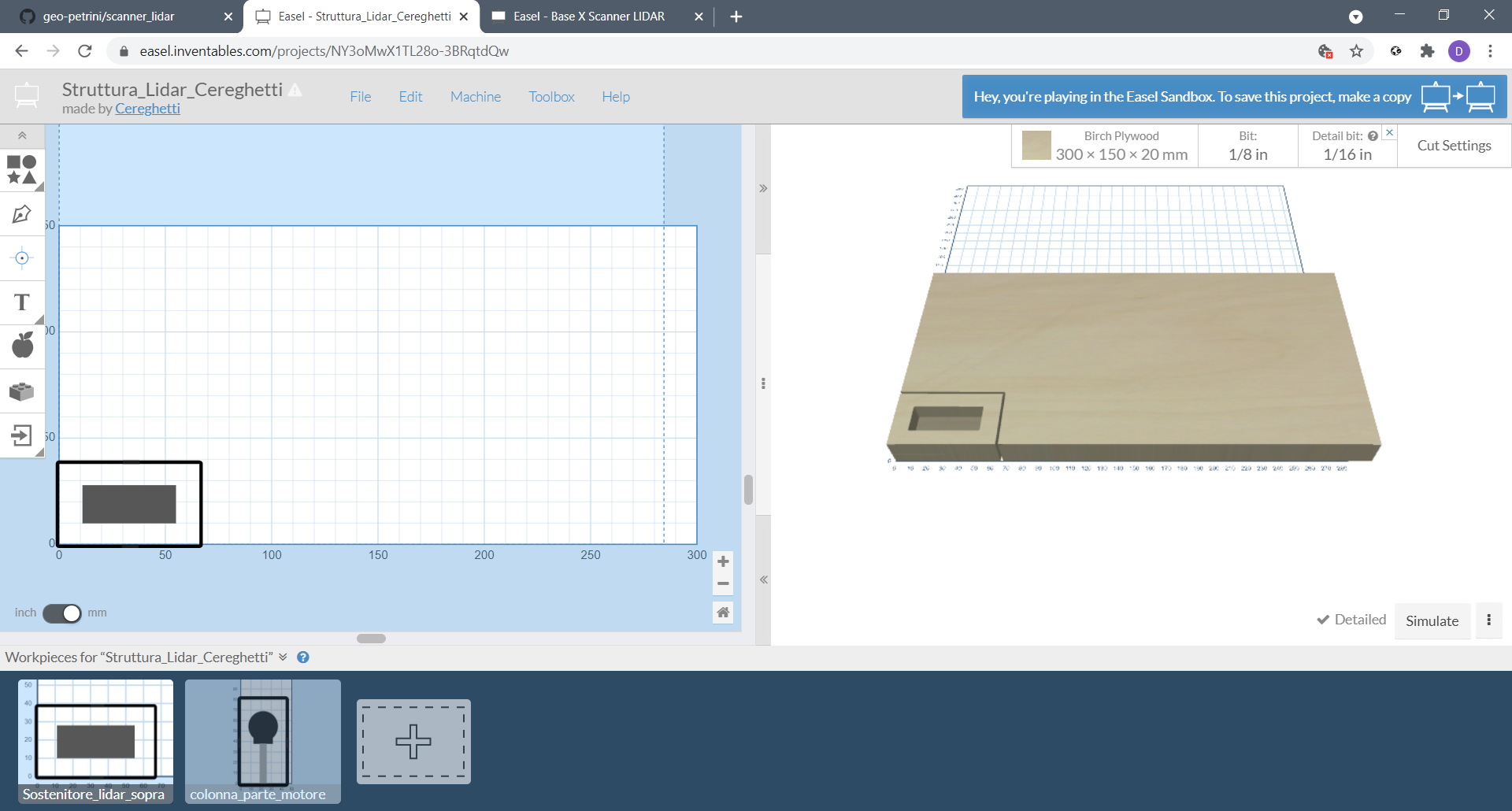
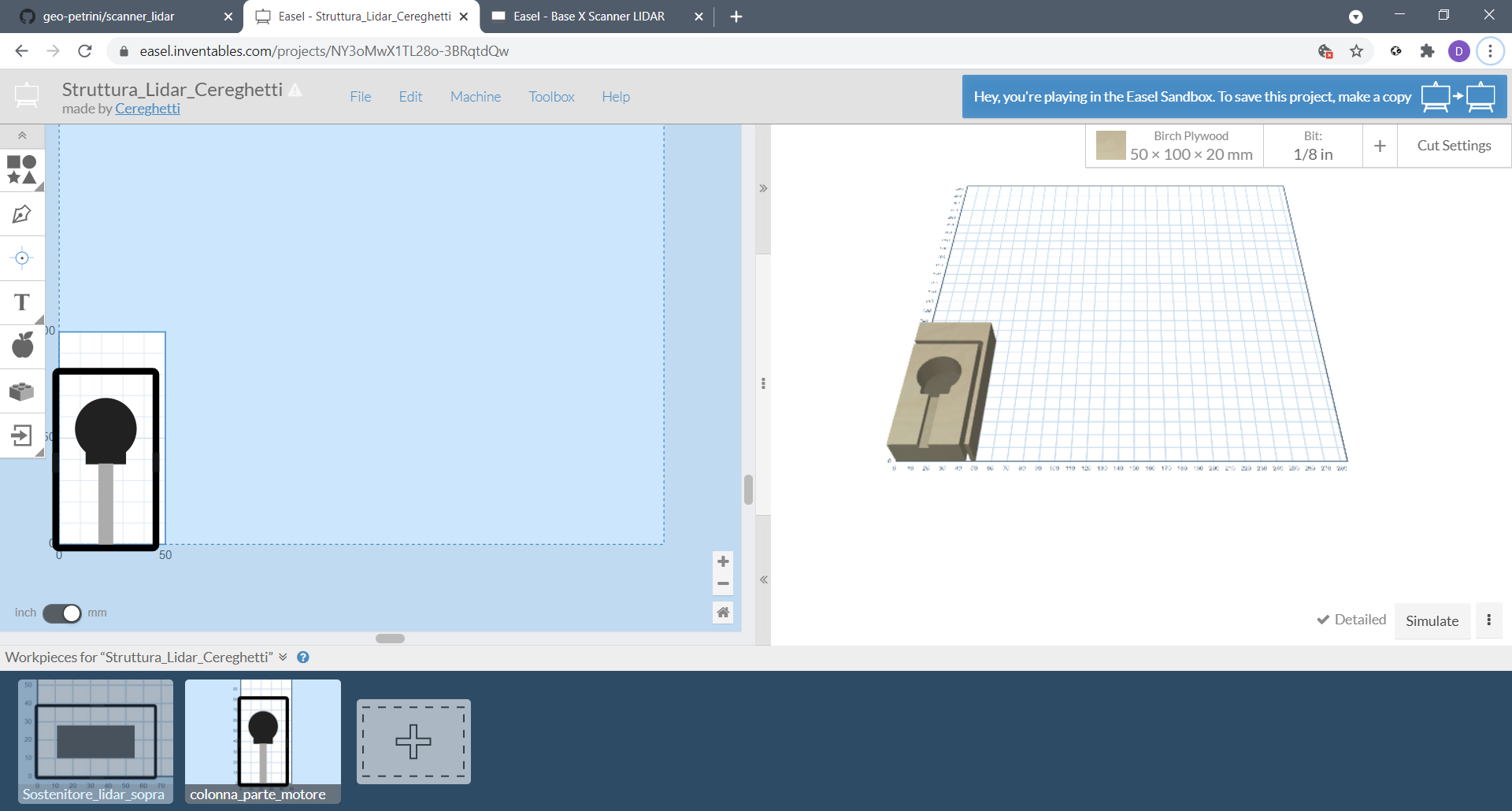
 

Figura - Sostegno Lidar e sostegno motore verticale

* Parte della base per lo scanner

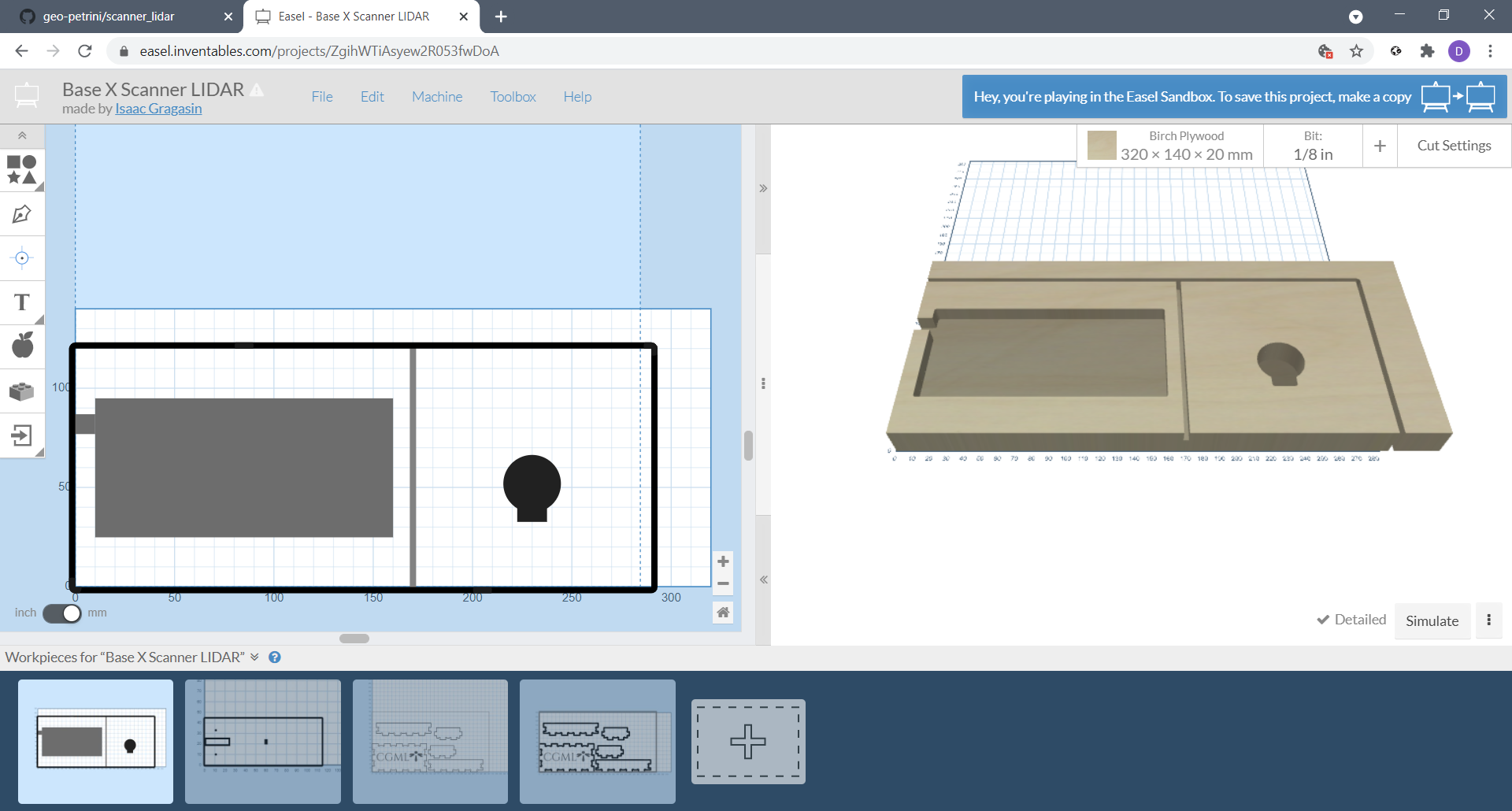


Figura - Base HW

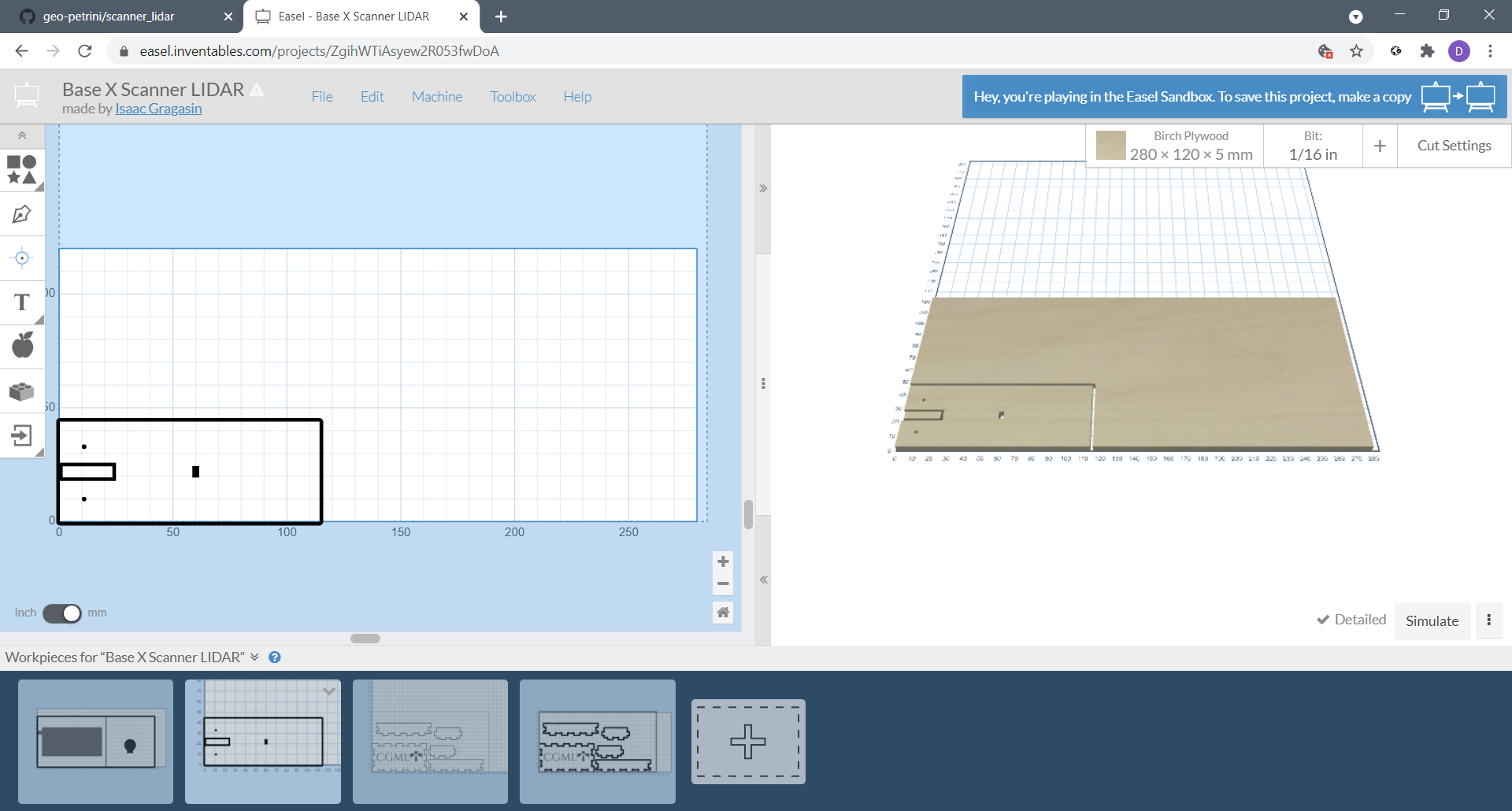


Figura - Base rotazione HW

Questa seguente immagine rappresenta il risultato finale dell’assemblaggio tra il circuito di Arduino e la sua dedicata struttura.



## Corretta posizione dei cavi

Per un fattore di funzionalità, prima di ogni scansione bisogna accertarsi che i cavi siano nella giusta posizione e che non siano tirati, sennò i motori non saranno in grado di eseguire le giuste rotazioni e si rischia di rovinare la struttura.

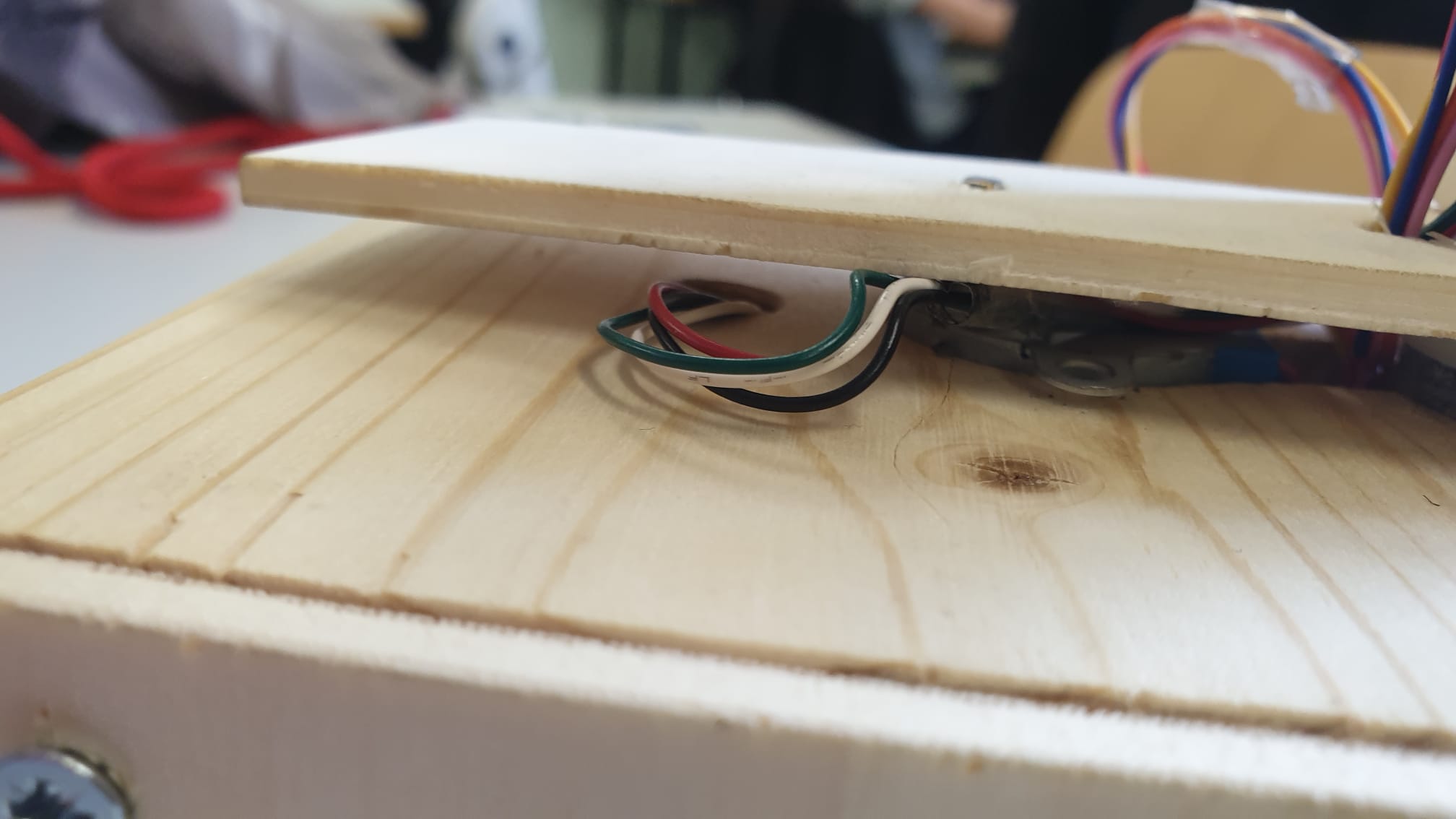


Figura - Posizionamento cavi, 1



Figura - Posizionamento cavi, 2

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 1  REQ-HW-1 | **Descrizione:** | Scansione a 360° sull’asse orizzontale |
| **Prerequisiti:** | - | | |
| **Procedura:** | Attaccare al perno del motore dell’asse orizzontale il foglio di carta da test, attaccare l’Arduino ad una presa USB, aspettare che comincia la sua esecuzione. | | |
| **Risultati attesi:** | Il motore sull’asse orizzontale esegua una rotazione di 360° | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 2  REQ-HW-2 | **Descrizione:** | Scansione a 90° sull’asse verticale |
| **Prerequisiti:** | REQ-HW-1 perfettamente funzionante | | |
| **Procedura:** | Attaccare al perno del motore dell’asse verticale il foglio di carta da test, attaccare l’Arduino ad una presa USB, aspettare il termine della sua esecuzione per determinare i gradi di rotazione. | | |
| **Risultati attesi:** | Il motore sull’asse verticale esegua una rotazione di 90° | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 3  REQ-HW-3 | **Descrizione:** | Minor delay possibile tra scansione LIDAR e rotazione motore |
| **Prerequisiti:** | - | | |
| **Procedura:** | Attaccare l’Arduino ad una presa USB, aspettare inizi l’esecuzione . | | |
| **Risultati attesi:** | Che il movimento del motore orizzontale sia fluido, evitando di fare scatti. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 4  REQ-HW-4 | **Descrizione:** | Invio dati al PC |
| **Prerequisiti:** | Arduino IDE installato | | |
| **Procedura:** | Attaccare l’Arduino ad una presa USB, aprire il monitor serial, aspettare inizi l’esecuzione. | | |
| **Risultati attesi:** | Sul monitor seriale devono apparire dei messaggi contenenti i valori di angolo orizzontale, angolo verticale e distanza. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 5  REQ-SW-1 | **Descrizione:** | Acquisire i dati da Arduino tramite USB |
| **Prerequisiti:** | Arduino deve essere attaccato ad una porta USB e il baud rate deve essere di 115200 e la porta COM deve essere riconosciuta. | | |
| **Procedura:** | Aprire la connessione con la seriale e stampare i dati letti. | | |
| **Risultati attesi:** | Stringhe di byte passate da Arduino. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 6  REQ-SW-2 | **Descrizione:** | Interpretare i dati in ambiente 3D |
| **Prerequisiti:** | Server attivo, connessione al server con successo | | |
| **Procedura:** | Inserire un valore di intervallo di punti e cliccare il bottone “Receive”. | | |
| **Risultati attesi:** | Il contatore di “Points” deve aumentare man mano che si creano pallini nello spazio 3D. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 7  REQ-PT-224 | **Descrizione:** | Logging |
| **Prerequisiti:** | - | | |
| **Procedura:** | Configurare il file di log e avviare l’applicativo. | | |
| **Risultati attesi:** | Struttura e messaggi formattati. | | |

## Risultati test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 1  REQ-HW-1 | **Descrizione:** | Scansione a 360° sull’asse orizzontale |
| **Risultati:** | il motore esegue una rotazione di 360° sull’asse orizzontale, questo è più visibile grazie al foglio di test attaccato al perno. | | |
| **Risultato finale:** | Positivo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 2  REQ-HW-2 | **Descrizione:** | Scansione a 90° sull’asse verticale |
| **Risultati:** | una volta terminata l’esecuzione del codice, notiamo che il motore verticale ha eseguito una rotazione di 90°. | | |
| **Risultato finale:** | Positivo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 3  REQ-HW-3 | **Descrizione:** | Minor delay possibile tra scansione LIDAR e rotazione motore |
| **Risultati:** | Il motore orizzontale è perfettamente fluido e non fa nessun scatto. | | |
| **Risultato finale:** | Positivo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 4  REQ-HW-4 | **Descrizione:** | Invio dati al PC |
| **Risultati:** | Sul monitor seriali appaiono i dati dell’angolo orizzontale, dell’angolo verticale e della distanza raccolta. | | |
| **Risultato finale:** | Positivo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 5  REQ-SW-1 | **Descrizione:** | Acquisire dati da Arduino tramite USB |
| **Risultati:** | Sulla console vengono stampati i dati fittizi di test. | | |
| **Risultato finale:** | Positivo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 6  REQ-SW-2 | **Descrizione:** | Interpretare i dati in ambiente 3D |
| **Risultati:** | Il contatore di “Points” aumenta e i pallini si creano in giro per lo spazio. | | |
| **Risultato finale:** | Positivo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 7  REQ-PT-224 | **Descrizione:** | Logging |
| **Risultati:** | Le cartelle vengono create in automatico, così come i file sono tutti formattati. | | |
| **Risultato finale:** | Positivo | | |

# Consuntivo



Figura - Gantt consuntivo

Il modello di progetto che abbiamo utilizzato è stato l’agile.

# Conclusioni

## Sviluppi futuri

In futuro può essere migliorata la qualità, ovvero fare in modo che lo scanner prenda più punti. Inoltre, quest’ultimi potrebbero essere salvati in un database per statistiche, ecc.... Un’altra aggiunta futura può essere quella di aggiungere delle texture ai punti e si potrebbe farli collidere creando una planimetria.

Si potrebbe aggiungere la gestione delle richieste tramite altri protocolli (es.: GET).

## Considerazioni personali

Veljko:

* Il progetto mi è piaciuto moltissimo e mi ha coinvolto molto. Rispetto al mio scorso progetto, questo era molto stimolante, infatti impegnarmi assieme ai miei compagni per raggiungere un obiettivo che, inizialmente, suonava impossibile all’orecchio, mi ha regalato una vera e propria soddisfazione personale. Inoltre, questo progetto mi ha fatto rendere conto che noi abbiamo le capacità per creare dei prodotti validi, il che mi rassicura molto e mi rafforza l’autostima riguardo la conoscenza del nostro ambito, ovvero d’informatica.

Daniele:

* Fare questo progetto mi è piaciuto moltissimo, il team ha lavorato benissimo, uniti, flessibili e aperti ai confronti. Occupandomi della parte dell’hardware, mi ha davvero stimolato, lavorare in una parte dove il fatto di scrivere codice non era così presente, l’ho trovato più rilassante e poco stressante. Il fatto anche di fare del lavoro manuale con il legno mi ha davvero divertito. Mi ha davvero stimolato anche discutere del progetto con una persona più esperta di noi. Lavorare in gruppo secondo me è molto più stimolante e divertente piuttosto che fare un progetto interamente da solo.

Matteo:

* Durante questo progetto ho perfezionato le mie conoscenze in C#, infatti fino ad ora non avevo mai sviluppato completamente un’applicazione che gestiva ed elaborava dati reali. All’inizio ero abbastanza propositivo per fare l’implementazione della parte HW, con il relativo sviluppo della piccola parte software per gestire Arduino, poi invece ho preferito prendere parte per il rendering e l’elaborazione dei dati. Questo progetto mi ha aiutato a non dubitare sul lavoro di squadra, siccome sono una persona che preferisce molto lavorare da solo, infatti sono riuscito a comunicare facilmente con i miei compagni e collaborare con loro. Per me è stato molto piacevole lavorare con questo team, poiché ognuno di noi, aveva delle competenze che aiutavano a completare le lacune degli altri e ciò ci ha portato a realizzare questo progetto, di cui mi ritengo molto soddisfatto, anche se avrei ambito ad un livello finale nettamente superiore, ma che a causa del tempo, era difficile da raggiungere, perciò sarà uno sviluppo futuro che mi farebbe piacere portare avanti, sperando con gli stessi compagni. Per quanto riguarda il mio lavoro, ritengo che avrei potuto osare qualcosa di più e non limitarmi nel solo realizzare ciò che era stato richiesto da mandato, però sono comunque molto soddisfatto.

Isaac:

* Svolgere questo progetto è stato un vero piacere e molto costruttivo per me. Malgrado gli intoppi che abbiamo avuto in tutto questo periodo, siamo riusciti ad adattarci al meglio delle nostre capacità, dividerci gli incarichi appropriatamente e a mantenere una produttività costante ogni volta. Lo sviluppo della struttura hardware è stata un’attività che mi ha particolarmente colpito, osservando come ci possono essere varie caratteristiche facilmente trascurabili ma vitali per la realizzazione di un modello ottimale, e ciò mi ha insegnato a tenere una certa meticolosità. Il risultato soddisfacente di questo progetto è per me una prova di come posso contribuire e collaborare meglio di quanto pensassi lavorando a gruppi, considerando che ho preferito lavorare da solo finora.

# Bibliografia

## Sitografia

* [https://gist.github.com/danielbierwirth/0636650b005834204cb19ef5ae6ccedb#file-tcptestclient-cs-L26](https://gist.github.com/danielbierwirth/0636650b005834204cb19ef5ae6ccedb%23file-tcptestclient-cs-L26), *GitHub*, 25 febbraio 2021
* <https://docs.unity3d.com/ScriptReference/GL.html>, *Unity*, 1 aprile 2020
* <https://answers.unity.com/questions/1344322/free-mouse-rotating-camera.html>, *Unity*, 1 aprile 2020
* <https://answers.unity.com/questions/1300019/how-do-you-save-write-and-load-from-a-file.html>, *Unity*, 22 aprile 2021
* <https://www.arduino.cc/reference/en/language/functions/communication/serial/>, *Arduino CC*, 25 febbraio 2021
* <https://www.instructables.com/C-Serial-Communication-With-Arduino/>, *Instructables*, 25 febbraio 2021

# Allegati

* [Manuale stepper motor](https://github.com/geo-petrini/scanner_lidar/blob/main/Progettazione/ManualeStepperMotor.pdf)
* [Manuale controller stepper motor](https://github.com/geo-petrini/scanner_lidar/blob/main/Progettazione/ManualeControllerStepperMotor.png)
* [Manuale scanner Lidar](https://github.com/geo-petrini/scanner_lidar/blob/main/Progettazione/ManualeScannerLidar.pdf)
* [Diari di lavoro](https://github.com/geo-petrini/scanner_lidar/tree/main/Diario)
* [Librerie arduino](https://github.com/geo-petrini/scanner_lidar/tree/main/Source_Code/Arduino/Librerie)