Scanner Lidar

Documentazione

Sommario

[1 Introduzione 3](#_Toc71900629)

[1.1 Informazioni sul progetto 3](#_Toc71900630)

[1.2 Abstract 3](#_Toc71900631)

[1.3 Scopo 3](#_Toc71900632)

[2 Analisi 4](#_Toc71900633)

[2.1 Analisi del dominio 4](#_Toc71900634)

[2.2 Analisi e specifica dei requisiti 4](#_Toc71900635)

[2.3 Use case 6](#_Toc71900636)

[2.4 Pianificazione 7](#_Toc71900637)

[2.5 Analisi dei mezzi 8](#_Toc71900638)

[2.5.1 Software 8](#_Toc71900639)

[2.5.1.1 Tools 8](#_Toc71900640)

[2.5.1.2 Librerie 8](#_Toc71900641)

[2.5.1.2.1 Arduino 8](#_Toc71900642)

[2.5.1.2.2 Server 8](#_Toc71900643)

[2.5.2 Hardware 9](#_Toc71900644)

[2.5.2.1 Arduino e le sue componenti 9](#_Toc71900645)

[2.5.2.2 Macchina Virtuale 9](#_Toc71900646)

[2.5.2.3 Macchina fisica scolastica 9](#_Toc71900647)

[2.5.2.4 Portatile 9](#_Toc71900648)

[3 Progettazione 10](#_Toc71900649)

[3.1 Design dell’architettura del sistema 10](#_Toc71900650)

[3.2 Design dei dati 10](#_Toc71900651)

[3.3 Design delle interfacce 10](#_Toc71900652)

[3.4 Design procedurale 11](#_Toc71900653)

[3.4.1 Arduino 11](#_Toc71900654)

[3.4.2 Software 11](#_Toc71900655)

[3.5 Design Hardware 11](#_Toc71900656)

[4 Implementazione 12](#_Toc71900657)

[4.1 Server 12](#_Toc71900658)

[4.2 Client 13](#_Toc71900659)

[4.3 Arduino 13](#_Toc71900660)

[4.3.1 Angolo di rotazione degli stepper motor 13](#_Toc71900661)

[4.3.2 Attesa comunicazione con server 13](#_Toc71900662)

[4.3.3 Movimento motore asse orizzontale 14](#_Toc71900663)

[4.3.4 Movimento motore asse verticale 14](#_Toc71900664)

[4.3.5 Invio dato 14](#_Toc71900665)

[4.3.6 Fine esecuzione e reset motori 14](#_Toc71900666)

[4.3.7 Struttura HW 15](#_Toc71900667)

[4.3.8 Corretta posizione dei cavi 17](#_Toc71900668)

[5 Test 19](#_Toc71900669)

[5.1 Protocollo di test 19](#_Toc71900670)

[5.2 Risultati test 19](#_Toc71900671)

[6 Consuntivo 20](#_Toc71900672)

[7 Conclusioni 20](#_Toc71900673)

[7.1 Sviluppi futuri 20](#_Toc71900674)

[7.2 Considerazioni personali 20](#_Toc71900675)

[8 Bibliografia 20](#_Toc71900676)

[8.1 Bibliografia 20](#_Toc71900677)

[8.2 Sitografia 20](#_Toc71900678)

[9 Allegati 20](#_Toc71900679)

# 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

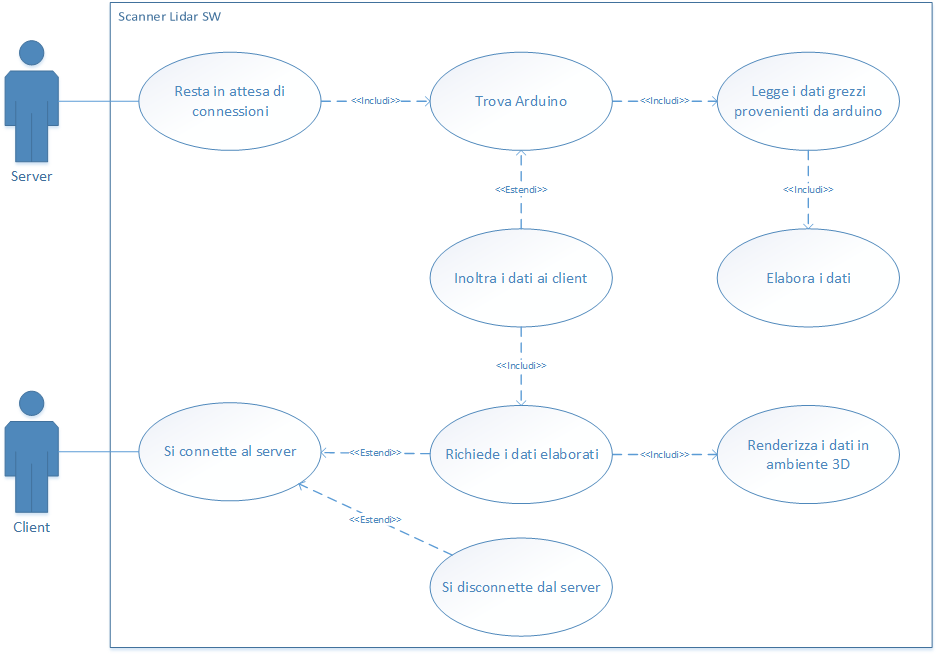


Figura - Use Case

## Pianificazione

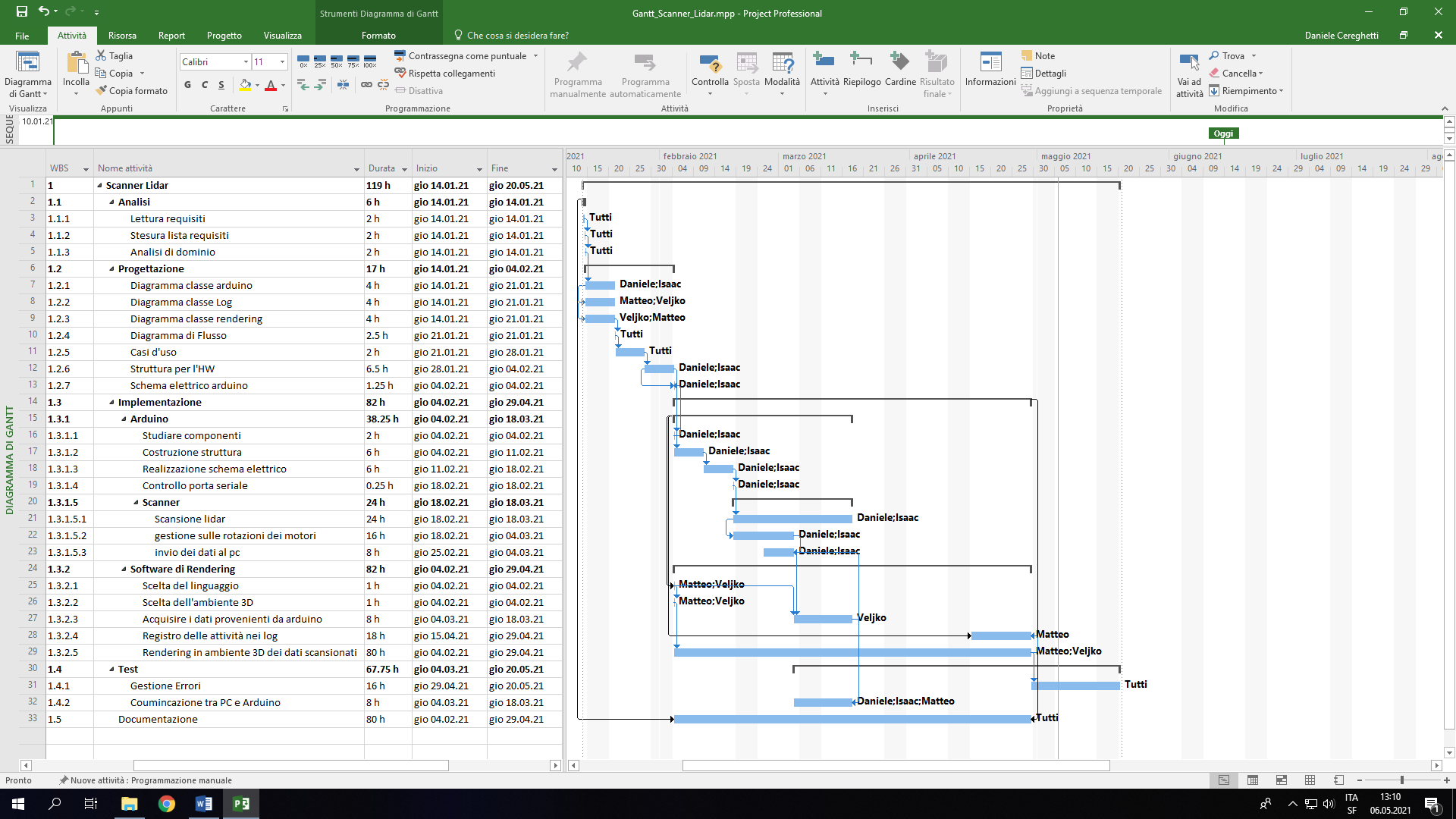


Figura - Gantt preventivo

## 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

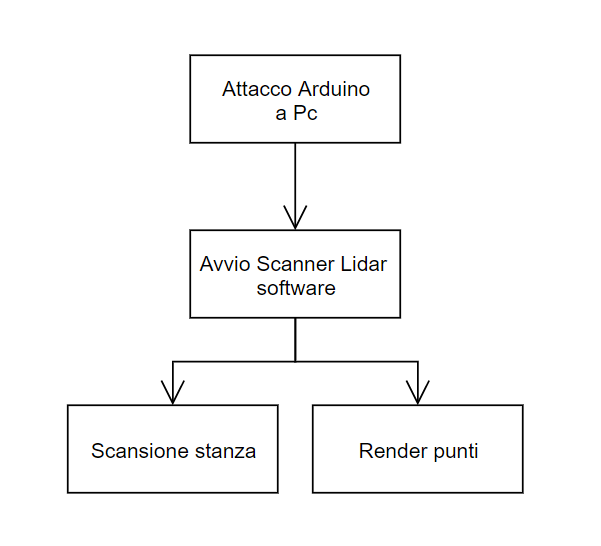
### Portatile

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



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.

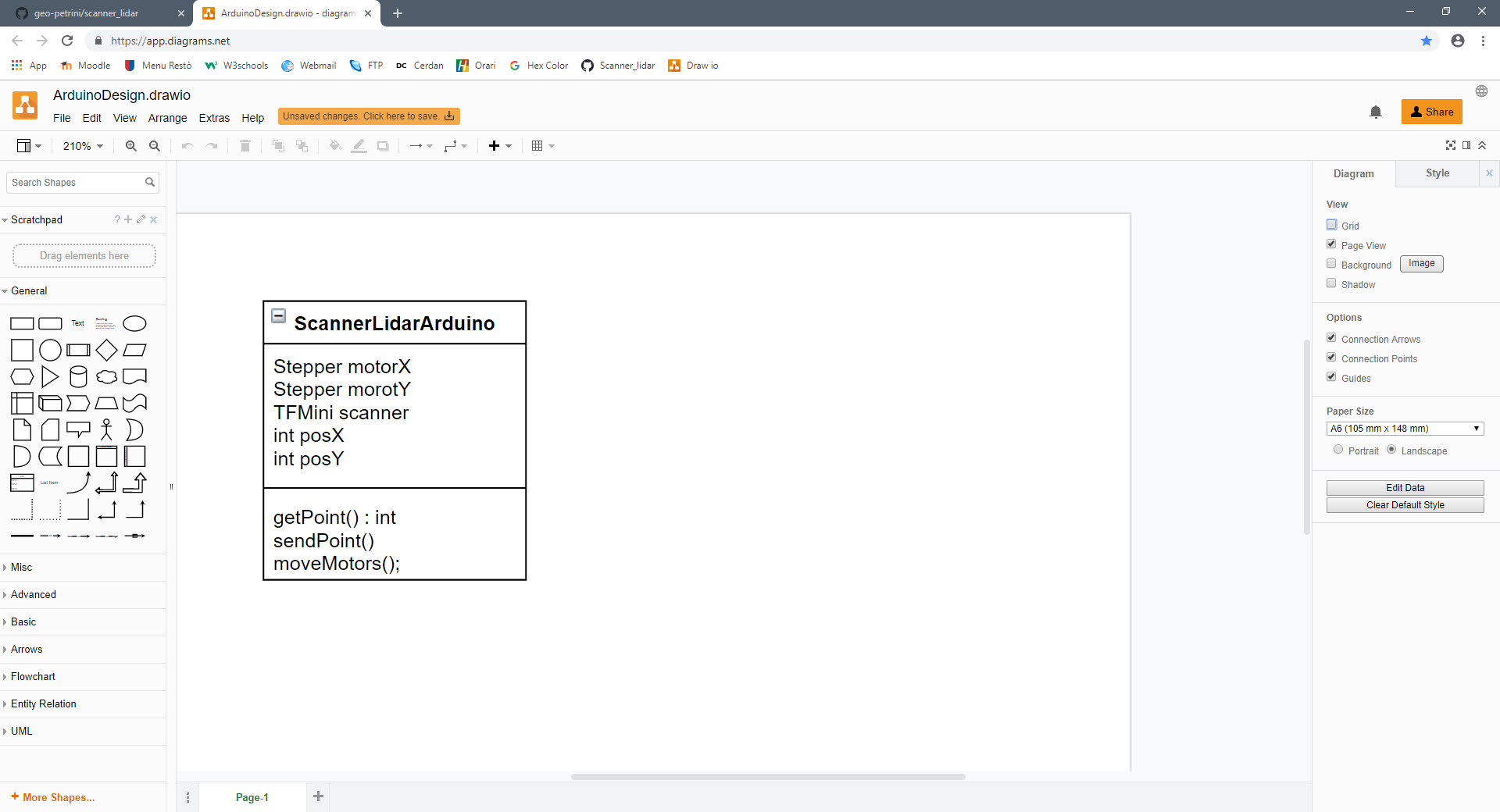
Il protocollo per passare i dati tra le due parti è simile a quello del CSV, ovvero che passiamo 3 valori separati da una virgola: il primo è la rotazione orizzontale, il secondo è la rotazione verticale e il terzo è la distanza presa.

## Design delle interfacce

<fare veljko>

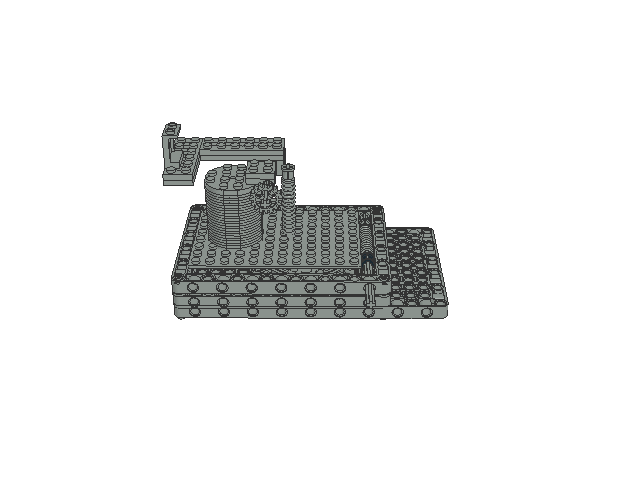
## Design procedurale

## Arduino

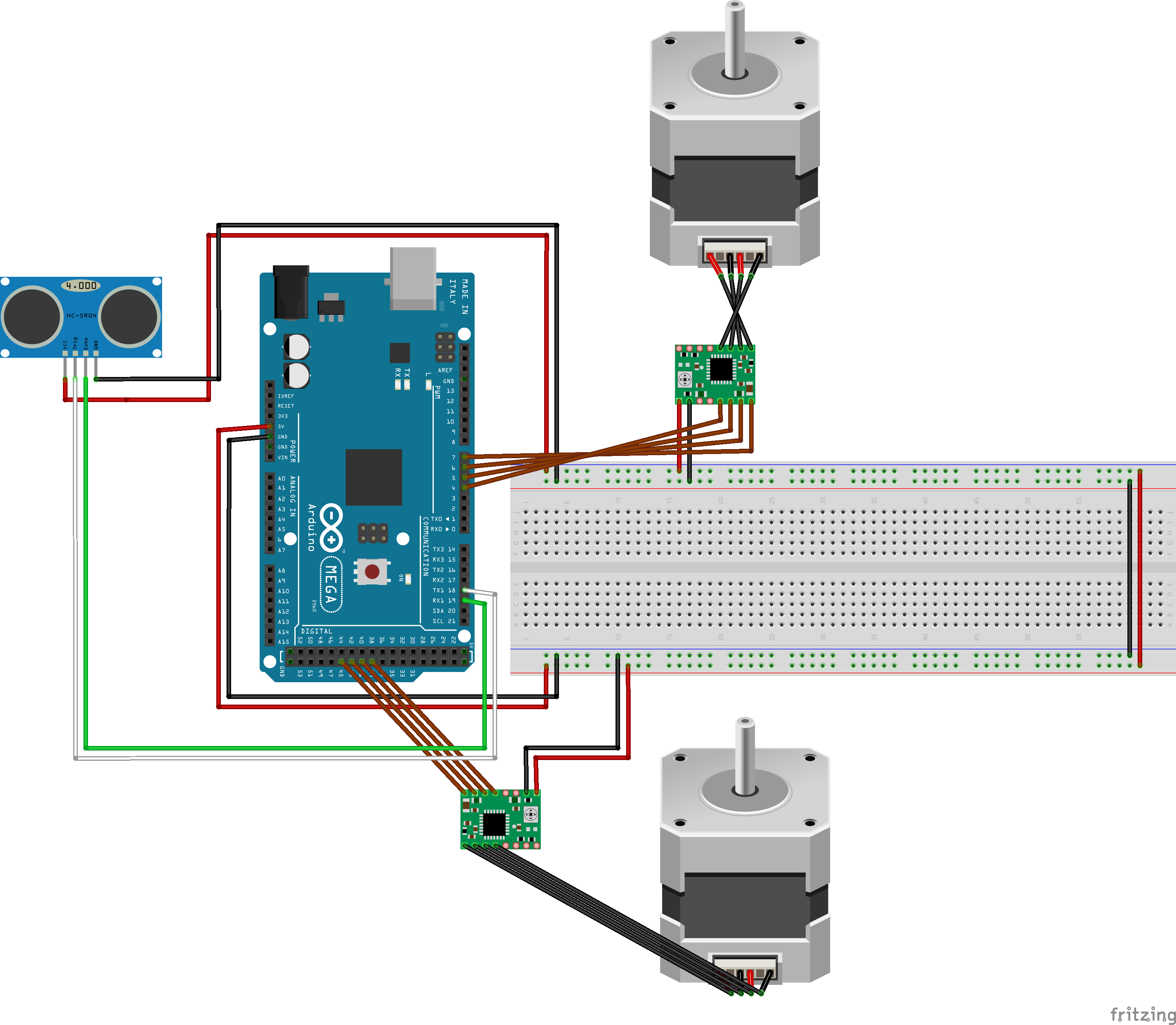


## Software

## Design Hardware



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).



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

## 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**);**

**}**

## 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>

## MultiThreading

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

TcpClient client **=** **(**TcpClient**)**obj**;**

Byte**[]** bytes **=** **new** Byte**[**1024**];**

string clientMessage**;**

newConnection**();**

var stream **=** client**.**GetStream**();**

int length**;**

MyThread mT **=** **new** MyThread**();**

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

**{**

var incomingData **=** **new** byte**[**length**];**

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

clientMessage **=** Encoding**.**ASCII**.**GetString**(**incomingData**);**

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

string header **=** message**[**0**];**

**if** **(**header**.**Equals**(**"Int"**))**

**{**

**if** **(!**mT**.**IsSending**)**

**{**

mT**.**IsSending **=** **true;**

**}**

string**[]** content **=** message**[**1**].**Split**(**','**);**

int min **=** int**.**Parse**(**content**[**0**]);**

int max**;**

**if** **(**content**[**1**].**Equals**(**"\*"**))**

**{**

max **=** vector3s**.**Count **-** 1**;**

**}**

**else**

**{**

max **=** int**.**Parse**(**content**[**1**]);**

**}**

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

**{**

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

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

**}**

**}**

**else** **if** **(**header**.**Equals**(**"Msg"**))**

**{**

string command **=** message**[**1**];**

**if** **(**command**.**Equals**(**"Stop"**))**

**{**

mT**.**Stop**();**

**}**

**}**

**else** **if** **(**header**.**Contains**(**"GET / HTTP/1.1"**))**

**{**

TcpClient web **=** client**;**

web**.**Close**();**

**return;**

**}**

**else**

**{**

myLogger**.**Warn**(**"No protocol supported"**);**

**}**

**}**

myLogger**.**Warn**(**String**.**Format**(**"The client {0} disconnected"**,** client**.**Client**.**RemoteEndPoint**));**

stream**.**Close**();**

**}**

## 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**

**{**

**return** 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.

**private** Vector3 CreateVector**(**string data**,** int id**)**

**{**

**...**

**return** **new** Vector3

**{**

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**)),**

Id **=** id

**};**

**}**

## 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.

## Client

## 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

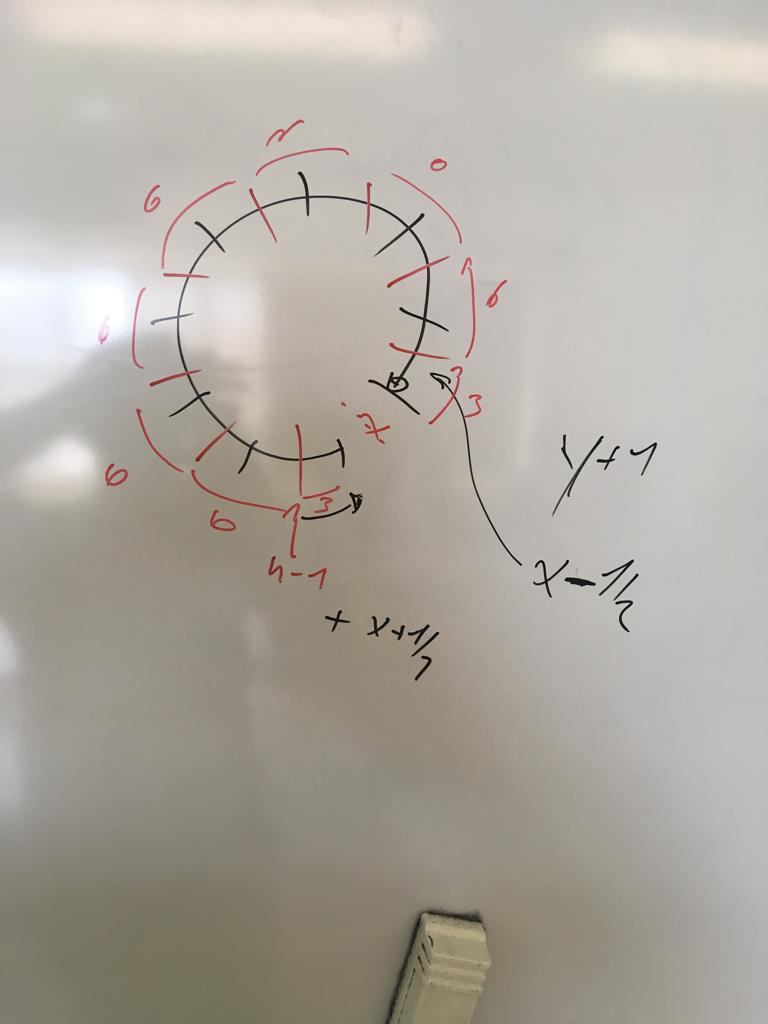
distanza ricavata dallo scanner.

…

    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.



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));

<foto risultato>

## 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 sapere se 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 viene inviato al server un messaggio che l’Arduino 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>");

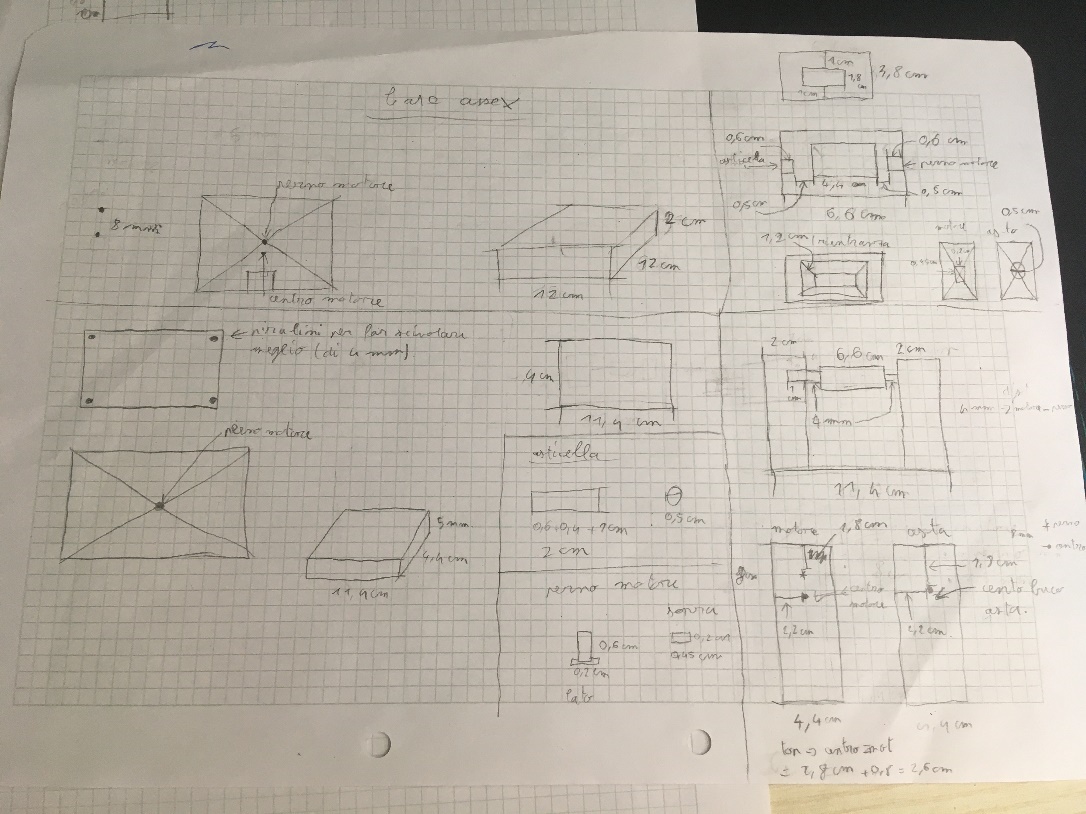
    }

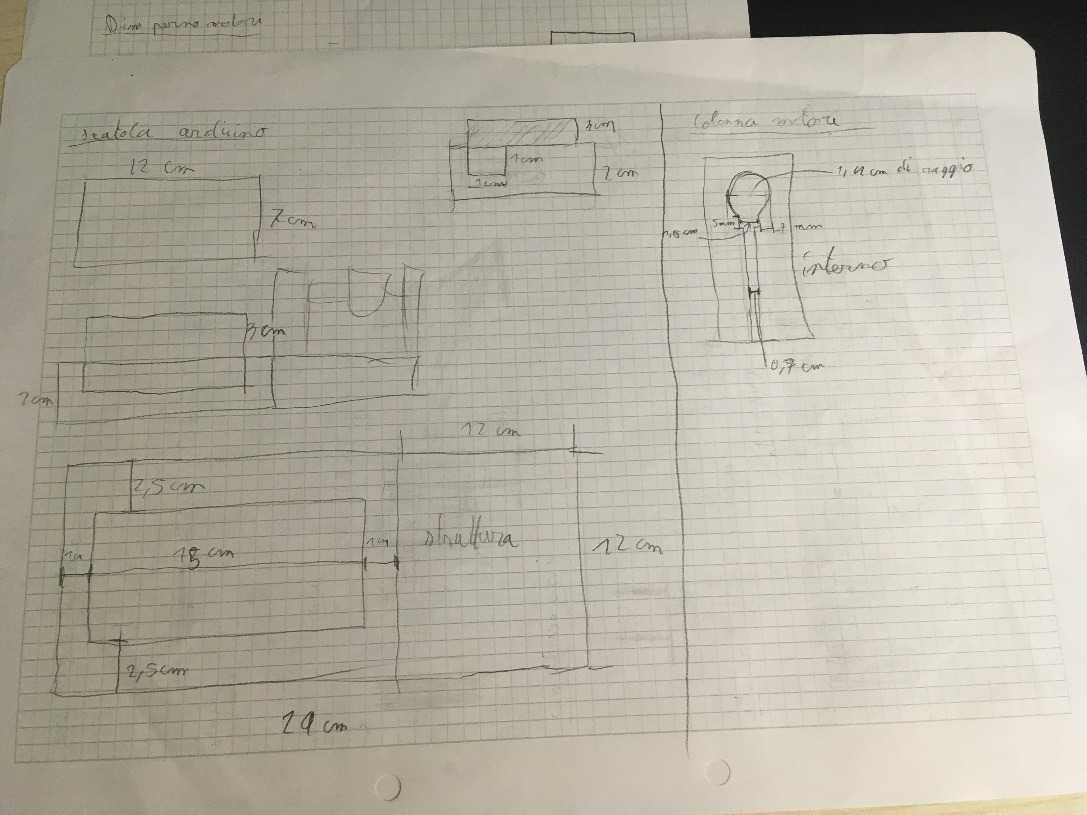
}

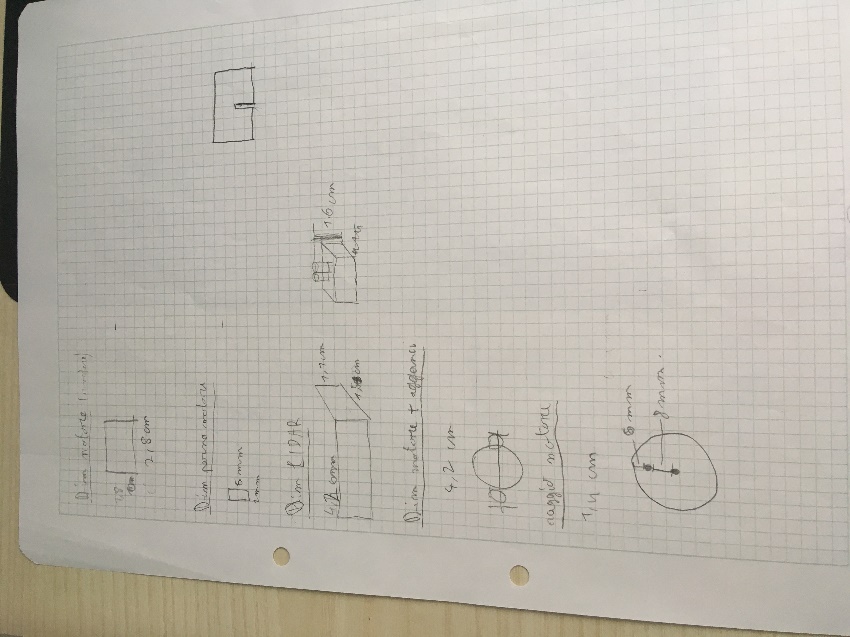
## Struttura HW

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.



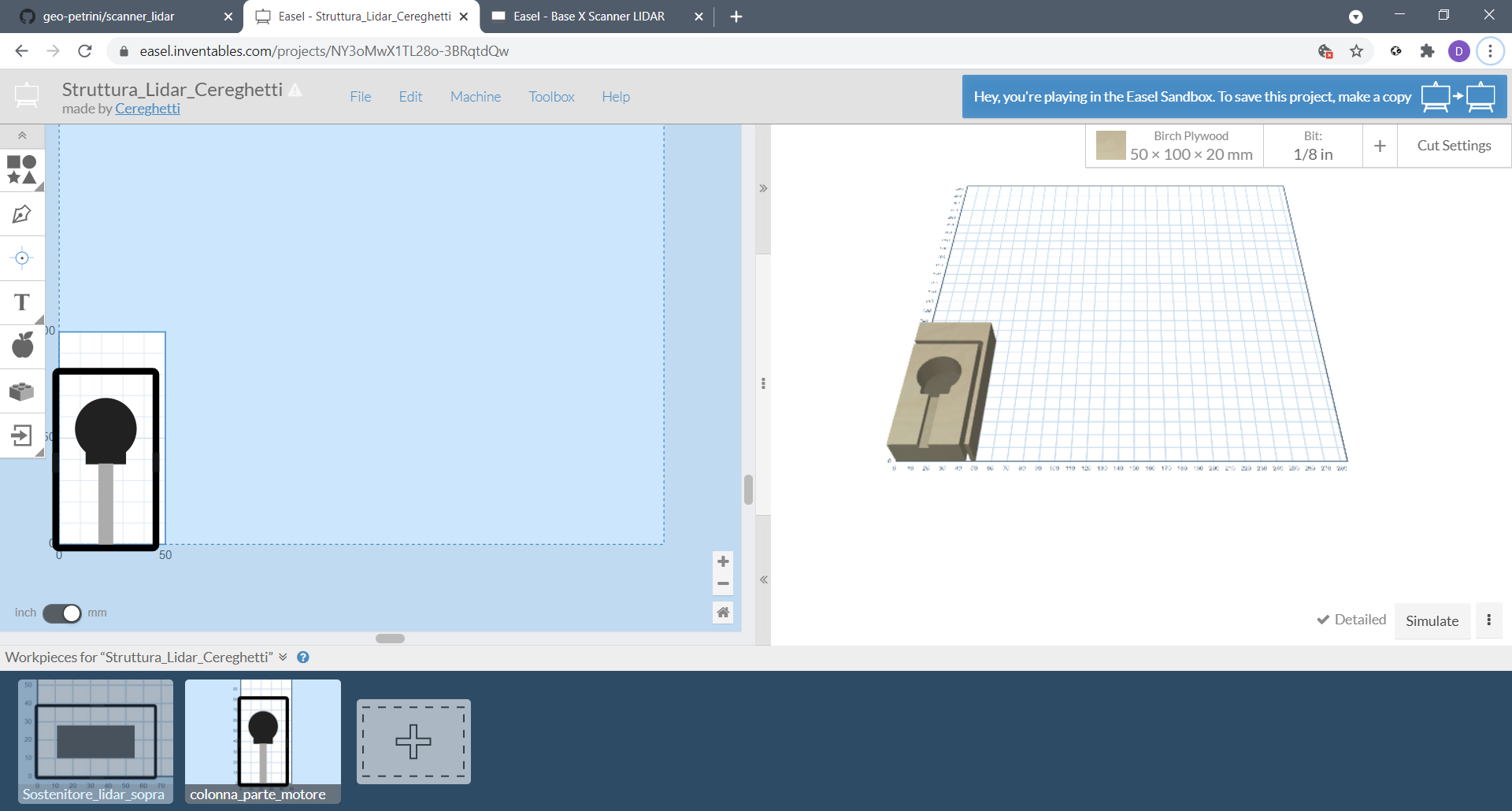
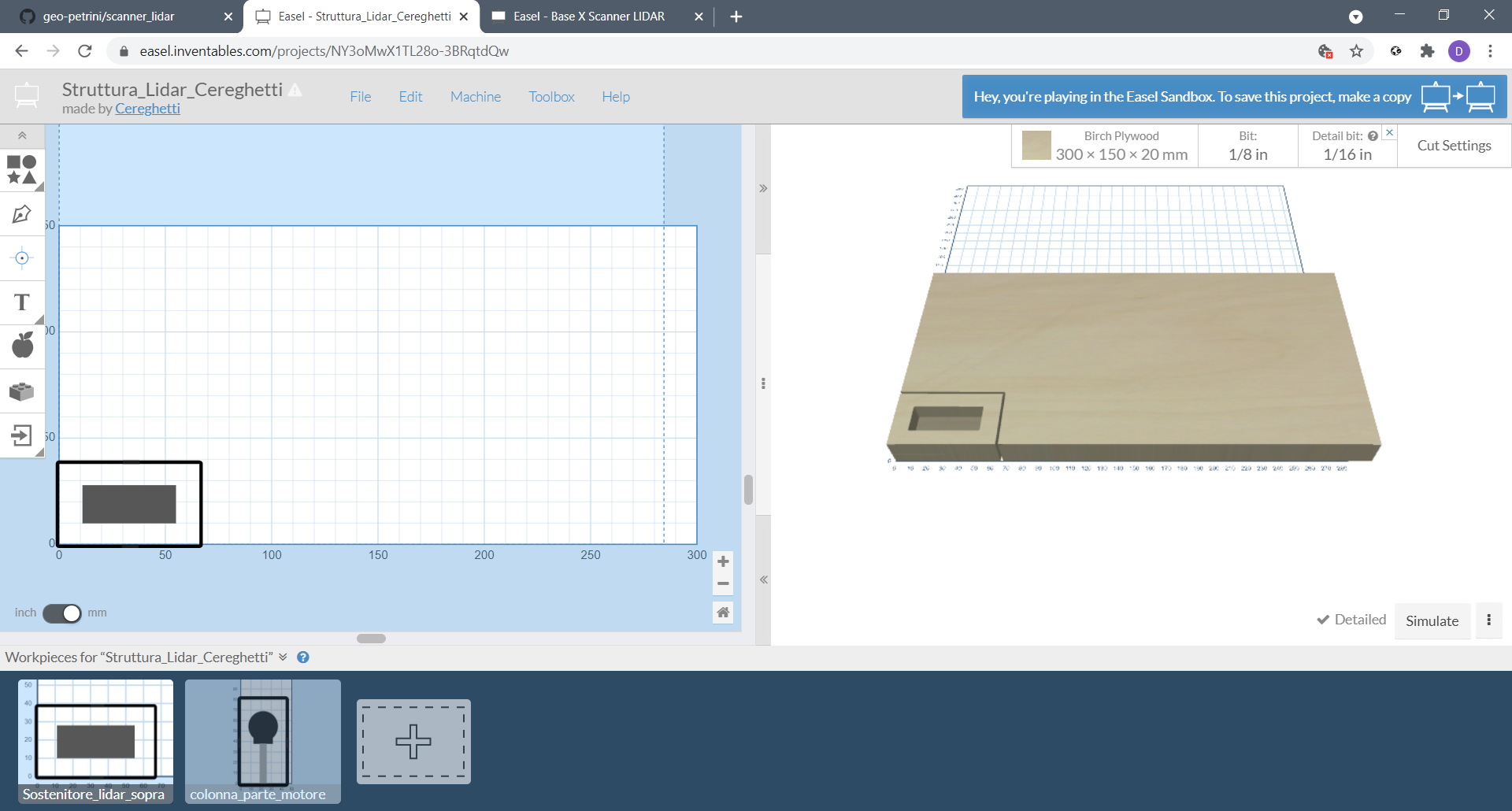




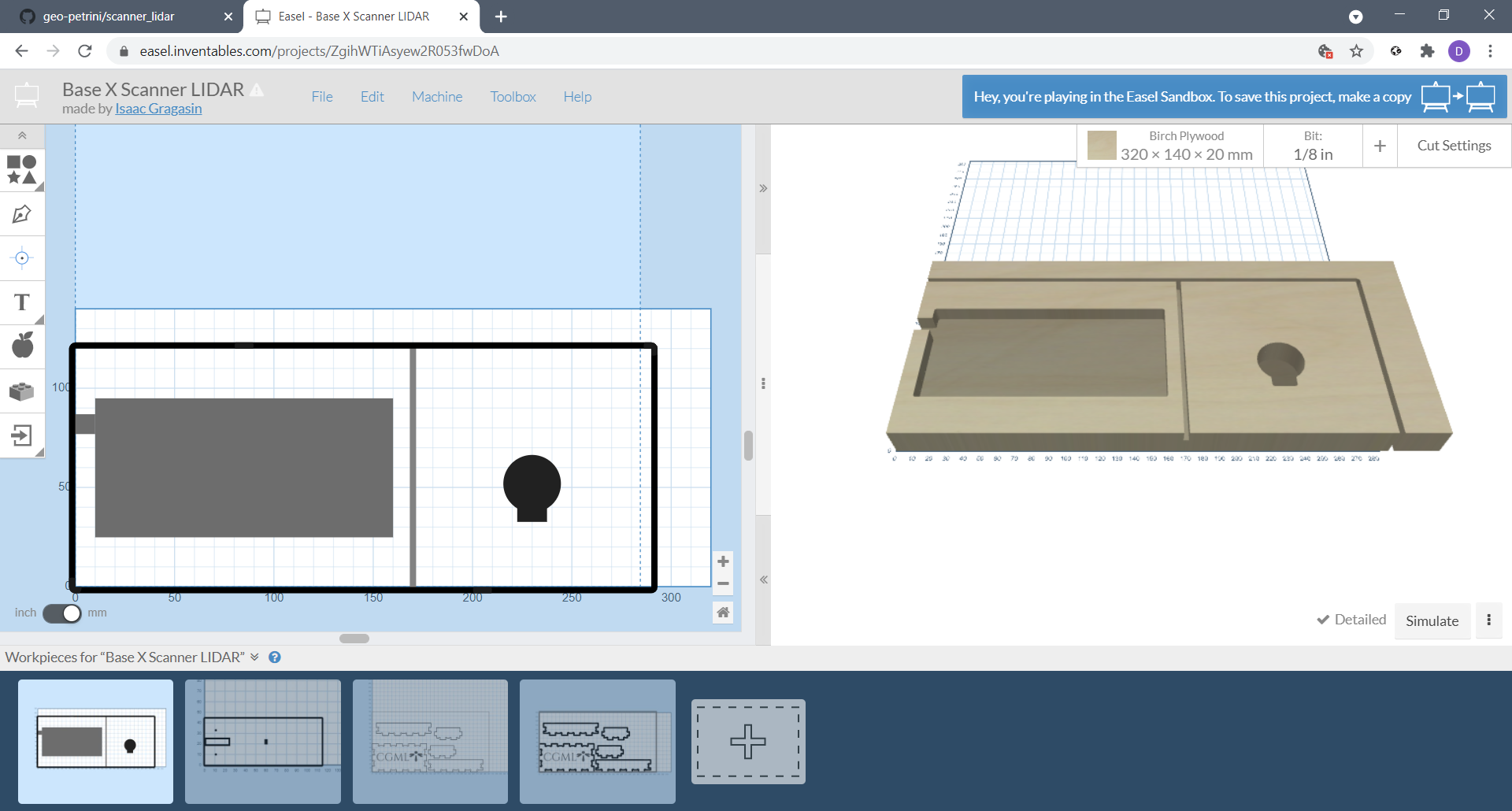
Il professore Petrini si è gentilmente offerto per la realizzazione dei pezzi, lui ci ha consigliato di farli in legno perché è un materiale facilmente modellabile, ci ha chiesto solamente di fare i disegni con un Easel, così che possa tagliarli.

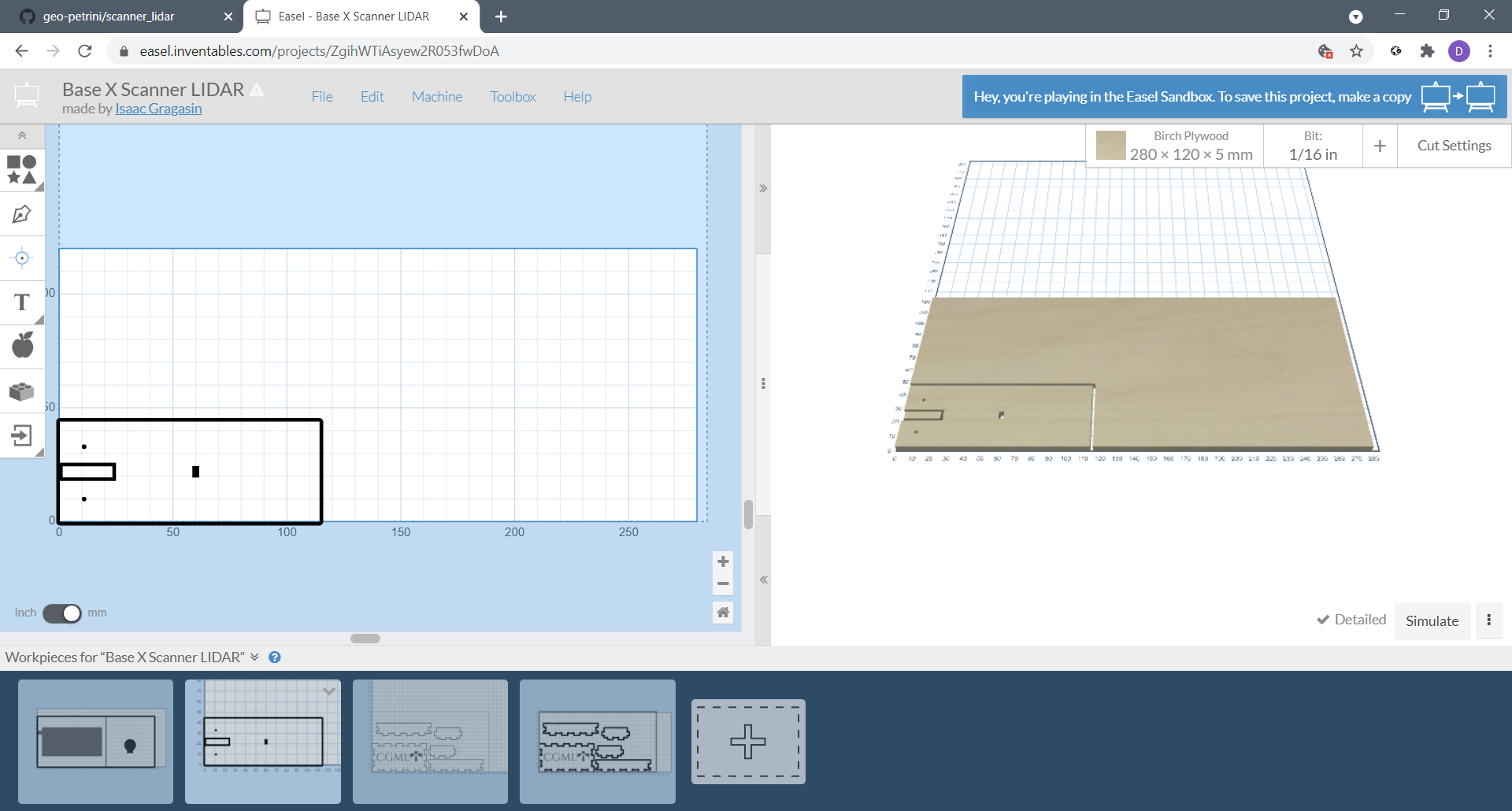
I disegni elettronici sono usciti nei seguenti modi:

* Parte di sostegno per lo scanner Lidar

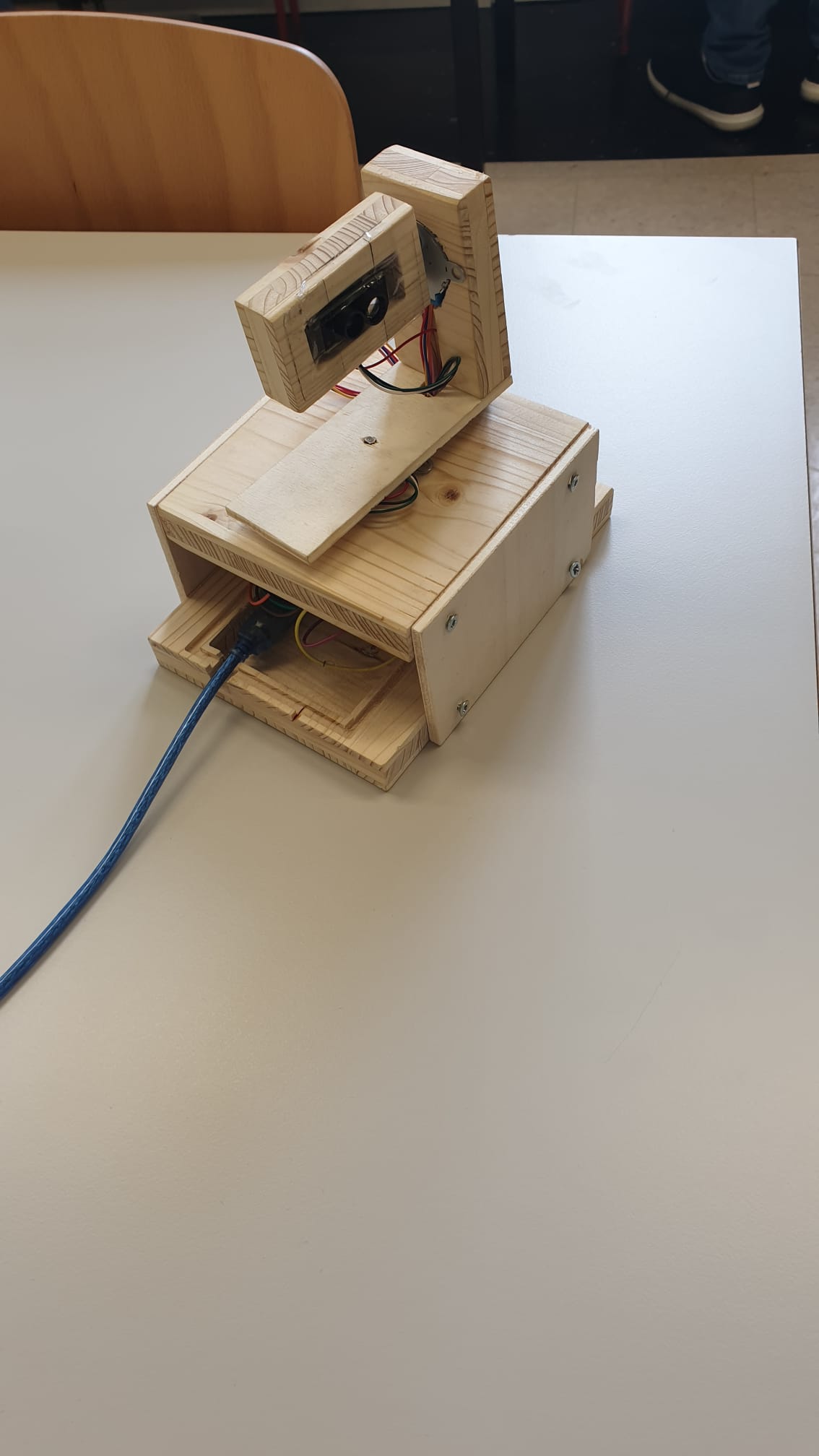


* Parte della base per lo scanner



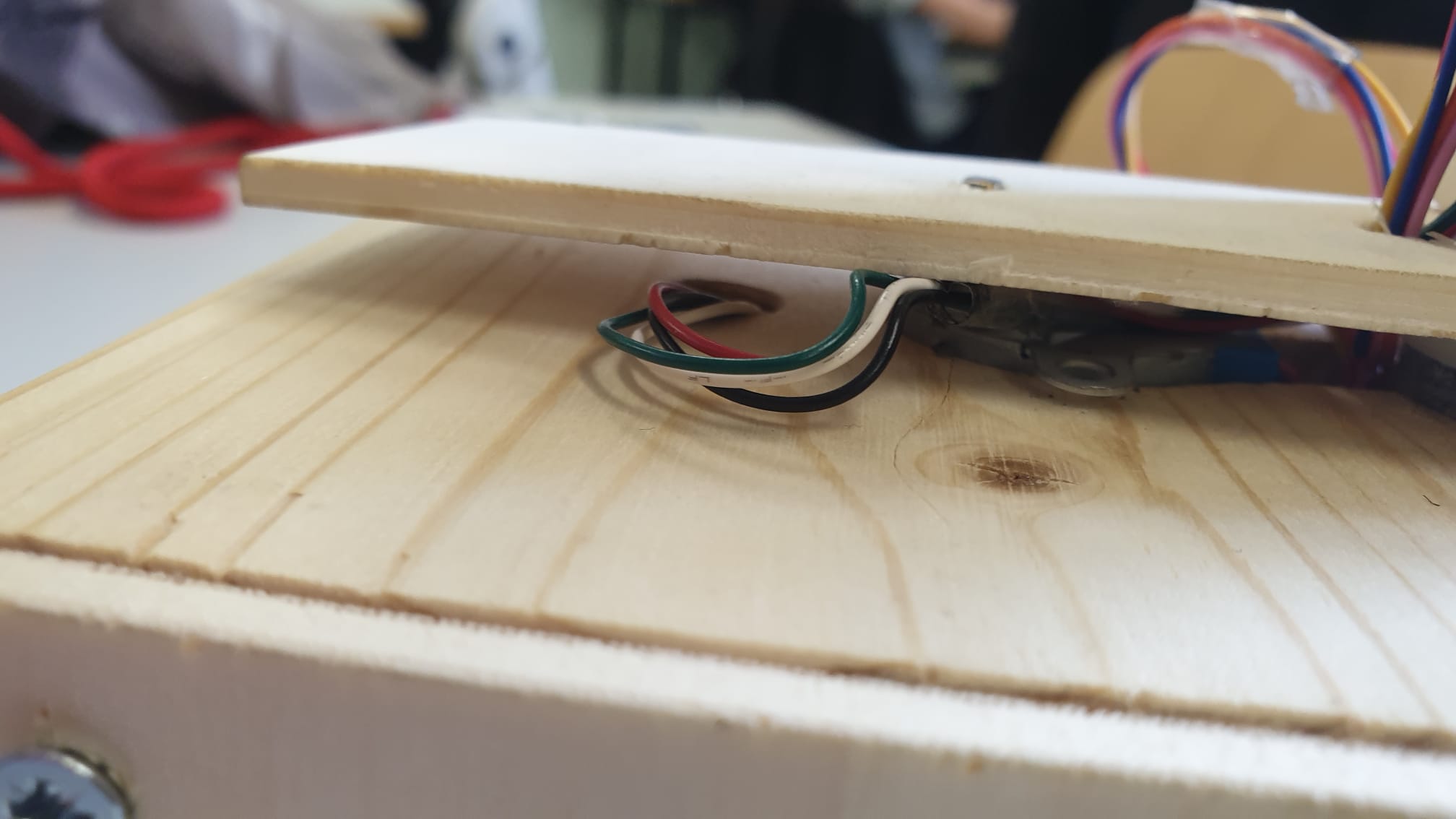


Questa seguente immagine rappresenta il risultato finale dell’assemblamento tra il circuito di arduino e la sua dedicata struttura.



## 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.





# 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. | | |

## Risultati test

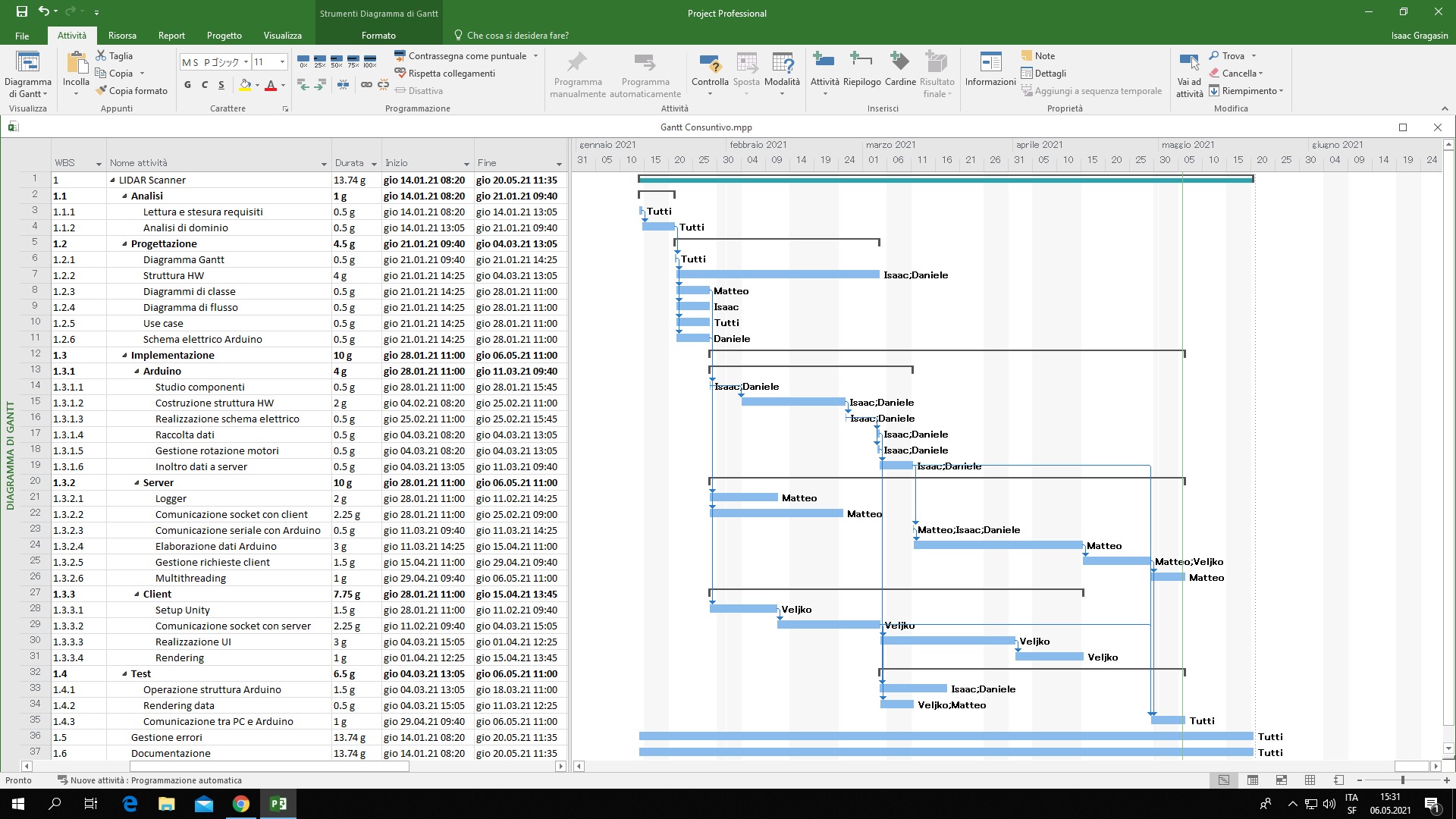
|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | 1  REQ-HW-1 | **Descrizione:** | Scansione a 360° sull’asse orizzontale |
| **Risultati:** | il motore esegue una rotazione di 360° sull’asse orizzontale, questo è più visibile grazie al foglio di test attaccato al perno. |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

# Consuntivo



# Conclusioni

## Sviluppi futuri

## Considerazioni personali

# Bibliografia

## Bibliografia

## Sitografia

# Allegati