# Abstract

This paper summarizes my research activity in the field of Computer Vision, focusing mainly on Lidar technologies. Compared to 2D imaging systems, 3D mapping can provide accurate information about the surrounding environment. The main objective of my work is to design a static Lidar-based 3D scanner capable of displaying a reconstruction of scenery. Furthermore, a mechanical implementation of a Lidar scanner is presented. A test bench has been designed aiming to acquire precise distance measurements and project them into a high-resolution point cloud. The goal is to demonstrate the algorithm behind the laser time-of-flight 3D reconstruction technique. The yawing scan method is used for the current application. The scanning method is based, on the rotation of the sensor around its vertical axis focusing on a vertical scan plane. A microcontroller application is designed to acquire data from the Lidar at different positions. Since the Lidar is a one-dimensional sensor, the mechanical setup must be capable of achieving the second and third dimensions. In the same idea, two motors are utilized to create a three-dimensional scanning system. One motor rotates the Lidar around the upright z-axis, while the second motor moves the sensor along the vertical field of view. Stepper motors have been selected for the application due to their known angle/step resolution. The information from the scanner is collected based on the spherical coordinates: θ - polar rotation angle, φ – azimuthal angle and r – Euclidian distance. An accelerometer module has been fixed on a plane orthogonal to the Lidar plane. The accelerometer is part of a control system designed to set the sensor at an initial azimuthal angle, prior to the scan. A PID control algorithm has been developed for this task. A desktop application has been designed to collect the information from the sensor. It includes a graphical user interface used for setting the desired scan parameters or commencing a scan routine. The communication between the desktop program and the microcontroller is realized using the serial bus. A data processing application takes the acquired measurements and calculates the Cartesian coordinates of each pixel based on the parameters of the scan. A 3D reconstruction of the scan field of view is generated. Multiple experiments have been conducted to test the performance of the system. The scan scenarios include human silhouette detection, object form and orientation, indoor mapping and object details detection. The systematic errors of the system have also been calculated. All the described methods are implemented and tested on a static, non-continuous Lidar scanner, designed using an Arduino Uno board.

# **Content**

Table of Contents

[1 Abstract 1](#_Toc44935072)

[2 Content 2](#_Toc44935073)

[3 Rezumat în limba română 4](#_Toc44935074)

[3.1 Stadiul actual 4](#_Toc44935075)

[3.2 Fundamentare teoretică 5](#_Toc44935076)

[3.3 Implementare 7](#_Toc44935077)

[3.4 Rezultate experimentale 9](#_Toc44935078)

[4 State of the art 10](#_Toc44935079)

[4.1 Current Lidar market 10](#_Toc44935080)

[4.2 Lidar in autonomous vehicles 11](#_Toc44935081)

[4.3 Lidar mapping solutions 12](#_Toc44935082)

[5 Theoretical fundamentals 13](#_Toc44935083)

[5.1 Photodetectors for Lidar 13](#_Toc44935084)

[5.2 3D scanning methods 18](#_Toc44935085)

[5.3 Terrestrial Lidar scanning 21](#_Toc44935086)

[5.4 Lidar Geometric Model and Calibration 22](#_Toc44935087)

[6 Implementation 24](#_Toc44935088)

[6.1 Data processing application 24](#_Toc44935089)

[6.2 Desktop application 27](#_Toc44935090)

[6.2.1 Toolbar 28](#_Toc44935091)

[6.2.2 Form Panel 29](#_Toc44935092)

[6.3 The microcontroller application 30](#_Toc44935093)

[6.3.1 Serial communication 31](#_Toc44935094)

[6.3.2 I2C communication 32](#_Toc44935095)

[6.3.3 Lidar Lite V3 33](#_Toc44935096)

[6.3.4 Stepper Motor 28 BYJ-48 35](#_Toc44935097)

[6.3.5 MPU 6050 module 38](#_Toc44935098)

[6.3.6 Set parameters routine 40](#_Toc44935099)

[6.3.7 Yawing Scan routine 40](#_Toc44935100)

[7 Experimental results 41](#_Toc44935101)

[7.1 3D scanner setup 41](#_Toc44935102)

[7.2 Scenario 1: Human Scan 42](#_Toc44935103)

[7.3 Scenario 2: Rectangular object scan 43](#_Toc44935104)

[7.4 Scenario 3: Interior room mapping 45](#_Toc44935105)

[7.5 Scenario 4: Object details detection 50](#_Toc44935106)

[7.6 Errors 51](#_Toc44935107)

[8 Conclusions 52](#_Toc44935108)

[9 References 53](#_Toc44935109)

[10 Appendix 55](#_Toc44935110)

[10.1 Appendix 1 - Matlab script 55](#_Toc44935111)

[10.2 Appendix 2 - Java Send Parameters Routine 57](#_Toc44935112)

[10.3 Appendix 3 – Electrical schematic 59](#_Toc44935113)

[10.4 Appendix 4 – Serial communication 60](#_Toc44935114)

[10.4.1 Header file 60](#_Toc44935115)

[10.4.2 C file 61](#_Toc44935116)

[10.5 Appendix 5 – I2C communication 63](#_Toc44935117)

[10.5.1 Header file 63](#_Toc44935118)

[10.5.2 C file 64](#_Toc44935119)

[10.6 Appendix 6 – Lidar Lite V3 66](#_Toc44935120)

[10.6.1 Header file 66](#_Toc44935121)

[10.6.2 C file 68](#_Toc44935122)

[10.7 Appendix 7 – Stepper motor 71](#_Toc44935123)

[10.7.1 Header file 71](#_Toc44935124)

[10.7.2 C file 72](#_Toc44935125)

[10.8 Appendix 8 – MPU 6050 73](#_Toc44935126)

[10.8.1 Header file 73](#_Toc44935127)

[10.8.2 C file 74](#_Toc44935128)

[10.9 Appendix 9 – Controller 78](#_Toc44935129)

[10.9.1 Header file 78](#_Toc44935130)

[10.9.2 C file 79](#_Toc44935131)

[10.10 Appendix 10 – Scanner 82](#_Toc44935132)

[10.10.1 Header file 82](#_Toc44935133)

[10.10.2 C file 83](#_Toc44935134)

[10.11 Appendix 11 – Main program 86](#_Toc44935135)

# **Rezumat în limba română**

Lucrarea de față reprezintă proiectul de diplomă a autorului, student al facultății de Electronică, Telecomunicații și Tehnologia informației, Universitatea Tehnică din Cluj Napoca. Proiectul cuprinde activitatea de cercetare și proiectare a autorului în domeniul viziunii artificiale. Tehnologia Lidar a fost utilizată pentru a dezvolta un montaj capabil de reconstrucție 3D a mediului înconjurător. În comparație cu sistemele imagistice 2D, sistemele de mapare 3D oferă informații precise în legătură cu împrejurimile sale. O implementare mecanică a unui scanner 3D este prezentată în conținutul lucrării. Cele patru capitole: Stadiu actual, Fundamentare teoretică, Implementare și Rezultate experimentale reflectă contribuția autorului. În continuare este prezentat pe scurt conținutul fiecărui capitol.

## Stadiul actual

Acest capitol fixează referința în care se situează proiectul de diplomă. Lidar este o abreviere de la Light Detection and Ranging și reprezintă o tehnologie de detecție care utlizeaza unde laser. Senzorul emite o undă luminoasă într-o anumită direcție. Dacă un obiect sau o suprafață a fost detectată, atunci unda se întoarce înapoi. Dispozitivul poate calcula distanța folsindu-se de timpul necesar undei laser să se întoarcă la senzor. Această metodă de detecție se numește în literatură de specialitate Time of Flight, pe scurt TOF. Prin colectarea mai multor măsurători de distanță din mediul înconjurător, sistemul Lidar va emite la ieșire un set de date numit nor de puncte (Point Cloud). Algoritmi complexi pot fi implementați pentru a realiza o reconstrucție 3D cât mai similară cu realitatea.

O creștere semnificativă este așteptată în piața tehnologiei Lidar, de la USD 844 million în 2019 la USD 2273 million până în 2024, conform experților de la Markets and markets [7]. Același studiu susține că modelul solid-state Lidar, sistem aflat în întregime pe un singur chip, va crește cel mai mult datorită avantajelor pe care le are în comparație cu sistemele mecanice. De asemenea, aplicațiile Lidar pe distanțe medii, între 200 m și 500 m, vor fi cele mai căutate în viitorul apropiat. Sistemele automotive de siguranță încorporează senzori Lidar cu scopul detecției de obiecte și evitării unei posibile coliziuni. Același studiu de piață demonstrează că zona Asia-Pacific va avea cea mai mare dezvoltare a pieței Lidar datorită proiectelor mari de infrastructură care se desfășoară.

În domeniul automotive, sistemele Lidar sunt montate pe mașini pentru a oferi asistență șoferului în condiții critice. Aceste metode de detecție sunt prezente începând cu nivelul 3 de conducere autonomă. Mediul înconjurător este monitorizat constant, iar în condiții de siguranță automobilul preia controlul anumitor funcționalități precum frână, accelerație sau schimbarea de benzilor. În aceeași ordine de idei, senzorul Lidar, alături de o camera video și de senzorul Radar formează ansamblul de viziune artificială a unei mașini. Lidarul este preferat față de Radar în condiții controlate unde precizia este importantă, pe când Radarul este mai util în condiții meteorologice extreme. De asemenea, camerele video oferă cel mai înalt nivel de detalii. Informația colectată de la cele 3 componente este corelată utilizând un procesor montat în bordul automobilului.

În domeniul cartografiei, sistemul Lidar este montat pe un vehicul aerian fără pilot, cunoscut sub numele de UAV. Sistemul laser poate să își schimbe unghiul de achiziție în timpul zborului. Cartografierea 3D a unor scene este utilizată în domenii precum construcțiile civile, silvicultură, operațiuni miniere. Tendința principala în domeniul cartografiei 3D este reducerea masei sistemului Lidar pentru a spori eficiența și timpul de zbor al vehiculelor UAV.

## Fundamentare teoretică

Acest capitol conține analiza și comentariul a câteva soluții care utilizează tehnologia Lidar. Sistemul Time of Flight (TOF) de măsurare a distanței este dezvoltat de către compania Hamamatsu și este format din următoarele elemente: un laser în domeniul infraroșu, un fotodetector, un circuit timer și un circuit de măsurare a timpului. Compania Hamamatsu oferă produse unde fotodetectorul utilizat ar putea fi una din următoarele: diodă PIN, fotodiodă cu avalanșă (APD) sau un fotomultiplicator de siliciu (Multi Pixel Photon Counter). În continuare este prezentată structura și modul de funcționare a fiecărui fotodetector.

Modul de funcționare din spatele oricărui fotodetector se bazează pe generarea unui curent în urma absorbției de energie luminoasă. În fotodetectorii semiconductori, fotocurentul este generat prin intermediul efectului fotoelectric intern, folosind o joncțiune P-N sau o joncțiune P-I-N. De asemenea, pe lângă fotocurent poate apărea un curent de întunericcare nu este influențat de intensitatea fasciculului de lumină incident. Fotodioda PIN este un dispozitiv care are o regiune nedopata (intrinsecă) între zona dopată pozitiv și zona dopată negativ. Fotonii sunt absorbiți în zona intrinsecă, generând purtători de sarcină care contribuie la intensitatea fotocurentului. În aplicații Lidar, fotodiodele sunt utilizate în modul fotoconductiv. O tensiune inversă este aplicată diodei, iar fotocurentul este măsurat de la anod la catod. Cea mai simplă soluție de măsurare a intensității curentului este conectarea unei rezistente în serie. Căderea de tensiune pe rezistență duce la o valoarescăzută a tensiunii pe dioda și la o creștere a intensității fotocurentului. Acest fenomen duce la o încărcare sau o descărcare a capacității senzorului de fiecare dată cand apare o schimbare în intensitatea fluxului luminos. Astfel, lățimea benzii este limitată de circuitul R-C format din rezistență și capacitatea diodei. Pentru a nega efectele capacității se utilizează un amplificator transimpedanță care menține constantă tensiunea inversă pe dioda.

Fotodiodă cu avalanșă este un dispozitiv semiconductor capabil să detecteze lumina. Configurația internă este bazată pe o structură p+-i-p-n+. Zona intrinsecă reprezintă zona de conversie a energiei luminoase incidente. Electronii și golurile sunt excitate de către fotonii absorbiți. Un câmp electric se formează, accelerând electronii și golurile în zona de avalanșă, generând, astfel, purtători secundari. Acest proces se mai numește efectul de avalanșă al fotodiodei, prin intermediul căruia se amplifică intensitatea fotocurentului dela ieșire. Cel mai important aspect al fotodiodei cu avalanșă este câștigul intern al dispozitivului la aplicarea unei tensiuni inverse. Principalele avantaje ale fotodiodei cu avalanșă față de dioda PIN reprezintă lățimea de bandă și o responsivitate îmbunătățită. Fotodiodele cu avalanșă din siliciu funcționează pentru lungimi de undă între 400 nm și 1000 nm. Responsivitatea maximă este atinsă între 600 nm și 800 nm.

Fotomultiplicatorul de siliciu este un un fotodetector solid-state care folosește o matrice de fotodiode cu avalanșă. Aceste fotodiode cu avalanșă funcționează în modul Geiger. Modul Geiger de funționare a unei fotodiode înseamnă aplicarea unei tensiuni inverse puțin mai mare decât tensiunea nominală de avarie a dispozitivului. Astfel, o singură pereche electron-gol va declanșa un efect puternic de avalanșă, generând un fotocurent amplificat. Un rezistor este montat în serie cu fiecare fotodiodă pentru a reduce tensiunea pe fotodiodă în momentul avalanșei. După un timp de recuperare de câteva zeci de nanosecunde, circuitul este pregătit din nou să detecteze fotoni. Acest proces permite fotodiodei cu avalanșă să fie folosită ca un numărător de fotoni. Amplitudinea semnalului de la ieșire a unei celule este aceeași indiferent de numărul de fotoni detectați. Aceste este motivul pentru care se folosește o matrice de fotodiode. Metoda de estimare a numărului de fotoni detectați se bazează pe integrarea pe o perioadă de timp a sarcinii electrice a fiecărei celule. Un vârf va fi detectat în domeniul de corelație, cu o amplitudine care variază in funcție de numărul de fotoni detectați.

Un scanner 3D a fost proiectat la Institutul de Ingineria Sistemelor de la Universitatea din Hannover, Germania. Această lucrare a fost prezentată la conferința: 14th International Conference on Control Systems and Computer Science în 2003 [30]. Autorii articolului sunt Oliver Wulf și Bernardo Wagner. Scopul cercetării este îmbunătățirea scannerelor 3D pe bază de laser pentru roboți industriali. Principalele aplicații ale proiectului de cercetare constau în sisteme de siguranță pentru medii industriale, automatizarea fabricilor și platforme mobile precum reboti de serviciu. Montajul experimental a fost construit folosind un senzor 2D Lms 200, produs de compania germană Sick. A treia dimensiune a sistemului este obținută prin intermediul unui servomotor. Articolul prezintă mai multe metode prin care se poate realiza o scanare 3D, precum rolling scan, pitching scan, yawing scan și yawing scan top. Metoda de scanare aleasă pentru proiectul de diplomă este yawing scan deoarece este o metodă robustă de detecție. Yawing scan se bazează pe învârtirea senzorului în jurul axei sale. Mai apoi, senzorul se va deplasa de-a lungul planului vertical. În aceeași ordine de idei, articolul prezintă o metodă de deducere a coordonatelor sferice a fiecărui pixel prin asocierea unui indice de timp fiecărei măsurători. Această metodă nu este folosită în proiectul de diplomă deoarece montajul experimental este realizat din motoare pas-cu-pas. Motoarele pas-cu-pas oferă un control precis al unghiului de rotație.

O altă soluție bazată pe tehnologia Lidar a fost publicată în jurnalul Agricultural and Forest Meteorology, volumul 149, publicația 9, 1 Septembrie 2009, paginile 1505-1515 [31]. Lucrarea prezintă o metodă de reconstrucție 3D a zonelor agricole. Un senzor Lms 200 fabricat de compania Sick este montat pe un tractor care mișcă cu o viteză constantă. A treia dimensiune a sistemului este derivată din viteza constantă a vehiculului. Datele sunt transmise serial către o unitate centrală pentru a fi procesate. Măsurătorile sunt transformate în coordonate carteziene. Împreună cu un sistem GPS, un model 3D al livezilor este realizat. Primele experimente au fost realizate într-un laborator de cercetare. Montajul experimental a fost fixat pe o platforma mobilă. Rezultatele experimentale oferă o analiză cantitativă a vegetației. Atât dimensiunile cât și formele tulpinelor pot fi studiate de un expert în silvicultură. O interfață grafică de utilizator a fost realizată în Matlab. Astfel, parametrii unui experiment pot fi modificați fără a interveni în programul de baza a unității de comandă. Atât transmisia serială a datelor cât și interfața de utilizator sunt concepte care sunt aplicate în proiectul de diplomă.

Al treilea articol de referință a fost publicat în revista științifică Sensors pe 20 mai 2020. Titlul articolului este “Geometric Model and Calibration Method for a Solid-State LiDAR”, iar autorii sunt: García-Gómez, Pablo & Royo, Santiago & Rodrigo, Noel & Casas, Josep [32]. Lucrarea prezintă o metodă de modelare geometrică a unui system Lidar solid-state. Principalul avantaj al unui sistem solid-state față de unul mecanic este faptul că toate componentele se află pe un singur chip. În acest fel, aceste sisteme sunt imune la vibrații sau șocuri mecanice. Câmpul de vizualizare a unui Lidar într-un sistem Time of Flight este format din toți vectorii unitari ai direcției de scanare. De asemenea, vectorul direcției de scanare trebuie să aibă o spațiere constantă pentru a minimaliza erorile de deformare. Modelul optic al sistemului de achiziție TOF se bazează pe legea lui Snell, presupunând că raza incidentă și raza reflectată se află în același plan de incidență. Sistemul de referință a Lidarului este considerat a fi o baza ortonormala. Astfel, orice punct care se află în spațiul euclidian rezultat poate fi exprimat ca fiind o combinație liniară a vectorilor de rotație. Articolul prezintă un model matematic de transformare a coordonatelor sferice în coordonate carteziene. Modelul ia în considerare unghiul polar de rotație, unghiul azimutal și distanța euclidiană de la senzor până la un pixel. Un algoritm de calibrare este apoi aplicat.

..

## Implementare

Acest capitol descrie proiectarea și implementarea unui sistem de detecție Lidar discontinuu. Dispozitivul dispune de metoda de scanare yawing scan ca fiind singura metodă de referință. Implementarea soluției este împărțită în trei aplicații diferite: aplicația de procesare a datelor, aplicația desktop și aplicația microcontroller. Implementarea este prezentată folosind o abordare top-down. Prima dată este prezentată procesarea datelor și crearea unui nor de puncte, iar la final este explicată funcționalitatea componentelor hardware.

Aplicația de procesare a datelor este dezvoltată în programul Matlab și are rolul de a reprezenta grafic rezultatele unei scanări. În urmă unei scanări aplicația desktop va genera două fișiere: *results.txt* și *params.txt*. Primul fișier conține măsurătorile de distanță colectate de către senzorul Lidar. Datele sunt reprezentate ca numere întregi fără semn, ele fiind măsurători de distanță în centimetri. Informația este împărțită în rânduri și coloane. Rezultatele de pe același rând aparțin aceleași întoarceri în jurul axei z. Coordonatele sferice descriu poziția unui punct în funcție de următorii parametrii: ρ – distanță euclidiană de la origine, θ – unghiul polar de rotație și φ – azimutul. Primul motor pas-cu pas rotește Lidarul în jurul axei z, modificând unghiul polar de rotație, θ. Al doilea motor pas-cu-pas rotește senzorul de-a lungul axei z pozitive, modificând azimutul, φ. Coordonatele sferice a fiecărui punct detectat în urma unei scanări sunt deduse folosind parametrii de scanare și măsurătorile de distanță colectate. Coordonatele sferice sunt transformate in coordonate carteziene, iar apoi sunt redate într-un grafic 3D.

Aplicația desktop conține o interfață grafică de utilizator și comunică serial cu aplicația microcontroller. Programul este scris în limbajul de programare Java, folosind mediul de programare Eclipse. Prima componentă principală a interfeței se numește Toolbar și conține 4 butoane și o listă verticală. Primul buton se numește Scan Ports și este utilizat pentru a căuta porturi seriale valabile. La finalul căutării, lista verticală este actualizată cu numele tuturor porturilor seriale la care s-ar putea conecta aplicația. Butonul Open Port este folosit pentru a stabili o conexiune cu portul serial selectat în lista verticală. Pentru a stabili o conexiune cu aplicația microcontroller, trebuie selectat portul serial unde este conectat microcontrollerul. Butonul Close Port închide conexiunea dintre aplicația desktop și portul serial. Butonul Start Scan este utilizat pentru a semnaliza începerea unei scanări. Când acest buton este apăsat, parametrii de scanare sunt salvați în fișierul params.txt, portul serial este setat în modul de transmisie și aplicația desktop trimite un caracter ASCII care semnalizează începerea unei scanări. Acest caracter va comanda aplicației microcontroller să înceapă o scanare. Portul serial este setat în modul de recepție, iar aplicația desktop este pregătită să primească date de la scanner. Toate rezultatele sunt salvate în fișierul results.txt. A doua component a interfeței se numește Form Panel. Această componentă îi oferă utilizatorului posibilitatea de modificare a parametrilor de scanare. Sunt 6 parametrii de scanare: yaw angle resolution, maximum yaw angle, pitch angle resolution, maximum pitch angle, pitch start position și Lidar configuration. Yaw angle resolution modifică distanța dintre doi pixeli pe axa orizontală a câmpului de vizualizare. Maximum yaw angle setează lățimea câmpului de vizualizare a unei scanări. Pitch angle resolution modifică distanța dintre doi pixeli pe axa verticală a câmpului de vizualizare. Maximum pitch angle și pitch start position reprezintă unghiul azimuthal de final și de început a scannerului. Diferența lor, luată în valoare absolută reprezintă înălțimea câmpului de vizualizare. Lidar configuration înseamnă modul de funcționare a senzorului pentru următoarea scanare. Un buton Set Parameters a fost adăugat. Când acest buton este apăsat, un caracter ASCII este trimis serial către microcontroller pentru a semnaliza rutina de schimbare a parametrilor. Între microcontroller și aplicația desktop a fost dezvoltat un protocol de sincronizare. Aplicația desktop va trimite un byte, după care intră în modul de recepție. Când aplicația microcontroller finalizează toate operațiile cu byte-ul respectiv, îi va semnaliza aplicației desktop că este pregătită să primească date.

Aplicația microcontroller este dezvoltată utilizând placa Arduino Uno, respectiv microcontrollerul At Mega 328P produs de compania Microchip. Frecvența de tact al acestui microcontroller este de 16 MHz. Programul este scris în limbajul de programare C, utilizând mediul de programare Atmel Studio. Senzorul Lidar ales pentru acest proiect este Lidar Lite V3 produs de către compania Garmin. Sistemul comandă senzorul să colecteze măsurători prin intermediul metodei de scanare yawing scan. Două motoare pas-cu-pas 28 BYJ-48 produse de compania Kiatronics au fost selectate pentru montajul experimental. Poziția Lidarului poate fi setată la diferite unghiuri într-un sistem cu două axe de coordinate. Primul motor va roti senzorul în jurul axei sale, modificând unghiul polar de rotire. Al doilea motor va mută senzorul de-a lungul planului vertical al scannerului, modificând azimutul. Un modul MPU 6050, accelerometru va fi folosit pentru a seta senzorul la unghiul azimutal dorit de utilizator la începutul unei scanări.

Aplicația microcontroller comunică serial cu aplicația desktop. Comunicarea serială este realizată utilizând următoarele setări: rată de transfer 78600 biți/secundă, cadru de 8 biți, un bit de stop, fără paritate. Acești parametrii de comunicare sunt setați prin intermediul regiștrilor USART speciali ai microcontrollerului. În aceeași ordine de idei, Lidar Lite V3 și MPU 6050 sunt senzori digitali, iar comunicarea este realizată folosind protocolul de comunicare I2C, cunoscut și sub numele de Two Wire. Acest protocol de comunicare utilizează două magistrale bi-direcționale de comunicare: SDA (magistrala de date) și ȘCL (magistrala de tact). Frecvența de tact pentru comunicarea I2C este setată la 400 kHz. At Mega 328P dispune de regiștrii speciali pentru comunicarea Two Wire. Module software au fost dezvoltate pentru ambele protocoale de comunicare și pot fi găsite la Anexe.

Lidar Lite V3 este un senzor care dispune de un laser infraroșu de 905 nm cu o lățime de undă de 12 x 2 mm. Dispozitivul măsoară distanța prin calcularea timpului de zbor al unei unde luminoase de la transmiterea ei, până la recepția aceluiași semnal. Distanța poate fi calculată utilizând viteza luminii. Senzorul execută o serie de achiziții și verifică dacă semnalul recepționat coincide cu semnalul transmis. Dacă cele două semnale au aceeași semnătură, rezultatul este stocat într-un domeniu de corelație, iar următoarea achiziție va fi însumată rezultatului anterior. Acest proces este repetat până când senzorul ajunge la limita maximă de achiziții, urmând a fi integrate. Un vârf apare în domeniul de corelație, determinând puterea semnalului recepționat. Dacă puterea semnalului depășește un prag, atunci măsurătoarea este considerată validă și senzorul va stoca distanța măsurată în cm. În caz contrar, senzorul va raporta valoarea de 1 cm. Pentru a iniția măsurătoare, trebuiesc executați o serie de pași. Valoarea 0x04 este scrisă în registrul Acquisition command (adresa 0x00) și senzorul va iniția achizițiile. Se monitorizează cel mai puțin semnificativ bit din registrul de Status (adresa 0x01). Cât timp acest bit este 1, dispozitivul este ocupat. În momentul in care bitul este 0, măsurătoarea poate fi extrasă din regiștrii de date.

Motorul pas-cu-pas 28 BYJ-48 este un motor unipolar, valabil pentru mai multe tipuri de aplicații. Motorul execută 2048 de pași pentru o rotație de 360°, având o rezoluție de 0.17578125°/ pas. Acest motor a fost ales în două exemplare pentru acest proiect datorită rezoluției sale și a costului redus. Două circuite de comandă ULN2003 bazate pe configurația Darlington sunt utilizate pentru a controla motoarele. Modulul software de comandă a motoarelor a fost dezvoltat cu algebra booleană. Motorul unipolar 28 BYJ-48 este controlat prin intermediul a 4 pini de comandă conectațila un port digital de ieșire al microcontrollerului. Există 8 combinații logice de comandă, fiecare atribuită unui indice al pasului curent. Pasul curent este un număr fără semn pe 3 biți, având valori între 0 și 7. Hărți Karnaugh au fost utlizate pentru a exprima valoarea logică a fiecărui pin de comandă în funcție de indicele pasului curent. Când pasul curent este incrementat și ajunge la limita superioară, el este resetat la 0. O secvență incrementala de pași va roți axul motorului în sensul invers acelor de ceasornic. În cazul în care acesta este decrementat și ajunge la limita inferioară, el este resetat la 7. O secvență decrementala de pași duce la o rotație în sensul acelor de ceasornic al axului.

Modulul MPU 6050 este un senzor digital care include un accelerometru pe 3 axe. Pentru această aplicație, o sensibilitate mai scăzută a fost aleasă. Valoarea capăt-de-scala a fost setată la ±16g, cu o sensibilitate de 2048 LSB/g. Senzorul poate colecta măsurători de la toate cele 3 axe, iar rezultatele sunt numere întregi cu semn, reprezentate pe 16 biți în codul complement față de 2. Un filtru de mediere este implementat pentru colectarea măsurătorilor de la accelerometru, datorită posibilelor erori. Modulul este atașat montajului experimental cu scopul de a seta senzorul la un unghiul azimutal de start înaintea unei scanări. Un algoritm de cotnrol PID este implementat cu acest scop. Bucla de control a sistemului este formată din: motorul pas-cu-pas ca actuator, scannerul 3D ca proces și accelerometru ca senzor.

Rutina de scanare yawing scan este inițiată dacă aplicația microcontroller recepționează serial caracterul ASCII aferent. În primul rând, aplicația setează senzorul la unghiul azimutal inițial, folosindu-se de sistemul de control. Apoi, scannerul execută o jumătate de întoarcere în direcția opusă primei întoarceri a scanării. Procesul de întoarce este constituit din trei operații: măsurare distanță, transmisie serială a rezultatului, execuție pași motor 1. La finalul unei întoarceri în jurul axei z, al doilea motor is execută pașii, modificând azimutul sistemului. Următoarea întoarcere se va executa în direcția opusă, după care al doilea motor va trece la următorul unghi azimutal. O scanare este finalizată cu succes în momentul în care senzorul ajunge la unghiul azimutal final, selectat de către utilizator. Aplicația microcontroller va semnaliza finalizarea unei scanări către aplicația desktop prin transmiterea caracterului ASCII de inițiere a rutinei.

## Rezultate experimentale

Au fost considerate mai multe scenarii de test pentru a testa performanța sistemului. Primul scenariu de test presupune detectarea unei persoane într-o camera. Acest scenariu este un succes, deoarece silueta umană poate fi identificată în norul de puncte rezultat. Distanța față de senzor și înălțimea persoanei coincid cu rezultatele experimentale ale scenariului de test. A doilea experiment presupune detecția de obiecte dreptunghiulare în spațiu. O cutie dreptunghiulară a fost plasată în fața montajului. Sistemul a detectat cutia în câmpul sau de vizualizare, având dimensiuni foarte apropiate de cele reale. Încă o scanare a fost executată ca parte a acestui experiment. De data aceasta, cutia a fost rotită cu 45 de grade și plasată la aceeași distanță în fața senzorului. Sistemul a reușit să identifice orientarea obiectului față de punctul de referință a Lidarului, plasând obiectul corect în planul 3D rezultat. Al treilea experiment presupune cartografierea unei încăperi și detecția diferitelor obiecte. Trei camere au fost cartografiate ca parte a acestui scenariu de test. Atât rezoluția scanării cât și dimensiunea câmpului de vizualizare au fost îmbunătățite pentru acest experiment. Rezultatele conțin un număr mare de puncte, dar timpii de scanare au crescut considerabil. Al patrulea experiment testează dacă sistemul este capabil să recunoască detalii. Concluzia acestui scenariu este că montajul realizat este capabil să distingă adâncime și forme, dar nu este in stare să identifice detaliile obiectelor. Un ultim experiment a fost realizat pentru a calcula erorile sistematice ale montajului. Senzorul colectează mai multe măsurători de pe o suprafață plata la o distanță cunoscută. Este calculată deviația standard și eroarea medie absolută a rezultatelor.

În concluzie, obiectivele inițiale ale proiectului au fost atinse. Montajul experimental s-a dovedit a fi capabil de a genera nori de puncte cu rezoluție ridicată. Implementarea proiectului ilustrează modul de lucru în dezvoltarea sistemelor de cartografiere 3D. Graficele 3D rezultate descriu cu succes obiecte și spații interioare. Îmbunătățiri pot fi aduse montajului experimental prin selectarea unor componente mai de calitate. De asemenea, implementarea unui algoritm de calibrare poate îmbunătăți rezultatele experimentale.

# **State of the art**

## Current Lidar market

Lidar is an abbreviation for Light Detection and Ranging. It is a technology which uses laser beams to create a 3D model of an environment [5]. A typical Lidar sensor emits a laser beam in a certain direction. If an object is detected, the laser beam will bounce back to the sensor [5]. The device can calculate the distance traveled by the beam, based on its time of flight [5]. By obtaining multiple measurements from the surrounding environment, a Lidar-based system will output a set of data called Point Cloud. Complex computer algorithms can be implemented to create a 3D model of the scene. Once mounted on the rooftop of a vehicle, viewed as a spinning cylinder, the Lidar provides a 360-degree image of the surrounding traffic [6]. Attributed to the rising demand of self-driving cars, the investments in research and design in Lidar market has increased in the past years [6].

A study performed by Markets and markets [7] estimates that the Lidar market will grow from USD 844 million in 2019 to USD 2273 million by 2024, which means a Compound Annual Growth Rate (CAGR) of 18.5% from 2019 to 2024. According to the same study, traditional Lidar systems rely on mechanical moving parts which provide accurate measurements required for automated navigation. Solid-state Lidar solutions are built entirely on a single silicon chip, which eliminate motorized mechanical scanning and makes it more resilient to vibrations [7]. It has been concluded that Solid-state Lidars will have a higher CAGR by 2024 and are expected to be adopted in robotics, drones and cars [7]. Ground-based Lidar systems, installed on Sports Utility Vehicles (SUV) and All-Terrain Vehicles (ATV) are projected to have the highest growth [7]. The study has found that the automotive industry is the main application ground for ground-based Lidar due to rising number of premium cars which have Advanced Driver Assistance Systems (ADAS) [7]. Additionally, medium range Lidar, between 200 m and 500 m, is used in mapping, construction and exploration applications [7]. The experts from Markets and markets expect medium-ranged Lidars to be used the most because they provide the perfect combination of range and cost-effectiveness. Last, but not least, the Asia-Pacific market will see the highest demand for Lidar solutions between 2019 and 2024. The growth of the market for this specific region is attributed to the increase in mapping and infrastructural development operations [7]. The main players in the market are: Teledyne Technologies (US), Hexagon (Sweden), Trimble (US), FARO (US), RIEGL (Austria), SICK AG (Germany), Quantum Spatial (US), Beijing Beiketian Technology Co., Ltd. (China), Velodyne Lidar (US), and YellowScan (France) [7].

Velodyne Lidar, the leading manufacturer in the industry, has received a lot media coverage ever since Ford and Baidu decided to invest 150 million USD in the company for the commercial rights of its laser-based sensors [6]. In 2019, autonomous driving operation has been achieved, based on Velodyne Ultra Puck; the sensor generates point clouds which are used for computing high resolution maps [8]. Velodyne mapping systems have proved to be useful in the robotics industry as well; repetitive tasks can be achieved by automated robots: a Lidar-equipped robot can navigate and provide medical support, limiting human exposure to the Covid-19 virus [9]. Luminar Technologies has announced in July 2019 that it enters the consumer automotive market with USD 250 million raised from financial investors [10]. In the same idea, Volvo, a global leader in automotive safety, has recently announced a partnership with Luminar Lidar with the objective to deliver the first self-driving technology for highways, which will be available for production starting from 2022 [11]. Quanergy Systems is one of the first companies to develop solid-state Lidar solutions and has attracted USD 120 million in 2016 with the goal of strengthening its presence in the European market [6].

## Lidar in autonomous vehicles

An autonomous vehicle can operate without human intervention. They use a combination of sensors and software to navigate safely. Currently, legislation does not cover fully autonomous cars, but partially autonomous vehicles with brake and lane assistance have been developed [1]. A brief description of the five levels of vehicle autonomy has been presented in [2]. Level Zero is equivalent to zero automation where the driver is responsible of executing all the tasks like braking, accelerating and steering [2]. Level One is also named Driver Assistance, the driver is still responsible of accelerating, braking and steering, while the system can assist the driver in certain controlled environments like driving on the highway [2]. Level Two, Partial Automation, the vehicle can assist with certain tasks like braking and steering, but the driver must be aware of the surrounding environment and be ready to assume control at any moment [2]. Starting from Level Three, the vehicle monitors the traffic using sensing elements like the Lidar and cameras [2]. When the conditions are safe, the driver can disengage from keeping track of critical functions, like braking [2]. Even though human attention is not necessary at very low speeds, the driver must be ready to assume control of the vehicle [2]. At Level Four, High Automation, the vehicle can respond to events and control all the critical functions: brake, acceleration, lane switch, signaling and traffic monitoring [2]. More dynamic traffic conditions need human intervention; the driver can switch to this mode only when driving conditions are safe [2]. Level Five represents Complete Automation, with no need for human monitoring [2]. There is no need for a steering wheel or pedals; a person can enter a fully autonomous car, plug in the destination and perform other tasks, while the vehicle executes all the driving-related operations [2].

With the aim of achieving full autonomy, vehicles need systems which monitor their surroundings and make correct decisions based on the input data. Sensors have strengths and weaknesses, which makes them suitable for different applications. The main components of a driver assistance system for a self-driving car are: Processing Unit, Radar, Lidar, Autonomous Emergency Braking and the Sonar [12]. The Lidar provides stereoscopic vision, with 360-degree view and acts as the eye of self-driving vehicles; they are incorporated in rotating mechanical devices which are mounted on the top of the vehicle [13]. The output is an animated 3D representation of the surrounding environment used for collision avoidance [13]. One of the ongoing debates is which sensor is better for applied autonomy, Lidar or Radar? Lidar uses laser light pulses, while Radar uses radio frequency waves; the output of the Lidar is a collection of high-precision laser pulses called point clouds which are then analyzed by an Artificial Intelligence software (AI) [14]. Lidar is preferred over Radar in applications which operate in controlled environments where precision is important (mining operations, agriculture, construction sites), while Radar is used in systems where reaction time is crucial (high-speed travelling) [14]. However, it has been found out that design diversity is crucial in mobility solutions; the use of redundancy and overlapping sensors increase the accuracy of the vehicle’s vision [21]. According to Bonnie C. Baker from Maxim Integrated [21], the components of computer vision systems are cameras (provide the best visual detail), Lidar (the output of the sensor provides shape and depth) and Radar (robust in bad weather conditions, but falls short for higher levels of autonomy). The data output from the three components are combined with an on-board processor. The same type of vehicle processing sensing has been suggested by the engineers from Analog Devices [3]. While, the Lidar is a key component of autonomous cars, it does not function as a standalone device, but rather as an element of an architecture.

## Lidar mapping solutions

Aside from the automotive development, multiple industries benefit from Lidar mapping solutions, including civil engineering, mining and forestry. Lidar mapping is faster and more precise than photogrammetry because aerial photography is restricted in the case of dense forests and low illumination conditions [4]. Integration of Lidar and photogrammetry is possible as digital images provide high spatial resolution, while the Lidar offers an in-depth view of the scenery [4]. The technologies integrated in a mapping system are Inertial Navigation System (INS), Laser, and Global Positioning System (GPS) [4]. The position and elevation of the platform is known by combining information from the GPS and from the ground control by subtracting the measured distance of the Lidar from the known height of the platform [20]. Mapping is performed using Unmanned Aerial Vehicles (UAVs) equipped with pulsed laser ranging systems [4]. When scanning dense vegetation, buildings or power lines, multiple returns of the laser beam can be recorded in order to determine the height and the structure of the objects [20]. By using a rotating mirror, the laser pulses can sweep through an angle, which makes it possible for the sensor to point at any side of the aircraft [20]. When the airborne vehicle is perpendicular to the ground, the output of the sensor is a sawtooth pattern along the flight path [4]. Sinusoidal, side-to-side of zig-zag patterns are also possible, depending on the scanning mechanism [20]. Depending on the altitude and speed of the UAV, the system creates a Digital Terrain Model (DTM) with a footprint of 10 to 15 centimeters [4]. The distance between Lidar points on the ground, called “postings”, is a function of scan frequency, measurement frequency and the elevation of the aircraft [20]. Mappers need to be aware that, depending on the angle of incidence, a laser beam will interact with the ground in different ways [20]. Lidar pulses at the edge of a building will strike the sides, while pulses at the center will strike the rooftops; sometimes the returning signature can be reflected at an angle, which gives birth to the “shadow” effect, where less energy returns to the Lidar receiver [20]. A Digital Elevation Model (DEM) is created using a specialized software which creates a Triangulated Irregular Network (TIN) and then interpolates the points [20].

More lightweight solutions are now available on the market, resulting in the production of smaller airborne vehicles and a reduction in costs [15]. Many companies have joined the race to produce the best product on the market. Redtail Lidar Systems have designed a lightweight laser-based mapping system which creates high-density point clouds; the scanning mechanism is based on a Microelectromechanical Mirror (MEMS), capable of generating accurate, high-resolution 3D images of objects [16]. The HDL-32E is a High-resolution real time sensor produced by Velodyne which features 32 lasers across a 40-degree angle; with a weight under 2kg, it is a viable choice for airborne mapping applications [17]. The RIEGL VUX-1UAV Lidar sensor is a very accurate and light laser scanner which meets the challenges of system integration in UAV/UAS applications [18]. The Focus3D X 130 designed by FARO comes with an integrated GPS receiver which enables the scanner to correlate individual scans in the post-processing phase [19]. YellowScan Surveyor is most suited for civil engineering and mining applications due to it being designed for scanning demanding terrain models [15]. In conclusion, Lidar-based solutions are currently used in multiple industries, while the market is projected to grow due to an increased financial investment in research and development. Even though the Lidar is a robust and effective device, the costs for implementation is rather high, due to it being an emerging technology.

# **Theoretical fundamentals**

## Photodetectors for Lidar

Time of Flight (TOF) is a method used to calculate distance by calculating the time for a pulse of light, emitted by a light source, to be reflected at a target object and then received by a photodetector [22]. The TOF method is typically used in laser applications, providing robust measurements up to 50 m with centimeter accuracy. The system illustrated in Figure 5-1 is designed by the Hamamatsu company and combines a photo-sensor with a timer circuit and a time measurement circuit [22]. Hamamatsu have developed TOF devices where the photosensor can be a Multi-Pixel Photon Counter (MPPC), an Avalanche Photodiode (APD) or a PIN diode. Compared to other methods of distance measurement (like active triangulation or stereo vision method), TOF can measure a wider range of distances with an increased accuracy [22].

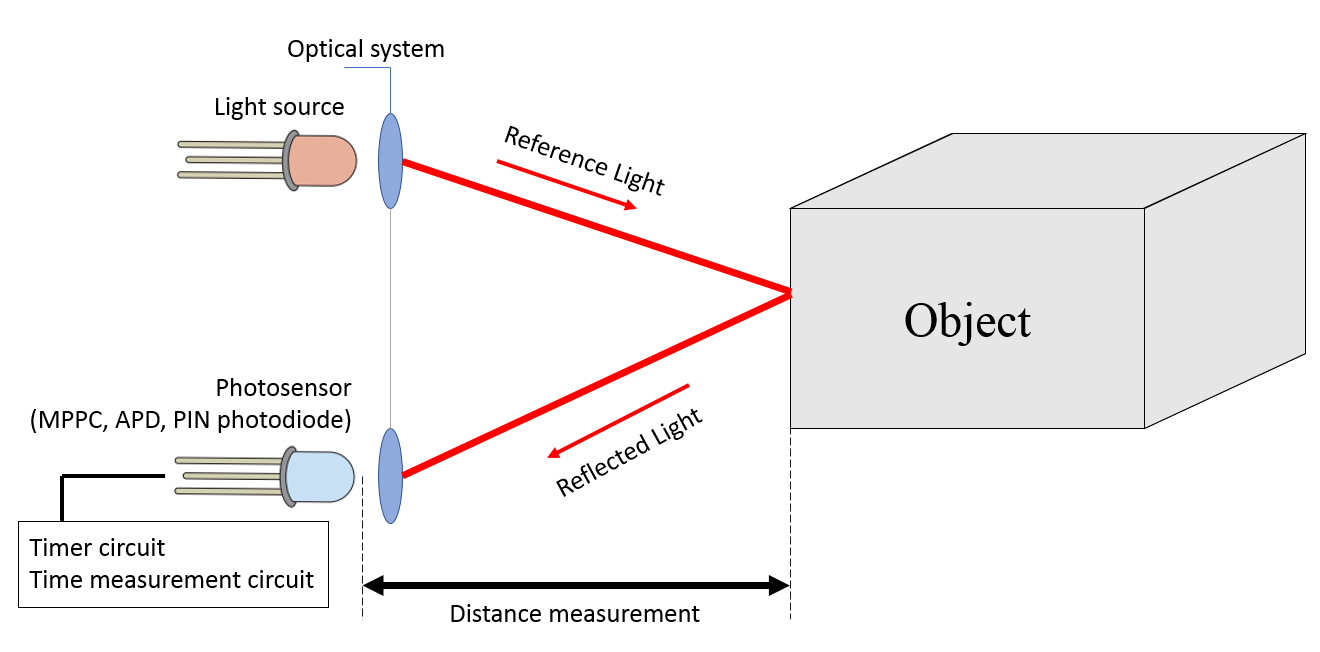


Figure 5‑1 Time of flight system [22]

The mechanism behind most photodetectors is based on generating a current as a result of light absorption [27]. In semiconductor photodetectors, the photocurrent is generated using the internal photoelectric effect, typically using a P-N junction or a P-I-N junction [27]. The quantum efficiency of the photodetector is defined by the number of absorbed photons which contribute in the output photoelectrons [27]. Equation (1), from [28], is used to evaluate the resulting amount of photocurrent, where η is the quantum efficiency and hν is the energy of one photon [28]. The P factor is called the responsivity of the sensor and is a proportionality factor between the photocurrent and the received optical power [32]. In addition to the given photocurrent, dark current may occur, which is not influenced by the intensity of the incident light beam [32]. Many photodetectors, such as avalanche photodiodes, employ a multiplication mechanism with the goal of obtaining an enhanced photocurrent [28].

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

The PIN photodiode is a device which has an undoped (intrinsic) region between the positively doped region and the negatively doped region, illustrated in Figure 5-2 [26]. Photons are absorbed in the intrinsic region, generating carriers which contribute to the photocurrent [26]. The green area on top of the P region represents an anti-reflection coating [26]. The main advantages of the PIN design compared to the regular PN structure are an increased quantum efficiency and a better detection bandwidth in the case when carriers are generated outside the depletion region [33]. The resulting photocurrent flows from the anode to the cathode; the anode typically has a ring shape so that light can be injected through the hole. The size of the ring may be increased in order to obtain a larger active area; however, a large active area increases the capacitance of the device, reducing the bandwidth and the efficiency of the photodetector [33].

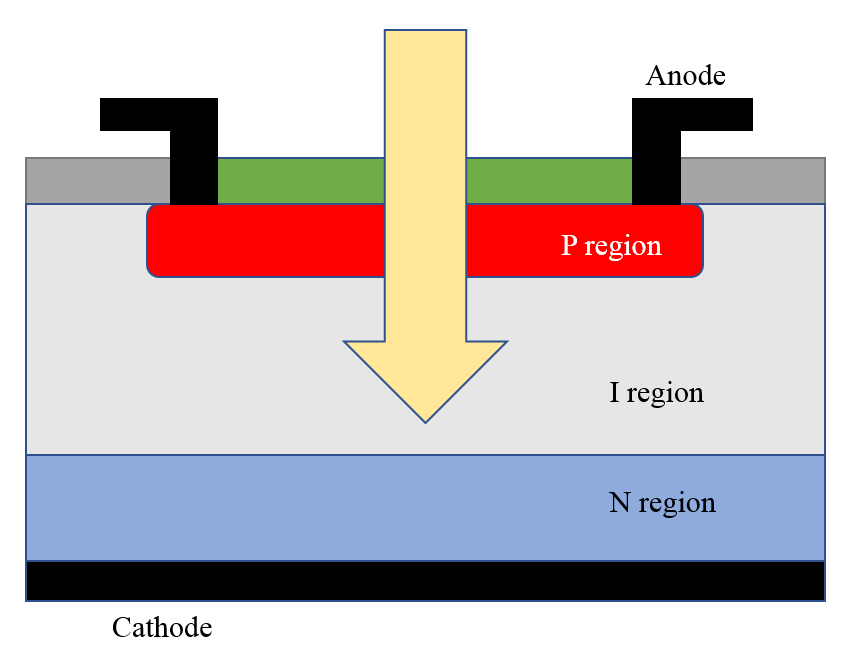


Figure 5‑2 The internal structure of a PIN diode, [26]

In Lidar applications, photodiodes are used in photoconductive mode: a reverse voltage is applied to the diode and the generated photocurrent is then measured [26]. Figure 5-3 illustrates the most basic solution for measuring output of a photodiode in reverse-bias mode, using a voltage source and a load resistor. The voltage drop on the load resistor results in a decreased voltage value on the diode, along with an increased photocurrent [26]. This process leads to a charge or a discharge of the sensor’s capacitance whenever a change in light intensity occurs, which means that the bandwidth is R-C limited [26]. Based on the value of the load resistor, a trade-off is made between the bandwidth and the responsivity of the system [32]. For example, a small resistor offers a higher bandwidth but also a reduced responsivity [32]. Additionally, a larger width of the N-I layer, also called “depletion region”, results in an increased capacitance. The solution to this problem is using a current amplifier, also known as transimpedance amplifier (TIA) [26]. The TIA keeps the reverse bias voltage at a constant level, thus negating the influence of the capacitance of the photodiode [26].

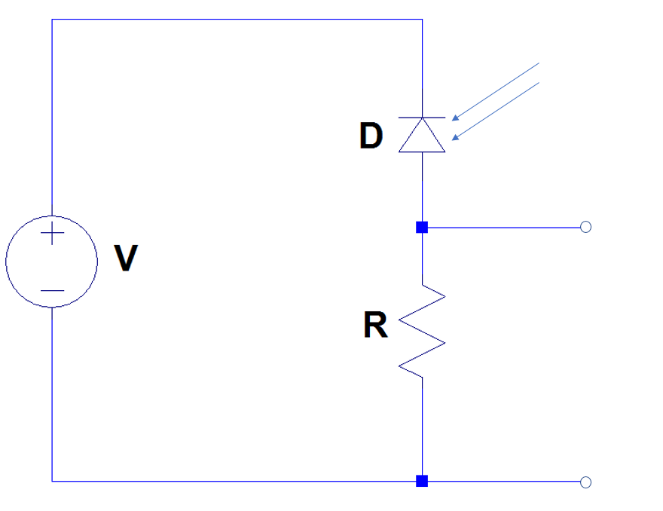


Figure 5‑3 Electronic circuit based on a photodiode

In Lidar applications, the transimpedance amplifier must have a bandwidth large enough to capture all the details in different operating conditions, while the amplifier noise should be as low as possible in order to avoid distorting the received signal [21]. The returning light can be sensed when the signal of a photodiode is amplified by a TIA and conditioned by a comparator [21]. Figure 5-4 depicts an automotive time of flight laser/receiver system designed by Maxim Integrated [21]. The system consists of a laser diode, avalanche photodiodes and supporting electronics [21]. A microcontroller commands the laser diode to emit multiple pulses, while TIA2 and COMP2 are used to record the transmission time of each laser pulse [21]. The signal travels through a glass which emits the light back to the MCU, enabling the timer circuit and the time measurement circuit [21]. The pulse travels until it reaches an object and is then reflected to the receiver. The photodiode converts light intensity to current intensity, which is then delivered to TIA1 and COMP1 [21].

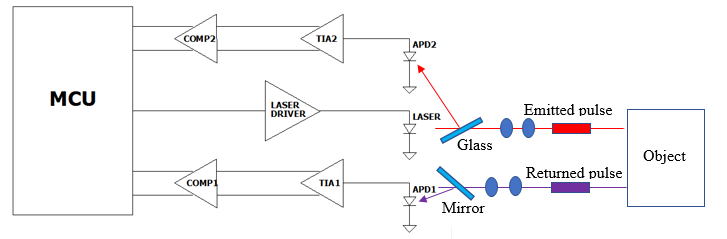


Figure 5‑4 Lidar system with Transimpedance amplifier and comparators, [21]

The avalanche photodiode is a semiconductor device capable of light detection. The operation mechanism relies on high reverse bias conditions [25]. The internal configuration is based on a p+-i-p-n+ structure, illustrated in Figure 5-5. The weakly doped intrinsic region of the diode is the conversion region for the incoming optical energy [28]. The absorbed photons excite the electrons and the holes, which accelerate in the strong internal electric field of the avalanche region, generating secondary carriers [25]. This process is known as the avalanche effect of the photodiode, which amplifies the output photocurrent of the diode [25]. The most important feature of avalanche photodiodes is the internal gain produced when a reverse voltage is applied [24]. The minimum light reception limit is characterized by the shot noise of background light, which can be reduced by using optical bandpass filters [22]. Therefore, the main application of the APD is in highly-sensitive photodetector circuits which require less signal amplification, reducing the amount of electronic noise [25]. The advantages of using and APD over the PIN diode is the higher achievable bandwidth as well as an enhanced responsivity [25]. Silicon avalanche photodiodes operate in a wavelength region from 400 nm to 1000 nm, with a maximum responsivity in the 600-800 nm area [25]. If an application requires a wider spectral range, germanium or indium gallium arsenide (InGaAs) photodiodes are used; these devices have a superior noise performance and a gain which varies between 10 and 40 [25].

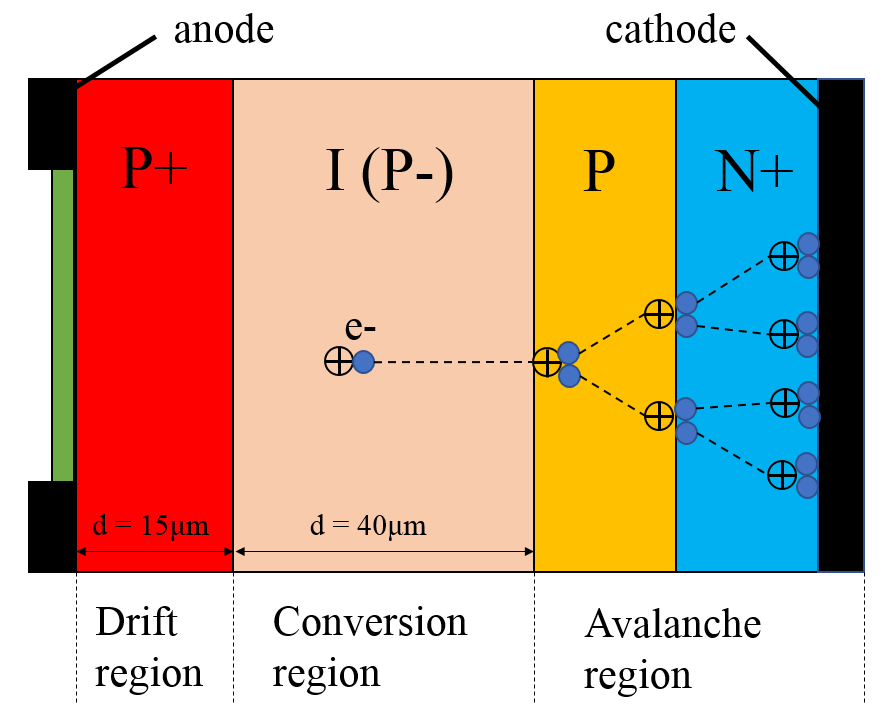


Figure 5‑5 Working principle of the avalanche photodiode

The MPPC, also known as silicon photomultiplier (SiPM), is a solid-state photodetector which uses an array of avalanche photodiodes operating in Geiger mode [23]. When an APD operates in Geiger mode, the applied reverse voltage is slightly higher than the nominal breakdown voltage [25]. The reason for using this mode of operation is that a single electron-hole pair (generated by the absorption of a photon) will trigger a strong avalanche effect [25]. The silicon will become conductive, resulting in an amplification of the original electron-hole pair into an electrical current flow [29]. In this case, the reverse bias voltage is reduced below the breakdown voltage by an electronic quenching circuit until the avalanche is stopped [25]. The most basic method of quenching is using a series resistor limiting the current drawn by the APD [32]. After a recovery time of several tens of nanoseconds, the diode is ready to detect further photons [25]. This process of breakdown, avalanche, quench, recovery allows the APD to work as a photon counter [25]. Regardless of the number of photons detected, the magnitude of the output signal is the same [33]. This is the reason why the MPPC employs an array of single photon avalanche diodes.

The basic element of the structure, the pixel, is an APD connected in series with a quenching resistor [23]. All the pixels available in the MPPC are electrically connected and arranged in a matrix structure; a simplified schematic is shown in Figure 5-6. When a pixel detects a photon, the avalanche process is triggered, followed by a quench and a recovery time, during which the pixel recharges to the initial bias voltage. While this process takes place, all the other “microcells” are charged and ready to detect further photons [29]. When multiple photons are detected at the same time, the amplitude of the output signal is equal to the superimposed pulses of the pixels [23]. Therefore, the magnitude of the photon flux can be calculated by summing the photocurrents generated by each individual cell [29]. When multiple photons are incident on the same pixel, the output of the MPPC becomes nonlinear [23]. The method used in estimating the number of detected photons is based on amplifying the output charge of the MPPC and then integrating it over a time period [23]. A peak will emerge in the correlation record, with an amplitude which varies based on the number of detected photons [23]. The schematic in Figure 5-6 illustrates a capacitor connected to the output of each cell, resulting in a fast output signal. The fast output signal is equal to the derivative of the internal switching of each pixel; the output of each cell is summed up, resulting in a pulse with a magnitude proportional to the incident photon flux [29]. The MPPC is suitable for long range measurements and direct TOF applications with low costs [33].

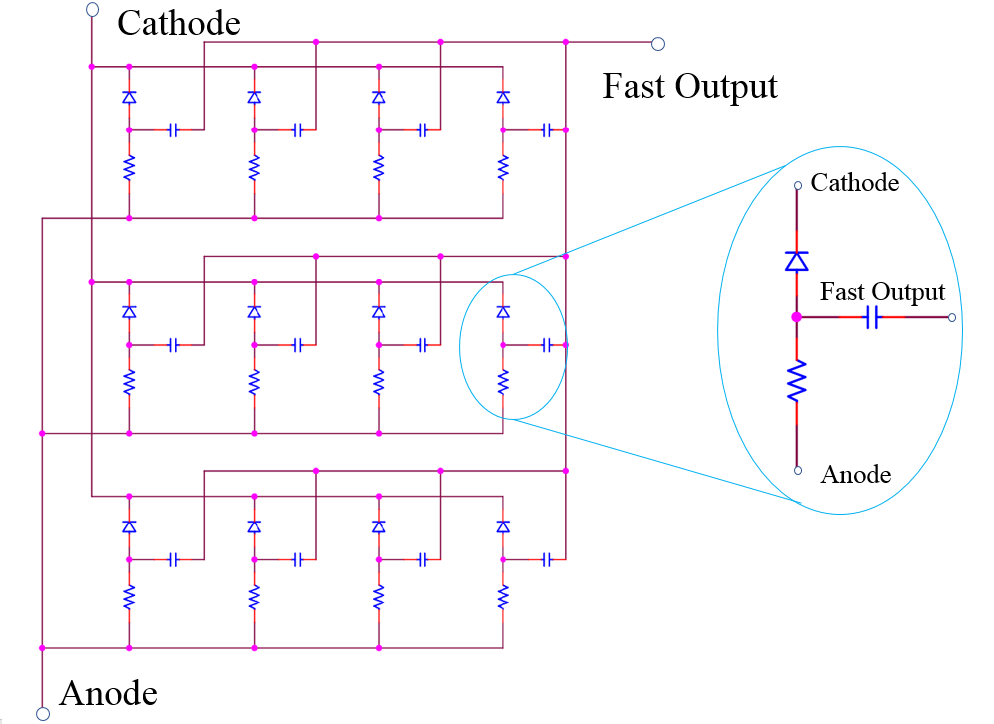


Figure 5‑6 Simplified schematic of an MPPC, [29]

Based on the multiple alternatives which have been presented, the Lidar Lite V3, from the company Garmin will be used in the 3D mapping application. The sensor working principle is based on the Time of Flight system which has been illustrated at the beginning of this paragraph. The system calculates the time delay between the transmission of an Infrared laser pulse and its reception. The receiver is an MPPC which detects returning light using the integration method. When multiple laser beams with the same signature have been received, a peak will emerge in the correlation record. The signal record is erased at the beginning of a new measurement. The device has its own signal processing algorithm used in detecting optical signal strength.

## 3D scanning methods

A 3D laser-based scanner has been designed at the Institute for Systems Engineering, University of Hannover, Germany by Oliver Wulf and Bernardo Wagner. The solution was presented at the 14th International Conference on Control Systems and Computer Science in 2003. The title of the article is “FAST 3D SCANNING METHODS FOR LASER MEASUREMENT SYSTEMS”. The reference article can be found at [30]. The goal of the research is to improve laser 3D scanners used for robots operating in industrial environments. Some of the main application areas of the research are: indoor safety systems, factory automation and moving platforms such as service robots. The methods which are presented in the paper have been implemented and tested on a robot running on a rea-time Linux platform. The following paragraphs describe an analysis and commentary of the solution, including the elements worth adopting in my Lidar-based project.

The solution presented in the paper implements a time-of-flight system, combining a laser-based 2D sensor and a servo motor drive, in order to achieve 3D vision. The blueprint for transforming a two- dimensional measurement device into a 3D scanner is by making use of a mechanical actuator, a servo motor. One such type of tool can rotate around its own axis, achieving a high density of points in the origin of the scan. Depending on the scanning mode selected, a different density of points can be achieved. The rotation of the sensor on two axes is provided with multiple modes of acquisition which lead to different fields of view. There are four scanning methods presented in the article: pitching scan, rolling scan, yawing scan and yawing scan top. In a pitching scan, the sensor has a horizontal plane and swipes up and down along the z-axis. The rolling scan has a fixed point around which the sensor will rotate, creating a set of field lines around it. The yawing scan and the yawing scan top are methods where the sensor has a vertical scan plane and rotates around the z-axis. A variation in the alignment of the sensor along the 2D rotation axis will result in a different scanning pattern. After studying the scanning modes and their output patterns, we have decided to implement a yawing-scan method. The functional scheme of the yawing scan is illustrated in Figure 5-7. The Lidar sensor which is used for the application is a one-dimensional device. Two motors are used to achieve the second and third dimensions. The first motor rotates the sensor around the z-axis, resulting in a two-dimensional measurement system. The second motor rotates the sensor along the vertical plane, achieving the third dimension. The yawing scan method is implemented by rotating the first motor around the Z-axis first, while performing distance measurements. The second motor will then move the sensor along the vertical scanning plane, at a new angle. The resulting point cloud pattern is that of circular field lines around a fixed reference point.

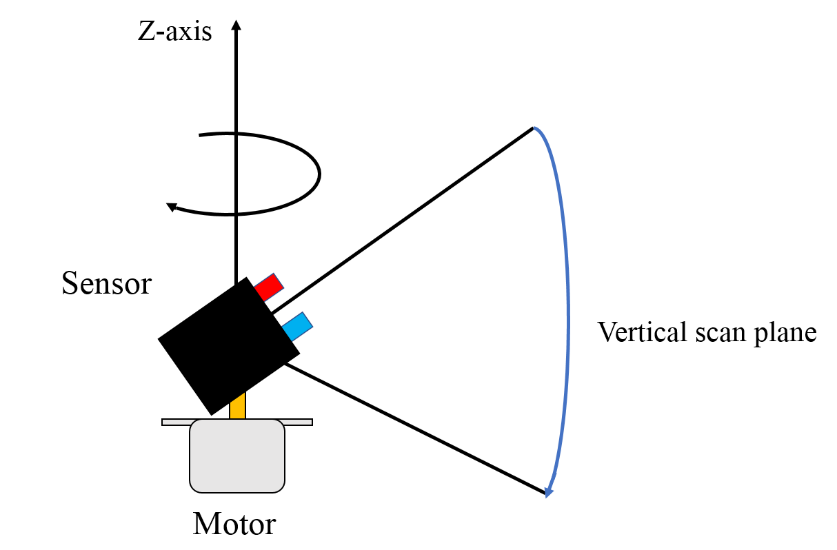


Figure 5‑7 Yawing scan functional scheme

Furthermore, the measured points are not placed on a regular grid, as opposed to camera systems. The density of the measured points is non-homogenous due to the accumulation of scan points along the rotation axis of the sensor. One of the methods of illustrating the density of points in the field of view is to plot the points on a 3D sphere around the sensor [30]. The paper further illustrates the resulting scanning patterns for each scanning method mentioned in the previous paragraph. Because the yawing scan method is used in the Lidar 3D scanner, it is worth studying the point distribution pattern of this scanning mode. The typical point distribution pattern of the yawing scan is illustrated in Figure 5-8. It can be observed that one large point distribution area is at the center-top part of the 3D sphere. The yawing scan has been best applied in situations where the horizontal scanning angle is larger than 180 degrees due to the favorable point distribution pattern.

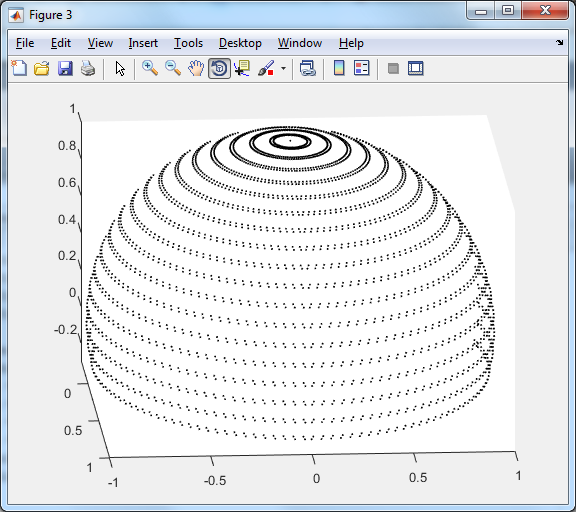


Figure 5‑8 Yawing scan point distribution pattern

In addition, the authors present a common method of 3D representation of point clouds, using Cartesian coordinates. A transformation is performed with the raw scan of the 2D sensor and the position of the motor as inputs. One of the most important aspects mentioned in the article is knowing the position of the scanner during data acquisition. The solution is obtaining a time-consistent data set of the servo drive and the sensor positions. Two methods to acquire data sets are described in the paper. The first method consists of associating a measurement from the sensor with the position of the servomotor at the time of the measurement. Since the mechanism presented in the article is based on a servomotor drive, this method is not time consistent. The position of the scanner is not specified at the exact time when a distance measurement is acquired. Because the servo drive gives a non-real time response, a time delay, which lasts up to 100 ms, occurs. Therefore, this method is discarded by the authors.

The second method is based on reading data continuously from the sensor and from the servo motor. The readings from the servo and from the sensor do not have to work with the same sampling frequency. An accurate time stamp is attached to each raw data sample so that the position of the scanner at the time of each measurement can be deduced. A correlation is made between the time stamps and the position data sets, calculating the transformation into Cartesian coordinates. This final task is considered not to be time critical and can be executed at a lower frequency. Additionally, this method is influenced greatly by the sensor and the mechanical elements of the scanner. The solution presented in the article is based on the LMS 200 sensor from the company Sick. The resolution of the scanner depends on the sampling frequency of the microcontroller and on the rotation speed of the servo motor.

We have decided to implement the first technique presented in the article. The drawback of not knowing the exact position of the scanner can be resolved by designing a mechanical setup based on stepper motors. Stepper motors, as opposed to servo motors, provide accurate angle control, without the need to save the actual position of the scanner at every iteration. The sensor used in the project, the Lidar Lite V3 from the company Garmin, is a one-dimensional. Coupled with two stepper motors, the second and the third dimensions are achieved. In order to complete a yawing scan, one motor will rotate the sensor around the z-axis and the second motor will move the sensor along the vertical plane. In this way, by controlling the angle of each stepper motor, a real-time response can be obtained from the 3D scanner. By replacing the servo drive with stepper motors, the second method can be discarded. There is no need for setting a time stamp when a distance measurement is performed since the positions of the scanner are known at any given instant. The positions of the two stepper motors represent the polar and azimuthal angles. The data collected by the Lidar constitutes the Euclidian distance of each pixel. The scan parameters and the distance measurements of the Lidar will be used as inputs to a script which transforms spherical coordinates into Cartesian coordinates.

One issue which has been pointed out by the authors of the article is the mechanical limitations of such a device. All the solutions presented in the paper concern 3D scanners which consist of 2D laser devices combined with a servo motor drive with limited turning angle. The turning angle limitation exists due to the power supply and data transfer cables. In the article, this mechanical problem results in a non-constant turning speed of the sensor, because the device needs to be accelerated at the start and at the end of each turn. Therefore, the servo drive requires more energy compared to a continuous scanner. This acceleration leads to a reduced mechanical stability and lifetime of the scanner in the case of fast scans. In order to transform a scanner with limited turning circle to a continuous scanner, the cable connections need to be replaced with slip rings. Since the objective of my project is to design a test bench for 3D reconstruction algorithms, we have decided to design a non-continuous scanner using cable connections. The scanning angle will be limited to maximum 180 degrees. One additional trade-off is represented by the accuracy and the stability of the scan which results in a decreased speed. This aspect is not of great concern in this situation since the system is designed to function in static conditions.

## Terrestrial Lidar scanning

Another Lidar-based solution has been described in an article published in the Agricultural and Forest Meteorology journal, Volume 149, Issue 9, 1 September 2009, Pages 1505-1515. The title of the article is “Obtaining the three-dimensional structure of tree orchards from remote 2D terrestrial LIDAR scanning “. The authors of the paper are: Joan R. Rosell, Jordi Llorens, Ricardo Sanz, Jaume Arnó, Manel Ribes-Dasi, Joan Masip, Alexandre Escolà, Ferran Camp, Francesc Solanelles, Felip Gràcia, Emilio Gil, Luis Val, Santiago Planas, Jordi Palacín. The reference article can be found at [31].

The paper presents a Lidar-based solution for detecting the structure of forest, crops or individual trees. The structure of a tree plays an essential role in the carbon exchange of the ecosystem [31]. Based on the structure of a plant, multiple parameters can be predicted, including: potential growth, stem density, basal area and the above ground biomass [31]. The structure and diversity of vegetation greatly influences the habitat selection of certain species of animals within an ecosystem [31]. The Lidar is a non-destructive method of detecting the structure and density of vegetation. The structural variables can be extracted from the high-resolution point clouds generated by a Lidar scanner. Forestry is an industry which has been using 3D information since 1933, using the stereo-photography technique [31]. Quantitative measurements such as tree height and crown diameter are essential in forestry. However, the agricultural crops are different from forest areas due to the accessibility of the zones for vehicles and humans. The forest areas are harder to access, which is an influence on the types of instrumentation used in each case. The main advantage of Lidar is the high-resolution point cloud provided at the output. The high cost of the device limits its use.

Cheaper, 2D Lidar sensors are used in agricultural applications. The 2D sensor provides a point cloud corresponding to a section of the frontal plane. The 3D dimension of the system is determined by mounting the 2D Lidar on a tractor conducting a linear movement with a constant, known speed. The experimental scan is executed in a pear orchard. The scanner used in the research is the Sick LMS200 model. The maximum scanning angle of the device is 180° [31]. The lateral resolution of the scanner ranges between 0.25°, 0.5° and 1° [31]. The working principle of the system is based on the Time-of-Flight method, using a 905 nm laser capable of detecting object up to 80 m [31]. The authors of the article have developed a software used to store and process the data from the scanner. The system is connected to a central unit via the RS232 serial port. The Lidar scanner is powered from 24 V power supply. A graphic interface application for data recording and offline processing has been developed using Matlab [31]. The experiments in the final test stage have been conducted using the Compact Field Point programmable automation controller from National Instruments. The automation controller offers the capability for real time operation. Even though the LMS200 model is a 2D scanner, the software has been developed for a 3D scanner. The third dimension of the system is achieved by moving the device, parallel to the rows of trees at a known speed. The data is transformed from polar coordinates to Cartesian coordinates. A GPS system is introduced to create a georeferenced point cloud. However, the GPS addition is useful only if the precision is within the cm range.

The system was fixed to a moving structure, attached to the ceiling of the laboratory. The user can change the velocity of the moving structure. The first laboratory experiments of the system aimed to measure the 3D dimensions of objects: height, width and thickness. The results of the 3D Lidar system were then compared to the manual measurements. After conducting measurements on a PVC tube and on steel frame, the authors decided to scan a medium sized tree: a Ficus Benjamina Variegata with 0.7 m width and 2 m height. The performance of the measurement system is tested in a controlled environment. The plant was placed inside a steel frame, while the scanner conducted measurements from lateral and front view of the plant. The minimum distance between the trunk of the plant and the Lidar sensor was set to 1 m. Furthermore, the paper presents measurements of real crops. The LMS200 device is mounted on a tractor on a vertical axis designed to move the sensor at different heights above the soil surface [31]. A field measurement consisted of multiple runs along either side of the vegetation row, at a distance between 1 and 3 m from the tree crops [31]. The vehicle was moving in a straight line, at a constant speed between 1 and 2 km/h. Well known objects were placed as reference planes at the beginning and at the end of the measurements. The planes consisted of wooden objects with flat surfaces, serving as references for the data processing software and 3D placement of the point clouds [31]. By combining the point clouds collected from both sides of the crops, the 3D model can be created using a specialized software like VBA (Visual Basic for Applications) [31].

Based on the research article presented in the Agricultural and Forest Meteorology journal, several solutions and experiment scenarios were adopted. First, the Lidar 3D scanner will communicate with a central unit via the RS232 serial interface. All the components necessary for the project will be powered from the same voltage source to avoid additional cable connections. In addition, a graphical user interface is necessary for changing the scan parameters without modifying the microcontroller software. In this way, the process of conducting experiments is easier because the user has no interaction with the hardware drivers. One of the testing scenarios will be based on detecting the 3D dimensions of flat objects. A data processing application is designed for creating the 3D plot of the point clouds.

## **Lidar Geometric Model and Calibration**

A solution for creating a 3D model of a Lidar scanner is published in the Sensors MDPI, on the 20th of May, 2020. The title of the article is “Geometric Model and Calibration Method for a Solid-State LiDAR”. The authors of the research article are: García-Gómez, Pablo & Royo, Santiago & Rodrigo, Noel & Casas, Josep. The paper can be found at [32].

Most mechanical Lidar systems work by spinning a macroscopic element, either an optical element such as a mirror or the whole sensor embodiment [32]. Solid-state Lidars are immune to mechanical vibrations, shock an impact. The Lidar Field of View (FOV), for a Time of Flight system consists of all the unitary scanning-direction vectors in the reference system of the sensor. The scanning-direction vector should have constant spacing to minimize the distortion error. No matter how the scanning is performed, the spacing is distorted due to the optical or mechanical mismatch of the system [32]. Since, the paper focuses on TOF systems, the optical model of the solid-state Lidar is based on Snell’s law. Light paths and angles can be estimated using Snell’s law, assuming that the incident and the reflected ray are in the same plane as the normal. The FOV of the Solid-State Lidar is mapped by representing in detail the angular resolution [32].

The Lidar reference system is considered to be an orthonormal basis, {S}, which results from applying the rotation matrices. Therefore, any point in the Euclidean space {L} can be expressed as a combination of a linear combination of the three bases of {S} [32]. The Lidar reference system {L} = {L1, L2, L3} is defined within the scanning reference system {S} and is centered at the origin of the space. Any point Q with known Cartesian coordinates can be expressed in terms of the Lidar reference system [32]. The Lidar TOF system calculates the distance between the sensor and the point Q. In this situation, the most useful system of coordinates is the spherical system. The authors continue with expressing the Cartesian coordinates in terms of spherical values. Equation (2), from [32], is used to calculate the Cartesian coordinates in terms of spherical coordinates.

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

C is the speed of light, tTOF is the time required for the laser ray to return to the receiver. In addition, θ is the polar rotation angle and φ is the azimuthal angle. The FOV of the Lidar is expressed in terms of the Horizontal and the Vertical fields of view. FOVH represents the maximum turning angle of the scanner. FOVV represents the maximum inclination angle of the Lidar. Since the Lidar reference system is centered in the origin, the FOV of the scanner is represented between [-FOVH/2; +FOVH/2] on the horizontal and between [-FOVV/2; +FOVV/2] on the vertical. The authors apply Equations (3) and (4) to calculate the polar and azimuthal angles.

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

θH and θV are the horizontal and vertical scanning angles. The relations from Equations (3) and (4) have been deduced by applying trigonometry. Ideally, the distance from one pixel to the next, also known as the angular resolution ∆θ, is constant. Because the research article describes a non-continuous Lidar scanner, the scanning lines are divided into odd and even lines. Odd lines are represented by left-to-right turns, while even lines result from right-to-left turns. In order to create a better model of the distorted FOV, non-linear terms are added to the data set. Since the radial symmetry is lost, a distortion model is needed [32]. The odd and even lines create two separate images and need to be treated separately. The two mapping functions of the scanner are found by solving a no-linear least squares system (NLSQ) for a set of control viewing angles [32]. The control viewing angles have been measured at different pixel positions. Three distortion mapping functions are proposed: Optical-Like Mapping, Cross Mapping, Multi-Decentered Cross-Mapping [32].

The article has given me an insight into what sensor calibration means. Even though the research paper is based on a solid-state Lidar scanner, the concepts can also be applied in a mechanical Lidar project. The relations and information found in the research article have been tested in the Lidar 3D scanner. By using Equations (2), (3) and (4), the program calculates the Cartesian coordinates of the pixels and generates two distorted images centered at the origin of the Lidar reference system. Good results are obtained for point clouds with fewer distance measurements. However, when larger point clouds, such as the data from a room scan have been tested, the resulting image offered no distinguishable surfaces. The research article states that only computing the Cartesian coordinates is not enough for obtaining detailed 3D plots. A distortion mapping algorithm needs to be applied calculating the odd and even lines data sets. Designing such an algorithm can be a future improvement for the Lidar 3D scanner.

# Implementation

This chapter describes the design and implementation of a non-continuous Lidar mapping system. An upgrade to continuous scanning can be implemented by using a slip ring and gears. The device features the yawing scan as the only scanning method of reference. Additional scanning methods, such as the pitching scan, can be implemented in the future. The project is divided into three main applications: data processing application, desktop application and microcontroller application. The implementation is described using a top-down approach. The first subchapter presents implementation of the data processing script. The input consists of two files: one which contains the scan parameters and one which contains the results of the scan. The input data is transformed from spherical coordinates to Cartesian coordinates, then plotted in a 3D representation. The second subchapter describes the design of the Java desktop application, which features a graphical user interface. The program establishes a connection between a personal computer and the Lidar 3D scanner. The scan parameters can be set and a scan can be commenced using the graphical user interface. The third subchapter is dedicated to the microcontroller application which is used to command the execution of the scan, according to its parameters. The results are transmitted serially to the desktop PC. Furthermore, this chapter includes a detailed description of the hardware components which have been integrated in the design: a Lidar sensor, stepper motors and an accelerometer.

## Data processing application

The role of the data processing application is to transpose each point from spherical coordinates to Cartesian coordinates. A Matlab script has been developed to create a 3D plot in order to observe the output of the system. The program takes two text files as input: a file containing the results of a scan and one which contains the scan parameters. Figure 6-1 provides a graphical representation of a point using spherical coordinates.

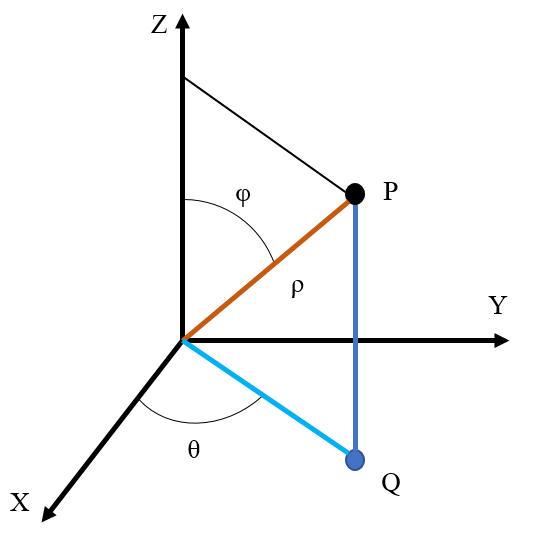


Figure 6‑1 Representation of a point using spherical coordinates, [35]

The first text file, *“results.txt”*, contains the distance measurements collected from the Lidar. The data is represented in the unsigned integer format, which constitutes distance measurements in centimeters. The information in the text file is divided into rows and columns. Since the data belongs to a yaw scan method, the results on the same row belong to the same turn around the z-axis. Spherical coordinates describe the position of a point by using its Euclidian distance ρ from the origin, the polar rotation angle θ and the azimuthal angle φ. According to Figure 6-1, the position any point in a three-dimensional space can be described using spherical coordinates. If P is the point represented in three dimensions, then Q is the projection of the point on the x-y-plane. The polar angle θ is the angle between the positive x-axis and the line between the origin and Q, while angle φ is the angle between the positive z-axis and the line between the origin and P [35]. The polar angle is also called yaw angle and the azimuthal angle is known as pitch angle. The system is controlled by two identical stepper motors. The first stepper motor drives the rotation of the scanner around the z-axis, changing the angle θ with every step. The second motor achieves the second dimension of the system, driving the scanner along the positive z-axis and changing the angle φ. Therefore, in the *“results.txt”* file, the information on the same row belongs to the same pitch angle and results on the same column belong to the same yaw angle.

The second text file, *“params.txt”*, contains the parameters of the 3D scan. The parameters are represented as integer numbers, divided by a space, on a single row. The Field of View (FOV) of the Lidar is defined on the horizontal and vertical axes. The scan parameters consist of five integer numbers: yaw resolution, yaw maximum steps, pitch resolution, pitch maximum angle and pitch start position. The yaw resolution represents the distance between each pixel on the horizontal axis. The pitch resolution is the distance between pixels on the vertical axis. They are defined as the number of steps executed by each stepper motor. The yaw maximum steps define the width of the FOV, also known as FOVH. The number of steps is converted into the maximum turning angle of the scanner. The pixels are defined horizontally in the range [-FOVH/2; FOVH/2]. The pitch starting and maximum angles define the height of the FOV, also known as FOVV. The pixels are defined vertically starting from the pitch initial angle and finishing at the pitch maximum angle. As the yawing scan is used in this project, it is important to know how the information is collected. The Lidar starts from a reference point, at the pitch start position. Then, the first stepper motor executes a half-turn around the z-axis, setting the start of the first line at -FOVH/2. When the scanner is turning, the sensor continuously collects distance measurements. When the scanner has reached its maximum turning angle around the z-axis, the second stepper motor executes the pitch resolution steps along its vertical plane. When the maximum angle along the z-axis has been reached, the scan has ended.

Using the input information, the program can associate correct spherical coordinates for each point. Equation (5) describes how the width of the FOV is calculated, as a product between the maximum yaw steps and the angle/step resolution of the stepper motor. Regarding the stepper motor chosen for this application, the resolution is constant and equal to 0.17578125 degrees per step

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Equation (6) presents the formula for calculating the number of measurements which have been done in a single turn. The maximum yaw steps are divided by the yaw resolution. One is added to the result, due to the starting yaw angle of the scanner at the beginning of each turn.

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

A model is presented in Equation (7) from [32], which computes the angle θ at any given position. The division between FOVH (the width of the FOV) and NH (the number of pixels on each row) gives the distance between two pixels on the horizontal axis. The polar angle varies between: [-FOVH and +FOVH]. The scanner changes the direction between turns. Therefore, the corresponding yaw angle is different for odd and even lines. The turn starts at the -FOVH angle and ends at +FOVH for odd lines. However, if the line is even, the starting and ending positions, with respect to the horizontal FOV, are inverted.

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

When it comes to the pitch angle φ, a starting angle and a final angle are provided at the input. Equations (8) and (9) are used to calculate the distance between two pixels on the vertical axis. First, the value is computed as the product of the stepper angle resolution and the pitch step resolution. This value is also known as the resolution of the pitch angle. When the pitch starting angle is larger than the final pitch angle, the value of the pitch angle resolution is negative. In this case, the sign of the pitch angle resolution will be changed. Otherwise, the sign of the pitch angle resolution remains the same.

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

Equation (7) is used to compute the number of turns executed around the z-axis. The first operation is the difference between the pitch starting angle and the pitch final angle. The result is divided by pitch angle resolution in absolute value. The *floor* function is then used, rounding the number to the nearest integer, less or equal. Finally, one is added to the result due to the first turn which was not considered.

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

A model is presented in Equation (11) which determines the angle φ at any given position. The azimuthal angle reflects the position of the scanner with respect to the vertical axis of the FOV. The values range between the pitch starting position and the pitch maximum value. The pitch angle resolution is the distance between two consecutive pitch angles.

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

The distance measurements from the 3D scanner are imported from the *“results.txt”* file. The information is organized in a two-dimensional array called ρ. In terms of spherical coordinates, the Lidar results represent the Euclidian distance between the sensor and the FOV pixel. The number of columns of the matrix is equal to the number of measurements collected in a turn, NH, deduced using Equation (6). The number of rows is equal to the total number of turns executed by the 3D scanner, NV, computed using Equation (10). The θ and φ angles of any point are calculated using Equations (7) and (11). Two one-dimensional arrays have been created for storing the values of the angles. Therefore, any point P (i, j), with i ranging from 0 to NV - 1 and j ranging from 0 to NH -1, can be described by the three spherical coordinates: ρ (i, j) - the distance from the origin, θ(j) - the yaw angle, φ(i) - the pitch angle. The next step is converting the spherical coordinates of each point to Cartesian coordinates. Equations (12), (13) and (14), from [35], represent the transformation of spherical coordinates into Cartesian coordinates for any given point, P (i, j).

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

The Cartesian coordinates of each point are two-dimensional vectors with size equal to the size of the R matrix. After all the Cartesian coordinates have been calculated, the program creates a 3D reconstruction using the Matlab *plot3* function. The two-dimensional vectors x, y and z are given as input parameters. The script plots the data as singular unconnected points. A more detailed discussion regarding the 3D reconstruction of scenery is described in the chapter dedicated to experimental results. The Matlab script for the data processing application has been added to Appendix 1 – Matlab script.

## Desktop application

The desktop application is designed using the Java programming language and Eclipse Integrated Development Environment (IDE). The application communicates serially with the microcontroller application and it features a graphical user interface (Figure 6-2).

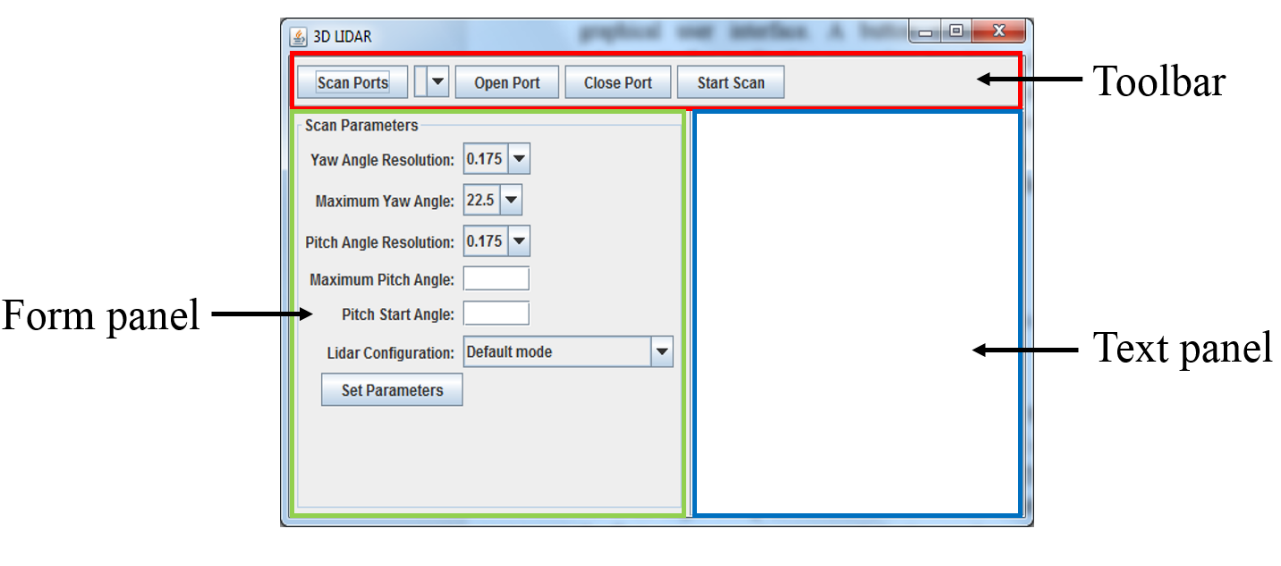


Figure 6‑2 Java Graphical User Interface

The user can search for available serial ports using the *Scan Ports* button. A connection to the microcontroller application can be established by pressing the *Open Port* button. The scan parameters can be changed by pressing the *Set Parameters* button. A special synchronization routine has been implemented for sending the parameters to the microcontroller. A scan can be started by pressing the *Start Scan* button. The default value of the Maximum Pitch angle is -90° and -10° for the Pitch Start angle. The graphical user interface is divided into three components: Toolbar, Form Panel and Text Panel which are placed on a Main Frame. The Text Panel (Figure 6-2, blue) has been added on the right side of the Main Frame. A message will be displayed on it every time an event occurs. The program is designed in such a way that the Main Frame controls all the functionalities. The components placed on the Main Frame have no interaction with each other. The definition of each event is placed in the Main Frame with the goal of making the program easier to understand. Additionally, this design structure makes the application scalable since the software components used in the Main Frame have no interaction. Any component which has been added to the Main Frame can be used in any other application as an individual element. In the same idea, due to the complexity of the program, it is important to avoid the situation where the components of the Main Frame depend on each other.

### Toolbar

The toolbar (Figure 6-2, red) consists of the upper part of the graphical user interface and it has four buttons and one Combo Box. A class called *Toolbar.java* has been created. An action listener has been implemented for each button. If the user presses any of the buttons a certain execution of program will be triggered. The Toolbar will define and place the graphical objects on its panel. Additionally, the Toolbar sets the action listeners of each button. The code which will be executed when a button which belongs to the Toolbar is pressed is defined in the Main Frame. All the serial port functions have been designed using the jSerialComm: SerialPort Java library.

When the *Scan Ports* button is pressed, the system will commence a search for available serial ports. The message *“Scan Com Ports”* is printed on the Text Panel. When the program finishes searching for the available serial ports, the Combo Box is updated with the names of the serial ports which can be accessed by our application. In order to connect to the microcontroller application, the user must select the serial port where the 3D scanner is connected. A connection can be established to the selected serial port by pressing the *Open Port* button. When the button is pressed, the currently selected element from the Combo Box is the name of the serial port to which the application will attempt to connect. If the attempt is unsuccessful a flag is raised and the message *“Open Port Fail”* is displayed on the Text Panel. If a serial connection is successfully established between the Java application and the 3D scanner, the name of the communications port is saved and the message *"Open Port Successful”* is printed on the Text Panel. The serial connection with the microcontroller application is done with 76800 bits/second Baud rate, 8 bits frame, no parity and one stop bit. Depending on the situation, the serial port can be set in transmitter mode or in receiver mode. Subsequently, the user can disconnect from the serial port by pressing the *Close Port* button. The connection to the serial port will be interrupted and the message *"Com Port Closed"* will be printed on the Text Panel. Interfaces have been defined for each button with the purpose of setting their action listeners within the Toolbar class. The events which follow the action listeners are defined in the higher level of the application: The Main Frame.

When *Start Scan* is pressed, the desktop application saves the parameters of the scan in a file called *params.txt*, sets the serial port in transmission mode and sends the character with the ASCII code 127. This ASCII character commands the microcontroller application to start a scan and will be referred to as Start Scan. The program then sets the serial port in receiver mode. The application is ready to save data received from the scanner. The microcontroller application will send the results serially with a *“new line”* delimitation between rows and a *“space”* delimitation between columns. Initially, the data is saved in two separate files, *file1.txt* and *file2.txt*. This solution has been chosen since the 3D scanner changes direction around the positive z-axis at the end of a turn. At the beginning of a scan *file1.txt* is the designated saving destination for the collected measurements. The data coming from the microcontroller application is represented in and 8-bit format and is saved in the destination file. When a turn ends, the character corresponding to a new line is received. The saving destination is changed from *file1.txt* to *file2.txt*. Until the end of the next turn, the incoming bytes are saved in *file2.txt.* When the next turn ends, and a new line is received, the save destination is changed back to *file1.txt.* The purpose of having two destination files is to save the odd and even lines in separate locations. A calibration algorithm can be implemented, based on the even and odd lines. This process is repeated until the microcontroller signals the end of a scan by sending the Start Scan character.

The rows of data in *file1.txt* start with the angle θ equal to -FOVH and end with +FOVH. The rows of data in *file2.txt* start with the angle θ equal to +FOVH and end with -FOVH. The elements in *file2.txt* are inverted with respect to the yaw angle because the scanner changes its direction at the end of a turn around the z-axis. The next step of the program is to merge the information from the two files into a single file called *“results.txt”*. The odd rows of the output file are equal to the rows of *“file1.txt”* and the even rows are equal to the rows of *“file2.txt”*. This operation is necessary in order to provide a correct input file for the data processing application. A loop is implemented where lines are read from both files until the files are empty. In this way, the output of the Java application will provide a correct input for the Matlab script.

### Form Panel

The Form Panel (Figure 6-2, green) consists of the middle-left part of the graphical user interface. The Panel provides the user is used to change the parameters of a scan. When a scan starts, the current parameters are saved in a separate text file. The yaw angle resolution, the yaw maximum angle and the, the pitch angle resolution and the Lidar configuration can be changed using a list of predefined elements. The predefined elements are contained in combo boxes with drop down lists. The maximum pitch angle and the pitch angle resolution can be changed using text boxes. Even though the user can type in any value, there is an upper limit and a lower limit.

The yaw angle resolution is a parameter which controls the number of steps performed around the z-axis after executing a measurement. This parameter can be changed using a Combo Box with multiple options. The stepper motor executes 2048 steps to complete a 360° rotation. By choosing 0.175 the first stepper will execute one step around the positive z-axis after each measurement. Choosing 0.351 will be equal to two steps, 0.703 to four steps, 1.406 to eight steps. The value chosen is the increment of the angle θ and it affects both the speed of the scan and the number of distance measurements collected in a turn. The maximum yaw angle is a parameter which controls the width of the measurement arc around the z-axis. By selecting a maximum angle of 22.5° means that the first stepper motor will execute 128 steps around the z-axis. The options also include 45° which is equal to 256 steps, 90° which is equal to 512 steps, 180° which is equal to 1024 steps and 360° which is equal to 2048 steps. The pitch angle resolution parameter controls the number of steps executed by the second stepper motor along the vertical plane of the scanner. The pitch steps are executed after the first stepper motor executes a turn. The parameter can be changed using a drop-down list with the same options as the yaw angle resolution parameter: 0.175 – one step, 0.351 – two steps, 0.703 – four steps, 1.406 – eight steps. The value chosen represents the increment of the angle ϕ and it affects the number of turns performed in a scan. The maximum pitch angle is a parameter which describes the final position of the 3D scanner along the z-axis. It is recommended to introduce integer values in the range [-120°, 120°]. The maximum and minimum values of the given range have been deduced experimentally. The pitch start position is a parameter which describes the initial position of the 3D scanner along the z-axis. A Text Box is used to control the parameter. The value chosen should be an integer number which belongs to the range presented above. The value of the pitch start position is equal to the initial value of the angle ϕ, which, along with the maximum pitch angle, influence the number of turns carried out in a scan. The relation between the pitch starting position and the pitch final position determine the moving direction of the scanner along its vertical plane. If the value of the starting position is higher, the result is a top-down movement of the scanner along the vertical plane; if the opposite situation occurs, it would result in a bottom-top movement. The Lidar configuration parameter is used to control the operation mode of the sensor. The configuration options include *“Default Mode”*, *“Short Range, High Speed”*, *“Default Range, High Speed”*, *“Maximum Range”*, *“High Sensitivity Detection”* and *“Low Sensitivity Detection”*. Depending on the selected mode of operation, the Java application will send a value between 1 and 6 to the microcontroller.

By pressing the *“Set Parameters”* button, the Java program will send the current values of the parameters to the microcontroller. First, the character with the ASCII code 126 signals to the microcontroller application. This character, also known as Set Parameters, commands the microcontroller to enter a routine dedicated to receiving the scan parameters from the desktop application. A synchronization protocol has been implemented between the Java application and the microcontroller application. The Java application will send the information starting from the most significant digit of each number. If the number is negative, the sign character is sent first and the microcontroller raises a flag. When a parameter has been successfully sent, the character *“space”* is sent as a delimitation. The microcontroller uses an array initialized with 0 for the scan parameters. The Java application will send one byte and then waits until it receives a code from the microcontroller. If received byte is between the ‘0’ and ‘9’ characters, the microcontroller application transforms the byte from ASCII to a decimal value. The result of the current parameter is multiplied by 10 and the new digit is added to the result. After the microcontroller receives and processes the byte, it will send back the character with the ASCII code 125. This character, also noted as the Continue Send character, is used to signal to the desktop application that the microcontroller is ready to receive a new byte. The process is repeated until the microcontroller application receives the character *“space”*, which means that the current scan parameter has been successfully received. The process is repeated for the next element in the array. When all the scan parameters have been sent, the Java application sends the Set Parameters character. When the microcontroller receives this character a flag is raised, signaling that the information regarding the scan parameters has been successfully received. The microcontroller then sets the internal variable of the scan parameters. The values of the scan parameters take effect when a new scan is commenced. The function used to transmit the scan parameters to the microcontroller application has been added to Appendix 2 – Java send parameters routine. The functions used in the Java application to perform the synchronized transmission to the microcontroller can be found there.

## The microcontroller application

The microcontroller application is centered around an Arduino Uno board featuring the At Mega 328P microcontroller produced by the company Microchip. The program for the microcontroller application is designed using C programming language. The IDE used to develop the program is Atmel Studio 7.0, supported by Visual Studio. The Lidar Lite V3 sensor made by Garmin is used to collect distance measurements from the surrounding scenery. The system commands the sensor to collect distance measurements using a yawing scan method. Two 28 BYJ-48 stepper motors made by the company Kiatronics have been added in a mechanical setup. The position of the Lidar can be set at different angles in a two-axis coordinate system. The first stepper motor will rotate the sensor around the z-axis, controlling the yaw angle of the scanner. The turning angle is bounded due to the electromechanical limitations of the design. The second stepper motor will move the sensor along the vertical plane of the scanner, controlling the pitch angle of the scanner. When a turn around the z-axis is completed, the second motor steps along the z-axis. The first motor changes its direction of rotation for the following turn. The MPU 6050 accelerometer and gyroscope module made by the company InvenSense is used to control the pitch angle of the 3D scanner. At the beginning of a scan, the Lidar needs to be set at the starting pitch angle. A PID controller has been implemented in order to drive the shaft position of the second motor, assuring a correct pitch angle. The mechanical setup of the system integrates all the hardware components presented above. The program can perform one out of two operations: set scan parameters or perform a 3D yawing scan. The results of a scan are transmitted serially to the Java application. The electrical schematic of the project is illustrated in Appendix 3 – Electrical schematic. All the components are powered from the 5V supply of the Arduino Uno board. A 680 μF electrolytic capacitor has been placed in parallel with the power supply of the Lidar. The Lidar and the MPU6050 module are commanded using Two-Wire protocol. Two Darlington array drivers are used to command the stepper motors. Port D drives the first motor using pins 4, 5, 6 and 7 and Port B drives the second motor using pins 0, 1, 2 and 3.

### **Serial communication**

The Arduino Uno board communicates serially with the Java application via the Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART). The serial communication parameters are: Baud Rate equal to 76800 bits/second, 8-bit frame, 1 stop bit, no parity. In order to set the Baud Rate, a value must be written in the USART Baud Rate Register 0 (UBBR). Equation (15) from [37] is used to compute the UBBR value.

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

In Equation (15), fosc is the internal clock frequency of the microcontroller and is equal to 16 MHz [37]. The UBBR0 is a 16-bit register, meaning that the least significant 8 bits of the result are written in UBBR0L and the most significant 8 bits are written in UBBR0H. The Baud Rate value has been selected experimentally at 76800 bits/second with the aim of obtaining fast data transfer speed, while maintaining synchronization with the Java application.

The USART Control and Status Register 0 C (UCSR0C) is an 8-bit register used to set the other parameters of the serial communication. The 8-bit frame configuration is obtained by setting the UCSZ00 and UCSZ01 bits to 1. The option for one stop bit is selected by setting the USBS0 bit to 0. The parity is disabled by setting the UPM01 and UPM00 bits to 0. UCSR0B is an 8-bit register used to enable the USART transmitter and receiver. The transmitter is enabled by setting the TXEN0 bit to 1 and the receiver is enabled by setting the RXEN0 bit to 1. In order to transmit one 8-bit frame, the transmit buffer must be empty. UCSR0A is an 8-bit register used to track the status of the USART Data Register 0 (UDR0). The USART Transmit data buffer register (TXB) and the Receive data buffer register (RXB) share the same I/O address, referred to as UDR0 [37]. The data written in UDR0 is saved in TXB. Reading the UDR0 register location will return the contents of RXB. The status of the transmit buffer is monitored by checking the UDRE0 bit in UCSR0A. If the bit is equal to 1, then the buffer is empty and ready to be written. An 8-bit value can be transmitted by writing it to UDR0. The process of monitoring the transmit buffer then writing data to UDR0 is implemented in a function used to transmit a byte to the Serial port. The status of the receive buffer is tracked by checking the RXC0 bit in UCSR0A. The bit is set when there is unread data in receive buffer and cleared when the receive buffer is empty [37]. An 8-bit value can be retrieved by reading the UDR0 register when there is available data in the buffer. This process is implemented in a function used to read a byte from the serial port. When an integer value is transmitted, the transmission starts with the most significant digit. A function which reverts the digits of an integer number has been implemented to accomplish this task. The program checks if all the digits of the original integer value have been transmitted. This check is used to detect the case when the number ends with one or multiple zero digits. In this situation, the remaining zero digits of the integer value are transmitted. If the number is negative, the sign is transmitted first. The functions used to enable serial communication and transmit/receive data are presented at Appendix 4.

### I2C communication

The Arduino Uno communicates with the Lidar Lite V3 and with the MPU 6050 using the I2C communication protocol, developed by Philips Semiconductor. I2C communication is realized using two bi-directional bus lines: one for clock (SCL) and one for data (SDA) [37]. The address of a device which is connected to the I2C bus is a 7-bit value, which limits the maximum number of connected devices to 128. The application makes use of the dedicated SDA and SCL pins of the Arduino Uno board. Data transmission is enabled when the Master issues a Start condition and ends when the Master issues a Stop condition [37]. The microcontroller is the Master device and the sensors are the Slave devices. A Start condition occurs when the SDA signal transitions from High (H) to Low (L) when the SCL signal is H. The Stop condition is issued when the SDA transitions from L to H when the SCL signal is H. The address packets are 9-bit values consisting of 7 bits of address data, one Read/Write (R/W) bit and one Acknowledge (ACK) bit. If the R/W bit is 1, a read operation will be performed, otherwise a write operation is to be performed [37]. If the slave device recognizes the address, the SDA line is pulled low in the ninth clock cycle (ACK = 0). If the slave is busy, the SDA will remain high, setting the ACK bit to 1. A slave address packet followed by a R/W bit is called SLA+R or SLA+W [37]. The data is transmitted on the I2C bus using 9-bit packets consisting of one byte of data and one ACK bit. During transmission the master generates the SCL clock and issues the Start/Stop conditions, while the slave acknowledges the reception on the ninth SCL cycle [37]. If the receiver is busy, a NACK is sent after the last received byte. A transmission of data using I2C consists of issuing a Start condition, an SLA+R/W, one or more data packets and a Stop condition (Figure 6-3). The sequence used for writing data in one of the registers of a slave device is: start condition, SLA+W, write the 8-bit register address, write data to the register, stop condition. The sequence used to read data from a device’s register is similar: start condition, SLA+W, write the 8-bit register address, stop condition, start condition, SLA+R, read the value from the register, receive NACK, stop condition.

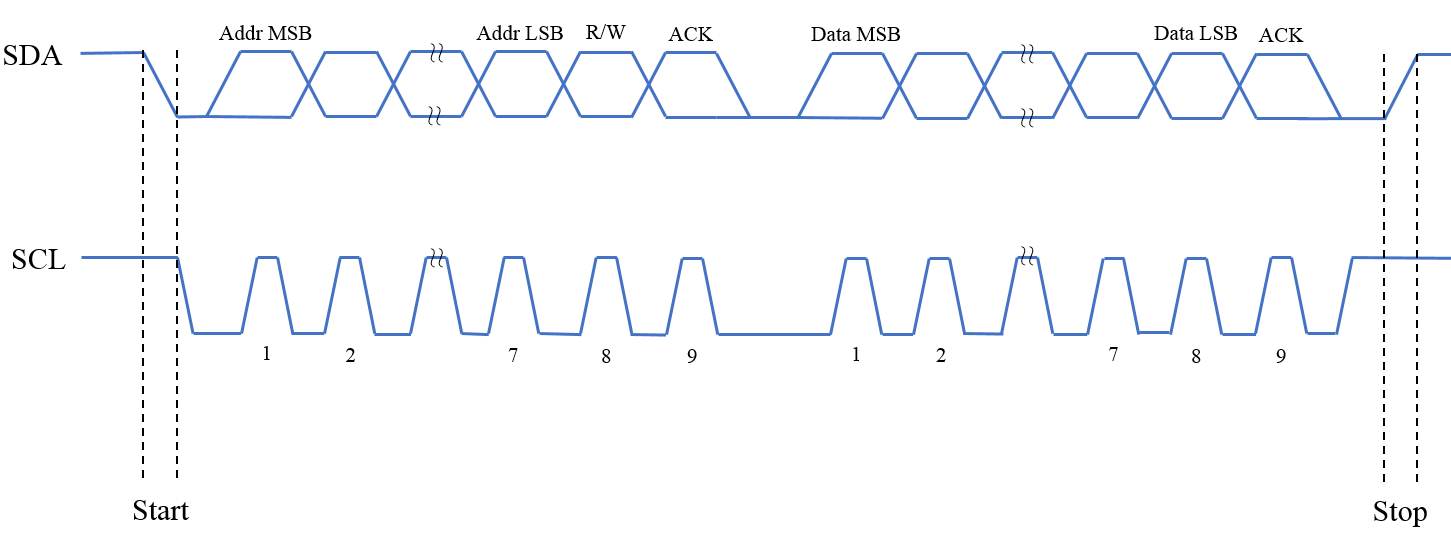


Figure 6‑3 I2C data transmission

The Bit Rate generator unit controls the frequency of the SCL signal when the microcontroller operates in master mode [37]. The SCL frequency is controlled using the settings in the TWI Bit Rate Register (TWBR) [37]. Both slave devices which are used in the application support the 400 kHz data transfer mode. Therefore, the frequency of the SCL signal is set to 400 kHz. According to [37], Equation (16) is used to compute the value of TWBR.

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

In Equation (16), fosc is the internal clock frequency of the microcontroller and fSCL is the desired frequency value for the SCL line. The TWI Status Register (TWSR) is used to monitor the communication bus. The most significant 5 bits of TWSR reflect the status of the serial bus, and the leas significant 2 bits (TWPS1 and TWPS0) control the bit rate prescaler. By setting the TWPS1 and TWPS0 bits to 0, the prescaler is equal to 1. The Two Wire Control Register (TWCR) is used to enable the I2C communication. The interface is initialized by setting the TWBR and setting the TWEN bit of TWCR to 1. A start condition is issued by setting the TWEN, TWSTA and TWINT bits to 1. Then, the TWINT flag is cleared and a start condition is issued as soon as the bus becomes free [37]. Once the start condition is transmitted, the hardware sets the TWINT flag to 1. The TWSR status value is set to 0x08 (Start code) if the Start condition has been issued successfully. A stop condition is issued by setting the TWEN, TWINT and TWSTO bits of TWCR to 1. The TWI Data Register (TWDR) is an 8-bit register which contains the next byte to be transmitted. In receiver mode, the TWDR contains the last byte received [37]. To communicate with a device using the I2C interface, first its address must be transmitted. The 7-bit address is written in TWDR, with the least significant bit (LSB) controlling the R/W operation. The next step is setting the TWEN and TWINT bits in TWCR. The TWINT flag is cleared and the transmission begins. The process is complete when TWINT is set back to 1. If the address was sent for a R operation, the TWSR status is set to 0x40 (SLA+R code). If a W operation was requested, the TWSR status is set to 0x18 (SLA+W code). In order to transmit a byte to the serial bus, the same set of operations are executed: write the byte to TWDR, set TWEN and TWINT, monitor TWINT until it is equal to 1, check TWSR. The status code for a successful data transmission is 0x28 (Master Transmitter data ACK). To read a byte from the I2C bus, the following sequence has been implemented: set TWEN and TWINT, monitor TWINT until it is equal to 1, check TWSR, read the value of TWDR. The status code for data reception is 0x58 (Master Receiver data NACK). With the help of examples from [37], a library has been designed for communicating with devices using I2C. The code of the Two-Wire library has been added to Appendix 5 - I2C Communication.

### Lidar Lite V3

The Lidar Lite V3 designed by Garmin (Figure 6-4) is the sensing element of the project. According to [Imp2], the operation mode of the sensor is based on a 905 nm laser with a beam width of 12 x 2 mm. The peak laser power is equal to 1.3 Watts and it is safe to look at even with an unaided eye. The device measures distance calculating the time of flight between transmitting a near-InfraRed laser signature and receiving the same signal [Imp2]. Distance can be calculated by using the known speed of light. In order to execute a distance measurement, the sensor undergoes a series of operations. First, a receiver bias correction routine is performed to calibrate the sensor to the light intensity of the environment [36]. Next, a reference signal is sent from the transmitter to the receiver and the signature of the laser beam is stored. Furthermore, the time delay is set to “zero” distance [36]. Then, the sensor performs a series of acquisitions and checks if the received signal matches the transmitted signal. If a match is detected, the result is stored in a correlation record and the next acquisition is summed with the previous result [36]. By repeating this process, a peak in the correlation record will emerge when the Lidar detects an object at a certain distance. The sensor stops initiating acquisitions when the acquisition count reaches a limit. The acquisitions are integrated until the signal peak in the correlation record reaches a maximum value [36]. The signal strength is computed from the magnitude of the signal record peak [36]. A valid signal threshold is calculated using the signal floor [36]. If the signal peak is above the threshold, then the measurement is considered valid and the sensor will return the measured distance in centimeters [36]. If the peak of the signal is below the threshold, then the sensor will report a distance equal to 1 cm. At the beginning of a new measurement, the signal record is cleared and the sequence starts again [36].



Figure 6‑4 Lidar Lite V3

The sensor module is connected to the 5V power supply of the Arduino Uno. Acting on the recommendation of [36], a 680 µF electrolytic capacitor is connected in parallel with the voltage supply wire and the ground connection wire. Upon power up the sensor performs a self-test sequence and the registers with their default values; distance measurements can be performed after a delay of roughly 22 ms [36]. The SDA and SCL wires of the sensor are connected to the SDA and SCL bi-directional bus lines of the Arduino board. The microcontroller is the Master device initiating the transmission, while the sensor is the Slave device. According to [36], the address of the Lidar is the 7-byte value of 0x62. Therefore, 8-bit address of the sensor is 0xC4 for writing and 0xC5 for reading. In order to obtain a distance measurement, the value 0x04 is written in the Acquisition Command register of the sensor (address 0x00). The sensor initiates a distance measurement with receiver bias correction. The LSB of the Status Register (address 0x01) indicates whether the sensor is busy or not. If the Busy flag is cleared, then the device is ready for a new command, otherwise the device is busy. The Busy flag is monitored until the distance measurement is completed. After that, the 16-bit result can be read by accessing the Full Delay Registers. The most significant byte of the measurement is stored in the Full Delay High register (address 0x0F) while the least significant byte of the result is stored in the Full Delay Low register (address 0x10). The functions developed for operating the Lidar Lite V3 have been added to Appendix 6.

Garmin has published an Arduino library for the Lidar which presents multiple configuration options for the sensor. The Default modeprovides a balanced performance with medium sensitivity and average speed. The Short Range, High Speedmode has a lower number of acquisitions for each measurement, which makes it ideal for fast systems which operate in closed environments. The maximum acquisition count can be changed by writing a new value in the Signal Count register (address 0x02). The Default Range, High Speed mode features quick termination of measurements when the device anticipates that the signal will reach its maximum value in the correlation record. The measurement quick termination is enabled by clearing the third bit of the Acquisition Configuration register (address 0x04). Otherwise, quick termination is disabled. The Maximum Rangemode performs the maximum number of acquisitions by setting the value of the Signal Count register to 0xFF. The High Sensitivity Detection mode comes with a lowered threshold bypass, which increases both the sensitivity of detection and the number of erroneous measurements. The threshold value can be changed by writing a new value in the Threshold Bypass register (0x1C). Additionally, the Low Sensitivity Detection has a higher threshold bypass, resulting in a lower sensitivity and less erroneous measurements.

### Stepper Motor 28 BYJ-48

Coming with a good angle/step resolution at a low cost, the 28 BYJ-48 stepper motor has been chosen. The 28 BYJ-48. Figure 6-5 (a), is a unipolar stepper motor suitable for a long range of applications including DVD drives or motion cameras. According to [38], the motor requires a 5V DC power supply and provides a resolution of 2048 steps for a 360° rotation. Therefore, the motor provides a resolution of 0.17578125° per step, which is a good accuracy for the application. However, due to the low operating voltage, the motor does not provide a high torque. The torque limitation can be adjusted by mounting an airscrew to the shaft. Overall, the 28 BYJ-48 is a compact, easy to use stepper motor with decent torque. Two identical stepper motors, along with two motor drivers, are used to achieve the second and third dimensions of the 3D scanner. The first motor rotates the sensor around the z-axis and the second motor moves the Lidar along the z-axis. The integrated circuit ULN2003 driver is used to control the motor. ULN2003 is a comprised of high-current Darlington Transistor arrays, ideal for driving stepper and DC brushed motors [40]. The ULN 2003 drivers are supplied from the 5V DC voltage source of the Arduino Board. The first stepper motor is commanded using Port D of the microcontroller, pins 4, 5, 6 and 7. The second stepper motor is driven using Port B, pins 0, 1, 2 and 3. The pins have been set in output mode by writing a logical 1 in the Data Direction Register corresponding to each port.

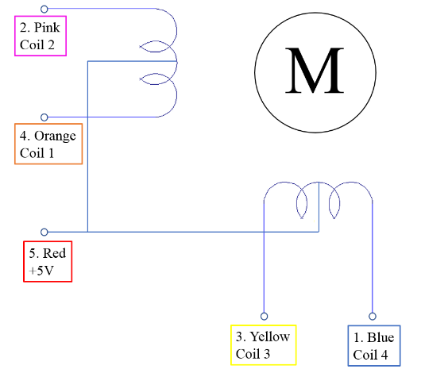
 

Figure 6‑5: 28 BYJ-48 stepper motor (a) and ULN2003 driver (b)

Figure 6-5 (b) illustrates the internal configuration of the unipolar stepper motor with 5 wires. The stepper motor has four coils, with one end of each coil connected to the +5V power supply of the Arduino Board. All the other ends (Blue, Pink, Yellow and Orange) are wired to Digital output ports of the microcontroller. Since one end of each coil is connected to a constant voltage source, current will flow through one of the coils only if the other end is grounded [39]. Therefore, a logical sequence is implemented to rotate the shaft of the motor in one direction or the other.

The sequence is implemented using the digital ports of the microcontroller. According to [39], a logical 1 means that the wire is energized and a logical 0 means that the wire is connected to ground. If a wire is connected to a logical 1, both ends of the coil are connected to a power supply and current will not flow through the coil. If a wire is connected to a logical 0, there is a voltage difference and current will flow through the coil. The motor is connected to the ULN 2003 driver which receives the logical sequence from the microcontroller. The input pins In1, In2, In3, In4of the driver are connected to the digital output pins of the microcontroller. Table 4-1 illustrates the logical sequence which needs to be implemented in order to drive the motor, according to [41]. The columns A, B and C contain the binary digits of the current step value. The current step value is an integer in the range [0, 7]. If the sequence of steps is incremental, the motor shaft counter-clockwise. When the value current step variable exceeds the upper limit, the variable is reset back to 0. If the sequence of steps is decremental, the shaft rotates clockwise. When the current step value is equal to 0, the variable is set to 7. The motor specifications require a minimum delay of 2 milliseconds between steps. The solution presented in [41] is inefficient because the logical command is generated using “switch … case” operations. A more efficient solution for computing the stepper motor command is using Boolean algebra. With the interest of expressing the value of each input as a function of the current step variable, Karnaugh maps have been implemented. By using this method, In1, In2, In3 and In4 are expressed in terms of A, B and C. Table 6-I illustrates the logical sequence corresponding to the current step value. The Karnaugh maps result from the values in Table 6-1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Current Step Value | | | Logical sequence | | | |
| A | B | C | In4 | In3 | In2 | In1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 |

*Table 6‑1 The logical sequence for driving the stepper motor*

Tables 6-2, 6-3, 6-4 and 6-5 illustrate the Karnaugh Map simplification for each logical input. In addition, Equations (17), (18), (19) and (20) are deduced from the Karnaugh Map simplifications. They represent the expression of the Boolean variables In4, In3, In2, In1 as a function of A, B, C. The program used to command the stepper motors has been added to Appendix 7 – Stepper motor.

|  |  |  |
| --- | --- | --- |
| C  AB | 0 | 1 |
| 0 0 | 0 | 0 |
| 0 1 | 0 | 0 |
| 1 1 | 1 | 1 |
| 1 0 | 0 | 1 |

*Table 6‑2 Karnaugh Map for In4*

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

|  |  |  |
| --- | --- | --- |
| C  AB | 0 | 1 |
| 0 0 | 0 | 0 |
| 0 1 | 0 | 1 |
| 1 1 | 0 | 0 |
| 1 0 | 1 | 1 |

*Table 6‑3 Karnaugh Map for In3*

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

|  |  |  |
| --- | --- | --- |
| C  AB | 0 | 1 |
| 0 0 | 0 | 1 |
| 0 1 | 1 | 1 |
| 1 1 | 0 | 0 |
| 1 0 | 0 | 0 |

*Table 6‑4 Karnaugh Map for In2*

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

|  |  |  |
| --- | --- | --- |
| C  AB | 0 | 1 |
| 0 0 | 1 | 1 |
| 0 1 | 0 | 0 |
| 1 1 | 0 | 1 |
| 1 0 | 0 | 0 |

*Table 6‑5 Karnaugh Map for In1*

|  |  |  |
| --- | --- | --- |
|  |  | (20) |

### MPU 6050 module

The MPU 6050 (Figure 6-6) is a device which combines a 3-axis gyroscope, a 3-axis accelerometer and a Digital Motion Processor. According to [42], the accelerometer provides multiple options of full-scale range. The full-scale range determines the sensitivity of the sensor using the following values: ±2g = 16384 LSB/g, ±4g = 8192 LSB/g, ±8g = 4096 LSB/g, ±16g = 2048 LSB/g. The accelerometer function of the module is necessary in setting the scanner at a correct starting pitch angle. The measurements collected from the accelerometer are delivered as a digital output. The communication between the module and the Arduino Board is done via I2C protocol. According to [42], the 7-bit address of the device is 0x68. Therefore, the address for reading is 0xD1 and the address for writing is 0xD0.

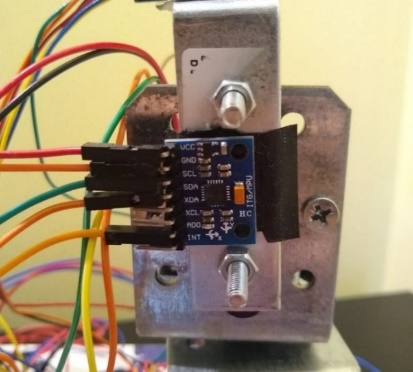


Figure 6‑6 MPU 6050 gyroscope and accelerometer module (a) relation between FS range and sensitivity (b)

For the current application a lower sensitivity is required for the accelerometer. Therefore, the full-scale range is set to ±16g by accessing the Accelerometer Configuration register (address 0x1C) and setting the AFS SEL bits to 1. Therefore, the sensitivity is set to 2048 LSB/g. The sensor can provide data measurements from all the coordinate axis: x-axis, y-axis and z-axis. The registers used for obtaining the accelerometer measurements are: X-H. X-L, Y-H, Y-L, Z-H and Z-L. The addresses of the registers start from 0x3B and end at 0x40 [43]. The results are 16-bit values, saved in 2’s complement format. The most significant byte of a value is stored in the H register, while the least significant bye is stored in the L register. The measurements for each axis can be obtained by combining the values of the H and L registers. The result is then converted into a decimal value and divided by the sensitivity of the accelerometer. The next step is converting the output of the sensor into an angle.

An application note on accelerometers was published by Analog Devices and can be found here: [44]. The article presents solutions for single-axis and double-axis tilt-angle calculation. In a single-axis system, the sine of the angle between the x-axis and the horizon is equal to the output acceleration generated by the gravity vector [44]. Equation (21) from [44] is used to compute the inclination angle, where AXOUT is the output of the sensor for one axis.

|  |  |  |
| --- | --- | --- |
|  |  | (21) |

The a double-axis system provides a major benefit due to the orthogonality of the axes [44]. The acceleration detected by the x-axis is proportional to the sine of the inclination angle [44]. Since the y-axis is perpendicular to the x-axis, the acceleration output of the y-axis is proportional to the cosine of the inclination angle. Therefore, the double-axis tilt is computed from the ratio of the two values, illustrated in Equation (22) from [44].

|  |  |  |
| --- | --- | --- |
|  |  | (22) |

Experimentally, it has been deduced that for the 3D it is relevant to measure the inclination angle between the x-axis and the y-axis. This deduction is influenced by the position of the sensor on the mechanical element of the scanner. In order to filter out potentially unwanted noise, an averaging filter has been implemented. Equation (23), from [44], illustrates the operation of an averaging filter applied on an array with n elements. The elements are summed up, then divided by n. The functions used to read data from the sensor and convert it to radians, as well as the code for the averaging filter can be found at Appendix 8 – MPU 6050.

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

A PID algorithm has been implemented with the goal of setting the 3D scanner at the initial pitch angle. The pitch angle is modified by driving the second stepper motor. A metallic connector is attached to the shaft of the motor. The connector consists of two metal plates joined at a 90-degree angle. The metallic connector is used to create two orthogonal planes. The accelerometer is secured on the first orthogonal plane. When the shaft of the stepper motor moves, the accelerometer starts to rotate rotates. The sensor can detect the two-axis tilt, which is the pitch angle of the 3D scanner. The Lidar is fixed on the second plane, perpendicular to the accelerometer plane. When the shaft of the stepper motor rotates, the Lidar is moving up and down, along the z-axis. The PID controller, along with the Process and the Sensor element comprise an industrial system. Figure 6-7 illustrates the Control Loop of the system, where Stepper motor is the Actuator, the 3D scanner is the Process and the Accelerometer as the Sensing element. The Set Point is the desired pitch angle of the system and the Process Variable is the current pitch angle measured by the accelerometer. The PID controller takes the Set Point and the Process Variable as inputs and outputs a number of steps [45]. The stepper motor executes these steps, driving the scanner closer to the desired pitch angle. The new pitch angle is measured and transmitted to the PID. The process is reiterated until the 3D scanner reaches its desired inclination. The software implementation of the PID control system has been added to Appendix 9 - Controller.

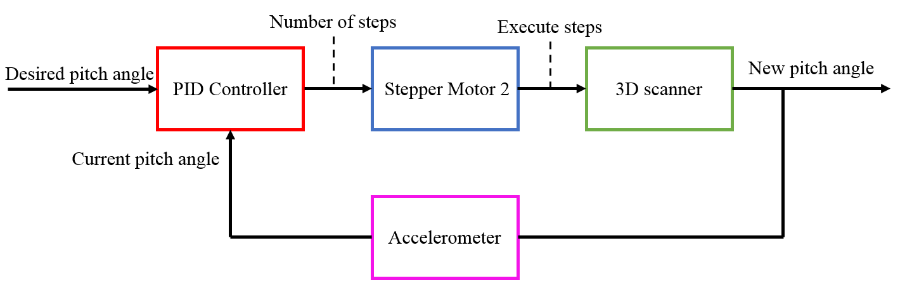


Figure 6‑7 The control loop of the system

### Set parameters routine

The main program consists of a loop which checks the Receiver buffer of the serial port. If the Set Parameters byte is received, then the program starts the routine for setting the scan parameters. The parameters are integer numbers, delimited by a ‘space’. The program reconstructs each parameter from the digits it receives from the Java application. The transmission starts with the most significant digit. If the parameter is negative, then the ‘minus’ character is sent first and a flag is raised. Each parameter is initially set to 0. The new digit is transformed from ASCII code to a decimal value. The current parameter is multiplied by 10 before adding the new digit to the result. The desktop application halts the transmission until the Continue Send byte is received from the microcontroller. When the new digit is set to the current scan parameter, the microcontroller sends the Continue Send byte, signaling it is ready to receive a new character. When a ‘space’ is received, it means that a parameter has been successfully sent. The program jumps to the next scan parameter. The process is repeated until the last scan parameter is successfully received, and the Scan Parameters character is received. The scan parameters are set as global variables in the microcontroller application. Their values are changed only when a new Set parameters routine is commenced.

### Yawing Scan routine

If the Start Scan character is received, the program enters the yawing scan routine. The rotation around the z-axis changes the yaw angle of the device and is executed by the first motor. The movement along the vertical plane changes the pitch angle and is commanded by the second stepper motor. First, the scanner is set at its initial pitch angle by making use of the control system: PID controller, accelerometer mounted on the metal connector and the second stepper motor. After that, the scanner rotates half the maximum yaw steps around the z-axis, in the direction opposite to the first turn of the scan. The turn process consists of three elements: distance measurement, serial transmission, execution of the yaw resolution steps. The Lidar collects distance measurements, starting at the yaw-reference position. The result is then transmitted serially, followed by a ‘space’ delimitation. The first stepper motor executes the yaw angle resolution steps before a new distance measurement is commenced. The total number of yaw steps are monitored by a counter. The cycle repeats until the counter reaches the maximum yaw steps. When the first stepper motor has executed the maximum yaw steps, the Lidar collects one extra distance measurement. A ‘newline’ is transmitted to the serial buffer as a delimitation between rows. The direction of the next turn is changed and the second stepper motor executes the pitch angle resolution steps. After that, a new turn starts, repeating the scan-transmit-step cycle until the maximum yawing steps are executed. An extra measurement is collected before the second stepper motor drives the scanner closer to the final pitch angle. A second counter is used to keep track of the current pitch angle of the scanner. The yawing scan consists of the following sequence of steps: turn, change direction, execute pitch resolution steps. When the pitch angle counter exceeds the final value, the yaw scan is complete. The scanner rotates half the maximum yaw steps in the direction opposite to the last turn. In this way, the device is set back in its initial position before the scan. The Start Scan character is then transmitted serially to the desktop application, signaling the end of the yawing scan. The Set parameters and Yawing scan routines have been added to Appendix 10 – Scanner. The main program of the microcontroller application has been added to Appendix 11- Main program.

# Experimental results

## 3D scanner setup

The mechanical setup of the 3D scanner has been designed using metal connectors used in construction. Figure 7-1 (a) illustrates the front view of the scanner. The first stepper motor is attached to the base of the setup with screws with the role of rotating the scanner around the z-axis. Therefore, the shaft of the first stepper motor is situated at the origin of the z-axis and is perpendicular to the horizontal plane of the scanner. A propeller is connected to the shaft of Motor 1. Two holes have been drilled in the propeller so that one face of a 90-degree connector can be attached to it using screws. Therefore, when Motor 1 executes steps, the second plane of the connector rotates around the shaft of the motor. Motor 2 is fixed to the interior side of the connector’s second face. The shaft of the second motor comes out on the exterior side of the orthogonal connector and is parallel to the x-y plane of the scanner. A propeller is fixed to the shaft of Motor 2 so that a 90-degree connector can be attached. The lateral view of the mechanical setup is illustrated in Figure7-1 (b). The MPU 6050 accelerometer module is secured on the exterior of the connector’s first face and is perpendicular to the shaft of Motor 2. The Lidar Lite V3 sensor is attached on the connector’s second plane, parallel to the shaft of Motor 2. Consequently, when the second motor is commanded, the accelerometer module rotates around the axis of the shaft and the Lidar moves along the vertical plane of the scanner. Due to cable connections, the mechanical setup can be used for scans with limited turn angles. Scans with turns up to 180° have proved to give good results. Additionally, the inclination angle of the second connector is limited mechanically. The pitch angle of the scanner is set in the range -120° and 120°. The scan parameters must be set according to the limitations of the system. The desktop application restricts the pitch angle values within the given boundaries.

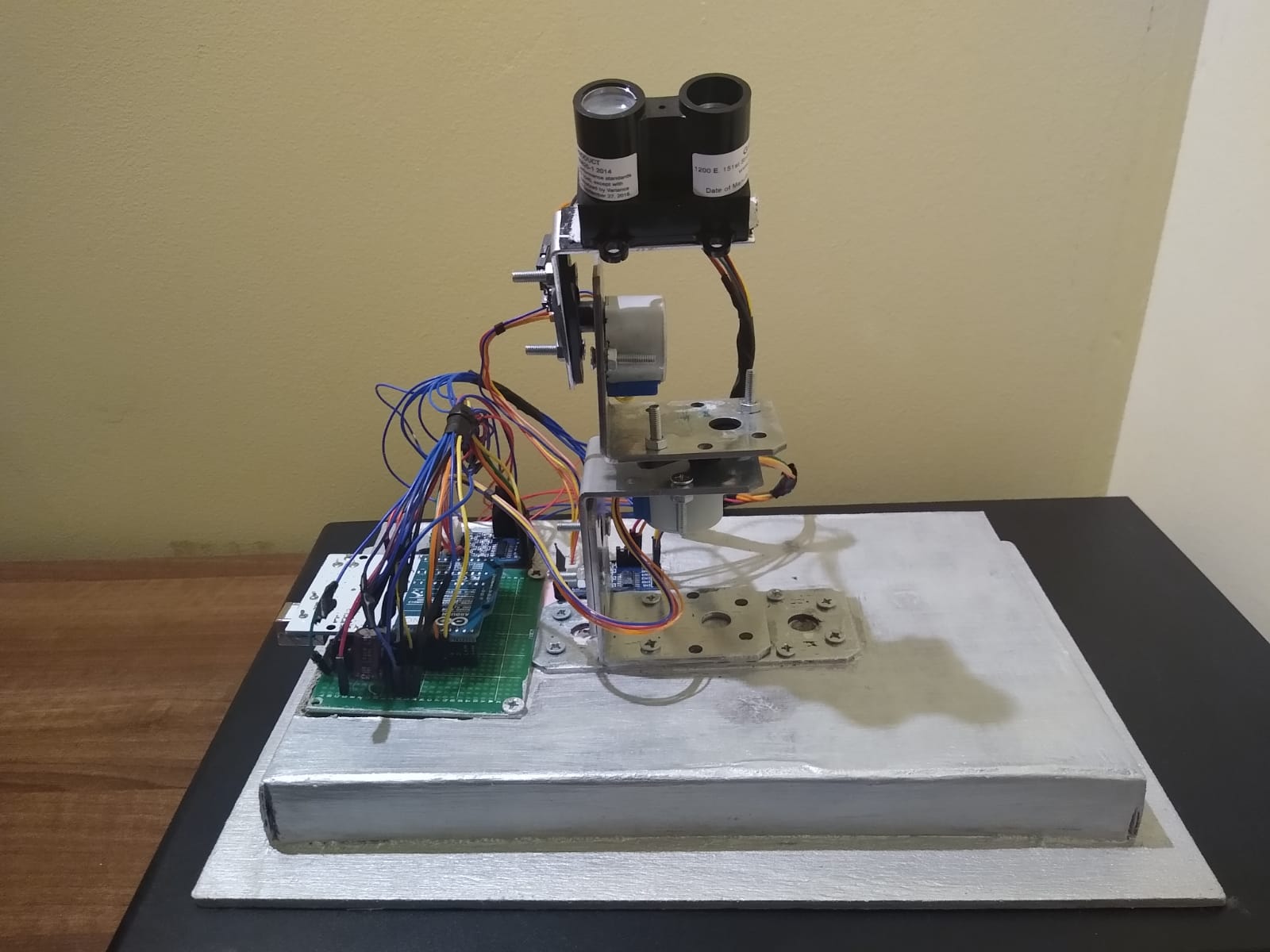
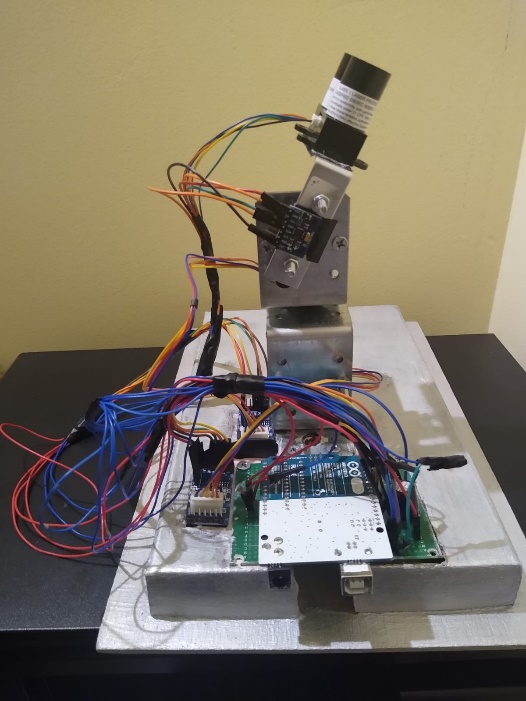
 

Figure 7‑1 Mechanical setup of the 3D scanner: Front view (a) and Lateral view (b)

## Scenario 1: Human Scan

The first scan experiment has been made in a simple indoor environment. A person is standing in an empty room at a distance of 2 m from the Lidar. The height of the person is approximately 175 cm. The scanner has been placed at an elevation of 45 cm. The goal of the experiment is to detect the form and height of a person, at a correct distance from the scanner. The scan parameters used in the experiment: Yaw angle resolution = 0.351° (2 steps), Maximum Yaw angle = 45° (256 steps), Pitch Angle resolution = 1.406° (8 steps), Maximum Pitch angle = -105°, Pitch Start angle = -40° and Lidar configuration = Maximum Range (code 3). The scan time is close to 8 minutes due to the increased yaw angle resolution. The resulting point cloud consists of 11997 points. Figure 7-2 illustrates the front view of the 3D plot. The lateral view of the representation is showed in Figure 7-3.

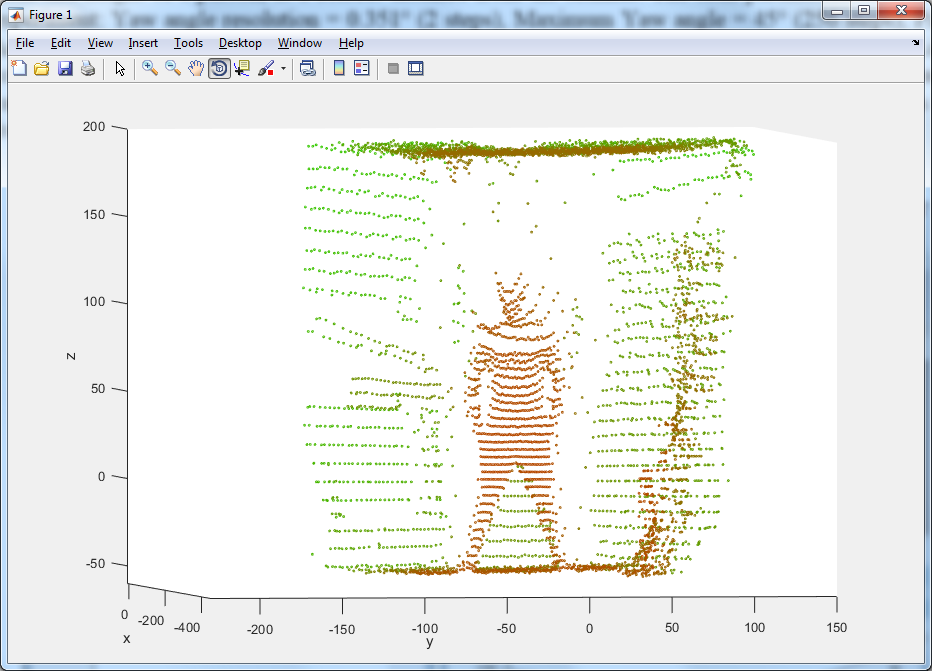


Figure 7‑2 Human scan results front view

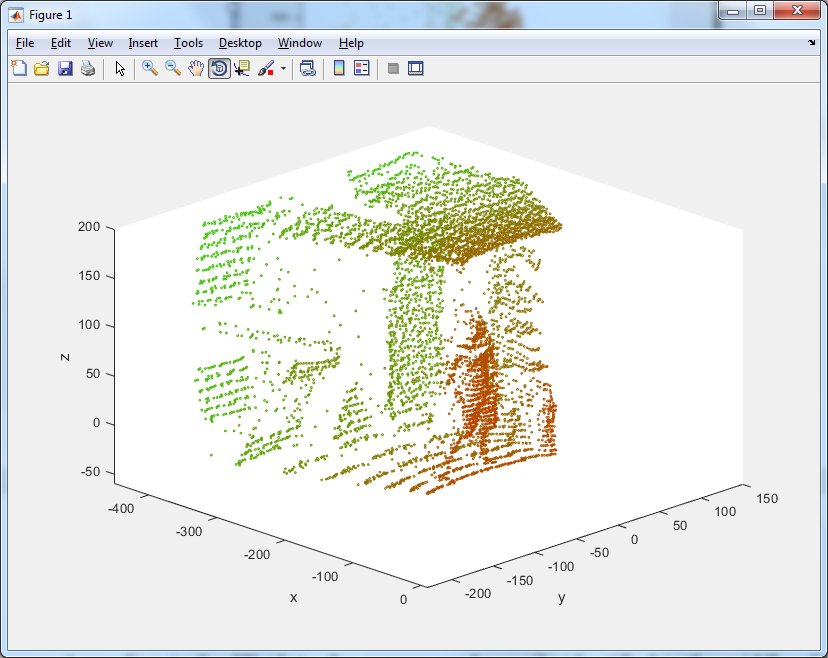


Figure 7‑3 Human scan results: Front view (a) and lateral view (b)

According to the 3D plots, the person can be easily identified in the middle of the front plane of the Lidar. Looking at the front view results, the shadow effect can be observed near the edges. Due to the yawing scan method a large number of points are detected on the surrounding walls. The person standing in front of the sensor is sensed with fewer points than the ceiling or the floor. The person has been detected in the range -50 and 120, resulting in a height of 170 cm. The result is consistent with the height of the person. Additionally, the 3D plot indicates a distance of approximately 2 m between the Lidar and the human, which corresponds to the experiment conditions. In conclusion, the first scan experiment confirms that the scanner is capable of sensing a person and places the data correctly in the 3D plane.

## Scenario 2: Rectangular object scan

The second scan scenario consists in detecting a rectangular object placed at 55 cm in front of a wall. The 3D scanner is placed in front of the rectangular cardboard box at a distance of 45 cm and at a distance of 1 m from the wall. The height of the box is 25.5 cm, while the width is equal to 15.5 cm. The objective of the second experiment is to detect the correct dimension of a rectangular object, at the correct distance from the scanner. Secondly, a test has been operated to detect the orientation of the object. For the first scan, the box is oriented with the flat surface in front of the 3D scanner, as shown in Figure 7-4. The scan parameters have been set to: Yaw angle resolution = 1.406° (8 steps), Maximum Yaw angle = 45° (256 steps), Pitch Angle resolution = 1.406° (8 steps), Maximum Pitch angle = -120°, Pitch Start angle = -30° and Lidar configuration = Default mode (code 0). The scan took approximatively 2 minutes to complete and consists of 2145 points.



Figure 7‑4 Object detection experimental setup

Figure 7-5 (a) illustrates the front view of the 3D plot, confirming that the shape of the object is reconstructed. The top view of the 3D plot scan is presented in Figure 7-5 (b) confirming that the rectangular object is in front of the wall.

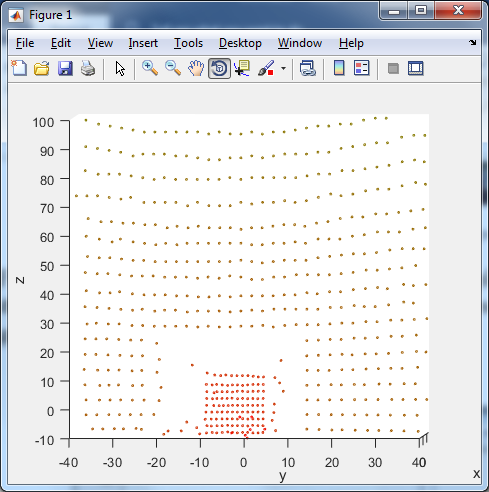
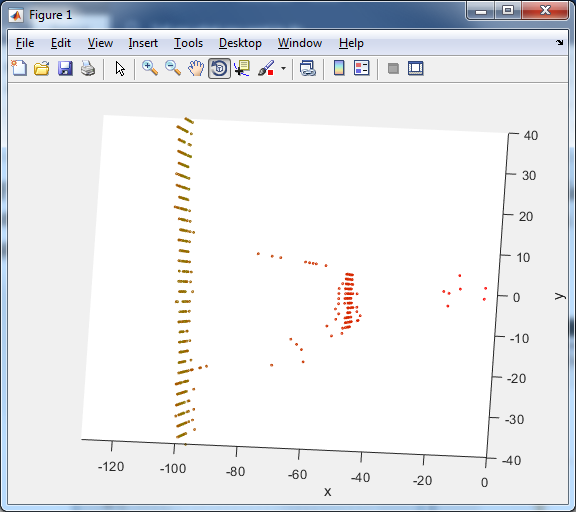
 

Figure 7‑5 Rectangular object scan results: Front view of the object (a) and top view of the object (b)

Shades of red and green have been used to visually represent the distance of a point from origin. According to the lateral representation, the distance between the box and the origin is equal to 47 cm. The distance between the wall and the origin is equal to 1 m. Therefore, the object is correctly placed in 3D space. Furthermore, flat surfaces such as walls and boxes are represented with minimal error. The lateral view illustrates a high density of points on the flat surface. It can be observed that a large density of points is collected on the wall behind the object. Looking at the front view of the object, the rectangular shape of the box can be easily distinguished. The lateral edges are projected between -10 and 5 on the y-axis, resulting in a width close to 15 cm. Some points are represented close to the edges of the box due to the shadow effect. The intensity of the effect is dependent on the position of the scanner and on the scan parameters. If the scan parameters allow for a greater density of points, then the shadow effect of the scan would be more visible. Additionally, the object is projected between -10 and 15 on the z-axis, resulting in a height close to 25 cm.

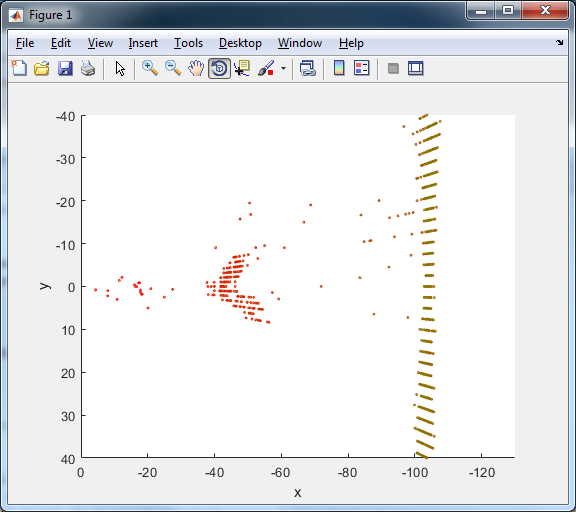
 

Figure 7‑6 Orientation detection: Experimental setup (a) and 3D plot (b)

Another scan has been commanded as part of the rectangular object experiment. This time, the objective is to detect the orientation of the object. The box has been rotated with 45° degrees towards the Lidar. The box is placed at the same distance from the 3D scanner and from the wall. The experimental setup for detecting the object orientation is illustrated in Figure 7-6 (a). The data has been collected using the same scan parameters as before. The top view of the object orientation plot is shown in Figure 7-6 (b). The front corner of the box can be identified, as well as the lateral faces of the box. Therefore, the system can also be used to detect the orientation of rectangular objects with respect to the position of the Lidar.

In conclusion, the second experiment has confirmed that the system is capable of detecting the shape and size of a rectangular object. The object positioning in 3D space is consistent with the measurements collected prior to the scan. In addition, the orientation of the box is represented accurately.

## Scenario 3: Interior room mapping

The third scenario consists in creating a 3D model of a room. The scanner setup is placed on a chair at a height of 45 cm. The scan parameters are the following: Yaw angle resolution = 0.351° (2 steps), Maximum Yaw angle = 180° (1024 steps), Pitch Angle resolution = 0.351° (2 steps), Maximum Pitch angle = -110°, Pitch Start angle = 0° and Lidar configuration = Maximum range (code 3). The maximum turn for the room mapping experiment is set to 180° with the goal of covering as much ground as possible. Due to higher resolutions for the pitch and yaw angles, as well as the larger yaw turn, the scan took 53 minutes to complete. Additionally, the final point cloud consists of 160569 measurements. The Lidar is placed in the middle of the room as shown in Figure 7-7.

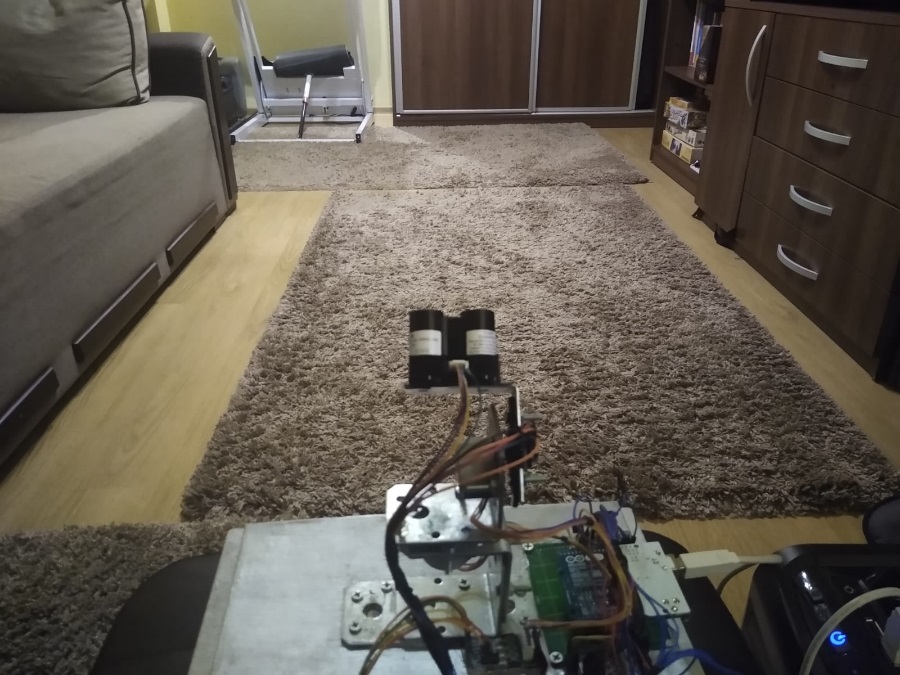


Figure 7‑7 Room mapping experimental setup 1

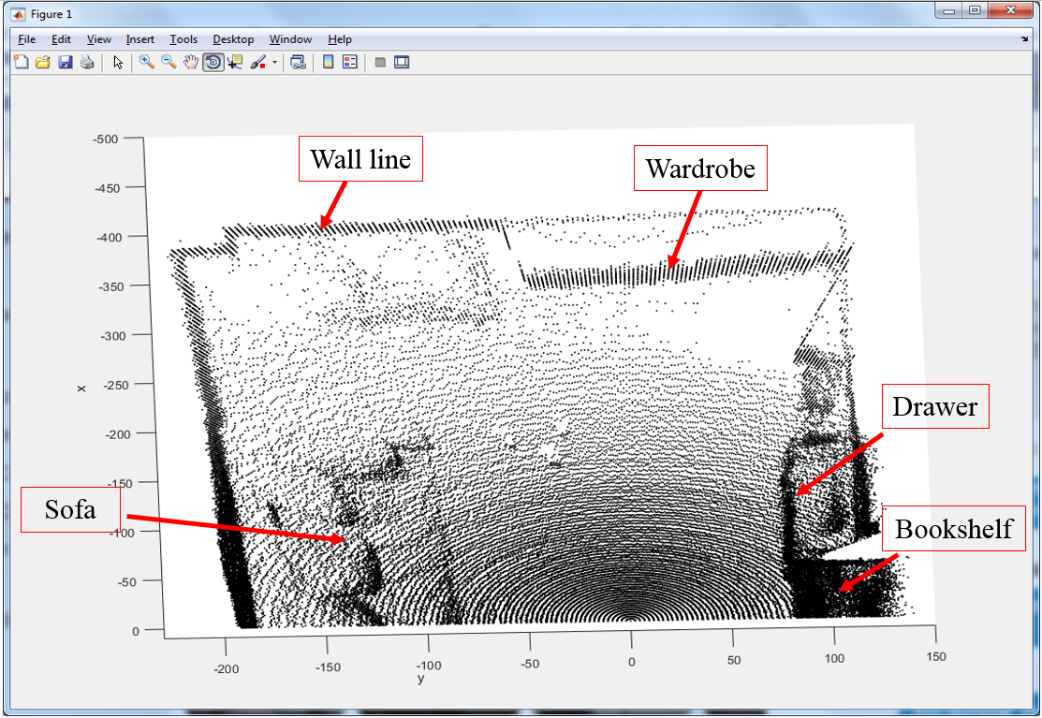


Figure 7‑8 Experiment 1: Room map, top view

Figure 7-8 illustrates the top view of the point cloud, making it easy to distinguish the surface of the room as well as the position of different objects. The wall lines are represented at the edges of the image. Based on the 3D plot, the approximated distances are: 4 m from the front wall, 2 m from the left-side wall and 1 m from the right-side. Additional objects have been sensed, including a wardrobe at 3m in the front-right view of the Lidar, a sofa at 1 m in the left view, a drawer at 70 cm in the right view and a bookshelf between the drawer and the Lidar end of turn. It can be observed that the ceiling area collected a large density of points due to the yawing scan method. Figure 7-9 illustrates the front view of the 3D plot, exposing a large density of points on the right-side bookshelf and on the left-side sofa. In addition, a better picture is shown in Figure 7-10, the right-side view of the plot in which the bookshelf is easily distinguishable.

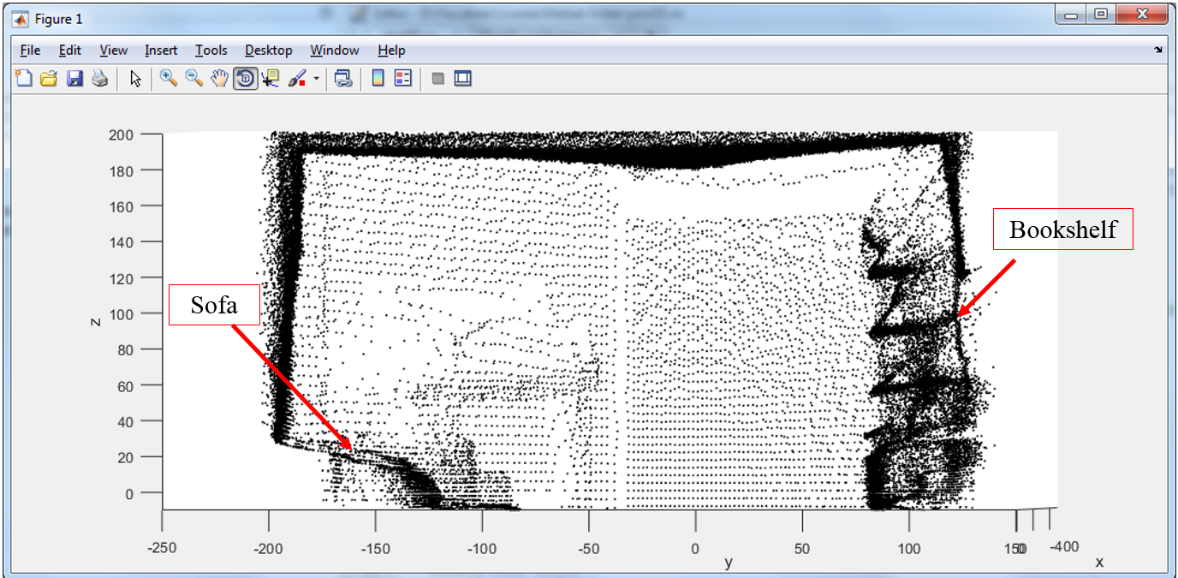


Figure 7‑9 Experiment 1: Room map, front view

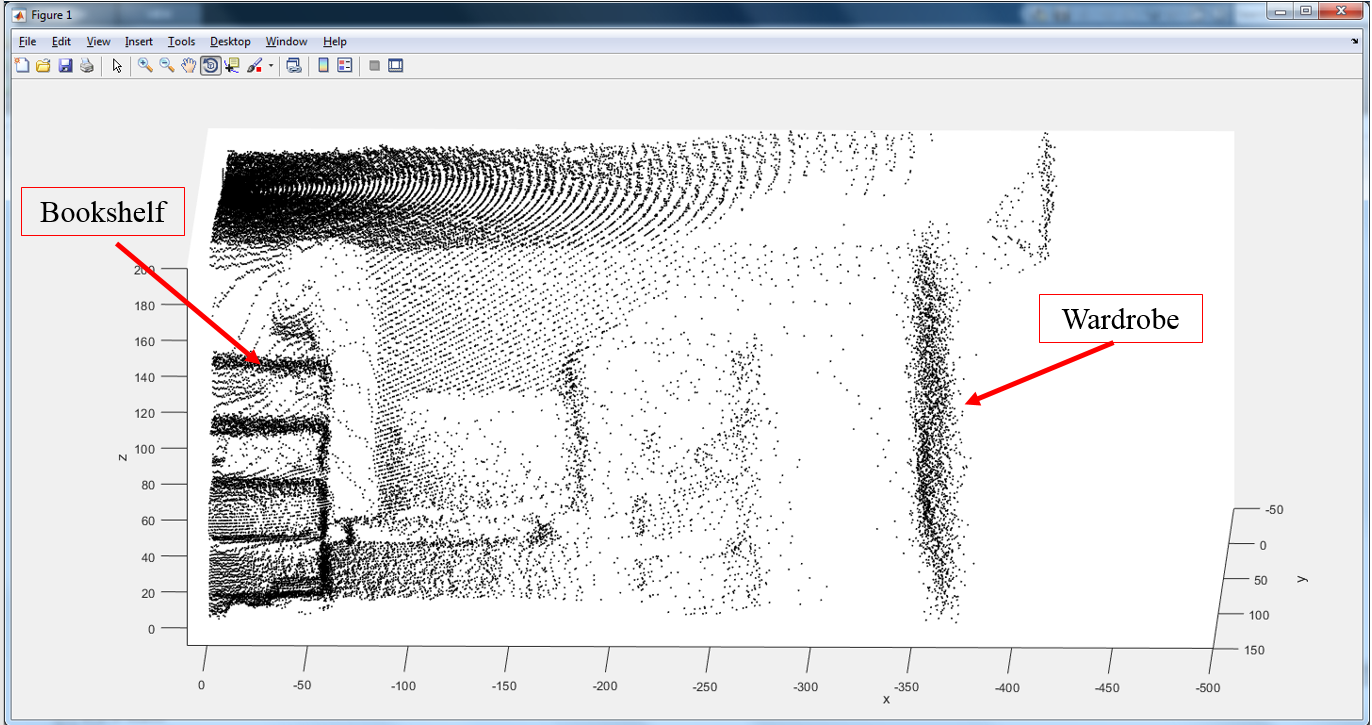


Figure 7‑10 Experiment 1: Room mapping, right-side view

Two more scans have been commanded as part of the room mapping experiment. The second scan aims to map the interior of a living room with two curtain areas and a flat screen TV. The scan parameters that are used are the same as the first room mapping experiment. The dimension of the point cloud is the same as well as the completion time of the scan. Figure 7-11 illustrates the scanning environment of the Lidar.

7‑11 Room mapping experiment setup 2

Figure 7-12 illustrates the top view of the point cloud from the second experiment. The structure of the room easily distinguishable, especially the walls. The areas which contain curtains as well as the flat screen TV can be identified from the top view of the results. In addition, the drawers and the cupboard have been detected by the sensor.

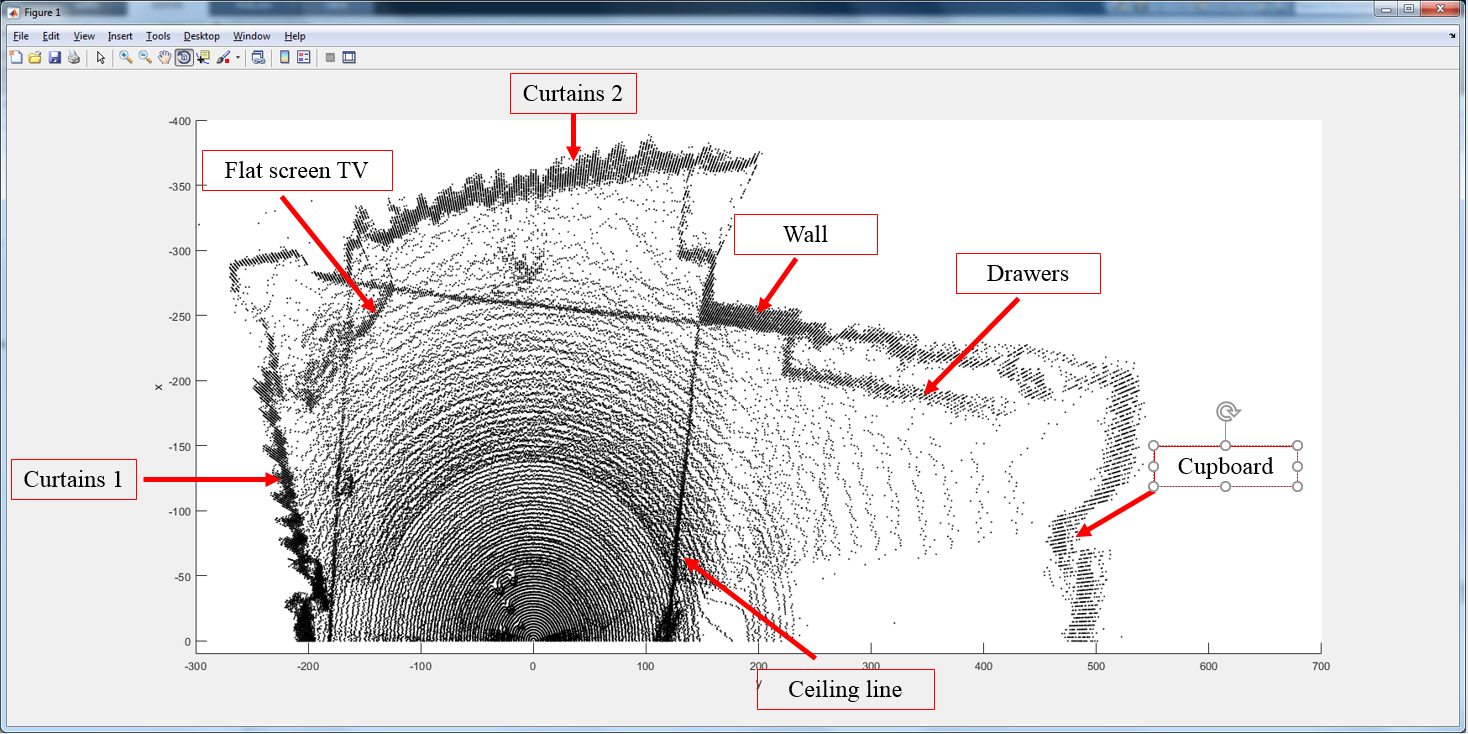


Figure 7‑12 Experiment 2: Room mapping top view

The third scan is executed in a kitchen room with the same scan parameters. Figure 7-13 illustrates the experimental setup and surroundings of the Lidar. The scanner was placed on a table with a height of 80 cm. The area contains many flat surfaces, which should make them easier to detect.

Figure 7‑13 Room mapping experiment setup 3

Figure 7-14 represents the top view of the point cloud of experiment 3. The room configuration can be identified. Three doors can be distinguished by the depth of the section. A better illustration is presented in Figure 7-15, the lateral view of the point cloud. Multiple pieces of furniture have been detected, including multiple drawers, cupboards and a window.

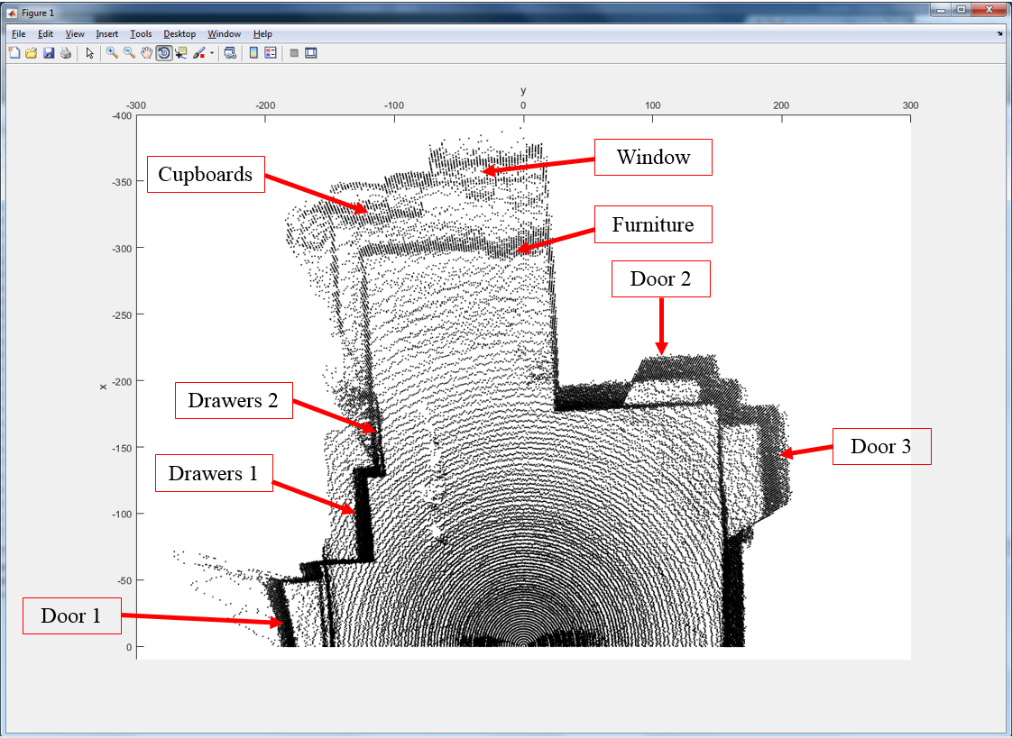


Figure 7‑14 Experiment 3: Room mapping top view

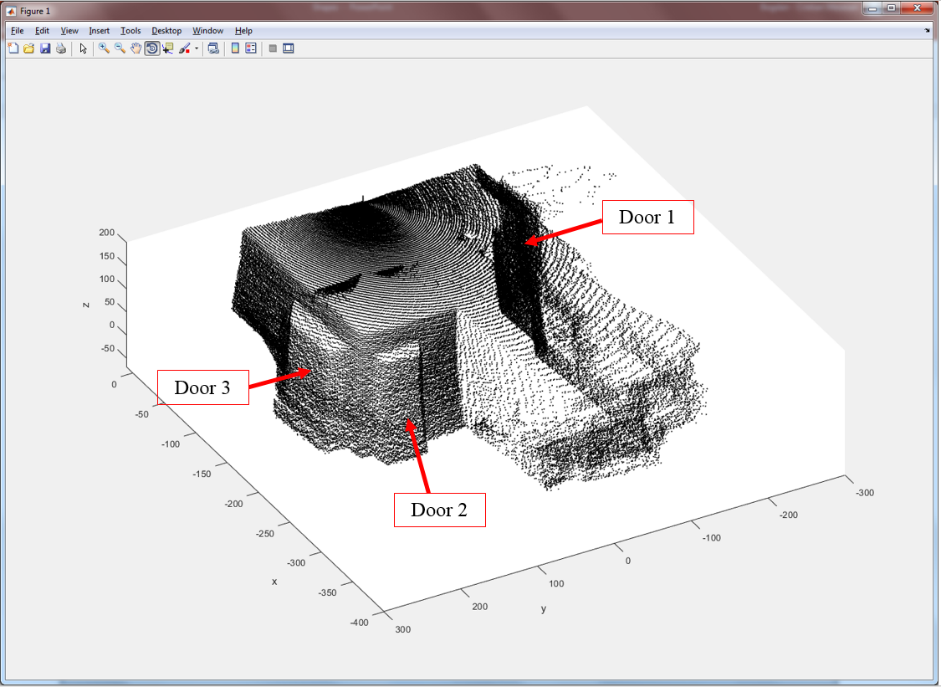


Figure 7‑15 Experiment 3: Room mapping lateral view

In conclusion, the room mapping experiments demonstrate that the system is capable of 3D reconstruction of scenery. The scanner yields its best results when detecting flat surfaces. The 3D plots are consistent with the configuration and dimensions of the rooms. Safety systems can be designed for industrial environments using interior mapping applications.

## Scenario 4: Object details detection

The fourth scenario tests whether the scanner is capable of detecting the form as well as the details of an object. A cup with multiple models has been placed at 0.5 m from the 3D scanner. The manual measurements of the cup indicate a height of 15 cm, a width of 7 cm at the bottom of the cup and a width of 17 cm at the cup handle. The Lidar is placed at a distance of 1 m from the back wall. The scan parameters are the following: Yaw angle resolution = 0.351° (2 steps), Maximum Yaw angle = 45° (256 steps), Pitch Angle resolution = 0.351° (2 steps), Maximum Pitch angle = -110°, Pitch Start angle = -45° and Lidar configuration = Default mode (code 0). The scan took 13 minutes to complete and the resulting point cloud has 23865 measurements. Figure 7-16 (a) illustrates the setup of the experiment.

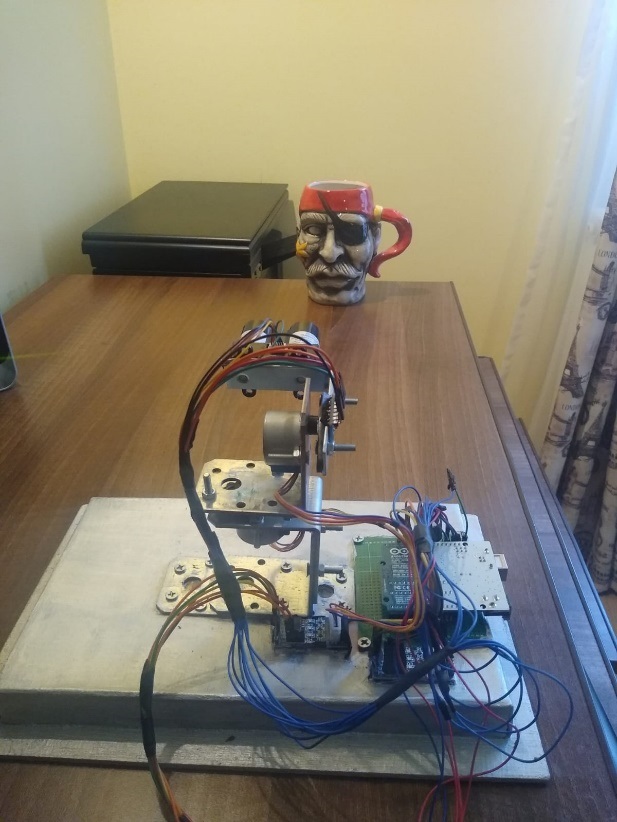
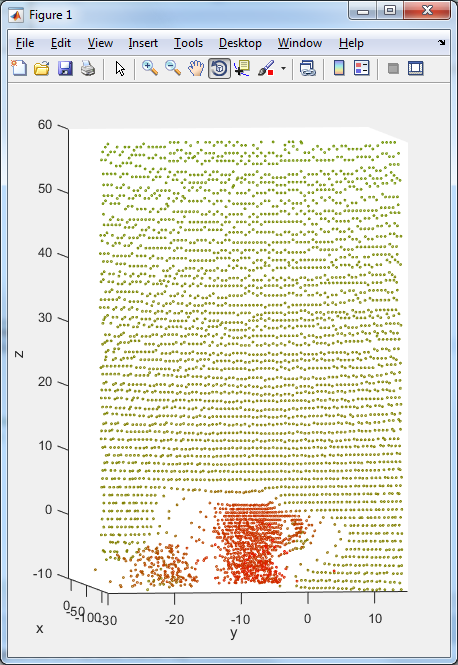
 

Figure 7‑16 Details detection: Experimental setup (a) and 3D plot (b)

Figure 7-16 (b) shows the resulting point cloud. From a quantitative point of view, the dimensions of the cup correspond with its manual measurements. The shape of the object can be distinguished in the 3D plot. However, the details and specific models of the object are distinguishable. Due to scanning a round object, as opposed to a flat surface object, the shadow effect is more intense around the edges of the cup. Since the object was not precisely placed in the middle of the Lidar’s field of view, the shadow effect is more intense. In conclusion, scenario 4 confirmed the second experiment by detecting the shape and the 3D dimensions of the object. The details of the object cannot be perceived in the point cloud.

## Errors

There are multiple sources of error in the system. A random error is unpredictable and cannot be replicated by repeating the experiment [46]. A systematic error is associated with bad experiment design or with a faulty equipment setup. The systematic error produces consistent errors which occur again if the experiment is repeated. The mechanical setup introduces instrumental errors due to the imperfect alignment of the mechanical parts. Additionally, while the scanner is performing measurements, the system is susceptible to vibration either from the shaft of the motor, either from the propeller attached to the shaft. The instrumentation error is doubled since the system uses two motors with propellers. A 3D model in specialized software like AutoCAD of the mechanical setup and using custom parts may aid in reducing the alignment error of the setup. A 3D printer can be used to obtain the necessary parts. In addition, the instrumental errors can be reduced by introducing a calibration algorithm. The field of view of the scanner can be plotted, and the field lines can be calibrated using a cross mapping algorithm for the odd and even lines.

Furthermore, an experiment has been set up to estimate the systematic errors of the 3D scanner in an ideal scenario. The Lidar is placed at a manually measured distance of 1 m from a wall. The Lidar will execute only one turn, collecting distance measurements from the wall. The scan parameters of the experiment are: Yaw angle resolution = 0.175° (1 step), Maximum Yaw angle = 22.5° (128 steps), Pitch Angle resolution = 0.175° (1 step), Maximum Pitch angle = -90°, Pitch Start angle = -90° and Lidar configuration = Default mode (code 0). Since the Lidar will execute only one turn, the maximum pitch angle and pitch starting position are equal. Additionally, the turn will be executed at a pitch angle equal to -90° so that the axis of the Lidar is orthogonal on the plane of the wall. The Cartesian coordinates corresponding to the x-axis reflect the distance between the front wall and the Lidar. Therefore, the mean value of the distance measurements is calculated using Equation (24) from [47]. The standard deviation of the measurements is computed using Equation (25) from [47]. The mean absolute error of a data set is calculated with Equation (26), from [48].

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

|  |  |  |
| --- | --- | --- |
|  |  | (26) |

The standard deviation of a set of data estimates the typical difference of each measurement from the mean. The sensor has collected NTheta = 129 distance measurements from the flat surface. The mean value of the data set is equal to -102.3873, resulting in a standard deviation equal to 0.99082, equal to 0.96772%. The mean absolute value is equal to 2.3917 %, from a measured distance of 1 m. The errors may be higher when detecting non-flat surfaces.

# Conclusions

We have designed a mechanical device capable of mapping its surrounding environment, including objects, humans and indoor surroundings. The output of the device consists of high-resolution point clouds. At the core of the 3D scanner lies the Lidar, an optical distance sensor working based on the Time of Flight principle. The Lidar typically includes an MPPC or an avalanche diode photodetector. Since the Lidar is a one-dimensional sensor, a mechanical setup has been developed to achieve the second and third dimensions. Two stepper motors are used to control the polar and the azimuthal angles of the Lidar. One accelerometer is utilized to set the sensor at a correct pitch angle, prior to the scan.

The 3D scanner implements a yaw scanning method based on rotating the Lidar around the upright z-axis. Multiple experiments have been conducted to test the performance of the system. The 3D scanner is capable of detecting flat surfaces and plot them in the correct position in field of view of the Lidar. Additionally, the form and dimensions of objects are consistent with the manual measurements collected prior to the experiment. The orientation of the objects with respect to the optical sensor can also be detected. Apart from objects and flat surfaces, the 3D scanner is able to distinguish a human silhouette in the resulting point clouds. The indoor mapping experiment yielded good results. The indoor objects were distinguishable and the dimensions of the space correspond to manual measurements. Safety systems for industrial applications can be developed using indoor mapping devices. Even though the 3D plot of an object illustrates correctly the form, the details are not identified. Systematic errors have been evaluated in an ideal case and yielded relatively low values. However, the biggest downside of the system is the time to complete a scan. A scan can last up to an hour and a half if the selected resolution is high and the maximum turn is set at 180°. The application can be integrated into a moving vehicle for outdoor scanning.

Many improvements and features can be added to the application. First, the mechanical design of the system can be created using a specialized software. The necessary parts can be produced using a 3D printer, contributing to minimizing errors. In addition, stepper motors with higher torque and better rotation speed can replace the ones used for the prototype. The accelerometer solution can be replaced with a more precise angle setting system. The non-continuous scanner can be transformed into a continuous scanner by replacing the cable connections with a slip ring. Gears can be added to increase the angle/step resolution of the motors by making use of the gear ratio. The desktop application and the data processing application can be merged and become one program which handles both the graphical user interface and the 3D plot of the results. A calibration algorithm would improve the 3D plot and give more accurate results. Research has uncovered many solutions, including Distortion mapping algorithms or Cross-mapping algorithms. Furthermore, the system can be transformed into a mobile device. The serial communication between the central unit and the microcontroller can be replaced with WLAN or Bluetooth communication protocols, while the device would be powered by a battery.

In conclusion, the initial goals of the project have been achieved. The experimental setup has proved to be effective in generating high-resolution point clouds. The project implementation illustrates the workflow in designing 3D mapping systems. The resulting 3D plots successfully map objects and indoor environments. Further development can be added to the application by improving the mechanical setup, selecting better hardware components or implementing a calibration algorithm.

# References

[1] The UCS USA website, <https://www.ucsusa.org/resources/self-driving-cars-101>

[2] The IOT for all website, <https://www.iotforall.com/5-autonomous-driving-levels-explained/>

[3] Analog Devices application note,

<https://www.analog.com/en/applications/markets/automotive-pavilion-home/autonomous-transportation-and-adas/autonomous-vehicle-perception-sensing.html>

[4] Geospatial World website, <https://www.geospatialworld.net/article/lidar-in-mapping/>

[5] Velodyne Lidar website, <https://velodynelidar.com/what-is-lidar/>

[6] Geospatial World website, <https://www.geospatialworld.net/blogs/lidar-technology/>

[7] marked study by Markets and markets, <https://www.marketsandmarkets.com/Market-Reports/lidar-market-1261.html>

[8] Velodyne Lidar website, <https://velodynelidar.com/success-stories/autonomoustuff/>

[9] Velodyne Lidar website, <https://velodynelidar.com/blog/robots-supporting-covid-19-and-public-health/>

[10] Luminar tech website, https://www-stg.luminartech.com/luminar-drives-expansion-into-automotive-market-with-over-250-million-total-raised/

[11] Luminar tech website, <https://www.luminartech.com/vcc/>

[12] Geospatial World website, <https://www.geospatialworld.net/blogs/find-goes-develop-autonomous-car/>

[13] Geospatial World website, <https://www.geospatialworld.net/blogs/why-lidar-is-important-for-autonomous-vehicle/>

[14] Semcon website,

<https://semcon.com/offerings/applied-autonomy/lidar-vs-radar-for-applied-autonomy/>

[15] Geospatial World website,

<https://www.geospatialworld.net/blogs/10-uav-lidar-products-creating-buzz/>

[16] Redtail Lidar website,

<https://redtaillidar.com/wp-content/uploads/2020/03/RedTail-LiDAR-RTL400-SpecSheet.pdf>

[17] Velodyne Lidar website, <https://velodynelidar.com/products/hdl-32e/>

[18] Riglusa website,

<http://products.rieglusa.com/product/all-categories-unmanned-scanners/vux-1uav-survey-grade-unmanned-laser-scanners>

[19] Faro website,

<https://www.faro.com/news/faro-launches-new-x-series-laser-scanner-the-focus3d-x-130/>

[20] USGS website, <https://pubs.usgs.gov/of/2007/1285/pdf/Giglierano.pdf>

[21] Maxim Integrated website, https://www.maximintegrated.com/en/design/blog/guide-to-lidar-sensors-for-self-driving-cars.html

[22] Hamamatsu website,

<https://www.hamamatsu.com/resources/pdf/ssd/Photodetector_lidar_kapd0005e.pdf>

[23] Hamamatsu website,

<https://www.hamamatsu.com/eu/en/product/optical-sensors/mppc/what_is_mppc/index.html>

[24] Hamamatsu website,

<https://www.hamamatsu.com/eu/en/product/optical-sensors/apd/index.html>

[25] RP Photonics website, <https://www.rp-photonics.com/avalanche_photodiodes.html>

[26] RP Photonics website, <https://www.rp-photonics.com/p_i_n_photodiodes.html>

[27] RP Photonics website, <https://www.rp-photonics.com/photocurrent.html>

[28] Sci. Instrum. 86, 053102 (2015); <https://doi.org/10.1063/1.4921195>

[29] Sensl company technical note: *An Introduction to the Silicon Photomultiplier*,

<https://www.sensl.com/downloads/ds/TN%20-%20Intro%20to%20SPM%20Tech.pdf>

[30] Oliver Wulf, Bernardo Wagner, *FAST 3D SCANNING METHODS FOR LASER MEASUREMENT SYSTEMS*, in the 14th International Conference on Control Systems and Computer Science – 2003

[31] Joan R. Rosell, Jordi Llorens, Ricardo Sanz, Jaume Arnó, Manel Ribes-Dasi, Joan Masip , Alexandre Escolà, Ferran Camp, Francesc Solanelles, Felip Gràcia, Emilio Gil, Luis Val, Santiago Planas, Jordi Palacín, *Obtaining the three-dimensional structure of tree orchards from remote 2D terrestrial LIDAR scanning*, Agricultural and Forest Meteorology, Volume 149, Issue 9, 1 September 2009, Pages 1505-1515

[32] García-Gómez, Pablo & Royo, Santiago & Rodrigo, Noel & Casas, Josep. (2020). *Geometric Model and Calibration Method for a Solid-State LiDAR*. Sensors. 20. 2898. 10.3390/s20102898.

[33] Pinliang Dong, Qi Chen, *LiDAR Remote Sensing and Applications* published December 28, 2017 by CRC Press

[34] Charles K. Toth, Jie Shan, *Topographic Laser Ranging and Scanning: Principles and Processing*, 18 november 2008

[35] Math Insight website, <https://mathinsight.org/spherical_coordinates>

[36] “Lidar Lite V3 Operation Manual and Technical Specifications”, Garmin, Revised 09/2016

[37] “ATmega48A/PA/88A/PA/168A/PA/328/P Data Sheet”, Microchip

[38] “28BYJ-48 – 5V Stepper Motor Data Sheet”, Kiatronics

[39] Components 101 website, <https://components101.com/motors/28byj-48-stepper-motor>

[40] “ULN200x, ULQ200x High-Voltage, High-Current Darlington Transistor Arrays datasheet”, Texas Instruments, Revised AUGUST 2019

[41] Robojax website, <https://robojax.com/>

[42] “MPU-6000 and MPU-6050 Product Specification Revision 3.4”, InvenSense Inc, 08/19/2013

[43] “MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2”, InvenSense Inc, 08/19/2013

[44] “Using an Accelerometer for Inclination Sensing, Analog Devices AN-1057 Application Note”, Christopher J. Fisher

[45] Control Solutions Minnesota, <https://www.csimn.com/CSI_pages/PIDforDummies.html>

[46] Statistics How to website, <https://www.statisticshowto.com/systematic-error-random-error/>

[47] BMJ website, <https://www.bmj.com/about-bmj/resources-readers/publications/statistics-square-one/2-mean-and-standard-deviation>

[48] Medium website, <https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d>

# Appendix

## Appendix 1 - Matlab script

%investigate 3D plotting

clear

clc

close all

format long;

%load the scan results from a text file

load results.txt;

%save the results in a two-dimensional array

r = results(:,:);

%load the scan parameters from a text file

load params.txt;

parameters = params(:,:);

%the angle/step resolution of the stepper motor

stepper\_angle\_constant = 0.17578125;

%save the scan parameters in different variables

yaw\_resolution = (parameters(1));

yaw\_max\_steps = (parameters(2));

pitch\_resolution = (parameters(3));

pitch\_max\_angle = (parameters(4));

pitch\_start\_position = (parameters(5));

%calculate the maximum yaw angle

MaxTheta = yaw\_max\_steps \* stepper\_angle\_constant;

%calculate the number of points in a turn

NTheta = yaw\_max\_steps / yaw\_resolution + 1;

%calculate the yaw angle resolution

IncrementTheta = stepper\_angle\_constant \* yaw\_resolution;

%initialize the yaw angle one-dimensional array

theta = (-MaxTheta/2):IncrementTheta:(MaxTheta/2);

negtheta = (MaxTheta/2):-IncrementTheta:(-MaxTheta/2);

%theta = (MaxTheta/2):-IncrementTheta:(-MaxTheta/2);

%calculate the pitch angle resolution

IncrementPhi = stepper\_angle\_constant \* pitch\_resolution;

%calculate the number of turns of a scan

%initialize the pitch angle one-dimensional array

if pitch\_start\_position - pitch\_max\_angle >= 0

NPhi = floor((pitch\_start\_position - pitch\_max\_angle) / IncrementPhi) + 1;

phi = (pitch\_start\_position):(-IncrementPhi):(pitch\_max\_angle);

else

NPhi = floor((pitch\_max\_angle - pitch\_start\_position) / IncrementPhi) + 1;

phi = pitch\_start\_position:(IncrementPhi):pitch\_max\_angle;

end

%initialize the two-dimensional arrays for the Cartesian coordinates

x = ones(NPhi, NTheta);

x = x.\*(-1000);

y = ones(NPhi, NTheta);

y = y.\*(-1000);

z = ones(NPhi, NTheta);

z = z.\*(-1000);

disp(['NTheta = ', num2str(NTheta)]);

disp(['NPhi = ', num2str(NPhi)]);

MaxDist = 2001;

MinDist = 1;

Interval = MaxDist - MinDist;

figure;

hold on;

%Transform spherical coordinates to Cartesian coordinates

for i = 1:1:NPhi

for j = 1:NTheta

if r(i,j) ~= 1

if r(i,j) <= 2000

if mod(i , 2) == 1

x(i, j) = r(i,j) \* sin(deg2rad(phi(i))) \* cos(deg2rad(theta(j)));

y(i, j) = r(i,j) \* sin(deg2rad(phi(i))) \* sin(deg2rad(theta(j)));

z(i, j) = r(i,j) \* cos(deg2rad(phi(i)));

else

y(i, j) = (r(i,j) \* sin(deg2rad(phi(i))) \* sin(deg2rad(negtheta(j))) + y(i - 1, j)) / 2;

x(i, j) = (r(i,j) \* sin(deg2rad(phi(i))) \* cos(deg2rad(negtheta(j))) + x(i - 1, j)) / 2;

z(i, j) = (r(i,j) \* cos(deg2rad(phi(i))) + z(i, j)) / 2;

end

Val = r(i,j) - MinDist;

ProcentG = (Val / Interval);

ProcentR = 1 - ProcentG;

%Color Plot

%plot3(x(i, j), y(i, j), z(i, j), 'o', 'MarkerSize', 1.7, 'markeredgecolor',[ProcentR, ProcentG, 0]);

end

end

end

end

%plot the results in 3D

plot3(x, y, z, 'ko', 'MarkerSize', 1);

xlabel('x');

ylabel('y');

zlabel('z');

xlim([-400, 10]);

ylim([-300, 300]);

zlim([-70, 200]);

## Appendix 2 - Java Send Parameters Routine

publicvoid **send\_parameters()** throws **IOException, InterruptedException {**

sp.setComPortTimeouts(SerialPort.***TIMEOUT\_WRITE\_BLOCKING***, 0, 0);

//Signal that the parameters will be changed

outstr.write(126);

System.***out***.println("\nSent: " + 126);

send\_Integer(yaw\_resolution);

send\_char(' ');

System.***out***.println("\nYaw resolution: " + yaw\_resolution);

send\_Integer(yaw\_max\_steps);

send\_char(' ');

System.***out***.println("\nYaw Max Steps: " + yaw\_max\_steps);

send\_Integer(pitch\_angle\_resolution);

send\_char(' ');

System.***out***.println("\nPitch angle resolution: " + pitch\_angle\_resolution);

send\_Integer(pitch\_max\_angle);

send\_char(' ');

System.***out***.println("\nPitch max angle: " + pitch\_max\_angle);

send\_Integer(pitch\_start\_position);

send\_char(' ');

System.***out***.println("\nPitch start position: " + pitch\_start\_position);

send\_Integer(lidar\_config);

send\_char(' ');

System.***out***.println("\nLidar config: " + lidar\_config);

//Signal that the operation ended

send\_char((**char**)(126));

System.***out***.println("\nSent: " + 126);

}

//checks if a new byte can be transmitted

**private** **void** checkSend() **throws** IOException {

**byte** x[] = **new** **byte**[2];

sp.setComPortTimeouts(SerialPort.***TIMEOUT\_READ\_SEMI\_BLOCKING***, 0, 0);

//the loop repeats until the character 125 has been received

**do** {

instr.read(x, 0, 1);

} **while** (x[0] != 125);

}

//sends a byte to the microcontroller

**public** **void** send\_char(**char** c) **throws** InterruptedException, IOException

{

//check if a new byte can be sent

checkSend();

//set the serial port in transmission mode

sp.setComPortTimeouts(SerialPort.***TIMEOUT\_WRITE\_BLOCKING***, 0, 0);

//Send a new byte

outstr.write((**byte**)(c));

}

//Sends an integer to the microcontroller

**public** **void** send\_Integer(**int** num) **throws** IOException, InterruptedException {

**int** val;

**int** cif;

// in case the number is negative

**if** (num < 0) {

send\_char('-');

num = num \* (-1);

}

// in case the number ends with a 0 we need to know the number of digits it has

cif = nr\_cif(num);

// we send the digits in reverse order

val = mirr\_number(num);

**while** (val > 0) {

// write digit by digit

send\_char((**char**) (val % 10 + 48));

val = val / 10;

cif--;

}

**if** (cif > 0 || num == 0) {

// case in which the original number ends with a '0' digit, we send out one more

// '0'

send\_char('0');

**if** (cif > 0)

cif--;

**while** (cif > 0) {

send\_char('0');

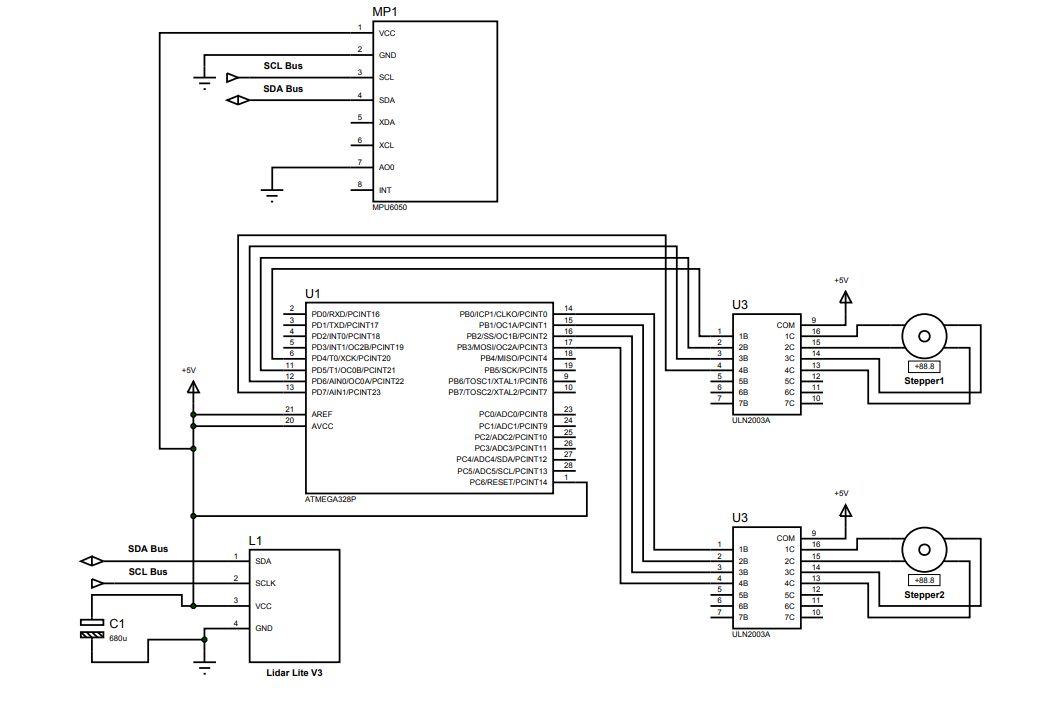
cif--;

}

}

}

## Appendix 3 – Electrical schematic



## Appendix 4 – Serial communication

### Header file

#ifndef SERIAL\_COMMUNICATION\_H\_

#define SERIAL\_COMMUNICATION\_H\_

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

//BAUD rate of the serial communication

#define BAUD 76800

#define BRC ((F\_CPU/16/BAUD) - 1)

//sets up the serial port, making it ready for communication

void Serial\_setup(void);

//Writes a single character on the serial line

void Serial\_write\_char(char c);

//Reads a single 8-bit value, incoming from the Serial port

unsigned char Serial\_read\_char(void);

//displays an integer number on the serial port

void Serial\_write\_int(int num);

//returns the mirrored number of the input

unsigned int mirr\_number(unsigned int num);

//returns the number of digits in a given number

unsigned int nr\_cif(unsigned int num);

//Displays a string on the serial port

void Serial\_write\_string(char\* str);

#endif /\* SERIAL\_COMMUNICATION\_H\_ \*/

### C file

#include "Serial\_Communication.h"

//sets up the serial port, making it ready for communication

void Serial\_setup(void)

{

//Enable Transmission and Receiver

UCSR0B = (1 << TXEN0) | (1 << RXEN0);

//8 bit frame, 1 stop bit, no parity

UCSR0C = (1 << UCSZ00) | (1 << UCSZ01);

//sets the High byte of the USART Baud Rate register

UBRR0H = (BRC >> 8);

//sets the Low byte of the USART Baud Rate register

UBRR0L = BRC;

}

//Writes a single character on the serial line

void Serial\_write\_char(char c)

{

/\* Wait for empty transmit buffer \*/

while ( !( UCSR0A & (1 << UDRE0) ) )

;

/\* Put data into buffer, sends the data \*/

UDR0 = c;

}

//Reads a single 8-bit value, incoming from the Serial port

unsigned char Serial\_read\_char(void)

{

/\* Wait for data to be received \*/

while ( !(UCSR0A & (1<<RXC0)) )

;

/\* Get and return received data from buffer \*/

return UDR0;

}

//displays an integer number on the serial port

void Serial\_write\_int(int num)

{

unsigned int val;

unsigned int cif;

//in case the number is negative

if (num < 0)

{

Serial\_write\_char('-');

num = num \* (-1);

}

//we send the digits in reverse order, starting with the most significant

//the number is mirrored

val = mirr\_number(num);

//the number of digits in a given number

cif = nr\_cif(num);

while (val > 0)

{

//write the digits serially

Serial\_write\_char(val % 10 + 48);

val = val / 10;

cif--;

}

if (cif > 0 || num == 0)

{

//case in which the original number ends with a '0' digit, we send out one more '0'

Serial\_write\_char('0');

if (cif > 0)

cif--;

//in the case where the number ends with more than one '0' digit

while (cif > 0)

{

Serial\_write\_char('0');

cif--;

}

}

}

//returns the mirrored number of the input

unsigned int mirr\_number(unsigned int num)

{

unsigned int val = 0;

while (num > 0)

{

val = val \* 10 + num % 10;

num = num / 10;

}

return (val);

}

//returns the number of digits in a given number

unsigned int nr\_cif(unsigned int num)

{

unsigned int sum = 0;

while (num > 0)

{

sum++;

num = num / 10;

}

return (sum);

}

//Displays a string on the serial port

void Serial\_write\_string(char\* str){

unsigned int i = 0;

while (str[i] != 0)

{

Serial\_write\_char(str[i]);

i++;

}

}

## Appendix 5 – I2C communication

### Header file

#ifndef TWO\_WIRE\_H\_

#define TWO\_WIRE\_H\_

#include "Serial\_Communication.h"

//value of the Bit rate generator register

//set the fscl to 400kHz

#define TWBR\_VAL ((F\_CPU / 400000UL) - 16) / 2

//TWSR codes for different situations during the I2C transmission

#define START 0X08

#define SLAW\_ACK 0x18

#define SLAW\_NACK 0X20

#define MT\_DATA\_ACK 0x28

#define MT\_DATA\_NACK 0x30

#define MR\_DATA\_ACK 0x50

#define MR\_DATA\_NACK 0x58

#define SLAR\_ACK 0x40

#define SLAR\_NACK 0x48

#define MPU\_6050\_READ 0b11010001

#define MPU\_6050\_WRITE 0b11010000

//initiates the the pre-scaler and the TWBR register

//sets the SCL frequency, with the formula: F = (F\_CPU)/(16 + 2 \* TWBR \* Prescaler\_value)

void I2c\_init(void);

//Starts the I2C communication

void I2C\_start\_condition(void);

//Stops the I2C communication

void I2C\_stop\_condition(void);

//Sends the sensor address for writing purpose

void I2C\_sensor\_addr\_write(unsigned int addr);

//Sends the sensor address for reading purpose

void I2C\_sensor\_addr\_read(unsigned int addr);

//Sends the register address which needs to be accessed

void I2C\_send\_register\_address(unsigned int addr);

//Sends one byte of data to the slave

void I2C\_data\_write(unsigned int data);

//reads one byte of data from the current accessed register and returns it

unsigned int I2C\_read\_byte(void);

#endif /\* TWO\_WIRE\_H\_ \*/

### C file

//sets the SCL frequency, with the formula: F = (F\_CPU)/(16 + 2 \* TWBR \* Prescaler\_value)

//F\_SCL = 6250Hz, T\_SCL = 160us

//F\_SCL = 400khZ / 16

void I2c\_init(void)

{

//Setting the value of the Bit rate register

TWBR = TWBR\_VAL;

TWCR |= (1 << TWEN);

Serial\_write\_string("\nI2C Init ok!");

}

//Starts the I2C communication

void I2C\_start\_condition(void)

{

//Sets the bits necessary in the control register

TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if TWSR has the correct code after START

if ((TWSR & 0xF8) != START)

Serial\_write\_string("\nI2C START ERROR!");

//else

// Serial\_write\_string("\nI2C START OK!");

}

//Stops the I2C communication

void I2C\_stop\_condition(void)

{

TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);

*\_delay\_ms*(1);

}

//Sends the sensor address for writing purpose

void I2C\_sensor\_addr\_write(unsigned int addr)

{

//put the address you want to send in the data register

TWDR = addr;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for ADDRESS WRITE ACK

if ((TWSR & 0xF8) != SLAW\_ACK)

Serial\_write\_string("\nI2C SLAW\_ACK ERROR!");

//else

// Serial\_write\_string("\nI2C SLAW\_ACK OK!");

}

//Sends the sensor address for reading purpose

void I2C\_sensor\_addr\_read(unsigned int addr)

{

//put the address you want to send in the data register

TWDR = addr;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for ADDRESS READ ACK

if ((TWSR & 0xF8) != SLAR\_ACK)

Serial\_write\_string("\nI2C SLAR\_ACK ERROR!");

//else

// Serial\_write\_string("\nI2C SLAR\_ACK OK");

}

//Sends the register address which needs to be accessed

void I2C\_send\_register\_address(unsigned int addr)

{

//put the address you want to send in the data register

TWDR = addr;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for MASTER TRANSMITTER MODE ACK

if ((TWSR & 0xF8) != MT\_DATA\_ACK)

Serial\_write\_string("\nI2C MT\_DATA\_ACK-register ERROR!");

//else

// Serial\_write\_string("\nI2C MT\_DATA\_ACK-register OK");

}

//Sends one byte of data to the slave

void I2C\_data\_write(unsigned int data)

{

//put the data you want to send in the data register

TWDR = data;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for MASTER TRANSMITTER MODE ACK

if ((TWSR & 0xF8) != MT\_DATA\_ACK)

Serial\_write\_string("\nI2C MT\_DATA\_ACK-data ERROR!");

//else

// Serial\_write\_string("\nI2C MT\_DATA\_ACK-data OK");

}

//reads one byte of data from the current accessed register and returns it

unsigned int I2C\_read\_byte(void)

{

//initiate the communication

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for MASTER RECEIVER MODE

if ((TWSR & 0xF8) != MR\_DATA\_NACK)

Serial\_write\_string("\nI2C MR\_DATA\_NACK ERROR!");

//else

// Serial\_write\_string("\nI2C MR\_DATA\_NACK OK!");

//return the data received from the slave

return (TWDR);

}

## Appendix 6 – Lidar Lite V3

### Header file

#ifndef LIDAR\_LITE\_V3\_H\_

#define LIDAR\_LITE\_V3\_H\_

#include "Two\_Wire.h"

//the LIDAR address for reading purposes

#define LIDAR\_READ 0b11000101

//the LIDAR address for writing purposes

#define LIDAR\_WRITE 0b11000100

//The register addresses of the Garmin LIDAR\_LITE\_V3 Sensor

#define ACQ\_COMMAND 0X00

//Status register

#define STATUS\_REG 0x01

//Maximum number of acquisitions during measurement

#define SIG\_COUNT\_VAL 0X02

//Acquisition mode control

#define ACQ\_CONFIG\_REG 0X04

//Velocity measurement output

#define VELOCITY 0x09

//Peak value in correlation record

#define PEAK\_CORR 0X0C

//Correlation record noise floor

#define NOISE\_PEAK 0X0D

//Received signal strength

#define SIGNAL\_STRENGTH 0X0E

//Distance measurement high byte

#define FULL\_DELAY\_HIGH 0X0F

//Distance measurement low byte

#define FULL\_DELAY\_LOW 0X10

//Burst measurement count control

#define OUTER\_LOOP\_COUNT 0X11

//Reference acquisition count

#define REF\_COUNT\_VAL 0X12

//Previous distance measurement high byte

#define LAST\_DELAY\_HIGH 0x14

//Previous distance measurement low byte

#define LAST\_DELAY\_LOW 0x15

//Serial number high byte

#define UNIT\_ID\_HIGH 0x16

//Serial number low byte

#define UNIT\_ID\_LOW 0x17

//Write serial number high byte for I2C address unblock

#define I2C\_ID\_HIGH 0x18

//Write serial number low byte for I2C address unblock

#define I2C\_ID\_LOW 0X19

//Write new I2C address after unblock

#define I2C\_SEC\_ADDR 0X1A

//Peak detection threshold bypass

#define THRESHOLD\_BYPASS 0X1C

//default address response control

#define I2C\_CONFIG 0X1E

//state command

#define COMMAND 0X40

//delay between automatic measurements

#define MEASURE\_DELAY 0X45

//second largest peak value in correlation record

#define PEAK\_BCK 0X4C

//correlation record low byte

#define CORR\_DATA 0X52

//correlation record high byte

#define CORR\_DATA\_SIGN 0X53

//correlation record memory bank select

#define ACQ\_SETTINGS 0X5D

//Power state control

#define POW\_CONTROL 0X65

//writes the 8-bit data to one of the registers

void LIDAR\_write\_data\_to\_register(unsigned int addr, unsigned int data);

//returns the 8-bit data from one of the registers

unsigned int LIDAR\_read\_data\_from\_register(unsigned int addr);

//returns the distance in centimeters

unsigned int LIDAR\_distance\_measurement(void);

//Sets up the sensing parameters, depending on the desired mode of operation

//The recommended configurations were taken from the library provided by Garmin

void LIDAR\_configuration(unsigned int mode);

//wakes up the LIDAR and performs a series of measurements in order to stabilize the sensor

//write 0x00 to PWR management register

void LIDAR\_Power\_Up(void);

//LIDAR enters sleep mode

//write 0x04 to PWR management register

void LIDAR\_Power\_Down(void);

#endif /\* LIDAR\_LITE\_V3\_H\_ \*/

### C file

#include "LIDAR\_LITE\_V3.h"

//writes the 8-bit data to one of the registers

void LIDAR\_write\_data\_to\_register(unsigned int addr, unsigned int data)

{

//start the I2C communication

I2C\_start\_condition();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_write(LIDAR\_WRITE);

//send the selected register address to the slave

I2C\_send\_register\_address(addr);

//send data to the selected register

I2C\_data\_write(data);

//stop the communication

I2C\_stop\_condition();

}

//returns the 8-bit data from one of the registers

unsigned int LIDAR\_read\_data\_from\_register(unsigned int addr)

{

unsigned int data;

//start the I2C communication

I2C\_start\_condition();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_write(LIDAR\_WRITE);

//send the selected register address to the slave

I2C\_send\_register\_address(addr);

//stop the communication

I2C\_stop\_condition();

//start the I2C communication

I2C\_start\_condition();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_read(LIDAR\_READ);

//read one byte from the slave

data = I2C\_read\_byte();

//stop the communication

I2C\_stop\_condition();

return (data);

}

//returns the distance in centimeters

unsigned int LIDAR\_distance\_measurement(void)

{

unsigned int data;

unsigned int distanceH;

unsigned int distanceL;

//Write the acquisition command to the ACQ\_COMMAND register

LIDAR\_write\_data\_to\_register(ACQ\_COMMAND, 0x04);

do

{

//monitor the first bit of the status register, to know when the measurement is done

data = LIDAR\_read\_data\_from\_register(STATUS\_REG);

} while ((data & 0x01));

//extract the HIGH byte of the measurement

distanceH = LIDAR\_read\_data\_from\_register(FULL\_DELAY\_HIGH);

//extract the LOW byte of the measurement

distanceL = LIDAR\_read\_data\_from\_register(FULL\_DELAY\_LOW);

//put together the high byte and the low byte

distanceH = distanceH << 8;

distanceH = distanceH + distanceL;

//return the result

return (distanceH);

}

//Sets up the sensing parameters, depending on the desired mode of operation

//The recommended configurations were taken from the library provided by Garmin

/\* Parameters

------------------------------------------------------------------------------

configuration: Default 0.

0: Default mode, balanced performance.

1: Short range, high speed. Uses 0x1d maximum acquisition count.

2: Default range, higher speed short range. Turns on quick termination

detection for faster measurements at short range (with decreased

accuracy)

3: Maximum range. Uses 0xff maximum acquisition count.

4: High sensitivity detection. Overrides default valid measurement detection

algorithm, and uses a threshold value for high sensitivity and noise.

5: Low sensitivity detection. Overrides default valid measurement detection

algorithm, and uses a threshold value for low sensitivity and noise.

\*/

void LIDAR\_configuration(unsigned int mode)

{

unsigned int i;

switch (mode)

{

case 0: // Default mode, balanced performance

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0x80); // Default

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x08); // Default

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0x00); // Default

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0x01); //Default

break;

case 1: // Short range, high speed

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0x1d);

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x08); // Default

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0x00); // Default

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0x01); //Default

break;

case 2: // Default range, higher speed short range

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0x80); // Default

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x00);

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0x00); // Default

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0x01); //Default

break;

case 3: // Maximum range

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0xff);

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x08); // Default

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0x00); // Default

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0x01); //Default

break;

case 4: // High sensitivity detection, high erroneous measurements

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0x80); // Default

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x08); // Default

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0x80);

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0x01); //Default

break;

case 5: // Low sensitivity detection, low erroneous measurements

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0x80); // Default

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x08); // Default

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0xb0);

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0x01); //Default

break;

case 6: //Burst measurements with the nr. of measurements = 0xFE

LIDAR\_write\_data\_to\_register(SIG\_COUNT\_VAL,0xff);

LIDAR\_write\_data\_to\_register(ACQ\_CONFIG\_REG,0x28);

LIDAR\_write\_data\_to\_register(THRESHOLD\_BYPASS,0x00); // Default

LIDAR\_write\_data\_to\_register(OUTER\_LOOP\_COUNT, 0xFE);

LIDAR\_write\_data\_to\_register(MEASURE\_DELAY, 0X14);//100 Hz frequency

break;

}

for (i = 0; i < 50; i++)

LIDAR\_distance\_measurement();

}

//wakes up the LIDAR and performs a series of measurements in order to stabilize the sensor

//write 0x00 to PWR management register

void LIDAR\_Power\_Up(void)

{

unsigned int i;

LIDAR\_write\_data\_to\_register(POW\_CONTROL, 0X00);

*\_delay\_ms*(30);

for (i = 0; i < 50; i++)

LIDAR\_distance\_measurement();

}

//LIDAR enters sleep mode

//write 0x04 to PWR management register

void LIDAR\_Power\_Down(void)

{

LIDAR\_write\_data\_to\_register(POW\_CONTROL, 0x04);

}

## Appendix 7 – Stepper motor

### Header file

#ifndef STEPPER\_H\_

#define STEPPER\_H\_

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#define STEPPER\_RESOLUTION 0.17578125

//D4, D5, D6, D7 are output ports

void Stepper\_Setup\_Ports();

//0 for Clock-Wise, 1 for Counter Clock-Wise

void Stepper\_Step(unsigned char dir, unsigned char \*stepvar, unsigned int stepper\_select);

//makes n steps in the given direction

void Stepper\_n\_Steps(unsigned int n, unsigned char dir, unsigned char \*stepvar, unsigned int stepper\_select);

//receives a value from the PID controller, and steps 'n' steps

void PID\_Steps(int n, unsigned char \*stepvar);

#endif /\* STEPPER\_H\_ \*/

### C file

//set the stepper pins as Outputs

void Stepper\_Setup\_Ports()

{

DDRD |= (1 << DDD7) | (1 << DDD6) | (1 << DDD5) | (1 << DDD4);

DDRB |= (1 << DDB0) | (1 << DDB1) | (1 << DDB2) | (1 << DDB3);

}

//Drive the motor to execute one step in a given direction

void Stepper\_Step(unsigned char dir, unsigned char \*stepvar, unsigned int stepper\_select)

{

unsigned char A, B ,C;

unsigned char aux = 0;

unsigned char i = \*stepvar;

//Set the parameters A, B and C for the current step

A = ((i & 0b00000100) >> 2);

B = ((i & 0b00000010) >> 1);

C = (i & 0b00000001);

//Apply the boolean equations and generate the output logic sequence

aux = (((A & B) | (A & C)) << 3) +

((((A & (B ^ 0x01)) | ((((A ^ 0X01) & B)) & C))) << 2) +

((((A ^ 0x01) & C) | ((A ^ 0x01) & B)) << 1) +

(((A ^ 0x01) & (B ^ 0x01)) | ((A & B) & C)) ;

//drive the selected stepper motor

if (stepper\_select == 1){

aux = (aux << 4);

//stepper 1 is connected to port D

PORTD &= (aux + 0X0F);

PORTD |= aux;

}

if (stepper\_select == 2){

//stepper 2 is connected to port B

PORTB &= (0XF0 + aux);

PORTB |= aux;

}

//drive the stepper in the selected direction

if (dir == 1){

//if the current step reached the upper limit, reset it to 0

if (i == 7){

i = 0;

\*stepvar = i;

}

else{

//increment the current step counter

i++;

\*stepvar = i;

}

}

else{

//if the current step reached the lower limit, reset it to 7

if (i == 0){

i = 7;

\*stepvar = 7;

}

else{

//decrement the current step counter

i--;

\*stepvar = i;

}

}

}

//makes n steps in the given direction

void Stepper\_n\_Steps(unsigned int n, unsigned char dir, unsigned char \*stepvar, unsigned int stepper\_select){

unsigned int i;

for (i = 0; i < (n \* 2); i++)

{

Stepper\_Step(dir, stepvar, stepper\_select);

*\_delay\_ms*(2);

}

}

//receives a value from the PID controller, and steps 'n' steps

void PID\_Steps(int n, unsigned char \*stepvar){

if (n < 0){

n = -n;

Stepper\_n\_Steps(n, 0, stepvar, 2);

}

else{

Stepper\_n\_Steps(n, 1, stepvar, 2);

}

}

## Appendix 8 – MPU 6050

### Header file

#ifndef MPU6050\_H\_

#define MPU6050\_H\_

#include "Two\_Wire.h"

#include <math.h>

//Pi constant value

#define PI 3.14

//Gyroscope and Accelerometer sensitivity

#define GYRO\_SENSITIVITY 131.0

#define ACCEL\_SENSITIVITY 2048.0

//Sensor I2C addresses

#define MPU\_6050\_READ 0b11010001

#define MPU\_6050\_WRITE 0b11010000

//Gyroscope configuration register

#define GYRO\_CONFIG 0x1B

//Accelerometer configuration register

#define ACCEL\_CONFIG 0x1C

//Accelerometer data registers

#define ACCEL\_XOUT\_H 0x3B

#define ACCEL\_XOUT\_L 0x3C

#define ACCEL\_YOUT\_H 0x3D

#define ACCEL\_YOUT\_L 0x3E

#define ACCEL\_ZOUT\_H 0x3F

#define ACCEL\_ZOUT\_L 0x40

//Gyroscope data registers

#define GYRO\_XOUT\_H 0x43

#define GYRO\_XOUT\_L 0x44

#define GYRO\_YOUT\_H 0x45

#define GYRO\_YOUT\_L 0x46

#define GYRO\_ZOUT\_H 0x47

#define GYRO\_ZOUT\_L 0x48

//Power management register

#define PWR\_MNG1 0x6B

//writes the 8-bit data to one of the registers

void MPU\_write\_data\_to\_register(unsigned int addr, unsigned int data);

//returns the 8-bit data from one of the registers

unsigned int MPU\_read\_data\_from\_register(unsigned int addr);

//powers up the module from SLEEP mode by writing the value 0x00 to the power management register 1

void MPU\_Power\_Up(void);

//Sets the device to sleep by writing 0x01 to the power management register 1

void MPU\_Power\_Down(void);

//data acquisition from the Gyroscope or from the Accelerometer

//mode = 0 => Gyroscope

//mode = 1 => Accelerometer

double MPU\_data\_acq(unsigned int addr, unsigned char mode);

//returns the inclination angle on a single axis

int MPU\_single\_axis\_angle(unsigned int addr);

//returns the inclination angle using two axis

int MPU\_double\_axis\_angle(unsigned int addr1, unsigned int addr2);

//applies an averaging filter on n measurements of the MPU two axis angle

int MPU\_filter\_double\_axis(int n);

//returns the inclination angle using all three axis

//axis = 0 for X axis

//axis = 1 for Y axis

//axis = 2 for Z axis

int MPU\_triple\_axis\_angle(unsigned int axis);

#endif /\* MPU6050\_H\_ \*/

### C file

#include "MPU6050.h"

//Starts the I2C communication

void I2C\_start\_condition\_MPU(void)

{

//Sets the bits necessary in the control register

TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if TWSR has the correct code after START

if ((TWSR & 0xF8) != START);

// Serial\_write\_string("\nI2C START ERROR!");

//else

// Serial\_write\_string("\nI2C START OK!");

}

//Sends the sensor address for writing purpose

void I2C\_sensor\_addr\_write\_MPU(unsigned int addr)

{

//put the address you want to send in the data register

TWDR = addr;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for ADDRESS WRITE ACK

if ((TWSR & 0xF8) != SLAW\_ACK)

{

// Serial\_write\_string("\nI2C SLAW\_ACK ERROR! Address = ");

// Serial\_write\_int(addr);

*\_delay\_ms*(500);

I2C\_start\_condition\_MPU();

I2C\_sensor\_addr\_write\_MPU(addr);

}

//else

// Serial\_write\_string("\nI2C SLAW\_ACK OK!");

}

//Sends the sensor address for reading purpose

void I2C\_sensor\_addr\_read\_MPU(unsigned int addr)

{

//put the address you want to send in the data register

TWDR = addr;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for ADDRESS READ ACK

if ((TWSR & 0xF8) != SLAR\_ACK){

//Serial\_write\_string("\nI2C SLAR\_ACK ERROR! Address = ");

// Serial\_write\_int(addr);

// \_delay\_ms(500);

I2C\_start\_condition\_MPU();

I2C\_sensor\_addr\_read\_MPU(addr);

}

//else

// Serial\_write\_string("\nI2C SLAR\_ACK OK");

}

//Sends the register address which needs to be accessed

void I2C\_send\_register\_address\_MPU(unsigned int addr)

{

//put the address you want to send in the data register

TWDR = addr;

//start operation

TWCR = (1 << TWINT) | (1 << TWEN);

//Monitor TWINT to see if the operation is done

while (!(TWCR & (1<<TWINT)));

//test if the correct code has been set in TWSR for MASTER TRANSMITTER MODE ACK

if ((TWSR & 0xF8) != MT\_DATA\_ACK){

// Serial\_write\_string("\nI2C MT\_DATA\_ACK-register ERROR!");

*\_delay\_ms*(500);

//start the I2C communication

I2C\_start\_condition\_MPU();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_write\_MPU(MPU\_6050\_WRITE);

I2C\_send\_register\_address\_MPU(addr);

}

//else

// Serial\_write\_string("\nI2C MT\_DATA\_ACK-register OK");

}

//writes the 8-bit data to one of the registers

void MPU\_write\_data\_to\_register(unsigned int addr, unsigned int data)

{

//start the I2C communication

I2C\_start\_condition\_MPU();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_write\_MPU(MPU\_6050\_WRITE);

//send the selected register address to the slave

I2C\_send\_register\_address\_MPU(addr);

//send data to the selected register

I2C\_data\_write(data);

//stop the communication

I2C\_stop\_condition();

}

//returns the 8-bit data from one of the registers

unsigned int MPU\_read\_data\_from\_register(unsigned int addr)

{

unsigned int data;

//start the I2C communication

I2C\_start\_condition\_MPU();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_write\_MPU(MPU\_6050\_WRITE);

//send the selected register address to the slave

I2C\_send\_register\_address\_MPU(addr);

//stop the communication

I2C\_stop\_condition();

//start the I2C communication

I2C\_start\_condition\_MPU();

//send the sensor address for writing purpose

I2C\_sensor\_addr\_read\_MPU(MPU\_6050\_READ);

//read one byte from the slave

data = I2C\_read\_byte();

//stop the communication

I2C\_stop\_condition();

return (data);

}

//powers up the module from SLEEP mode by writing the value 0x00 to the power management register 1

void MPU\_Power\_Up(void)

{

MPU\_write\_data\_to\_register(PWR\_MNG1, 0x00);

*\_delay\_ms*(30);

}

//Sets the device to sleep by writing 0b01000000 to the power management register 1

void MPU\_Power\_Down(void)

{

MPU\_write\_data\_to\_register(PWR\_MNG1, 0b01000000);

}

//data acquisition from the Gyroscope or from the Accelerometer

//mode = 0 => Gyroscope

//mode = 1 => Accelerometer

double MPU\_data\_acq(unsigned int addr, unsigned char mode)

{

int data;

unsigned int aux;

double output = 0;

//read the HIGH value of the measurement

data = MPU\_read\_data\_from\_register(addr);

//shift the result with 8 bits

data = data << 8;

//add the LOW value of the measurement

data = data + MPU\_read\_data\_from\_register(addr + 1);

//the result is 2's Complement, so we check the sign bit

aux = data & (1 << 15);

//if the number is negative, convert it to decimal by complementing all the bits, adding 1, then multiply with -1

if (aux != 0)

{

data = ~(data) + 1;

data = data \* (-1);

}

//divide the result by the correct sensitivity

if (mode == 0)

output = (double)(data / GYRO\_SENSITIVITY);

if (mode == 1)

output = (double)(data / ACCEL\_SENSITIVITY);

//return the data

return (output);

}

//returns the inclination angle on a single axis

int MPU\_single\_axis\_angle(unsigned int addr)

{

double accel, angle;

//acceleration data acquisition for the required axis

accel = MPU\_data\_acq(addr, 1);

//calculate the inclination angle

angle = *asin*(accel);

//transform the angle from radians to degrees

angle = (angle \* 180.0) / PI;

//return the result as an integer approximation

return ((int)(angle));

}

//returns the inclination angle using two axis

int MPU\_double\_axis\_angle(unsigned int addr1, unsigned int addr2)

{

double accel1, accel2, angle;

//acceleration from the first axis

accel1 = MPU\_data\_acq(addr1, 1);

//acceleration from the second axis

accel2 = MPU\_data\_acq(addr2, 1);

//calculate inclination angle based on 2 axes

angle = *atan*(accel1/ accel2);

//transform the angle from radians to degrees

angle = (angle \* 180.0) / PI;

//if the result is in the second quadrant, the operand is negative

//180 degrees will be added to the result

if (accel1 >= 0 && accel2 <= 0)

angle += 180;

//if the result is in the third quadrant, the operand is negative

//180 degrees will be subtracted from the result

if (accel1 <= 0 && accel2 <= 0)

angle -= 180;

//return the result as an integer approximation

return ((int)(angle));

}

//applies an averaging filter on n measurements of the MPU two axis angle

int MPU\_filter\_double\_axis(int n)

{

unsigned int i;

int sum = 0;

for (i = 0; i < n; i++)

{

MPU\_Power\_Up();

//add all the results from the accelerometer

sum = sum + MPU\_double\_axis\_angle(ACCEL\_XOUT\_H, ACCEL\_YOUT\_H);

*\_delay\_ms*(10);

}

//return the average result

return (sum / n);

}

//returns the inclination angle using all three axis

//axis = 0 for X axis

//axis = 1 for Y axis

//axis = 2 for Z axis

int MPU\_triple\_axis\_angle(unsigned int axis)

{

double accel\_x, accel\_y, accel\_z;

double angle, aux;

//read the acceleration data on all of the 3 axis

accel\_x = MPU\_data\_acq(ACCEL\_XOUT\_H, 1);

accel\_y = MPU\_data\_acq(ACCEL\_YOUT\_H, 1);

accel\_z = MPU\_data\_acq(ACCEL\_ZOUT\_H, 1);

switch (axis)

{

//X axis inclination

case 0:

aux = *sqrt*(accel\_y \* accel\_y + accel\_z \* accel\_z);

aux = accel\_x / aux;

angle = *atan*(aux);

angle = (angle \* 180.0) / PI;

return ((int)(angle));

break;

//Y axis inclination

case 1:

aux = *sqrt*(accel\_x \* accel\_x + accel\_z \* accel\_z);

aux = accel\_y / aux;

angle = *atan*(aux);

angle = (angle \* 180.0) / PI;

return ((int)(angle));

break;

//Z axis inclination

case 2:

aux = *sqrt*(accel\_x \* accel\_x + accel\_y \* accel\_y);

aux = aux / accel\_z;

angle = *atan*(aux);

angle = (angle \* 180.0) / PI;

return ((int)(angle));

break;

}

return (0);

}

## Appendix 9 – Controller

### Header file

#ifndef CONTROLLER\_H\_

#define CONTROLLER\_H\_

#include "Serial\_Communication.h"

#include "MPU6050.h"

#include "Stepper.h"

//the maximum steps allowed at a single iteration of the algorithm

#define MAX\_STEPS 512

//proportionality constant

#define KP 1

//a boundary for the error, as to when it will be added to the Integral Error

#define IMAX\_LIM 5

//the maximum contribution of the integral element of the algorithm

#define IERROR\_LIM (double)(4 / KI)

//integral constant

#define KI 0.15

//derivative constant

#define KD 0.5

//the number of null consecutive measurements required

#define ERROR\_DIM 5

//the number of measurements done by the averaging filter

#define NR\_MEASUREMENTS 5

//returns the current error between the Set Point and the Process Variable

int getError(int sp, int pv);

//returns the necessary steps given by the PID algorithm

int getPIDSteps(int error, int prev\_error, double \*errorI);

//inserts the error at a certain position in the error buffer

void insertError(int error\_buff[], int error, unsigned int \*pos);

//initiates the error buffer with all values equal to 1

void initBuff(int error\_buff[], unsigned int dim);

//displays the buffer

void displayBuff(int error\_buff[], unsigned int dim);

//returns '1' if all the elements of the buffer are equal to 0

//returns '0' otherwise

int checkBuff(int error\_buff[], unsigned int dim);

//sets the mechanical element at the selected pitch angle

void setAngle(int angle, unsigned char \*stepvar);

#endif /\* CONTROLLER\_H\_ \*/

### C file

#include "Controller.h"

//returns the necessary steps given by the PID algorithm

int getPIDSteps(int error, int prev\_error, double \*errorI)

{

double aux\_errorI = \*errorI;

double stepsP, stepsI, stepsD, stepsPID;

//Contribution of the Proportional element

stepsP = (double)(error \* KP);

//the Integral Drive will have a contribution only when the error is not very large

//The integral will control the DC error

//if the current error is equal to 0, the Integral error will be set to 0

if (error < IMAX\_LIM && error > -IMAX\_LIM && error != 0)

{

//add the current error to the Integral error

aux\_errorI += error;

}

else

{

//if the Integral Error exceeds its boundaries, it is set to 0

aux\_errorI = 0;

}

//The integral error is limited by its upper and lower boundaries

if (aux\_errorI > IERROR\_LIM)

{

aux\_errorI = IERROR\_LIM;

}

if (aux\_errorI < -IERROR\_LIM)

{

aux\_errorI = -IERROR\_LIM;

}

//the current Integral Error

\*errorI = aux\_errorI;

//The contribution of the Integral element

stepsI = (double)(KI \* aux\_errorI);

//if the error is 0, then the derivative drive will not have a contribution

if (error == 0)

{

stepsD = 0;

}

else

{

//The contribution of the Derivative element, acting on past errors

stepsD = (double)((error - prev\_error) \* KD);

}

//The final necessary drive is the sum of all the contributions

stepsPID = stepsP + stepsI + stepsD;

//the final number of steps is limited

if ((int)(stepsPID) > MAX\_STEPS)

stepsPID = MAX\_STEPS;

if ((int)(stepsPID) < -MAX\_STEPS)

stepsPID = -MAX\_STEPS;

//return the result in integer format

return ((int)(stepsPID));

}

//returns the current error between the Set Point and the Process Variable

int getError(int sp, int pv)

{

return (sp - pv);

}

//inserts the error at a certain position in the error buffer

void insertError(int error\_buff[], int error, unsigned int \*pos)

{

unsigned int aux = \*pos;

aux++;

//if the value of the position exceeds the upper limit, the position is set to 0

if (aux == ERROR\_DIM)

{

//the new error is inserted on the position 0

error\_buff[0] = error;

aux = 0;

}

else

{

//the new error is inserted in the buffer

error\_buff[aux] = error;

}

\*pos = aux;

}

//initiates the error buffer with all values equal to 1

void initBuff(int error\_buff[], unsigned int dim)

{

unsigned int i;

for (i = 0; i < dim; i++)

{

error\_buff[i] = 1;

}

}

//displays the buffer

void displayBuff(int error\_buff[], unsigned int dim)

{

unsigned int i;

Serial\_write\_string("\nBuff = ");

for (i = 0; i < dim; i++)

{

Serial\_write\_int(error\_buff[i]);

Serial\_write\_char(' ');

}

}

//returns '1' if all the elements of the buffer are equal to 0

//returns '0' otherwise

int checkBuff(int error\_buff[], unsigned int dim)

{

unsigned int i;

for (i = 0; i < dim; i++)

{

if (error\_buff[i] != 0)

return (0);

}

return (1);

}

//sets the mechanical element at the selected pitch angle

void setAngle(int angle, unsigned char \*stepvar)

{

//current error

int error = 0;

//previous error

int prev\_error = 0;

//Integral error

double errorI = 0;

//current angle

int current\_angle = 0;

//the output of the PID algorithm

int stepsPID = 0;

//error buffer

int error\_buff[ERROR\_DIM];

//current position in the error buffer

unsigned int pos = 0;

//initiate the error buffer

initBuff(error\_buff, ERROR\_DIM);

do

{

//the current pitch angle of the system

current\_angle = MPU\_filter\_double\_axis(NR\_MEASUREMENTS);

//set the previous error

prev\_error = error;

//set the current error

error = getError(angle, current\_angle);

//insert the current error in the error buffer

insertError(error\_buff, error, &pos);

//the necessary steps, received from the PID algorithm

stepsPID = getPIDSteps(error, prev\_error, &errorI);

//drives the 2nd stepper motor

PID\_Steps((int)(stepsPID), stepvar);

//the process repeats until the error buffer is filled only with values equal to 0

} while (checkBuff(error\_buff, ERROR\_DIM) != 1);

}

## Appendix 10 – Scanner

### Header file

#ifndef SCAN\_H\_

#define SCAN\_H\_

#include "Serial\_Communication.h"

#include "Stepper.h"

#include "Servo\_Setup.h"

#include "LIDAR\_LITE\_V3.h"

#include "Controller.h"

#define START\_SCAN 127

#define SET\_PARAMS 126

#define CONTINUE\_SEND 125

//Sets the scanning parameters

void set\_params(void);

//the scanner transmits a measurement serially and executes a number of steps around the z-axis

void ScanStep(unsigned char dir, unsigned char \*stepvar, int \*yawnCount);

//The scanner makes a swipe scan on the X axis

void ScanYawn(unsigned char \*dir, unsigned char \*stepvar1, unsigned char \*stepvar2);

//the system executes a full yawing scan according to the parameters

void Scan(unsigned char \*dir, unsigned char \*stepvar1, unsigned char \*stepvar2);

//Display the scan parameters on the serial monitor

void DisplayParameters(void);

#endif /\* SCAN\_H\_ \*/

### C file

#include "Scan.h"

//Default values of the scan parameters

int yawn\_resolution = 8;

int max\_yawn\_steps = 256;

int pitch\_resolution = 8;

int max\_pitch\_angle = -90;

int start\_angle = -10;

int lidar\_config = 0;

//init the parameters array

int params[6] = {0, 0, 0, 0, 0, 0};

//Sets the scanning parameters

void set\_params(void)

{

//counter for the parameters buffer

int i;

//flag which signals whether a number is negative

char neg = 0;

//variable where the received bytes are stored

unsigned char receive = 0;

//set each value of the parameters buffer to 0

for (i = 0; i < 6; i++){

params[i] = 0;

}

i = 0;

receive = 0;

//signal to the Java application that a new byte can be sent

Serial\_write\_char(CONTINUE\_SEND);

//The new byte which has been received serially

receive = Serial\_read\_char();

while(receive != SET\_PARAMS)

{

//if the character ' ' has been received, then a parameter has been successfully received

if (receive == ' ')

{

//if the number has been signaled to be negative then multiply it by -1

if (neg == 1)

{

params[i] = params[i] \* (-1);

}

//set the negative flag to 0

neg = 0;

//increment the counter

i++;

}

else

{

//if '-' is received, then raise the negative flag

if (receive == '-'){

neg = 1;

}

else

{

//if the received byte is a numerical digit between 0 and 9, transform it from ASCII to decimal

//the most significant digits are transmitted first, so the received value needs to be multiplied by 10

//finally the new digit is added to the result

if (receive >= 48 && receive <= 57) {

params[i] = params[i] \* 10 + (receive - 48);

}

}

}

//the application signals that it is ready to receive a new byte

Serial\_write\_char(CONTINUE\_SEND);

receive = Serial\_read\_char();

}

//finally set the value of each of the scan parameters

yawn\_resolution = params[0];

max\_yawn\_steps = params[1];

pitch\_resolution = params[2];

max\_pitch\_angle = params[3];

start\_angle = params[4];

lidar\_config = params[5];

//configure the LIDAR to the selected configuration routine

LIDAR\_configuration(lidar\_config);

}

//the scanner transmits a measurement serially and executes a number of steps around the z-axis

void ScanStep(unsigned char dir, unsigned char \*stepvar, int \*yawnCount){

unsigned int i = \*yawnCount;

//take a measurement with the LIDAR and transmit it serially

Serial\_write\_int(LIDAR\_distance\_measurement());

//transmit a delimitation between each measurement

Serial\_write\_char(' ');

//The first stepper motor will make its steps equal to the yaw resolution around the z-axis

Stepper\_n\_Steps(yawn\_resolution, dir, stepvar, 1);

//increment the stepper counter

i += yawn\_resolution;

\*yawnCount = i;

}

//The scanner makes a swipe scan on the X axis

void ScanYawn(unsigned char \*dir, unsigned char \*stepvar1, unsigned char \*stepvar2)

{

//initialize the stepper counter

int count = 0;

unsigned char dir\_aux = \*dir;

//the stepper makes a number of steps, taking one measurement at every step

while (count < max\_yawn\_steps)

{

ScanStep(dir\_aux, stepvar1, &count);

}

//The end of the yaw angle interval has been reached

//One more distance measurement is performed and transmitted

Serial\_write\_int(LIDAR\_distance\_measurement());

Serial\_write\_char(' ');

//Send new line and carriage return

Serial\_write\_char(13);

Serial\_write\_char('\n');

//change the direction of the scanner around the z-axis

dir\_aux ^= 0x01;

\*dir = dir\_aux;

}

//the system executes a full yawing scan according to the parameters

void Scan(unsigned char \*dir, unsigned char \*stepvar1, unsigned char \*stepvar2){

//initialize current angle variable

double currentAngle = start\_angle;

//initialize the final angle variable

double finalAngle = max\_pitch\_angle;

double increment;

//the direction along the z-axis

int pitch\_dir = 0;

unsigned char dir\_aux = \*dir;

setAngle(start\_angle, stepvar2);

//auxiliary for the initial rotation of the scanner

dir\_aux ^= 0x01;

//Before starting a scan, the scanner moves half of the maximum steps

//in the direction opposite from the first scan

Stepper\_n\_Steps(max\_yawn\_steps / 2, dir\_aux, stepvar1, 1);

*\_delay\_ms*(100);

//initialize the increment for monitoring the current pitch angle

increment = (double)(STEPPER\_RESOLUTION \* pitch\_resolution);

//the system has been designed for starting pitch angles which are higher than the final angles

if (start\_angle > max\_pitch\_angle){

//if the start pitch angle is bigger than the final angle, the direction is set clockwise

pitch\_dir = 0;

}

else{

//if the start pitch angle is less than the final angle, the direction is set counter clockwise

pitch\_dir = 1;

//the sign for of the current angle and the final angle is changed

currentAngle = -currentAngle;

finalAngle = -finalAngle;

}

//the loop continues as long as the final pitch angle has not been reached

while (currentAngle >= finalAngle)

{

//a scan around the z-axis is performed

ScanYawn(dir, stepvar1, stepvar2);

//set the new value of the current angle

currentAngle -= increment;

*\_delay\_ms*(200);

//The second stepper makes a number of steps equal to the pitch resolution

Stepper\_n\_Steps(pitch\_resolution, pitch\_dir , stepvar2, 2);

*\_delay\_ms*(200);

}

dir\_aux = \*dir;

//in order to get back in the initial position:

//half of the maximum yaw steps are executed in the direction opposite from the last scan

Stepper\_n\_Steps(max\_yawn\_steps / 2, dir\_aux, stepvar1, 1);

dir\_aux ^= 0x01;

\*dir = dir\_aux;

Serial\_write\_char(127);

//when the XY scan is done, the stepper stops and is set back in its initial position

//setAngle(0, stepvar2);

*\_delay\_ms*(1000);

}

//Display the scan parameters on the serial monitor

void DisplayParameters(void){

Serial\_write\_string("\nParameters: ");

Serial\_write\_string("\nYawn Resolution = ");

Serial\_write\_int(yawn\_resolution);

Serial\_write\_string("\nYawn Max Steps = ");

Serial\_write\_int(max\_yawn\_steps);

Serial\_write\_string("\nPitch Resolution = ");

Serial\_write\_int(pitch\_resolution);

Serial\_write\_string("\nPitch Max Angle = ");

Serial\_write\_int(max\_pitch\_angle);

Serial\_write\_string("\nStart Angle = ");

Serial\_write\_int(start\_angle);

Serial\_write\_string("\nLidar Config = ");

Serial\_write\_int(lidar\_config);

}

## Appendix 11 – Main program

#include "Scan.h"

#include <stdlib.h>

void Setup(void);

void NewStep(unsigned char \*stepvar);

int main(void)

{

unsigned char stepvar1 = 0;

unsigned char stepvar2 = 0;

unsigned char dir = 1;

unsigned char receive = 0;

//int angle = 0;

//int var = 0;

Setup();

*\_delay\_ms*(1000);

while (1)

{

//Check if one of the operations has been signalled to start

while (receive != START\_SCAN && receive != SET\_PARAMS)

{

receive = Serial\_read\_char();

}

//Enter the Set\_Parameters routine

if (receive == SET\_PARAMS){

set\_params();

}

//Enter the Scan Routine

if (receive == START\_SCAN){

Scan(&dir, &stepvar1, &stepvar2);

}

receive = 0;

}

}

void Setup(void)

{

I2c\_init();

LIDAR\_configuration(0);

Stepper\_Setup\_Ports();

Serial\_setup();

}

# Author’s CV