UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTOS ACADÊMICOS DE ELETRÔNICA E MECÂNICA
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

DOUGLAS DIOGO FRANCZAK

MYMAPPER: plataforma de mapeamento de ambientes 3D

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2016

DOUGLAS DIOGO FRANCZAK

MYMAPPER: plataforma de mapeamento de ambientes 3D

Trabalho de Conclusão de Curso de Graduação, apresentado ao Curso Superior de Tecnologia em Mecatrônica Industrial, dos Departamentos Acadêmicos de Eletrônica e Mecânica, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Walter Luis Mikos, Dr. Eng. Mec.

CURITIBA
2015

TERMO DE APROVAÇÃO

DOUGLAS DIOGO FRANCZAK

MYMAPPER: plataforma de mapeamento de ambientes em 3D

Este trabalho de conclusão de curso foi apresentado no dia 14 de dezembro de 2015, como requisito parcial para obtenção do título de Tecnólogo em Mecatrônica Industrial, outorgado pela Universidade Tecnológica Federal do Paraná. O aluno foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Milton Luiz Polli
Coordenador de Curso
Departamento Acadêmico de Mecânica

_____
Prof. Esp.Sérgio Moribe
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

_____
Prof. Sidney Carlos Gasoto
UTFPR

_____
Prof. Luiz Carlos de Abreu Rodrigues
UTFPR

_____
Prof. Walter Luis Mikos, Dr. Eng. Mec
Orientador - UTFPR

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso

RESUMO

FRANCZAK, Douglas. **Mymapper: plataforma de mapeamento de ambientes 3D**. 2015. 86 páginas. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Mecatrônica Industrial), Departamentos Acadêmicos de Eletrônica e Mecânica, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

Na robótica a área aonde se estuda as técnicas de mapeamento e localização em mapas é chamada de SLAM, essa área visa desenvolver métodos de mapeamento e localização cada vez mais precisos, pois um dos problemas mais comuns em SLAM é a deformação ou imperfeição de mapas quando se deseja localizar em tais mapas. Esse projeto busca criar um sistema SLAM concentrando na integração de dados dos sensores de orientação visando realizar uma varredura do ambiente e formar o mapa. Outro foco do projeto está no método em que os dados são integrados e na escolha dos componentes, pois ambos os métodos e componentes precisam ter valor didático. Esse trabalho irá demonstrar, a construção mecânica da plataforma; a interpretação dos dados vindos dos sensores: acelerômetro, giroscópio e telêmetro infravermelho; a integração dos dados do acelerômetro e giroscópio a partir do filtro complementar e a reconstrução desses dados em um ambiente 3D. O *LabVIEW* é um software para programação visual em blocos que foi escolhido por ser muito utilizado na indústria moderna. O *hardware myRIO* foi utilizado como coletor de dados pois é programado no *LabVIEW* e possui uma variedade de funcionalidades muito úteis para esse projeto, além de ser feito especificamente para ser didático. A *Unity Engine* é um *software* de criação de jogos, onde há uma maior facilidade de recriar ambientes em 3D por causa de sua popularidade, o que significa que há um suporte maior, e por causa da possibilidade de utilizar programação visual de certos aspectos do sistema em conjunto com a programação C# tradicional. O MyMapper foi construído de forma a possuir uma interface simples, fornecendo ao usuário todas as opções de calibração necessárias e mantendo o código limpo e de forma modular, para facilitar melhorias futuras. No fim, o MyMapper consegue mapear um ambiente de no máximo um metro, o limite em que o sensor telêmetro consegue funcionar, mas os detalhes desse ambiente são deformados por vários motivos, dependendo da calibração do sistema. Mesmo assim, o sistema pode ser utilizado para detectar obstáculos, pois as deformações são pequenas e modificam apenas o formato da peça e não a sua posição geral.

Palavras chave: sensor. Integração. Ambiente. SLAM. Plataforma.

ABSTRACT

FRANCZAK, Douglas. **MyMapper: platform for 3D environment mapping**. 2015. 86 pages. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Mecatrônica Industrial), Academic Departments of Eletrônica and Mecânica, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

In robotics the area where it's studied the techniques of mapping and localization is called SLAM, this area aims to develop more precise methods of mapping and localization, because one of the more common problems in SLAM is the deformation or imperfection of the maps when it's necessary locate itself on those maps. The objective of this project is to create a SLAM system focusing on the integration of data coming from orientation sensors aiming to realize a sweep of the environment to create a map. Another focus of this project is in the method that the data is integrate and in the choice of the components, because both the methods and the mechanical construction needs to have some didactic value. This document will demonstrate, the mechanical construction of the platform; the interpretation of the data coming from the sensors: accelerometer, gyroscope and infrared rangefinder; the integration of the data from accelerometer and gyroscope starting from the complementary filter and the reconstruction of this data on a 3D environment. The *LabVIEW* is a software for visual programming that has been chosen because it's used a lot on modern industry. The *myRio* is used as a data collector because it is programmed on *LabVIEW* and it has a variety of functionalities that are very useful for this project, and also it's made specifically to be didactic. The *Unity Engine* is a software for game creation, which is easier to recreate 3D environments because of its popularity, which means that there's a bigger support, and because of the possibility of utilizing visual programming on certain aspects of the system in conjunct with the traditional C# programming. The MyMapper has been constructed in way that it has a simple interface, providing to the user all the necessary options for calibration and maintaining a clean and modular code, to facilitate future improvements. In the end, the MyMapper can map the environment of a meter, maximum, the limit in which the telemeter can work, but the details of this environment are deformed for various motives, depending on the calibration of the system. Even so, the system can be used to detect obstacles, because the deformations are small and modify only the format of the piece and not its general position.
Keywords: sensor. Integration. Environment. SLAM. Platform.

LISTA DE ILUSTRAÇÔES

SUMÁRIO

1 INTRODUÇÃO

1.1 TEMA

Com o aumento da utilização da robótica na indústria, novas áreas de pesquisa tecnológica aplicada são abertas, necessitando de diferentes e, cada vez mais complexas formas de controle. Para isso, também é necessário a evolução dos métodos de coleta de dados utilizados pelos robôs, um desses métodos é o de mapeamento e localização simultâneo, também conhecido como SLAM (*Simultaneous Localization And Mapping*) (THRUN, 2002).

O SLAM visa desenvolver métodos de mapeamento e localização cada vez mais precisos, pois um dos problemas mais comuns em SLAM é a deformação ou imperfeição de mapas quando se deseja localizar em tais mapas. Outros problemas surgem como a necessidade de se detectar objetos móveis, obstáculos ou terreno perigoso.

Para resolver esses problemas várias soluções são utilizadas, como sensores de inércia para detectar orientação e movimento e sensores de obstáculos para se fazer a varredura do ambiente, também mais recentemente são utilizadas câmeras *RGB-D* para se criar mapas de nuvem de pontos coloridos muito mais próximos da realidade (KERL; STURM; CREMERS, 2013).

A escolha da solução também varia de acordo com os limites dos projetos, pois os sensores podem chegar a preços muito altos, dependendo da tecnologia utilizada. Alguns sensores também requerem maiores níveis de processamento e mais cálculos matemáticos, como é o caso das câmeras *stereo* onde se manipulam duas imagens para se criar o ambiente 3D.

1.2 DELIMITAÇÃO DO ESTUDO

O projeto busca criar um sistema SLAM concentrando, primeiramente, na integração de dados dos sensores de orientação visando realizar uma varredura do ambiente e formar o mapa. O foco do projeto está no método em que os dados são integrados e na escolha dos componentes pois ambos os métodos e componentes precisam ter valor didático. Esse valor é estimado pensando na popularidade, na complexidade e custo de utilização, portanto o sistema tem em mente o que pode ser aplicado no meio industrial e no meio acadêmico.

1.3 PROBLEMA

Como integrar sensores de inércia e um sensor telêmetro para se realizar a varredura para mapeamento do ambiente, mantendo o sistema modular, didático e com aplicabilidade em sistemas de SLAM?

1.4 OBJETIVOS

1.4.1 GERAL

Construir uma plataforma de mapeamento de ambiente 3D integrando sensores inerciais e telêmetro para aplicação didática em SLAM.

1.4.2 OBJETIVOS ESPECÍFICOS

- Integrar os sensores de inércia;
- Integrar o cálculo da orientação da plataforma com a distância do obstáculo;
- Desenvolver um método de varredura do ambiente;
- Mapear o ambiente e visualizá-lo em 3D;
- Criar um meio de armazenamento das medições;
- Fornecer as opções necessárias para que o usuário tenha todo o controle sobre o sistema;
- Manter o sistema modular;
- Manter o código limpo para facilitar o entendimento;
- Testar as capacidades e limitações.

1.5 JUSTIFICATIVA

O estudo dos sistemas de visão aplicados a robôs representa um grande desafio, tanto do ponto de vista acadêmico quanto do industrial, pois tal sistema colhe grande parte dos dados necessários para que o robô possa tomar decisões, com isso, o funcionamento do sistema de visão influencia muito a utilização de um robô, delimitando suas capacidades. Portanto estudar os sistemas de visão robótica ou criar sistemas com boas funcionalidades se torna importante para criar progresso no ramo da robótica. Também ter em mãos um sistema simples que possa ser

utilizado como base introdutória ao estudo de visão robótica aumenta-se em muito o valor do estudo.

A criação de um sistema modular de visão robótica proporciona uma grande variedade de projetos em que tal projeto pode ser utilizado. O sistema foi criado de forma a possuir a qualidade de ser modular e assim, há um grande número de outras funções que podem ser substituídas de acordo com a necessidade. Como o método de montagem dos componentes foi criado pensando na substituibilidade, foi eliminado qualquer necessidade de soldagem ou montagem permanente, também foram implementadas uma grande variedade de opções para calibração, assim o sistema pode ser adaptado facilmente para outras finalidades ou funcionalidades.

1.6 PROCEDIMENTOS METODOLÓGICOS

Para facilitar a observação da estrutura do projeto foi criado um diagrama da Estrutura Analítica de Projeto, que divide o projeto em uma árvore hierárquica partindo do módulo mais geral para o mais específico. Essa estrutura pode ser observada na figura 1.

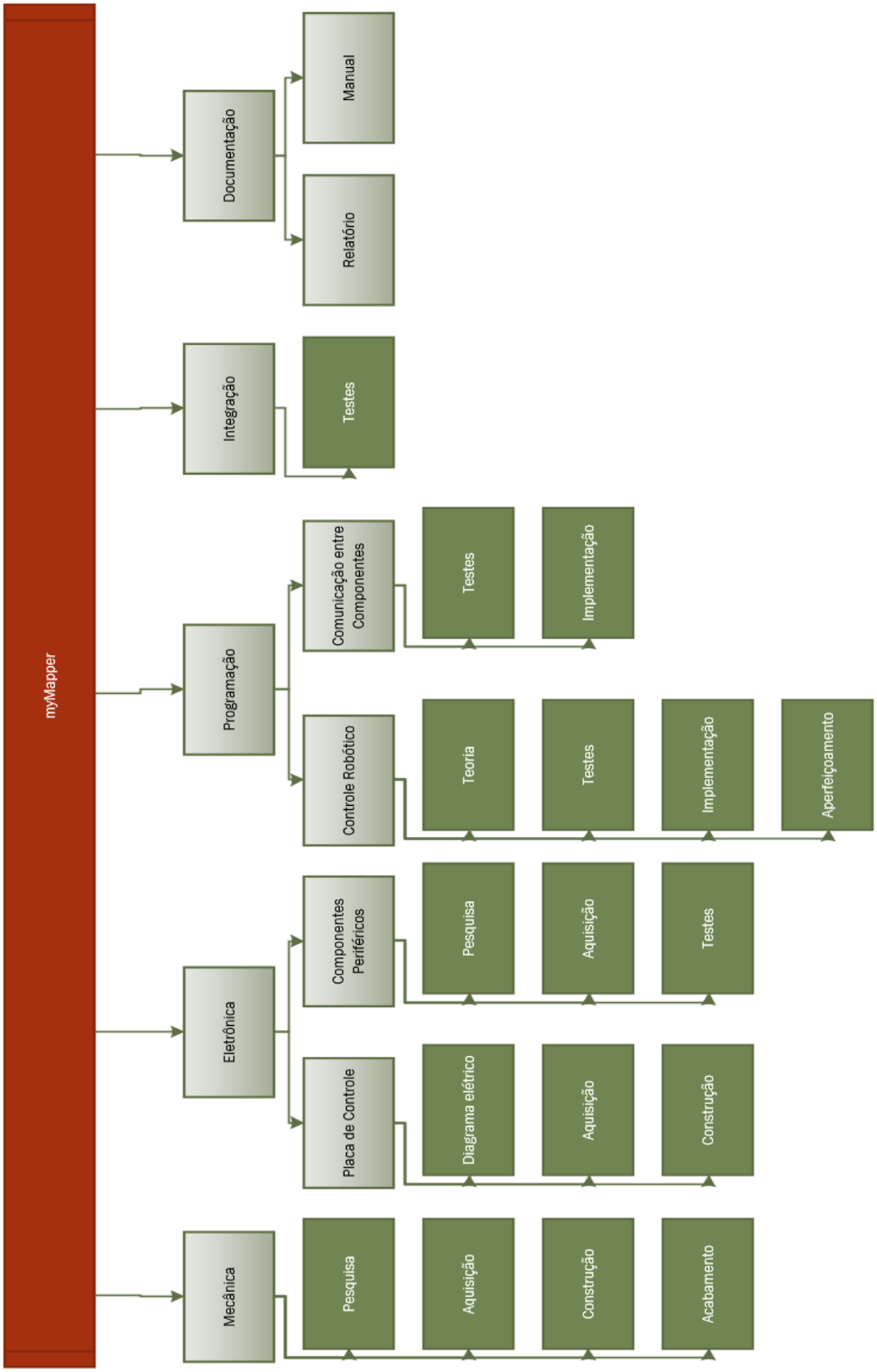


Figura 1 - Estrutura Analítica do Projeto

Fonte: Autoria Própria

Nessa estrutura pode-se observar como o projeto foi dividido em cinco pacotes:

- A mecânica envolve a fabricação e montagem das placas metálicas de suporte e os sensores;
- A eletrônica envolve a criação da placa de controle dos componentes e o método em que os componentes são conectados ao coletor de dados;
- A programação inclui a representação dos dados de medição, o método de armazenamento dos dados de medição, a comunicação entre os componentes e o controle dos componentes. Esse pacote é o mais importante, pois o estudo está voltado ao método de interpretação e integração dos dados vindos dos sensores, o que é resolvido via programação;
- A integração envolve o método de funcionamento da plataforma, juntando todas as funcionalidades implementadas em uma interface simples e otimizada;
- A documentação convém à criação desse relatório e de um manual em inglês onde é explicado em mais detalhes as funcionalidades implementadas e como calibrar o sistema.

A modelagem das funções pode ser observada mediante o digrama IDEF0 na figura 2. Nessa figura pode-se observar as relações de entrada/saída, controle e mecanismos de cada função do sistema, nesse caso o diagrama demonstra o relacionamento entre os componentes mais importantes.

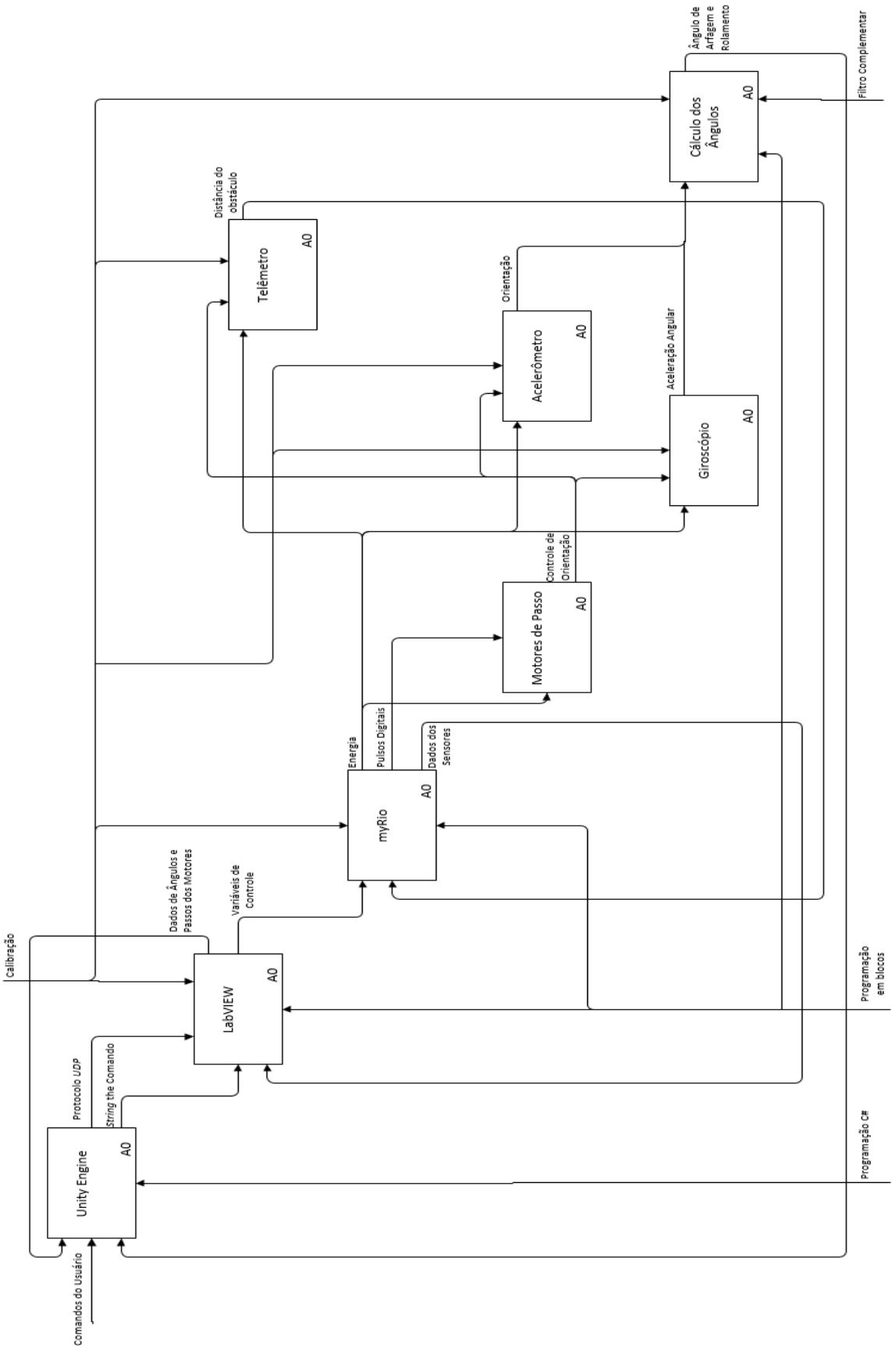


Figura 2 - IDEF0 do Projeto

Fonte: Autoria Própria

Nesse diagrama, cada bloco representa um componente, as setas que entram no bloco a esquerda, acima e abaixo e saem do bloco a direita representam entrada de dados, método de controle, mecanismo de controle e saída de dados, respectivamente.

Pode-se considerar a *Unity* Engine como o meio de controle e interface de usuário principal, o *LabVIEW* se torna o software principal de cálculo de dados, mas o usuário só interage como *LabVIEW* para calibração. O hardware myRIO se torna o hardware principal de coleta de dados e através do *LabVIEW*, são controlados os componentes restantes do sistema.

1.7 EMBASAMENTO TEÓRICO

Durante a pesquisa de SLAM dois projetos se destacaram, um por Ulrich Weiss e outro por Gao et. Al., tais projetos demonstram as utilidades, métodos de funcionamento e dificuldades de implementação de sistemas SLAM que mais influenciaram as características desse projeto.

Weiss e Biber (2011) apresentam a utilidade de mapeamento de ambientes, criando um sistema que detecta plantas, esse tipo de visão robótica é necessário na agricultura, pois ainda há muita necessidade de trabalho humano no cultivo de plantas e cada tipo de planta possui um tratamento diferenciado e delicado.

A automação na área da agricultura precisa de evolução, as máquinas comumente utilizadas são de grande porte e não possuem a capacidade de trabalho especial e delicado que as plantas menores requerem. Com um sistema de visão avançado, se tem mais controle sobre o robô e maior grau de coleta de dados, facilitando o monitoramento e estudo das plantações.

O sensor utilizado é um sensor 3D, esse sensor funciona como uma câmera, medindo centenas de pontos e construindo uma imagem com esses pontos. O sensor 3D emite um raio pulsante infravermelho, esse raio é refletido em um pequeno espelho que é girado em duas direções, medindo o "tempo de vôo" que o raio leva para atingir o alvo e retornar ao sensor, construindo assim a imagem de distância. No projeto de Weiss e Biber os pontos medidos pelo sensor são colocados em um ambiente 3D, utilizando um algoritmo para diferenciar os pontos pertencentes a uma planta dos pontos pertencentes ao chão.

O presente projeto deverá empregar o mesmo funcionamento que o sensor de Weiss, a diferença desse projeto é que o movimento da emissão do raio

infravermelho será feito girando o telêmetro nos 2 graus de liberdade utilizando dois motores de passo. O foco está no posicionamento do robô ao contrário do projeto de Weiss, que foca na criação de um mapa estático que possa ser reutilizado.

Por sua vez, Gao et. al. (2008) apresentam um sistema de visão destinado a acoplar uma cadeira de rodas automaticamente em um dispositivo na traseira de um veículo. Esse sistema necessita de poucas modificações em um veículo comum e permite que o cadeirante entre e saia do veículo com pouca dificuldade, automatizando todo o processo. O sistema possui um sensor LiDAR (*Light Detection and Ranging*) que é utilizado para localizar a cadeira de rodas e, assim, ele consegue guiar a cadeira para a plataforma de acoplamento na traseira do veículo.

O sensor funciona medindo a distância e refletividade de um objeto, nesse caso, dois cilindros retro reflexivos presos em cada lado da parte frontal da cadeira de rodas. Esse projeto é um exemplo de como as técnicas para detecção de objetos se tornam complexas e equipamentos muito específicos são necessários, pois é preciso um cilindro de material específico, preso em posições específicas para que o sensor possa detectá-los e os dados precisam ser interpretados e convertidos mediante de um algoritmo para que se possa determinar a posição da cadeira de rodas no espaço e finalmente o controlador possa mover a cadeira na posição de acoplamento.

1.8 ESTRUTURA DO TRABALHO

O trabalho terá a estrutura abaixo apresentada.

Capítulo 1 - Introdução: serão apresentados o tema, as delimitações da pesquisa, o problema, os objetivos da pesquisa, a justificativa, os procedimentos metodológicos, as indicações para o embasamento teórico, e a estrutura geral do trabalho.

Capítulo 2 - Fundamentação Teórica: serão apresentados a base teórica e as referências necessárias para o entendimento do trabalho.

Capítulo 3 - Desenvolvimento da Plataforma: será apresentado como a plataforma foi construída e como ela funciona.

Capítulo 4 - Apresentação e Análise de Resultados: será apresentado o resultado final mostrado o funcionamento do sistema completamente integrado.

Capítulo 5 - Considerações Finais: será feita uma síntese final da análise das funcionalidades do sistema.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo será demonstrado a fundamentação teórica necessária para entender o desenvolvimento do projeto. A aplicação da teoria será vista no capítulo 3.

2.1 SLAM

Um robô autônomo necessita da capacidade de poder navegar através de um meio sem supervisão humana, para tal, o robô obtém o mapa do ambiente e se localiza nesse mapa, esse sistema é chamado de SLAM (*Simultaneous Localization and Mapping*). Para se realizar um sistema SLAM robusto, surgem muitos problemas que aumentam as incertezas do sistema, aumentando os riscos e diminuindo as capacidades.

Para realizar o mapeamento do ambiente são utilizados sensores como telêmetros de ultrassom e de infravermelho, múltiplas câmeras, sensores de toque, hodômetros, radares, bússolas, GPS e entre outros. O problema é que todos esses sensores possuem erro, o que cria deformação em mapas e aumenta o risco de colisões. Isso ocorre pois as medições são estatisticamente dependentes, ou seja, as medições futuras são dependentes das passadas, então se há erro nas medições passadas esse erro irá se acumular quanto mais medições são feitas.

A dimensão do SLAM também se torna um problema, o número de medições necessárias para se mapear um ambiente simples, como criar uma planta em duas dimensões da casa, pode chegar na casa de milhares, o mapeamento de um ambiente complexo como uma cidade em três dimensões aumenta exponencialmente, podendo chegar a bilhões.

Outro problema é a realização de medições em ambientes cíclicos, pois como há um erro acumulativo, quando o robô retorna ao ponto de origem o mapa pode estar tão deformado que a posição final é diferente da inicial. Isso vem da dificuldade de interpreta-se duas medições de ângulos diferentes correspondem ao mesmo ponto ou não.

Na criação de mapas também é difícil conseguir detectar ambientes dinâmicos, uma medição pode detectar um obstáculo como uma porta ou uma pessoa que está agindo como um obstáculo no momento mas pode não estar mais no mesmo local ao se retornar a esse ambiente. O reconhecimento de objetos

móveis ou estáticos é muito difícil por causa do número inimaginável de formatos, cores e poses tais objetos podem tomar.

E por último vem a dificuldade de criar uma estratégia de mapeamento que controle o robô a fim de explorar o ambiente e continuar construindo o mapa. Essa dificuldade existe pois há uma grande variedade de ambientes e situações disponíveis, o que torna necessário criar planos de contingência que cubra todas as situações possíveis que o robô possa encontrar, ou então haverá sempre um risco de acidentes.

2.2 ACELERÔMETRO

O acelerômetro é um sensor de inércia utilizado para detectar a aceleração, ele utiliza um pequeno material suspenso dentro de uma caixa, também chamado de "peso", para realizar as medições. Como o peso está solto dentro da caixa ele sofre inércia, a resistência à aceleração, quando a caixa se move, essa inércia cria uma pressão na direção oposta ao movimento que é convertida em eletricidade de várias maneiras.

Os acelerômetros mais comuns utilizam capacitores móveis ou cristais piezoelétricos que criam uma corrente elétrica que passa pelo peso e que modificam tais cristais de acordo com a pressão ou distância entre os capacitores, essa corrente elétrica é medida e convertida em aceleração. O peso também sofre influência da gravidade, criando pressão, que pode ser interpretada como a direção e intensidade da aceleração gravidade, por isso o acelerômetro é bastante utilizado para se determinar orientação enquanto está parado ou sobre velocidade constante (WALTER, 2007).

Matematicamente o acelerômetro pode ser representado por eixos em um espaço tridimensional. Todas as forças de aceleração sendo exercidas no acelerômetro são então somadas criando um vetor final no espaço tridimensional com direção, sentido e intensidade. Cada eixo então faz a projeção do eixo sobre o vetor final criando um triângulo retângulo possibilitando o cálculo do ângulo entre o vetor e cada eixo utilizando os princípios de trigonometria.

2.3 GIROSCÓPIO

O giroscópio é um sensor de inércia que detecta a rotação em torno de seus eixos, existe uma variedade de métodos de funcionamento para giroscópios, mas

um dos métodos mais populares é o de utilizar dois acelerômetros lineares em pontos opostos de cada eixo (YAZDI et al, 1988). Desse modo, durante a rotação os acelerômetros se movem em direções opostas e durante a translação eles se movem na mesma direção.

Matematicamente pode-se visualizar o eixo a ser medido como um vetor no espaço tridimensional, os acelerômetros são vetores paralelos com direções opostas posicionados em lados opostos de um plano perpendicular ao eixo. Quando há uma rotação em torno do eixo os acelerômetros se movimentam em direções opostas, mas possuem um valor com o mesmo sinal, os valores então são somados e o resultado convertido na unidade de grau por tempo. Quando há translação ambos os acelerômetros se movimentam na mesma direção, mas como eles possuem sentidos opostos os valores dos vetores serão iguais com sinais opostos, então o resultado da soma dos vetores se torna 0. Desse modo é possível detectar a direção e intensidade da rotação e anular qualquer translação que ocorra no giroscópio, mesmo que ocorra translação e rotação ao mesmo tempo.

Para se comunicar com o giroscópio é utilizado o protocolo I²C (STMicroeletronics, 2013), onde é necessário mandar no mínimo um *byte* para o sensor a partir de uma sequência de números binários, esse byte então é convertido em um número que acessa os registros do sensor. Alguns registros são somente de leitura, portanto um segundo *byte* não é necessário, mas para acessar e modificar os registros do sensor um segundo *byte* é enviado com os valores de configuração do sensor.

2.4 FILTRO COMPLEMENTAR

O acelerômetro e o giroscópio possuem sinais de resposta que podem ser interpretados da mesma maneira, mas é necessário fundi-los matematicamente de uma forma a utilizar as suas qualidades e diminuir os erros. Dois métodos foram encontrados para fazer tal fusão, o filtro de Kalman (KALMAN, 1959) e o filtro complementar (HIGGINS,1975).

O filtro de Kalman necessita de no mínimo duas distintas equações, uma para atualizar as predições e outra para atualizar o erro, é também necessário adaptar a equação para cada situação e manter um ciclo, atualizando ambas as equações em ordem a cada iteração.

Por outro lado o filtro complementar tem um funcionamento muito mais simples e fácil de se implementar, funcionando como um filtro passa-baixa e um filtro passa alta que são somados para criar um sinal final. Como não há a necessidade de um estudo da precisão dos filtros o único fator avaliado foi a facilidade de implementação, o que favoreceu a escolha do filtro complementar.

O filtro complementar mais básico é representado pelo seguinte diagrama de blocos:

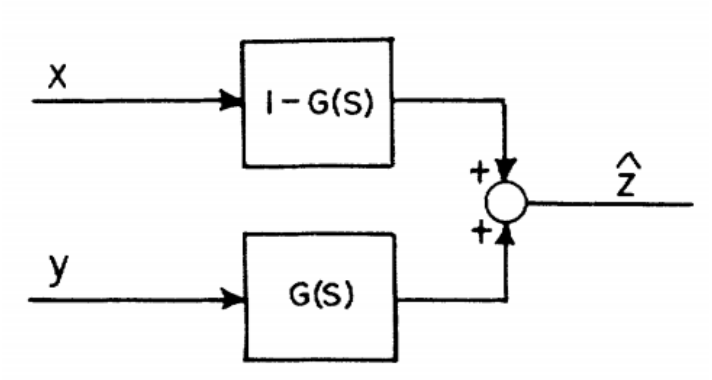


Figura 3 - Filtro Complementar Básico
Fonte: Higgins, W.T., 1975, p.322

Sendo *x* e *y* os valores vindos dos sensores e *G(S)* uma função qualquer. É comum adaptar o filtro para utilizá-lo em várias situações.

3 DESENVOLVIMENTO DA PLATAFORMA

3.1 ESCOLHA DOS COMPONENTES

O myRIO da National Instruments é um hardware embarcado, esse hardware foi escolhido pois ele foi feito, especificamente, para ser didático, possuindo uma grande variedade de funções úteis para ensino e que também são utilizadas na indústria. Sendo necessário a utilização do *LabVIEW* para programar o myRIO, que também é um software de valor didático e muito utilizado na indústria, o myRIO se torna perfeito para esse projeto, pois não é necessário a robustez de um hardware de coleta de dados de nível industrial, sendo muito mais valioso a maior variedade de funções do myRIO.

O sensor telêmetro de infravermelho da Sharp, o GP2Y0A21YK0F, foi escolhido por sua simplicidade de funcionamento. Em comparação com sensores mais avançados e de nível industrial, o sensor telêmetro Sharp é muito mais simples

de se conectar ao myRIO, também possuindo um tamanho reduzido, podendo ser facilmente parafusado a uma placa de suporte. O sensor necessita de um processo matemático mais complicado para se converter os dados mas não necessita de protocolos de comunicação. Por fim, esse sensor também tem a vantagem de possuir um raio infravermelho que resulta em um baixo nível de propagação, podendo manter o foco em um pequeno ponto mesmo a largas distâncias, diferente de um sensor telêmetro de ultrassom, que detecta obstáculos em uma área e não em um ponto.

O sensor acelerômetro da Freescale, o MMA7361L, foi escolhido como principal método de detecção de orientação pois ele consegue detectar a orientação atual em relação com o vetor de aceleração da gravidade da Terra enquanto está sobre repouso ou com velocidade constante. Não há a necessidade de calibração avançada, mas é necessário a utilização de um filtro para reduzir o ruído causado por outras forças de aceleração exercidas no sensor. Esse sensor também não necessita de implementar métodos de comunicação, usando uma saída analógica para cada eixo medido.

O sensor giroscópio da STMicroeletronics, o L3GD20 foi escolhido, pois é necessário complementar o sensor acelerômetro, já que o acelerômetro não consegue detectar rotação em torno do vetor de aceleração da gravidade da Terra, com o giroscópio consegue-se detectar a rotação em todos os eixos independente da orientação atual. O único problema desse giroscópio é o eixo onde o acelerômetro não consegue detectar rotação, esse eixo necessitaria de uma bússola eletrônica para manter o funcionamento igual aos outros eixos, pois como não há outro sensor para se utilizar o filtro complementar, que integra os eixos do acelerômetro com o giroscópio e elimina tal erro, o eixo está vulnerável ao erro incremental típico dos giroscópios,. Existem outros giroscópios mais simples, mas como o giroscópio necessita de uma sincronização na leitura e escrita de dados mais avançado que outros sensores, é necessário um sistema de comunicação compatível com o myRIO, algo que o sensor L3GD20 possui (comunicação I²C). Esse sensor também pode ser reconfigurado de acordo com a necessidade.

3.2 CONSTRUÇÃO MECÂNICA

Para a montagem mecânica dos componentes foram utilizadas 3 placas metálicas onde os componentes foram parafusados e colados ao invés de soldados

para manter a substituibilidade do sistema. A figura 4 demonstra a montagem em perspectiva.

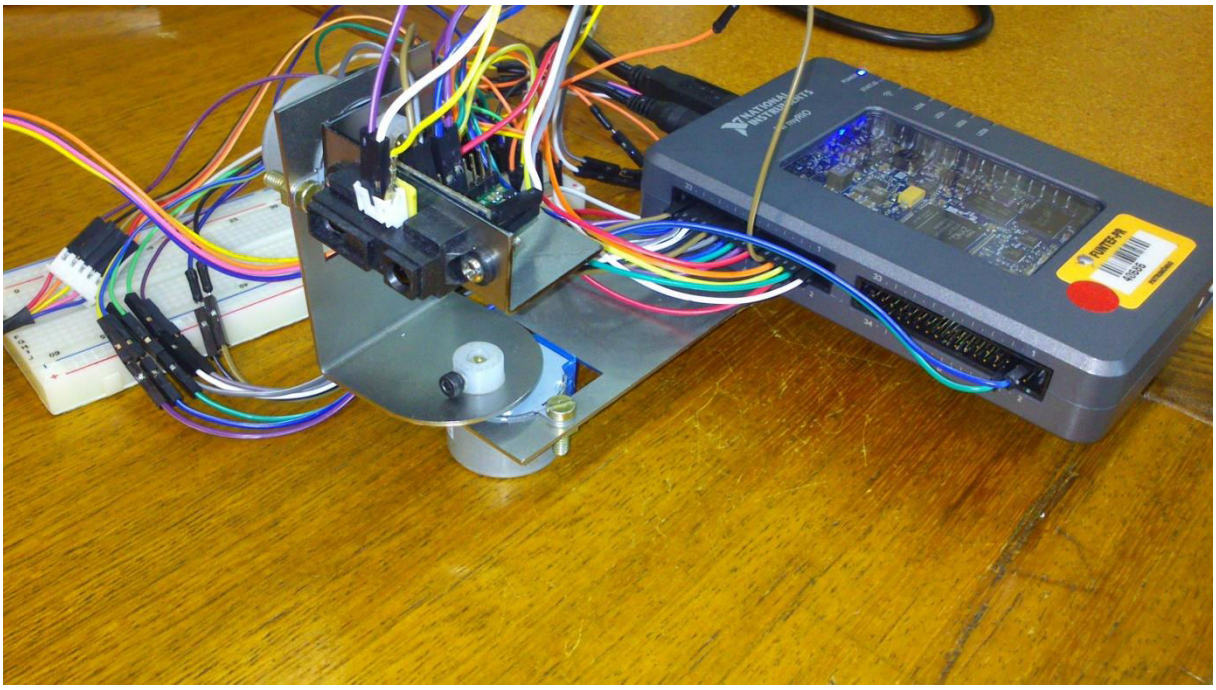


Figura 4 - Plataforma MyMapper montada

Fonte: Autoria Própria

O sistema utiliza dois motores de passo, um dos motores possui o eixo de rotação apontando na vertical, com isso, o motor roda para a esquerda ou direita, o outro motor possui o eixo de rotação paralelo ao horizonte, assim, o motor roda o sistema para cima e para baixo. O motor que faz a rotação para a cima ou para baixo é chamado de S1, o motor que faz a rotação para a esquerda ou direita é chamado de S2.

Quando os motores são acionados o número de passos executados é armazenado de forma incremental, onde o número armazenado pode ser positivo ou negativo. Em uma direção adiciona-se o número de passos à esse número, na outra direção diminui-se passos. Os passos do S1 são positivos quando o motor roda o sistema para cima e negativos quando roda para baixo e os passos do S2 são positivos quando o motor roda o sistema para direita e negativos quando roda para esquerda.

3.3 IMPLEMENTAÇÃO DO FILTRO COMPLEMENTAR

Para utilizar esse filtro com o acelerômetro e giroscópio é necessário algumas modificações, como vistas na figura 5.

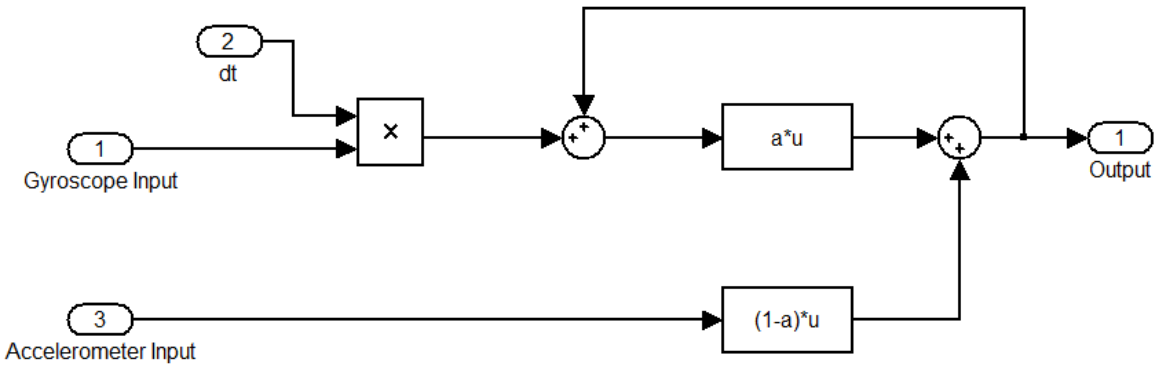


Figura 5 - Filtro Complementar para Acelerômetro e Giroscópio
Fonte: Autoria Própria

Como o giroscópio fornece a medição da taxa de rotação em graus por segundo é necessário fornecer o tempo entre cada medição, representado pela variável *dt*, também é necessário adicionar o valor do ângulo em *t-1* pois o giroscópio possui comportamento incremental e o filtro necessita o valor total do ângulo. Com esse diagrama de blocos a equação pode ser representada por:

Ângulo(t)=((Coeficiente α*)*(Ângulo(t-1)+Ângulo Giro*(dt))+(1-Coeficiente α)*(Ângulo Acel)*

Sendo o coeficiente α um número entre 0 e 1, esse filtro reduz o valor do ângulo vindo do acelerômetro, eliminando o ruído, mas criando um atraso entre o valor real do ângulo e o valor calculado pelo filtro, para eliminar esse atraso o giroscópio é utilizado. Esse filtro não é tão preciso como o filtro de Kalman, também utilizado para combinar dois sinais, mas em compensação o filtro complementar é muito mais simples de se construir e gasta muito menos ciclos de processamento, fazendo-o ideal para utilizar nesse projeto.

3.4 CÁLCULO DOS ÂNGULOS

Para se calcular os ângulos uma convenção de nomeação dos ângulos foi estabelecida. A figura 6 demonstra a definição dos três eixos da torre de sensores.

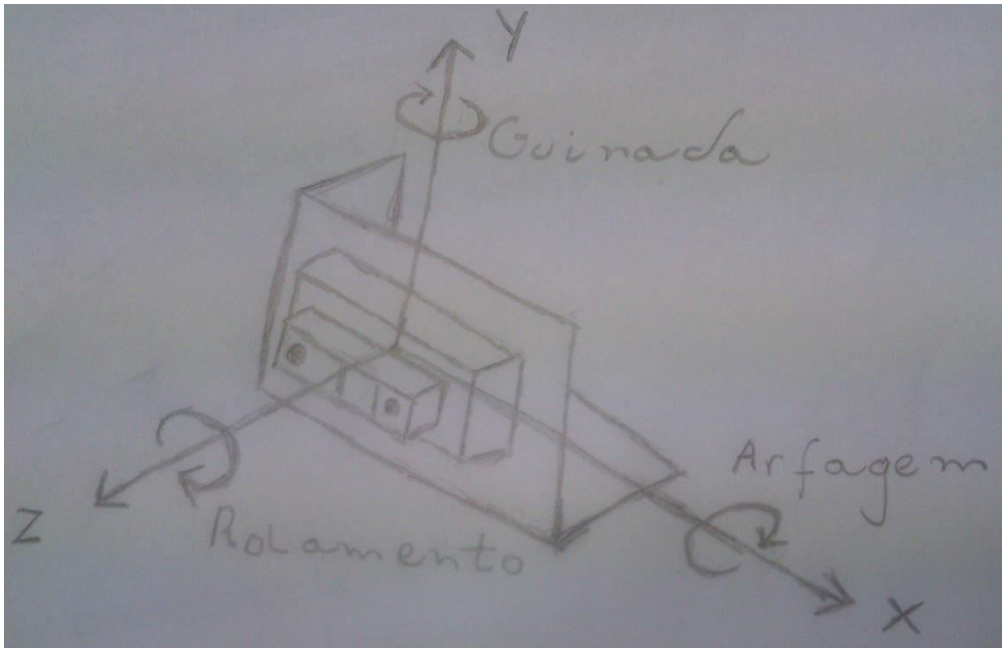


Figura 6 - Eixos do sistema
Fonte: Autoria Própria

Sendo que a direção para onde o sensor infravermelho aponta pode ser considerada como a "frente" do sistema os eixos podem ser definidos da maneira: Z apontando para frente, X apontando para a esquerda, Y apontando para cima. Assim, os valores X, Y e Z que o *Unity* utiliza podem ser interpretados como arfagem, guinada e rolamento, respectivamente.

O motor S1 está com seu eixo de rotação alinhado com o eixo X, assim, ele controla a arfagem do sistema, já o motor S2 está com seu eixo de rotação alinhado com o eixo Y, que controla a guinada do sistema.

Utilizando essa convenção como base, os eixos do acelerômetro são interpretados conforme a figura 7.

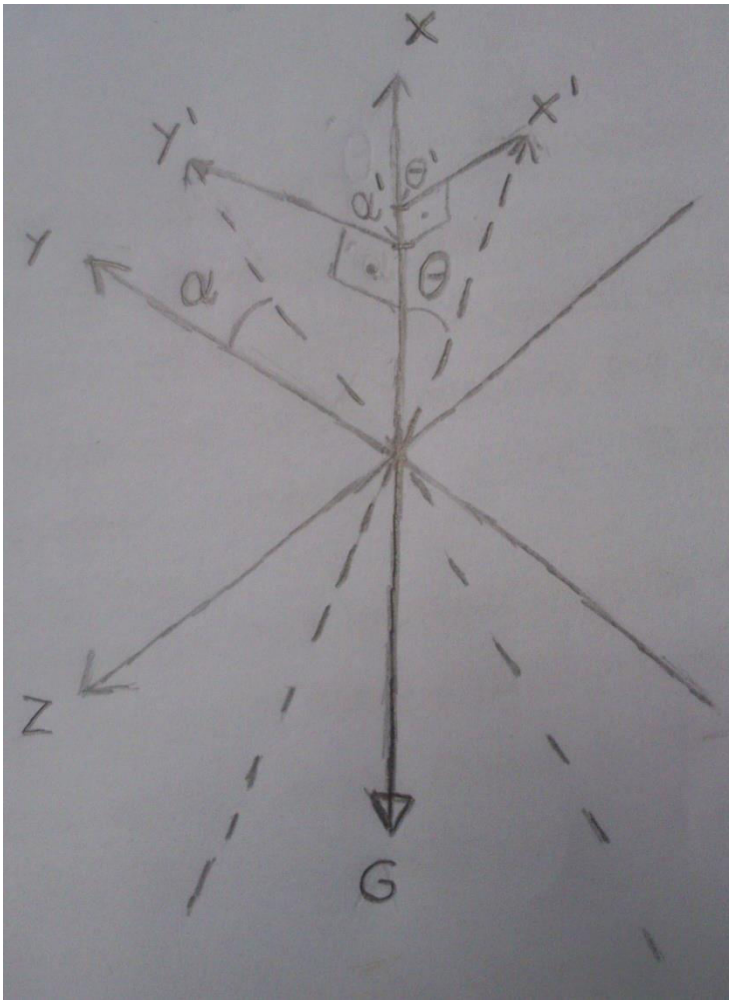


Figura 7 - Eixos do acelerômetro
Fonte: Autoria Própria

Quando o sensor acelerômetro é inclinado, os eixos X e Y se movem para a posição X' e Y', criando um ângulo entre os eixos e o vetor G, chamados de θ e α, representando arfagem e rolamento, respectivamente. Os eixos após serem movidos são projetados no vetor G que representa o vetor de aceleração da gravidade da terra, essa projeção cria os valores θ' e α', que são convertidos em graus através da fórmula:

$$\mathrm{X}° = (\sin^{-1}\theta'/\pi) * 180 \quad \text{e} \quad \mathrm{Y}° = (\sin^{-1}\alpha'/\pi) * 180$$

Já o giroscópio possui os eixos definidos conforme a figura 8.

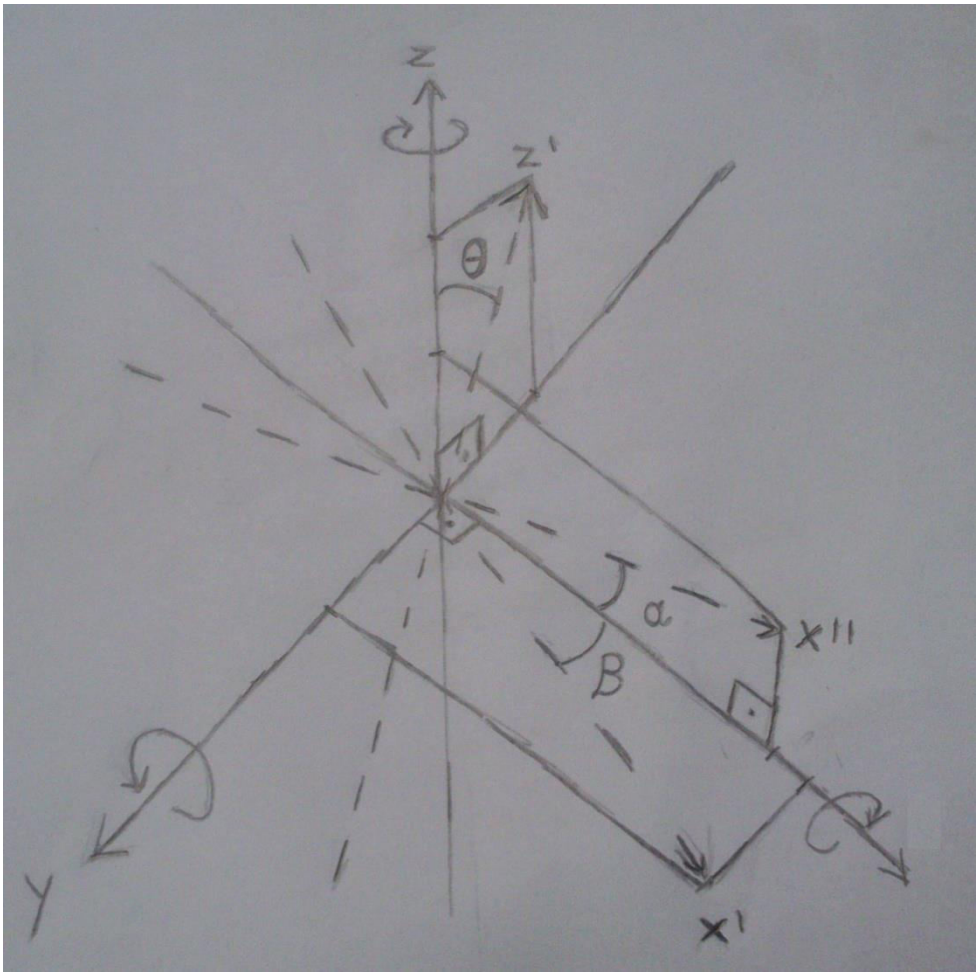


Figura 8 - Eixos do giroscópio
Fonte: Autoria Própria

Uma rotação em torno do eixo Y move o eixo X para X'' e cria um ângulo α, que é medido em graus por segundo; a rotação em torno do eixo Z move o eixo X para X', criando um ângulo β; a rotação em torno do eixo X move o eixo Z para Z', criando um ângulo θ. Assim, os ângulos α, β e θ representam o rolamento, a guinada e a arfagem.

Com isso, mesmo que os eixos X e Y de ambos os sensores não se coincidam, os valores finais desses eixos dos sensores possuem o mesmo significado.

Para se fazer a conversão dos dados vindos dos sensores para ângulos que possam ser interpretados pelo *Unity*, é utilizado, primeiramente, o filtro complementar para unir os dados do eixo X do acelerômetro com o eixo X do giroscópio e o eixo Y do acelerômetro com o eixo Y do giroscópio. Depois da

filtragem foi observado que quanto mais a plataforma é inclinada em um eixo, menor é o valor máximo obtido no outro eixo, o que cria um erro na medição.

Esse erro acontece pois o acelerômetro faz a projeção dos seus eixos internos X, Y e Z no vetor composto de todas as forças de aceleração atuando no acelerômetro. O acelerômetro possui uma massa entre duas placas capacitivas, que, de acordo com as forças de aceleração submetidas a ele cria pressão em uma mola através da inércia e move a massa, alterando o valor de tensão que percorre o sistema (YAZDI et al, 1988). Esse valor de tensão é manipulado para emitir o valor 1 quando submetido a 9,8m/s² e -1 quando submetido ao mesmo valor só que na direção oposta ao eixo medido. Portanto, quando há inclinação em dois eixos o vetor da gravidade diminui a pressão máxima em cada eixo, o que resulta em um valor de inclinação menor que o real. Esse erro do valor de inclinação não possui uma proporção linear em relação a inclinação do sensor, possivelmente devido ao método que o sensor foi construído, portanto criar uma relação matemática quando há inclinação em mais de um ângulo se torna um problema complexo.

Para minimizar esse erro duas fórmulas matemáticas e algumas condições lógicas foram usadas. Uma variável, chamada de *threshold*, foi criada, essa variável seria um ponto de limiar, no eixo X e Y, que quando o valor de algum desses eixos se torna maior que o *threshold*, o sistema modifica o método que o segundo eixo é calculado (por exemplo: o cálculo do ângulo Y seria modificado se o valor do ângulo X for maior que o valor especificado no *threshold*). Após um eixo cruzar o *threshold,* o sistema começa a armazenar o ângulo do segundo eixo utilizando somente o valor medido pelo giroscópio para aumentar e diminuir esse ângulo, que é diferente do ângulo calculado pelo filtro complementar. O ângulo final do segundo eixo é calculado então pela soma de uma porcentagem do valor do filtro complementar e o restante da porcentagem vindo do armazenamento do ângulo medido somente pelo giroscópio, essa porcentagem inicia com 100% do valor do filtro complementar e diminui linearmente de acordo com o valor do primeiro eixo, até chegar a 0% quando o primeiro eixo alcança 90°.

Como o acelerômetro não consegue detectar rotação em torno do eixo coincidente com o vetor de aceleração da gravidade (inutilizando o valor que viria do eixo Z), as fórmulas do filtro complementar e a fórmula para minimizar o erro são utilizadas somente nos eixos X e Y.

A primeira e a segunda fórmulas utilizadas para o cálculo do ângulo X são descritas da maneira:

*AngleXPureOut=((AngleXGyro*dt)+AnglePureXIn)*

*AngleXFinal= (AngleXPureIn*b)+(AngleXOut*(1-b))*

Sendo:

- *AngleXPureOut*, o ângulo que é calculado somente utilizando os valores medidos pelo giroscópio;
- *AngleXGyro*, o ângulo em graus por segundo medido pelo giroscópio;
- *AngleXIn*, o ângulo igual a AngleXpureOut quando t-1, necessário pois o sensor giroscópio mede a aceleração e não a rotação, então precisa-se adicionar o valor de rotação ao valor atual do ângulo;
- *AngleXFinal*, o ângulo final que é enviado ao *Unity* para reconstrução em 3D;
- *AngleXOut*¸ o ângulo vindo do filtro complementar;
- "*b*", esse valor é calculado utilizando a fórmula:

  b=(|AngleYOut|-threshold)/(90-threshold)

  Onde, *AngleYOut* seria o valor vindo do filtro complementar que calcula o ângulo Y. Também se tem a condição lógica:

  Se *(|AngleYOut|-threshold)<0*, *"b"=0 e AngleXPureIn=AngleXOut*.

O cálculo do valor final do ângulo do eixo Y é exatamente igual o do eixo X, somente os nomes das variáveis que contém "X" mudam para "Y" e vice-versa. Após os cálculos os valores finais dos eixos X e Y são enviados para o *Unity*.

Com essa fórmula pode se eliminar em parte o erro quando dois eixos estão inclinados, mas essas fórmulas introduzem erro incremental causado pelo giroscópio, por isso não é recomendável utilizar o MyMapper em ângulos inclinados por períodos longos de tempo, pois para corrigir o valor do ângulo que utiliza somente o giroscópio (e portanto possui erro incremental) é necessário que o ângulo do outro eixo esteja menor que o *threshold*.

Para calcular o valor do eixo Z é possível utilizar dois métodos. O primeiro método envolve utilizar as contagens de passos realizados pelo motor "S2" e multiplicá-lo pelo valor da opção *Stepper Conversion*, que transforma o número de passos em graus, esse método é o mais confiável, pois somente durante extensiva

utilização foi notado erro incremental, mas esse método não consegue detectar rotação no eixo Z exercida no sistema. O segundo método utiliza os valores enviados pelo giroscópio sobre o eixo Z para calcular a rotação, esse método consegue detectar qualquer movimento mas possui o erro incremental típico do giroscópio.

3.5 AJUSTE DE CURVA

Para se fazer a conversão do valor da tensão de resposta do sensor telêmetro em centímetros é utilizado o princípio de ajuste de curva (PEDROSA, 2014). O ajuste de curva utiliza uma tabela que representa o gráfico da curva de tensão de resposta por distância, com isso, o ajuste de curva retorna vários valores numéricos representantes dos coeficientes de uma equação polinomial que representa essa curva. Com os coeficientes polinomiais, é subtraído do primeiro termo (o termo "A" de "A*X^0") o valor atual medido pelo telêmetro. Então é utilizado a função para encontrar as raízes da equação polinomial, sendo que, os valores das raízes devem ser todos números complexos, exceto um dos valores, que é o valor convertido de tensão em distância. A unidade de medida desse valor é a mesma utilizada na tabela da curva, sendo assim, nesse projeto a tabela foi feita utilizando valores em centímetros.

A figura 9 mostra o erro da curva em vários pontos a partir do grau do polinômio da curva utilizado.

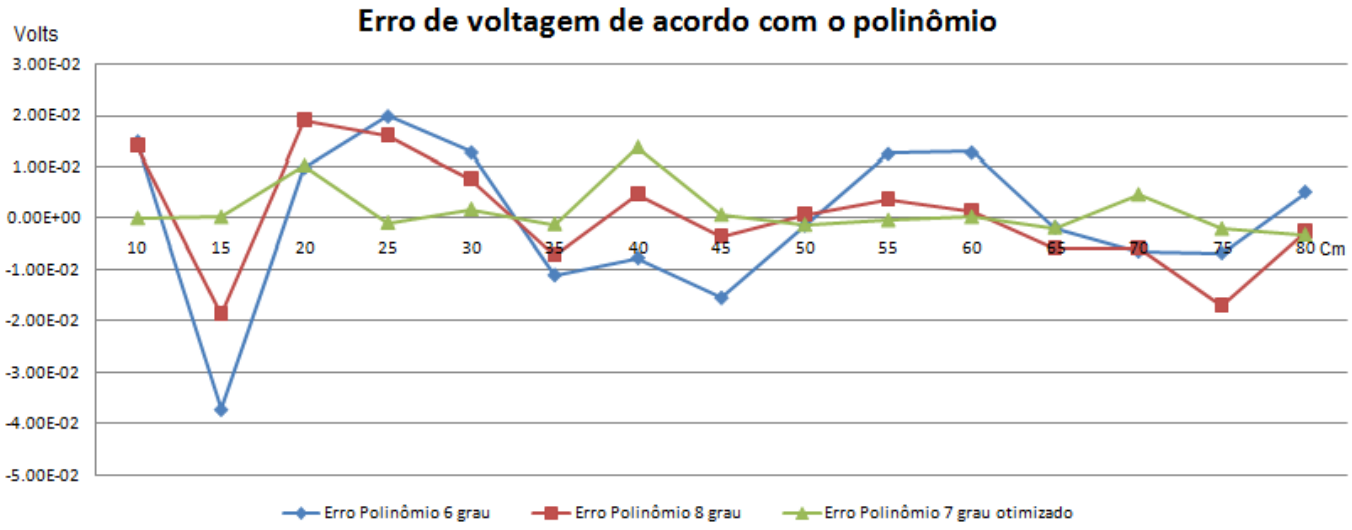


Figura 9 - Erro de Voltagem de Acordo com o Polinômio

Fonte: Autoria Própria

Como pode se observar o erro não necessariamente diminui com o aumento o grau do polinômio, com um polinômio de grau 9 ou maior ocorre o “fenômeno de Runge” (EPPERSON, 1987). Esse fenômeno explica que toda tabela de pontos equidistantes possui um número máximo de grau de polinômio que pode ser utilizado para interpolação polinomial, quando o grau desse polinômio excede o número máximo pode ser observado um grande aumento na instabilidade do sistema a partir de suas bordas.

Com esse fenômeno e a figura 9 em mente, pode-se concluir que o polinômio de grau 7 possui a melhor precisão para esse sistema. Utilizando algumas opções na função de ajuste polinomial disponível no *LabVIEW* é possível aumentar um pouco mais a precisão do ajuste da curva.

3.6 INTERFACE E PROGRAMAÇÃO

A interface foi criada com foco na simplicidade, os gráficos utilizados foram somente os padrões disponíveis no *Unity*, utilizando somente gráficos de *skybox* (FANTASY SKYBOX FREE, 2016) de licença livre, para que se possa diferenciar quando se está olhando para cima ou para baixo. Foram criadas todas as opções consideradas mais relevantes para configuração e calibração dos componentes, pensando também na permutabilidade dos componentes, pois componentes como o giroscópio e o sensor infravermelho possuem várias versões que são compatíveis fisicamente e eletronicamente mas possuem capacidades e métodos de calibração diferentes. Portanto, para manter o projeto mais flexível a futuras modificações as opções são completamente abertas aos usuários. A interface também está organizada de modo a sempre poder exibir todos os dados sendo lidos e enviados em cada sistema de forma simples para que se possa fazer testes mais específicos de cada componente.

A programação foi feita com o foco na flexibilidade do sistema, criando-se blocos de código modular, visíveis no *LabVIEW* como grandes blocos de código envoltos pela estrutura *Case* ou *While* ou como blocos de VI's próprias quando possui um número de entradas e saídas menores que 6. Também visíveis no *Unity* como funções separadas e com nomes relacionados a sua função, reduzindo a lógica externa de modo que fique mais fácil de se compreender o código.

3.7 COMUNICAÇÃO UDP

O método utilizado para realizar a comunicação entre *LabVIEW* e *Unity* é chamado de UDP (*User Datagram Protocol*), esse protocolo se diferencia do TCP, que é mais comumente utilizado, pois não é necessário uma conexão para enviar mensagens, o que simplifica o sistema, mas não é possível saber quantos programas estão "escutando" as mensagens sendo enviadas através do protocolo, o que cria a necessidade de uma rotina para verificar que ambos os programas *Unity* e *LabVIEW* estão habilitados a escutar e enviar mensagens através do protocolo.

O *software LabVIEW* possui suporte para a comunicação UDP, mas não possui uma função muito importante, a função para checar se há dados disponíveis para a leitura. Sem essa função o sistema sofre uma pausa de um segundo todo o momento em que tenta ler a próxima mensagem no *buffer* (o local aonde as mensagens a serem lidas são armazenadas) sem que haja algo para ser lido. Para resolver esse problema foi necessário o desenvolvimento de uma livraria de funções em C# utilizando as funções da livraria .NET, com essa livraria foi criado um arquivo .dll que pode ser importado pelo *LabVIEW* e permite a utilização de todas as funções .NET, com isso pode-se utilizar a função para checar o número de *bytes* disponíveis para leitura, eliminando as pausas no *LabVIEW*.

Todos os comandos disponíveis no sistema estão citados no próximo quadro.

Quadro 1 - Lista de Comandos da Comunicação UDP

Comando	**Variáveis**	**Função**
	UNITY* PARA *LABVIEW	
1,UInt	UInt =Valor da variável UDP Message Delay	Mensagem enviada ao *LabVIEW* com o *Delay* das mensagens, esse comando retorna a *string* "1-0.READY" do *LabVIEW*.
5,Bool1= Bool2= Bool3= Bool4	Bool1 = Controle S1; Bool2 = Inverter rotação S1; Bool3 =Controle S2 (1=acionado, 0=desativado); Bool4 = Inverter rotação S2 (1=sentido anti-horário, 0=sentido horário).	Controle do acionamento e sentido de rotação de cada motor.
6,Bool1= Bool2= Bool3	Bool1=Habilitar o Acelerômetro; Bool2= Habilitar o Giroscópio; Bool3= Habilitar sensor IV; (1=habilitado, 0=desabilitado).	Habilita ou desabilita a leitura dos dados dos sensores vindos do MyRIO.

10,Int1=Int2	Int1=Quantos passos o motor S1 deve executar; Int2= Ordem de quantos passos o motor S2 deve executar; (número positivo para giro horário, número negativo para giro anti-horário).	Esse comando envia o número de quantos passos cada motor deve executar.
12,Bool1	Bool1=Zerar os passos do motor S1; Bool2=Zerar os passos do motor S2; (1= zerar os passos, 0=não zerar).	Esse comando zera o número de passos armazenados de cada motor.
LABVIEW* PARA *UNITY		
0,READY		Resposta ao comando 1 do *Unity*, finaliza a rotina de conexão.
1A		Comando que faz o pedido de envio do comando 1 do *Unity*.
2,Float1,Float2,Float3	Float1= Eixo X do acelerômetro; Float2= Eixo Y do acelerômetro; Float3= Eixo Z do acelerômetro.	Comando com os valores medidos pelo sensor acelerômetro.
3, Float1,Float2,Float3	Float1= Eixo X do giroscópio; Float2= Eixo Y do giroscópio; Float3= Eixo Z do giroscópio.	Comando com os valores medidos pelo sensor giroscópio.
4,Float1	Float1= Distância do obstáculo detectado pelo sensor IV.	Comando que envia um valor calibrado representando a distância do objeto detectado pelo sensor IV.
8,Bool1, Bool2, Bool3, Bool4	Bool1=Estado do motor S1; Bool2=Direção de rotação do motor S1; Bool3=Estado do motor S2 (1=acionado, 0=parado); Bool4=Direção de rotação do motor S2 (1=sentido anti-horário, 0=sentido horário).	Valores booleanos que representam o estado de cada motor.
9,Int1,Int2,Bool1,Bool2	Int1= Passos feitos pelo motor S1; Int2= Passos feitos pelo motor S2; Bool1=Alvo alcançado pelo motor S1; Bool2=Alvo alcançado pelo motor S2 (1=alcançado,0=não alcançado/motor em movimento).	Comando que envia o número de passos efetuados pelos motores e se o número alvo de passos já foi alcançado.
11,Float1,Float2,Float3	Float1=Ângulo Final X; Float2= Ângulo Final Z; Float3=Conversão de passos em	Comando que envia os ângulos calibrados e filtrados X e Y medidos

	ângulos.	pelos sensores e o valor de conversão de passos realizados pelos motores em graus.

Fonte: Autoria Própria

Para que o comando seja corretamente interpretado ele precisa seguir o seguinte modelo:

*[Número **n** de comandos na string]-[Comando1]/[Comando2]/.../[Comando **n**]*

Se esse modelo não for seguido, os leitores de comandos do *Unity* e *LabVIEW* não irão funcionar corretamente.

A rotina de conexão inicia, normalmente, com o *LabVIEW* enviando o comando "1A", mas ela também pode começar a partir do *Unity* mandando o comando "1,Uint". Quando *Unity* recebe o comando "1A" o programa responde com o comando "1,Uint" e espera a resposta "0,READY" para confirmar que a conexão está funcionando corretamente. O sistema de comunicação UDP não detecta quando um dos lados da comunicação desconecta, por isso foi necessário programar a rotina de conexão para que se possa executá-la múltiplas vezes, o que pode ser usada para se ter certeza que ambos os lados estão "escutando".

3.8 FUNÇÕES DA INTERFACE *UNITY*

A interface *Unity* possui vários métodos de funcionamento, sendo eles: manual, onde a entrada de dados de posição de cada esfera é manual; auto, onde os sensores são utilizados para capturar os dados e mapear automaticamente, podendo ter algumas mudanças no método de captura dependendo das opções escolhidas pelo usuário; olho livre, onde o usuário pode mover a câmera pelo ambiente 3D; Salvar e carregar, todas as posições das esferas são salvas em um arquivo de texto que pode ser editado e carregado de volta.

A tela principal da interface *Unity* é mostrada na figura 10.

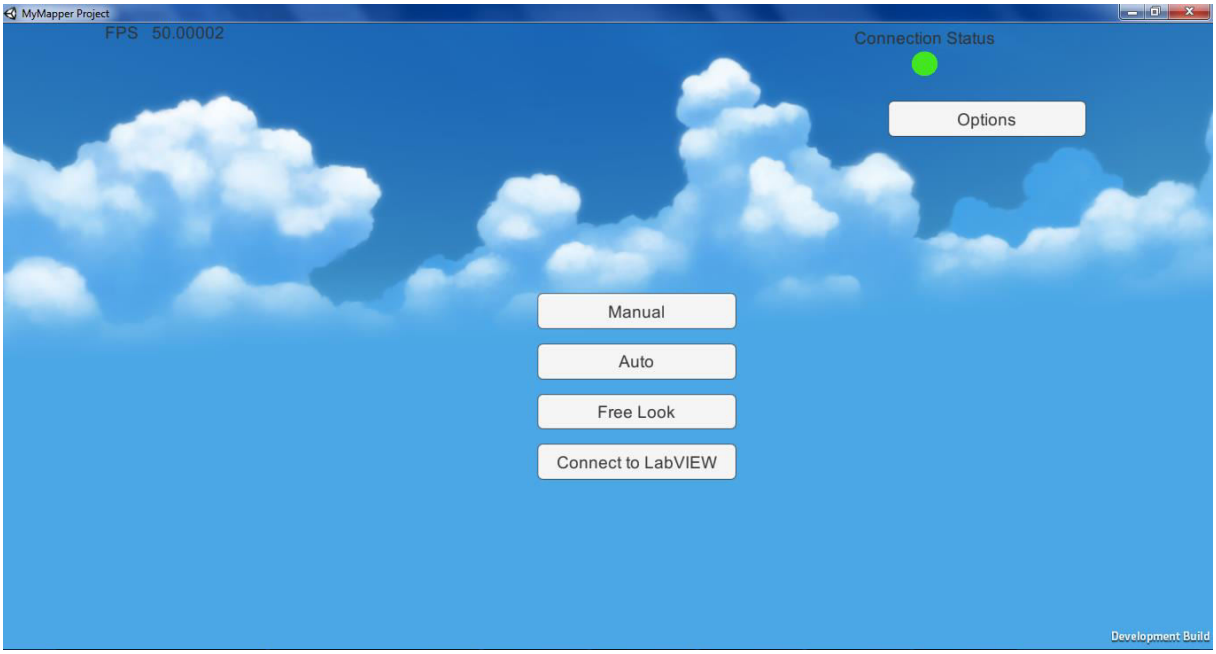


Figura 10 - Menu principal *Unity*
Fonte: Autoria Própria

O círculo colorido abaixo do texto *Connection Status* no canto superior direito da tela demonstra o estado da conexão com o *LabVIEW*. Se o círculo está vermelho, o *Unity* não está conectado, se amarelo, o *Unity* está tentando conectar com o *LabVIEW*, se verde, o *Unity* conseguiu completar a rotina de conexão com o *LabVIEW*.

3.8.1 **MAPEAMENTO MANUAL**

O botão *Manual* muda a interface para o método de mapeamento manual, como visto na figura 11.

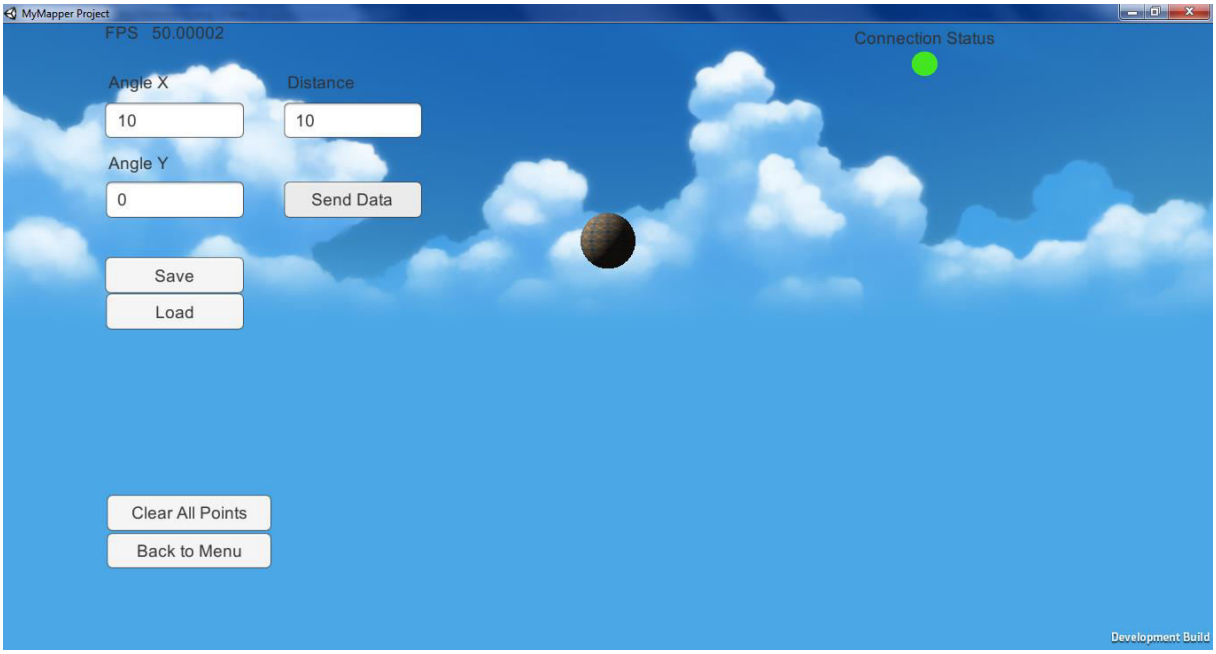


Figura 11 - Tela de Controle Manual *Unity*
Fonte: Autoria Própria

Nessa interface pode-se inserir manualmente os dados de posição de cada esfera. *Angle X* para rotação em torno do eixo que aponta para a direita, sendo o número positivo representando rotação para cima, *Angle Y* para rotação no eixo que aponta para baixo, sendo o número positivo representando a rotação para a direita, *Distance* para representar a distância em centímetros entre a câmera e o objeto detectado.

Pode-se salvar e carregar arquivos com as posições das esferas, esse arquivo é salvo em texto puro com o formato de (X,Y,Z) em uma linha para cada esfera, esse arquivo pode ser editado a qualquer momento. Para salvar pode-se adicionar as posições das esferas atuais com as das salvas em um arquivo já existente utilizando a opção *Add* ou sobrescrever o arquivo inteiro com os valores das novas esferas com a opção *Overwrite*.

3.8.2 **MAPEAMENTO AUTOMÁTICO**

De volta a tela principal, ao selecionar o botão *Auto* a interface se modifica para o método de mapeamento automático, apertando o botão *Options* e o botão *Toggle Manual Control* a interface se torna igual à figura 12.

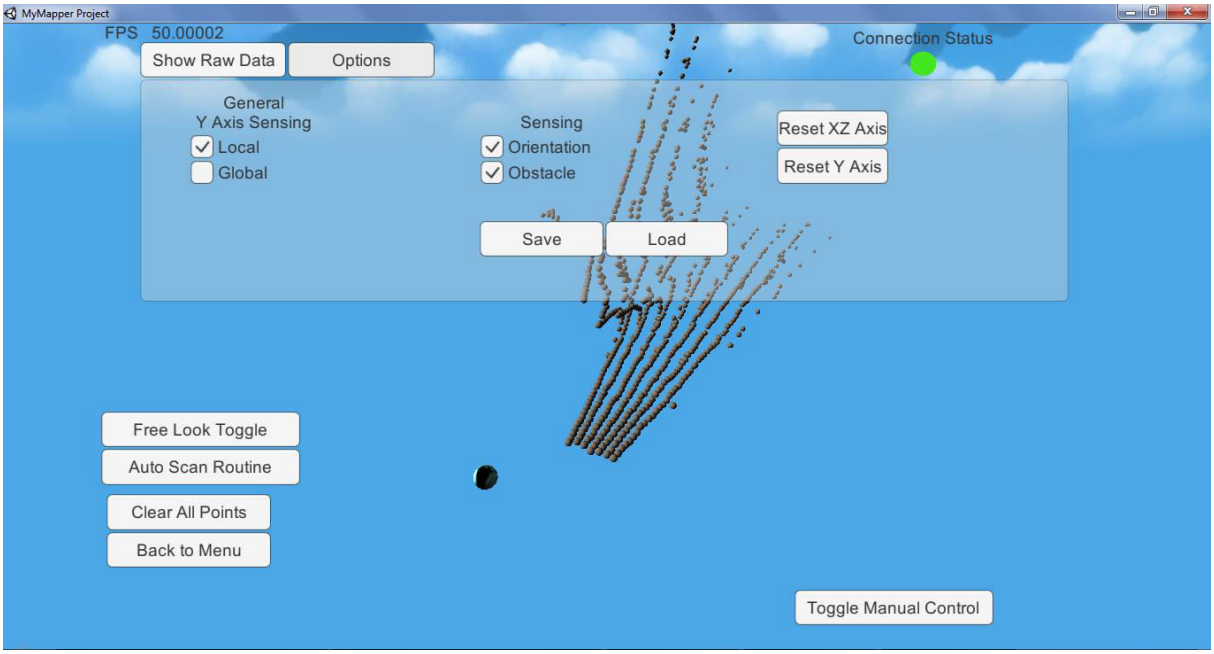


Figura 12 - Tela de Controle Automático *Unity*

Fonte: Autoria Própria

Nessa imagem pode-se observar a rotina de varredura automática em funcionamento. A maior esfera na parte inferior da tela representa a posição e a orientação do sensor de infravermelho. Apertando o botão *Free Look Toggle* pode-se navegar pela tela com o mesmo método de navegação do modo *Free Look*.

Clicando no botão *Show Raw Data* uma tela mostra todos os dados sendo recebidos pelo *Unity*, com a opção de zerar os passos de ambos os motores apertando os botões *Reset S1* e *Reset S2*.

A tela *Options* possui as seguintes opções: *General Y Axis Sensing*, essa opção controla o sensoriamento do eixo Y (guinada), podendo ser local, que utiliza os passos do motor S2 para calcular a rotação ou pode ser global, utilizando o eixo Y (marcado como Z no giroscópio) do giroscópio para calcular a rotação em Y; *Sensing*, que pode se habilitar o sensoriamento da orientação do sistema e o sensoriamento dos obstáculos em frente ao sensor; *Reset XZ Axis* que reseta os eixos X e Z, mas se o sensoriamento de orientação estiver habilitado o sistema irá para a posição zero por uma tela e depois voltará a posição medida pelos sensores; *Reset Y Axis*, zera a orientação no eixo Y. Os botões *Save* e *Load* na tela de opções salvam e carregam essas opções no arquivo *AutoOptions.txt* no formato 1 ou 0 para cada opção.

3.8.2.1 ROTINA DE MAPEAMENTO AUTOMÁTICO

A função de mapeamento automático, iniciada ao se clicar no botão *Auto Scan Routine*, funciona seguindo uma rotina específica, utilizando a posição inicial de descanso como ponto médio. A rotina automática gira os sensores utilizando os valores máximos estabelecidos nas opções *S1 Vertical Limit* e *S2 Horizontal Limit*, disponíveis na tela de opções do *Unity*, esses valores devem ser sempre interpretados como graus.

No começo da rotina automática os sensores são girados para a direita (interpretando a direção inicial em que o sensor telêmetro está apontando como a "frente" do sistema) num valor de metade do valor da opção *S1 Vertical Limit* e para baixo em um valor de metade do valor da opção *S2 Horizontal Limit*. Após o posicionamento inicial os sensores são girados para cima e para baixo no valor total da opção *S1 Vertical Limit* enquanto são girados para a esquerda aos poucos até alcançar uma rotação total igual ao valor da opção *S2 Horizontal Limit*, que então começa a girar novamente para a direita aos poucos até atingir a posição inicial e reiniciar a rotina, efetivamente fazendo uma varredura da área em frente ao sensor. A rotina utiliza o número de passos para calcular em quantos graus foram girados os sensores, por isso é importante que a conversão de passos para graus seja precisa para que a rotação máxima seja igual à especificada nas opções de limite.

A figura 13 exemplifica o padrão da varredura do telêmetro.

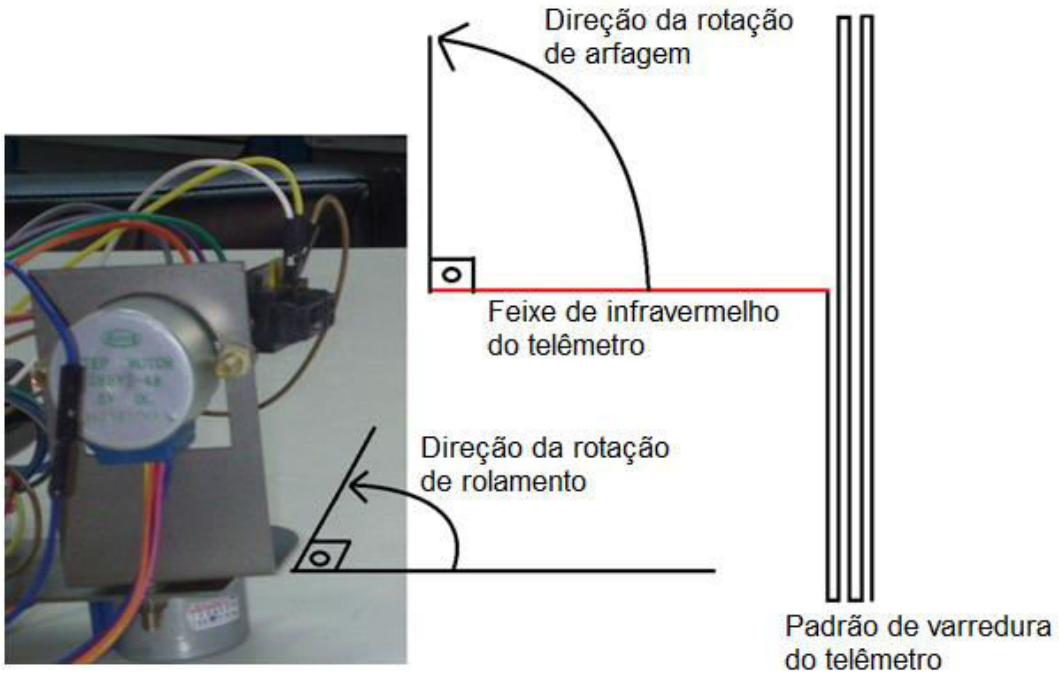


Figura 13 - Esquema do Padrão de Varredura Automático
Fonte: Autoria Própria

Apertando o botão *Toggle Manual Control* (ou clicando no botão *Auto Scan* de novo) pode-se cancelar o *scan* automático e acionar os motores manualmente, ao segurar os botões *Up*, *Down*, *Left* ou *Right* pode-se girar o sistema nas direções especificadas. Inserindo um número positivo ou negativo no campo *S# Target Steps* e apertando o botão *Send Steps to Target* pode-se controlar quantos passos cada motor deve executar. É importante observar que quando os dois motores estão sendo acionados, o myRIO cria um ruído muito alto no circuito, o que influencia muito o sensor telêmetro, criando um alto erro de medição se o sensoriamento de obstáculos estiver ligado, ele aparenta ser causado somente pelo myRIO quando se acionam duas saídas digitais.

3.8.3 **OPÇÕES**

O botão *Options* abre a interface onde se pode configurar as opções e controles, como visto na figura 14.

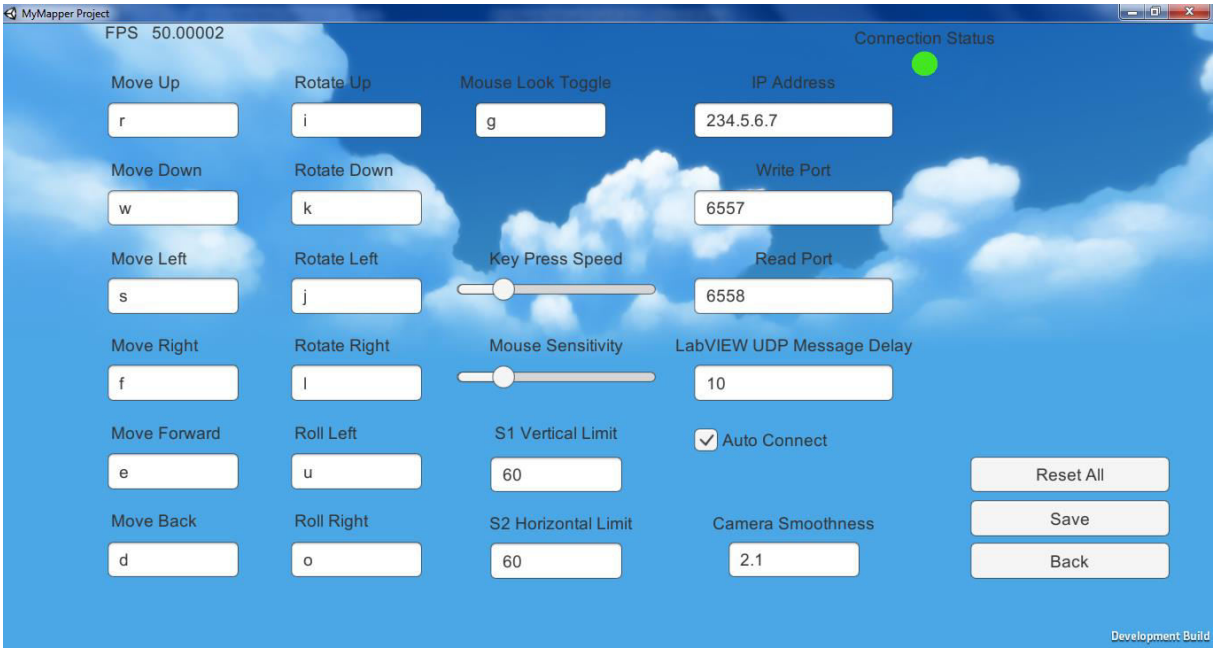


Figura 14 - Tela de Opções *Unity*

Fonte: Autoria Própria

Os controles *Move Up*, *Move Down*, *Roll Left, Roll Right* e etc., são utilizados no controle de olho livre, esses controles permitem se navegar pelo ambiente 3D, sendo o botão *Mouse Look Toggle* utilizado para selecionar se o giro da câmera é controlado pelo teclado ou pelo mouse. A opção *Key Press Speed* controla a velocidade de translação da câmera quando se segura o botão *Move Up* e etc. As opções *S# Vertical/Horizontal Limit* controlam, em graus, o máximo que os motores giram os sensores durante o método de *scan* automático. A opção *LabVIEW UDP Message Delay* controla o tempo entre cada mensagem enviada pelo *LabVIEW* para o *Unity*, esse controle automaticamente se ajusta caso o número de bytes no *buffer* do *Unity* passe de 2000. A opção *Camera Smoothness* controla a movimentação da câmera para que ela não faça movimentos muito repentinos, o que pode causar desconforto ao usuário.

3.8.4 **MODO OLHO LIVRE**

O botão *Free Look* na tela principal inicia o modo de olho livre, nesse modo pode-se navegar pelo ambiente utilizando o mouse e teclado. As opções e controles

configurados no menu de opções modificam os controles e a velocidade do mouse e da translação desse método.

3.8.5 CONECTANDO AO *LABVIEW*

Essa interface pode-se conectar ao *LabVIEW*, observar e testar a comunicação UDP, a figura 15 mostra a tela primária:

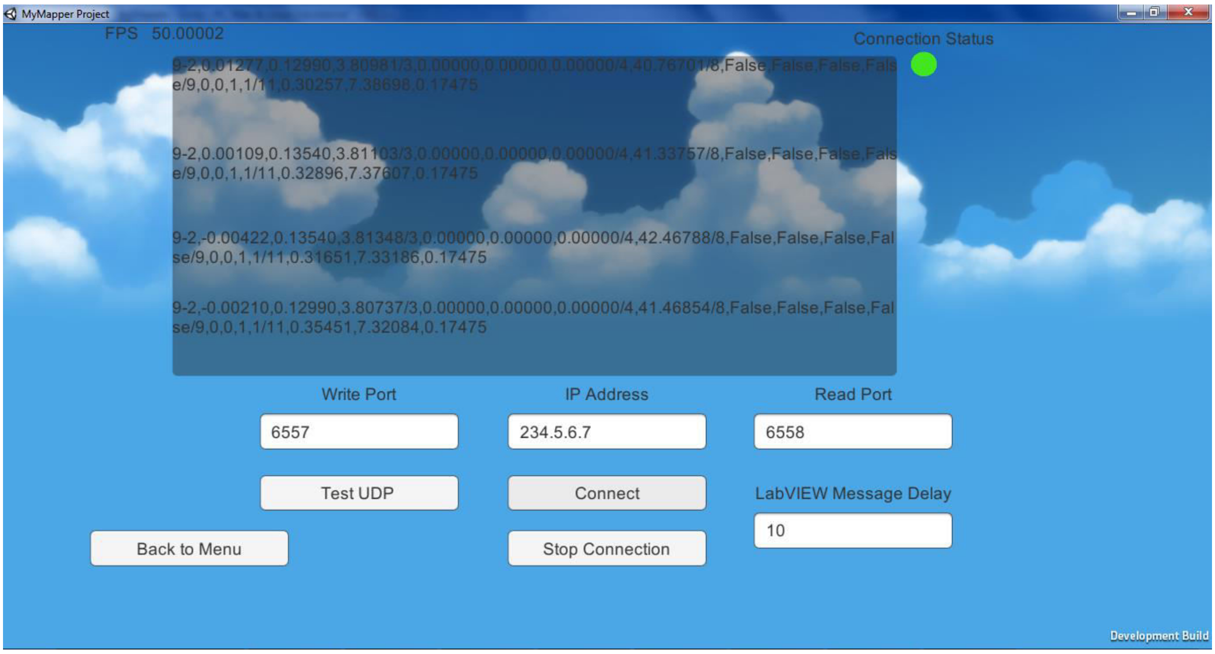


Figura 15 - Tela de Conexão *Unity*

Fonte: Autoria Própria

Nessa tela pode-se observar as *strings* sendo lidas pelo *Unity* e os botões abaixo da tela que permitem conectar com o *LabVIEW* de acordo com os dados nos campos visíveis, normalmente não será necessário modificar esses campos. O campo *LabVIEW* Message Delay controla o tempo entre cada mensagem enviada pelo *LabVIEW* ao *Unity*, essas mensagens são guardadas em um *buffer* até que sejam lidas e enfim descartadas, se o *buffer* contém mais de 2000 *bytes* o *Unity* aumenta o *Delay* aos poucos automaticamente, se o *buffer* enche todo o espaço disponível ele deleta as mensagens mais velhas para abrir espaço para as mais novas.

Apertando o botão *Test UDP* abre uma nova interface para o teste da comunicação UDP como visto na figura 16.

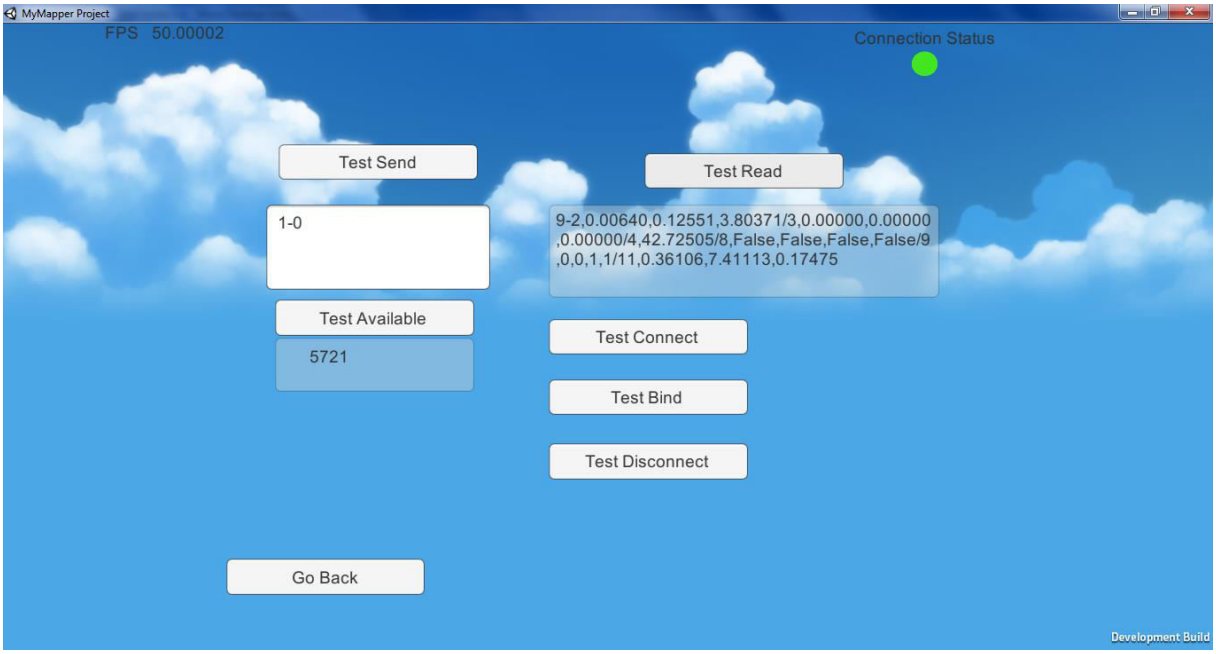


Figura 16 - Tela de Teste de UDP *Unity*

Fonte: Autoria Própria

Essa interface pode-se testar a comunicação UDP com o *LabVIEW*, é importante lembrar que se a opção *Auto Send Sensor Data* estiver habilitada no *LabVIEW* o *buffer* rapidamente se encherá e ficará difícil observar respostas específicas, pois as mensagens com os dados dos sensores são enviadas em rápida sucessão. O botão *Test Available* é utilizado para se observar o número de *bytes* no *buffer* disponíveis para leitura.

Para-se conectar ao *LabVIEW* é necessário primeiramente apertar o botão *Test Bind*, que vincula o IP e o *Port* ao *Unity*, após pode-se apertar o botão T*est Connect* que irá inicializar todos os dados necessários para a conexão. Para finalizar a conexão é necessário enviar o comando "1,Uint" manualmente.

3.9 FUNÇÕES DA INTERFACE *LABVIEW*

3.9.1 TELA PRINCIPAL

A tela principal pode ser observada na figura 17.

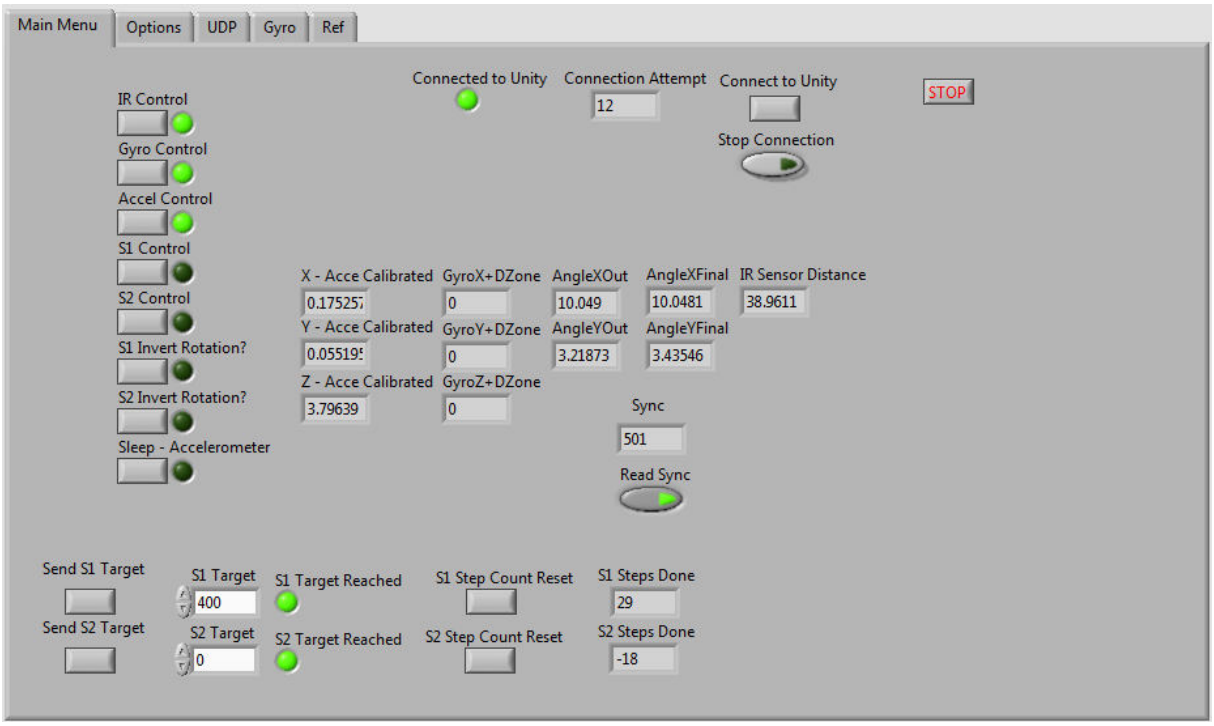


Figura 17 - Menu Principal *LabVIEW*

Fonte: Autoria Própria

Os botões a esquerda controlam os sensores e os motores, desabilitando os sensores também desabilita o envio dos dados para o *Unity*. Os motores normalmente giram em direção horária, para fazê-los girar na direção oposta é necessário habilitar o botão *S# Invert Rotation?*

Os botões na parte inferior controlam os motores de forma mais precisa, colocando qualquer número na caixa de entrada *S# Target* e apertando o botão *Send S# Target* envia o número de passos desejados para os motores. Apertando o botão S# *Step Count Reset* retorna o número de passos para 0, esse número de passos também é utilizado para orientação no eixo Z, controle de passos da rotina automática e controle do método utilizado pelo *Send S# Target*, portanto, apertar o botão de *Reset* enquanto os motores estão em movimento irá afetar o funcionamento do sistema.

Na parte central do painel estão todos os dados importantes vindos dos sensores, esses são os dados já calibrados que são enviados para o *Unity*. Em baixos desses indicadores está o indicador Sync, ele serve para se ter confirmação que as variáveis compartilhadas com o myRIO estão funcionando corretamente. O Sync é um contador que adiciona 1 a cada iteração do loop e retorna a 0 quando chega a 1000, ele pode também ser utilizado para observar a velocidade do loop do myRIO e se há algum problema com as variáveis compartilhadas. Se a VI principal do myRIO estiver sendo utilizada com o código fonte, na tela frontal dessa VI há um indicador mostrando o Sync, se o Sync da VI do myRIO estiver mais rápido que o Sync do Host significa que há uma sobrecarga na comunicação das variáveis compartilhadas.

Na parte superior do painel se pode controlar o método de conexão com o *Unity*, como o padrão de comunicação é UDP não há uma conexão direta como no método TCP, mas uma rotina para garantir que a comunicação entre os programas está funcionando corretamente é executada. A conexão começa automaticamente se a opção *Auto Connect* é habilitada, durante a conexão todos os outros botões não terão efeito até a conexão terminar, somente o botão comutador *Stop Connection* pode parar a tentativa de conexão e resumir as funcionalidades do sistema.

3.9.2 **OPÇÕES**

A tela de opções pode ser observada na figura 18.

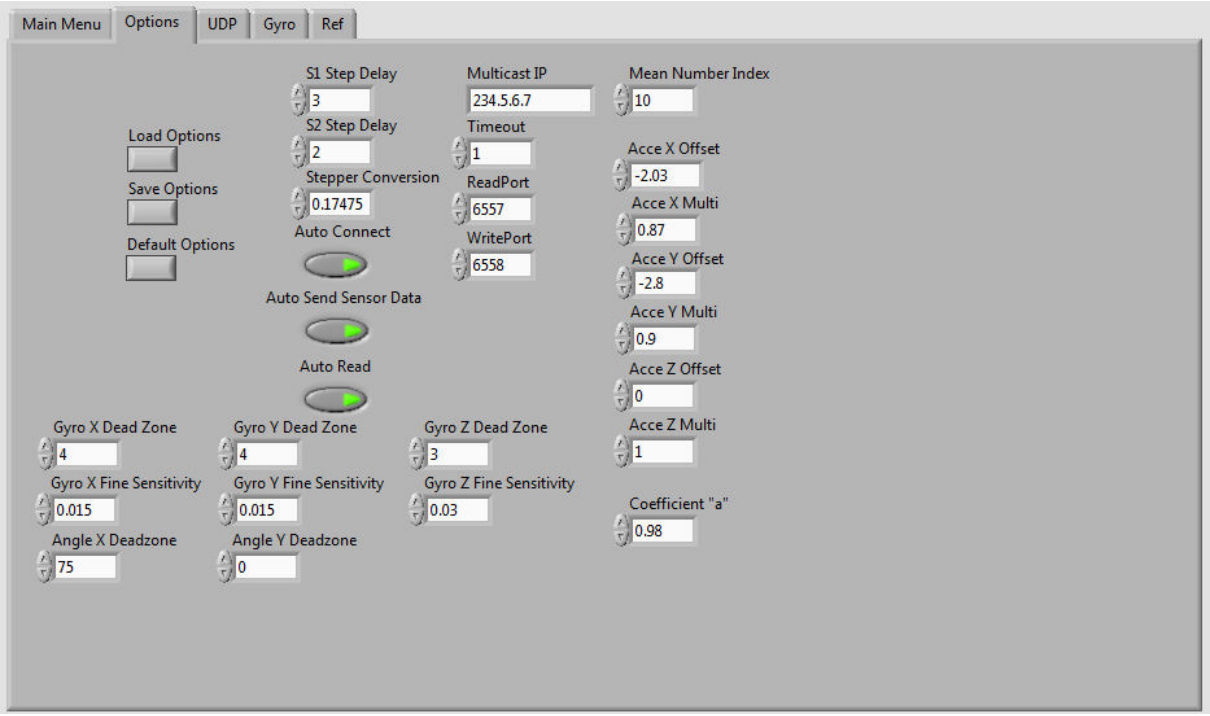


Figura 18 - Tela de Opções *LabVIEW*
Fonte: Autoria Própria

Nessa tela todas as mudanças das opções precisam ser salvas e depois recarregadas para garantir que elas tenham efeito. Se por algum acaso as opções não estiverem sendo carregadas é possível que as variáveis compartilhadas com o myRIO não estejam funcionando, pois grande parte das variáveis necessitam estar armazenadas no myRIO.

Os três botões comutadores funcionam da seguinte maneira: a opção *Auto Connect* inicia a conexão UDP toda vez que o sistema inicia; a opção *Auto Send Sensor Data* envia automaticamente todos os dados para o *Unity* nos intervalos especificados pelo *Unity* no ato da conexão, útil desabilitar quando quiser testar a comunicação; a opção *Auto Read* automaticamente lê e executa os comandos enviados pelo *Unity*, útil desabilitar quando quiser testar a comunicação. Atenção ao funcionamento do sistema, se as mensagens continuarem a serem enviadas e não forem lidas, uma fila no *buffer* de cada programa começa a aumentar, esse *buffer* possui um limite e se o limite é alcançado as mensagens mais velhas são descartadas a favor das mensagens mais novas.

Os controles numéricos *S# Step Delay* controlam a velocidade dos motores de passo, como os motores foram configurados para executar somente um passo a cada iteração do *loop*, a velocidade máxima é alcançável com um *delay* de 1, mas como esse número o motor se torna instável e tem uma chance irregular de não efetuar corretamente o passo, por isso o mínimo recomendável é 3. O controle *Stepper Conversion* é o valor pelo qual os passos dos motores serão multiplicados por para se obter o giro em graus efetuado pelo motor.

As opções *Multicast IP*, *Timeout*, *ReadPort* e *WritePort*, são opções relevantes somente ao método de comunicação entre o *Unity* e o *LabVIEW*, recomenda-se que não se faça nenhuma alteração nessas opções pois não há necessidade de calibração, as mudanças feitas nessas opções também poderão ser necessárias de serem feitas no lado do *Unity*.

Mean Number Index controla o número de valores do sensor de IV de iterações passadas que são armazenadas para se criar a média desses valores. Quanto maior o número, maior a *array* que armazena esses valores, mais devagar que o sistema fica, também mais devagar fica para se detectar objetos pelo sensor, mas também aumenta-se muito mais a precisão e diminui o ruído do sensor.

As opções *Acce (X/Y/Z) Offset* e *Acce(X/Y/Z) Multi* são utilizadas para calibrar os eixos do acelerômetro. O acelerômetro pode mandar uma tensão de reposta de VCC até 0V, esse valor precisa ser convertido para 1 até -1, sendo 1 igual a 9,8m/s². Para calibrar o acelerômetro primeiramente se posiciona o acelerômetro na orientação em que o eixo medido fique com um ângulo de 90° ao vetor de aceleração da gravidade da terra, o valor do controle *Acce(X/Y/Z) Offset* se torna o valor negativo do valor atual do eixo, o valor final calibrado a 90° deve ser o mais próximo de 0. Após esse passo posiciona-se o sensor para que o eixo esteja o mais próximo possível do vetor de aceleração da gravidade, se o valor está maior ou menor que 1 é necessário aumentar ou diminuir o valor de *Acce (X/Y/Z) Multi*, esse valor seria multiplicado pelo valor atual da inclinação. No fim, não é recomendável que os valores máximos e mínimos passem de 1,1 ou fiquem menores que 0,95.

As opções *Gyro (X/Y/Z) Dead Zone* controlam o valor mínimo do eixo do giroscópio que é necessário para ser registrado como válido. Essa opção é necessária pois o giroscópio envia valores mesmo enquanto em repouso, esses valores mudam irregularmente, podendo chegar de +3 até -3, por isso é necessária

essa “zona morta" ou então a reconstrução em 3D corre o risco de derivar lentamente ao longo do tempo.

As opções *Gyro (X/Y/Z) Fine Sensitivity* controlam a conversão dos valores enviados pelo giroscópio em °/s, esse valor seria mais preciso se coletado programaticamente através de *loops* cronometrados, mas como isso aumenta a complexidade do sistema pois cria a necessidade de sincronização do *LabVIEW* com o myRIO e da medição do tempo entre cada iteração do *loop*, uma simples variável arbitrária é o suficiente para se alcançar a precisão desejada nesse projeto.

As opções *Angle (X/Y) Deadzone* controlam o ângulo mínimo para que se inicie a utilização de um segundo valor de ângulo de inclinação que utiliza somente o giroscópio como sensor. É recomendável que esse ângulo seja sempre maior que 0 pois o segundo valor de ângulo precisa ser corrigido regularmente pois ele possui o típico erro incremental do giroscópio. Quando o ângulo atual está menor que o valor da opção *Deadzone* o segundo valor de ângulo se torna igual ao valor vindo do filtro complementar "corrigindo" esse valor.

A opção *Coefficent "a"* controla o filtro complementar. Um coeficiente alto pode melhorar a precisão do sistema, diminuindo o ruído do acelerômetro, mas cria um atraso entre o valor do acelerômetro e o valor atual do filtro complementar, um valor baixo elimina esse atraso, mas aumenta o ruído vindo do acelerômetro, por isso é importante calibrar corretamente os sensores de acordo com o coeficiente.

3.9.3 **UDP**

A aba UDP pode ser utilizada para se testar a comunicação UDP do mesmo modo que a tela de teste do *Unity* como pode se ver na figura 19.

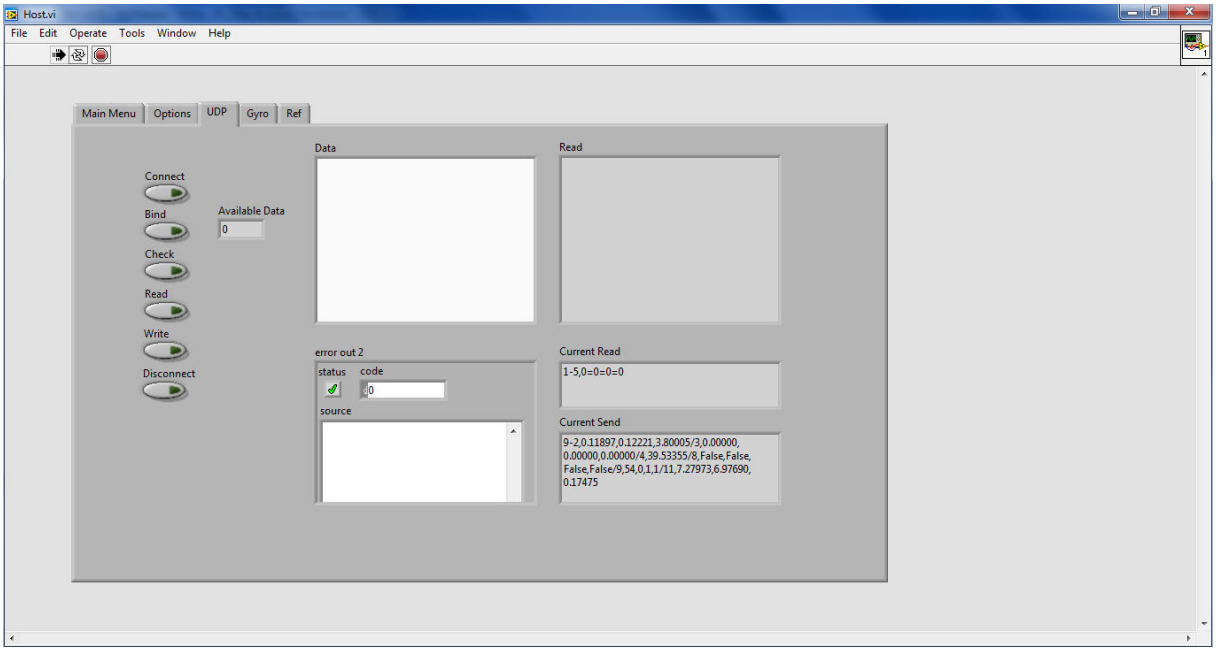


Figura 19 - Tela de Teste UDP *LabVIEW*
Fonte: Autoria Própria

Essa aba mostra também o último valor lido pelo *LabVIEW* e o último comando enviado para o *Unity*. É recomendado desativar a opção *Auto Read* para que se possa ter controle sobre quando ler os dados vindos do *Unity*, também é novamente recomendado desativar a opção *Auto Send Sensor Data* para facilitar o teste da comunicação.

3.9.4 **GIROSCÓPIO**

O giroscópio pode ser programado e configurado a partir dessa aba, para que se possa testá-lo é necessário desabilitar o sensor no menu principal do *LabVIEW* ou através do comando 6 do *Unity*, que pode desabilitar o sensor giroscópio.

Não é necessário configurar o sensor através dessa interface pois ele já é programado automaticamente com as configurações adequadas toda vez que o myRIO é inicializado.

O *LabVIEW* não envia os dados de sensores a cada iteração do *loop*, por causa disso foi necessário criar um acumulador de dados, esse acumulador faz a

somatória de todas as medições, divide pelo número de medições feitas e então esse número é enviado ao *Unity*, quando esse número é enviado o acumulador zera novamente e acumula os dados do sensor até o próximo envio de dados. Somente o giroscópio necessita desse acumulador pois o *Unity* incrementa o ângulo Y baseado no giroscópio, sem esse acumulador, quanto mais a demora entre um envio de dados e outro, maior seria a perda de dados.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Foram feitos nove testes, divididos em dois métodos: o primeiro método mapeia uma folha de papel em uma linha reta vertical, esse método tem a intenção de entender os efeitos dos diversos parâmetros de calibração dos sensores; o segundo método mapeia uma peça qualquer utilizando a rotina de mapeamento automática, esse método tem a intenção de observar a capacidade do sistema de detectar o formato de peças.

Três parâmetros principais são modificados nesses testes:

- *Step Delay*: controla o tempo entre cada passo de ambos os motores, quanto mais próximo de zero, mais rápido os motores se movem.
- α: controla o filtro complementar, para manter o sistema estável esse valor deve se manter entre 0 e 1, sendo que quanto mais próximo de 1, mais peso é dado ao giroscópio no filtro complementar.
- Valor Médio: também chamado de ***n***, controla uma função, onde o sistema armazena as últimas ***n*** medições do telêmetro, então elas são somadas e divididas por ***n***, resultando na média do valor das medições do telêmetro.

4.1 RESULTADOS

Todos os testes estão resumidos na tabela abaixo.

Quadro 2 - Testes Feitos com Diferentes Parâmetros

Teste Nº	Peça	Método	Parâmetros	Resultados
1	Folha de Papel	450 passos em uma linha vertical	Step Delay: 4 α: 0,5 Valor Médio:1	Folha levemente deformada e com ruído
2	Folha de Papel	450 passos em uma linha vertical	Step Delay: 4 α: 0,999 Valor Médio:1	Folha completamente deformada
3	Folha	450 passos	Step Delay: 25	Posição da folha com erro, ruído

	de Papel	em uma linha vertical	α: 0,99 Valor Médio:1	pequeno e erro de bordas detectado
4	Folha de Papel	450 passos em uma linha vertical	Step Delay: 25 α: 0,99 Valor Médio:200	Posição da folha com erro, ruído mínimo e erro de bordas muito visível
5	Folha de Papel	450 passos em uma linha vertical	Step Delay: 10 α: 0,8 Valor Médio:5	Posição da folha com pouco erro, leve deformação, ruído do telêmetro visível
6	Cilindro de Metal	Rotina de mapeamento automático	Step Delay: 25 α: 0,8 Valor Médio:5	Peça detectada, impossível de identificar o formato da peça
7	Cilindro de Metal	Rotina de mapeamento automático	Step Delay: 5 α: 0,8 Valor Médio:5	Peça detectada, impossível de identificar o formato da peça
8	Bloco de Motor	Rotina de mapeamento automático	Step Delay: 25 α: 0,8 Valor Médio:5	Peça detectada, detalhes não visíveis, deformação visível no formato da peça
9	Bloco de Motor	Rotina de mapeamento automático	Step Delay: 5 α: 0,8 Valor Médio:5	Peça detectada, detalhes não visíveis, diminuição na deformação da peça

Fonte: Autoria Própria

A próxima seção irá demonstrar em mais detalhes os testes feitos.

4.2 ANÁLISE DE RESULTADOS

4.2.1 **Primeiro Teste**

Nesse teste foi utilizado uma folha A4 presa a um pilar como objeto a ser medido, também foi eliminado qualquer rotação em torno do eixo Y (guinada) e Z (rolamento) zerando as opções de calibração *Acce Y Multi*, *Acce Z Multi*, *Gyro Y Fine Sensitivity* e *Gyro Z Fine Sensitivity* na interface do *LabVIEW*. Com isso, somente será detectada rotação no eixo X (arfagem), melhorando a visualização das deformações.

A figura 20 apresenta como o sistema foi montado. Primeiramente é alinhado a câmera ao horizonte, e então utilizando a função *Send S1 Target* comanda-se o motor na vertical a girar 150 passos para baixo, assim o sistema está pronto para teste.

Figura 20 - Mapeamento de uma Folha de Papel
Fonte: Autoria Própria

A figura 21 demonstra o resultado da primeira medição. Utilizando uma velocidade alta, de valor 4 no motor de passo vertical (seria a opção *S1 Step Delay* na interface *LabVIEW*) e um coeficiente α baixo de valor 0,5 (coeficiente principal do filtro complementar), o sensor é girado 450 passos para cima, armazena-se a medida do mapeamento resultante, remove-se todas as medições, roda-se o sistema novamente, agora 450 passos para baixo e armazena-se a segunda medida. As imagens foram tiradas de perfil, sendo o eixo Z (eixo na horizontal) a distância e o eixo Y (eixo na vertical) a altura do objeto. A câmera é o ponto branco a

esquerda da foto, as linhas que saem da câmera são as bordas do campo de visão da câmera e a linha vertical o horizonte.

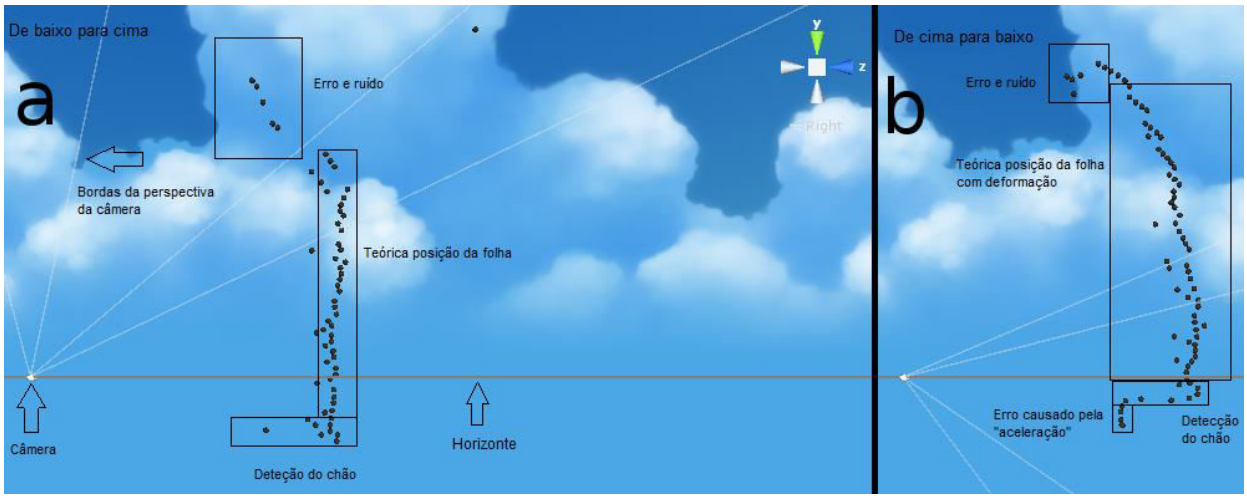


Figura 21 - Mapeamento Rápido de Folha de Papel

Fonte: Autoria Própria

A figura 21a representa uma medição de baixo para cima e a figura 21b representa uma medição de cima para baixo. Nessa figura pode-se observar uma diferença no formato da peça dependendo da direção de mapeamento e indícios de erros.

4.2.2 **Segundo Teste**

Nesse teste foi utilizado um coeficiente α muito mais alto (0,999), o que cria um atraso no valor real do ângulo e o valor do ângulo calculado pelo filtro complementar.

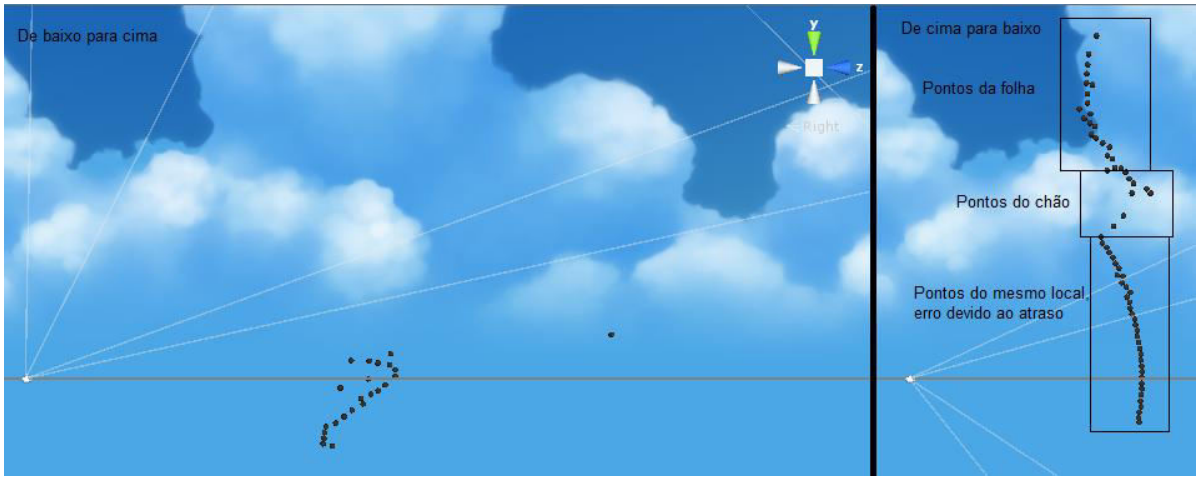


Figura 22 - Mapeamento com α Alto

Fonte: Autoria Própria

A figura 22 mostra uma grande deformação pois como o coeficiente α está muito alto, o valor do giroscópio possui maior peso no cálculo do filtro complementar, mas nesse teste não há uma zona morta para o valor do giroscópio, assim ele se mantém sempre negativo. Portanto, quando a rotação é positiva (para cima) ele possui um valor menor que o normal, fazendo com que ele varie entre positivo e negativo, resultando em valor de ângulo que dificilmente consegue sair do zero. Como o valor do filtro complementar não consegue acompanhar o valor real, seria necessário aumentar a sensibilidade do giroscópio para se reduzir a distorção da medição.

Na medição de cima para baixo pode-se observar pontos com um erro grande, o que significa que o valor do ângulo do filtro complementar demorou a alcançar o valor real, ainda tendo que ser calculado mesmo depois que o sistema já parou. Como o valor vindo do telêmetro não tem esse atraso, isso cria uma dessincronização entre os valores, sendo que os pontos de erro, mesmo estando em posições diferentes na interface *Unity,* representam o mesmo local. O comportamento desse erro pode ser entendido como uma "aceleração" exponencial de acordo com o erro entre o valor atual do ângulo e o valor de resultado do filtro complementar, sendo que quanto maior esse erro, mais rapidamente o valor do filtro complementar tenta alcançar o valor real.

4.2.3 **Terceiro Teste**

Nesse teste reduziu-se a velocidade dos motores (*delay* de 25) e foi utilizado um parâmetro α menor (0,99), diminuindo o atraso do filtro complementar. A figura 23 demonstra o resultado do teste.

Figura 23 - Mapeamento Devagar

Fonte: Autoria Própria

O desvio inferior horizontal que pode ser vista na figura 23b, seriam os pontos do chão detectados pelo sensor. O chão não aparece na figura 23a por causa da direção do movimento do sensor. Como a distância do obstáculo aumenta bastante mas sua altura aumenta pouco o sistema automaticamente deleta todos as esferas que estão na frente da esfera atual sendo detectada, isso é necessário pois não se é possível detectar a diferença entre uma peça que se moveu e aumentou sua distância do sensor e a detecção de superfícies que fazem um ângulo próximos de 0° entre o vetor do telêmetro e a superfície, como é o caso dos pontos do chão.

Abaixo do desvio horizontal há mais um desvio, esse na vertical. Esse desvio é causado pela dessincronização entre o valor do telêmetro e do ângulo, criando um atraso, igual ao segundo teste, só que muito menor.

Nessa figura também está visível uma anomalia do sensor. A folha está perfeitamente vertical, mas o sensor detectou uma "dobra" no topo da folha, como se ela estivesse inclinada para a frente. Essa é uma fraqueza do sensor, ele cria um erro quando há uma mudança muito drástica da distância entre dois pontos, esse fenômeno pode ser visto mais adiante pois ele cria muitas deformações.

Nesse teste pode-se ver como a exatidão aumentou em comparação com o teste anterior, mas pode-se observar que ainda há um atraso, por causa dos pontos que fazem uma um desvio na parte inferior da figura.

4.2.4 **Quarto Teste**

Nesse teste foi utilizado a função de valor médio de distância do sensor telêmetro. Essa função visa a diminuir o ruído existente no sensor telêmetro, fazendo uma média das medições anteriores, a quantidade de medições utilizadas é definida pelo usuário. Essa função cria um problema similar ao do filtro complementar, pois aparece um atraso entre o valor real e o valor sendo calculado pelo filtro, o que pode ser visto na figura 24.

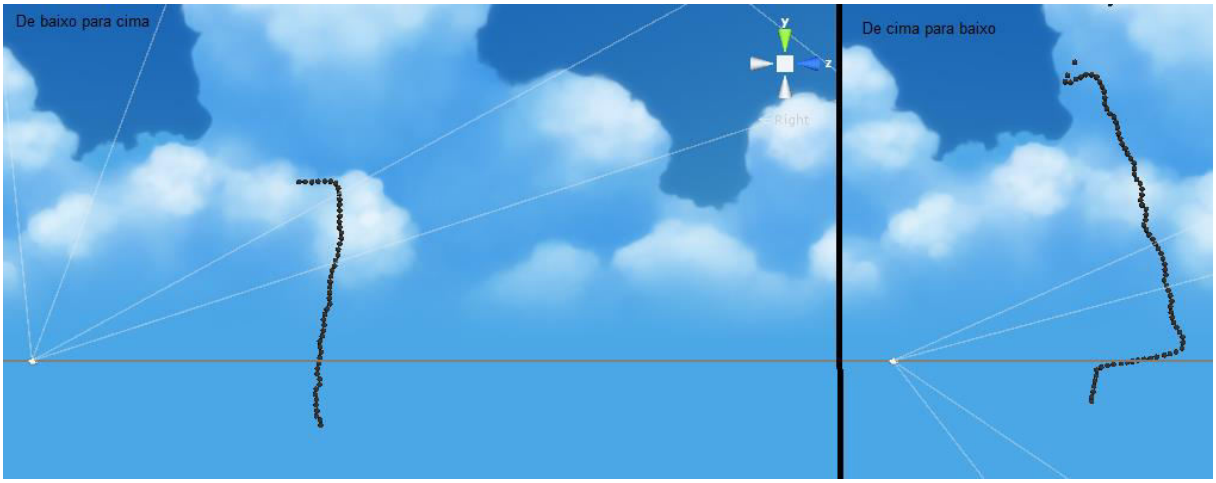


Figura 24 - Mapeamento com Função de Valor Médio
Fonte: Autoria Própria

Utilizando um valor de 200 na função de valor médio, pode-se observar uma diminuição entre o espaçamento dos pontos medidos, formando uma linha muito mais reta e consistente do que as medições anteriores. Infelizmente essa função cria distorções quando utilizado com velocidades maiores nos motores de passo, pois é necessário mais tempo para o valor se estabilizar antes que se possa executar o próximo passo do motor.

4.2.5 **Quinto Teste**

Nesse teste é utilizado uma média de valores dos parâmetros de calibração para diminuir as deformações das medições e manter a velocidade de medição em valores mais altos.

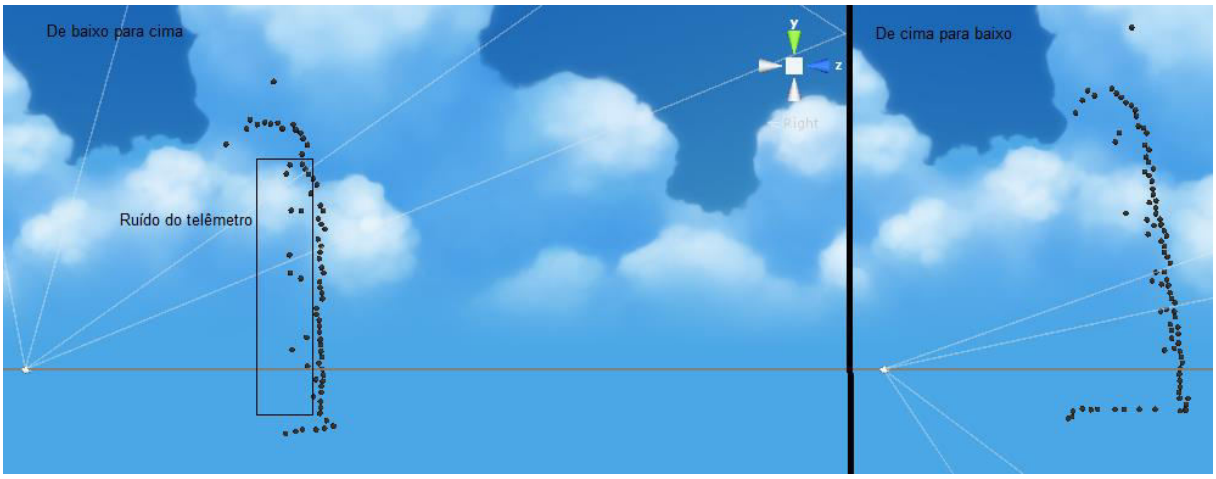


Figura 25 - Mapemaneto com Parâmetros Balanceados
Fonte: Autoria Própria

Utilizando um *delay* de 10, coeficiente α em 0.8 e utilizando um valor de 5 na função de valor médio obtêm-se a figura 25, o que mantém um formato mais próximo do real, com um pouco de atraso e ruído no sensor, mas o suficiente para se poder identificar a presença de um objeto ou obstáculo.

4.2.6 **Sexto Teste**

Esse teste visa o mapeamento de uma peça em 3D para demonstrar a capacidade do sistema de recriar formatos mais complexos. A figura 26 mostra a peça a ser medida.

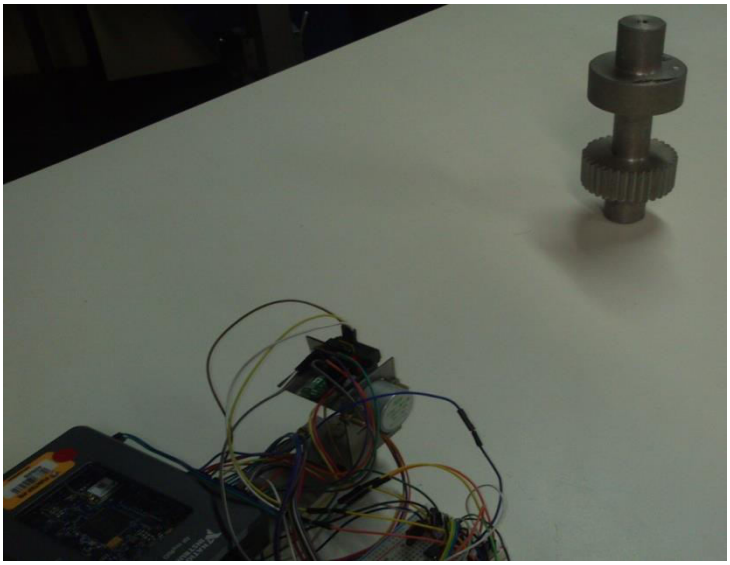


Figura 26 - Mapeamento de Peça Cilíndrica

Fonte: Autoria Própria

Esse cilindro foi medido a 30cm de distância e em dois modos, um em velocidade baixa para maior fidelidade (demorando por volta de 10 minutos) e o outro modo com uma velocidade moderada (demorando por volta de 2 minutos). O sistema faz uma varredura da área, fazendo linhas verticais e aos poucos girando o sensor na horizontal.

O resultado do primeiro modo pode ser visto abaixo:

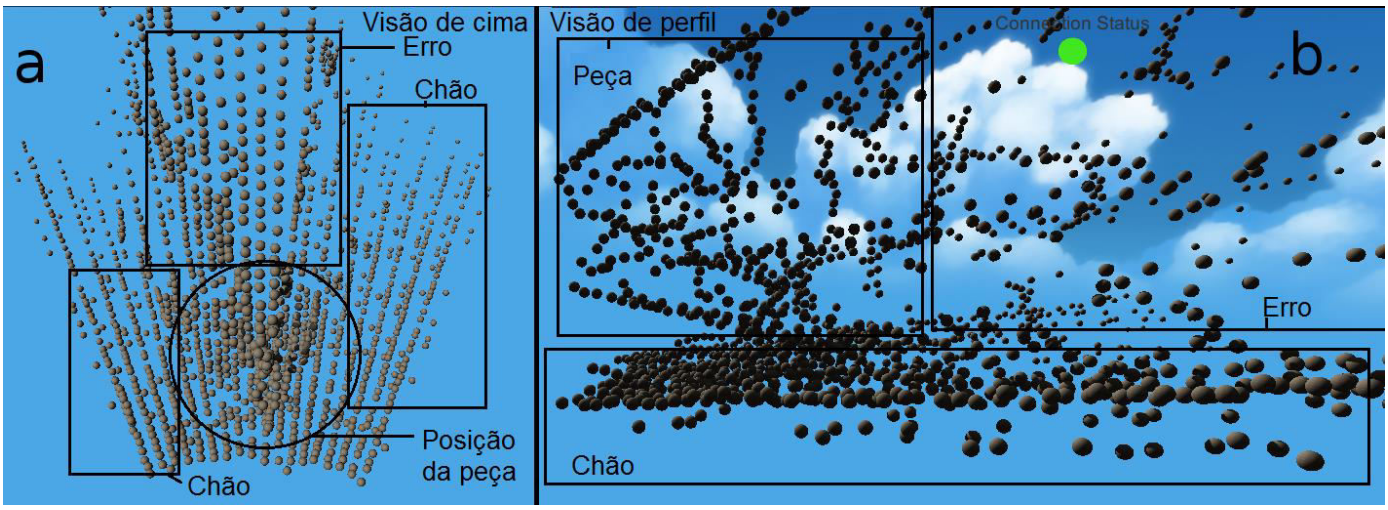


Figura 27 - Mapeamento do Cilindro em Velocidade Baixa

Fonte: Autoria Própria

Na figura 27a, a câmera está na olhando de cima para baixo em direção a peça, na figura 27b a câmera está na posição de perfil. Por causa da perspectiva da câmera, quanto maior o tamanho da esfera, mais perto da câmera essa esfera está.

Nesse teste pode-se observar que o sistema consegue detectar uma peça, mas ela está deformada, sendo impossível de discernir o formato da peça.

4.2.7 **Sétimo Teste**

No segundo modo (velocidade moderada), não há muita diferença se comparado ao teste anterior, possivelmente graças ao erro do telêmetro ao passar por bordas, a figura fica deformada. A diferença mais visível é que as esferas da posição do chão estão mais espaçadas e mais irregulares ao se olhar de perfil.

A figura 28 demonstra o resultado do teste, utilizando as mesmas posições de câmera.

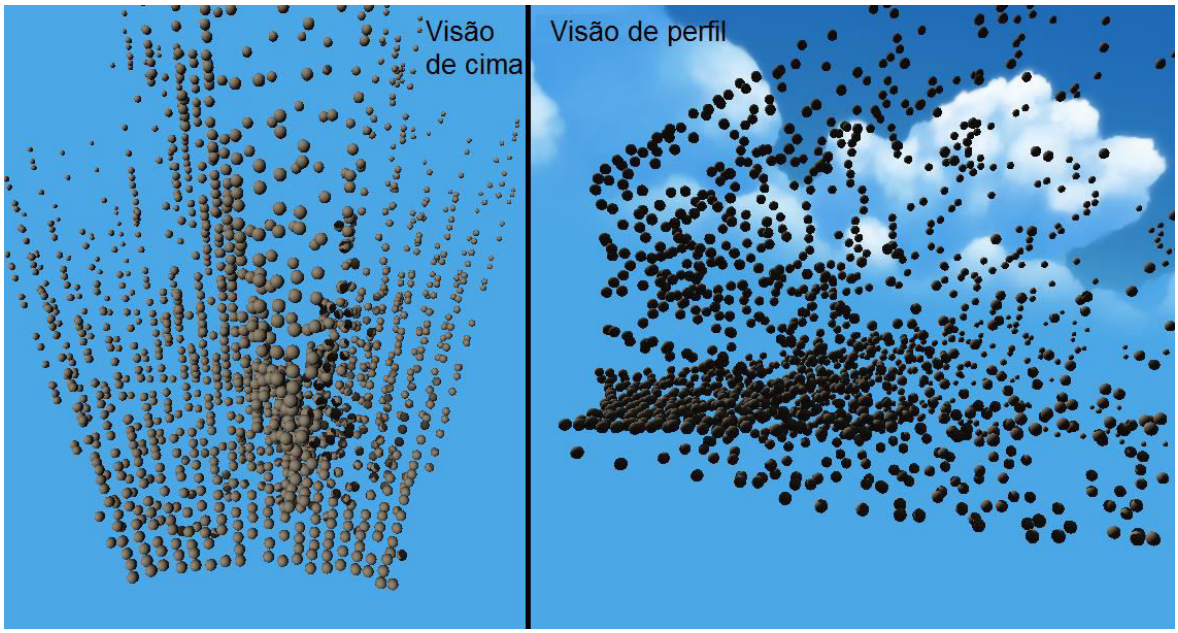


Figura 28 - Mapeamento do Cilindro em Velocidade Alta

Fonte: Autoria Própria

4.2.8 **Oitavo Teste**

O próximo teste utiliza uma carcaça de bomba, posicionada como demonstra a figura 29.

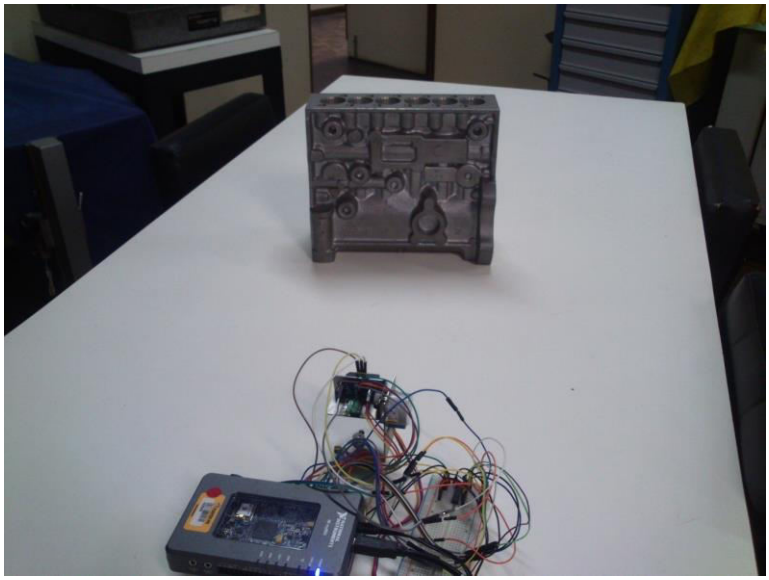


Figura 29 - Mapeamento de Bloco de Motor

Fonte: Autoria Própria

Nesse teste pode-se observar mais claramente as deformações criadas pelo erro quando o sensor encontra uma borda. Utilizando os mesmos parâmetros que o os dois últimos testes, esse teste será feito lentamente enquanto o próximo será feito em velocidade alta.

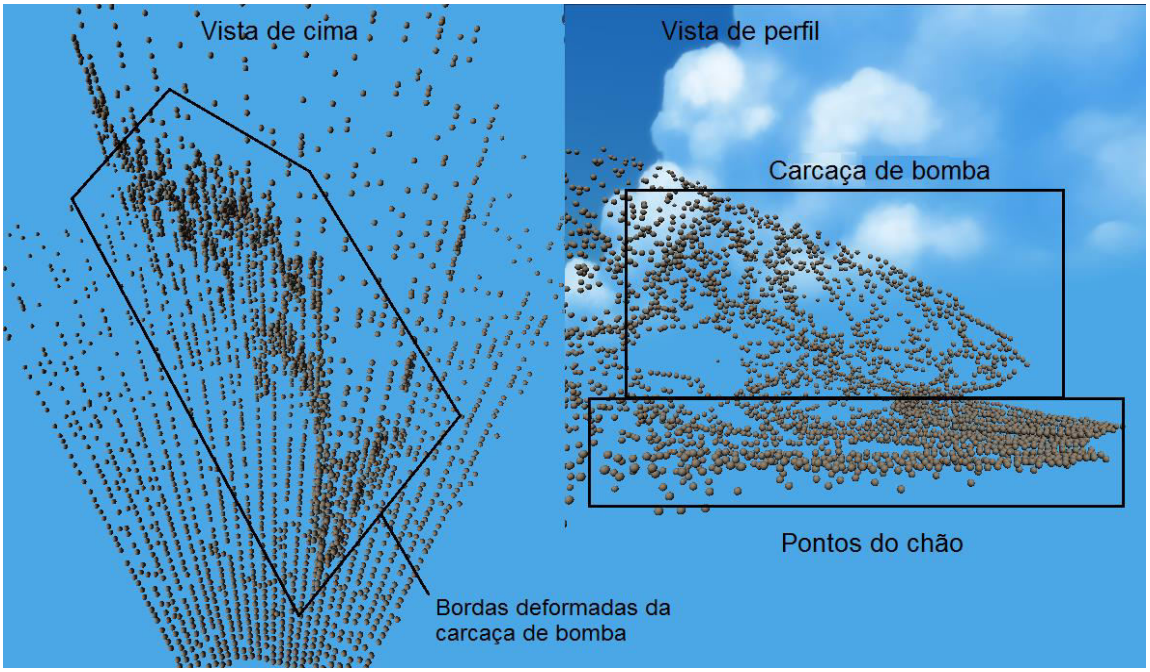


Figura 30 - Bloco de Motor Mapeado Lentamente

Fonte: Autoria Própria

A figura 30 mostra o resultado do mapeamento, a esquerda está uma visão de cima da peça, a direita está uma visão de perfil a esquerda da peça. Essa figura mostra como a peça, que está com a face perpendicular ao sensor, aparenta estar em diagonal. Isso se deve ao erro de quando o sensor encontra bordas, como o bloco de motor possui muitos detalhes o erro fica exacerbado.

4.2.9 **Nono Teste**

Nesse teste foi utilizado uma velocidade de mapeamento maior.

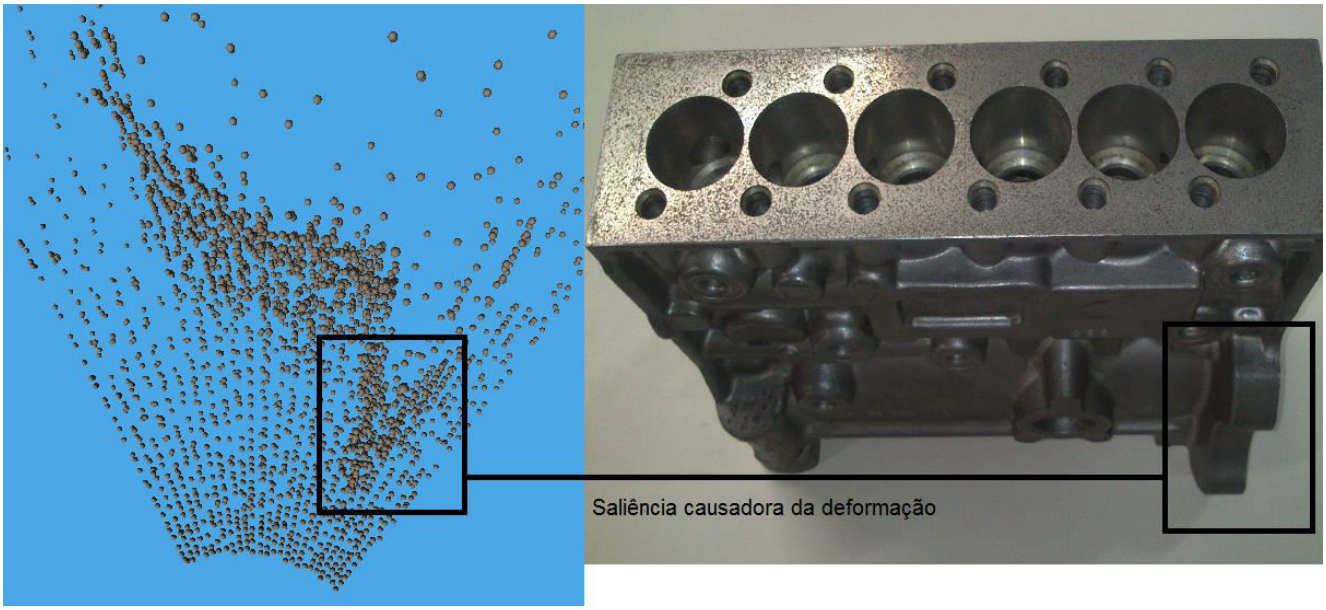


Figura 31 - Bloco de Motor Mapeado Rapidamente

Fonte: Autoria Própria

Na figura 31 pode-se observar como a maior saliência no bloco de motor foi a fonte de maior deformação no mapeamento da peça. Como o erro das bordas cria deformações relativamente grandes ao se girar o sensor em poucos graus, utilizando um valor baixo da função de valor médio (valor 5) do sensor telêmetro e aumentando a velocidade acaba diminuindo os pontos medidos, resultando em uma forma mais generalizada da peça sendo medida. Assim, as mudanças drásticas nos valores são minimizadas, pois o sistema não tem tempo suficiente para detectar as deformações e os erros das bordas em detalhe, resultando em uma peça com menores deformações.

5 CONSIDERAÇÔES FINAIS

O MyMapper está pronto para uso, com a integração dos sensores finalizada e todas as opções consideradas relevantes já adicionadas. Uma utilização mais prática do MyMapper irá requer algumas mudanças, como a soldagem dos componentes e a possível integração com algum outro software requerido, mas como o MyMapper já possui um protocolo de comunicação com o *Unity* programado, pode ser que não haja a necessidade de grandes mudanças no código ou uma grande dificuldade de nova programação.

O método de varredura funciona como desejado, mas possui uma baixa velocidade graças aos limites dos motores de passo. O sensor telêmetro é muito sensível ao ruído, então se motores mais rápidos forem utilizados é preciso garantir um baixo ruído ou então a imprecisão do sistema pode aumentar.

O sistema consegue mapear e visualizar o ambiente em 3D, mas as ferramentas de visualização são limitadas pois é difícil de prever quais ferramentas serão úteis para usos no futuro. Uma melhoria que poderia ser feita quanto a visualização seria modificar as cores das esferas para refletir a distância entre elas e o sensor, o que facilitaria a visualização do formato da peça.

O código pode ser observado com mais detalhes no manual em inglês, nesse manual pode se observar as funções e a disposição modular do código, onde esses módulos são grupos de funções conectados a partir de variáveis de controle, o que facilita futuras modificações.

O MyMapper consegue servir como sistema de visão para robôs mais estáticos pois ainda há dificuldades de se implementar com precisão um sistema onde se possa girá-lo em todos os seus eixos. Também o sistema não possui um algoritmo para se detectar translação, mas com a presença de dois acelerômetros (um na torre de sensoriamento e um presente embutido no myRIO) há uma possibilidade de implementação de tal algoritmo.

O algoritmo de cálculo de ângulos possui capacidade de melhorias, podendo ser realizado um estudo do comportamento do acelerômetro enquanto inclinado em mais de um de seus eixos e também a possibilidade de implementação de uma bússola eletrônica, o que ajudaria no cálculo de giros no ângulo Y, habilitando a utilização do filtro complementar nesse eixo.

O formato das peças sendo mapeadas sofrem deformações de várias partes do sistema, desde o método que os sensores são calibrados até por causa do próprio funcionamento do sensor infravermelho, que cria deformações ao encontrar

bordas e também pode sofrer interferência quando os dois motores são acionados. Os motores funcionam de modo *latch*, sendo que mesmo quando parados uma das quatro saídas se mantém acionadas, portanto, para resolver esse problema as saídas precisam ser acionadas somente quando o motor está em movimento.

No fim, o MyMapper se tornou um sistema de mapeamento muito sensível a calibração e interferências, sendo mais útil em ambientes estáticos e não recomendável a utilização desse sistema em ambientes onde os obstáculos são móveis e imprevisíveis pois a velocidade dos motores de passo é limitada, criando uma demora no mapeamento do ambiente.

6 REFERÊNCIAS

EPPERSON, James F. On the Runge example. **Amer. Math. Monthly**, vol. 94, 4 ed., p. 329-341, 1987. Disponível em: https://www.maa.org/sites/default/files/pdf/upload_library/22/Ford/Epperson329-341.pdf. Acesso, 22 Jan. 2016.

FANTASY SKYBOX FREE. Disponível em: https://www.assetstore.unity3d.com/en/#!/content/18353. Acesso, 21 Jan. 2015.

GAO, Chao et al. Autonomous docking of a smart wheelchair for the automated transport and retrieval system (ATRS). **Journal of Field Robotics**, vol. 25, 4 ed., p. 203, 2008. Disponível em: https://www.researchgate.net/profile/John_Spletzer/publication/220647954_Autonomous_docking_of_a_smart_wheelchair_for_the_Automated_Transport_and_Retrieval_System_(ATRS)/links/559df96708ae04e3650909e2.pdf. Acesso, 19 Dez. 2014.

HIGGINS, W.T. A Comparison of Complementary and Kalman Filtering. **IEEE Transactions on Aerospace Electronic Systems**. vol. 11, 3ed., pg.321-325, 1975. Disponível em: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4101411. Acesso, 16 Mar. 2015.

KALMAN, R. E. A New Approach to Linear Filtering and Prediction Problems. **Journal of basic Engineering**. vol. 82, 2ed., pg.35-4, 1959. Disponível em: http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402. Acesso, 7 Out 2015;

KERL, Christian; STURM, Jurgen; CREMERS, Daniel. Dense Visual SLAM for RGB-D Cameras. **Intelligent Robots and Systemas (IROS), 2013 IEEE/RSJ International Conference**. pg. 2100-2016, 2013. Disponível em: http://vision.in.tum.de/_media/spezial/bib/kerl13iros.pdf. Acesso, 20 Ago 2015.

PEDROSA, Diogo Pinheiro Fernandes. **Ajuste de Curvas.** Caicó: Universidade Federal do Rio Grande do Norte, 2014. Disponível em: http://www.dca.ufrn.br/~diogo/FTP/dca0304/ajustedecurvas.pdf. Acesso, 20 Ago 2015.

STMICROELETRONICS. **MEMS motion sensor: three-axis digital output gyroscope**. 2013, v. 2. Disponível em: https://www.pololu.com/file/0J563/L3GD20.pdf. Acesso, 7 Set. 2015.

THRUN, Sebastian et al. Robotic Mapping: A Survey. **Exploring artificial intelligence in the new millenium**. vol. 1, pg. 1-35, 2002. Disponível em: http://robots.stanford.edu/papers/thrun.mapping-tr.html. Acesso, 16 Mar 2015.

WALTER, Patrick L. The history of the accelerometer. **Sound and vibration**, vol. 31, 3 ed., pg. 16-24, 2007. Disponível em: http://www.sandv.com/downloads/0701walt.pdf. Acesso, 23 Jan 2016.

WEISS, Ulrich; BIBER, Peter Plant detection and mapping for agricultural robots using a 3D LIDAR sensor. **Robotics and Autonomous Systems**, vol. 59, 5 ed., pg. 265–273, 2011. Disponível em: http://www.sciencedirect.com/science/article/pii/S0921889011000315. Acesso, 18 Dez. 2014.

YAZDI, Navid et al. Micromachined Inertial Sensors. **Proceedings of the IEEE**, vol. 86, 8 ed., pg. 1640-1659, 1988. Disponível em: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=704269. Acesso, 09 Nov. 2015.

7 ANEXOS

MyMapper Project User Manual
Created by Douglas Diogo Franczak

Table of Contents

8 Components

8.1 STMicroeletronics Gyroscope L3GD20.

This gyroscope measures the rotation rate in °/s on three axis. It has a maximum output of + or - 2000 degrees per second (dps) with selectable measurement rates. The L3GD20 uses either the SPI or the I²C communications protocols to send and retrieve data from the myRIO, I²C is active by default and it's the one used on this project.

The communications is done using the I²C Function Block available on myRIO, the address of the gyroscope is 1101011 in binary or 107 in base 10. It works by firstly sending an 8-bit binary of the register address desired to access, if it's a writable address a second byte needs to be sent to change the configurations, read only addresses such as reading the output of the sensor don't need the secondary byte. Read-only addresses respond with a byte depending on the address, configuration addresses respond by sending back the second byte to signify that the configuration has changed. Each address has its own interpretation of the second byte, so reading the datasheet for this sensor is necessary, it's also necessary to pay attention to which bytes are being sent as some addresses bits should not be changed.

8.2 Freescale Accelerometer MMA7361L.

This accelerometer is a very easy to use sensor. Each axis data is sent directly from each of its pins, but calibration is necessary for it to be able to be used with the project. This sensor requires the "Sleep" pin to be pulled high for it to work. The pin "G-Select" switches between 1.5g and 6g sensing, the default setting is 1.5g, which gives more sensibility to the sensor.

The operating voltage is between 2.2v and 3.6v, with the center of each measurement pin being VDD/2. To calibrate this sensor the value of the voltage is reduced by an offset number to center it and then multiplied by another number so it reaches the -1 and plus +1 maximum values.

8.3 Sharp Infrared Rangefinder GP2Y0A21YK.

This sensor is a very basic infrared rangefinder, which detects the intensity of the IR light being reflected on the object it's being pointed at. The signal returned is an polynomial curve with an inverted proportion of distance vs. return signal. To convert this signal into an usable one its used a curve fitting algorithm that creates a polynomial equation based on the real value curve table created for calibration. With this polynomial equation the current voltage value being measured is used to substitute the variable of the equation and then solve it, the resulting value is the distance between the object and the sensor in measurement unit used in the real value curve table.

8.4 Kiatronics Stepper Motors 28BYJ-48.

These stepper motors are used to rotate the sensors around to scan the area. They have four phases and are slightly slow for this project, but they have a good precision and an easy way to control them.

The steppers are controlled with a single-step method, this method being activating each phase individually and in sequence, the single-step method has been chosen for its ease to program and control. The stepper use a input voltage pin different than the sensors because of it creates a lot of noise in the system, which affects the Sharp IR sensor the most.

8.5 Toshiba ULN2803A.

The ULN2803 is a Darlington transistor array that is used as a driver for the stepper motors because each of its phases requires 5v to function, but the myRIO does not have enough analog outputs for it and its digital outputs only supplies 3.3v. The ULN serves as a amplifier of current by connecting the myRIO digital outputs at the base and the stepper inputs at the collector and the 5v supply at the Common Node.

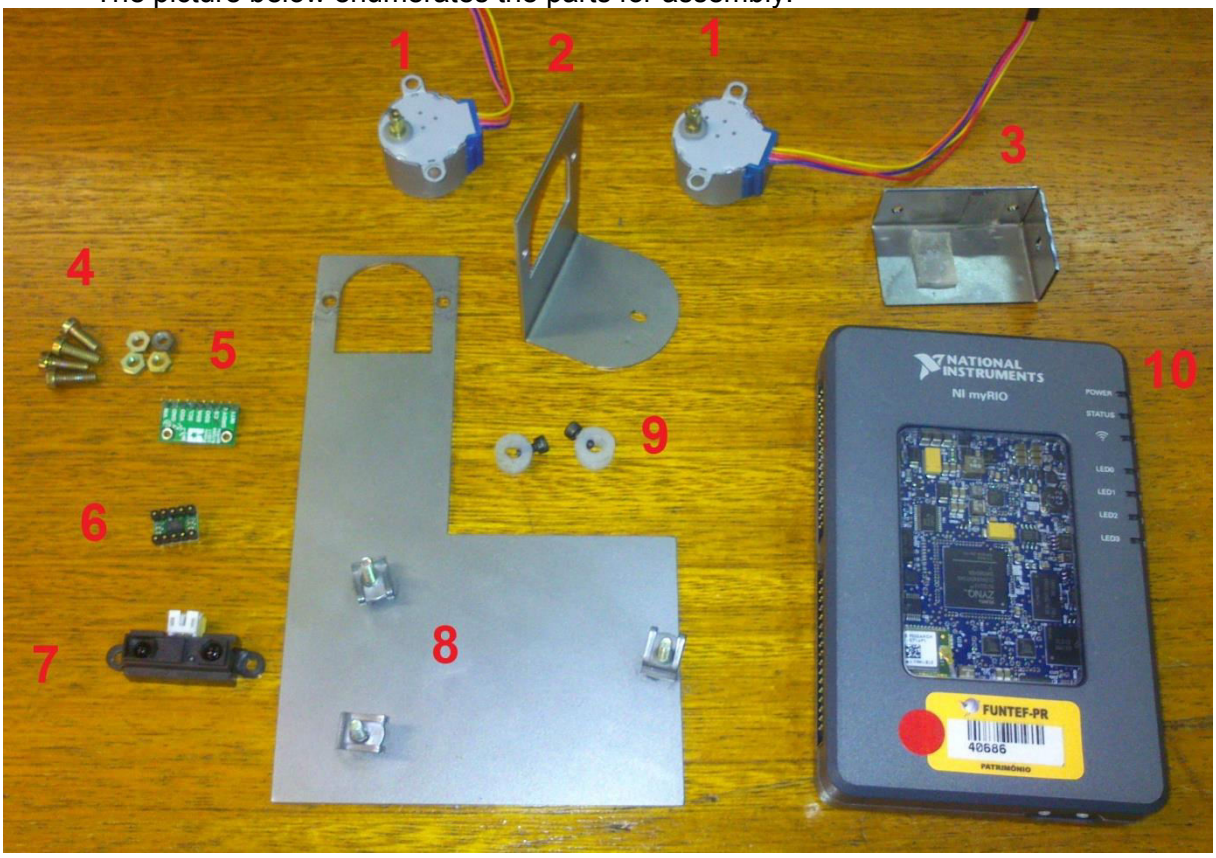
8.6 National Instruments myRIO.

This is an embedded hardware used normally as a didactic device for student projects, it has multiple analog and digital ports with a variety of functions such as I²C, SPI, UART, PWM, Encoding and FPGA. FPGA being its most advanced tool enabling real parallel processing, the FPGA works by modifying the way the logic blocks are "wired" allowing for specific configuration of inputs and outputs, although it does not have all of the more complex functions of a normal application, it allows for faster and more reliable control of data.

The myRIO functions as the data collector of the project, while the host computer calculates the data, the myRIO has been programmed to only execute functions and reduce the amount of data processing so it can increase its precision and speed on reading the sensor data.

9 Assembly

The picture below enumerates the parts for assembly:

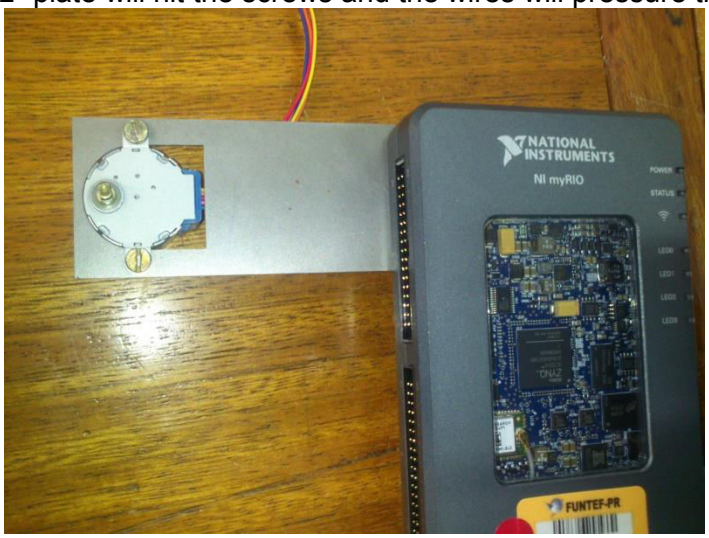


1. Stepper Motors
2. "L" Plate.
3. "Sensor tower" plate.
4. Stepper screws and nuts.
5. Gyroscope.
6. Accelerometer.
7. IR Rangefinder.
8. "P" Plate with the screws on it.
9. Axis plug.
10. MyRIO.

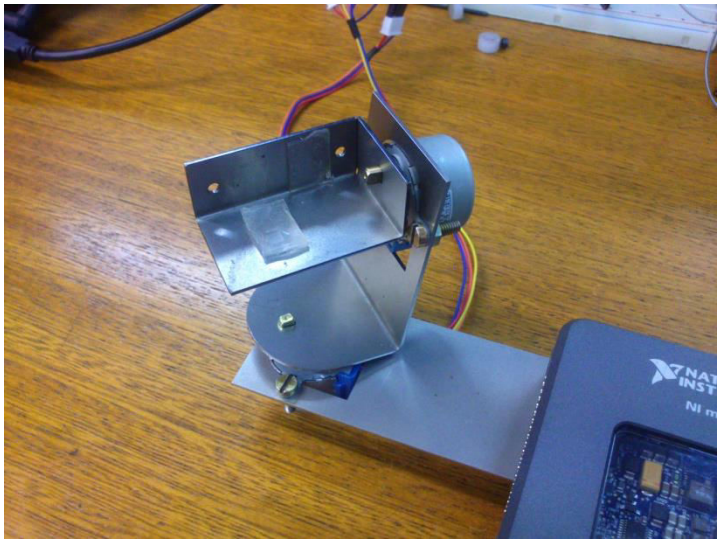
1. To start three screws should be put on the three openings making a triangle formation.

2. After that the "P" plate should be mounted and screwed. Pay attention to the side of the plate, it should be pointing to the side of the myRIO with the A and B group of pins.

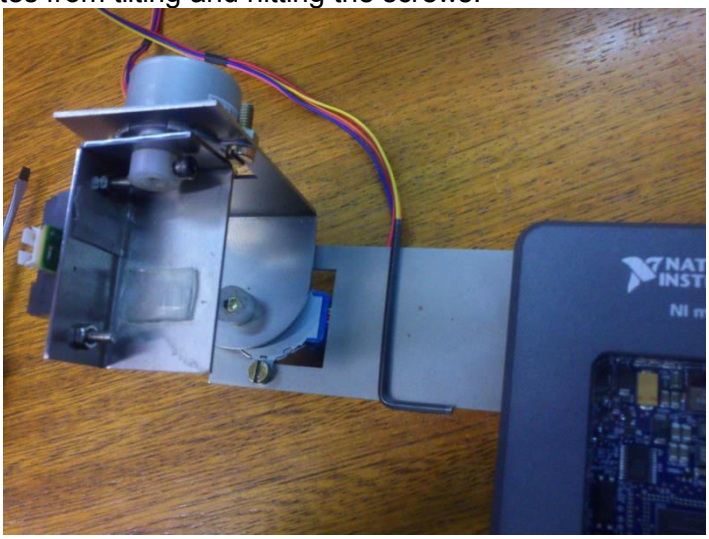
3. The first stepper should be screwed on the opening of the "P" plate. The stepper screwing hole needs to be above the "P" plate and the wiring needs to go below it, otherwise the "L" plate will hit the screws and the wires will pressure the stepper to bend.



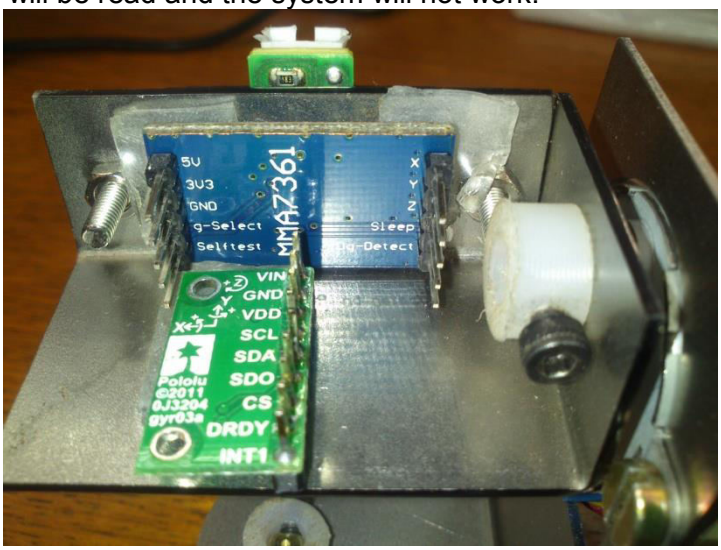
4. Do the same on the "L" plate. Fit the L plate on the first stepper and the sensor tower on the second stepper.



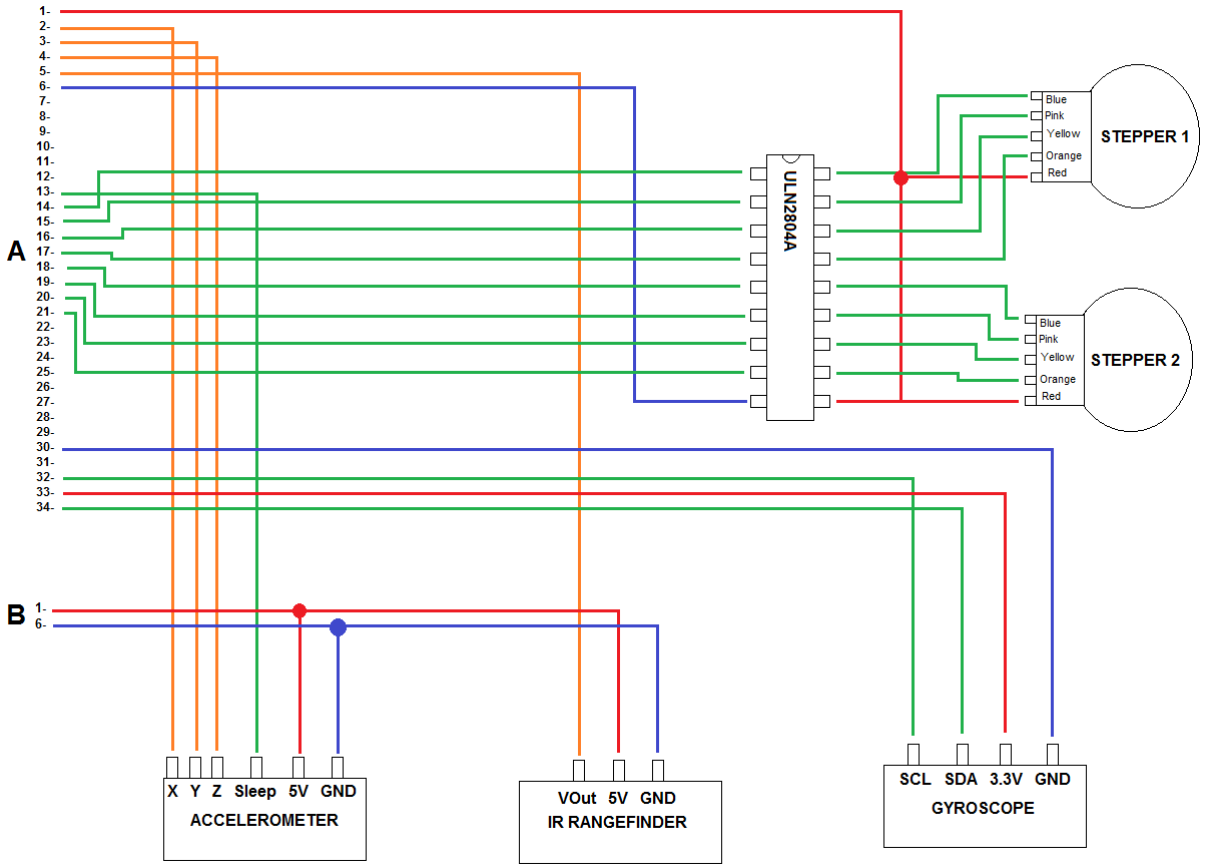
5. Screw the IR Rangefinder on the sensor tower and then put the plugs on each axis to prevent the plates from tilting and hitting the screws.



6. Glue the gyroscope and accelerometer on the same orientation as the picture. It is important that the accelerometer and gyroscope are in the correct positions otherwise the wrong axis will be read and the system will not work.



7. After mounting the system the wiring can be done. The electrical schematic is shown below:



Labels:
Red: Power wire.
Blue: Ground wire.
Green: Digital wire.
Orange: Analog wire.

10 Startup

If the myRIO has not been deployed it is necessary to open the source code project on LabVIEW, it is named "TCC MyMapper.lvproj", under NI-myRIO and under build specifications the file "MyMapper myRIO main" should be right-clicked and the option "Run As Startup" should be chosen and then rebooting the myRIO when asked. With this option every time the myRIO is powered it will automatically run the MyMapper main VI.

It is required that on the LabVIEW folder the files "Options.txt" and "Default Options.txt" should be on the same folder as the LabVIEW .exe, on Unity it is required the files "Options.txt", "Default Options.txt" and "Auto Options.txt" under the folder "start_Data/Resources/Options", if these files are missing or incorrectly formatted the system will not work as they contain essential information.

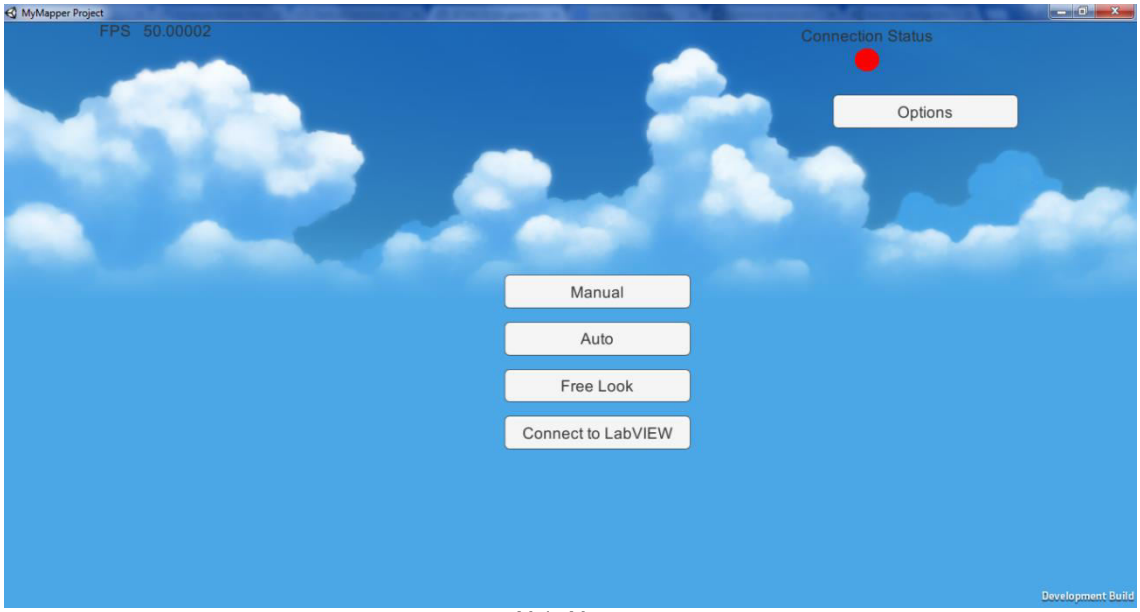
Once the myRIO is running (it has no interface, unless using the source code) the labVIEW and the Unity.exe can be opened, the order of execution of the three programs is not important, the Unity and LabVIEW will automatically connect to each other and the LabVIEW will automatically pull data from the myRIO on default configuration.

11 Interface

11.1 Unity

11.1.1 Operating Modes

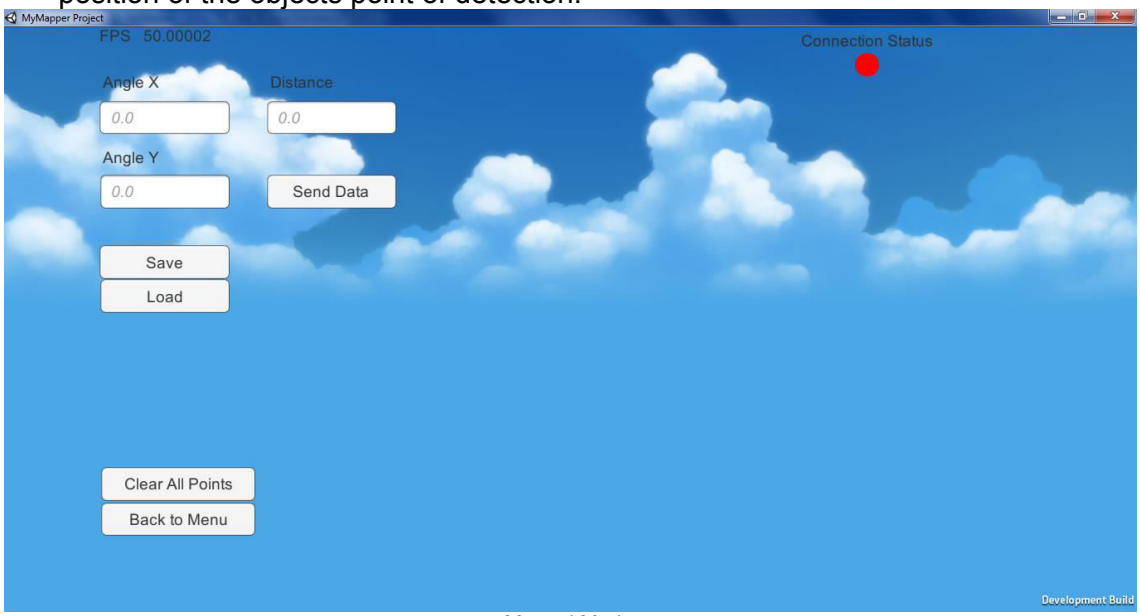
The Unity initial interface is show on the picture bellow.



Main Menu

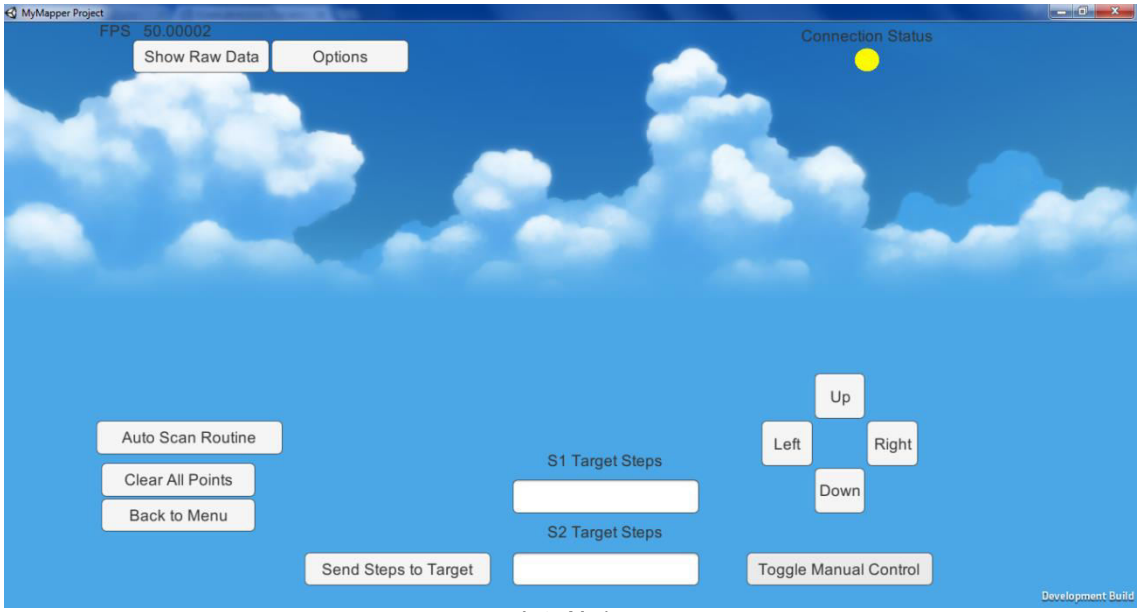
The system has a few operating modes:

- Manual Button: switches the interface to the manual input mode. In this mode object and orientation input is entered manually, it's also possible to used the keyboard to rotate and more around the 3D ambient. It's possible also to load and save the position of the objects point of detection.



Manual Mode

- Auto Button: switches the interface to the automatic input mode. In this mode it is used the sensors to acquire the orientation, rotation and object distance. This mode also allows to load and save the points but does not allow for manual rotation and movement on the 3D ambient.

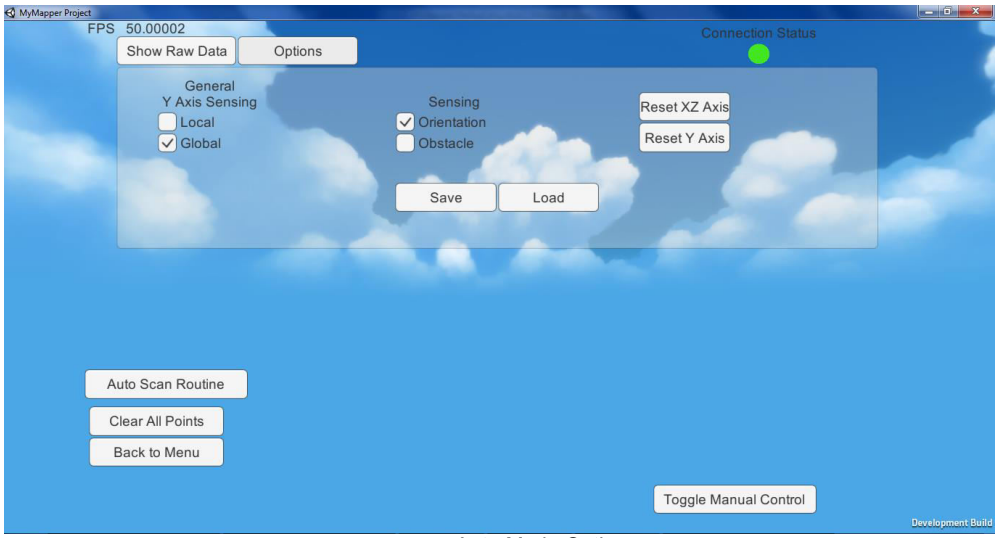


Auto Mode

This mode has the following options:

- Auto Scan Routine: Pressing this button will move the sensors in a programmed pattern, the starting direction the IR sensor is pointed at will be the center of the scanned area. This mode uses the *S1 Vertical Limit* and the *S2 Horizontal Limit* to calculate the scan area. During startup it goes first to the bottom-right of the scan area and then scan to the maximum point on the left and then comes to the right and repeats it all again.
- Manual Control: The rotation of the stepper motors can be controlled manually by pressing the buttons or by sending the number of steps desired to be executed by each stepper to LabVIEW.

Under the Options tab it can be seen:

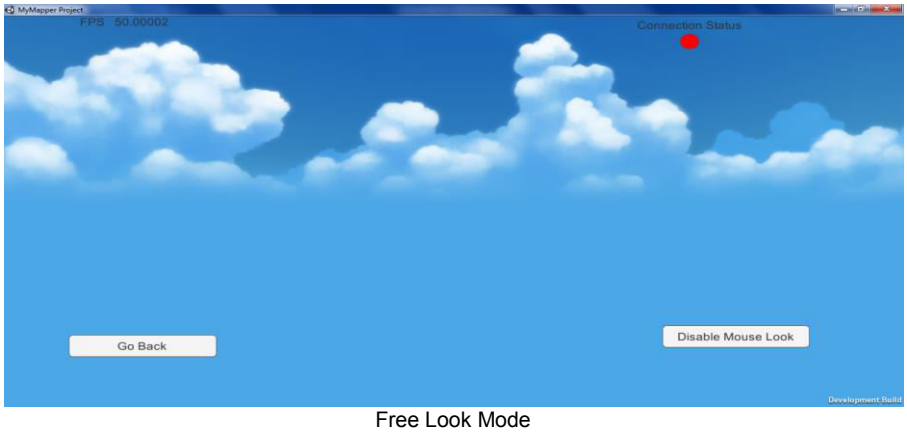


Auto Mode Options

- General Y Axis Sensing: The Y Axis, also known as Yaw, can be determined by utilizing the counting of the steps the second motor has done or by utilizing the gyroscope. Counting the steps is more precise at speed 3 or slower, but it cannot detect when the platform is rotate on the Y Axis. Using the gyroscope allows for the

detection of the rotation on the Y axis but it also introduces incremental errors on the measurement and it's advised to regularly reset the Y Axis.

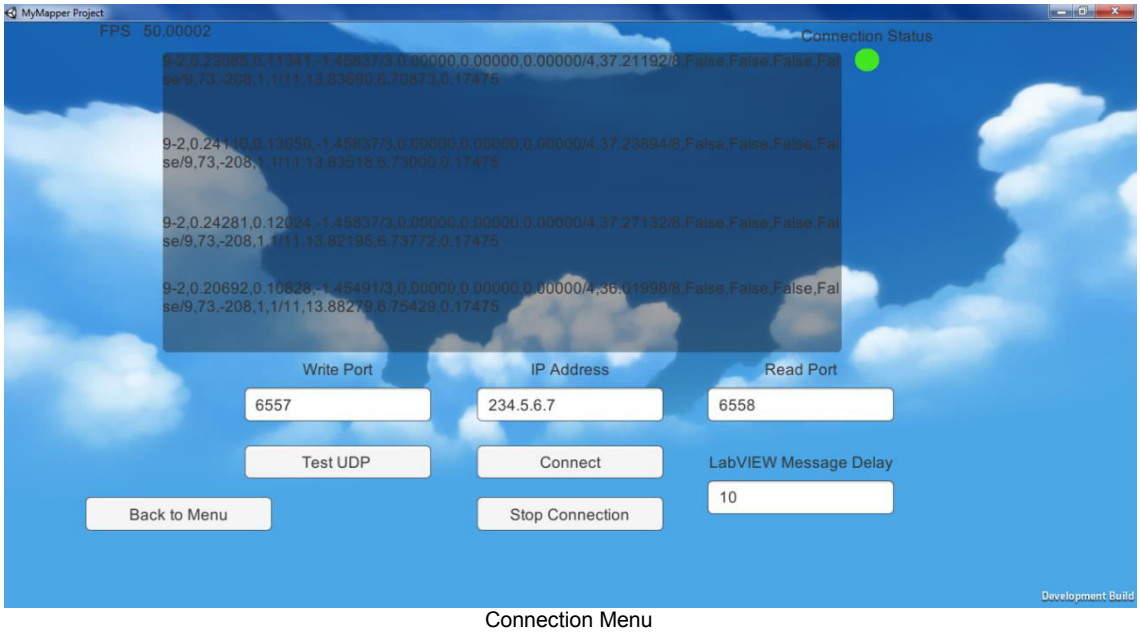
- Sensing: Orientation sensing activates the rotation of the Unity camera based on sensor data. Obstacle sensing activates the sensing of obstacles using the IR rangefinder sensor.
- Clear All Points Button: Clears all the obstacles from the screen.
- Free Look Button: switches the interface to Free Look mode, optimal to navigate the 3D ambient but it cannot load or save the obstacle detection data.



Free Look Mode

11.1.2 **Connection**

The "Connect to LabVIEW Button" Opens the interface to connect manually to labVIEW.

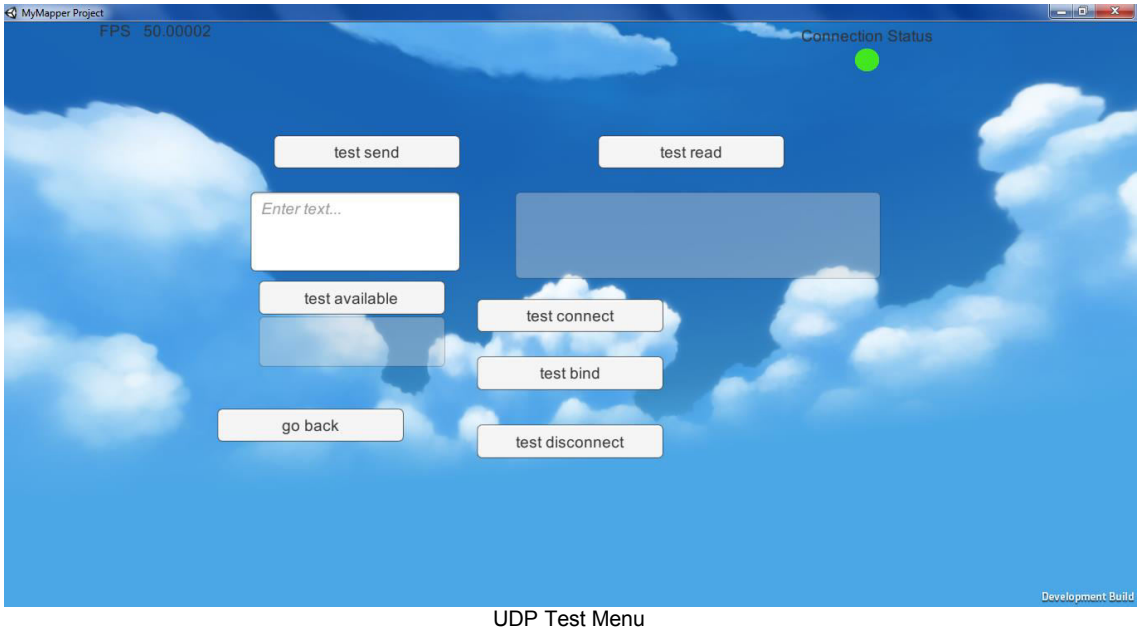


Connection Menu

This interface also shows the current strings being read from the UDP connection.

- Write Port: The port that Unity writes data, it must be the same port LabVIEW reads data.
- Read Port: The port that Unity reads data coming from LabVIEW, it must be the same port LabVIEW writes data to.
- LabVIEW Message Delay: The amount of loop iterations labVIEW has to wait to write data for Unity to read, values too low will overload the buffer and data will be lost, values too high will lose data and make the update of sensor data too slow.

- Test UDP: Opens the interface to test the UDP connection. This button automatically stops the auto-reading of messages coming from LabVIEW, but it doesn't stop the auto-sending of messages, so make sure to disable the option to auto-send sensor data on LabVIEW otherwise the message *buffer* will quickly be filled.



UDP Test Menu

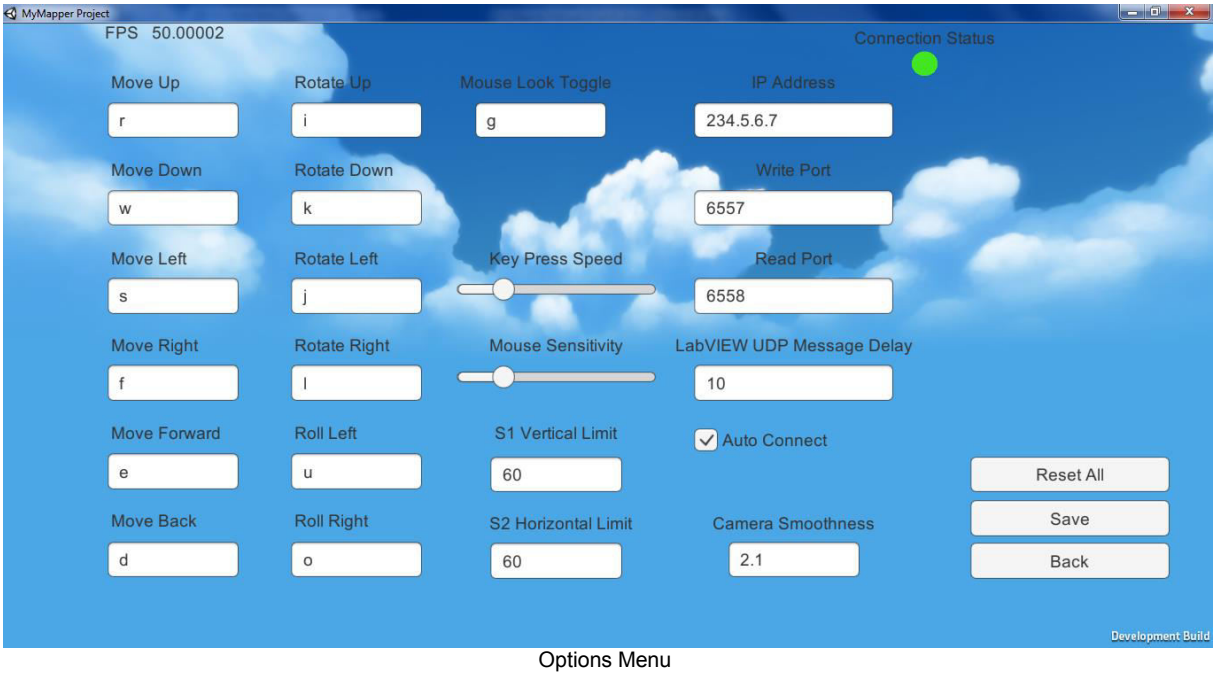
- Test Read: Reads data if available, if no data available a small stop will happen. The data will appear on the transparent rectangle below the button.
- Test Send: Sends the string written on the input box below the button.
- Test Available: Returns the number of bytes available to read, the number appears on the transparent rectangle under the button.
- Test Connect: Assigns all the variables required for connection but does not send the standard connection string to LabVIEW.
- Test Bind: binds the IP and port, required to properly communicate, cannot be undone unless the program is restarted.
- Test Disconnect: drops the connection but cannot reconnect until the program is restarted.

Connection Status: This small circle shows the connection status to LabVIEW.

- Red: Not connected to LabVIEW.
- Yellow: Connecting to LabVIEW.
- Green: Connected to LabVIEW. Due to the way the UDP connection works, if the LabVIEW side stops the Unity side won't be able to detect it.

11.1.3 **Options**

The options interface of Unity can be seen below:



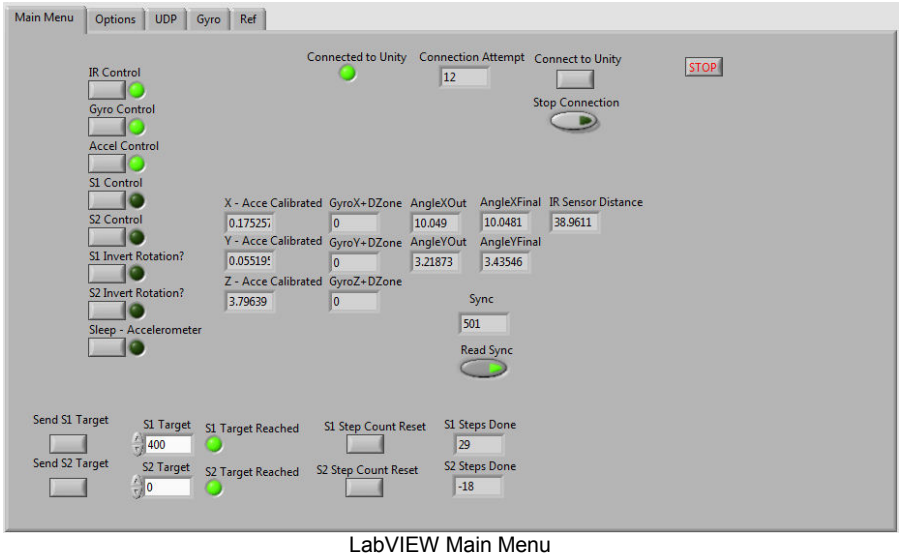
Options Menu

- Move Up, Down, Left, Right, Forward, Back: These buttons are the controls used in Free Look mode to move the camera around.
- Rotate Up, Down, Left, Right, Forward, Back: These buttons control the camera rotation when the mouse look is disabled.
- Mouse Look Toggle: Toggles the rotation of the camera being control by either the mouse or the keyboard.
- Key Press Speed: Controls the speed by which the camera move while pressing the move or rotation buttons.
- Mouse Sensitivity: Controls the speed of the camera rotation while the mouse look mode is activated.
- S1 Vertical Limit, S2 Horizontal Limit: controls the maximum rotation on each axis during the automatic scan routine, the steppers wil always rotate half of this value to each side of the starting position. This value is always in degrees. This value uses the Stepper Conversion option on LabVIEW to convert the steps done by the motors to degrees.
- Multicast IP: an IP is necessary even though it's not used to connect to the internet, this IP needs to be the same one specified on the options at the LabVIEW interface.
- Timeout: the time to wait for data when there's no data while trying to read data from the *buffer*, the system will never use the timeout since it's programmed to only read data when there's data available, so it's only relevant when testing the communication.
- Read Port: the port from which data should be read, it should be the same as the Write Port in the options of LabVIEW Interface.
- Write Port: the port from which data will be written to, it should be the same as the Write Port in the options of LabVIEW Interface.
- LabVIEW UDP Message Delay: this option controls the amount of loop iterations to wait between each message sent by LabVIEW, if the value is too low the Unity *buffer* will fill up, if it's too high, the system will lose precision. The Unity automatically increases this value if the buffer has more than 2000 bytes waiting to be read.
- Auto-Connect: Controls if the Unity will start the connection routine after start up.
- Camera Smoothness: Control the camera velocity acceleration. Lower values increase the camera jitter because of the noise in the sensors, higher values decrease jitter but can create errors on the obstacle detection positioning.

11.2 LabVIEW

11.2.1 **Main Menu**

The LabVIEW initial interface is shown on the picture below.



LabVIEW Main Menu

The Labview interface is divided by tabs.

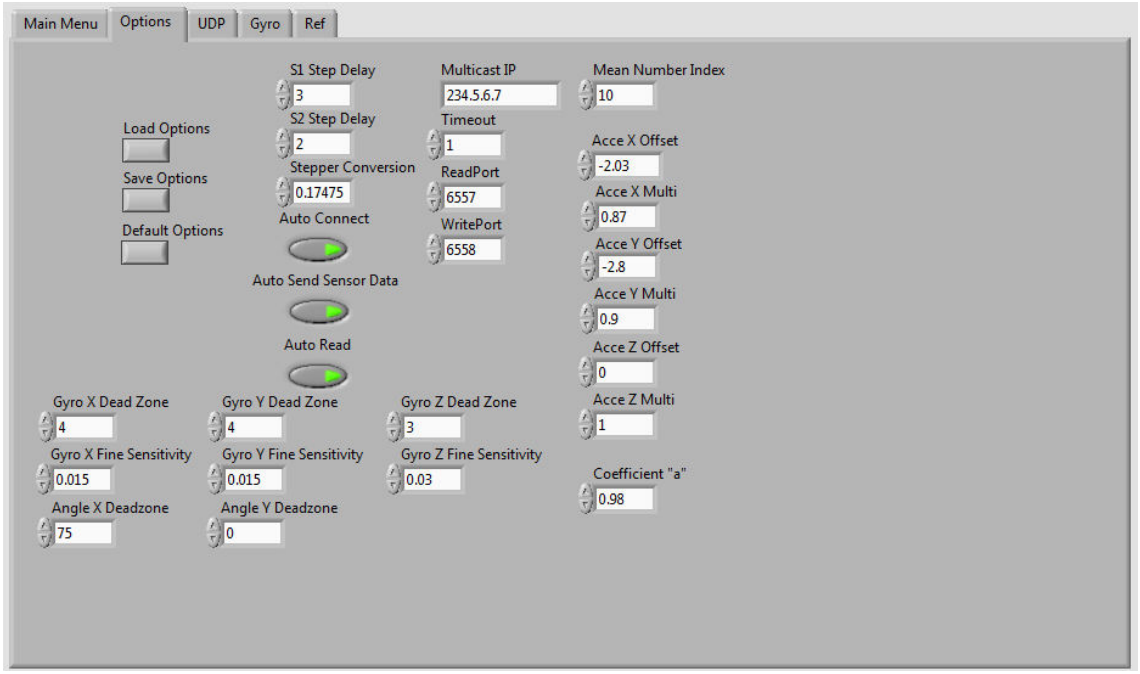
- Main Menu: Contains the main controls for the sensors and steppers, also the indicators for the data read by the myRIO.
- Options: Contains all the options required to calibrate the sensors and the communication options to connect to Unity.
- UDP: Used to test the UDP communication between Unity and LabVIEW.
- Gyro: Used to configure and control the gyroscope through I²C. To use this option the Gyro Control on the Main Menu should be disabled.
- Ref: Contains the indicators to all the variables used by the system. These references are used so the variables can be called by using the Local Variable or the Property Node block function.

On the main interface you can:

- Control which sensor is activated by pressing the "(Sensor) Control" buttons, disabling the sensor will also disable the sensor data being sent to Unity.
- Control each stepper with the "S# Control" buttons, both steppers will rotate clockwise, to invert rotation the buttons "S# Invert Rotations?" needs to be pressed.
- Send the amount of steps each steppers needs to execute from the "S# Target" fields by pressing the "Send S# Target" buttons. The "S# Target Reached" light shows whether or not the stepper is still moving.
- Reset the memorized number of steps done by each stepper. CAUTION: Resetting this number while the stepper is still moving can confuse the system and make it keep rotation longer than wanted.
- Start or stop the connection routine. During the connection all other functions are disabled, so it's possible to stop it by enabling the "Stop Connection" switch.
- The "Sync" indicator shows the velocity of the loop iterations, it starts at 1 and when reaches 1000 it resets to 1. If the myRIO source code is available, in its front panel there's another Sync indicator, if the myRIO Sync indicator is faster than the Host indicator, then the shared variable system is overloaded.

11.2.2 **Options**

The options tab is shown below:



LabVIEW Options Menu

- Load Options, Save Options, Default Options: these buttons read and modify the files Options.txt and DefaultOptions.txt located at the same folder the Host.exe file is located.
- S# Step Delay: the myRIO executes one step per loop iteration, increasing this number increases the number of loop iterations waited before executing another step.
- Stepper Conversion: this value is used by the Unity to convert the number of steps executed by the motors into a angle value in degrees.
- Auto Connect: This option automatically starts the connection to Unity routine when the Host is started.
- Auto Send Sensor Data: this option automatically sends the sensor data collected by myRIO to Unity. If it's necessary to test the communication disable this option, otherwise the Unity buffer will be filled quickly.
- Auto Read: automatically reads the commands sent by Unity to LabVIEW and executes them. If it's necessary to test the communication disable this option, otherwise it will be impossible to read data sent my Unity manually.
- Multicast IP: an IP is necessary even though it's not used to connect to the internet, this IP needs to be the same one specified on the options at the Unity interface.
- Timeout: the time to wait for data when there's no data while trying to read data from the *buffer*, the system will never use the timeout since it's programmed to only read data when there's data available, so it's only relevant when testing the communication.
- Read Port: the port from which data should be read, it should be the same as the Write Port in the options of Unity Interface.
- Write Port: the port from which data will be written to, it should be the same as the Write Port in the options of Unity Interface.
- Mean Number Index: the number of numbers on the array that gives the mean number of the data coming from the IR sensor. The bigger the number, the more data will be used and the slower the response from the sensor will be, but it increases the stability of the data.
- Acce (X/Y/Z) Offset: option to calibrate the accelerometer, the numbers put on these fields will be added to the current value of each axis. More about calibration on the next section.

- Acce (X/Y/Z) Multi: option to calibrate the accelerometer, the numbers put on these fields will be multiplied to the current value of each axis after being adding the value at the Acce (X/Y/Z) Offset option. More about calibration on the next section
- Coefficient "a": this option is used on the complementary filter used to fuse the accelerometer and gyroscope data from each axis. It should not be necessary to modify this option.
- Gyro (X/Y/Z) Dead Zone: even during rest the gyroscope does not give the value 0 all the time, it usually varies from +3 to -3, so a dead zone ensures that the gyro will not send values if they are lower than the Dead Zone value, this prevents the system from "drifting" because the gyro would otherwise keep sending rotation values even during rest.
- Gyro (X/Y/Z) Fine Sensitivity: the value that multiplies each axis raw value to convert it to degrees per second.
- Angle (X/Y) Dead Zone: A secondary filter and logical conditions are used to calibrate the final angle value and reduce the error, the secondary filter uses these values. The second filter will be explained on the next section.

12 Calibration

12.1 Calibrating the gyroscope

The gyroscope has the most complex calibration settings due to its reliance on timing. The gyroscope only updates its axis outputs at configurable intervals, meaning that the loop speed of the myRIO needs to have a matching frequency with the gyroscope, otherwise if it's too slow information gets lost or if it's too high the same information can be read multiple times depending on the functioning mode.

Every time the myRIO starts up the configuration bytes are sent to the gyroscope to ensure there is no problems with timing. The startup configuration is as follows:

- Address: 20 (in hex). Configuration: 10001111 (in binary). This configuration sets the Output Data Rate to 380Hz, the cut-off frequency to 20, sets the gyroscope on normal mode and enables all three gyroscope axes.
- Address: 24 (in hex). Configuration: 01000000 (in binary). This configuration enables the FIFO, which stores data from the axes readings until it's read by the myRIO.
- Address: 2E (in hex). Configuration: 01000000 (in binary). This configuration puts the FIFO on stream mode, which stores all unread axis sensor data on the FIFO until read, deleting the older data in favor of the new one if the FIFO is full. The last three bits set the watermark level for the FIFO, if the FIFO is filled up to or more than the watermark level the most significant bit of the response from the address 2F (in hex) will be 1. The myRIO uses the third most signifcant bit from address 2F, which tells if the FIFO is empty or not, if not then it will start the sequence to read each axis data.

12.2 Calibrating the accelerometer

The accelerometer sends a response current centered on the supply voltage. This voltage number is converted in -1 to +1, first centering the center on 0 by reducing the voltage by an offset number, then multiplying by another number. Each axis needs to be manually calibrated. The calibration is done on the Options tab of the LabVIEW interface, the indicators "Acce (Axis) Offset" or "Acce (Axis) Multi" are to altered and then saved and loaded so they can be used, just altering the indicator is not enough.

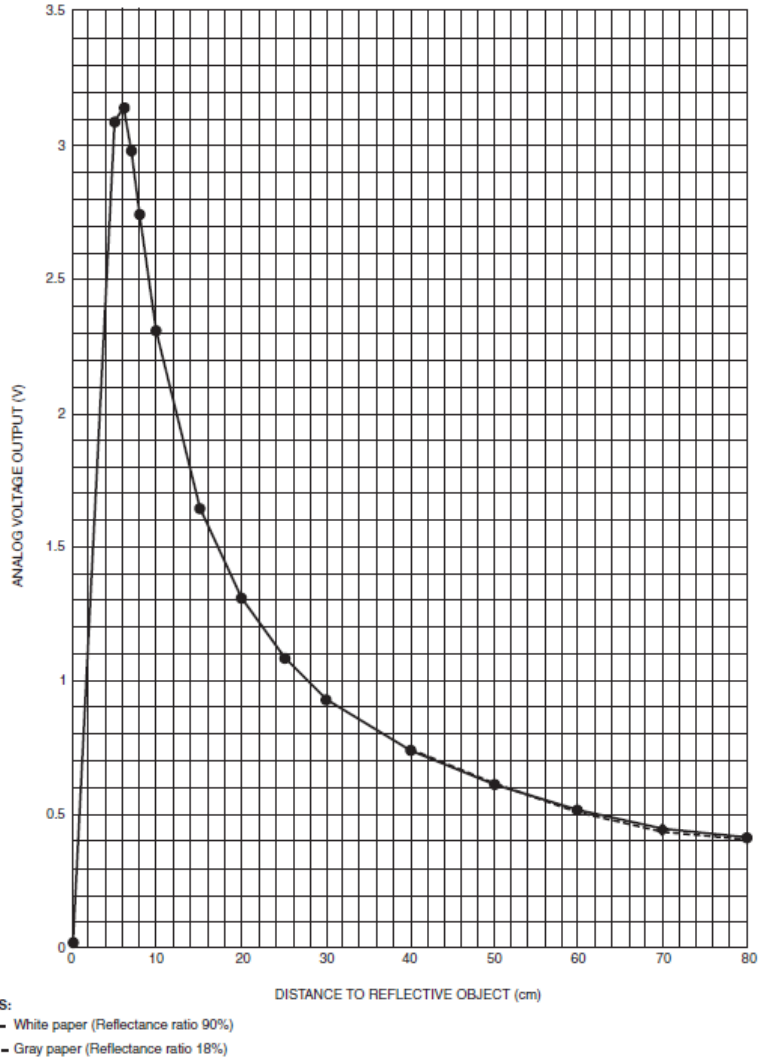
To confirm that the calibration is correct the maximum and minimum values need to be observed by inclining each axis, the optimum max and minimal values are between + or - 1 and + or - 1.2, going above this is going to affect the angle calculation around the 90° angle, so careful calibration of the accelerometer is essential to increase the precision of the system.

Only two of the accelerometer axes are used, the X (pitch) and Y (roll). This happens because the accelerometer gives the *projection* of the accelerometer local axes to the composite vector of all the acceleration forces on the accelerometer. Since the

accelerometer is used to detect the vector of acceleration of gravity, the interpretation of the Z axis becomes equal to the axis X, only offset in 90°.

12.3 Calibrating the IR rangefinder

The IR Rangefinder does not need to be calibrated unless it's introduced a capacitor to the circuit. The sensor works by detecting the intensity of a IR pulse from its LED, the brighter the pulse, the closer it is, this explains the random behavior of the sensor when pointed at light bulbs and mirrors. The sensor returns and inverse exponential signal, as shown on the picture below, that needs to be converted into real range values by a mathematical formula.



A table has been manually written containing the data points of response tension for the distance measured. With this table it is used the technique of curve fitting using the LabVIEW "General Polynomial Fit.vi", this VI has been adjusted using the following parameters: Polynomial Order: 7; Tolerance: 1E-10; Method: Bisquare; Algorithm: Givens.
This functions returns a series of polynomial coefficients, which are used to create a regular polynomial equation. The first value of these coefficients is subtracted by the current value in volts being measured by the sensor, this new value and the rest are used to solve the polynomial equation using the "Polynomial Roots.vi" function, which finds the zeroes on the voltage axis of the equation. By reducing the value of the first value based on the voltage the curve moves on the voltage axis and crosses the distance axis, this point where it crosses the distance axis is the final value of the distance being measured by the sensor, this value is the first zero found by the "Polynomial Roots.vi" function.

12.4 Calibrating the stepper motors

The steppers do not require calibration for normal function, but they need a step to angle conversion number so it can be used for orientation sensing on Unity, this number is called stepper conversion and can be found on the "Options" under the LabVIEW interface.

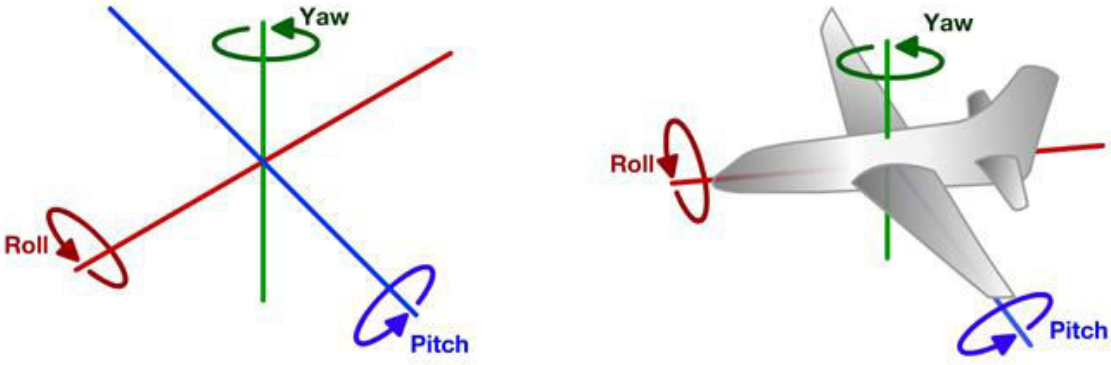
Since the steps angle are constant recalibration is not necessary unless the stepper is damaged or a different model is used. The conversion value was taken from solving this equation: (steps done)*(conversion value) = (angle measured).

12.5 Calculating the Angles

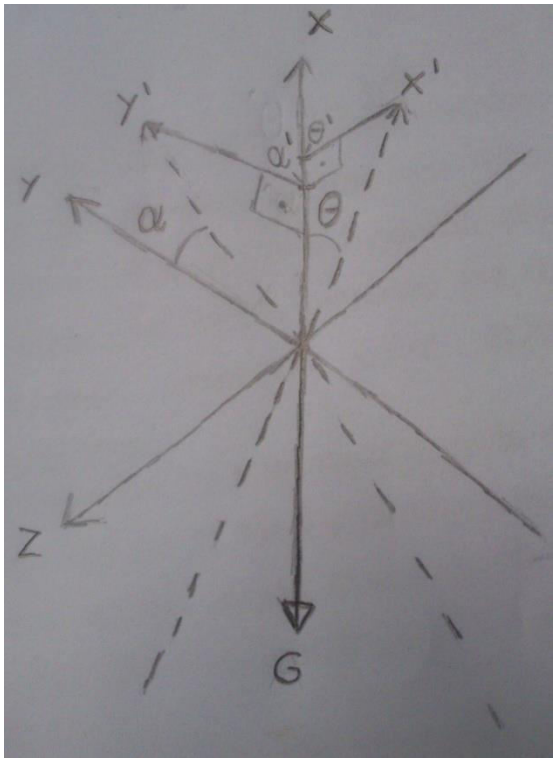
For a better comprehension the sensor axis have been defined as:

- Accelerometer X Axis: pointing up;
- Accelerometer Y Axis: pointing right;
- Accelerometer Z Axis: pointing forward;
- Gyroscope X Axis: pointing left;
- Gyroscope Y Axis: pointing forward;
- Gyroscope Z Axis: pointing up
- Final X Axis: pitch.
- Final Y Axis: yaw.
- Final Z Axis: roll.

The final axes are represented on the next picture:



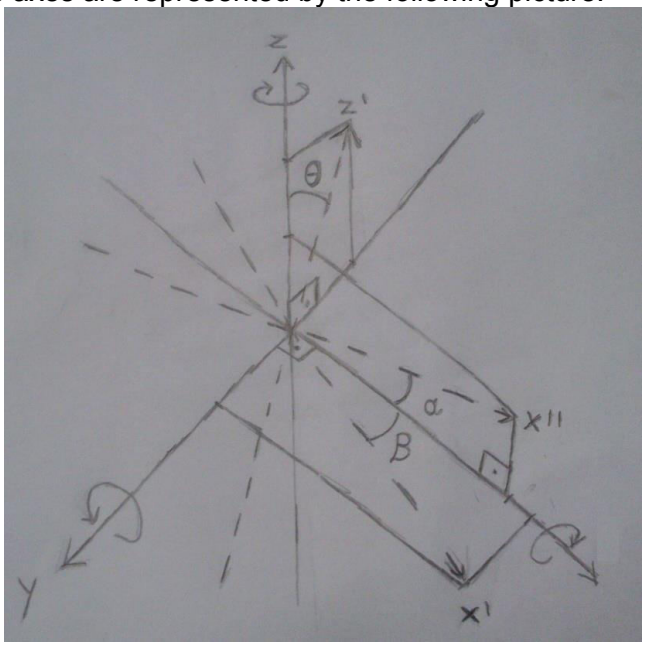
Using these axis as basis the accelerometer aces can be represented in the following way:



Rolling moves the y axis to position y', which gets projected to vector G - that represents the gravity acceleration vector - creating value α', that gets converted to angle α. Pitching does the same, it's angle is represented by θ. The conversion to the angle in degrees is done by the following equations:

$$X° = (\sin^{-1}\theta'/\pi) * 180 \qquad \text{and} \qquad Y° = (\sin^{-1}\alpha'/\pi) * 180$$

The gyroscope axes are represented by the following picture:



A rotation around the Y axis moves the X axis to X'' and creates the angle α, which is measured in degrees per second; the rotation around the Z axis moves the X axis to X', creating the angle β; the rotation around the X axis moves the Z axis to Z', creating the angle θ. This way, the angles α, β and θ represent the rolling, yaw and pitch.

Even though the axis X and Y from both sensors do not coincide, due to how the sensors work, the data coming from these axis can be interpreted as being the same.

To make the mathematic conversion of the data coming from the sensors it's necessary to first use the complementary filter to fuse the data of the X and Y axes. After the filter it has been observed that the more one axis is tilted, the less the maximum tilt value can be obtained on the other axis, which is an error.

This fact happens because the accelerometer makes the projection of its own internal axis X, Y and Z on the vector composed by all the acceleration forces actuating on the accelerometer. The accelerometer has a "weight" made of piezoelectric material, which based on the acceleration forces being done upon it creates pressure through the inertia between the weight and the sensor, varying the tension value sent by the sensor. This tension value is manipulated to show 1 when submitted to 9,8m/s² and -1 when submitted by the same acceleration on the inverse direction. Therefore, when there's a tilt in two axis the gravity acceleration vector reduces the maximum pressure in each axis, which results in a tilt value less than the real. This error on the tilt value doesn't have a linear proportion probably due to the way it is built, therefore creating a mathematical relation when there's tilting in more than one axis becomes a complex endeavor.

12.5.1 **Complementary Filter**

To mathematically utilize the data for each axis of the accelerometer and gyroscope the complementary filter has been used. This filter enhances the useful proprieties from both sensors and eliminates the negative ones. The filter is represented by the following equation:

Angle(t)=(Coefficient α)(Angle (t-1)+ Angle Gyro*(dt))+(1- Coefficient α)*(Angle Accel)*

Coefficient α being a number between 0 and 1, this filter reduces the value coming from the accelerometer, eliminating the noise, but creating a delay between the real angle value and the value created by the filter, to eliminate this delay the gyroscope data is used. Since the gyroscope measurement unit is degrees per second, it's necessary to multiply this value by the time between each measurement to obtain the value in degrees that the sensor has spun. This filter is not a precise as the filter of Kalman, also used to fuse two signals, but in compensation the complementary filter is much more simpler to build and uses a lot less processing cycles, making it ideal for this project.

12.5.2 **The Second Filter**

Since the accelerometer creates an error when tilted on two axis, two mathematical formulas and a few logical conditions have been created to minimize this error. On the LabVIEW's options it's possible to choose a threshold on the X and Y axes, when the value of one of these axis becomes bigger than the threshold, the system changes the method to calculate the second axis. After the first axis crosses the threshold, the system starts to store the angle of the second axis using only the value coming from the gyroscope to increase or decrease this value, which is different from the way to calculate the original value, that is calculated by using the complementary filter. The final value of the angle is calculated by adding a percentage of the value coming from the complimentary filter to the inverted percentage of the value coming from the gyroscope-only axis value. The percentage value starts at 100% of the complementary filter while the first axis value is under the threshold and starts to decrease linearly after crossing the threshold until 0% when the first axis it reaches 90°.

Since the accelerometer can't detect rotation around the vector of acceleration of gravity, the formulas for the complementary filter and the second formula are utilized only on the axis X and Y.

The first and second formulas to calculate the angle X are described as:

*AngleXPureOut=(AngleXGyro*dt)+AnglePureXIn))*

*AngleXFinal= (AngleXPureIn*b)+(AngleXOut*(1-b))*

Being:

- *AngleXPureOut*, the angle that is calculates using only the values coming from the gyroscope;
- *AngleXGyro*, a value in degrees per second coming from the gyroscope;
- *AngleXIn*, the angle equal to AngleXPureOut when t-1, necessary because the gyroscope only measures the acceleration and not the overall rotation;
- *AngleXFinal*, the angle that is sent to Unity for 3D reconstruction;
- *AngleXOut*, the angle coming from the complementary filter;
- "*b*", this value is calculated using the formula:

$$b=(AngleYOut-threshold)/(90-threshold)$$

Where *AngleYOut* is the value coming from the complementary filter that calculates the angle Y. Theres also this logical condition:

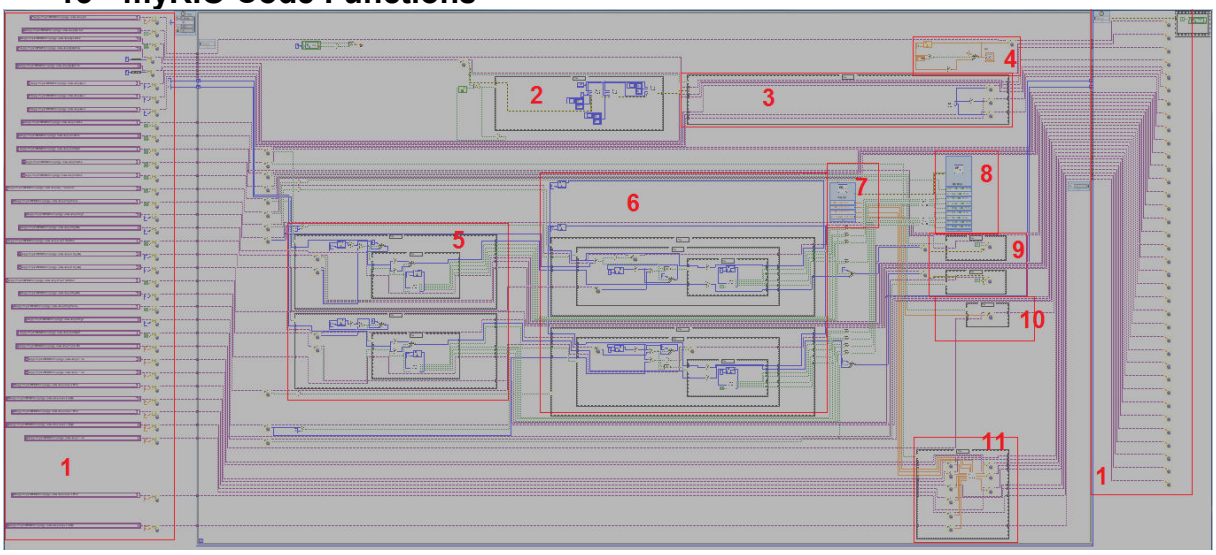
If *(AngleYOut-threshold)<0*, *"b"=0 and AngleXPureIn=AngleXOut*.

The calculation of the final angle of the axis Y is equal to the calculation of the axis X, only the names of the variables that contains "X" are change to "Y" and the inverse.

With this formula a part of the error can be eliminated when two axis are tilted, but this formula creates an incremental error caused by the gyroscope, therefore it's not advised to utilize the MyMapper in steep angles for long periods of time because to correct the value of the final angle it's necessary that the angle from the other axis stays less than the threshold.

The final values from the axis X and Y are sent directly to Unity, to calculate the value of the Z axis it's possible to use two methods. The first method revolves around using the counting of the steps made by the stepper motor "S2" and multiply this value by the value on the option "Stepper Conversion", which transforms the number of steps in degrees, this method is the most realiable one, only during extensive use that it has been noted incremental error, but this method does not utilizes the values sent by the gyroscope about the Z axis to calculate the rotation. The second method uses the values coming from the gyroscope to calculate the rotation, this method can detect all rotation around that axis, but it has the typical incremental error of the gyroscope.

13 myRIO Code Functions

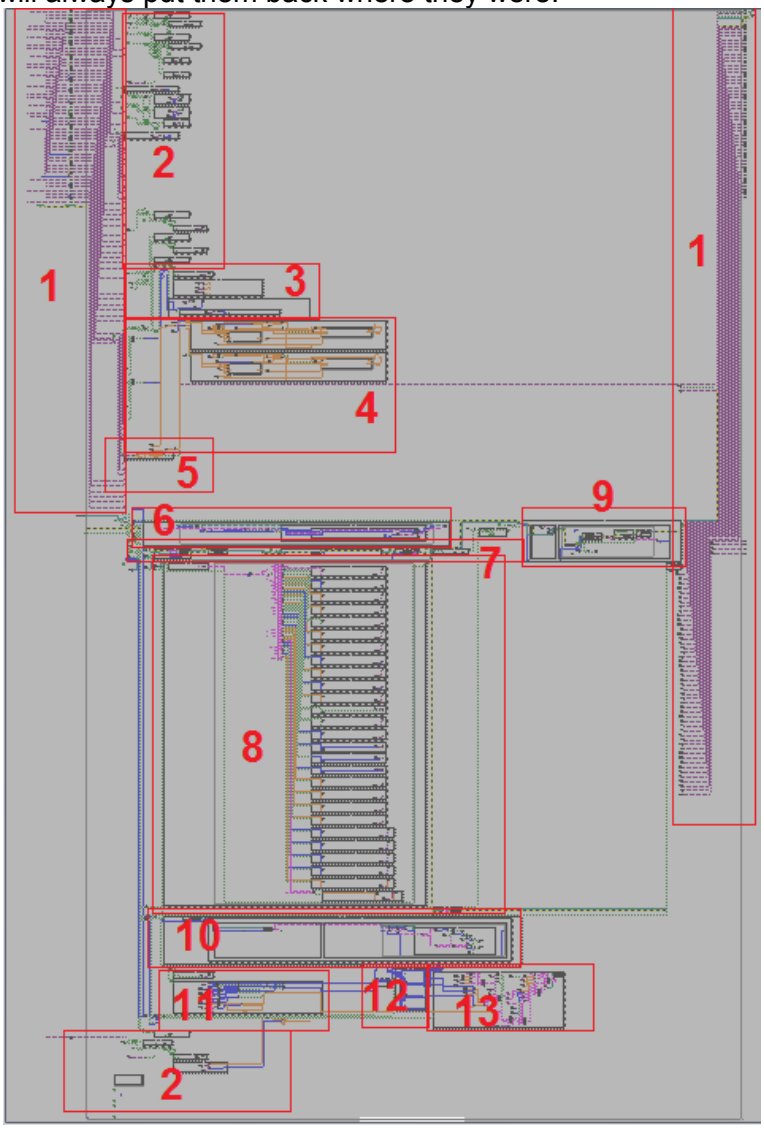


1. Shared Variables opener (left) and closer (right).
2. Gyroscope startup configuration.
3. Gyroscope axis reading blocks.

4. Sync block.
5. Binary stepper motor control. Upper block is for the S1, lower block is for the S2.
6. Step target control block. Controls the stepper motors by the number of steps done. Upper block is for the S1, lower block is for the S2.
7. Accelerometer and IR sensor analog input read block.
8. S1 and S2 motors control digital output.
9. Writing of the step count shared variable.
10. Writing of the IR sensor data to the shared variable.
11. Accelerometer data manipulation and writing to the shared variable.

14 LabVIEW Code Functions

The LabVIEW functions map is shown below, unfortunately the automatic wire cleanup function does not make a perfectly organized map, so some function blocks are unnecessarily far from each other and manually organizing them is not useful since the cleanup function will always put them back where they were.



1. Shared Variables opener (left) and closer (right).
2. Logic controls. These are the functions that control the sensors and the myRIO, the logic functions for these controls are mostly switches and latches to adapt the various ways you can enable/disable these controls.
3. Readers and indicators for the sensor raw data.

4. X and Z angle calculations. All the angle calculations being done by using the complementary filter using the Accelerometer and the Gyroscope and the second filter.
5. Sync indicator.
6. Options reader. It loads options from a file in the same directory of the LabVIEW files and then manipulates the string data into the proper variables.
7. UDP test blocks. These blocks execute the test UDP functions when necessary.
8. Options saver. It creates a string to save and update all the current options variables into a save file in the same directory of the LabVIEW files.
9. Connection sequence. The logic and the loop used to connect to LabVIEW. Normally LabVIEW will start by sending command "1A" to Unity, asking for command "1,X", X being the number of loop iterations LabVIEW has to wait to send sensor data to prevent memory overload, once comman "1,X" is sent LabVIEW gives a final "0,READY" command to Unity.
10. Command reader. This block reads all the commands coming from the Unity and executes the functions.
11. Sensor data manipulation. Manipulates the gyro data to insert the dead zone and uses the mean number function on the IR sensor to reduce the noise, then sends the data to the data sending block.
12. Gyro data accumulator. Since the LabVIEW doesn't send data every iteration there needs to be an accumulator that adds all the degrees per second measured by the gyroscope between each data sending.
13. Sensor data sending. Collects and sends all the sensor data to Unity by using a single and standardized string.

15 Unity Code Functions

Unity has five files which contain all the code used, these files are:

- UIScript: This file has all the functions related to the modification of the interface, also it controls the load and saving of the point cloud data.
- OptionsMenu: This file controls the loading and saving of the options available.
- Movement: This file is contains the functions to move and rotate the camera around based on the variables loaded from the options menu, it also has the control used by the mous look feature.
- UDPConnect: The most important file, it is responsible for the connection to LabVIEW by UDP, reading of the messages from LabVIEW, organization of sensor data and control of automatic sensing features on the Auto mode.

Most files contains a few standard functions, like the Start() function, which runs once during the start of the program, most variables are initialized on this function, the Update() function, which is a loop that runs as fast as the processor can execute them and the FixedUpdate() function, which runs on a regular pace during every physics check.

UIScript functions:

Function	**Description**
StarManual(); StartAuto(); StartSave(); Start Load; StartFreeLook(); StartOptions(); LeaveOptions(); StartConnect(); LeaveConnect(); BackFromFreeLook(); BackToMenu();	Standard functions to change the UI to its different parts. It works by activating and deactivating GameObjects based on their tag.
Send()	Used on the Manual mode, it sends the values written at the input boxes, creates the new object and moves it to the desired position.
Save(); Load();	Saves and loads the point cloud into a .txt file on the "Measurement Files" folder.
StartSave(); StartLoad();	Opens up the interface for saving and loading.
ClearAll()	Removes all points from the cloud.

ToggleRawData(); ToggleManualStep(); ToggleAutoOptions(); ToggleAutoOptions()	Toggles the panels and interface on the Auto mode.
ShowWarning(); HideWarning();	Shows or hides a standard warning with one button to close the warning, the text can be altered by using the .text component of the GameObject. Currently not used but ready for future use.
starttetstui(); leavetestui();	Opens or closes the UI for testing the UDP communication.

OptionsMenu functions:

During Start() it automatically reads the file "Options", that contains the configuration and loads the variables.

Function	**Description**
StartOptions()	Initializes all the variables and updates the input fields.
SaveOptions()	Loads the Options.txt file into a single string, breaks the string based on the option name, updates the string with the new values from the input fields, writes the new string into the options file and updates the options variables.
LoadOptions()	The same as the SaveOptions(), but it does not update the string with the values from the input field, instead it loads the string values into the input fields
DefaultOptions()	The same as the LoadOptions(), but instead of loading the Options.txt file it loads the DefaultOptions.txt file
AutoOptionsSave()	Based on the state of the toggles on the auto options menu writes 1's and 0's into the AutoOptions.txt file
AutoOptionsLoad()	Loads the AutoOptions.txt file into a single string, the string should only be 0's and 1's, which represent a toggle based on it's order

Movement functions:

OnGUI()	This function is based on the Legacy code for Unity GUI manipulation, it is called whenever an event happens, being able to be called multiple times per frame, making it excellent for precision controls. On this function it is detected whenever a key is pressed or released and compares that key with the one saved on the options menu
Fixed Update()	On this function all movement happens. The "if(mouselookToggle&&FreeLookToggle)" is where all the movement related to the mouse is calculated. The "if (FreeLookToggle)" is where all the movement related to keyboard input is calculated.
EnableMouselook(); DisableMouselook(); EnableFreeLook(); DisableFreeLook().	These functions enable or disable the variables that control the Mouse Look (using the mouse to rotate the camera) and Free Look (using mouse and keyboard to control the camera).
UpdateControls()	Updates the control options by reading the variables from the options script.

UdpConnect functions:

StartConnect()	Updates the variables used for connection by reading the variables from the options script.
ConnectToLabVIEW()	Either reads the connection variables from the options script or uses the values from the

	"Connect to LabVIEW" interface input fields. The function then binds Unity to an IP, if another IP needs to be used then Unity needs to be restarted. This function starts the regular sending of the connection command to LabVIEW on the "Fixed Update()" function
Fixed Update()	The main function that calls all the other functions. If the boolean "WaitForResponse" changes to true the connection command (a message asking for response from LabVIEW) starts to be sent at regular intervals.
StopConnect()	Stops the sending of the connection command to LabVIEW.
ReadSensorData()	Once connected, read all the commands strings from LabVIEW and categorize the information received.
ToggleRawData()	Toggles the variable that activates the Raw Data display on the Auto mode.
RawDataDisplay()	Formats the sensor data into readable forms for the Raw Data display.
ToggleManualStep()	Toggles the variabla that activates the Manual Control on the Auto mode.
UpSend(); DownSend(); LeftSend(); RightSend()	Sends the command to activate the steppers in the specified direction while the buttons is being pressed.
ResetS1(); ResetS2()	Reset the counter of steps of the specified stepper motor.
ToggleAutoRoutine()	Toggles the variable that activates the Auto Routine on the Auto mode.
AutoRoutine()	Controls the steppers to move in a specific order to scan the area in front of it based on the limits of vertical and horizontal rotation specified on the options menu.
AutoOptionStart()	Update the Auto mode options by reading the variables from the options script.
SendStepTarget()	Send the desired amount of steps each motor has to do.
ToggleOrientationSensing()	Toggles the variable that activates the orientation sensing on the Auto mode
OrientationSensing()	Calculates the angle of the camera and its control parents based on data read by the ReadSensorData() function.

16 Known Bugs

- The LabVIEW has a known problem that the Shared Variable Engine does not start correctly sometimes, it is not known why this happens, but a restart or redeploy on the myRIO can fix this problem. This problem has not yet known on the built code, only on the source test code.

17 Improvement Areas

The project has multiple areas where improvement is possible. These areas exist because of either budget constraints, since keeping the expenses low is one of the objectives of this project, and time constraints, since this project is being made by only one person there is not enough time to perfect and implement every desired feature. The most easily identified areas that can be improved are shown below:

- Increase sensor precision: the three sensors used (rangefinder, accelerometer and gyroscope) are relatively cheap and the precision available is enough to make the system work properly but it does not have the best precision possible, which makes it unviable as a proper measurement system.
- Add a Y axis sensor: currently the Y axis is the most imprecise axis because it uses only the gyroscope, which has incremental error, or the stepper, which does not detect global rotation, to sense rotation around the Y axis, the most obvious solution would be to add a digital compass to the system.
- Use faster motors: the current stepper motors have their speed limited to the maximum speed of processing the myRIO can achieve, since each step is executed once on each loop iteration on myRIO.
- Reduce noise on the rangefinder power supply: this sensor is very sensitive to noise, the output signal is based on the supply current and since the signal is exponential and inversed in relation to the distance, the farther the detected object is, the smaller the proportion of tension per distance measured, the more the system is sensitive to noise. This sensitivity is the reason why the steppers use a different power supply pin than the sensors.
- Add translation sensing: currently only rotation is being sensed, to add translation and rotation sensing working together would have a difficulty one order of magnitude higher than just the rotation, so due to time constraints it could not be developed. The difficulty arises mainly due to the accelerometers not being able to differentiate the vector of earth's gravity acceleration from other force vectors being inflicted upon it, so a very precise integration of the accelerometers and gyroscope needs to be done to differentiate rotation, translation and the orientation in relation to earth's gravity vector.