Aceleração por GPU de serviços em sistemas robóticos focado no processamento de tempo real de nuvem de pontos 3D

Leonardo Milhomem Franco Christino

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

Leonardo Milhomem Franco Christino

Aceleração por GPU de serviços em sistemas robóticos focado no processamento de tempo real de nuvem de pontos 3D

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Fernando Santos Osório

USP – São Carlos
Junho de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

M555a

Milhomem Franco Christino, Leonardo
Aceleração por GPU de serviços em sistemas robóticos focado no processamento de tempo real de nuvem de pontos 3D / Leonardo Milhomem Franco Christino; orientador Fernando Santos Osório . -- São Carlos, 2015.
106 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2015.

1. Robótica. 2. ROS. 3. GPU. 4. CUDA. 5. Veículo Auntônomo. I. , Fernando Santos Osório, orient. II. Título.

Leonardo Milhomem Franco Christino

GPU Acceleration of robotic systems services focused in real-time processing of 3D point clouds

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação - ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Fernando Santos Osório

USP – São Carlos
June 2016

Agradecimentos

Primeiramente e principalmente a Deus, meu Pai e Amigo. (1Tes 5:18).

Aos meus pais Janio e Ana que me enviaram e mantiveram aqui na cidade, dando suporte de todos os tipos, e minha família de Goiânia por me receberem tantas vezes nos momentos livres, Wander, Yeda, Fábio, Iraci, Ceci e Mariana.

Agradeço ao meu irmão e amigo Eduardo e amigos reais de Belém que me mantiveram como tal mesmo depois de tanto tempo longe, Arthur e Helder.

A todos amigos locais, ajudando-me a promover tudo que vivi até aqui: Anderson, Suelen, Simei, Débora, Daniel, aos meus antigos amigos e parceiros de turma de graduação, Carlos, Ricardo e Thiago, e aos colegas e amigos do laboratório de pesquisa.

A minha noiva que me apoiou no final do processo, abrindo novas portas e ajudando nos meus defeitos, Ana Elisa.

À Fundação de Amparo à Pesquisa do Estado de São Paulo, pela concessão da bolsa de mestrado e pelo apoio financeiro para a realização desta pesquisa.

Aos professores que dedicaram uma pequena parte das suas vidas a me ensinarem e ajudarem em meu caminho, e especialmente ao Prof. Osório por me auxiliar em na graduação, iniciação científica, tcc e mestrado.

Ao Instituto de Ciências Matemáticas e Computação, por colocar à disposição a área experimental e o laboratório

Muito Obrigado!

Mas em todas estas coisas somos mais do que vencedores, por aquele que nos amou. Porque estou certo de que, nem a morte, nem a vida, nem os anjos, nem os principados, nem as potestades, nem o presente, nem o porvir, nem a altura, nem a profundidade, nem alguma outra criatura nos poderá separar do amor de Deus, que está em Cristo Jesus nosso Senhor.

Romanos 8:37-39

Resumo

O projeto de mestrado, denominado de forma abreviada como “GPUServices”, se insere no contexto da pesquisa e do desenvolvimento de métodos de processamento de dados de sensores tridimensionais aplicados a robótica móvel. Tais métodos serão chamados de serviços neste projeto e incluem algoritmos de pré-processamento de nuvens de pontos 3D com segmentação dos dados, a separação e identificação de zonas planares (chão, vias), e detecção de elementos de interesse (bordas, obstáculos). Devido à grande quantidade de dados a serem tratados em um curto espaço de tempo, esses serviços utilizam processamento paralelo por GPU para realizar o processamento parcial ou completo destes dados. A área de aplicação em foco neste projeto visa prover serviços para um sistema ADAS: veículos autônomos e inteligentes, forçando-os a se aproximarem de um sistema de processamento em tempo real devido ao contexto de direção autônoma. Os serviços são divididos em etapas de acordo com a metodologia do projeto, mas sempre buscando a aceleração com o uso de paralelismo inerente: O pré-projeto consiste de organizar um ambiente que seja capaz de coordenar todas as tecnologias utilizadas e que explore o paralelismo; O primeiro serviço tem a responsabilidade de extrair inteligentemente os dados do sensor que foi usado pelo projeto (Sensor laser Velodyne de múltiplos feixes), que se mostra necessário devido à diversos erros de leitura e ao formato de recebimento, fornecendo os dados em uma estrutura matricial; O segundo serviço em cooperação com o anterior corrige a desestabilidade espacial do sensor devido à base de fixação não estar perfeitamente paralela ao chão e devido aos amortecimentos do veículo; O terceiro serviço separa as zonas semânticas do ambiente, como plano do chão, regiões abaixo e acima do chão; O quarto serviço, similar ao anterior, realiza uma pré-segmentação das guias da rua; O quinto serviço realiza uma segmentação de objetos do ambiente, separando-os em *blobs*; E o sexto serviço utiliza de todos os anteriores para a detecção e segmentação das guias da rua. Os dados recebidos pelo sensor são na forma de uma nuvem de pontos 3D com grande potencial de exploração do paralelismo baseado na localidade das informações. Porém, sua grande dificuldade é a grande taxa de dados recebidos do sensor (em torno de 700.000 pontos/seg.), sendo esta a motivação deste projeto: usar todo o potencial do sensor de forma eficiente ao usar o paralelismo de programação GPU, disponibilizando assim ao usuário serviços de tratamento destes dados.

Palavras-chave: Robótica, ROS, GPU, CUDA, Veículo Autônomo

Abstract

The master's project, abbreviated hence forth as “GPUServices”, fits in the context of research and development of three-dimensional sensor data processing methods applied to mobile robotics. Such methods will be called services in this project, which include a 3D point cloud preprocessing algorithms with data segmentation, separation and identification of planar areas (ground track), and also detecting elements of interest (borders, barriers). Due to the large amount of data to be processed in a short time, these services should use parallel processing, using the GPU to perform partial or complete processing of these data. The application area in focus in this project aims to provide services for an ADAS system: autonomous and intelligent vehicles, forcing them to get close to a real-time processing system due to the autonomous direction of context.The services are divided into stages according to the project methodology, but always striving for acceleration using inherent parallelism: The pre-project consists of organizing an environment for development that is able to coordinate all used technologies, to exploit parallelism and to be integrated to the system already used by the autonomous car; The first service has a responsibility to intelligently extract sensor data that will be used by the project (Laser sensor Velodyne multi-beam), it appears necessary because of the many reading errors and the receiving data format, hence providing data in a matrix structure; The second service, in cooperation with the above, corrects the spatial destabilization due to the sensor fixing base not perfectly parallel to the ground and due to the damping of the vehicle; The third service separates the environment into semantics areas such as ground plane and regions below and above the ground; The fourth service, similar to the above, performs a pre-segmentation of street cruds; The fifth service performs an environmental objects segmentation, separating them into blobs; The sixth service uses all prior to detection and segmentation of street guides.The received sensor data is structured in the form of a cloud of points. They allow processing with great potential for exploitation of parallelism based on the location of the information. However, its major difficulty is the high rate of data received from the sensor (around 700,000 points/sec), and this gives the motivation of this project: to use the full potential of sensor to efficiently use the parallelism of GPU programming, therefore providing data processing services to the user, providing services that helps and make the implementation of ADAS systems easier and/or faster.

Keywords: Robotics, ROS, GPU, CUDA, Autonomous Vehicle

Lista de Publicações

Os trabalhos desenvolvidos no período do projeto foram:

- Poster apresentado na *GPU Technology Conference* (GTC) em 04/2015 sob o título "*GPUService: GPU Acceleration of Robotic Services: Real Time 3D Point Cloud Processing*" (CHRISTINO e OSORIO, 2015);
- Apresentação no workshop do CROB realizado em 29/04/2015 em São Carlos;
- Publicação de artigo científico e apresentação na *Latin American Robotics Symposium* (LARS) sob o título "*GPU Service Acceleration: Robotic Systems Services Acceleration by GPU Parallelization Focused in 3D Point Clouds in Real-Time Processing*" em 10/2015 (CHRISTINO e OSORIO, 2015).

Índice de Figuras

Fig. 1 Projeto ALVINN 31
Fig. 2 Mapa gerado pelo veículo Stanley, apresentando obstáculos e via navegável ... 33
Fig. 3 O veículo Boss da equipe da universidade Carnegie Mellon, vencedor do DARPA Urban Challenge 34
Fig. 4 Veículos do Projeto Carina: (a) CaRINA I à esquerda (b) CaRINA 2 à direita . 36
Fig. 5 (a) Sensor Velodyne HDL-32E de 32 feixes a esquerda; (b) Leitura do Sensor Velodyne (10 quadros por segundo no HDL-32E com cerca de 70.000 pontos em cada quadro) 38
Fig. 6 Arquitetura da GPU Nvidia (NVIDIA CUDA Compute Unified Device Architecture Programming Guide 1.1) 41
Fig. 7 Dependência dos serviços 46
Fig. 8 Dados do Velodyne. Esquerda cores representando a intensidade e direita representando o índice do anel. 52
Fig. 9 Mapeamento da sequência original não estruturada para um vetor estruturado, reorganizando os dados e garantindo uma melhor ordem para o seu processamento pela GPU 53
Fig. 10 Correção Estrutural: Comparação de Tempos 56
Fig. 11 Correção Estrutural: Comparação de Tempos sem o PC-Embarcado..... 57
Fig. 12 Correção Estrutural mostrado pelo tempo..... 57
Fig. 13 Correção Estrutural: Número de Quadros Processados 58
Fig. 14 Correção Estrutural: Tempos do PC-Embarcado com ROSCore e dados remotos ou locais..... 58
Fig. 15 Correção Estrutural: Número de Quadros Processados pelo PC-Embarcado ... 59
Fig. 16 Erro Angular introduzido por não paralelidade da base do sensor com o chão 60
Fig. 17 Demostração das Regiões para Cálculo das Referências para a Matriz de Transformação (os 6 anéis em destaque são os considerados no cálculo e calibragem dos dados do sensor) 61
Fig. 18 Resultado da Correção de Distorção Angular: (a) Antes, (b) Depois 62
Fig. 19 Correção de Distorção Angular: Comparação de Tempos..... 62
Fig. 20 Correção de Distorção Angular: Comparação de Tempos..... 63
Fig. 21 Correção de Distorção Angular: Número de Quadros Processados..... 63
Fig. 22 Demostração da Altura do Sensor..... 65

Fig. 23 Segmentação do Chão: chão em azul, acima em roxo e abaixo em verde 66
Fig. 24 Segmentação do Chão: Comparação de Tempos 66
Fig. 25 Segmentação do Chão: Comparação de Tempos sem o PC-Embarcado 67
Fig. 26 Segmentação do Chão: Número de Quadros Processados 67
Fig. 27 Detecção de variações na hipotenusa dos pontos entre duas radiais 69
Fig. 28 Visualização com o RViz da segmentação das guias com os pontos com os pontos indentificados como guia pela variação de hipotenusa 69
Fig. 29 Detecção de variação do ângulo entre duas radiais por lei dos senos 70
Fig. 30 Visualização com o RViz com os pontos indentificados como guia pela variação de tangente 70
Fig. 31 Visualização com o RViz com os pontos indentificados como guia pela variação de intensidade 71
Fig. 32 Visualização com o RViz com os pontos indentificados como guia pelo filtro de convolução 72
Fig. 33 Visualização com o RViz com os pontos indentificados como guia pelo pelo algorítimo resultante 73
Fig. 34 Segmentação Fina de Guias: Comparação de Tempos 74
Fig. 35 Segmentação Fina de Guias: Comparação de Tempos sem o PC-Embarcado . 74
Fig. 36 Segmentação Fina de Guias: Numero de Quadros Processados 75
Fig. 37 Segmentação de dados geométricos retirados após a detecção de guia. 76
Fig. 38 Segmentação de Objetos demonstrado por cores 78
Fig. 39 Segmentação de Objetos: Comparação de Tempos 79
Fig. 40 Segmentação de Objetos: Comparação de Tempos sem o PC-Embarcado 79
Fig. 41 Segmentação de Objetos: Número de Quadros Processados 80
Fig. 42 Segmenação por Laser Scan 81
Fig. 43 LaserScan: Comparação de Tempos 82
Fig. 44 LaserScan: Comparação de Tempos sem o PC-Embarcado 82
Fig. 45 LaserScan: Número de Quadros Processados 83
Fig. 46 Comparação do Somatório de Tempos 86
Fig. 47 Comparação do Somatório dos Tempos sem o PC-Embarcado 86
Fig. 48 Detecção de início de início de bifurcação 88
Fig. 49 Máscaras de convolução para identificação da topologia da faixa de trânsito.. 89
Fig. 50 Diferenças das arquiteturas de CPU e GPU 98
Fig. 51 Arquitetura Fermi 99

Fig. 52 Dentro de uma GPU 100
Fig. 53 Hierarquia de uma GPU 101
Fig. 54 Hierarquia da API CUDA 102
Fig. 55 Comparativo entre GPUs com diferentes números de multiprocessadores 103
Fig. 56 Ciclo de vida de um kernel CUDA 103
Fig. 57 GIPY-1 Robot 106

Lista de Tabelas

Tabela 1 Resumo dos serviços desenvolvidos 46
Tabela 2 Detalhamento dos computadores usados para testes 47
Tabela 3 Exemplo da ordem dos dados recebidos pelo driver do sensor Velodyne 53
Tabela 4 Correção Estrutural: Resultados Numéricos Finais 59
Tabela 5 Correção de Distorção Angular: Resultado Numérico Final 64
Tabela 6 Segmentação do Chão: Resultado Numérico Final 68
Tabela 7 Segmentação Fina de Guias: Resultado Numérico Final 75
Tabela 8 Segmentação de Objetos: Resultado Numérico Final 80
Tabela 9 LaserScan: Resultado Numérico Final 83
Tabela 10 Resumo do Resultado Numérico Final 84
Tabela 11 Comparação do Somatório: Resultado Numérico Final 87
Tabela 12 Dados de uma GPU Tesla 100

Sumário

1. INTRODUÇÃO 25

1.1. MOTIVAÇÃO 25

1.2. OBJETIVO E ESCOPO DO PROJETO 27

1.3. PESQUISAS RELACIONADAS 27

1.4. ESTRUTURA DA DISSERTAÇÃO 29

2. TRABALHOS RELACIONADOS E BASE CONCEITUAL 31

2.1. CARROS AUTÔNOMOS 31

2.1.1. Projeto ALVINN 31

2.1.2. DARPA 32

2.1.3. Projetos Atuais 34

2.2. TECNOLOGIAS USADAS 36

2.2.1. ROS 36

2.2.2. Velodyne 38

2.2.3. Processamento em Placas de Vídeo (CUDA) 39

2.3. CONSIDERAÇÕES FINAIS 42

3. METODOLOGIA GERAL 43

3.1. CONSIDERAÇÕES FINAIS 47

4. DESENVOLVIMENTO E RESULTADOS 51

4.1. ROS E CUDA 48

4.2. PRÉ-PROCESSAMENTO DA NUVEM DE PONTOS 51

4.2.1. Serviço de Correção Estrutural 52

4.2.2. Serviço de Correção de Distorção Angular 60

4.3. SEGMENTAÇÃO DE ZONAS 64

4.3.1. Segmentação do Chão 64

4.3.2. Segmentação Fina de Guias 68

4.4. SEGMENTAÇÃO DE OBJETOS 75

4.5. LASER SCAN 80

4.6. CONSIDERAÇÕES FINAIS 84

5. CONCLUSÃO 85

5.1. PRINCIPAIS CONTRIBUIÇÕES 87

5.2. USO DOS SERVIÇOS E TRABALHOS FUTUROS 88

5.3. ESTÁGIO EM PESQUISA NO EXTERIOR 89

5.4. AGRADECIMENTOS FINAIS 90

6. REFERÊNCIAS 92

7. ANEXOS 97

7.1. TRABALHO DE CONCLUSÃO 97

7.2. PROGRAMAÇÃO CUDA 98

7.3. PROJETO DO ROBO TERAPEUTA DE CRIANÇAS AUTISTAS (EM INGLÊS) 105

1. Introdução

Este projeto se insere no contexto do processamento de dados de sensores usados em aplicações de robótica móvel. A seguir será descrita a motivação deste projeto, detalhes sobre o contexto em que se insere e os objetivos nele propostos.

1.1. Motivação

A robótica móvel se apresenta como uma área em constante e acelerada expansão, tendo hoje e em um futuro próximo aplicações em diversas áreas, desde o uso doméstico até veículos terrestres e aéreos com aplicações civis e militares. Por causa desta grande diversidade de áreas, a robótica móvel tornou-se uma área de extensiva pesquisa e que requer conhecimentos de diversos campos como, por exemplo, de Engenharia Mecânica, Elétrica e Computação.

Dentro da área de robôs móveis, existem aqueles capazes de efetuar a sua tarefa designados sem grande supervisão e com pouca intervenção e/ou coordenação humana, chamados de Robôs Móveis Autônomos (RMAs). Esses robôs realizam tarefas como mapeamento de ambientes, auto-localização, planejamento de trajetória, planejamento de ações e execução da navegação (seguimento de trajetórias e desvio de obstáculos). Para desempenhar essas e outras tarefas os robôs móveis precisam receber informações sobre o ambiente (através de sensores), tomar decisões e planejar ações (através de algoritmos de tomada de decisão) e executar ações no ambiente (através de atuadores).

A área da robótica móvel autônoma é relativamente recente e tem como maior desafio o desenvolvimento de robôs capazes de interagir com o ambiente, aprender e tomar decisões corretas para a conclusão de suas tarefas com sucesso. Uma aplicação direta dessa tecnologia é o desenvolvimento de veículos autônomos inteligentes. Tal área que surgiu no escopo de um sistema de rodovias autônomas na década de 60 (FENTON, COSGRIFF, *et al.*, 1968), podendo contribuir para o aumento da segurança em estradas e aperfeiçoar o fluxo de veículos em uma malha rodoviária, gerando uma redução no número de acidentes no trânsito, diminuindo engarrafamentos e auxiliando no transporte de deficientes e idosos. Além de aplicações urbanas, onde o projeto CaRINA (FERNANDES, SOUZA, *et al.*, 2014) se destaca, sendo inclusive discutido nas seções seguintes, há também um grande interesse em aplicações de automatização de transporte de pessoas e de cargas, assim como também em aplicações na área agrícola.

No desenvolvimento dos algoritmos de um veículo autônomo, muitas áreas da computação são exercitadas. Um primeiro exemplo são as áreas de controle e atuação em robótica, que envolvem controle de velocidade, aceleração e torque do motor para execução de uma missão. Áreas relacionadas ao planejamento, tal como a criação das missões do veículo ao traçar a rota que ele deverá realizar, são de muita importância também. Finalmente, as áreas de sensoriamento, como detecção de obstáculos, identificação de pedestres e rastreamento de objetos da cena, também mostra como pode ser complexo um sistema robótico autônomo.

Um sério problema e limitação imposta a estes tipos de aplicações é o tempo de resposta. Como se trata de uma ***aplicação crítica***, esta necessita de um ambiente de tempo real para controlar o veículo e evitar acidentes de trânsito. Portanto, procura-se desenvolver metodologias mais rápidas para o tratamento de dados. Essa velocidade se torna ainda mais crítica pela necessidade de que diversos algoritmos estejam sendo executados ao mesmo tempo (p.ex. mapeamento, auto-localização, planejamento de trajetória e desvio de obstáculos). Metodologias de exploração do paralelismo, como o processamento em *cluster*, em *grid* e outras técnicas, têm sido muito procuradas para solucionar estes problemas, mas cada uma delas apresenta também custos de comunicação, custos de energia e ainda podem apresentar elevados custos financeiros. Todos esses recursos são limitados dentro de um veículo, por isto é privilegiada a opção por uma tecnologia de processamento paralelo com custo mais reduzido e menor consumo de espaço e de energia.

A tecnologia de desenvolvimento de algoritmos paralelos em supercomputação tem usado muito do potencial de placas de vídeo com o advento da extensão da linguagem C desenvolvida pela Nvidia, o CUDA, para computação em GPU (SHANE, RODRIGUES, *et al.*, 2008). Como os dados recebidos pelo sensor são estruturados em nuvens de pontos 3D, permitindo um processamento com grande potencial para a exploração do paralelismo (processamento local e de pontos vizinhos da nuvem), o uso de GPU é uma boa forma de resolver os problemas de tempo. Outra característica do sensor a ser adotado neste projeto (sensor Laser Velodyne, descrito na seção 2.2.2) é a grande taxa de dados recebidos do sensor (em torno de 700.000 pontos por segundo), o que completa a motivação deste projeto: usar todo o potencial do sensor de forma eficiente, usando para isto o paralelismo de programação GPU para obter uma resposta em um tempo adequado para a aplicação.

1.2. Objetivo e Escopo do Projeto

Este projeto se faz parte das pesquisas em desenvolvimento junto ao Laboratório LRM (Laboratório de Robótica Móvel – ICMC/USP), e visa o desenvolvimento de serviços de processamento de sensores tridimensionais aplicados à robótica móvel, e mais em particular, com aplicações em veículos autônomos. Esses serviços deverão utilizar elementos de processamento paralelo, usando a força de supercomputação das GPUs. A área de aplicação em foco deste projeto foi de prover serviços de processamento de dados sensoriais para um carro autônomo, associado ao projeto CaRINA do LRM/ICMC-USP, forçando os serviços a se aproximar de um sistema em tempo real, que no contexto deste projeto é definido como realizar todo o pré-processamento antes do recebimento do próximo quadro do sensor (10Hz). Os serviços serão divididos em etapas, mas sempre buscando uma maior rapidez do processamento, através da exploração do paralelismo. Cada serviço terá como objetivo disponibilizar dados tratados da nuvem de pontos 3D, referentes à via sobre a qual o veículo está navegando e sobre o ambiente onde o carro autônomo se encontra (p.ex. identificando obstáculos). Estes resultados possuem diversas aplicações junto às pesquisas em andamento no LRM/ICMC-USP, e que podem ser inferidos ao se conhecer melhor outras pesquisas relacionadas e que estão em desenvolvimento neste laboratório, sendo discutidas na seção seguinte.

Portanto, o objetivo principal desta dissertação de mestrado é criar um ambiente favorável que una as tecnologias de veículos autônomos inteligentes, GPU e sensores 3D, e nela fornecer aplicativos de processamento de nuvens 3D eficientes ao aproveitar os traços paralelizáveis de sua natureza.

1.3. Pesquisas Relacionadas

O Projeto de mestrado GPUServices tem como foco o desenvolvimento de aceleradores por paralelismo de processamento de dados brutos de sensores em tempo real. Este trabalho irá ter um impacto relevante sobre outros trabalhos e pesquisas em desenvolvimento junto ao Laboratório LRM do ICMC/USP e espera-se que possa incrementar os conhecimentos da área de veículos autônomos inteligentes.

Um dos trabalhos similares é o projeto do aluno Daniel Sales (SALES, CORREA, *et al.*, 2014) que visa desenvolver um sistema de navegação inteligente para veículos autônomos baseado em pontos de referência e mapas topológicos do ambiente. Este trabalho faz uso de

dados e sensores que precisam ser previamente tratados a fim de identificar e destacar os elementos que são posteriormente usados na tomada de decisão e controle da navegação.

Alberto Hata (HATA, OSÓRIO e WOLF, 2014) (HATA, HABERMANN, *et al.*, 2014), Danilo Habermann (HABERMANN, SILVA e OSÓRIO, 2013) e Patrick Shinzato (SHINZATO e WOLF, 2015) desenvolveram trabalhos baseados nas informações obtidas a partir do sensor Velodyne. Estes trabalhos visam realizar uma navegação segura dos veículos do projeto CaRINA (ver seção 2.1.3.3) em vias públicas, consequentemente precisam detectar as bordas da pista e os limites da zona navegável da via. Em especial, o trabalho do Patrick Shinzato será referenciado neste projeto, por ser usado atualmente no carro e por ter sido melhorado com as pesquisas realizadas por ele em carros autônomos na Alemanha. O processamento conforme proposto pro Shinzato é dividido nas seguintes etapas:

1. Transformar os dados do Velodyne em uma estrutura matricial, chamada de RangeImage;
2. Filtrar a RangeImage de erros;
3. Identificar guias dentro da RangeImage;
4. Identificar obstáculos;
5. Delimitar a área navegável pelas guias e obstáculos;
6. Fundir dados do Velodyne com de Câmeras para melhor resultado.

Para manter o projeto GPUServices em linha com os desenvolvimentos na área, os serviços desenvolvidos foram arquitetados com base na necessidade que o projeto de Shinzato proveu: estruturação, filtragem e segmentação. É importante destacar também o desenvolvimento do projeto da Arquitetura Robótica do veículo CaRINA[1] (FERNANDES, SOUZA, *et al.*, 2014). Este trabalho também se insere neste contexto, pois é baseado no uso da ferramenta ROS (ver seção 2.2), adotada na arquitetura de base do CaRINA. Este trabalho também prevê a integração de dados sensoriais, onde será de grande valia o tratamento prévio, em tempo real, dos dados do sensor laser Velodyne (ver seção 2.2.2).

Por fim, cabe destacar que outros trabalhos de pesquisa poderão usar esse projeto para como base de aquisição sensorial e pré-processamento, como o trabalho com o intuito de detectar pedestres através da fusão de sensores (imagens e laser) (ALENCAR e WOLF, 2015) entre outros.

[1] Projeto CaRINA: Carro Robótico Inteligente para Navegação Autônoma.
Web: http://lrm.icmc.usp.br/carina

Esse trabalho também espera incrementar na pesquisa de processamento paralelo em veículos autônomos, tendo como inspiração o trabalho realizado na Audi pelo zFAS (KISS, 2015), Tesla Motors (EBERHARD e TARPENNING, 2006), Google (GUIZZO, 2011), e na própria NVidia (HUANG, 2015) (fabricante das placas gráficas usadas). Neste trabalho o uso de dados do Velodyne tratados e disponíveis em tempo real pode representar um importante diferencial na busca de melhores algoritmos para a detecção e rastreamento de pedestres.

1.4. Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma:

- Capítulo 2: Descreve diversos trabalhos relacionados, mostrando a motivação e contexto deste projeto. Também descreve as tecnologias usadas e seu contexto na área;
- Capítulo 3: Apresenta os objetivos de cada parte do projeto assim como sua estrutura e forma de desenvolvimento;
- Capítulo 4: Descreve detalhadamente a metodologia, o desenvolvimento e os resultados de cada serviço deste projeto;
- Capítulo 5: Apresenta a conclusão do projeto, formas de uso dos resultados deste trabalho e sugestões de possíveis trabalhos futuros;
- Capitulo 6: Lista a bibliografia completa utilizada no projeto;
- Anexos: Sumariza documentos importantes para o desenvolvimento deste projeto.

2. Trabalhos Relacionados e Base conceitual

2.1. Carros Autônomos

Devido ao atual grande interesse das indústrias automobilísticas e dos centros de pesquisa na área, diversos projetos de veículos autônomos surgiram nos últimos anos, como é o caso dos projetos originados nas competições dos DARPA Challenges (BUEHLER, IAGNEMMA e SINGH, 2009). Nessa secção, serão apresentados os projetos que deram origem a área, os principais projetos atuais internacionais e nacionais relevantes a este projeto e uma breve discussão sobre algumas arquiteturas de sistemas utilizadas na literatura importantes para o projeto.

2.1.1. Projeto ALVINN

O Projeto ALVINN (Autonomous Land Vehicle in a Neural Network) (BATAVIA, POMERLEAU, *et al.*), desenvolvido na universidade Carnegie Mellon utilizando o veículo NAVLAB II exibido na Fig. 1, utilizava técnicas de processamento de imagens baseado em redes neurais artificiais para conduzir o veículo através de diversos ambientes de forma autônoma.

Uma câmera montada na parte superior do veículo extraía imagens do terreno à frente do veículo. Tais imagens eram, então, utilizadas como entradas para uma rede neural. A rede neural utilizada no projeto era treinada de forma a correlacionar as imagens obtidas e o esterçamento do veículo durante uma operação humana.

Fig. 1 Projeto ALVINN

2.1.2. DARPA

Quase uma década após o projeto pioneiro na área, o projeto ALVINN[2], três competições patrocinadas pelo DARPA em 2004, 2005 e 2007, definiram importantes marcos no desenvolvimento de veículos autônomos. Nas duas primeiras edições os participantes deveriam desenvolver veículos capazes de percorrer grandes distâncias no deserto de forma autônoma (BUEHLER, IAGNEMMA e SINGH, 2007), e em sua última edição os veículos deviam completar um percurso em ruas (ambiente urbano) (BUEHLER, IAGNEMMA e SINGH, 2009) respeitando todas as leis de trânsito vigentes.

2.1.2.1. Darpa Grand Challenge

O *DARPA Grand Challenge* foi proposto no ano de 2003 para incentivar a pesquisa e o avanço na área de veículos autônomos. Com um prêmio de 1 milhão de dólares, em sua primeira edição que ocorreu um 2004, seu objetivo era criar um veículo capaz de percorrer aproximadamente 220 quilômetros de terrenos acidentados no deserto em no máximo dez horas. Apesar de 107 times terem se registrado e 15 iniciado o percurso, nenhum dos times foi capaz de percorrer mais que 5% do trajeto proposto, resultado que claramente mostra a dificuldade envolvida no desafio. Uma segunda edição do desafio foi realizada no ano de 2005, com um prêmio de dois milhões de dólares. Nessa edição 195 equipes se inscreveram e 23 iniciaram o percurso e dessas apenas 5 foram capazes de terminá-lo. Com um tempo de 6 horas e 53 minutos o veículo Stanley, da universidade de Stanford venceu a competição (DARPA Grand Challenge 2005).

A estratégia utilizada pela equipe de Stanford foi criar um mapa 3D em tempo real do terreno a frente do veículo, desviando de possíveis obstáculos e depressões como visto na Fig. 2. Também foram utilizadas câmeras de vídeo para identificar mudanças no padrão do terreno. Além dos sensores, foram utilizados 6 microcomputadores dentro do carro para o processamento das informações e tomada de decisão. É importante notar que os pontos de GPS que descreviam com precisão considerável a trajetória que o veículo deveria seguir foram previamente fornecidos às equipes, reduzindo o problema à navegação e desvio de obstáculos locais (THRUN, MONTEMERL, *et al.*, 2006)

---

[2] Projeto ALVINN: http://www.ri.cmu.edu/research_project_detail.html?project_id=160&menu_id=261

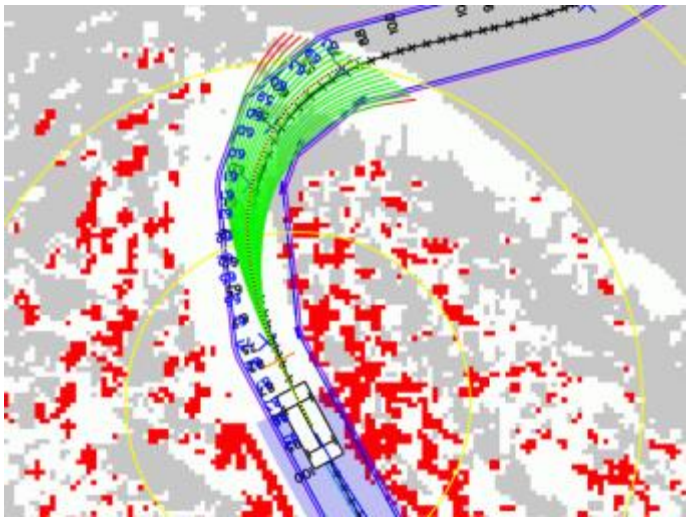

Fig. 2 Mapa gerado pelo veículo Stanley, apresentando obstáculos e via navegável

Com o sucesso da edição de 2005 do Darpa Grand Challenge o DARPA realizou, em 2007, um terceiro desafio chamado Darpa Urban Challenge.

2.1.2.2. Darpa Urban Challenge

Em novembro do ano de 2007 a agencia DARPA realizou o *Urban Challenge* em uma base da força aérea no estado da Califórnia. Nele, os competidores desenvolveram veículos autônomos para navegar através de um ambiente urbano, movendo-se ao meio do tráfego de veículos, negociando cruzamentos e evitando obstáculos sempre respeitando as leis de trânsito do estado da Califórnia (HENDERSON , MINOR , *et al.*, 2007).

Nessa edição, 89 times foram inicialmente aceitos par participar, 11 passaram pelas qualificações e participaram do desafio e desses apenas seis completaram o percurso de 96 quilômetros. O veículo Boss mostrado na Fig. 3, da equipe Tartan Racing liderada pela universidade Carnegie Mellon, obteve o primeiro lugar ao completar o trajeto em 4 horas e 10 minutos, com a equipe de Stanford ficando em segundo lugar com seu carro Junior. O Boss utilizava um algoritmo de navegação capaz de se localizar e estimar o formato da via onde se encontrava, além de um sistema para detectar obstáculos fixos e móveis ao longo do tempo (MONTEMERLO , BECKER , *et al.*, 2007).

Fig. 3 O veículo Boss da equipe da universidade Carnegie Mellon, vencedor do DARPA Urban Challenge

2.1.3. Projetos Atuais

Após os desafios DARPA, outras competições e desafios foram criadas, como a iniciativa europeia ERLOB[3] que é realizada regulamente, e o GCDC[4]. Também diversas empresas e universidades dedicaram laboratórios inteiros para o desenvolvimento dessa tecnologia (LUETTEL, HIMMELSBACH e WENSCHE, 2012).

2.1.3.1. Projetos Empresariais

O responsável pelo projeto de sucesso da Universidade de Stanford nas competições do DARPA, Sebastian Thrun[5], foi contratado pela Google em 2007 para fundar a GoogleX, parte da Google responsável por pesquisa e desenvolvimento de projetos totalmente inovadores. A GoogleX então desenvolveu o carro autônomo mais robusto atualmente (GUIZZO, 2011) e deu início a uma grande quantidade de pesquisas em montadoras de carros, como Audi[6] Mercedes[7] e Tesla Motors[8], entre outros. Também startups entraram na corrida, como AuroBots, Peleoton, GetCruise e Über. Há também as empresas que desenvolvem de forma terceirizada, como Bosch, Continental, TRW e a própria NVidia. Cada um dos carros usam uma configuração diferente, apesar de haver similaridades, de sensores no carro, e dentre as possibilidades esse projeto se mostra bem inserido frente às tecnologias usadas e os métodos de processamentos mais

[3] ELROB: http://www.elrob.org/

[4] GCDC: http://www.gcdc.net/

[5] Curriculum de Stanford de Sebastian Thrun: Sebastian Thrun – Home

6 Audi: http://www.audi.com/content/com/brand/en/vorsprung_durch_technik/content/2014/10/piloted-driving.html

7 Mercedes: https://www.mercedes-benz.com/en/mercedes-benz/innovation/research-vehicle-f-015-luxury-in-motion/

[8] Tesla Motors: http://my.teslamotors.com/fr_CH/forum/forums/tesla%E2%80%99s-musk-sees-fully-autonomous-car-ready-5-years

buscados: Aceleramento de processamento de nuvem de pontos 3D em Placas de Vídeo. Da mesma forma, os projetos nacionais buscam metodologias similares.

2.1.3.2. Outros Projetos Nacionais

No Brasil, as pesquisas relacionadas ao desenvolvimento deste tema (projeto e desenvolvimento de veículos autônomos) são recentes. Podemos destacar o projeto do Sistema Embarcado de Navegação Autônoma da Escola de Engenharia de São Carlos (SENA), o grupo da Universidade Federal de Minas Gerais (UFMG) com seu projeto CADU (Carro Autônomo Desenvolvido na UFMG), o projeto da UNIFEI (Drive4U), o projeto VERO (Veículo Robótico) do CTI/Campinas, o projeto de veículo autônomo do LCAD-UFES[9], e o projeto CaRINA (Carro Robótico Inteligente para Navegação Autônoma), do Laboratório de Robótica Móvel (LRM) do ICMC (São Carlos) da Universidade de São Paulo. Dentre estes projetos, aqueles que apresentaram melhores resultados até o presente, considerando a implementação de um veículo urbano e autônomo, foram o projeto CADU da UFMG e o projeto do CaRINA do LRM-ICMC/USP (FERNANDES, SOUZA, *et al.*, 2014).

2.1.3.3. Projeto CaRINA

Em abril do ano de 2010, o primeiro veículo do projeto, batizado de CaRINA I, foi adquirido com apoio de instituições como FAPESP, CNPq e o Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos (INCT-SEC). Esse consiste de um veículo elétrico Club Car modelo Carryall 232, automatizado pela própria equipe do LRM e instrumentado com sensores para localização e percepção. O CaRINA I, visto na Fig. 4(a) foi de grande importância para o projeto, pois auxiliou na definição dos requisitos e no desenvolvimento dos projetos de percepção, localização e planejamento.

O veículo batizado de CaRINA 2, um Fiat Palio Adventure, visto na Fig. 4(b), foi adquirido em julho do ano de 2011 e instrumentado para permitir o controle eletrônico do esterçamento, câmbio, freio e aceleração. Foi instalado nele também um Velodyne HDL-32, sensor laser omnidirecional com 32 feixes em diferentes alturas, um laser SICK 180 graus, uma câmera estéreo *Point Grey Bumblebee2* e uma *Point Grey Lady Bug*, uma câmera hemisférica com visão 360 graus e superior. O CaRINA 2 é a principal plataforma de pesquisa deste projeto de mestrado. Diversos projetos de pesquisa são desenvolvidos sobre a plataforma CaRINA 2.

---

[9] LCAD-UFES: https://lcadufes.wordpress.com/

Fig. 4 Veículos do Projeto Carina: (a) CaRINA I à esquerda (b) CaRINA 2 à direita

Uma das tarefas essenciais ao CaRINA 2 é de receber e analisar informações sobre o ambiente rapidamente e de forma precisa o suficiente para controlar o veículo e trafegar com segurança, como uma pessoa o faria se estivesse conduzindo o veículo. Para que isso seja possível, existe a necessidade de se desenvolver sistemas especiais para receber e tratar esses dados do ambiente. A sessão a seguir apresenta os conceitos e ferramentas para o desenvolvimento de sistemas capazes de prover serviços com essas características.

2.2. Tecnologias Usadas

O projeto utiliza diversas das tecnologias adotadas pelos projetos citados. A tecnologia sensorial adotada neste projeto é o sensor laser Velodyne. Para controle de comunicação entre o sensor e o computador afim de desenvolver toda a arquitetura de percepção e fornecimento de dados, foi adotado o ROS (*Robot Operating System*). E para atingir a força de processamento necessária neste projeto, foi adotado a tecnologia de processamento paralelo por placas de vídeo da NVidia com tecnologia CUDA.

2.2.1. ROS

O ROS[10] (*Robot Operating System*) é um sistema operacional que provê bibliotecas e ferramentas para ajudar desenvolvedores de software para criar aplicações em robótica (TRONIAK, 2012). Ele é executado sobre plataformas bem conhecidas, como o Linux. Nele, desenvolvedores contam com uma abstração do hardware e dos *drivers* utilizados nessa área de pesquisa. Ele permite que sejam desenvolvidos e reaproveitados códigos e funções, de modo

[10] ROS e o TurtleBot: http://www.ros.org e http://www.willowgarage.com/turtlebot

que um pesquisador de um lugar do mundo use o resultado de uma pesquisa de outro lugar, sem que eles nunca se conheçam. O ROS é adotado amplamente pela comunidade científica internacional na área de robótica, sendo a base do desenvolvimento das plataformas CaRINA I e II.

A arquitetura do ROS foi proposta pela Willow Garage, de modo a integrar em uma plataforma modular e aberta os conhecimentos espalhados pela comunidade, propondo assim uma arquitetura que é controlada por comunicação de rede. Essa comunicação possui um núcleo de "*subscribe*/*post*" de tópicos por diversos nós (disponibilizando serviços robóticos). Um nó é equivalente a um programa independente, mas contendo as facilidades de abstração do ROS. Eles podem compilar, iniciar e parar independentemente de outros nós. A comunicação entre eles é feita por tópicos ou serviços, que utilizam pacotes de rede, e eles são construídos a partir de tipos padrões de dados, como inteiros, ou estruturas (*structs*) a partir desses tipos padrões.

O ROS teve sua origem inspirado por um projeto chamado player[11], que era amplamente usado pela comunidade robótica. Então, por ser uma evolução de uma ferramenta que já era amplamente adotada, o ROS continuou sendo algo familiar para os desenvolvedores da área, mas, além disso, agora conta também com diversos novos recursos mais avançados, como o controle sobre a rede e a comunicação P2P entre os nós. O elemento *roscore* é o "servidor" dos tipos e endereços de nós e de mensagens disponíveis. A estrutura interna de diretório do ROS mais atual é composta de *packets*. Eles contêm os projetos com os nós, que por sua vez são configurados por um arquivo XML que fica na sua respectiva raiz. Por meio desse XML, é possível executar um conjunto de nós em simultâneo. Todos esses elementos existem para facilitar o desenvolvimento de um sistema que seja complexo.

Dentre os *packets* disponíveis, os que serão mais importantes neste projeto são o *rviz*[12], que é o visualizador de tópicos e mensagens padrão do ROS, e o *rosbag*[13], capaz de armazenar os tópicos e mensagens que foram distribuídas durante algum período de tempo (gerenciador de *logs* de dados). Dessa forma, é possível realizar uma captura de dados e reutilizá-los para o desenvolvimento e para os testes. O primeiro elemento a ser integrado com o ROS em um projeto de um robô ou veículo autônomo deve ser o sensor que irá observar o ambiente. Atualmente um sensor muito usado em veículos autônomos é o Velodyne, que foi usado extensivamente no

[11] Player: http://playerstage.sourceforge.net/

[12] RVIZ: http://www.ros.org/wiki/rviz

[13] ROSBAG: http://www.ros.org/wiki/rosbag

Darpa Urban Challenge, e que pode ser usado com o auxílio dos recursos disponibilizados pelo ROS.

2.2.2. Velodyne

O Velodyne[14] é um sensor omnidirecional (com percepção de 360 graus do entorno do veículo) que utiliza múltiplos feixes de sensores laser (dispositivo do tipo LIDAR). Ele possui duas versões, uma composta por 32 feixes de sensores lasers (HDL-32) e outra composta de 64 feixes. Estes dois tipos de Velodyne possuem lasers empilhados na vertical, o que cria os múltiplos feixes laser, permitindo assim a observação do ambiente em alturas diferentes (múltiplas camadas/recortes verticais de planos 3D do ambiente). Os lasers são então conectados a uma placa controladora que identifica qual a distância, em (x,y,z), e qual a intensidade de reflexão do ponto de incidência de cada feixe para cada ângulo de varredura. No sensor, que pode ser visto na Fig. 5(a), observa-se que sua metade de cima possui uma separação da base. Ele é o responsável por girar os feixes lasers de tal modo que cubram 360 graus. Por isso, além da distância e do índice de reflexão, o sensor também fornece qual o anel que o ponto pertence, ou seja, a qual dos feixes horizontais o ponto pertence, e qual o ângulo atual de rotação da leitura (entre 0 e 360 graus), obtendo assim uma posição (x,y,z) do ponto de incidência do laser. Um exemplo de leitura desse sensor pode ser visto na Fig. 5(b)[15].

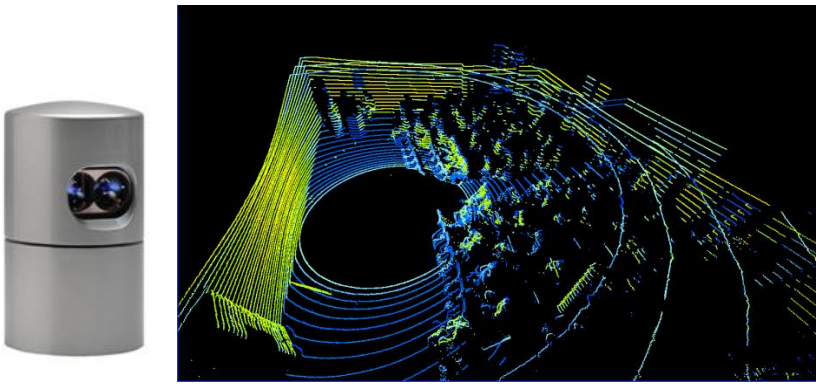

Fig. 5 (a) Sensor Velodyne HDL-32E de 32 feixes a esquerda; (b) Leitura do Sensor Velodyne (10 quadros por segundo no HDL-32E com cerca de 70.000 pontos em cada quadro)

O controlador do sensor envia os dados da nuvem de pontos 3D por um cabo de dados (*ethernet*) ao computador onde é executado o núcleo do ROS. Há a recepção do dado e a

[14] Velodyne: http://velodynelidar.com/lidar/lidar.aspx

[15] Leitura 3D Velodyne: http://www.youtube.com/watch?feature=player_embedded&v=P99eKc2uNJU

conversão dele em mensagem ROS[16]. A composição dessa mensagem (estrutura PointCloud2[17]) é de um vetor de pontos (x, y, z, i, r), onde temos as seguintes características:

- X: Coordenada X do ponto;
- Y: Coordenada Y do ponto;
- Z: Coordenada Z do ponto;
- I: Intensidade de reflexão do ponto;
- R: Anel que o ponto pertence;

O sensor Velodyne tem ganhado muito espaço na área de robótica devido a sua grande utilidade nos desafios DARPA descritos anteriormente (HALTERMAN e BRUCH, 2010). Sua precisão centimétrica, vinda da tecnologia LIDAR, e sua robustez de construção permitem que ele seja usado em ambientes onde carros autônomos devem circular mesmo com objetos dinâmicos na cena (MACLACHLAN, DUGGINS, *et al.*, 2012).

Sua maior vantagem é também sua maior desvantagem, visto que o giro do sensor, que ocorre a 10Hz, fornece cerca de 70.000 pontos a cada giro. Devido à grande quantidade de pontos, apenas aqueles que possuem uma incidência dentro dos limites de distância do sensor são enviados, sendo descartados onde houve erros de reflexão ou onde não foi encontrado nenhuma incidência até a distância máxima do sensor. O formato de entrada no computador é um grande vetor desorganizado de pontos que, devido à taxa de dados fornecida pelo sensor (70.000 pontos/giro, 10 giros/segundo), é necessário que haja um rápido do processamento para ser possível utilizar toda a qualidade e quantidade de dados do sensor sem perder os dados do próximo giro. Para isso, é usado processamento em placas de vídeo em CUDA.

2.2.3. Processamento em Placas de Vídeo (CUDA)

CUDA™[18] é uma plataforma de computação paralela e um modelo de programação, ambos inventados pela NVIDIA (NVIDIA). Ela permite aumentos significativos de desempenho computacional ao aproveitar a potência da unidade de processamento gráfico (GPU) e seu paralelismo, representado pela execução em paralelo de programas distribuídos em um grande número de núcleos de processamento. Alguns exemplos de uso da plataforma CUDA estão em: processamento de imagens para a identificação de placas ocultas em artérias, análise do fluxo

[16] Velodyne ROS: http://www.ros.org/wiki/velodyne

[17] Mensagem PointCloud2: http://www.ros.org/doc/api/sensor_msgs/html/msg/PointCloud2.html

[18] CUDA: http://www.nvidia.com.br/object/cuda_home_new_br.html

de tráfego aéreo, simulação de fluídos e visualização de moléculas (LANGDON, 2010) (GOMES). Todas estas aplicações requerem um alto poder de processamento.

A aplicabilidade do paralelismo não só é preso apenas aos métodos que são usados por usuários finais, como os listados acima, mas também pode ser aplicado a técnicas paralelizáveis para desenvolvedores. Por exemplo, o OpenCV[19] provê muitos elementos prontos para processamento de imagens, e a PCL[20] provê elementos de processamento de nuvens de pontos (KIM, HAWANGBO e KANADE) (VOUZIS e ARNOLD).

Observa-se que o uso de nuvens de pontos 3D, a exemplo dos dados vindos do sensor Velodyne, implica em grandes volumes de dados e processamentos altamente paralelizáveis. Pensando nisto, a equipe de desenvolvedores de GPU da NVidia e da comunidade científica internacional vem desenvolvendo materiais e produtos que incentivam o uso das duas tecnologias em conjunto (GONZALEZ, NAIKAL, *et al.*, 2010).

A programação feita em CUDA é baseada nas linguagens populares C e C++. Algumas extensões criadas pela NVidia possibilitam que o código de CPU possa delegar tarefas para a GPU executar. A grande dificuldade de se desenvolver para GPU é o paralelismo que isso envolve, tanto da parte de conhecimento e estrutura de programação quanto a arquitetura de software. Para entender como serão as etapas de desenvolvimento dos métodos do projeto, são necessárias algumas noções iniciais sobre GPUs e CUDA. A arquitetura geral (mais usual) das GPUs Nvidia é composta da forma apresentada na Fig. 6.

[19] OpenCV: http://opencv.willowgarage.com/wiki/

[20] PCL: http://pointclouds.org/

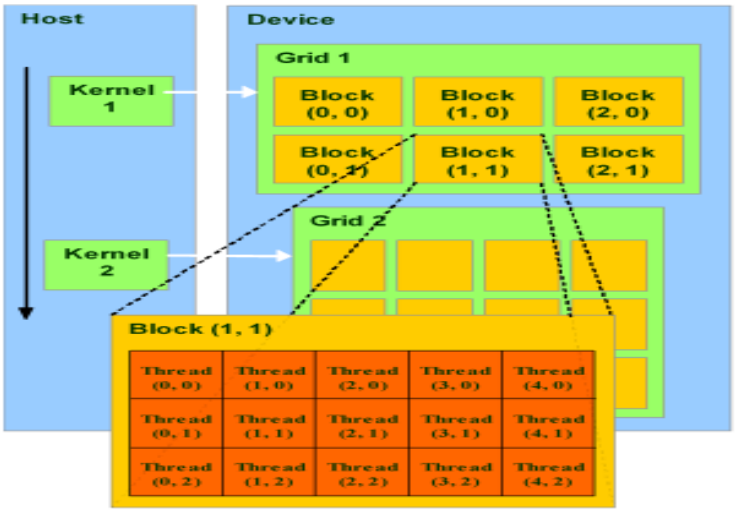

Fig. 6 Arquitetura da GPU Nvidia (NVIDIA CUDA Compute Unified Device Architecture Programming Guide 1.1)

- Kernel é a chamada que o programa em C/C++ faz para a placa de vídeo, onde mais de um Kernel pode ser chamado. O Kernel é composto por uma grid;
- *Grid* é a arquitetura de multiprocessos que a placa de vídeo está executando naquele instante. Várias *grids*, cada uma de seu próprio Kernel, podem ser criadas, com cada uma sendo executado em diferentes GPUs, por exemplo. Uma *Grid* é composta por uma estrutura de *Blocks*;
- *Block* é a primeira subdivisão da *Grid*. Ele possui várias características especiais, como gerenciamento de memória compartilhada, *warp*, e outros detalhes técnicos e de implementação. Em uma execução de um Kernel, o *Block* sempre executa simultaneamente, sendo necessário não ultrapassar os limites físicos de memória e cuda-cores da placa de vídeo. Apesar de mais de um bloco poder ser executado simultaneamente, isso não é controlado pelo desenvolvedor, logo não há comunicação entre *Blocks*. Um *Block* é composto de um conjunto de *Threads*;
- *Thread* é a execução unitária da GPU, ou seja, cada núcleo de processamento da GPU estará executando uma *thread*. Em uma GPU de 3.000 núcleos CUDA, existem no máximo 3.000 *threads* executando simultaneamente, podendo ser por exemplo um *Block* de 3.000 *Threads* ou 3 *Blocks* de 1.000 *Threads*. (LANGDON, 2011).

Para o uso dessa ferramenta é necessário a instalação de *drivers* específicos que trazem consigo o SDK (kit de desenvolvimento) da Nvidia para CUDA. Mais informações na arquitetura de placas de vídeo, de desenvolvimento de Kernels CUDA e análises de memória e de separabilidade no anexo 7.2.

2.3. Considerações Finais

A união do uso das tecnologias de GPU, ROS e Velodyne é um dos focos nesse trabalho. Dentre as referências bibliográficas pesquisadas, foi notado que não há muitas pesquisas que una estas três tecnologias, mas algumas unem duas delas. Os projetos pesquisados que utilizam pelo menos duas das tecnologias citadas fazem parte do fundamento de pelo menos um dos serviços.

As pesquisas mais relevantes são a de Patrick Shinzato (SHINZATO e WOLF, 2015) e sua técnica de correção matricial em CPU dos dados desorganizados do Velodyne, o alinhamento de nuvem de pontos em CUDA (MIAO, FU, *et al.*, 2010), detecção de guias pelos projetos Boss e Junior (HENDERSON , MINOR , *et al.*, 2007), técnica de LaserScan, ou RangeScan por simulação a partir de sensores LIDAR de um feixe por Patrick Shinzato (SHINZATO e WOLF, 2015), segmentação de objetos em pontos de nuvens por proximidade (DBScan) (BIRANT e KUT, 2007), trabalhos de processamento de nuvens 3D em placas de vídeo (ZENG, BI, *et al.*, 2013), localização e mapeamento de robôs acelerado por placas de vídeo (ZHANG e MARTIN, 2013) e segmentação de nuvens de pontos de terreno, usando ROS, Velodyne HDL-64E S2 e CUDA (BAKER e SADOWSKI, 2013).

O uso das três tecnologias em conjunto em um produto final ainda é escasso diante dos projetos estudados e citados durante a revisão bibliográfica, mas foi percebido que é o caminho mais comum que está sendo trilhando dentre eles (DOUILLARD, UNDERWOOD, *et al.*). Para realizar o desenvolvimento deste projeto, primeiramente foi necessário configurar um ambiente que pudesse unir as diferentes tecnologias adequadamente, e então cada tecnologia pôde ser usada de forma eficaz ao mostrar suas devidas vantagens quando adicionadas à aplicação completa.

3. Metodologia Geral

Visando agregar as tecnologias e as técnicas previamente citadas, o cerne do projeto é a criação de um ambiente que as una e que faça uso eficiente e eficaz da grande quantidade de dados fornecidos, fornecendo para outros projetos dados rapidamente processados. A aplicabilidade do processamento dos dados de nuvens de ponto não se limita ao sensor Velodyne ou aos carros autônomos, permitindo uma boa abrangência para uso dos resultados aqui descritos, porém a principal aplicação e foco deste trabalho que será descrita é para uso nessa configuração. No final do projeto eram esperados os seguintes resultados e algoritmos desenvolvidos: correção de estrutura de dados e correção espacial da nuvem de pontos e detecção de elementos relevantes para navegação do veículo (guias e obstáculos).

A estrutura do projeto foi montada agregando elementos requisitados por outros projetos e a arquitetura de desenvolvimento das três tecnologias citadas previamente: ROS, CUDA e Velodyne. O ROS, descrito na seção 2.2.1, é usado nos projetos de veículos autônomos atualmente desenvolvidos junto ao laboratório onde foi realizada esta pesquisa, dando sua importância de uso no projeto, porém a sua maior vantagem é sua descentralização dos programas e uso de rede e sua arquitetura em serviços (ou *nodes*), o que deu origem ao nome deste projeto (GPUServices). Esses elementos permitem integração deste projeto em outros que necessitem de sua funcionalidade por meio da arquitetura ROS e também usando-os como em modelo de API. Logo, o projeto foi separado em serviços que se comunicam pela rede (também é possível utilizá-los como API[21]), sendo dois de pré-processamento (estrutural e angular) e outros três de segmentação (chão, guias e objetos).

Da mesma forma que o ROS, o sensor Velodyne é usado em outros projetos relacionados ao veículo autônomo do laboratório desta pesquisa. Este sensor é usado devido a sua grande quantidade de dados e a precisão que esses dados são gerados. Com ele, procura-se detectar com mais fidelidade elementos do carro e aumentar a quantidade de informação com uma futura fusão sensorial (com câmeras e radares). O sensor é conectado pelo cabo de rede Ethernet e possui um programa que roda no computador que junta os dados recebidos e os fornece para o ambiente ROS. Porém, a grande frequência de dados gerados requer uma atenção especial.

---

[21] Interface de Programação de Aplicações é uma interface de acesso a funcionalidades e dados a programas externos que apenas pretendem usar seus serviços e não diretamente envolver com detalhes de programação.

Essa grande frequência faz necessário desenvolver um software altamente veloz e para isso foi escolhido usar aceleração por placas de vídeo. Para desenvolver em GPU é necessário se preocupar com todos os elementos descritos na seção anterior. Por isso, foi desenvolvida uma metodologia que foi aplicada no desenvolvimento de todos os serviços a fim de dar mais eficiência a esse processo:

1. Desenvolver um algoritmo eficiente em CPU;
2. Portar o algoritmo para um Kernel GPU, considerando uma *Grid*, um *Block* por *Grid* e uma *Thread* por *Block*;
3. Analisar o paralelismo inerente do algoritmo, como laços de execução, e dividi-los aumentando o número de *Threads* por *Block* e o número de *Blocks* por *Grid*;
4. Analisar e melhorar a coalescência de memória, acelerando também o processo de I/O de dados;
5. Analisar outros paralelismos, como: executar o algoritmo inteiro várias vezes ou a separabilidade do Kernel entre vários Kernels diferentes mais eficientes;

Portanto, diante do escopo onde o projeto está inserido, a abordagem adotada visa a pesquisa e o desenvolvimento de um sistema de soluções paralelas (denotadas de serviços) processadas em GPUs, para aceleração do processamento até tempo real, o qual é definido neste projeto como processamento completo antes do próximo frame disponibilizado já que maior velocidade não acarretaria mais informações ao sistema e sim tempo ocioso. Esta definição de tempo real leva em consideração que o tempo de reação de motoristas é de cerca de 1,5 segundos (WU, YUAN, *et al.*, 2009) e que os tempos de processamento conseguintes ao deste projeto não são conhecidos, deixando a eles a responsabilidade de reações necessárias do veículo autônomo.

Estes serviços usam técnicas de processamentos de dados 3D e 2D aplicadas a dados 3D, os quais servirão de ponto de partida para outros projetos e aplicações ao fazer uso destes serviços providos. O sistema deverá primeiramente permitir uma interface e uma arquitetura amigável para o desenvolvimento CUDA em ambiente ROS, e prover uma API, assim como tópicos e mensagens ROS com informações processadas dos dados recebidos do sensor Velodyne, informações tais que deverão ser úteis para outros projetos em andamento e usando técnicas que também serão úteis para aceleração de processamentos de dados 3D em outros projetos. Desta forma, os serviços propostos implementados em GPUs também irão retirar uma parte da carga de processamento sob a CPU central, permitindo que outras tarefas de mais alto nível possam ser tratadas pela CPU.

Para o desenvolvimento deste projeto foi usado o sensor Velodyne modelo HDL-32E, o sistema ROS versão Jade e processamento paralelo em GPU com placas de vídeo NVidia com

CUDA toolkit versão 7.5 utilizando ponteiros de precisão simples, e cujos hardwares são detalhados na próxima sessão. Primeiramente foi integrada e configurada uma estação de trabalho que fosse capaz de unir todas as tecnologias referentes a este projeto. Após concluída esta etapa, foi realizada a etapa de estudo e pesquisa referente ao desenvolvimento e paralelização dos métodos de processamento da nuvem de pontos 3D do sensor Velodyne. Nessa fase, trabalhos de pesquisa referentes aos métodos de pré-processamento e tipos de serviços que podem ser oferecidos relativos ao tratamento dos dados da nuvem de pontos fornecida pelo Velodyne foram estudados, assim como sua capacidade de serem integrados às outras tecnologias do projeto. Também foi feita uma análise das formas de como integrar e comunicar dos dados repassados entre sensor, GPU e programa principal do serviço prestado, considerando o uso da plataforma ROS.

Uma vez realizadas estas etapas iniciais, foi desenvolvido o pré-processamento nos dados recebidos do sensor Velodyne, caracterizando assim o primeiro e o segundo serviços disponibilizados pelo projeto. Com a estrutura correta (módulos, dados e comunicação) e representação adequada da nuvem de pontos 3D, foi realizada uma etapa de segmentação do ambiente pelo terceiro e quarto serviço e detecção de elementos relevantes pelo quinto e sexto serviço. Os serviços estão muito ligados uns aos outros, montando uma arquitetura de dependência, como mostrada na Fig. 7. Ao usar os serviços, foi possível extrair dados como guias da rua segmentadas, separação planar do ambiente em chão e não-chão, segmentação de objetos da cena e correção de dados do sensor rápido o suficiente para atingir tempo real. Iniciamos então pela arquitetura de desenvolvimento: ROS e CUDA. Um resumo dos serviços descritos pode ser visto na Tabela 1.Todos os serviços serão mais detalhados na próxima seção, com suas metodologias próprias e justificativas de estarem no projeto.

Tabela 1 Resumo dos serviços desenvolvidos

Nome do Serviço	Descrição do Serviço
Correção Estrutural	Correção matricial dos dados desorganizados e desordenados recebidos do sensor
Correção de Distorção Angular	Correção da distorção angular dos dados do sensor por imperfeição na sua base fixa e variação de ângulo do carro por amortecimento
Segmentação do Chão	Separação das regiões planares horizontais para uso posterior
Segmentação Fina de Guias	Pré-segmentação das guias da rua para uso posterior
Segmentação de Objetos	Segmentação da nuvem de pontos em *blobs*, ou objetos, para uso de outros projetos e do último serviço
Laser Scan	Simulação de técnica de Sensor laser com apenas um Laser usado primeiro por Shinzato (SHINZATO e WOLF, 2015) com o uso da arquitetura aqui usada, e posterior segmentação entre guias

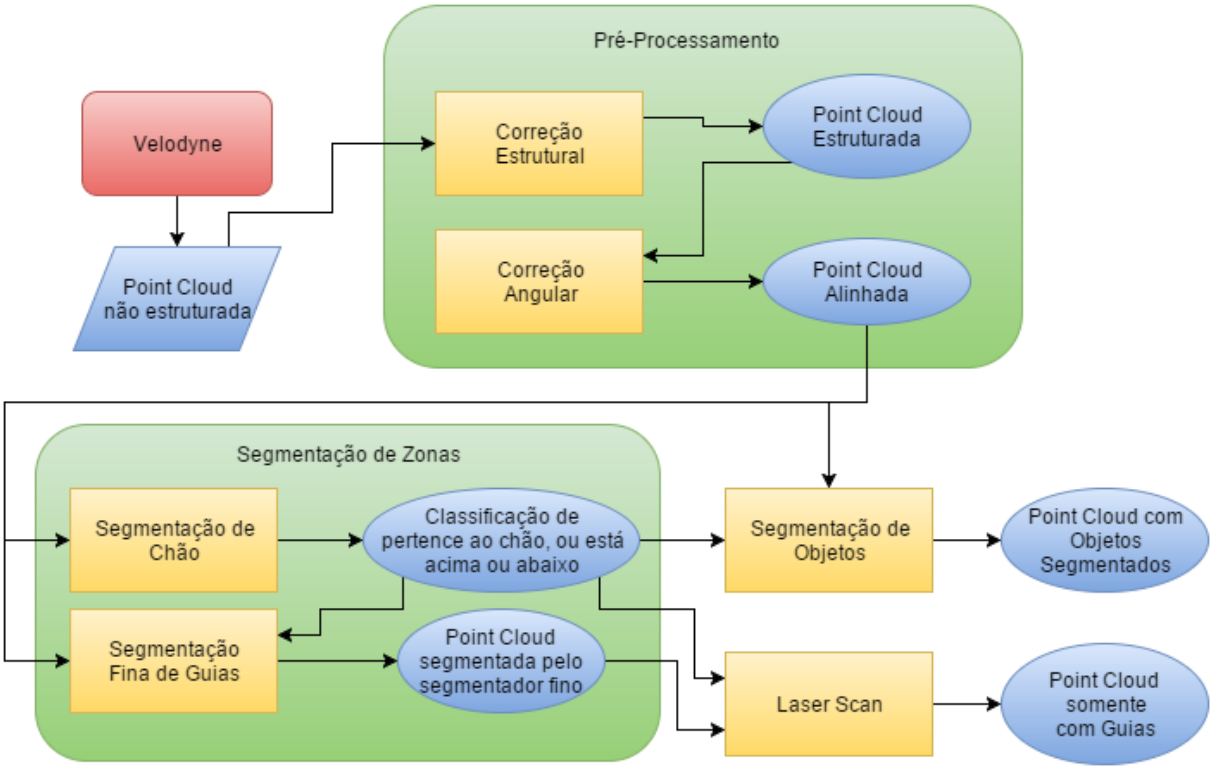

Fig. 7 Dependência dos serviços

3.1. Equipamento Utilizado

Para a programação em placas de vídeo, foram estudadas as características de placas de vídeos que poderiam ser usadas. Para uma placa ser usada num carro, é necessária uma placa com baixo consumo de potência, mas que ainda possa ser eficiente. Por isso foi testado a placa Nvidia 620 GT[22], que conta com 96 *cores* CUDA e consumo de 49W, e a placa embarcada Jetson. Para comparação, também foi incluída uma placa de vídeo extremamente eficiente, a Titan X, e uma de médio porte usada mais em ambientes de jogos, a 650 GTX. Neste capítulo serão descritas as técnicas adotadas e implementadas e serão apresentados os resultados obtidos em experimentos com dados reais. Serão então apresentados benchmarks para comparação de desempenho de códigos desenvolvidos pelo projeto em CPU e também suas versões em GPU. Também será apresentado como configurar o ambiente para compilação e execução dos códigos. As implementações foram testadas nas configurações descritas na Tabela 2.

Tabela 2 Detalhamento dos computadores usados para testes

Nome	Detalhamento de Hardware
PC-Carro	Intel core i5 2.9GHz Quad-core, 6GB de memória RAM, 1TB de HD 7200 RPM, e NVidia GeForce 620 GT com 96 CUDA Cores e usando no máximo 49W
PC-Comum	Intel core i3-3220 3.3GHz Quad-core, 8GB de memória RAM, 256 GB de HD-SSD, e NVidia GeForce 650 GTX com 384 CUDA Cores e usando no máximo 64W
PC-Processamento	Intel core i7 4.0GHz Quad-core, 16GB de memória RAM, 1T de HD 7200 RPM, e NVidia Titan X com 3072 CUDA Cores e usando no máximo 250W
PC-Embarcado	NVidia Jetson TK1 com Arm Cortex A15, 2 GB de memória RAM, 16 GB de memória 4,51 eMMC, e NVidia Tegra K1 com 192 CUDA Cores[23] e usando no máximo 12W

[22] Especificação da Nvidia 620 GT: http://www.nvidia.com.br/object/geforce-gt-620-br.html

[23] Apesar do maior número de cores na Jetson, esta placa é mais limitada em termos da CPU embarcada, inclusive tendo menor consumo de energia e um clock mais baixo.

3.2. ROS e CUDA

Devido à complexidade de se montar um ambiente operacional onde é corretamente configurado tanto o uso do compilador CUDA quanto o compilador ROS em conjunto, essa etapa foi utilizada para ganhar proficiência com os diferentes ambientes e ferramentas de desenvolvimento, e para descrever os passos que precisam ser feitos para criar um ambiente semelhante para uso do sistema.

O desenvolvimento de uma metodologia de trabalho e de ferramentas de apoio são também componentes importantes deste projeto. Portanto, primeiramente são instalados os programas necessários para utilização de cada um dos componentes usados na implementação dos serviços. Então, para uso em conjunto destes componentes, são utilizados arquivos especiais que os integram. Com o ambiente pronto, o uso das tecnologias se torna muito facilitado, permitindo que mais pesquisadores possam criar novas pesquisas que possam usufruir de suas vantagens. A suíte de desenvolvimento conta com Linux Ubuntu[24] versão 15.4, ROS Jade[25], CUDA toolkit 7.5[26], Python 2.7, NumPy 1.9.2[27] e PyCUDA 2015.3[28].

Para o uso das tecnologias em conjunto, pesquisas foram feitas junto a diversas fontes. Para realizar a configuração do ambiente foi usado o trabalho detalhado no anexo 7.1, que foi desenvolvido previamente a este projeto de mestrado como projeto de conclusão de curso. Neste trabalho foram reunidas informações dos elementos mínimos necessários para a compilação integrada das ferramentas CUDA e ROS considerando projetos desenvolvidos em C/C++. Iniciou-se pela instalação do driver da placa de vídeo e compilação dos códigos exemplos. Após isso, foi realizada a instalação do ROS completo e a construção de um *workspace* para compilação do projeto.

As diferentes configurações de hardware possuem necessidades distintas devido a sua arquitetura, e isso foi importante para essa etapa do projeto. O PC-Comum e o PC-Carro necessitaram de uma configuração simples como definida na seção 3.2, o PC-Processamento foi configurado remotamente por SSH e foi testado duas arquiteturas ROS: na primeira o processo principal do ROS estava em um computador diferente e o PC-Processamento apenas fornecia o

---

[24] Ubuntu 15.04, sistema operacional usado: http://www.ubuntu.com/download/desktop

[25] ROS Jade: http://wiki.ros.org/jade/Installation/Ubuntu

[26] CUDA Toolkit 7.0: https://developer.nvidia.com/cuda-downloads com instalação: http://www.r-tutor.com/gpu-computing/cuda-installation/cuda7.0-ubuntu

[27] NumPy 1.9.2 e pacotes adicionais para cálculo numérico e processamento de dados para Python: http://www.scipy.org/install.html

[28] PyCUDA 2015.3, biblioteca de integração de CUDA e Python: http://wiki.tiker.net/PyCuda/Installation/Linux

serviço, recebendo e respondendo pela rede, e no segundo o PC-Processamento possuía o processo principal, não tendo o *overhead* de comunicação pela rede. Os testes com o processo principal ROS em um computador diferente mostrou que o atraso causado pela comunicação em rede é maior que a taxa de envio de dados do Velodyne, fazendo com que seja inviável essa metodologia para uso em real-time. Semelhantemente, o PC-Embarcado foi testado em ambas configurações, porém ele demonstrou ainda outro problema: devido ser embarcado, ao ter todos os processos rodando somente nele, há outro atraso devido à baixa capacidade de processamento do processador Tegra. Logo, é sugerido que nesses casos que o processo principal do ROS esteja rodando em outro computador, porém o sensor Velodyne esteja conectado diretamente ao computador que fará o processamento descrito neste projeto, que rodará o driver do sensor e os serviços aqui propostos. Esse detalhe será mais discutido nos resultados do Serviço de Correção Estrutural na seção 4.1.1.

3.3. Considerações Finais

Neste capítulo foi apresentada uma visão geral das etapas e processos que compõem o sistema proposto e implementado neste projeto de mestrado. É importante destacar que os veículos autônomos e inteligentes, em grande parte, realizam a aquisição de dados locais obtidos a partir de seus sensores, de forma a permitir que o veículo navegue sobre a via (ruas, avenidas, estradas), e que não colida com obstáculos presentes no ambiente, seja saindo fora dos limites (bordas) da pista, ou seja atingindo algum obstáculo que está sobre a via (p.ex. pessoas, ciclistas, outros veículos, que atravessam a via). Sendo assim, é de grande importância a disponibilização de serviços que permitam, a partir dos sensores como o Velodyne, determinar a região por onde o veículo pode trafegar, o que em termos práticos consiste em identificar a via: zona segura, plana, bem delimitada, sobre a qual o veículo pode trafegar. Por outro lado, é importante detectar também os limites da pista, ou seja, as bordas da calçada, que separam a zona segura da via da zona insegura e não trafegável (p.ex. calçada). Além disto, podemos ter obstáculos que estão sobre a via ou calçadas e que devem ser monitorados, pois representam elementos que podem causar acidentes. Os obstáculos são todos os elementos que estão acima (ou abaixo no caso de buracos) do plano seguro e navegável da via, sejam eles outros carros, pessoas, ciclistas, animais, ou mesmo, elementos como a borda da calçada que é um obstáculo, e elementos do cenário em geral, como árvores, placas de sinalização e postes de iluminação.

A metodologia proposta visa, primeiro, estruturar as ferramentas e serviços para que possam ser disponibilizados, e depois, oferecer serviços básicos de ajuste e calibração dos dados do sensor pelas (1) Correção Estrutural e (2) Correção de Distorção Angular, seguido dos

serviços de detecção da via navegável (plana), como (3) Segmentação do Chão e das bordas da via com a (4) Segmentação Fina de Guias, terminando pelos serviços de detecção de obstáculos, (5) Segmentação de Objetos e (6) Laser Scan.

4. Desenvolvimento e Resultados

4.1. Pré-processamento da Nuvem de Pontos

Com o ambiente configurado, o próximo passo é o de modelar a nuvem de pontos fornecida pelo sensor. Para que seja possível alcançar os objetivos dos próximos serviços e ajudar outros projetos em andamento, foi necessário o remanejamento dos dados recebidos, pois não são suficientemente/adequadamente estruturados para uso eficiente. Além de problemas estruturais, o sensor é preso a uma base fixa do carro autônomo que não é perfeitamente paralela ao plano do chão. Somando o erro devido à imperfeição da rua e o de amortecimento durante o deslocamento, a nuvem de pontos também possui um erro angular que é problemático para serviços posteriores. Em função disto, são aplicados métodos de mapeamento e correção de dados.

Foi então preciso estudar a estrutura dos dados da nuvem de pontos pelo código de tratamento de pacotes de rede específicos do Velodyne, disponibilizado pela distribuição do ROS na forma de um driver. A execução deste driver cria mensagens ROS com os dados na forma de um vetor de quíntuplas de acordo com o algoritmo a seguir. Para permitir uma melhor visualização desses dados, a ferramenta RViz[29] usa as três dimensões de distância mais uma dimensão representada por uma escala de cor (espectro de cores do arco-íris), como mostrada na Fig. 8. Todos esses elementos podem ser encontrados em maior detalhamento nas seções 2.2 e 2.2.2.

```
INICIO
   Ponteiro_para_dados_do_velodyne ← scan(numero_de_pacotes)
   PARA i DE 0 ATÉ numero_de_pacotes FAÇA:
        ENQUANTO obter um pacote
                receberPacote(Ponteiro_para_dados_do_velodyne)
        FIM ENQUANTO
   FIM PARA
   Publicar_mensagem_ROS(Ponteiro_para_dados_do_velodyne)
   RETORNAR true
FIM
```

[29] RVIZ: http://www.ros.org/wiki/rviz

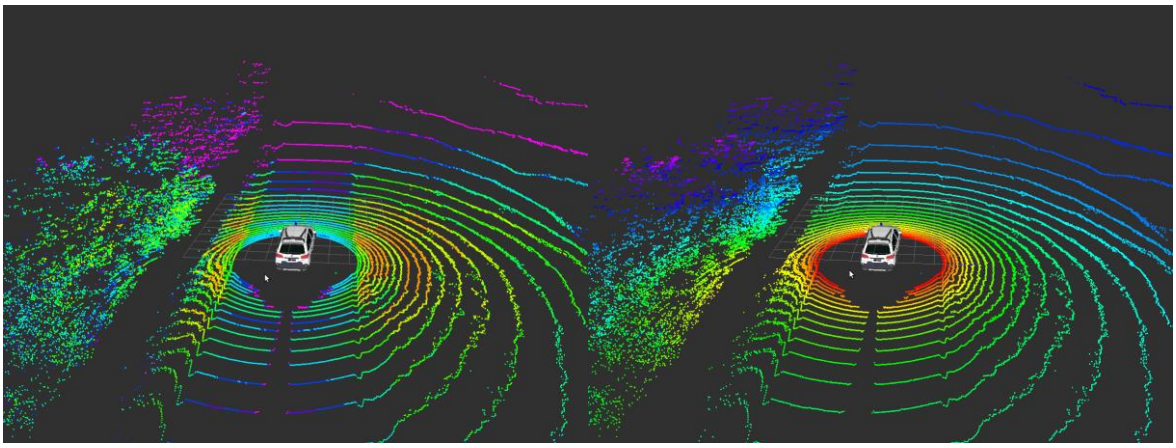

Fig. 8 Dados do Velodyne. Esquerda cores representando a intensidade e direita representando o índice do anel.

O driver do Velodyne recebe os pacotes em uma ordem não definida por utilizar uma conexão sem confirmação (UDP[30]), portanto não há nenhuma garantia na ordem de recebimento e nem como saber se o dado chegou ou se houve falha na comunicação. Foi então estudada a viabilidade de integrar o algoritmo de remanejamento proposto na seção 4.1 ao driver do Velodyne, porém foi preferido criar um receptor de mensagens ROS para receber os dados do driver e processar posteriormente. Ou seja, a execução dessa etapa do projeto recebe os dados do driver e então os processa conforme proposto separadamente.

Para adaptar o código para também ser capaz de ser executado em CUDA, foi realizado um estudo da dimensão da nuvem de pontos para definir o tamanho ideal de *Block* e *Grid* conforme descrito na seção 2.2.3. Os parâmetros escolhidos foram uma matriz de 32x1800. A divisão foi feita tal que o tamanho seja o mais divisível possível pela quantidade de CUDA cores das placas de vídeo usadas. Ou seja, para uma placa de vídeo de 96 cores, é usada uma divisão de 4x24 (que é o número de cores) de *Threads* por *Block*, e consequentemente uma divisão de 8x75 (equivalente a $\frac{32x1800}{4x24}$) de *Blocks* por *Grid*.

4.1.1. Serviço de Correção Estrutural

Para correção da estrutura dos dados, um algoritmo foi desenvolvido que processa a nuvem de pontos não estruturada original fornecida pelo sensor e a mapeia em outra nuvem de

[30] UDP: http://www.hedensberg.se/documents/UDP_protocol.doc

pontos, como exemplificado na Fig. 9, onde é mostrada uma reorganização e reordenação dos dados fornecidos a partir de um vetor exemplo demonstrativo do formato dado pelo driver do sensor. Esse algoritmo busca tornar um vetor desordenado, desestruturado e com falhas (dados faltando) em uma estrutura de dados matricial, permitindo uma maior eficiência à projetos e serviços seguintes, pois é a base de todos os outros serviços.

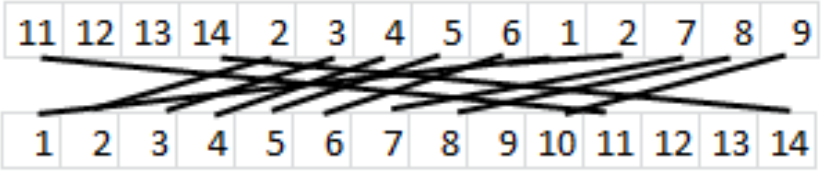

Fig. 9 Mapeamento da sequência original não estruturada para um vetor estruturado, reorganizando os dados e garantindo uma melhor ordem para o seu processamento pela GPU

Com esse serviço, são colocadas diversas informações referentes a nuvem de pontos original em uma estrutura matricial de 32 linhas e 1800 colunas. A matriz pode ser mudada e customizada, porém esses valores foram escolhidos por serem bem divisíveis pelo número de núcleos multiprocessadores (cores) das placas de vídeo e também devido a ser o formato mais similar à estrutura gerada internamente pelo sensor (32 feixes lasers com *throughput* de aproximadamente 70 mil pontos por quadro, que equivale a uma varredura de um giro completo de 360 graus). Ao fim desta etapa é fornecido um serviço que trata e fornece uma estrutura de nuvem de pontos com as seguintes características como saída:

- Cada linha é um anel diferente do sensor;
- Cada coluna é um ângulo de leitura diferente do sensor;
- Leituras não recebidas do sensor terão um valor padrão de 0 para todos valores (x, y, z, anel e intensidade), representando que o sensor não obteve distância para aquela leitura (p.ex. sem ponto de incidência detectado por estar apontado para o céu);

Para a correção proposta, é importante notar os detalhes da estrutura recebida pelo sensor, pois serão importantes para o prosseguimento do projeto:

Tabela 3 Exemplo da ordem dos dados recebidos pelo driver do sensor Velodyne

Ordem de chegada do pacote	1	1	1	2	2	2	3	3	3	...	7	7	7	...
Elemento na matriz (anel x coluna)	1,1	1,2	1,3	1,14	1,16	1,17	1,6	1,7	1,8	...	1,32	2,1	2,2	...

1. Cada pacote ethernet possui uma sequência de pontos ordenados do anel mais baixo ao anel mais alto, como mostrada na ordem da Tabela 3;
2. Os pacotes podem ser interrompidos em qualquer lugar da sequência, como mostrado do terceiro para o quarto elemento da Tabela 3;
3. O sensor não garante que todos os valores da matriz de pontos são enviados, pois dados de pontos que não foram identificados pelo sensor são descartados, o que é o caso de pontos no infinito, sob uma superfície com índice de reflexão prejudicial (como um preto fosco) ou superfícies com inclinação muito elevada, como feixes lasers quase paralelos com o horizonte. Consequentemente há pontos "perdidos" como do quarto pro quinto elemento na Tabela 3;
4. A ordem dos pacotes não é bem estruturada, portanto é possível receber dados da primeira coluna, e então da terceira, e posteriormente da primeira novamente, como visto na Tabela 3 ao perceber que cada cor representa um pacote e sua sequência não é estruturada.

Em primeira instância foi desenvolvido um código em Python que realiza o remanejamento necessário utilizando uma estrutura de dados bem conhecida pelos desenvolvedores Python, o Numpy. Foi também utilizada uma pequena biblioteca adaptada que originalmente realizava a captura de nuvens de pontos 3D de outro tipo de sensor. Essa biblioteca se utiliza de sequência de dados para acelerar sua transferência entre variáveis e apenas reescreve o cabeçalho dos dados, permitindo assim resultados de conversões mais velozes. O algoritmo desenvolvido é simplificadamente demostrado a seguir:

```
def fixCloud(data, height=32, width=1800):
  cloud = pointclouds.pointcloud2_to_array(data)
  tans = arctan2(cloud['x'], cloud['y'])
  cloud2 = np.zeros(cloud, dtype=coud.dtype)
  for t in range(0,32):
        c = cloud['ring'] == t
        cloud2[t,tans[c[0]]]=cloud[c]
```

Um código em C usado para comparação é o código desenvolvido pelo Patrick Shinzato (SHINZATO e WOLF, 2015) descrito na seção 1.3 que atualmente é usado no carro autônomo. Apesar do resultado do código em Python ter um resultado bom, e principalmente estar na estrutura do Numpy, ainda é perceptível a diferença de eficiência em comparação a um código em C, e por isso o uso do código do carro autônomo como base para criar o Kernel da placa de

vídeo foi considerado. A grande vantagem do código em Python frente ao C é sua velocidade de prototipação, permitindo um desenvolvimento bem mais rápido (GAREY e LANG, 2008) (LUTZ e MARK, 2001). Em ambos os casos, uma rotina de republicação por mensagem ROS é responsável em permitir uso dos dados estruturados por outras partes do carro.

Para o desenvolvimento do Kernel para placa de vídeo, foi realizada a paralelização do processamento dependente da quantidade de dados recebido. Em CPU é inicializada uma matriz de tamanho 32x1800x5 totalmente zerada (sendo esse o valor padrão previamente definido caso haja erro de detecção do sensor) e o número de *threads* na placa de vídeo é o tamanho do vetor de pontos recebido do driver do sensor. A paralelização máxima de cada hardware descrito é dependente no número de CUDA *cores* de sua placa de vídeo, sendo esse o valor de processos simultâneos sendo executados. Cada *thread* então calcula aonde na matriz final o ponto pertence e o coloca na posição.

Os resultados das diferentes implementações e dos diferentes hardwares são mostrados na Fig. 10, onde cada eixo é um tipo de execução (tempo em CPU x tempo em GPU). Na Fig. 10 cada ponto representa um quadro de dados 3D fornecido pelo Velodyne e processado, sendo que o eixo X representa o tempo de processamento deste frame de dados junto a CPU, e o eixo Y representa o tempo de processamento deste mesmo quadro de dados junto a GPU, permitindo assim comparar o tempo da CPU e GPU para um mesmo conjunto de dados e visualizando a variação dos tempos considerado vários frames de dados. Note que os tempos na CPU tem uma maior variabilidade (eixo X) do que os tempos de GPU (eixo Y).

Os resultados apresentados na Fig. 10 apresentam claramente que os tempos em GPU são inferiores e mais agrupados aos tempos em CPU. O espalhamento dos tempos em CPU foi creditado ao seu uso em outras tarefas, como controle de SO, execução do driver do sensor e tarefas de I/O, e esses motivos especialmente prejudicaram a eficiência do PC-Embarcado por conter um processador ARM bem inferior aos demais. Na GPU a execução é feita em até 5ms e com baixo espalhamento, conforme destacado na Fig. 11, exceto para o caso do PC-Embarcado - NVidia Jetson (na ordem de até 25ms neste dispositivo). Os tempos em CPU variam de 10ms a 120ms, sendo que na maioria dos casos ficam em torno de 20ms (bem acima dos 5ms na GPU - ver Fig. 11), exceto novamente para o PC-Embarcado que, assim como na GPU, também consome tempos bem maiores (40 a 120ms sequencial em CPU, contra 5 a 25ms paralelizado em GPU).

A Fig. 12 demonstra a dificuldade de uma boa análise em usar um gráfico temporal, porém mostra tanto o espalhamento de algumas das linhas temporais quanto a baixa eficiência do PC-Embarcado. As execuções foram realizadas sob o mesmo conjunto de dados e a alta

separabilidade no eixo horizontal identifica o fato do CPU ser usado em outros processos ao mesmo tempo, como controle de SO e drivers. Finalmente, a Fig. 13 mostra quantos quadros foram processados no conjunto de dados, o que se mostra importante uma vez que é possível que quadros sejam perdidos devido aos processos demorarem mais que 100ms para o processo ou ainda excesso de processamento pela CPU do computador. As Fig. 14 e Fig. 15 mostram os resultados de usar o *ROSCore* e o driver do sensor localmente na placa embarcada ou remoto, recebendo os dados pela rede. Já que a CPU é menos usada, é mais eficiente para o algoritmo ter os dados recebidos remotamente, porém o tempo de transferência é tão grande que já ultrapassa o limite de 100ms e consequentemente alguns quadros são perdidos e o resultado é fornecido com mais atraso. Isso fez com que não fosse utilizado esse método para comparação dos outros serviços. E a Tabela 4 mostra o resultado final numericamente.

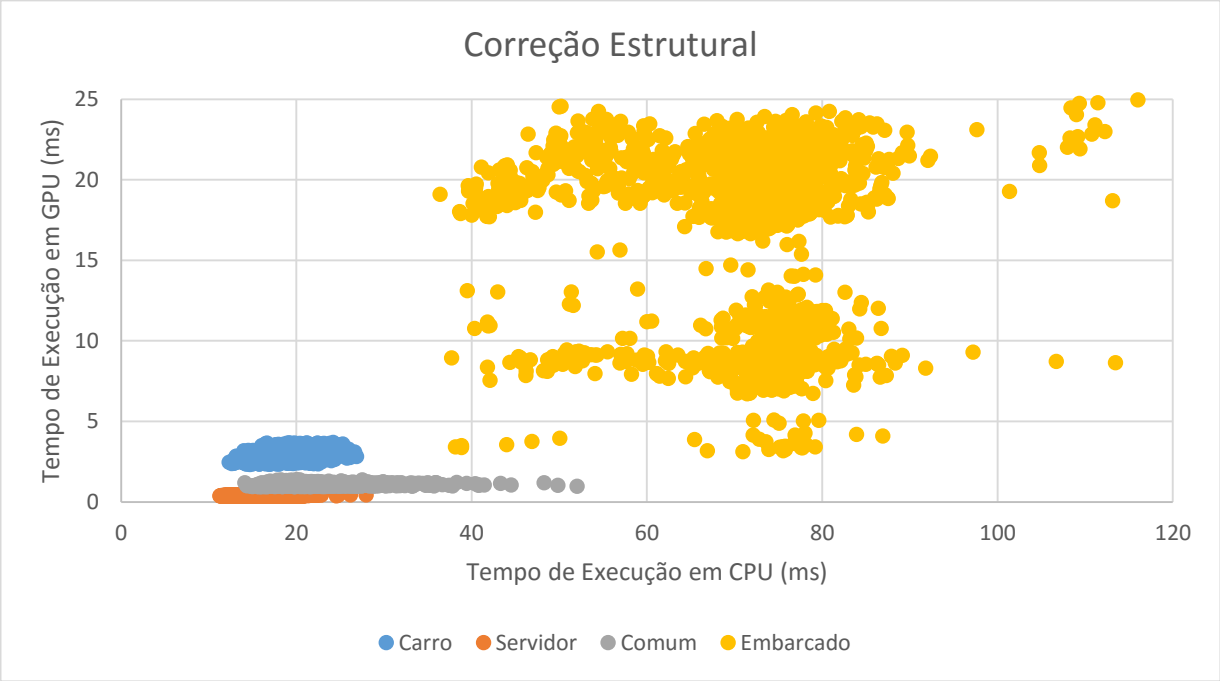

Fig. 10 Correção Estrutural: Comparação de Tempos

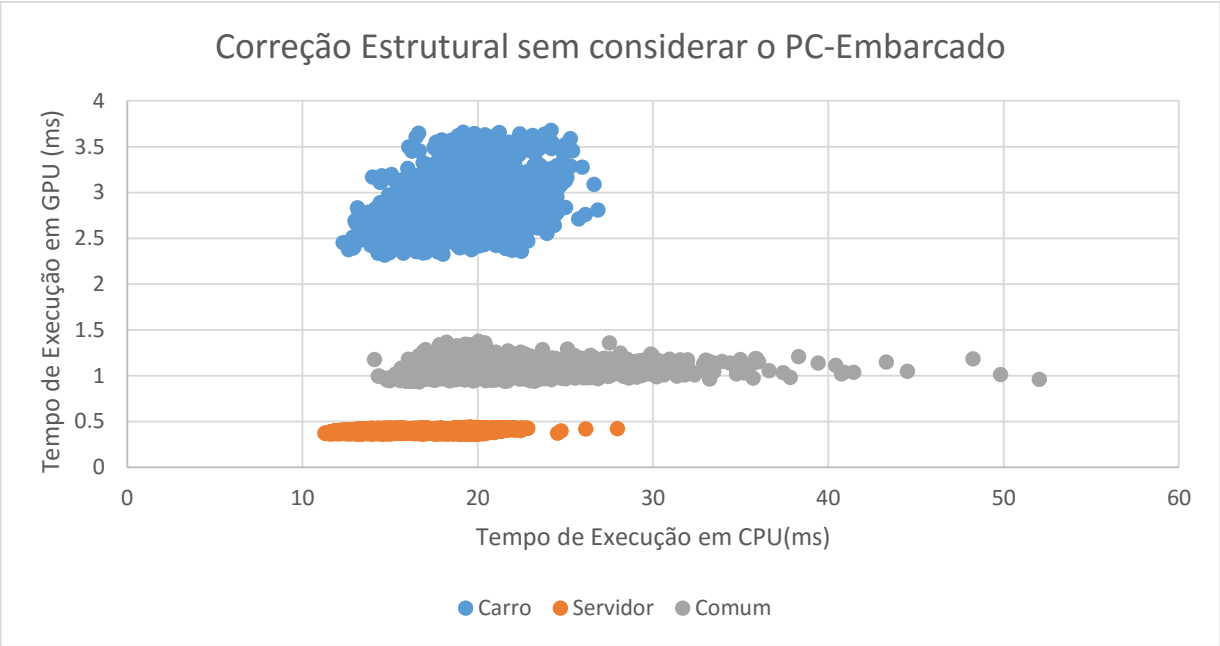

Fig. 11 Correção Estrutural: Comparação de Tempos sem o PC-Embarcado

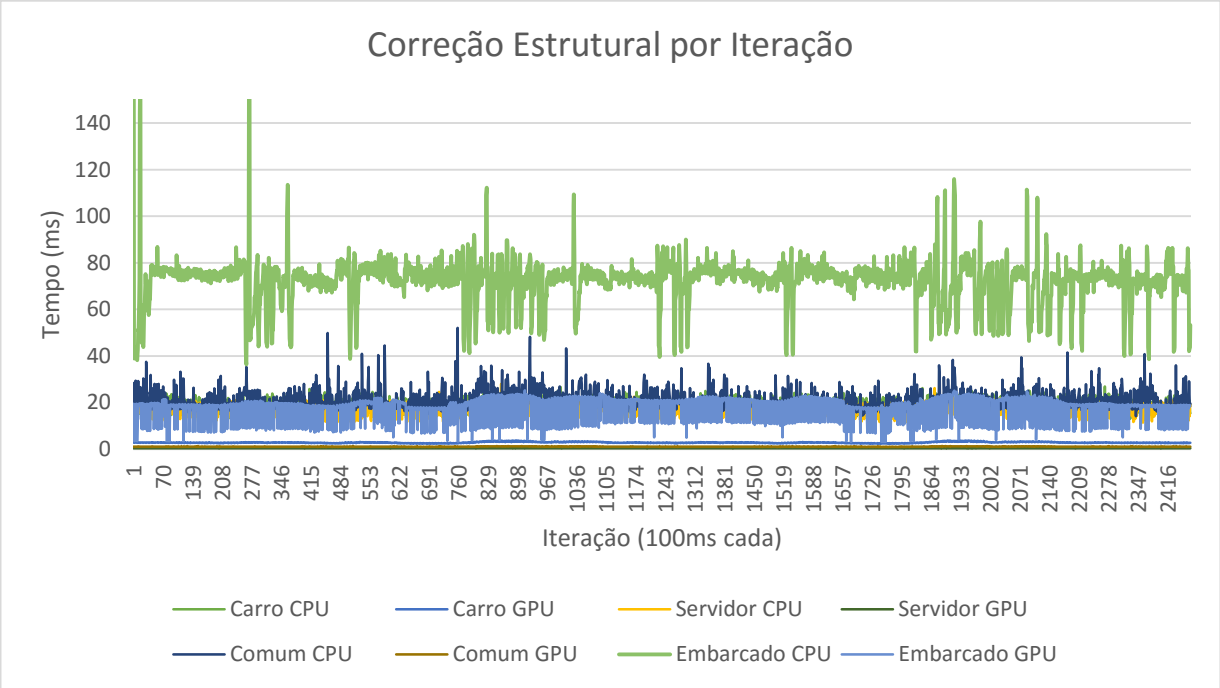

Fig. 12 Correção Estrutural mostrado pelo tempo

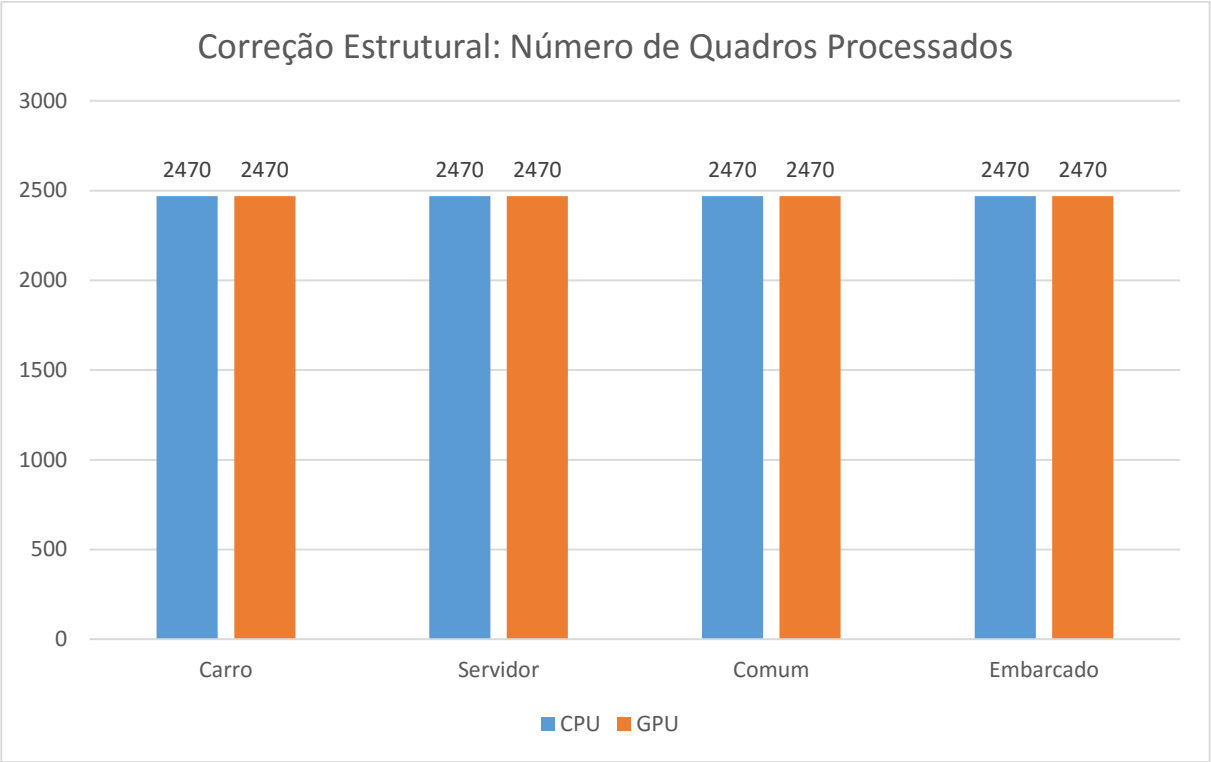

Fig. 13 Correção Estrutural: Número de Quadros Processados

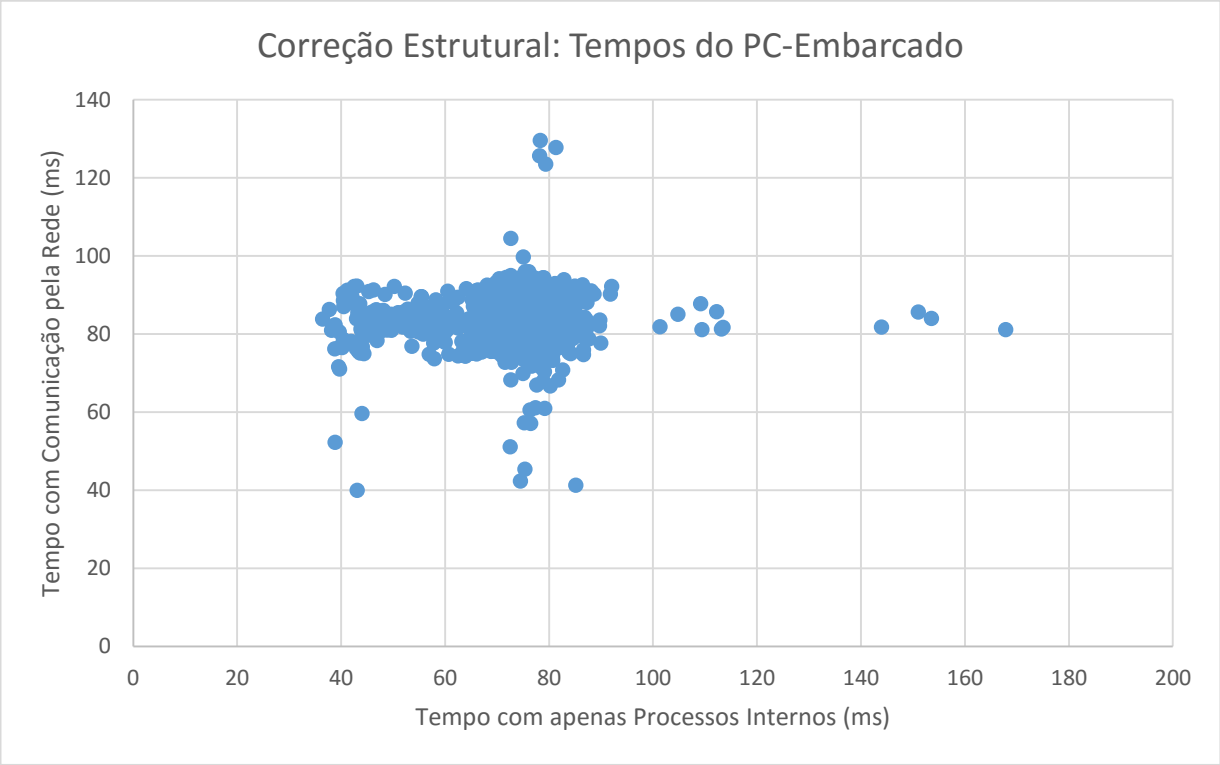

Fig. 14 Correção Estrutural: Tempos do PC-Embarcado com ROSCore e dados remotos ou locais

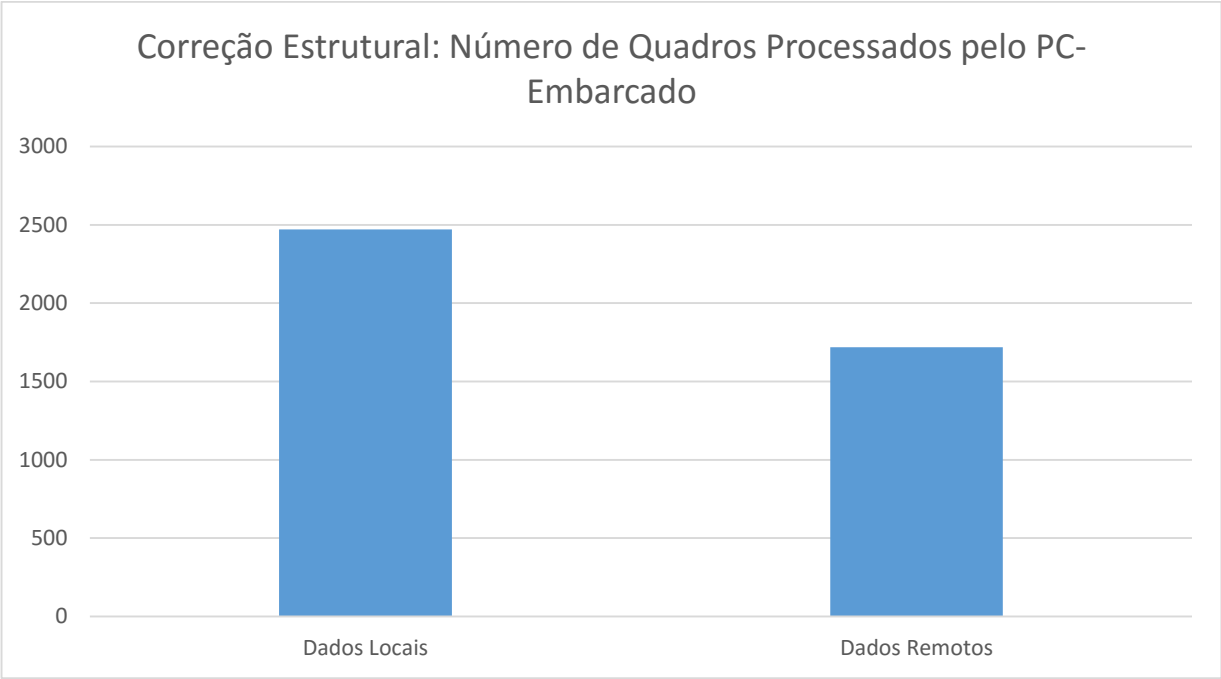

Fig. 15 Correção Estrutural: Número de Quadros Processados pelo PC-Embarcado

Tabela 4 Correção Estrutural: Resultados Numéricos Finais

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	12.3	2.3	11.3	0.4	14.1	0.9	36.4	3.1
Med	18.9	2.8	17.6	0.4	20.9	1.1	72.5	17.7
Max	26.9	3.7	28.0	0.4	52.0	1.4	390.7	25.2

Considerando as avaliações dos tempos dos resultados de correção estrutural acima apresentadas, fica claro que: o PC-Embarcado (Jetson) não possui uma capacidade de processamento adequada para atender a demanda necessária de tratamento dos dados fornecidos pelo Velodyne (100ms por frame); apesar de não atender a demanda de processamento o PC-Embarcado permitiu demonstrar que o processamento em GPU tem um ganho significativo em relação ao processamento apenas em CPU neste dispositivo; a implementação da correção estrutural em CPU e em GPU (exceto no PC-Embarcado), obteve bons resultados, aceitáveis em termos de desempenho; contudo, o desempenho em GPU foi significativamente melhor em todos os testes realizados, apresentando uma performance que permite ter uma "sobra" de tempo para executar outras tarefas.

4.1.2. Serviço de Correção de Distorção Angular

O sensor é preso a uma base fixa do carro autônomo que não é paralela ao plano do chão como exemplificado pela Fig. 16. Esse erro causa uma distorção angular de cerca de 10º, que somado ao erro devido à imperfeição da rua e ao amortecimento durante o deslocamento que, em testes práticos, mostrou ser cerca de 5º a -5º, um ponto a 10 metros que deveria estar com altura igual ao do chão como próximo do veículo acaba possuindo uma altura de quase 2 metros.

Fig. 16 Erro Angular introduzido por não paralelidade da base do sensor com o chão

Para evitar sérios problemas de má classificação, um serviço de correção angular dos dados recebidos foi escolhido como segundo serviço a ser desenvolvido. Ele usa o resultado do primeiro serviço em um algoritmo que calcula as médias das regiões de pontos em frente e atrás do veículo (duas matrizes de 900x6, representando uma abertura de 90º graus e 6 anéis de profundidade entre o anel 3 ao 8 como mostrado na Fig. 17). Para o cálculo é considerado que o sensor está em um ambiente veicular típico, onde as regiões próximas do veículo fazem parte de uma superfície plana, o chão, sendo usada como referência. Com o cálculo dos dois pontos de referência, é realizada a correção angular por transformação geométrica ao calcular e multiplicar a matriz de rotação com cada ponto da nuvem.

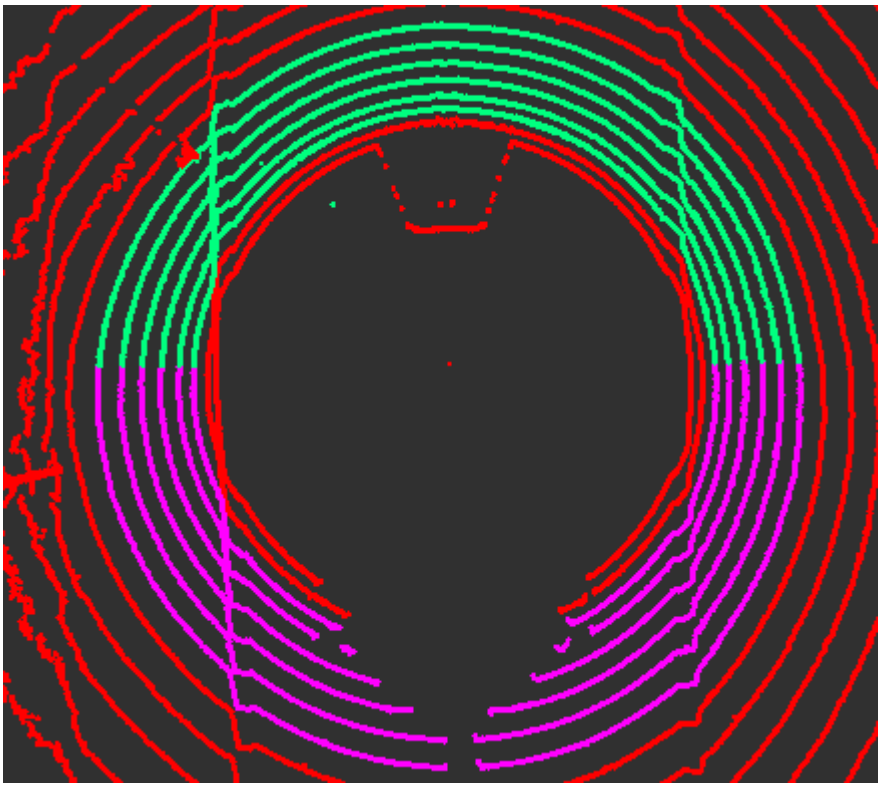

Fig. 17 Demostração das Regiões para Cálculo das Referências para a Matriz de Transformação (os 6 anéis em destaque são os considerados no cálculo e calibragem dos dados do sensor)

Assim como como foi feito no primeiro serviço, primeiramente foi desenvolvida uma implementação em Python e Numpy, foi então estudada a forma de paralelização e então desenvolvido o método em GPU. A paralelização usada foi o cálculo da transformação geométrica (a matriz de rotação) e a aplicação da transformação aos 32x1800 pontos da nuvem. Foi deixado para o CPU o trabalho de encontrar o ângulo de distorção, pois demonstrou ser um processo rápido e eficiente, não necessitando da GPU para aceleração do tempo de processamento (em média de 1ms). Os resultados estão detalhados a seguir na Fig. 18. Os resultados das diferentes implementações e dos diferentes hardwares são mostrados na Fig. 19, onde novamente cada eixo é um tipo de execução (tempo em CPU x tempo em GPU) e com melhor comparação na Fig. 20. O número de quadros processados é mostrado na Fig. 21. E a Tabela 5 mostra o resultado final numericamente.

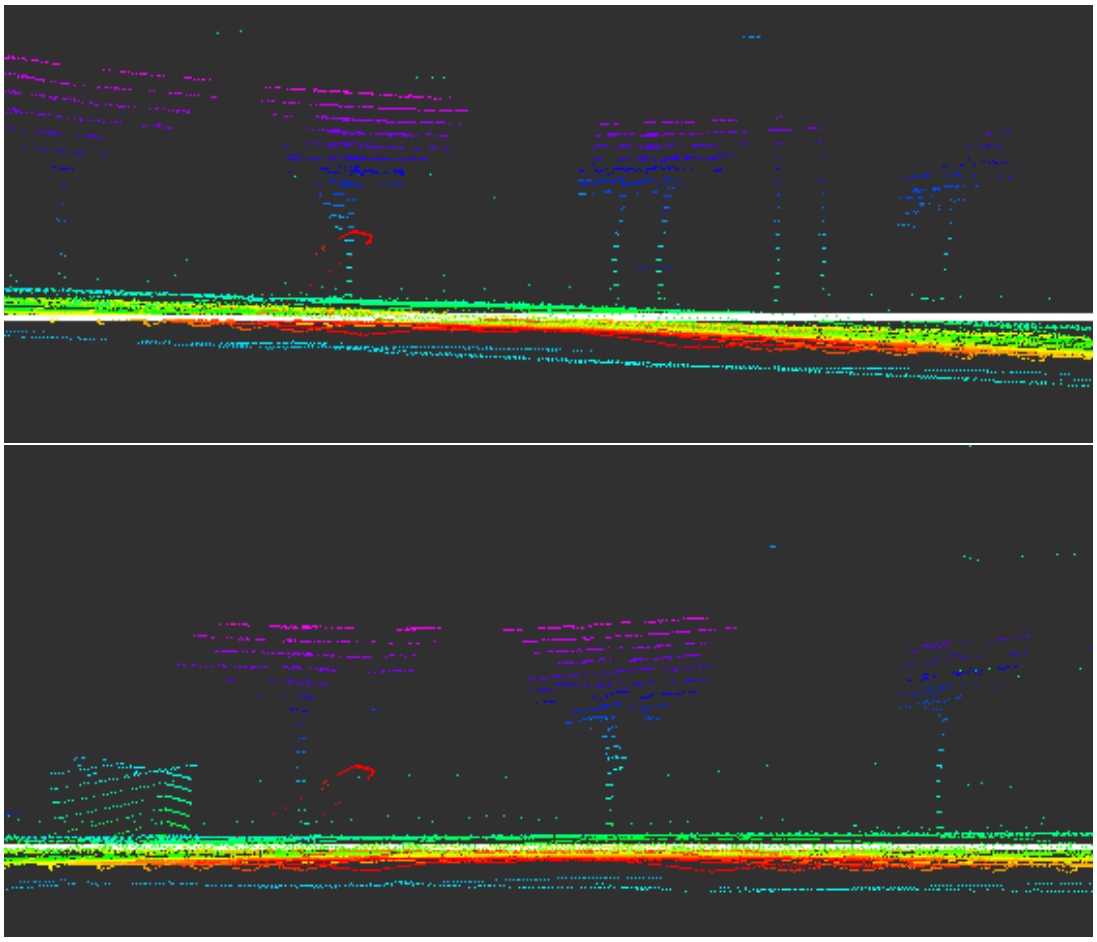

Fig. 18 Resultado da Correção de Distorção Angular: (a) Antes, (b) Depois

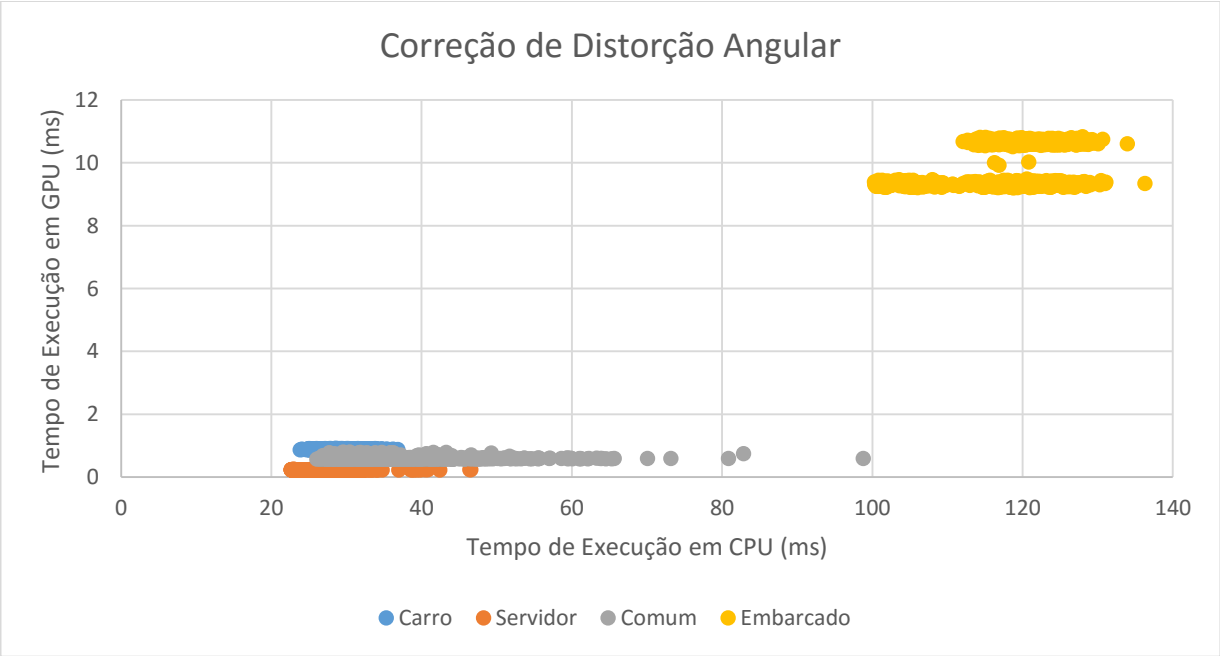

Fig. 19 Correção de Distorção Angular: Comparação de Tempos

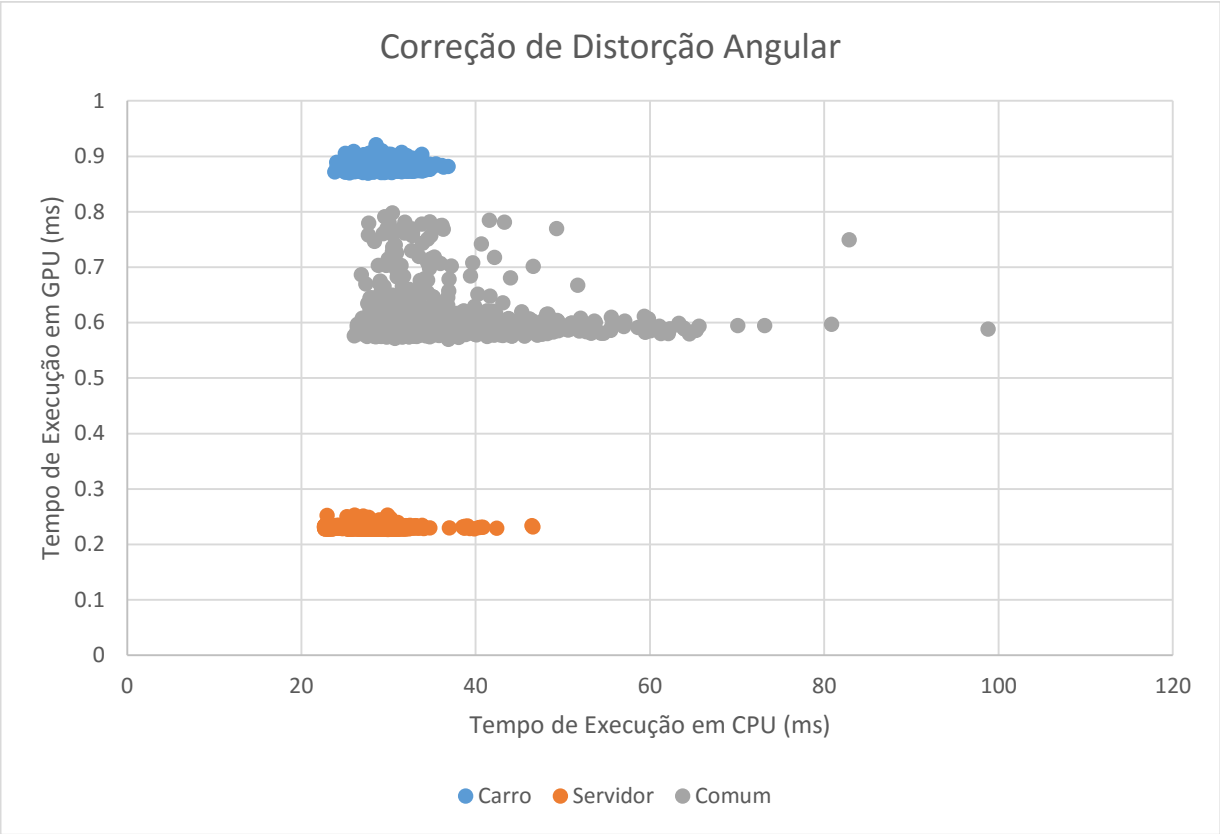

Fig. 20 Correção de Distorção Angular: Comparação de Tempos

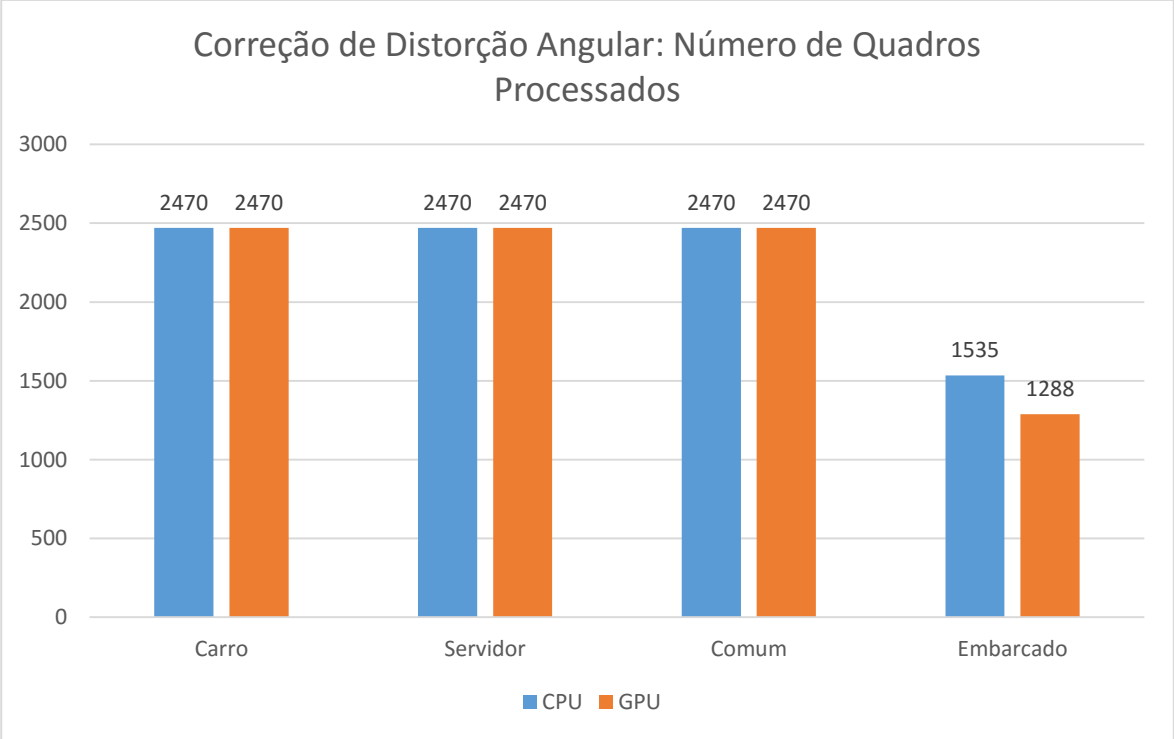

Fig. 21 Correção de Distorção Angular: Número de Quadros Processados

Tabela 5 Correção de Distorção Angular: Resultado Numérico Final

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	23.83	0.87	22.64	0.23	26.05	0.57	100.26	9.20
Med	29.63	0.88	28.40	0.23	33.40	0.60	117.91	9.92
Max	36.84	0.92	46.56	0.25	98.78	0.80	317.90	10.83

Conforme pode ser constatado junto a Fig. 19, o tempo de execução em GPU foi de menos de 2 ms (exceto para o PC-Embarcado, que consumiu entre 9 e 11 ms). Por outro lado, o tempo de execução em CPU mostrou-se significativamente superior, entre 20 e 100ms (exceto para o PC-Embarcado). No caso do PC embarcado ficou sempre acima de 100 ms. Também, da mesma forma que o serviço anterior, o agrupamento dos tempos em GPU é bem melhor que os em CPU. Conclui-se que a aceleração por GPU permite obter um ganho importante em termos de performance, viabilizando o processamento dos dados em taxas bem superiores a taxa de aquisição dos dados.

4.2. Segmentação de zonas

Usando os resultados obtidos com os serviços prévios, temos uma estrutura matricial com a nuvem de pontos alinhada ao plano do chão. O próximo passo é prover serviços capazes de dividir o grande conjunto de dados para detecções locais de elementos. Os dados são divididos em regiões menores permitindo especialização e aceleramento dos próximos serviços.

4.2.1. Segmentação do Chão

O primeiro serviço de segmentação de zonas é o segmentador de chão. Ele tem o objetivo de separar o plano do chão dos planos superiores e inferiores ao chão, caracterizando áreas perigosas de tráfego. Essa etapa é essencial para segmentação de dados em um sensor Velodyne (DOUILLARD, UNDERWOOD, *et al.*) e será usada extensivamente pelos outros serviços. A técnica usada é de segmentação por limiar na variável Z (altura) da nuvem de pontos, recebendo por parâmetro a faixa dos valores que corresponderia ao chão. O resultado do serviço são quatro matrizes booleanas para indexação dos pontos acima do chão, no chão, abaixo do chão e pontos vazios (caracterizados por 0 em todas as variáveis). Neste projeto foi usado um Velodyne a cerca de dois metros de altura do chão, como mostrado na Fig. 22.

Fig. 22 Demostração da Altura do Sensor

O serviço foi desenvolvido inicialmente em Python e Numpy usando o resultado do serviço de correção de distorção angular. Posteriormente foi implementado em GPU, que mostrou ser bem eficaz em sua paralelização por possuir bons resultados. Este serviço analisa o componente de altura classificando os pontos próximos do chão ao usar um filtro por limiar. Para este serviço, a altura do sensor é tratada como 1,8 m, com cerca de 20 cm em torno dessa marca para classificado pertencente à zona do chão, no entanto estes parâmetros podem ser ajustados e calibrados conforme a necessidade do usuário. Os resultados são detalhados nas Fig. 23, Fig. 24, Fig. 25 e Fig. 26 e por análise dos gráficos apresentados podemos chegar a conclusões similares as obtidas nos serviços anteriores: grandes melhoras no tempo e no agrupamento ao usar GPU em comparação com CPU. Nota-se que o tempo máximo de processamento em GPU foi de 1ms (excluindo o PC-Embarcado), sendo de 2 até 6 vezes mais rápido que o processamento em CPU. E a Tabela 6 mostra o resultado final numericamente.

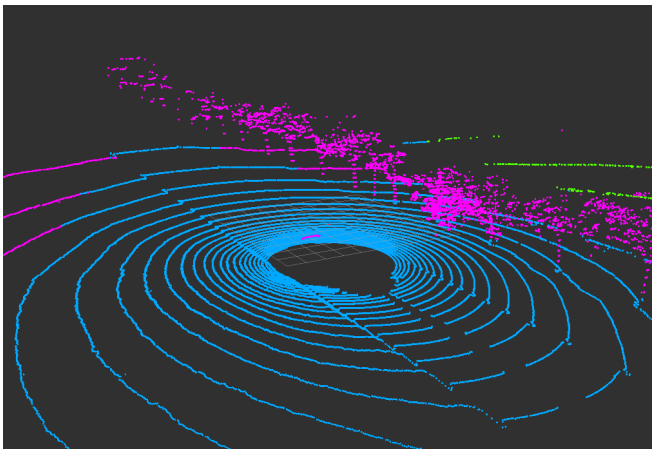

Fig. 23 Segmentação do Chão: chão em azul, acima em roxo e abaixo em verde

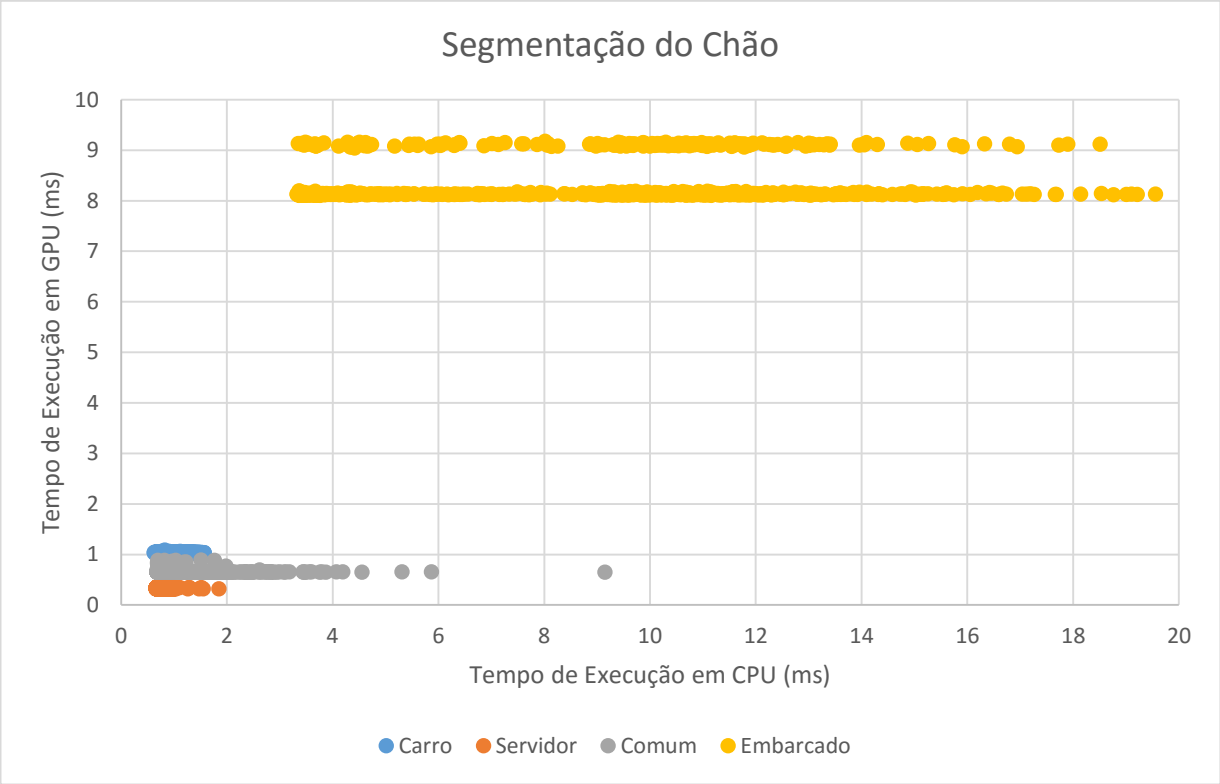

Fig. 24 Segmentação do Chão: Comparação de Tempos

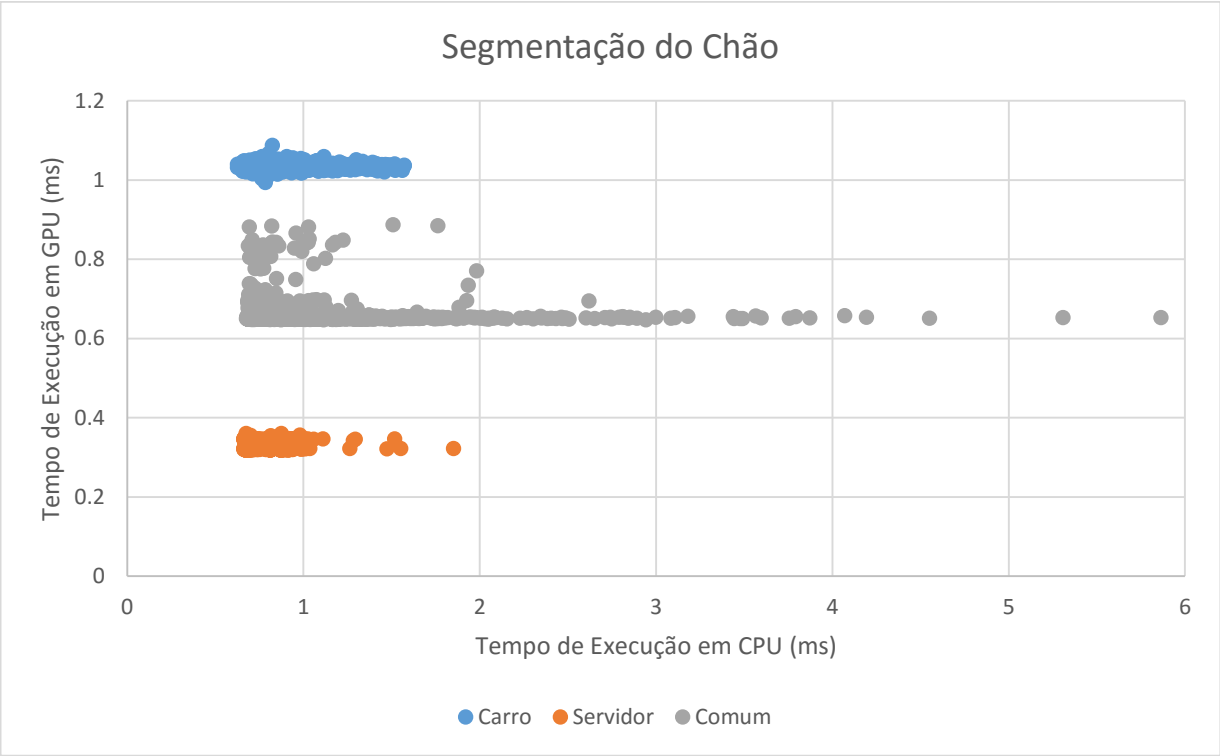

Fig. 25 Segmentação do Chão: Comparação de Tempos sem o PC-Embarcado

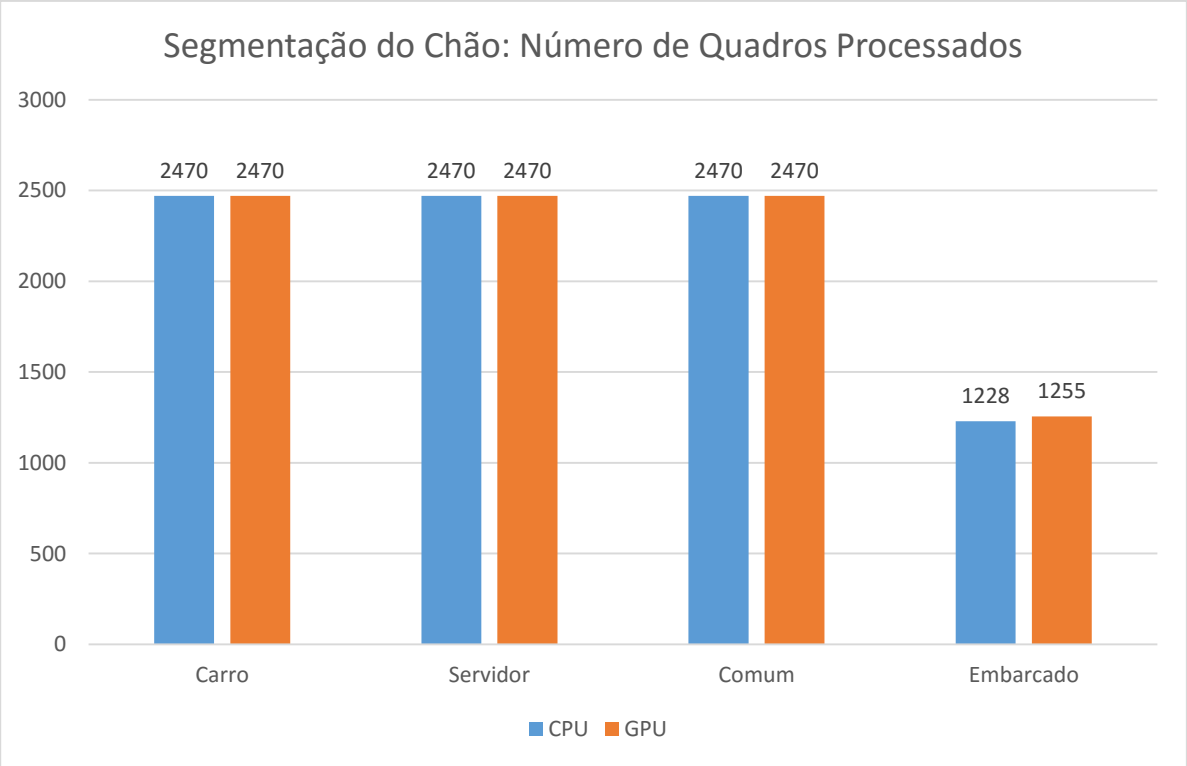

Fig. 26 Segmentação do Chão: Número de Quadros Processados

Tabela 6 Segmentação do Chão: Resultado Numérico Final

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	0.63	0.99	0.66	0.32	0.68	0.65	3.33	8.11
Med	0.86	1.04	0.73	0.33	0.94	0.66	8.24	8.31
Max	1.57	1.09	1.85	0.36	9.15	0.89	22.94	9.18

4.2.2. Segmentação Fina de Guias

Com objetivo de ser usado pelos projetos de sistemas de controle e de planejamento de trajetória, este serviço é responsável pelo segundo segmentador de zona. Ele usa como referência dois sistemas de busca de guias: o método de detecção por compressão de anéis (BUEHLER, IAGNEMMA e SINGH, 2009), usado pelo carro autônomo JUNIOR, que participou do DARPA Urban Challenge, conforme descrito na seção 2.1.2, e o sistema de detecção de regiões planares usado pelo carro autônomo BOSS (BUEHLER, IAGNEMMA e SINGH, 2009), ganhador do DARPA Urban Challenge, conforme indicado na seção 2.1.2.2. O primeiro método mostra que dada uma área plana, como uma rua, um objeto que esteja sobre essa região plana pode ser identificado pelo fator de compressão dos anéis do sensor, sendo detectado por comparação aos anéis próximos. Um uso dessa técnica pode ser visto no trabalho de Alberto Hata (HATA, OSÓRIO e WOLF, 2014) e descritos do trabalho de Hillel (HILLEL, LERNER, *et al.*, 2012). O outro sistema se baseia no uso de convoluções do dado de intensidade de reflexão recebido do sensor com uma função degrau, visto que as guias costumam ter um índice de reflexão diferente do asfalto.

A partir dessas fontes e de outros trabalhos relacionados, algumas metodologias diferentes foram desenvolvidas para a segmentação de guias com objetivo de ser paralelizáveis em GPU. Utilizando a metodologia proposta na seção 2.2.3, as soluções implementadas foram comparadas a fim de avaliar o seu desempenho, tempo de processamento e carga computacional. O sistema com melhor eficiência e que melhor se adaptar a processo de paralelização, foi projetado para GPU. Os métodos desenvolvidos foram:

1. Detectar variações relevantes na hipotenusa dos pontos entre duas radiais (Fig. 27). Neste método foram usadas as distâncias planares (x e y) de cada ponto para o cálculo da distância da origem, e após isso normaliza-se pela média esperada da altura do anel. Essa média é calculada a partir da região próxima ao ponto em questão (região final: 5 pontos ao redor do ponto principal em seu próprio anel). Seu resultado pode

ser visto na Fig. 28, com quatro valores possíveis: região planar em verde, região com alta variação em violeta, região com extrema variação indicando possíveis erros de leitura ou regiões com plantas em vermelho e regiões intermediárias em azul.

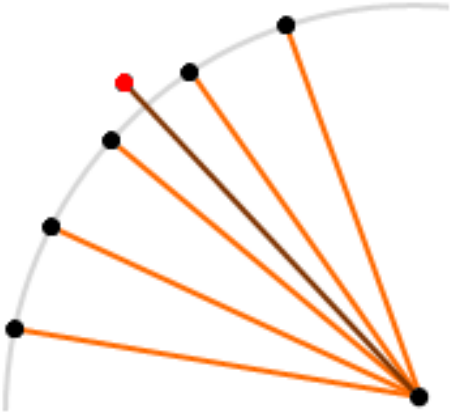

Fig. 27 Detecção de variações na hipotenusa dos pontos entre duas radiais

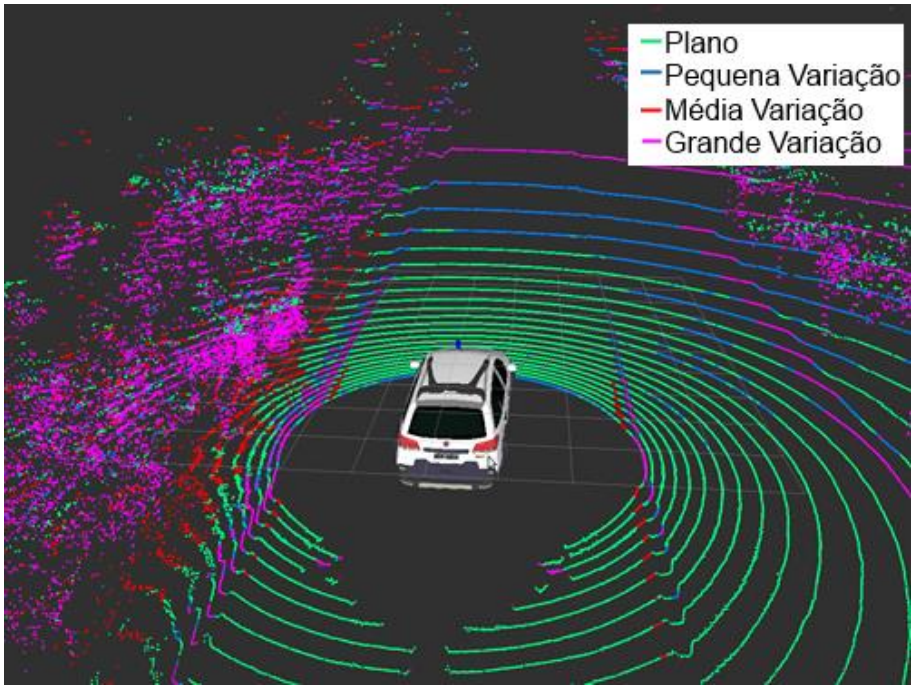

Fig. 28 Visualização com o RViz da segmentação das guias com os pontos com os pontos indentificados como guia pela variação de hipotenusa

2. Detecção de variação do ângulo entre duas radiais por lei dos senos (Fig. 29), que é feito através da detecção da diferença entre h e h2, as quais são calculadas através de α & d1 e d2 & β respectivamente. Este método utiliza os ângulos considerando os

pontos e suas distâncias planares (X e Y) normalizadas pelo índice de anel (que simbolizam uma hipotenusa) e depois divide-se pela altura (Z) do ponto pelo seno do valor normalizado. O resultado na Fig. 30 possui a coloração dos três valores possíveis resultantes da detecção: pontos verdes são planares, pontos violeta com alta variação positiva, e vermelho com alta variação negativa.

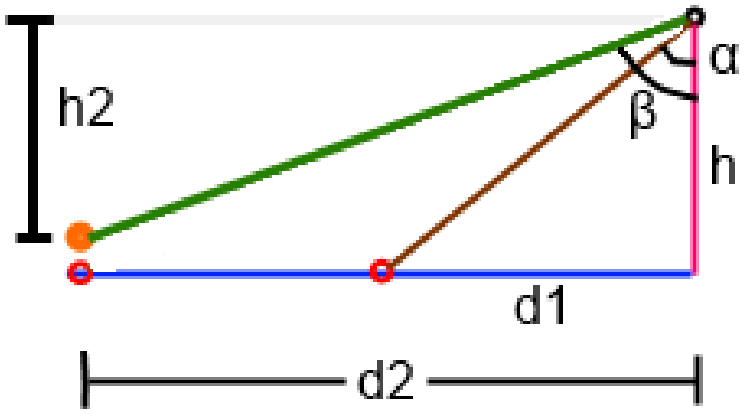

Fig. 29 Detecção de variação do ângulo entre duas radiais por lei dos senos

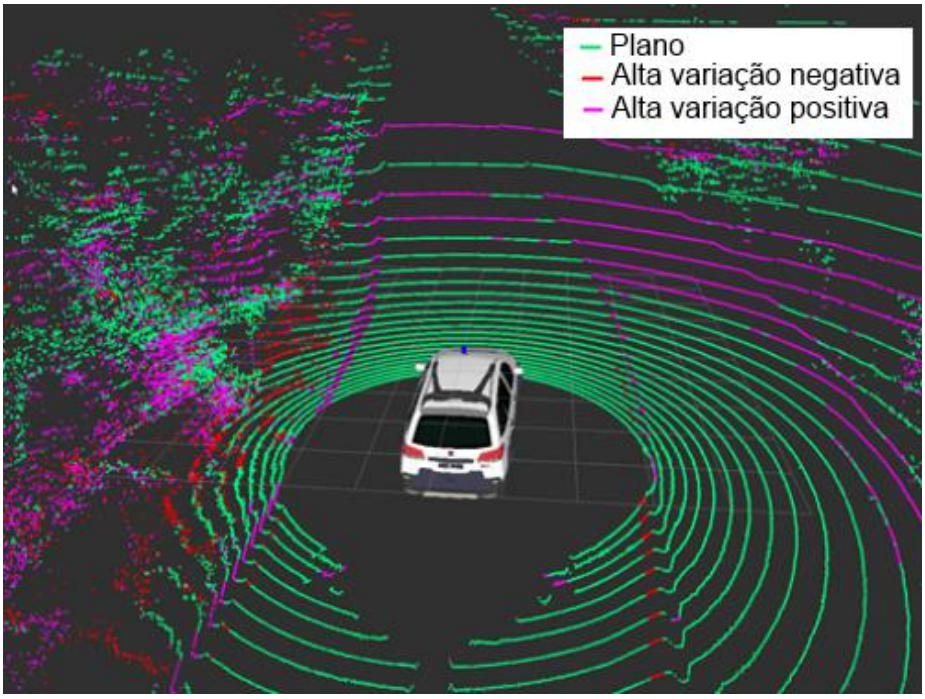

Fig. 30 Visualização com o RViz com os pontos indentificados como guia pela variação de tangente

3. Variação de intensidade dos pontos (Fig. 31). Neste método foi usada a variação da intensidade de reflexão dos pontos com relação à mediana do anel do ponto. Três valores foram possíveis, próximo à mediana de verde, acima de violeta, e abaixo de vermelho. Essa técnica foi usada por ter sido identificado que a reflexão da via asfaltada é próxima da mediana, e há variação significante com relação à guia e as marcações no asfalto. Tais variações podem ser interessantes para a realização de trabalhos futuros que venham a considerar tais marcações. O resultado está apresentado na Fig. 31 com os três valores que foram possíveis de serem destacados, próximo à mediana de verde, acima de violeta, e abaixo de vermelho.

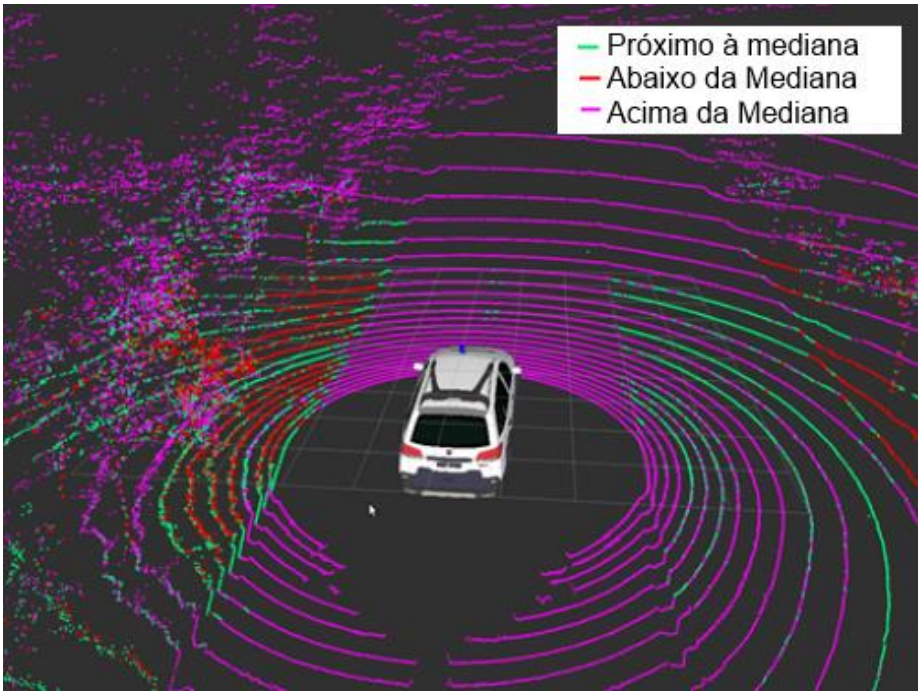

Fig. 31 Visualização com o RViz com os pontos indentificados como guia pela variação de intensidade

4. Aplicando um filtro de convolução usando o filtro de Sobel (SOBEL, 2014) (ZHANG, GUAN e ZHU, 2012) (Fig. 32), também foi possível detectar as guias (bordas). Esse método utiliza mascaras da técnica Sobel[31] de processamento de imagens. Essa técnica utiliza duas matrizes de convoluções, identificando variações no eixo x e no eixo y, e então tomando a média geométrica dos dois resultados. No caso da nuvem de pontos, ele identificará bordas na matriz de 1800 colunas por 32

[31] Referência da técnica Sobel: http://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.filters.sobel.html

anéis. Regiões violetas e vermelhas foram as identificadas como bordas, e regiões verdes como não bordas.

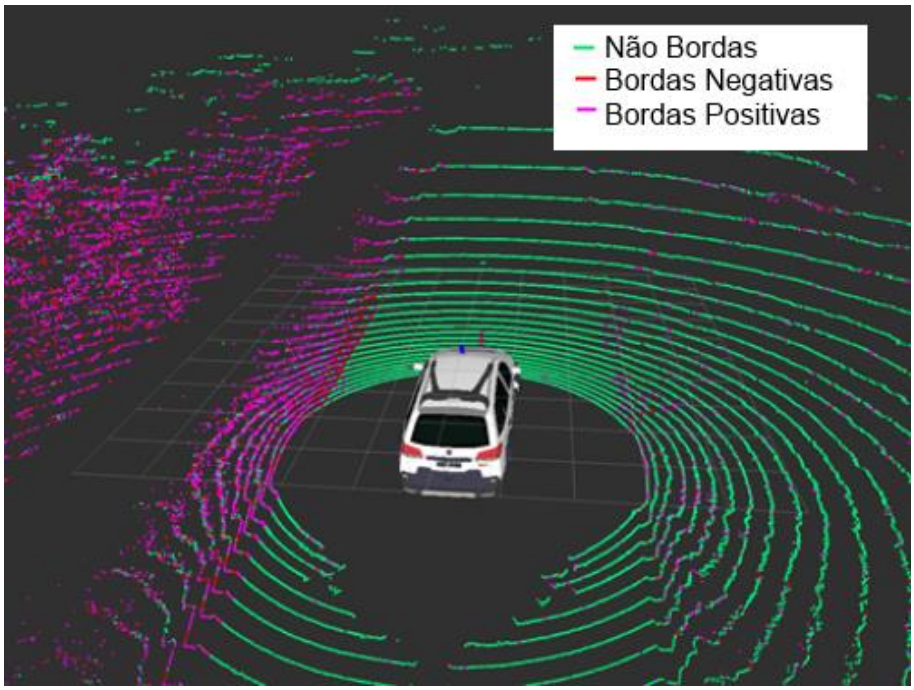

Fig. 32 Visualização com o RViz com os pontos indentificados como guia pelo filtro de convolução

As quatro soluções acima foram desenvolvidas em CPU e, em seguida, foram comparadas a fim de avaliar o seu desempenho, cargas computacionais e, em particular, a sua precisão. O método que apresentou a melhor eficiência, eficácia e que possui a melhor arquitetura para o processo de paralelização foi escolhido para ser desenvolvido em GPU. O método adotado foi a primeira técnica (hipotenusa) que posteriormente é filtrado de valores conhecidamente errôneos devido à interferência de outros sensores acoplados perto do sensor Velodyne. A saída do serviço é composta por uma classificação dos pontos em três categorias: pontos planares (exemplos: rua e calçada), pontos com pequena variação (exemplos: guia da rua), e com alta variação (exemplos: erros de leitura e árvores).

Para a paralelização do método, foi realizada uma análise ao dividir o processamento de cada ponto para uma *Thread* diferente. Para este processamento, cada ponto necessita da sua distância para o ponto do anel seguinte (ângulo, anel+1) da matriz e também da mediana dos 5 pontos próximos no mesmo anel (de [ângulo-2, anel] até [ângulo+2, anel]). Para uma correta paralelização dos acessos, o Kernel possui dois acessos à memória lenta para obter os pontos necessários, resolvendo o primeiro problema, e também possui uma função interna extra onde,

por compartilhamento da memória dentro de um *Block*, *Threads* próximas se auxiliam a encontrar suas medianas. Seus resultados e a avaliação dos mesmos podem ser vistos nas Fig. 33, Fig. 34, Fig. 35 e Fig. 36. Nota-se um tempo de processamento de na maioria dos casos 5 a 20ms (podendo atingir 30 ms) em CPU e em GPU de 0,2 a 1,4 ms (excluindo PC-Embarcado). A Tabela 7 mostra o resultado final numericamente.

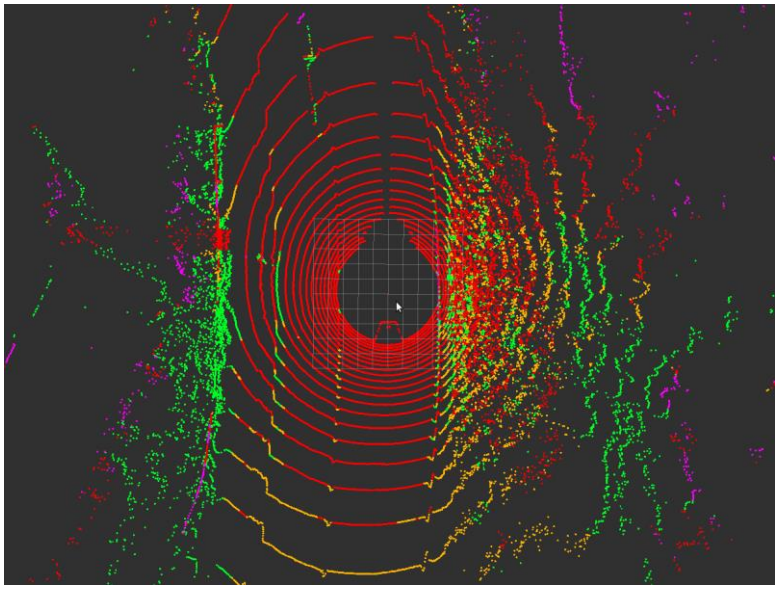

Fig. 33 Visualização com o RViz com os pontos indentificados como guia pelo pelo algorítimo resultante

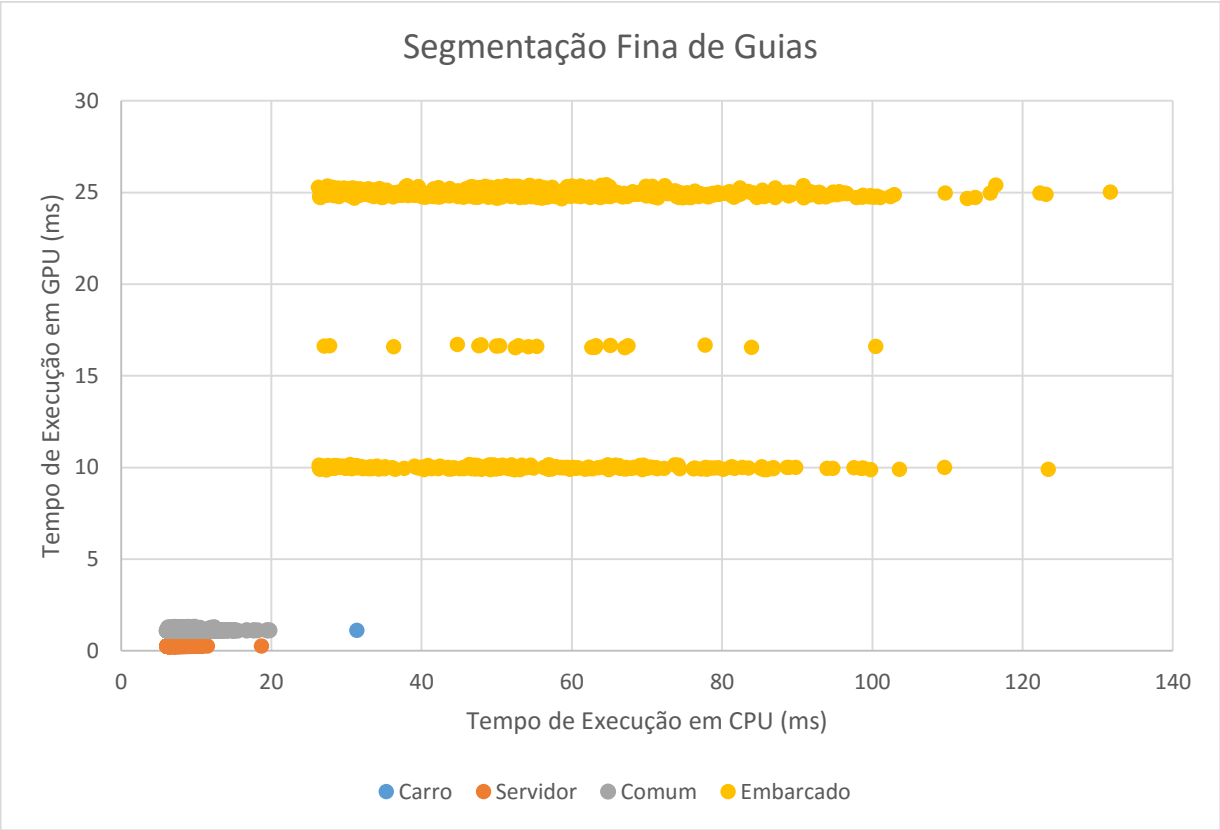

Fig. 34 Segmentação Fina de Guias: Comparação de Tempos

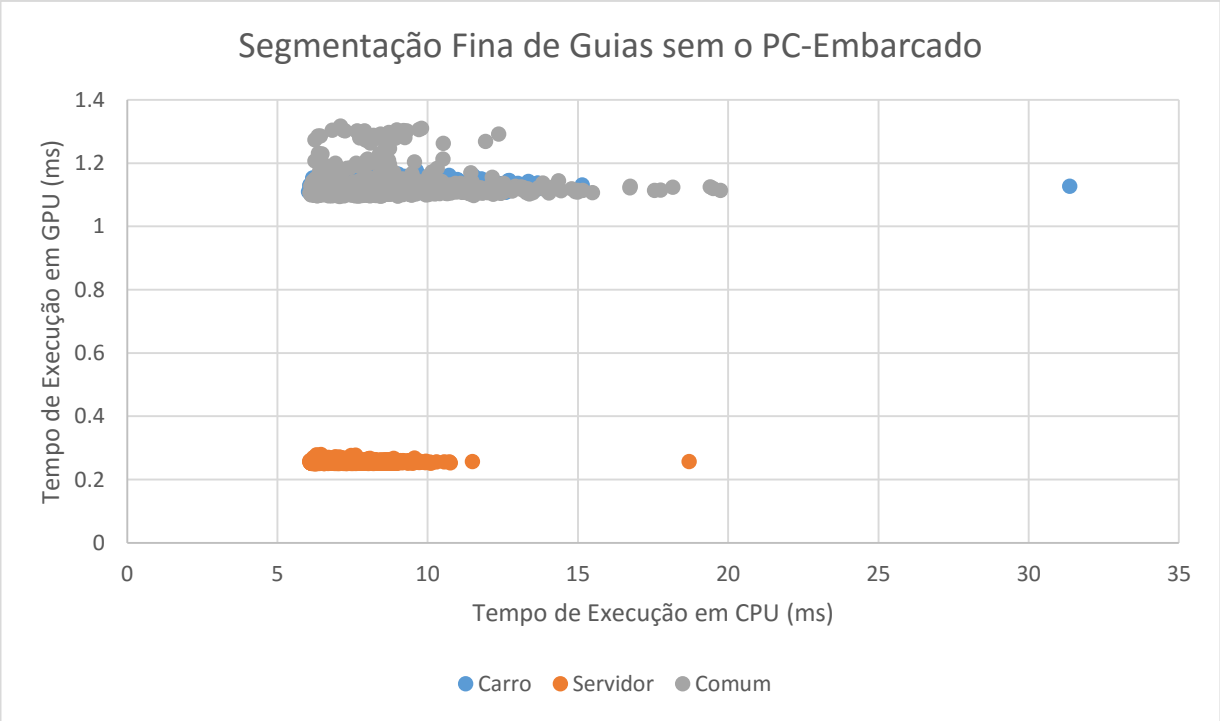

Fig. 35 Segmentação Fina de Guias: Comparação de Tempos sem o PC-Embarcado

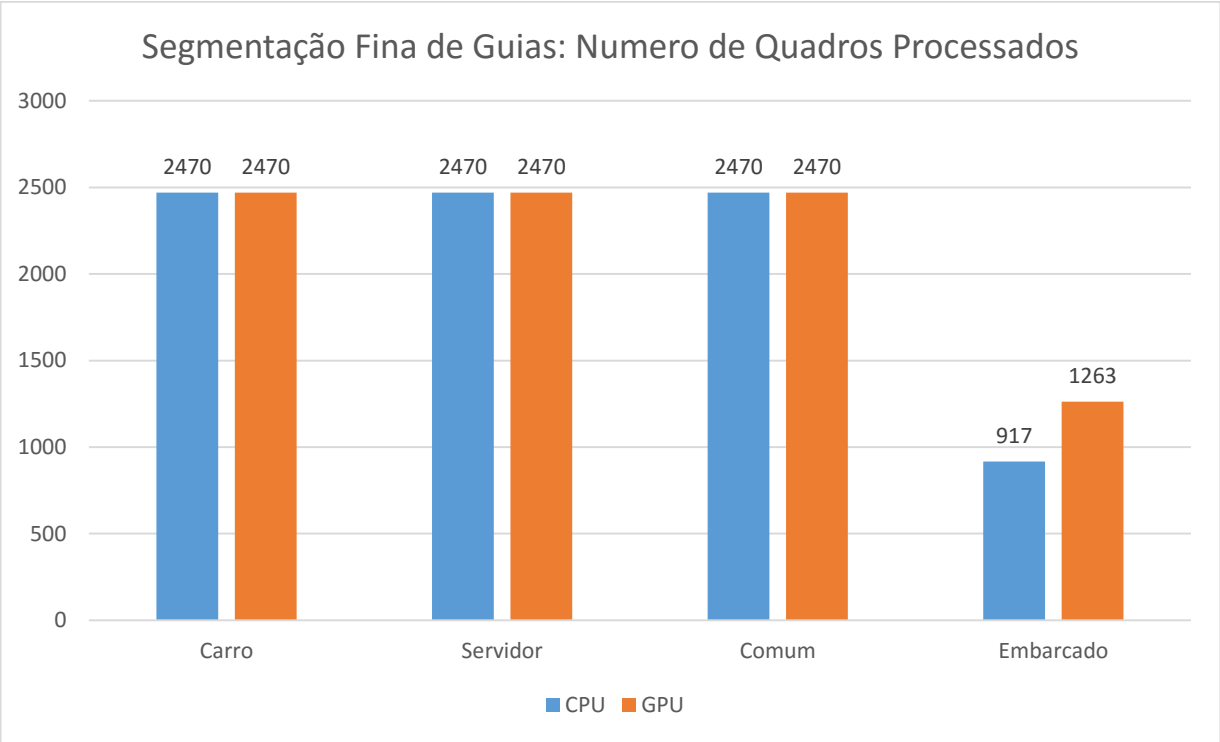

Fig. 36 Segmentação Fina de Guias: Numero de Quadros Processados

Tabela 7 Segmentação Fina de Guias: Resultado Numérico Final

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	6.04	1.10	6.07	0.25	6.09	1.09	26.28	9.87
Med	8.02	1.13	7.11	0.26	8.12	1.12	54.92	21.12
Max	31.37	1.18	18.70	0.28	19.74	1.32	131.68	25.43

4.3. Segmentação de Objetos

O serviço de segmentação permite que projetos e serviços externos utilizem técnicas de classificação e novas metodologias para detectar pedestres, carros e outros objetos que geralmente compõem a cena em torno de um veículo. Esse serviço procura desenvolver formas de paralelamente extrair informações estruturais do ambiente com contornos bem definidos, ou seja, a partir de uma nuvem de pontos procura-se identificar objetos que estejam na cena, ou dentre os pontos da nuvem fornecida. O serviço responde com um índice que identifica qual unidade estrutural ela pertence. Um exemplo de aplicação seria o uso da nuvem de pontos do Velodyne para identificar um poste como diferente do resto da nuvem, como mostrado na Fig. 37. É importante ressaltar que esse serviço tem o objetivo de segmentação, e não de classificação.

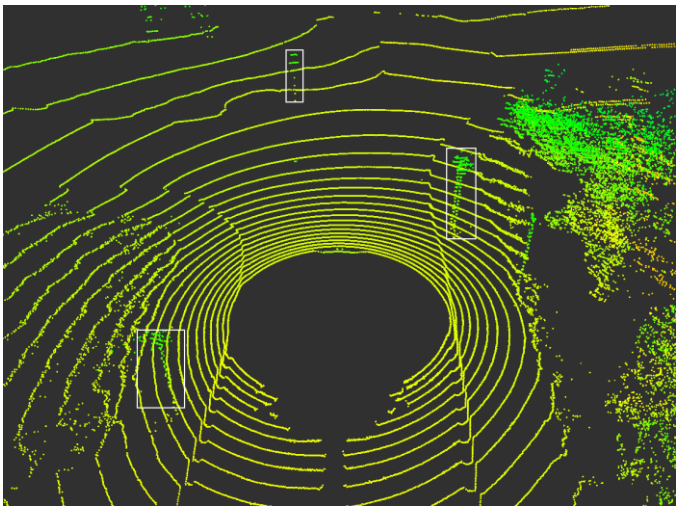

Fig. 37 Segmentação de dados geométricos retirados após a detecção de guia.

Esse serviço é genérico, implementando a técnica DBScan (ESTER, KRIEGEL, *et al.*, 1996) (BÖHM, NOLL, *et al.*, 2009) em GPU, um método de segmentação já bem conhecido pela literatura e já previamente explorado e aplicado em problemas de visão computacional em nuvem de pontos, como nos trabalhos listados na seção 1.3. Seu uso pode ser requisitado passando uma nuvem de pontos que caracteriza o contexto da cena, onde se busca a segmentação dos objetos presentes. Logo, esta etapa é usada para qualquer segmentação do ambiente e consequentemente usada por outros serviços descritos aqui e dependendo de que parte da nuvem de pontos for usada por esse serviço, é possível identificar a localização de obstáculos na rua ou na calçada, utilização posterior para classificação e identificação de objetos, e de localização das guias, que nesse caso será tratado no próximo serviço.

O método é caracterizado por analisar as distâncias entre cada ponto da nuvem e sua execução usa um ponto raiz e repetidamente encontra seus vizinhos com uma distância menor que a especificada, e ao encontrar um objeto passa para outro ponto raiz. Várias versões do algoritmo foram desenvolvidas em CPU em busca de uma boa eficiência. No entanto, para desenvolvimento em GPU o processo se mostrou desafiador e complexo de paralelizar. Foram desenvolvidas implementações que não alcançaram os resultados desejados, os quais possuíam o mesmo tempo de execução do algoritmo feito para CPU ou até pior. O resultado final, porém, se mostrou muito eficiente frente ao processamento original. Perceptivelmente, o algoritmo DBScan não é facilmente paralelizável e requereu certas adaptações para uso em ambiente paralelo que foi dividido em duas etapas:

- Identificação: essa etapa insere um identificador único para cada ponto da nuvem e gera uma matriz de tamanho limitado (devido a limitações de hardware) identificando os pontos vizinhos de cada ponto.
- Numeração: essa etapa é uma etapa de repetição onde o identificador dos pontos vizinhos identificados previamente são modificados tal que represente o valor mínimo dentre o grupo. Portanto, devido a execuções sucessivas, eventualmente toda a nuvem de pontos estará segmentada.

Desta forma foi possível paralelizar e executar o algoritmo DBScan. Os valores de tamanho da matriz de pontos vizinhos e a quantidade de repetições são definidos dependendo do tamanho da nuvem de pontos e a quantidade de objetos diferentes a serem segmentados que é esperada a cada quadro seguindo a Equação 1, onde alfa indica o nível de precisão da segmentação. Dependendo da capacidade do sistema, diferentes configurações são recomendadas, já que um vetor grande implica grande uso de memória e gasto em acessos em memória, onde um grande número de repetições exige uma boa capacidade de processamento ou será lento. Através de testes preliminares, foi observado que uma precisão de 0,1 é o suficiente para identificar a localização de objetos no ambiente com boa precisão. Por exemplo, uma nuvem de pontos com 1000 pontos e cerca de 10 objetos a serem segmentados teria um vetor de pontos vizinhos de tamanho 5 e 2 repetições num nível de precisão 0,1. Após a execução, uma limpeza de dados em grupos de pontos com poucos elementos, caracterizando erros do sensor, é realizada.

$$\frac{Tamanho\ da\ Nuvem}{Quantidade\ de\ Objetos} = \frac{Tamanho\ do\ vetor\ de\ vizinhos\ por\ ponto * Repetições}{\alpha}$$

Equação 1 Cálculo dos valores usados no DBScan paralelo. Alfa indica a precisão da segmentação

O objetivo deste método paralelo foi utilizar o poder computacional da GPU novamente junto com suas vantagens, para extrair objetos da cena sem se preocupar com a semântica associada ao objeto. Para melhora do uso da GPU, os dados intermediários entre as duas etapas e entre as repetições da segunda etapa não foram transferidos de volta para a CPU, permitindo que fosse alcançado um tempo de execução condizente com a aplicação.

O serviço desenvolvido possui um algoritmo tão versátil quanto o DBScan original, permitindo qualquer nuvem de pontos como entrada e usando os parâmetros do método para customização do processo de detecção, porém ele possui o auxílio da placa de vídeo,

aumentando consideravelmente sua eficiência. Aplicando este serviço no contexto de segmentação de objetos no ambiente veicular, utiliza-se como entrada a nuvem de pontos resultante do serviço de segmentação de chão e, utilizando os pontos acima do chão, o serviço fornece uma nuvem de pontos classificados como *blobs*, que são objetos acima do chão no ambiente próximo do veículo. O objetivo nesse caso é fornecer uma forma de diferenciar e listar os objetos para outros projetos de fusão sensorial, classificação por *machine learning* dos objetos, *object tracking*, entre outros. Os resultados são mostrados na Fig. 38, Fig. 39, Fig. 40 e Fig. 41. Nota-se um tempo de processamento de na maioria dos casos 250 a 3000ms em CPU e em GPU de 10 a 120ms (excluindo PC-Embarcado), indicando uma melhora significativa de eficiência. A Tabela 8 mostra o resultado final numericamente.

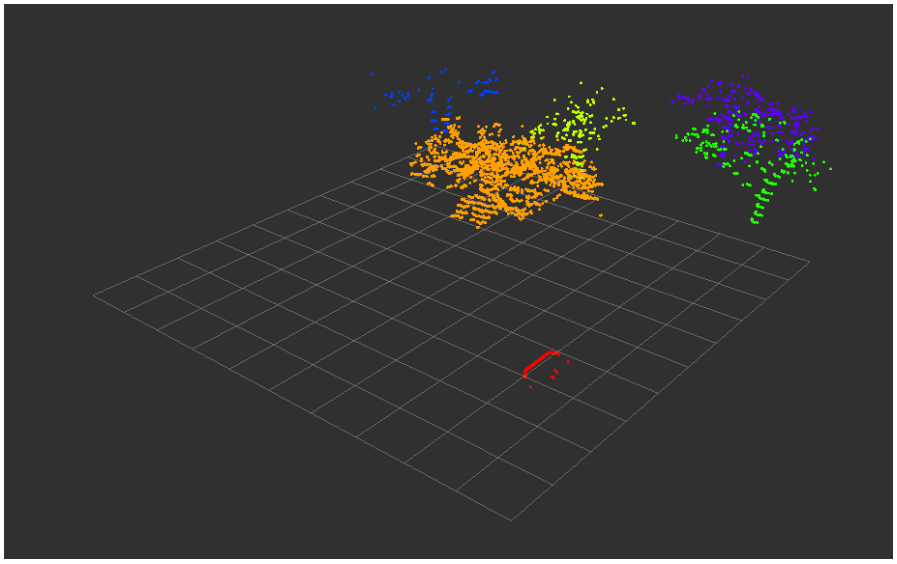

Fig. 38 Segmentação de Objetos demonstrado por cores

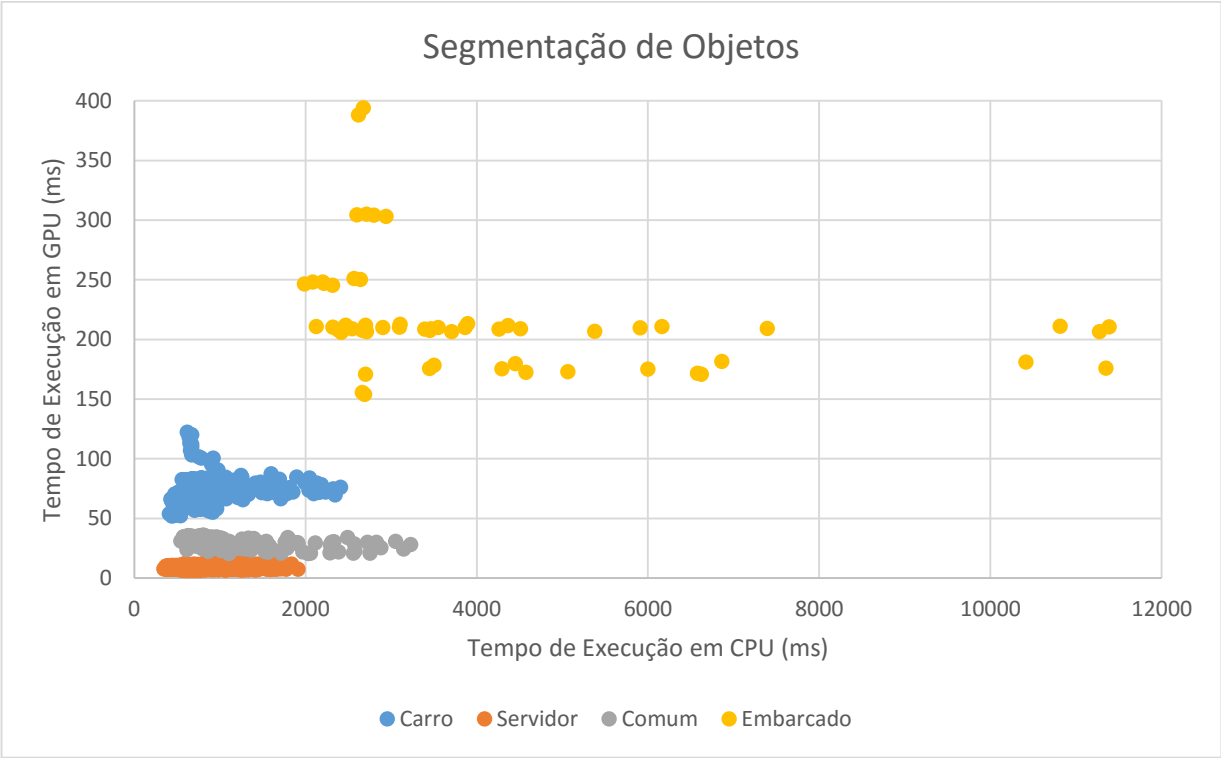

Fig. 39 Segmentação de Objetos: Comparação de Tempos

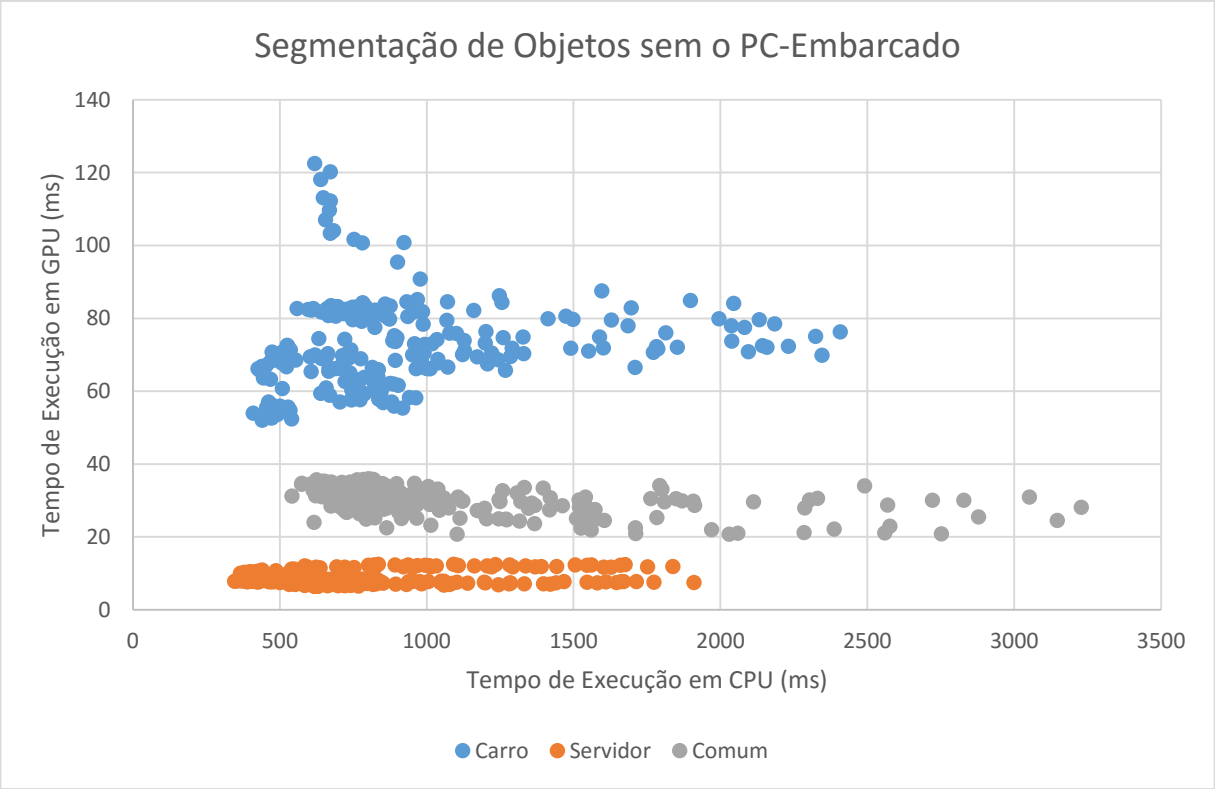

Fig. 40 Segmentação de Objetos: Comparação de Tempos sem o PC-Embarcado

Tabela 8 Segmentação de Objetos: Resultado Numérico Final

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	409.24	47.23	346.29	5.83	541.82	18.51	1988.31	101.61
Med	919.67	106.96	761.57	12.33	1114.43	50.96	4162.99	237.77
Max	2408.49	401.77	1910.67	33.89	3229.05	164.20	11385.15	1348.24

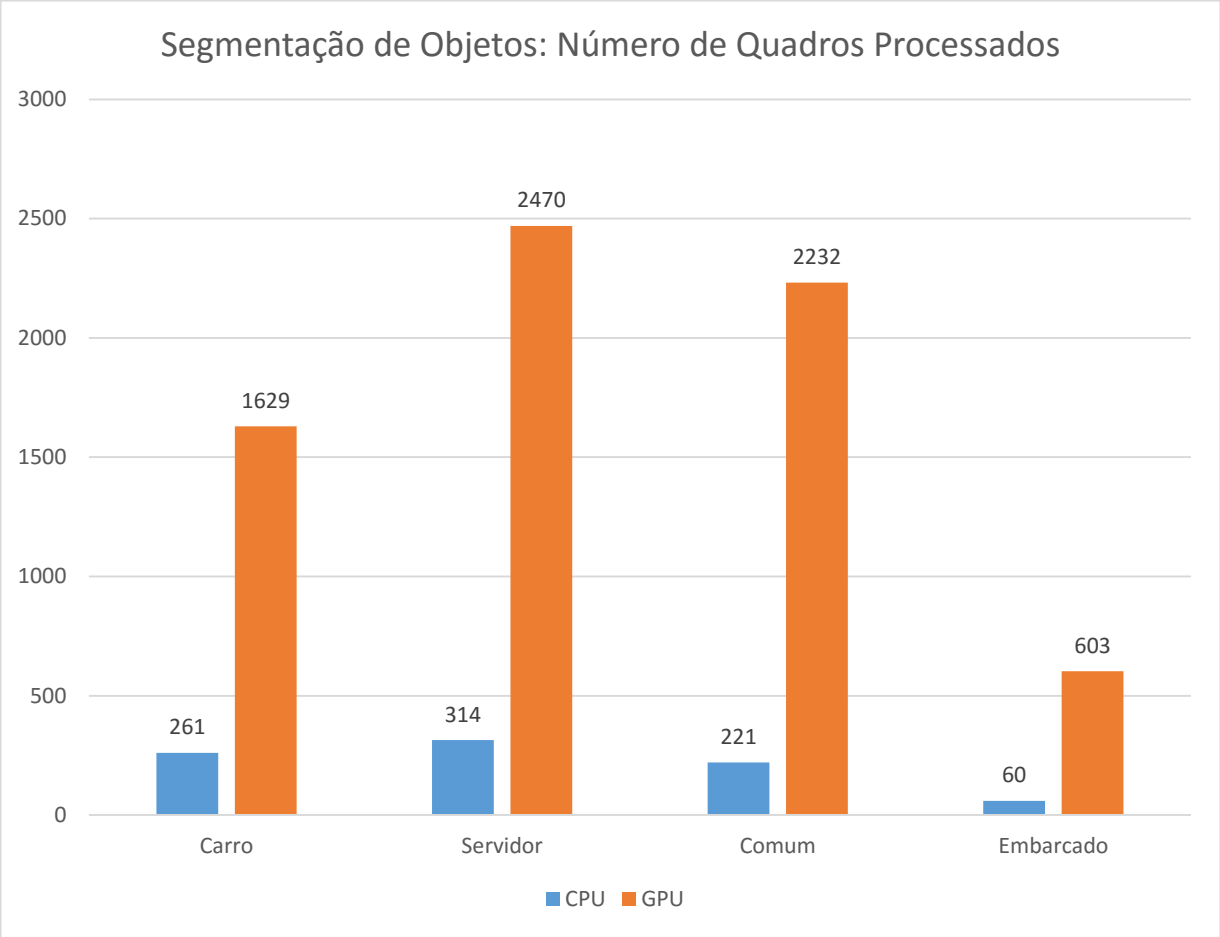

Fig. 41 Segmentação de Objetos: Número de Quadros Processados

4.4. Laser Scan

O serviço final une diversos dos serviços prévios com a finalidade de identificar de modo correto e ainda mais preciso as guias da rua. Para isso, partindo da nuvem de pontos com segmentação fina de guia, utiliza-se uma tática similar a empregada no veículo BOSS, descrito na seção 2.1.2.2, que foi posteriormente desenvolvida e aperfeiçoada por Patrick Shinzato (SHINZATO e WOLF, 2015). A leitura Laser Scan é uma redução de dados transformando a nuvem de pontos completa em uma leitura de um sensor de laser planar. Para isso, utiliza-se o

resultado do serviço de segmentação fina de guia e, ao excluir os pontos que não fazem parte da guia, guarda-se o ponto mais próximo da origem para cada ângulo radial de leitura.

O algoritmo foi primeiramente desenvolvido em C++ por Patrick Shinzato, e então novamente desenvolvido em Python e Numpy de forma a encaixar neste projeto. Sua entrada é a matriz dada pelo serviço de Segmentação Fina de Guias e analisa o ponto mais próximo da origem com variação relevante de altura por ângulo, ou seja, por coluna da matriz. O método desenvolvido para GPU utiliza uma *Thread* por ângulo, e assim cada uma analisa a sua coluna da matriz começando pelo primeiro anel até encontrar um ponto com alta variação de altura, classificando-o como guia e excluindo o resto da coluna. O resultado é um vetor de 1800 posições, e os tempos de execução são detalhados nas Fig. 42, Fig. 43 e Fig. 44. Após essa etapa, o resultado ainda passa pelo serviço de segmentador de objetos, fornecendo as guias segmentadas com respeito ao ambiente e com respeito a outra guia. Nota-se um tempo de processamento de na maioria dos casos 10 a 50ms em CPU e em GPU de 0,15 a 0,5ms (excluindo PC-Embarcado), indicando uma melhora significativa de eficiência. A Tabela 9 mostra o resultado final numericamente.

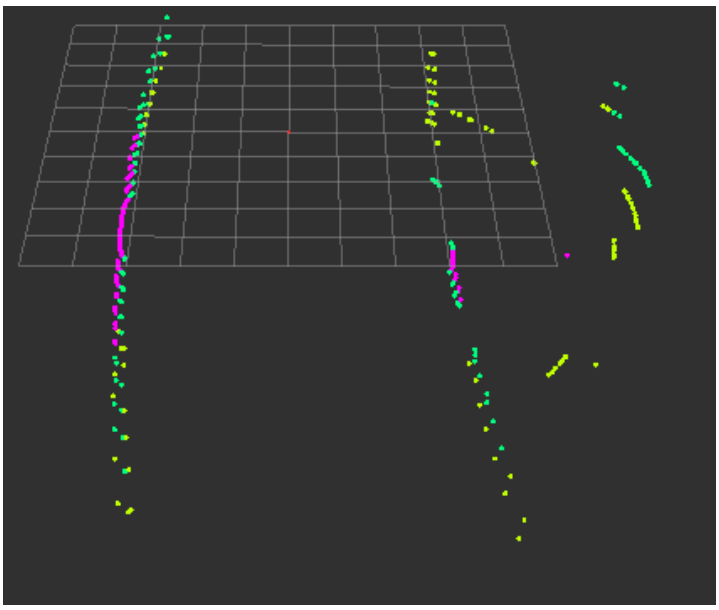

Fig. 42 Segmenação por Laser Scan

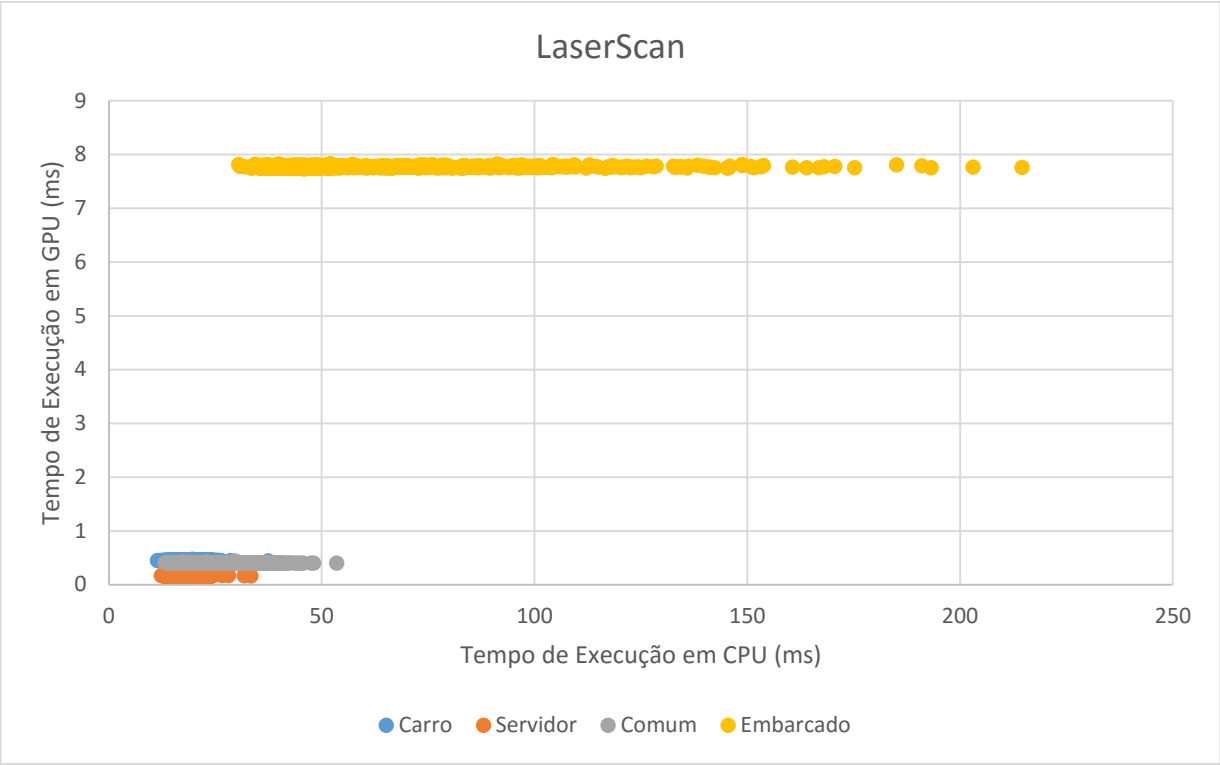

Fig. 43 LaserScan: Comparação de Tempos

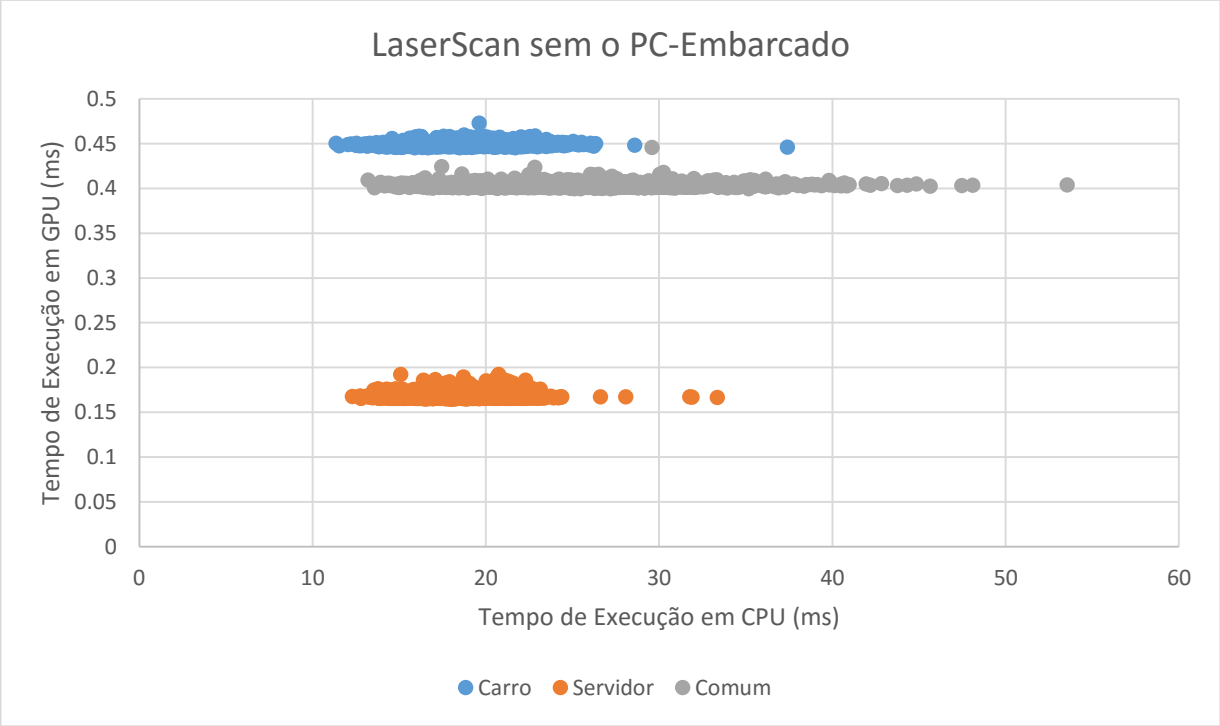

Fig. 44 LaserScan: Comparação de Tempos sem o PC-Embarcado

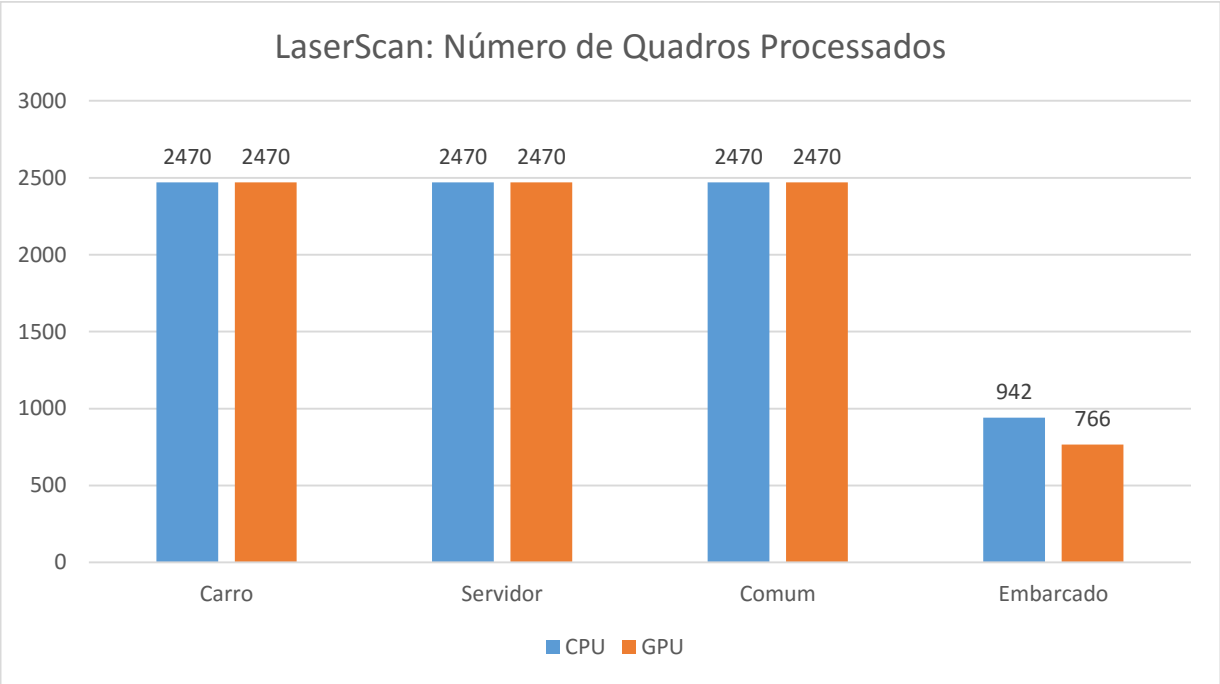

Fig. 45 LaserScan: Número de Quadros Processados

Tabela 9 LaserScan: Resultado Numérico Final

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	11.36	0.45	12.30	0.16	13.21	0.40	29.84	7.73
Med	18.53	0.45	18.11	0.17	25.26	0.40	62.06	7.78
Max	37.40	0.47	33.35	0.19	53.54	0.45	214.53	7.83

Foram testadas outras metodologias, como K-Means e separação por histograma, porém o Laser Scan foi o método de melhores resultados e de melhor paralelização. O primeiro método de teste, K-Means[32], não obteve um resultado aceitável a fim de ser selecionado e usado como o método final. Este método usa o algoritmo K-means para separar o ambiente em faixas longitudinais, e as duas faixas mais próximas de origem, uma a esquerda e uma a direita, passam por uma aproximação polinomial para detecção das guias. Configurações diferentes foram testadas, mas nenhuma delas forneceu uma boa qualidade de detecção devido aos muitos pontos não-planares perto do meio-fio. O método de histograma demonstrou problemas na paralelização e na resolução da detecção, sendo difícil identificar com precisão o suficiente as guias. É importante notar que para uso dessa técnica já era necessário a leitura Laser Scan

[32] K-Means: http://www.mathworks.com/help/stats/kmeans.html

previamente feita e então é criado um histograma de pontos 2D ao aplicar um limiar de modo que cada linha da matriz do histograma só tem o valor máximo da linha, e de zero para as outras posições.

4.5. Considerações Finais

Neste capítulo foram descritos os serviços desenvolvidos para o processamento da nuvem de pontos obtida pelo Velodyne, provendo métodos de tratamento dos dados que são importantes e bastante úteis na implementação de sistemas de apoio ao motorista (ADAS) e de auxílio na condução autônoma de veículos. Além dos métodos que foram implementados através de serviços, também foram apresentados e discutidos os resultados da performance de tais métodos, testados em diferentes plataformas (CPU e GPUs NVIDIA-CUDA de diferentes modelos). Foi possível com isto demonstrar o excelente ganho em termos de tempo de execução obtido quando implementados explorando o paralelismo e executados em GPUs. A Tabela 10 mostra o resumo dos resultados finais numericamente.

Tabela 10 Resumo do Resultado Numérico Final

Nome do Serviço		Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Correção Estrutural	Min	12.3	2.3	11.3	0.4	14.1	0.9	36.4	3.1
	Med	18.9	2.8	17.6	0.4	20.9	1.1	72.5	17.7
	Max	26.9	3.7	28	0.4	52	1.4	390.7	25.2
Correção Angular	Min	23.8	0.9	22.6	0.2	26	0.6	100.3	9.2
	Med	29.6	0.9	28.4	0.2	33.4	0.6	117.9	9.9
	Max	36.8	0.9	46.6	0.3	98.8	0.8	317.9	10.8
Segmentação do Chão	Min	0.6	1	0.7	0.3	0.7	0.6	3.3	8.1
	Med	0.9	1	0.7	0.3	0.9	0.7	8.2	8.3
	Max	1.6	1.1	1.9	0.4	9.1	0.9	22.9	9.2
Segmentação Fina de Guias	Min	6	1.1	6.1	0.3	6.1	1.1	26.3	9.9
	Med	8	1.1	7.1	0.3	8.1	1.1	54.9	21.1
	Max	31.4	1.2	18.7	0.3	19.7	1.3	131.7	25.4
Segmentação de Objetos	Min	409	47	346.3	5.8	541.8	18.5	1988.3	102
	Med	920	107	761.6	12	1114	51	4163	238
	Max	2409	402	1911	34	3229	164	11385	1348
LaserScan	Min	11.4	0.4	12.3	0.2	13.2	0.4	29.8	7.7
	Med	18.5	0.4	18.1	0.2	25.3	0.4	62.1	7.8
	Max	37.4	0.5	33.4	0.2	53.5	0.4	214.5	7.8

5. Conclusão

As diferentes tecnologias foram integradas conforme o andamento do projeto e, ao mesmo tempo, o conhecimento gerado pela montagem e uso do ambiente que une as tecnologias criou novas possibilidades, permitindo que outros projetos usem recursos aqui desenvolvidos. Isso possibilita melhorias neste próprio projeto e em outros projetos, permitindo também que novos projetos sejam criados com base no conhecimento agregado e gerado. Um exemplo é o projeto do estágio de pesquisa no exterior detalhado posteriormente que, apesar de não ter usado todas as tecnologias pesquisada neste projeto de mestrado, promoveu novos usos do ambiente e trouxe novas tecnologias para o laboratório.

Em relação aos GPU-Services, a Fig. 46 mostra a comparação entre as arquiteturas usadas com o tempo de todos os serviços somados. Apenas os 60 primeiros quadros são mostrados para correta comparação, visto que foi o número de quadros processados pelo PC-Embarcado no serviço de Segmentação de Objetos em CPU. E a Tabela 11 mostra o resultado final numericamente. Na totalidade dos resultados numéricos, foi observado que apesar do PC-Embarcado ser inferior em poder de processamento, considerando o baixo uso de potência usado em comparação aos outros equipamentos mostra sua vantagem visto que seu *hardware* equivale ao de um *smartphone* moderno, e analogamente é notável o ganho dos equipamentos inferiores em questão de custo monetário e custo energético, deixando então a escolha final do uso deste projeto à análise de requisitos do sistema no qual seria usado. Visando o uso no Carina 2 por exemplo, foi observado melhor custo/benefício o uso de uma das duas arquiteturas intermediárias.

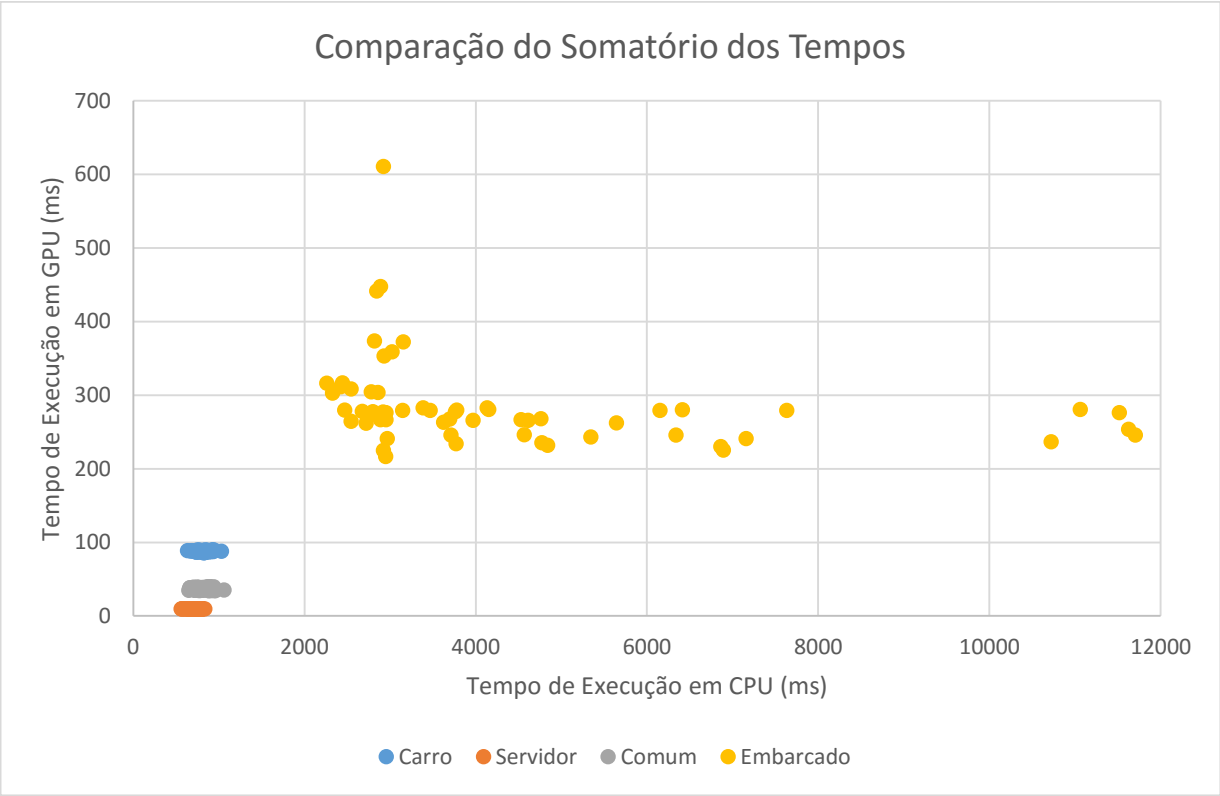

Fig. 46 Comparação do Somatório de Tempos

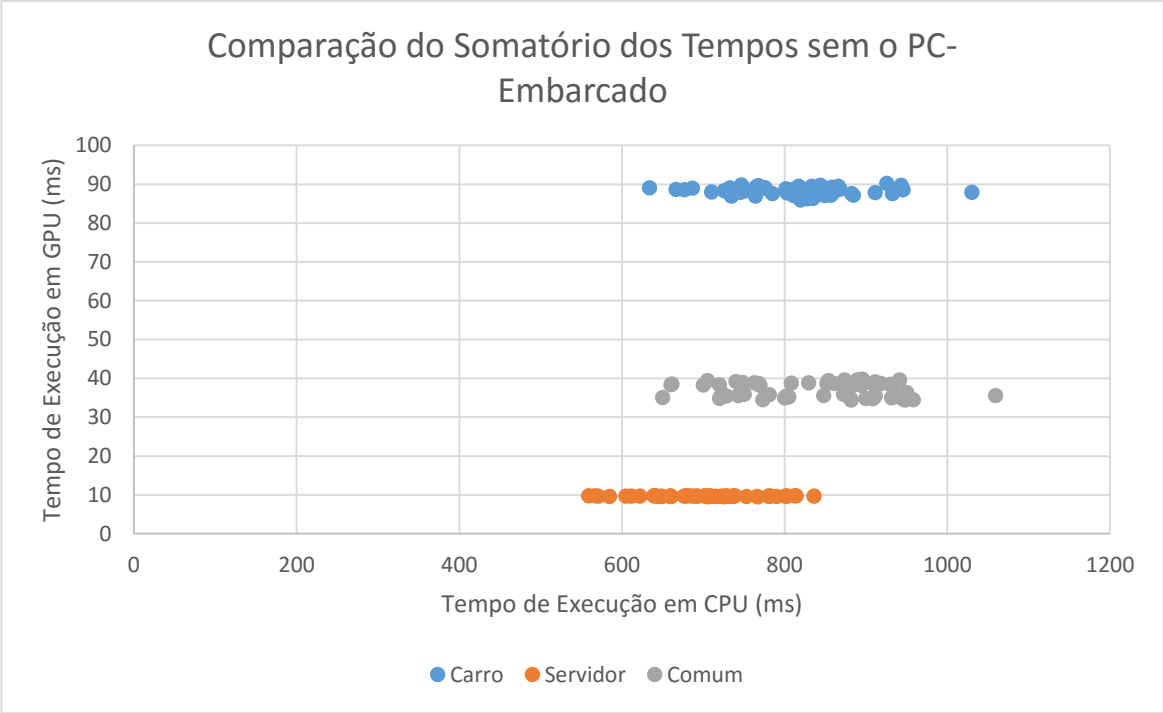

Fig. 47 Comparação do Somatório dos Tempos sem o PC-Embarcado

Tabela 11 Comparação do Somatório: Resultado Numérico Final

Tempo (ms)	Carro CPU	Carro GPU	Servidor CPU	Servidor GPU	Comum CPU	Comum GPU	Embarcado CPU	Embarcado GPU
Min	463.41	52.96	399.23	7.15	601.95	22.15	2184.42	139.62
Med	993.59	113.11	831.49	13.71	1200.20	54.75	4477.21	302.41
Max	2534.56	408.25	2028.76	35.38	3431.10	168.71	12144.71	1419.20

5.1. Principais Contribuições

Com todos os serviços descritos, teremos então as seguintes contribuições, resultados e implementações desenvolvidas nesta dissertação:

1. Ambiente de desenvolvimento que mistura as tecnologias usadas, notadamente: ROS, Velodyne e GPU-CUDA;
2. Serviço para adaptação da nuvem de pontos do Velodyne em estrutura matricial 32x1800 (CPU e GPU);
3. Serviço para correção da distorção Angular dos dados (CPU e GPU);
4. Serviço de segmentação das regiões do ambiente: de chão e de guias (CPU e GPU);
5. Serviço de segmentação de Objetos (CPU e GPU);
6. Serviço de processamento da nuvem de pontos com leitura LaserScan para detecção de guias (CPU e GPU).

Os trabalhos desenvolvidos no período do projeto foram:

- Poster apresentado na *GPU Technology Conference* (GTC) em 04/2015 sob o título "*GPUService: GPU Acceleration of Robotic Services: Real Time 3D Point Cloud Processing*" (CHRISTINO e OSORIO, 2015);
- Apresentação no workshop do CROB realizado em 29/04/2015 em São Carlos;
- Publicação de artigo científico e apresentação na *Latin American Robotics Symposium* (LARS) sob o título "*GPU Service Acceleration: Robotic Systems Services Acceleration by GPU Parallelization Focused in 3D Point Clouds in Real-Time Processing*" em 10/2015 (CHRISTINO e OSORIO, 2015).

5.2. Uso dos Serviços e Trabalhos Futuros

Com o grande número de informações vindas das técnicas apresentadas nas seções anteriores, projetos externos de mais alto nível podem usar tais informações para montar mapas e descrições/representações topológicas do ambiente. Usando as guias segmentadas e detectadas, obtém-se as descontinuidades nos dados geométricos do LaserScan, podendo ser armazenado temporalmente ou por localização (usando GPS[33]) dos locais, correlacionando locais de descontinuidade das guias, locais de identificação de obstáculo na pista (serviço de detecção de objetos), e orientação das guias por exemplo, como feito no trabalho de Badino (BADINO, HUBER e KANADE, 2012). Um esboço dessas possibilidades de detecção é mostrado na Fig. 48. Outras características extraídas poderão ser adicionadas conforme a necessidade como trabalhos futuros. Essa etapa é deixada para outros projetos devido à complexidade e demanda de trabalhos adicionais que extrapolam o tempo necessário para a conclusão deste trabalho, assim como o atual projeto já supre as necessidades atuais de paralelização para fins do desenvolvimento de veículos inteligentes. Nem todas tarefas precisam (e podem) ser paralelizadas como as que foram implementadas nos serviços disponibilizados, podendo ser implementadas em CPU (p.ex. planejamento de rotas), e uma vez que os processamentos mais críticos e de maior uso computacional já foram executados e a aplicação de um mapa topológico não requer tempo real.

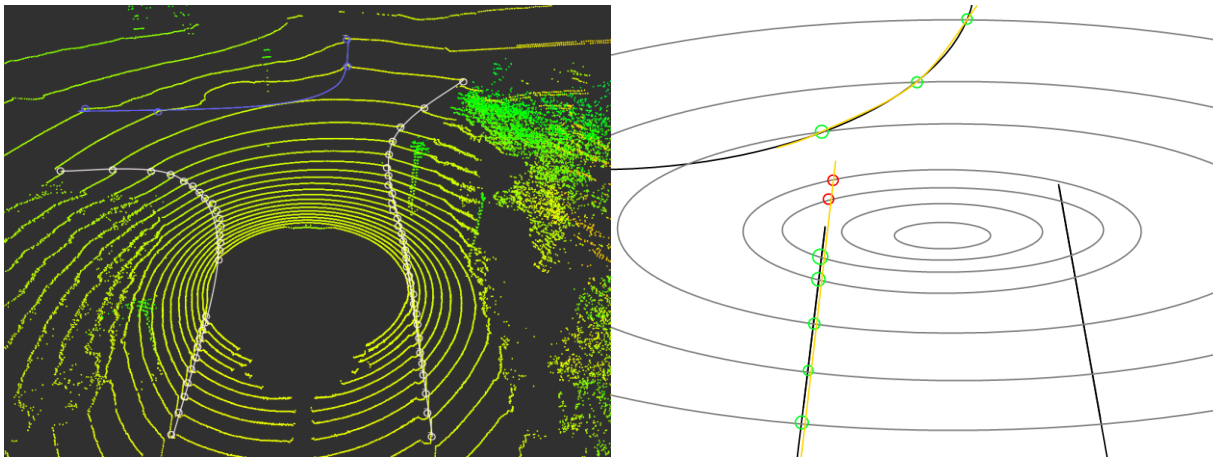

Fig. 48 Detecção de início de início de bifurcação

[33] Global Positioning Sysrtem: https://pt.wikipedia.org/wiki/Sistema_de_posicionamento_global

Por fim, a faixa central de trânsito também pode ser obtida e classificada ao utilizar a região interna das guias detectadas pelo Laser Scan. Utilizando máscaras de convolução aplicadas à técnica usada na seção 4.3, baseada na aplicação do carro autônomo BOSS, é possível identificar qual das quatro possibilidades de faixa central está presente e sobre a qual o carro está trafegando no momento. Exemplos de máscaras são apresentadas e diferenciadas conforme o exemplo apresentado na Fig. 49. Os dados segmentados também podem fazer parte do mapa topológico descrito previamente.

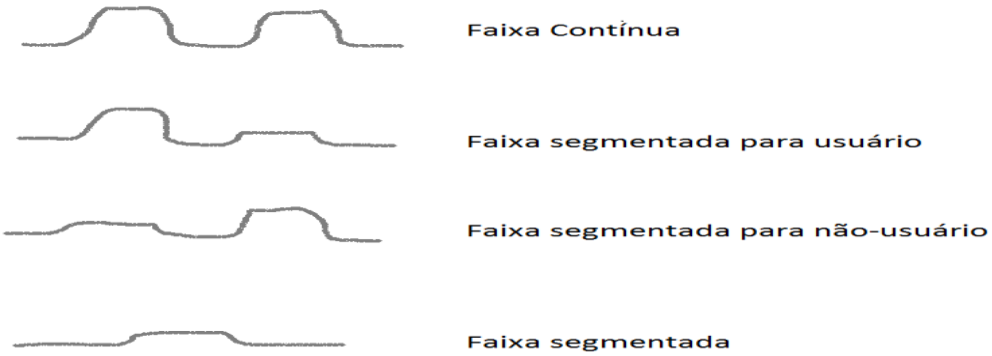

Fig. 49 Máscaras de convolução para identificação da topologia da faixa de trânsito

5.3. Estágio em Pesquisa no Exterior

Aplicações do uso de sensores 3D como o usado neste projeto são escassas e ainda bem direcionadas para uso dentro da própria computação, ofuscando o potencial dos conhecimentos e projetos desenvolvidos a partir desses sensores. Por esta razão a troca de experiências e conhecimentos com grupos de pesquisa que possuam trabalhos com outros focos, mas que usem ambientes similares é de suma importância. Por isso, foi realizado um estágio em uma universidade inglesa que participa com projetos com características similares, mas um enfoque diferente da computação pela computação com financiamento pela bolsa BEPE-FAPESP número 2013/25812-8. A Universidade de Loughborough, com seu pesquisador e professor Kaddour Marouf aceitou realizar uma pesquisa em conjunto em um projeto que aplica conhecimentos de processamentos de nuvem de pontos 3D, robótica, tendo inclusive uma aplicação diferenciada no tratamento de crianças autistas em estado grave. Algumas das técnicas usadas neste projeto foram inspiradas por técnicas usadas no projeto realizado na Inglaterra, assim como o oposto. Esse projeto pode ser encontrado no anexo 7.3.

5.4. Agradecimentos Finais

Os envolvidos no projeto são muito agradecidos à FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) pelo apoio e suporte financeiro durante a elaboração e desenvolvimento do projeto na Universidade de São Carlos e ao financiamento da bolsa BEPE para realização do mestrado sanduíche como pesquisador visitante na *Loughborough University* – LE, GB de março a setembro de 2014.

Também prestamos agradecimentos aos institutos relacionados ao INCT-SEC, ao CROB/SC e ao ICMC-USP/SC que, por meio do LRM, proveu toda infraestrutura, equipamento e recursos humanos necessários para o desenvolvimento.

Estre trabalho foi financiado por FAPESP (processo N. 2013/13880-9).

6. Referências[34]

ALENCAR, F. A. R. D.; WOLF, D. F. **Detecção de pedestres para veículos inteligentes**. São Carlos. 2015.

BADINO, H.; HUBER, D.; KANADE, T. Real-time topometric localization. **International Conference on Robotics and Automation**, 2012.

BAKER, S. P.; SADOWSKI, R. W. GPU assisted processing of point cloud data sets for ground segmentation in autonomous vehicles. **Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on**, p. 1 - 6, 22-23 April 2013.

BATAVIA, P. H. et al. **Applying Advanced Learning Algorithms to ALVINN**. [S.l.]: [s.n.].

BIRANT, D.; KUT, A. ST-DBSCAN: An algorithm for clustering spatial–temporal data. **Data & Knowledge Engineering 60.1**, 2007. 208-221.

BÖHM, C. et al. Density-based Clustering using Graphics Processors. **Proceedings of the 18th ACM conference on Information and knowledge management. ACM**, 2009. 661-670.

BUEHLER, M.; IAGNEMMA, K.; SINGH, S. **The 2005 darpa grand challenge:** The great robot race. [S.l.]: Springer, 2007.

BUEHLER, M.; IAGNEMMA, K.; SINGH, S. **The DARPA Urban Challenge autonomous vehicles in city traffic**. [S.l.]: Berlin Springer, 2009.

CHRISTINO, L.; OSORIO, F. **GPU Service Acceleration:** Robotic Systems Services Acceleration by GPU Parallelization Focused in 3D Point Clouds in Real-Time Processing. LARS/SBR 2015 (). [S.l.]: [s.n.]. oct 2015.

CHRISTINO, L.; OSORIO, F. **GPUService:** GPU Acceleration of Robotic Services: Real Time 3D Point Cloud Processing. GTC: GPU Technology Conference. [S.l.]: [s.n.]. apr 2015.

DARPA Grand Challenge 2005. [S.l.]: [s.n.], 2005.

DOUILLARD, B. et al. On the Segmentation of 3D LIDAR Point Clouds. **The Australian Centre for Field Robotics**.

EBERHARD, M.; TARPENNING, M. **The 21st Century Electric Car**. [S.l.]. 2006.

[34] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. **Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)**, 1996. 226–231.

FENTON, R. E. et al. **One approach to highway automation**. Proceedings of the IEEE. [S.l.]: [s.n.]. 1968. p. 556 - 566.

FERNANDES, L. et al. CaRINA Intelligent Robotic Car: Architectural design and applications. **Journal of Systems Architecture**, 2014.

GAREY, D.; LANG, S. High Performance Development with Python. **Scientific Computing**, 2008.

GOMES, S. D. A. K. B. A. A. **CUDA-based Triangulations of Convolution Molecular Surfaces**. [S.l.]: [s.n.].

GONZALEZ, et al. **Rapid Integration and Calibration of New Sensors Using the Berkeley Aachen Robotics Toolkit (BART)**. [S.l.]: [s.n.]. 2010.

GUIZZO, E. How google's self-driving car works, 18 out. 2011.

GUIZZO, E. How google's self-driving car works., 18 October 2011.

HABERMANN, D.; SILVA, R.; OSÓRIO, F. Detecção e Classificação de Objetos com uso de Sensor Laser para Aplicações em Veículos Autônomos Terrestres. **XV Simpósio de Aplicações Operacionais em Áreas de Defesa (SIGE)**, São José dos Campos, 2013. 55-59.

HALTERMAN, R.; BRUCH, M. **Velodyne HDL-64E LIDAR for Unmanned Surface Vehicle Obstacle Detection**. [S.l.]: [s.n.], 2010.

HATA, A. Y. et al. Road geometry classification using ANN. **2014 IEEE Intelligent Vehicles Symposium (IV), 2014, MI. 2014 IEEE Intelligent Vehicles Symposium Proceedings**, São Carlos, 2014. p. 1319-1324.

HATA, A. Y.; OSÓRIO, F. S.; WOLF, D. F. Robust curb detection and vehicle localization in urban environments. **IEEE Intelligent Vehicles Symposium (IV) Proceedings**, São Carlos, 2014. 1257-1262.

HENDERSON , et al. **Robust Autonomous Vehicles DARPA Urban Challenge**. [S.l.]: [s.n.], 2007.

HILLEL, A. B. et al. Recent progress in road and lane detection. **Machine Vision and Applications**, 2012.

HUANG, J.-H. **Keynote GTC2015**. Palo Alto. 2015.

KIM, J.-S.; HAWANGBO, M.; KANADE, T. **Realtime Affine-photometric KLT Feature Tracker on GPU in CUDA Framework**. [S.l.]: [s.n.].

KISS, M. **Audi Piloted Parking on zFAS: Valet Parking for the 21st Century**. Palo Alto. 2015.

LANGDON, H.-G. B. A. W. B. L. A. C. G. A. W. B. **with CUDA for Genetic Programming**. [S.l.]: [s.n.]. 2010.

LANGDON, W. B. Performing with CUDA. In: **_____ CIGPU 2011**. [S.l.]: ACM, 2011.

LUETTEL, T.; HIMMELSBACH, M.; WENSCHE, H.-J. Autonomous ground vehicles -concepts and a path to the future. **Proceedings of the IEEE**, 2012. 1831–1839.

LUTZ; MARK. **Programming Python**. 2. ed. [S.l.]: O'Reilly, 2001. Capítulo 21.

MACLACHLAN, R. et al. **Moving object detection with laser scanners," JFR**. Christoph Mertz and Luis E. Navarro-serment: [s.n.], 2012.

MIAO, Q. et al. Parallel Improved RANSAC Based on CUDA. **Journal of Sichuan University (Engineering Science Edition)**, n. 4, 2010. 020.

MONTEMERLO , et al. **Junior:** The Stanford Entry in the Urban Challenge. [S.l.]: [s.n.], 2007.

NVIDIA. NVIDIA CUDA Reference Manual. Disponivel em: <http://docs.nvidia.com/cuda/pdf/CUDA_Runtime_API.pdf>. Acesso em: ago. 2015.

SALES, D. O. et al. Adaptive finite state machine based visual autonomous navigation system. **Engineering Applications of Artificial Intelligence, v.1**, 2014. 1.

SHANE, R. et al. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. **Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming**, 2008. 73-82.

SHINZATO, P.; WOLF, D. F. **Estimação de obstáculos e área de pista com pontos 3D esparsos**. São Carlos. 2015.

SOBEL, I. **History and Definition of the Sobel Operator**. Palo Alto: [s.n.], 2014.

THRUN, S. et al. The Robot that Won the DARPA Grand Challenge. **Journal of Field Robotics**, 23, 2006. 661-692.

TRONIAK, D. M. **PR2 Rides the Elevator A Problem in Vision-based Localization**. [S.l.]: [s.n.], 2012.

VOUZIS, P.; ARNOLD, M. **A GPU Implementation of the Complex Logarithmic Number System**. [S.l.]: [s.n.].

WU, Y. et al. A Study on Reaction Time Distribution of Group Drivers at Car-Following. **Intelligent Computation Technology and Automation**, Changsha, Hunan, 10-11 Oct 2009. 452 - 455.

ZENG, X. et al. Real-time structure-light-based 3D terrain sensing for mobile robot using CUDA. **Cyber Technology in Automation, Control and Intelligent Systems (CYBER)**, p. 417 - 422, 26-29 May 2013.

ZHANG, ; GUAN, X.-F.; ZHU, Q. Probe into Image Segmentation Based on Sobel Operator and Maximum Entropy Algorithm, 2012. 238 - 241.

ZHANG, H.; MARTIN, F. CUDA accelerated robot localization and mapping. **Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on**, p. 1 - 6, 22-23 April 2013.

7. Anexos

7.1. Trabalho de conclusão

O trabalho de conclusão de curso de graduação, desenvolvido anteriormente pelo autor deste projeto de qualificação de mestrado, está relacionado com o presente projeto, e por este motivo é apresentado aqui como uma referência complementar em relação ao tema.

Segue o resumo deste Trabalho de Conclusão de Curso (TCC): "Em função das limitações dos sistemas atuais de simulação virtual de robôs móveis e veículos inteligentes, principalmente no que se refere à simulação de sensores avançados, este projeto visa estudar, projetar e usar um sensor virtual capaz de reproduzir o funcionamento do sensor real Kinect no simulador robótico GAZEBO, rodando sobre a plataforma ROS (*Robot Operating System*), e desenvolver o uso das capacidades computacionais de uma GPU, integrando as tecnologias de ponta usadas hoje. O sensor é simulado de tal forma a garantir que apresente o comportamento mais próximo possível ao do sensor real (incluindo erros e imprecisões), permitindo assim seu uso no desenvolvimento de aplicações simuladas de robótica. A tecnologia CUDA para supercomputação em GPU Nvidia permite processar a nuvem de pontos 3D vinda do sensor virtual, pela obtenção da mesma por meio da biblioteca PCL (*Point Cloud Library*). Portanto, este trabalho permitiu a integração de diferentes módulos, de simulação e de processamento, dos dados de um sensor virtual do Kinect, que poderá assim ser integrado e usado em diferentes aplicações de simulação e estudo de aplicações robóticas."

O texto da monografia completa do TCC pode ser visualizado acessando o link a seguir: https://dl.dropboxusercontent.com/u/19010263/LeoMilho-modelo%20tcc%20USP%20-%20Final%202.pdf

7.2. Programação CUDA

Este texto foi elaborado a partir das informações do documento do curso disponível on-line em: http://eradsp2010.files.wordpress.com/2010/10/curso2_cuda_camargo.pdf. As imagens e conceitos abaixo foram devidamente reproduzidos a partir do material do curso citado acima, onde os créditos de autoria das mesmas são do autor do referido texto, Prof. Raphael Y. de Camargo – UFABC.

Jogos 3D evoluíram muito nos últimos anos e passaram a exigir um poder computacional gigantesco. Além de gerar o cenário 3D, é preciso aplicar texturas, iluminação, sombras, reflexões, etc. As folhas individuais das plantas são desenhadas. Sombras são calculadas dinamicamente. Para tal, as placas gráficas passaram a ser cada vez mais flexíveis e poderosas. GPUs utilizam um número maior de transistores para colocar mais ALUs (Arithmetic Logic Unit) simplificadas. Assim, é possível fazer um maior número de cálculos ao mesmo tempo. A Fig. 50 mostra a diferença entre as arquiteturas de um CPU e uma GPU.

Fig. 50 Diferenças das arquiteturas de CPU e GPU

Exemplo da arquitetura da GPU Fermi, sendo ilustrada na Fig. 51:

- 16 Multiprocessadores:
- 32 processadores
- 16 unidades Load/Store
- 4 unidades funções especiais
- 64 kB Memória compartilhada

- 32768 registradores
- Cache de constantes e textura

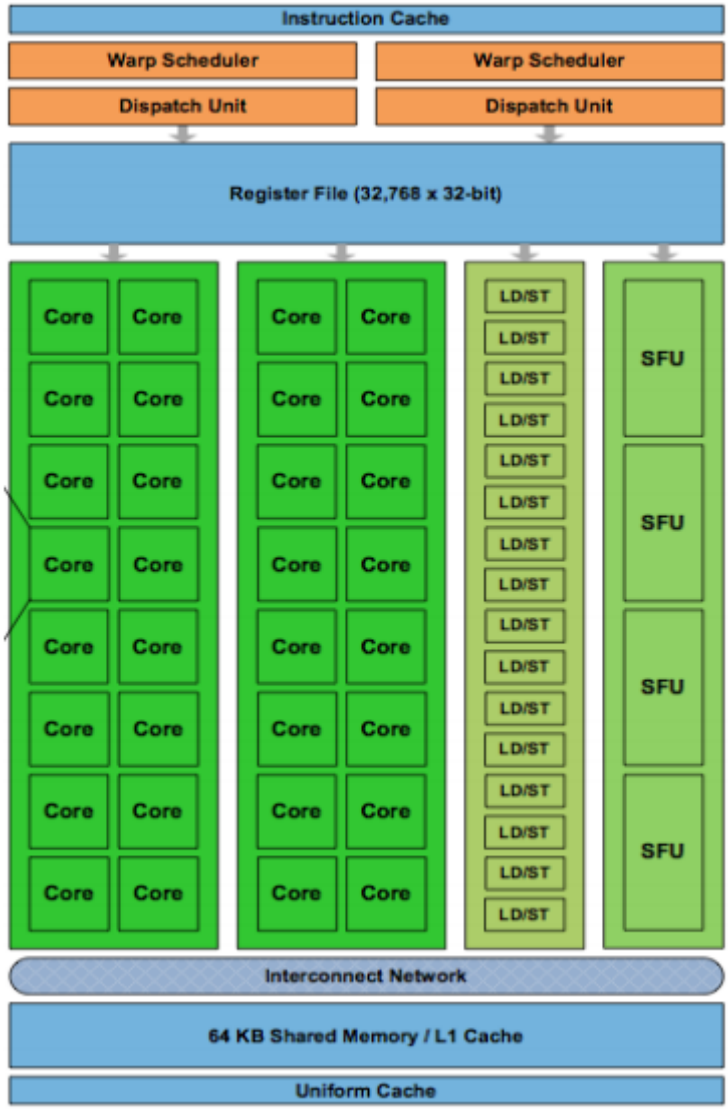

Fig. 51 Arquitetura Fermi

Os multiprocessadores da GPU compartilham uma memória global, onde são armazenados texturas ou dados de aplicações. A Fig. 52 mostra todos os elementos discutidos, e a Tabela 12 mostra a configuração de uma das melhores GPUs do mercado.

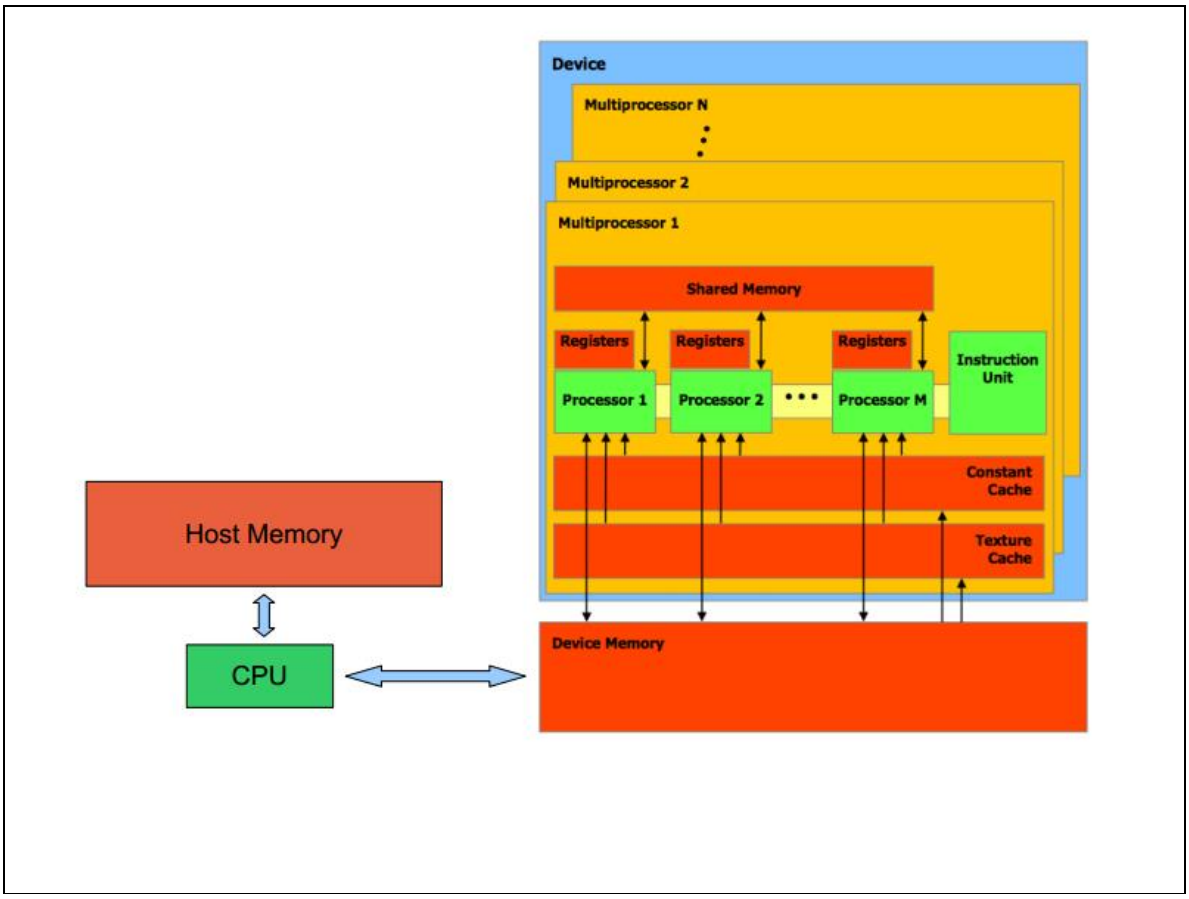

Fig. 52 Dentro de uma GPU

Tabela 12 Dados de uma GPU Tesla

Form Factor	1U
# of Tesla GPUs	4
GPU Memory Speed	1.55 GHz
GPU Memory Interface	384-bit
GPU Memory Bandwidth	148 GB/sec
Double Precision floating point performance (peak)	2. Tflops
Single Precision floating point performance (peak)	4.13 Tflops
Total Dedicated Memory*	12GB GDDR5

CUDA é uma arquitetura paralela de propósito geral destinada a utilizar o poder computacional de GPUs nVidia. É uma extensão da linguagem C, e permite controlar a execução de *threads* na GPU, além de gerenciar sua memória. Na hierarquia das memórias de uma GPU, cada execução do *kernel* é composta por uma grade, que é composta por blocos, que contém *threads*. Cada *thread* tem seus registradores. Os blocos podem se comunicar por meio da

memória compartilhada. Programas em CUDA normalmente utilizam o CUDA *Runtime*, que fornece primitivas e funções de alto-nível, e além disso também é possível utilizar uma API com funções, que permitem o gerenciamento da memória da GPU e outros tipos de controle. A memória global é acessível por todas as *threads*. Pode-se ver uma ilustração disso na Fig. 53 e na Fig. 54.

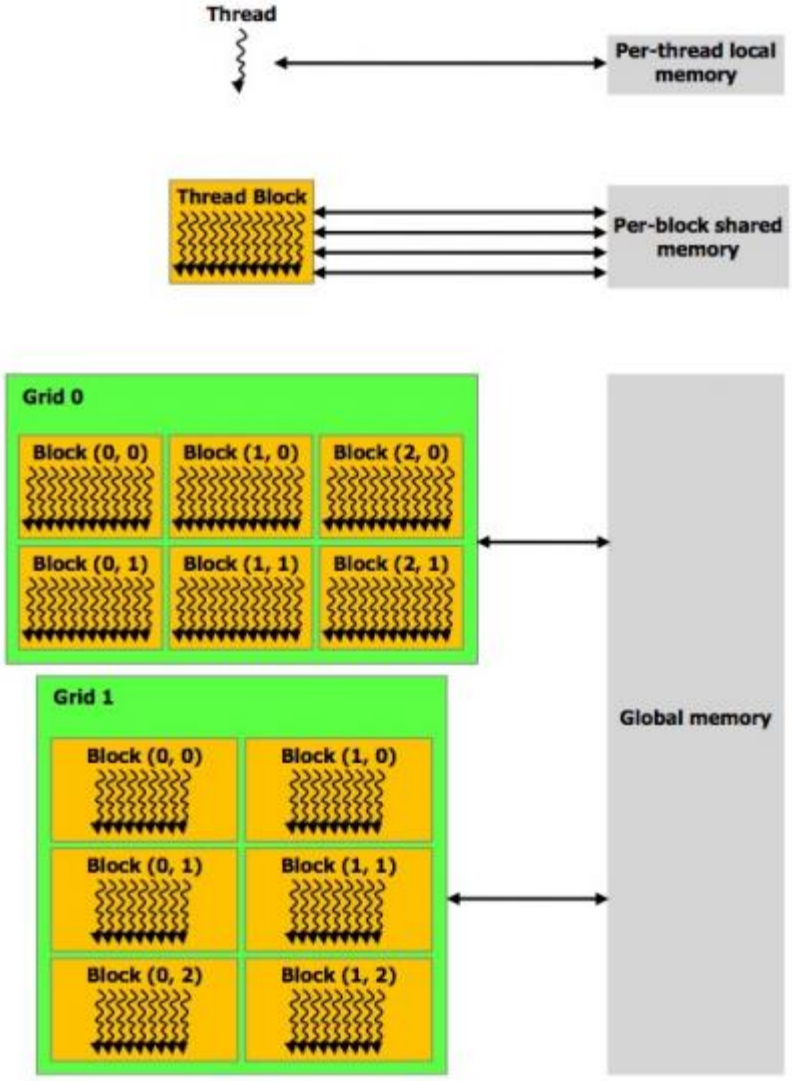

Fig. 53 Hierarquia de uma GPU

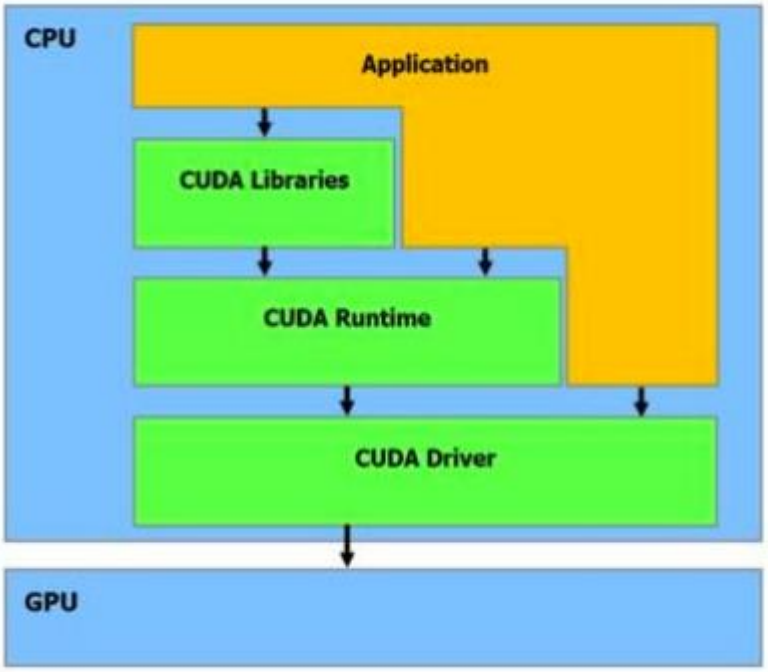

Fig. 54 Hierarquia da API CUDA

Ao executar uma aplicação, cada bloco é alocado a um multiprocessador da GPU, que pode executar um ou mais blocos simultaneamente. Cada multiprocessador executa 16 *threads* no modelo SIMT (*Single Intruction*, *Multiple Thread*). Pode-se ver um comparativo entre placas de vídeo com diferentes números de multiprocessadores na Fig. 55.

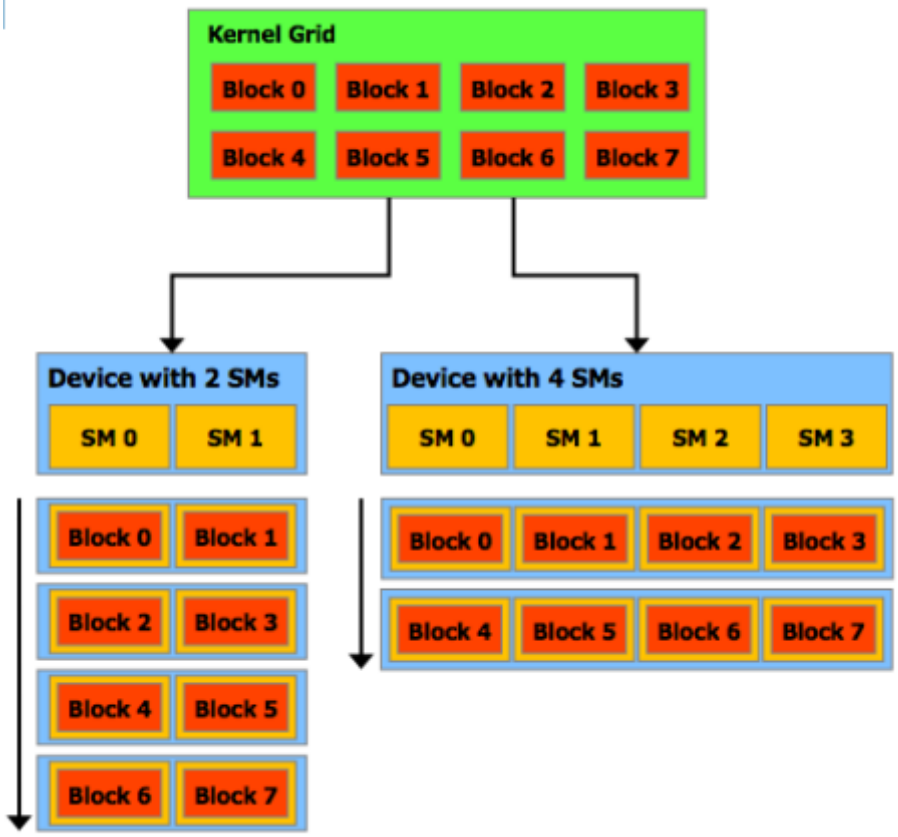

Fig. 55 Comparativo entre GPUs com diferentes números de multiprocessadores

A execução do programa controlada pela CPU que pode lançar *kernels*, que são trechos de código executados em paralelo por múltiplas *threads* na GPU. A execução de programas CUDA é vista na Fig. 56 e é composta por ciclos da seguinte forma:

CPU, GPU, CPU, GPU, … , CPU, GPU, CPU.

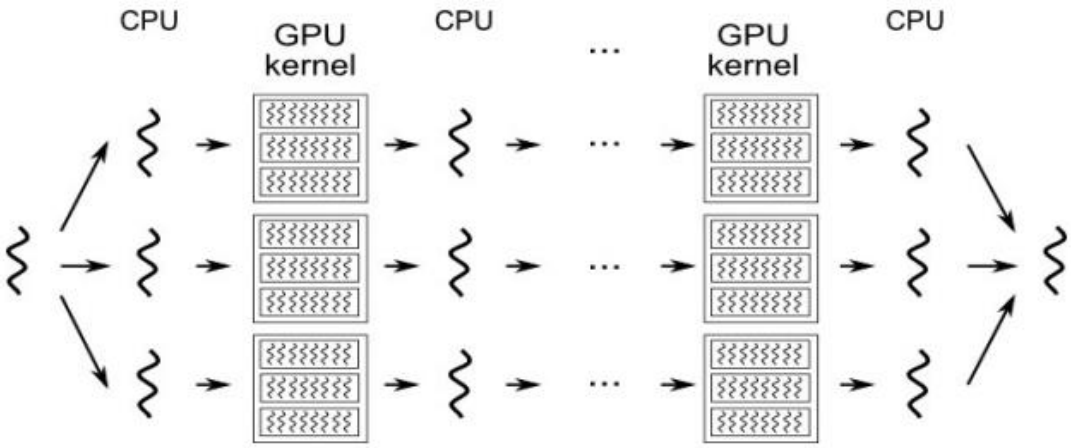

Fig. 56 Ciclo de vida de um kernel CUDA

Os programas que conseguem bons incrementos de velocidade em GPUs são os programas que podem ser subdivididos em pequenos subproblemas, que são alocados a diferentes blocos e *threads*. Cada *thread* mantém uma pequena quantidade de estado, mas que possui uma alta razão de operações de ponto flutuante por acesso à memória. Os subproblemas são mais eficientes quando fracamente acoplados[35]. Isso se deve ao uso da memória compartilhada. Todos os processadores acessam a memória global utilizando o mesmo barramento com o uso de mecanismos primitivos de sincronização entre threads, e o acesso à memória do computador é feita através da PCI-Express (8GB/s).

Aplicações como algoritmos genéticos, simulações virais, detalhamento de exames de ultrassom, conversão de fotos e vídeos, processamento de dados e imagens sísmicas e análise de flutuação do mercado de ações são exemplos de programas bons para usarem da capacidade de processamento de uma GPU.

[35] Acoplamento: quando um problema pode ser dividido em subproblemas menores, o acoplamento indica o quanto os subproblemas são dependentes entre si.

7.3. Projeto do Robo Terapeuta de Crianças Autistas (em Inglês)

O texto a seguir apresenta a proposta das atividades a serem realizadas em um estágio no exterior, a ser realizado pelo proponente deste projeto de qualificação de mestrado. Estas atividades estão diretamente relacionadas com o presente projeto, uma vez que envolvem o desenvolvimento de um sistema de percepção baseado no sensor Kinect e no processamento dos dados 3D através do uso de GPUs. Por este motivo o projeto proposto é apresentado aqui como uma referência complementar em relação ao tema.

O texto completo do projeto de estágio no exterior está em Inglês, onde segue aqui o resumo em Português deste: "Em crianças em desenvolvimento típico, as atitudes cognitivas e emocionais para com objetos e pessoas constituem os ingredientes de relações sociais. Autismo infantil, que é um distúrbio do desenvolvimento neurológico, é caracterizada por dificuldades na comunicação social e interpessoal, bem como no processamento de emoções das pessoas próprios e outros. Na terapia de reabilitação autismo, diferentes abordagens estão sendo utilizadas para melhor compreender a capacidade das crianças autistas para a interação social e emocional. Essas abordagens são baseadas na crença de que os ambientes artificiais, constituído de robôs, parecem ser mais úteis do que ambientes reais em permitir que as crianças autistas a manifestem um interesse social; este por sua vez pode levar ao desenvolvimento de suas habilidades sociais. Atualmente, o robô que é usado para interagir socialmente com essas crianças está sendo totalmente teleoperado pelo terapeuta de acordo com o comportamento da criança por meio de um controle remoto (Fig. 57). O objetivo do projeto é desenvolver a percepção e a reação do robô usando sensores de imagem e de profundidade, tornando o robô capaz de tomar algumas decisões automaticamente, como desvio de obstáculos ou reação sobre a ação da criança, bem como propor novas interações automáticas dependendo em gestos da criança. Estas características adicionam muito valor ao projeto já em andamento que está sendo executado."

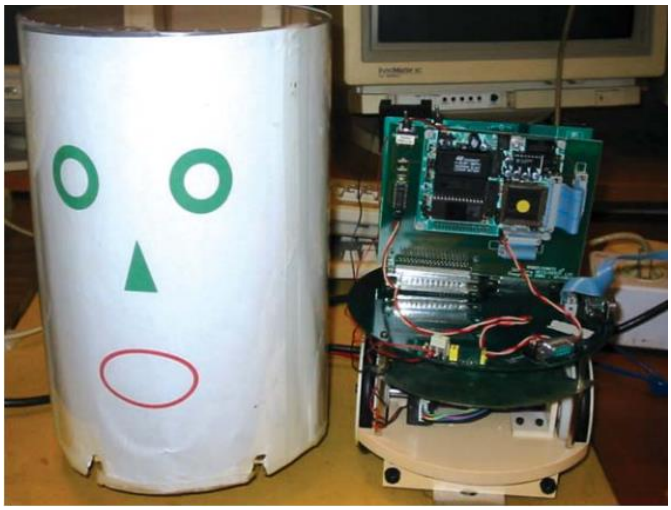

Fig. 57 GIPY-1 Robot

O texto completo do projeto de estágio no exterior pode ser visualizado acessando o link a seguir: https://dl.dropboxusercontent.com/u/19010263/Proj-Robot-Therapist-BEPE-V3.pdf.