FACULDADE DE TECNOLOGIA
FATEC SANTO ANDRÉ

Tecnologia em Eletrônica Automotiva

JESSICA MADOKORO

APLICAÇÕES DE TÉCNICAS DE IDENTIFICAÇÃO UTILIZANDO CÂMERA RGB-D

Santo André
2018

CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA
FATEC SANTO ANDRÉ

Tecnologia em Eletrônica Automotiva

JESSICA MADOKORO

APLICAÇÕES DE TÉCNICAS DE IDENTIFICAÇÃO UTILIZANDO CÂMERA RGB-D

Trabalho de Conclusão de Curso entregue à Fatec Santo André como requisito parcial para obtenção do título de Tecnólogo em Eletrônica Automotiva.
Orientador: Prof. Esp. Carlos Alberto Morioka

Santo André
2018

LISTA DE PRESENÇA

SANTO ANDRÉ, 20 DE DEZEMBRO DE 2018.

LISTA DE PRESENÇA REFERENTE À APRESENTAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO COM O TEMA **"APLICAÇÕES DE TÉCNICAS DE IDENTIFICAÇÃO UTILIZANDO CÂMERA RGB-D"** DOS ALUNOS DO 6º SEMESTRE DESTA U.E.

BANCA

PRESIDENTE:
PROF. CARLOS ALBERTO MORIOKA _____

MEMBROS:
PROF. PAULO TETSUO HOASHI _____

PROF. FERNANDO GARUP DALBO _____

ALUNOS:

JÉSSICA MADOKORO Jéssica Madokoro

M183a
Madokoro, Jéssica
Aplicação de técnicas de identificação utilizando câmera RGB-D / Jéssica Madokoro. - Santo André, 2018. – 65f: il.

Trabalho de Conclusão de Curso – FATEC Santo André. Curso de Tecnologia em Eletrônica Automotiva, 2018.

Orientador: Prof. Esp. Carlos Alberto Morioka

1. Eletrônica. 2. Sistemas eletrônicos. 3. Câmera RGB-D. 4. Veículos autônomos. 5. Obstáculos. 6. Localização. 7. Sensor de profundidade. 8. Visão computacional. 9. YOLO V3. I. Aplicação de técnicas de identificação utilizando câmera RGB-D.

621.389

AGRADECIMENTOS

Agradeço a todos meus familiares pelo apoio incondicional, em especial ao meu marido e filho. Agradeço a Fatec Santo André e a todos os professores pelo apoio para a elaboração desse projeto.

Você não é derrotado quando perde, mais sim quando você desiste

Vegeta (Dragon Ball Z)

RESUMO

Os veículos autônomos são o futuro dos sistemas de transporte urbano. Para que essa realidade possa ser concretizada, os sistemas de visão estão recebendo constante aprimoramento. Diversas técnicas foram propostas nos últimos anos para melhorar sua capacidade de detectar elementos e definir seu posicionamento. Dado esse cenário, o objetivo deste trabalho foi construir um sistema que utiliza um Kinect v2 como câmera RGB-D para detectar obstáculos e definir sua distância do ponto de observação. Foi utilizada uma metodologia experimental, onde as principais técnicas apresentadas na literatura, como as redes neurais convolucionais foram utilizadas. Como resultados, a localização dos objetos na imagem conseguiu ser realizada com sucesso, contudo, devido ao posicionamento não simétrico do sensor de visão com o sensor de profundidade, a marcação com a distância de cada elemento não foi realizada.

Palavras-chave: Visão computacional, câmera RGB-D, YOLO V3.

ABSTRACT

Autonomous vehicles are the future of urban transport systems. For this reality to be come true, the vision systems are receiving constant improvement. Several techniques have been proposed in recent years to improve their ability to detect elements and define their positioning. Given this scenario, the objective of this work was to build a system that uses a Kinect v2 as an RGB-D camera to detect obstacles and define its distance from the observation point. An experimental methodology was used, where the main techniques presented in the literature, such as convolutional neural networks were used. As a result, the location of the objects in the image could be successfully achieved, however, due to the non-symmetrical positioning of the vision sensor with the depth sensor, the marking with the distance of each element was not performed.

Keywords: Computational vision, RGB-D camera, YOLO V3.

SUMÁRIO

ÍNDICE DE ILUSTRAÇÕES

ÍNDICE DE TABELAS

LISTA DE ABREVIAÇÕES

Kinect v2	Kinect de segunda geração, para Xbox ONE
NUI	*Natural User Interface*
RGB-D	*Red, Green, Blue plus Depth*
YOLO	*You Only Look Once*

1. INTRODUÇÃO

O mundo que vivemos está sob constante mudança e a evolução a passos largos das tecnologias permite que os sonhos mais distantes possam ser alcançados.

A origem da locomoção veio com o homem utilizando suas próprias fontes internas de energia para se deslocar de um ponto ao outro. Posteriormente, o homem passou a utilizar outros animais como forma de energia para realizar esses deslocamentos. Em ambos os casos, a capacidade de se deslocar estava limitada as condições físicas do elemento motor, animal ou humano, tomando grande quantidade de tempo para realizar alguns deslocamentos, quando esses eram possíveis de se realizar.

Com a criação dos automóveis, a humanidade resolveu uma parte de suas dificuldades de mobilidade, permitindo a locomoção mais rápida, com maior conforto e torne-se possível alcançar maiores distancias. Os veículos tornaram-se objetos do cotidiano das pessoas, gerando uma grande dependência deles.

Com a utilização massiva dos veículos para deslocamento de pessoas e cargas, o número de acidentes envolvendo-os cresceu exponencialmente.

Em 2010, o Brasil ficou em quarto lugar dos países com mais mortes no trânsito no mundo, perdendo apenas para China, Índia e Nigéria. A Tabela 1 mostra os países com maior quantidade de mortes absolutas no trânsito.

Tabela 1 – Índice de Mortes Absolutas no Trânsito

Países com maiores números absolutos de morte no trânsito - 2010

Ranking	País	Posição no IDH	População estimada[1]	Nº de mortes[2]	Taxa de Mortes por 100 mil hab.	Número de veículos registrados	Taxa de mortes por 1 mil veículos
1º	China	101º	1.348.932.032	275.983	20,5	207.061.286	1,33
2º	Índia	136º	1.224.614.272	231.027	18,9	114.952.000	2,01
3º	Nigéria	153º	158.423.184	53.339	33,7	12.545.177	4,25
4º	Brasil[3]	85º	194.946.488	42.844	22	64.817.974	0,66
5º	Indonésia	121º	239.870.944	42.734	17,8	72.692.951	0,59
6º	Estados Unidos	3º	310.383.968	35.490	11,4	258.957.503	0,14
7º	Paquistão	146º	173.593.384	30.131	17,4	7.853.022	3,84
8º	Rússia	55º	142.958.156	26.567	18,6	43.325.312	0,61
9º	Tailândia	103º	69.122.232	26.312	38,1	28.484.829	0,92
10º	Irã	76º	73.973.628	25.224	34,1	20.657.627	1,22

Instituto Avante Brasil, PNUD, OMS, Datasus

[1] Os dados populacionais foram extraídos do banco de dados da Divisão de População das Nações Unidas

[2] As taxas de mortalidade no trânsito foram extraídas dos registros de morte reportados pelos Estados à Organização Mundial da Saúde, dos registros oficiais divulgados por cada país e através de um modelo regressivo para estimar se o número de mortes no trânsito do modificado na publicação Global Status Report on Road Safety 2013.

[3] Número de mortes no trânsito no Brasil de acordo com os dados oficiais do Datasus, em 2010.

(Retirado de: https://thumbs.jusbr.com/filters:format(webp)/uploads.jusbr.com/publications/artigos/113704460/images/1393352195.jpg. Acesso em 26/09/2018.)

Com a criação de veículos autônomos, estima-se que é possível reduzir em até 90% os acidentes automotivos, pois sem um condutor humano pode-se eliminar variáveis como sono, cansaço, negligência, imprudência. Ainda há progresso a ser feito pelos veículos autônomos, a capacidade de localização e identificação do ambiente em tempo real é parte fundamental para evitar outros veículos e pedestres, e assim evitar acidentes.

1.1 OBJETIVO

Dado o ambiente exposto anteriormente, ficou claro a demanda por soluções que possam colaborar com as ferramentas e tecnologias atuais para detecção de ambientes e objetos nos veículos e robôs autônomos. Sendo assim, esse trabalho teve como objetivo a validação do uso de câmeras RGB-D, para a aplicação de algo-

ritmos de detecção de objetos e a determinação da distâncias desses objetos em relação a câmera, tudo em tempo real.

É proposta a utilização da câmera com um algoritmo de rede neural convolucional já treinada, chamada YOLO (*You Only Look Once*) para a detecção de objetos, e a câmera RGB-D para a determinação da distância entre desses objetos, além de visão computacional para o processamento de imagens.

1.2 JUSTIFICATIVAS

Desde a invenção dos automóveis, buscou-se mais conforto e segurança para o condutor e os passageiros. Cada vez mais utiliza-se de sistemas que auxiliam o condutor, como *Adaptive Cruise Control*, sistemas que estacionam o veículo sozinhos, que controlam a aceleração e desaceleração, e mais recentemente os veículos autônomos. (UROOJ, FEROZ E AHMAD; 2017).

Como apontado por (LITMAN; 2018); (UROOJ, FEROZ E AHMAD; 2017) e (BIMBRAW, 2015), uma das maiores causas de mortes no mundo, é em decorrência de acidentes no trânsito, em maior parte acarretadas por erro humano. O veículo autônomo pode aumentar a segurança. Ao retirar o controle do veículo do piloto humano é possível reduzir em torno de 90% dos acidentes no trânsito, diminuindo drasticamente o número de mortes. Além da melhora de eficiência do consumo de energia, conforto, aumento de produtividade, e outras vantagens.

Este trabalho buscou contribuir para o desenvolvimento da tecnologia de veículos autônomos, procurando validar algoritmos de detecção de objetos que possam ser aplicados mesmo em ambientes externos (*outdoor*).

1.3 ESTRUTURA DO TRABALHO

Este trabalho foi organizado da seguinte maneira, a fundamentação teórica buscou trazer os principais autores e os trabalhos mais relevantes relacionados ao

desenvolvimento de veículos autônomos, redes neurais convolucionais, YOLO, câmeras RGB-D, tratamento de imagem e visão computacional.

No capítulo sobre a metodologia, os principais autores do campo foram consultados para elaboração do procedimento metodológico utilizado no trabalho. A forma como os testes de cada sensor foi elaborada, quais dados foram observados e quais as limitações do estudo realizado.

Na etapa de desenvolvimento, a implementação computacional de cada algoritmo é descrita junto dos dados que foram obtidos em cada uma das etapas, as condições de cada um dos testes também foram aqui relatadas.

Os resultados e as discussões do projeto são sumarizados na seção de Resultados e Discussões. A análise estatística realizada no conjunto de resultados obtidos é apresentada neste ponto.

As considerações finais são apresentadas seguida das sugestões para os trabalhos futuros e as referências do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Nesse capitulo foi abordado uma base teórica para o desenvolvimento do trabalho. As referências mais conceituadas e os trabalhos mais importantes sobre YOLO, câmeras RGB-D com ênfase no Kinect V2, carros autônomos e visão computacional foram consultadas.

2.1 VEÍCULOS AUTÔNOMOS

Um veículo autônomo combina tecnologias e automação. Nesses veículos o condutor é eliminado ou tem seus controles sobre a condução limitadas. Utiliza-se de sensores, poder computacional, comunicações de curto alcance, obtenção de dados em tempo real de veículos e acontecimentos na mediação. Esses veículos tem a capacidade de navegação, e de decisão, mesmo em situações críticas (LITMAN; 2018).

Segundo Bimbraw, (2015), já em 1926 pensava-se em um veículo controlado sem um motorista dentro, nesse ano criou-se um carro controlado a radio, chamado de “Linriccan Wonder” onde os comandos eram transmitidos por uma antena, que assim controlava um pequeno motor elétrico onde comandava os controles dos movimentos desse carro. Depois ano após ano, conforme a tecnologia avançava, diversos protótipos foram criados, muitos deles por universidades. Houve grandes avanços em 1980, com a Van robótica da Mercedes-Benz, que utilizava como guia sistemas de visão, junto ao LIDAR, radar, GPS e visão computacional.

Em 2010 quatro vans já trafegavam autonomamente da Itália a China. Em 2012 o Google lançava seu carro autônomo, que percorreu sem quaisquer acidentes mais de 482.000 quilômetros (cerca de 300.000 milhas). (LITMAN; 2018); (UROOJ, FEROZ E AHMAD; 2017). Na Figura 1, temos um veículo autônomo do Google em parceria com a Waymo, e uma breve descrição de seu funcionamento.

Figura 1 – Veículo autônomo Google com parceria Waymo

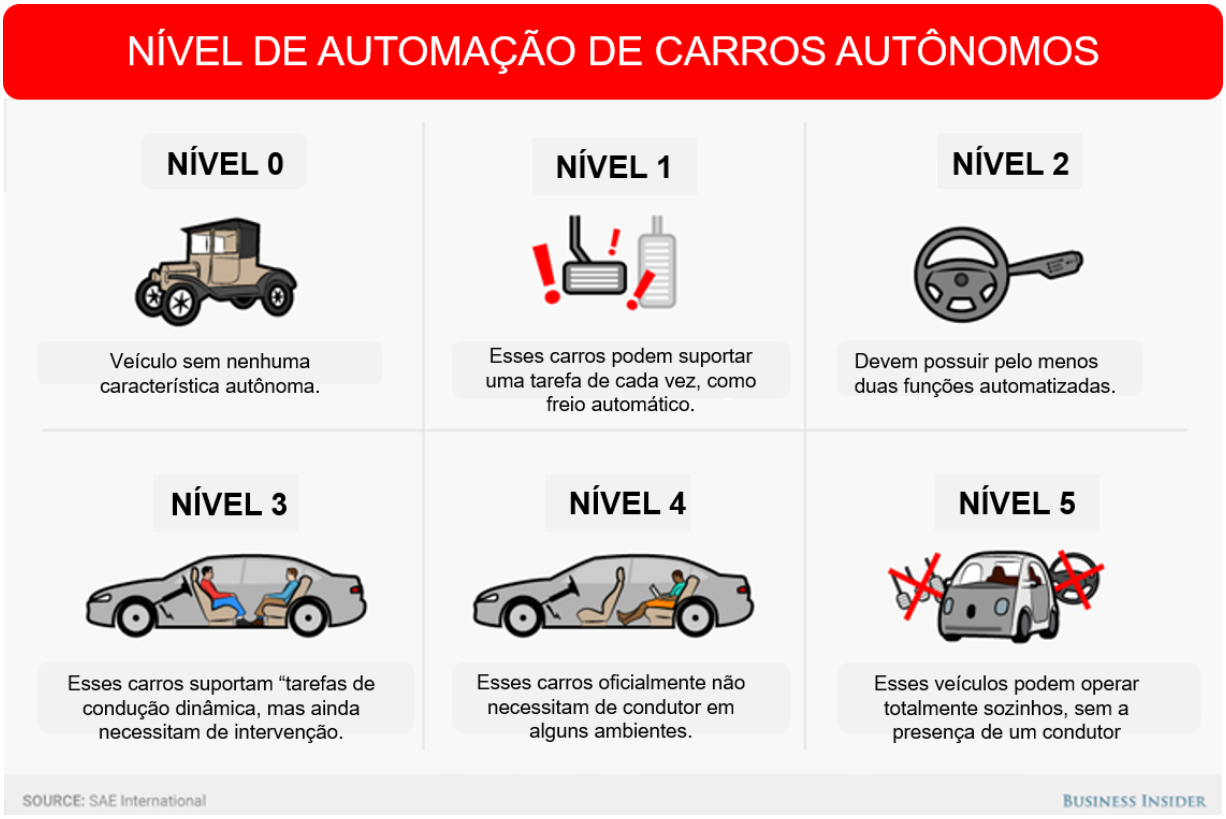
(Adaptado de: https://i.pinimg.com/originals/c6/4d/a1/c64da1f1cc2ae77b041542b1b21e8307.jpg) Acesso em 14/05/2018.

Diversas empresas entraram na corrida para a criação de veículos autônomos, como a Audi, Mercedes e Tesla, com destaque para a última, por ser a primeira empresa a comercializar um carro com um sistema de piloto automático (nível 3 de autonomia, os níveis de autonomia foram abordados mais à frente no trabalho), o Model S (UROOJ, FEROZ E AHMAD; 2017).

2.1.1. NÍVEIS DE AUTONOMIA DE UM VEÍCULO

Yamane, Smart e Forlizzi (2017), afirmam que em 2014 a Sociedade de Engenheiros Automotivos dos Estados Unidos definiu 6 níveis de autonomia em um veículo, como ilustrado na Figura 2, esse padrão adotado também em 2016 pela NHTSA (*National Highway Traffic Safety Administration*), órgão estadunidense que regula padrões utilizados no transporte.

Figura 2 - Níveis de automação de carros autônomos



(Adaptado de: https://geospatialmedia.s3.amazonaws.com/wp-content/uploads/2018/02/bi-graphics_autonomous-cars.png) Acesso em 17/05/2018.

A partir da figura 2 podermos definir:

- Nível 0: O motorista controla todas as ações do veículo, sem o uso de nenhuma ferramenta automática para auxilio.
- Nível 1: O motorista deve estar em prontidão para assumir o controle do veículo a todo momento. Um sistema de automatizado pode auxiliar o motorista em alguma parte de uma tarefa. Exemplos são: controle de estabilidade, controle de velocidade e ABS.
- Nível 2: Deve-se haver a automação de pelo menos de 2 funções de controle primaria do veículo, como *Adaptive Cruise Control*, assistente de estacionamento, centralização de faixas. O motorista pode por um breve período, abandonar suas funções como condutor, mas devendo tomar o controle sempre que o veículo sair da faixa por exemplo.
- Nível 3: Em ambientes limitados, como grandes rodovias, o motorista pode se abster de suas funções, mas devendo assumir o controle do veículo em condições adversas, onde o sistema autônomo por algum motivo não pode operar.
- Nível 4: Neste estágio o veículo pode trafegar em vários tipos de ambiente, somente quando há mau tempo o motorista deve conduzir o veículo e só poderá acionar o modo autônomo novamente, quando o ambiente estiver seguro.
- Nível 5: Não há necessidade de condutor, sendo que ele somente introduz no sistema o local de destino, o veículo pode trafegar em qualquer ambiente onde se é permitido.

.

2.1.2 BENEFÍCIOS E CONFLITOS DE UM SISTEMA VEICULAR AUTÔNOMO.

Independentemente do nível de automação, necessitamos verificar os benefícios e conflitos de um veículo autônomo, alguns desses pontos são: (LITMAN; 2018):

- **Segurança**: Acidentes no trânsito são responsáveis por grande parte da mortalidade mundial, grande parte destes acidentes ocorrem por falha humana, cansado, imprudência, doença, uso de drogas ao dirigir. Ao retirar o fator humano, estima-se diminuir cerca de 90% dos acidentes, e consequentemente diminuir drasticamente o número de mortes. (LITMAN; 2018); (UROOJ, FEROZ E AHMAD; 2017); (BIMBRAW, 2015). O tempo de reação do veículo autônomo é muito superior ao humano, assim ele é capaz de desviar ou parar o veículo muito mais rápido, evitando uma colisão. (LITMAN,2018).
- **Produtividade:** Perde-se muito tempo no deslocamento, e em congestionamentos. Ao utilizar um carro autônomo, o tempo dentro do carro passa a contar não mais como horas de deslocamento, mas sim como horas produtivas para realizar outras tarefas, inclusive descansar. Como o veiculo locomove-se em uma velocidade maior com segurança, também é possível alcançar o objetivo, mais rapidamente. (UROOJ, FEROZ E AHMAD; 2017).
- **Consumo de combustível:** Com veículos autônomos diminui-se a dependência de combustíveis, podendo migrar para facilmente para formas alternativas de obtenção de energia. Esses veículos otimizam sua fonte de energia, pois podem se ajustar sua velocidade para o melhor consumo energético possível. Assim os veículos automatizados também são benéficos ao meio ambiente, ajudando a reduzir os níveis de emissões, tanto em motores de combustão, buscando a melhor queima possível, e em veículos elétricos. (UROOJ, FEROZ E AHMAD; 2017).
- **Congestionamentos:** Haveria a diminuição de congestionamentos, pois o veículo autônomo pode traçar diferentes rotas, buscando o melhor caminho, evitando pontos de lentidão. Podem melhorar também a mobilidade, pois podem operar em uma velocidade maior, mantendo a segurança. Com os veículos autônomos a áreas destinas para estacionamento devem diminuir, pois o car-

ro pode transportar uma pessoa ao seu destino e logo em seguida voltar para a casa dessa pessoa, e quando solicitado voltaria a buscar o indivíduo. (UROOJ, FEROZ E AHMAD; 2017).

- **Aceitação**: Os grupos mais propensos a aceitação dos veículos autônomos são os mais jovens, pois aceitam novos tecnologias com maior facilidade, e pessoas idosas, cujo reflexos e movimentos já não são mais tão eficientes para condução e veriam nesse tipo de veículo uma oportunidade de continuar fazendo suas tarefas, sem a dependência de outras pessoas. Por outro lado, ainda há pessoas que gostam de dirigir, que seriam relutantes de deixar de conduzir o veículo. (LITMAN,2018).
- **Falhas de software e hardware:** Em sistemas complexos, há maiores chances de ocorrerem falhas, uma distorção de sinal, erro no software, podem ocasionar falhas que podem causar acidentes. (LITMAN,2018).
- **Uso indevido da tecnologia:** Veículos podem ser vítimas de manipulação maliciosa, onde o sistema pode ser “hackeado” com intenções criminosas. (LITMAN,2018).
- **Conforto:** Com o veículo autônomo as pessoas se sentiriam mais confortáveis em transitar, mas com o conforto excessivo as pessoas se tornam descuidadas. Sabendo que o carro autônomo é seguro, deixariam de usar cintos de segurança, pedestres andariam de modo descuidado. (LITMAN,2018).
- **Comboio de veículos**: Nos veículos autônomos não seria necessária a distância de segurança entre um veículo e outro, pois havendo a comunicação entre eles, o veículo que vem atrás saberia a hora que o veículo na frente pararia, mas essa situação só é possível se todos os veículos forem autônomos. Se em uma frota de veículos autônomos, um motorista humano se colocar entre eles, a chance de um acidente acontecer é maior. (LITMAN,2018).
- **Tempo maior de viagem:** Tendo um maior conforto e conveniência, os veículos devem aumentar a distância percorrida e o tempo de viagem, sendo submetido assim a uma chance maior de ocorrer um acidente. (LITMAN,2018).
- **Infraestrutura:** Para que o veículo possa se comunicar com a via, investimento na infraestrutura devem ser feitas. (UROOJ, FEROZ E AHMAD; 2017).

- **Legislação e implicações éticas:** Em caso de colisão com vítima fatal, quem deve assumir a culpa, o motorista ou a empresa fabricante do veículo, ainda as implicações legais não estão bem definidas. (UROOJ, FEROZ E AHMAD; 2017).

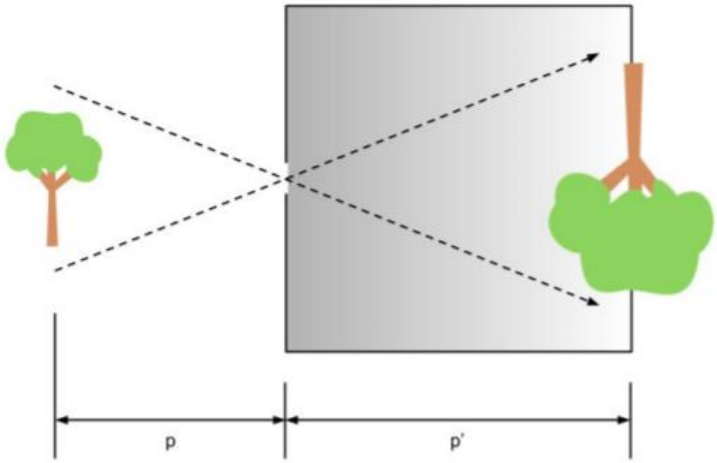
Apesar de haver diversas questões ainda a serem estudadas e melhoradas, é visível os benefícios que os veículos autônomos possuem, mesmo com alguns fatores como falha de hardware e software, e *"hacking"*, os números de acidentes seriam muito inferiores aos 90% de acidentes que podem ser evitados. (LITMAN,2018).

2.2 CÂMERAS PARA CAPTURA DE IMAGENS

As câmeras digitais seguem os mesmos princípios das câmeras fotográficas antigas, onde possuem uma camara escura e um orifício com uma lente feita de vidro, que permite que a luz entre. (BARELLI, 2018)

O orifício é preenchido por uma lente convergente, assim diversos feixes de luz são redirecionados para um único ponto. A lente permite que a luz da imagem a ser captada penetre a câmera e propague em seu interior, assim é projetada uma imagem invertida, na parte interna oposta ao orifício. A Figura 3 ilustra esse funcionamento. (BARELLI, 2018)

Figura 3 – Esquema de uma câmara escura



(Fonte: Introdução à Visão Computacional; 2018; pg:21)

Para que imagem projetada fosse guardada, utilizava-se em câmeras analógicas, filmes com químicas sensíveis a entrada de luz. Nas câmeras digitais são utilizados sensores eletrônicos que também são sensíveis a luz, onde convertem a imagem analógica para uma imagem digital. (BARELLI, 2018)

Os sensores presentes nas câmeras digitais convertem a luminosidade de uma imagem capturada, para valores numéricos, onde são ordenadas em linhas e colunas e com sua representação de cor, formando a imagem. (BARELLI, 2018)

2.2.1 CÂMERAS RGB-D

RGB-D vem do inglês *Red* (Vermelho)*, Green* (Verde)*, Blue* (Azul) *plus Depth* (profundidade), ou seja, câmeras com esse tipo de sensor, diferentemente das câmeras tradicionais, além de apresentarem uma imagem, seus pixels trazem informações sobre a profundidade de cada pixel contido na imagem. As câmeras RGB-D mais antigas chegavam a custar cerca de $10.000,00 dólares, atualmente é possível encontrar câmeras por menos de $200,00 dólares por exemplo. (LITOMISKY, 2012).

2.2.1.1 SENSOR KINECT MICROSOFT

O sensor Kinect foi criado pela empresa Microsoft, em conjunto com uma empresa israelita chamada PrimeSense, que usa como princípio o NUI (*Natural User Interface*), ou interfaces naturais, que utiliza como comandos a linguagem natural humana, a fala, gestos, movimentos do corpo. Na Figura 4 podemos ver o sensor da PrimeSense. (CARDOSO, 2017) (LITOMISKY, 2012).

Figura 4 – Sensor PrimeSense

.

(retirado de:
http://images.eurogamer.net/articles//a/1/0/2/8/1/4/4/PrimeSensor_depth_audio_diagram.jpg.jpg)
Acesso em: 08/05/2017

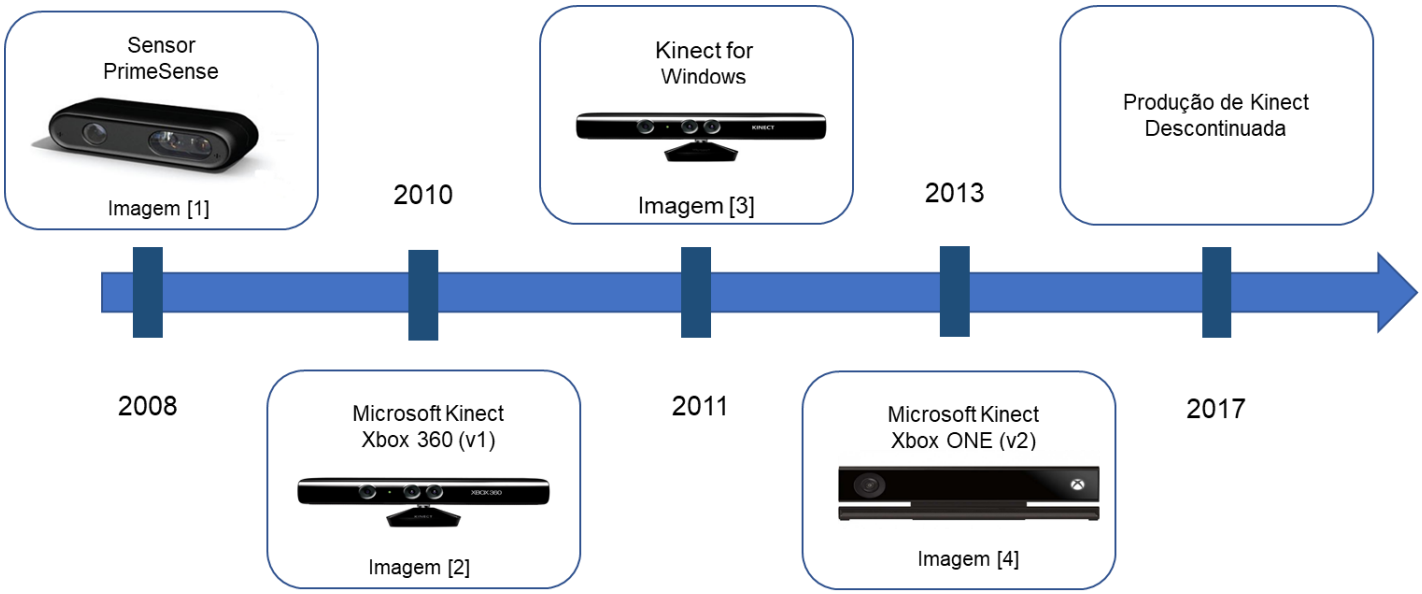
O Kinect é como qualquer outra câmara RGB-D, fornecendo informações de cor e profundidade de cada pixel, porém com um custo menor em relação aos seus antecessores, por conta disso o interesse em câmeras de RGB-D foi renovado, podendo ser usado em reconstruções, realidade virtual e mapeamento 3D. (LITOMISKY, 2012).

Cardoso (2017), ressalta que o Kinect herdou grande parte de suas características do seu antecessor, porém se tornou mais robusta e com um novo *design* foi considerada uma revolução na indústria de jogos pois criou novas formas de jogo e uma nova forma de jogar. Seu nome de origem era Natal, referência à cidade natal de um de seus idealizadores, o brasileiro Alex Kipman, além de Natal em latim significa “nascer”.

Foram colocados no mercado 3 tipos de Kinect, contudo sua produção foi descontinuada pela Microsoft em 2017.

Na Figura 5 uma linha do tempo do sensor Kinect, desde sua concepção pela empresa PrimeSense, posteriormente sua parceria com a Microsoft e o lançamento do Kinect para Xbox 360, assim como o Kinect para Windows, console lançado especialmente para uso em computador. Após alguns anos a Microsoft trouxe o Kinect para Xbox ONE, mas devida as baixas vendas descontinuou a produção do Kinect.

Figura 5 – Linha do tempo da evolução do sensor Kinect

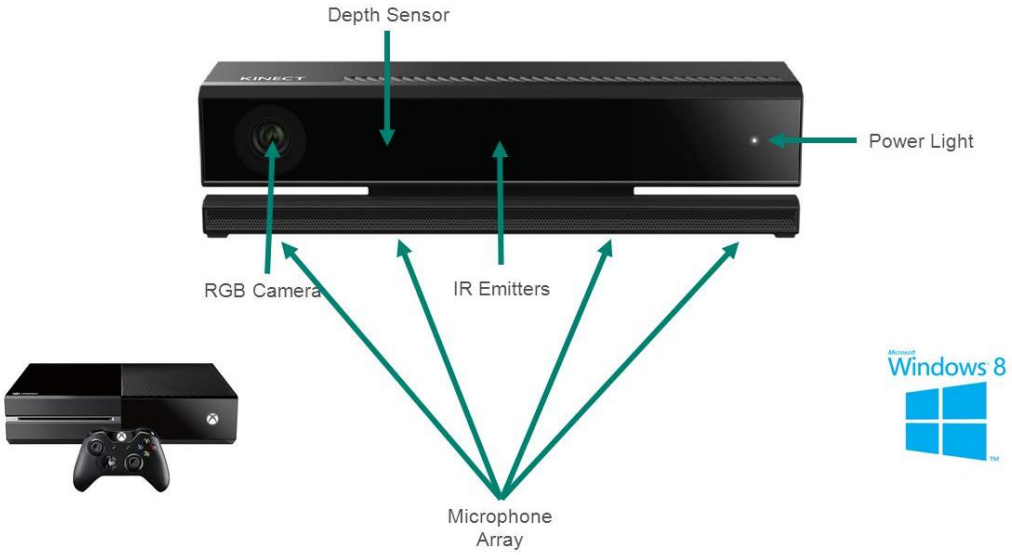


(Fonte: Autoria Própria)

O modelo Kinect foi o primeiro a ser lançado, o Kinect para Windows foi lançado para ser utilizado em um computador, tendo microfones de melhor qualidade e sensores aprimorados para curtas distâncias, em especial se o usuário estiver sentado. Já o Kinect Xbox 360 e Xbox ONE, também podem ser utilizados para o computador, porém somente utilizando um adaptador comprado a separadamente (LITOMISKY, 2012).

O sensor é composto por câmera RGB-D, emissores infravermelhos, sensor de profundidade e conjunto de microfones, na Figura 6 é possível ver um pouco do Kinect v2 e os seus componentes.

Figura 6 – Kinect para Xbox ONE, seus componentes e suas respectivas localizações



(retirado de: http://slideplayer.com/3357546/12/images/7/What+is+Kinect+Depth+Sensor+Power+Light+RGB+Camera+IR+Emitters.jpg) Acesso em: 25/03/18)

2.2.1.2 O KINECT V2

Nesse capitulo será mostrado as especificações e caracteristicas do Kinect para Xbox ONE, também chamado de Kinect V2. (ZENNARO *et al*, 2015).

Na Tabela 2 a seguir podemos ver as caracteristicas técnicas do sensor.

Tabela 2 –Especificações Técnicas do Kinect v2

	Kinect v2	
	Resolução	Taxa de Quadros
	[Pixel X Pixel]	[Hz]
cor	1920 x 1080	30
profundidade	512 x 424	30
infravermelho	513 x 424	30

Adaptado de: Comparison of Kinect v1 and v2 Depth Images in Terms of Accuracy and Precision. Wasenmuller e Stricker.2016. Pg.6)

O Kinect v2 obtém as informações de profundidade através de algo chamado Tempo de Voo, que mede o tempo que leva pra que a luz emitida atinja um objeto e volte novamente. (WASENMÜLLER E STRICKER, 2016) (BUTKIEWICZ, 2014) (ZENNARO *ET al*, 2015).

Em termos de influência ao calor recomenda-se que deixe o Kinect v2 ligado antes de se iniciar as atividades, apesar de ele possuir um cooler ele pode sofrer com variações de temperatura se não houver seu aquecimento. (WASENMÜLLER E STRICKER, 2016).

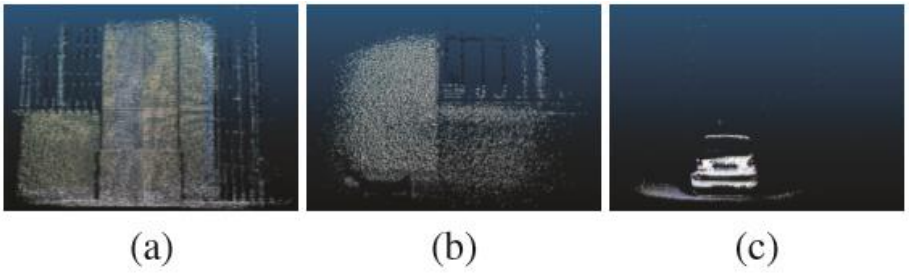
Tendo um aumento na distância em relação ao objeto, o Kinect v2 possui um offset por pixel, ele sendo variável, pois depende de sua localização na imagem, quanto mais longe do centro e mais perto das bordas esse offset é maior, esse comportamento é devido ao cone de infravermelho, que não ilumina o cenário de forma homogênea. (WASENMÜLLER E STRICKER, 2016).

Havendo o aumento da distância entre a câmera e o cenário, o Kinect perde precisão, mas mantem sua acurácia, tornando o offset constante fundamental para fazer compensações e modelagens sobre as imagens. (WASENMÜLLER E STRICKER, 2016).

Em ambientes externos, a luz solar deve ser uma variável a ser considerada, pois influencia na qualidade da imagem. (ZENNARO *et al*, 2015).

A Figura 7 mostra o desempenho do Kinect v2 em diferentes luminosidades. Figura (a) parede sombreada, (b) parede sob a influência da luz solar, (c) carro sob luz solar.

Figura 7 – Desempenho do Kinect v2, em ambientes diferentes com luminosidade variada.



(Adaptado de: Comparison of Kinect v1 and v2 Depth Images in Terms of Accuracy and Precision. Pg 6)

O Kinect v2 consegue obter as referências de profundidade de um cenário em até mesmo 3 metros de distância, isso em sua pior condição possível, sendo em um dia com constante e elevada luz solar, apontado para uma parede branca, onde há grande índice de reflexibilidade. Retirada a parede os resultados podem ser obtidos a até 4 metros. Em um dia nublado a distância pode chegar a 4,5 metros. (BUTKIEWICZ, 2014).

Outro fator de influência ao mapear um ambiente são as cores do ambiente em questão. O Kinect v2 as cores mais escuras apresentam um valor maior de profundidade do que as cores claras, assim cores menos reflexivas são menos confiáveis de estimar. Para obter melhores resultados do offset de profundidade é reco-

mendado o uso de cores com reflexividade similares. (WASENMÜLLER E STRICKER, 2016)

Ao falar em veículo autônomos, é de suma importância considerar a presença humana, como os pedestres, tem-se a incumbência de identificá-los em tempo hábil e assim poder se esquivar dos mesmos, ou até mesmo fazer sua parada em casos de emergência. Após testes realizados em um ambiente externo o Kinect v2 conseguiu obter a nuvem de pontos das pessoas presentes, em uma distância maior do que 4 metros. (ZENNARO *et al*, 2015).

2.3 IMAGENS

Já a resolução refere-se à quantidade de informação que a imagem comporta, seu grau de detalhamento, e sua qualidade apresentada. É dividida em três tipos: espacial, de intensidade ou resolução de bit e temporal (BACKES e JUNIOR,2016).

- **Espacial:** é a quantidade de pontos que são utilizados para representação da imagem, podendo ser medida em dpi (*dots per inch* – pontos por polegada), onde quantifica a densidade de pontos da imagem. Também pode ser utilizada número de pixel na horizontal pelo número de pixel na vertical, sendo o produto o número de pixels total da imagem.
- **Intensidade ou Resolução de bit:** determina o valor de intensidade de cor que um pixel pode ter. Como em uma imagem preto e branco, sendo 0 e 1, essa imagem possui uma resolução de 2 níveis.
- **Temporal:** é utilizada em sucedidas capturas de imagens, como em vídeos, representando quantas imagens são capturadas em um período. Geralmente utilizando a unidade qps (quadros por segundo), cada quadro representa uma única imagem.

As imagens podem ser agrupadas em alguns tipos. A imagem binária é formada por apenas duas intensidades preto e branco, sendo representadas por 1 e 0. Assim cada pixel é representado juntamente com seu valor numérico. Na Figura 12, mostra a representação de uma imagem, juntos com os valores dos pixels (BARELLI, 2018).

Figura 8 – Imagem já binarizada

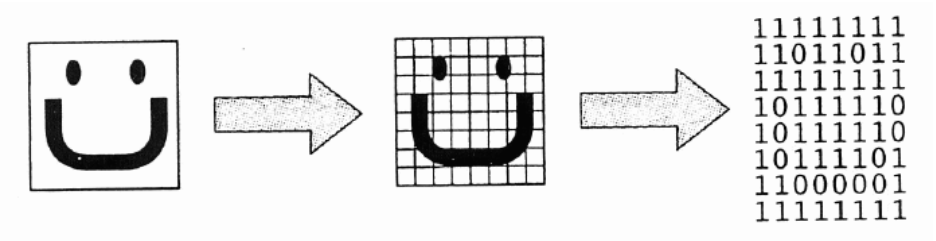


Imagem adaptada de (http://www.cse.buffalo.edu/~rapaport/111F04/111-bitmap.gif). Acesso em: 28/05/2018

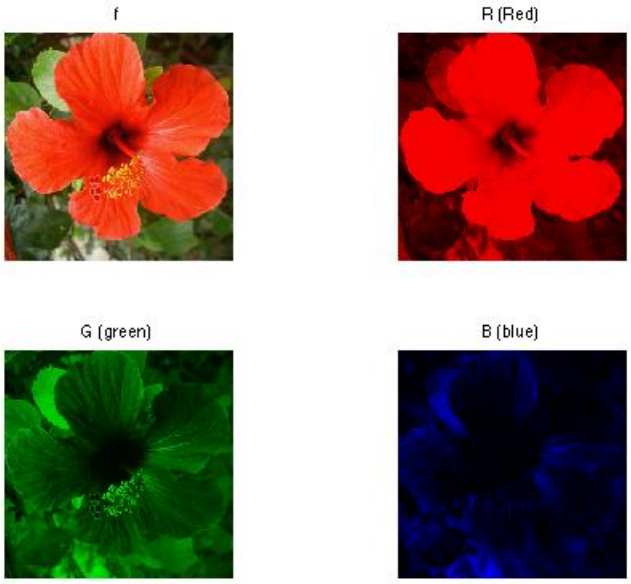
Nas imagens em escala de cinza, os pixels são representados conforme a intensidade de cinza, variando entre 0 e 255, sendo 0 preto e 255 branco. Muito utilizada em visão computacional, pois consegue-se identificar a imagem, e seu processamento é muito mais leve e rápido. A Figura 13, exemplifica uma imagem em escala de cinza.

Figura 9 – Exemplo de imagem em escala de cinza

(Retirado de: https://i.stack.imgur.com/kP0u2.png) . Acesso em: 22/05/2018

As imagens coloridas ou RGB são imagens coloridas, também chamadas de imagens reais. Utilizam os três filtros de cores primárias, vermelho, verde e azul. Cada pixel é representado por três valores inteiros de 0 a 255, correspondentes a cada umas das cores primarias, assim a combinação das cores e de suas intensidades formam todas as outras cores. Em visão computacional esse tipo de imagem não é muito utilizada, por ter um processamento mais lento. Na Figura 14, mostra-se a decomposição de uma imagem RGB (BACKES e JUNIOR,2016).

Figura 10 – Decomposição RGB de uma imagem



(Retirado de: http://www.numerical-tours.com/matlab/multidim_1_color/index_01.png) Acesso em: 24/06/2018

2.4 VISÃO COMPUTACIONAL

A visão computacional é uma área de estudo que busca repassar para maquinas a capacidade de visão, não só a habilidade de captar imagens, mas de processá-las e melhorá-las afim de obter diferentes informações sobre uma imagem, como profundidade, cor, textura, forma, e relacioná-las com imagens já vistas. (BACKES e JUNIOR; 2016). Na Figura 8 tem-se o ciclo do sistema de visão computacional dividido em fases, onde cada fase pode receber nomes diferentes de acordo com cada autor.

Figura 11 – Fluxo de um sistema de visão computacional

(Fonte: Introdução à Visão Computacional; 2018; pg:6)

As fases de um sistema de visão computacional são:

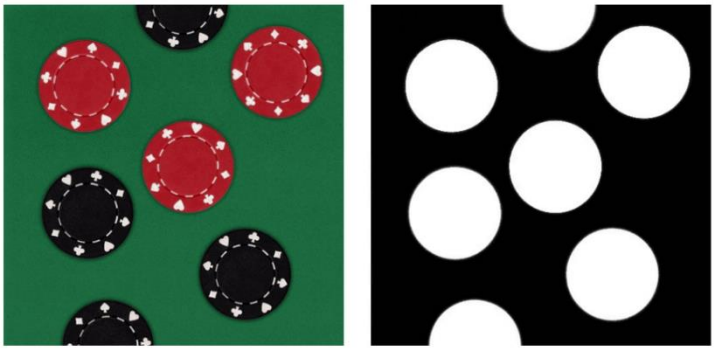
- **Aquisição**: É responsável pela captação e digitalização das imagens com quais vão ser trabalhadas, utilizando ferramentas como câmeras fotográficas, scanners, filmadoras. As imagens digitalizadas podem ser em 2D, 3D ou até mesmo sequências de imagens, como em vídeos. (BARELLI, 2018)(BACKES e JUNIOR,2016). Na Figura 9 podemos ver alguns exemplos de ferramentas de captura, onde o ultrassom e o raio x não necessitam de luz para a obtenção da imagem. (BARELLI, 2018)(BACKES e JUNIOR,2016).
-

Figura 12 – Ferramentas de captura de imagem

(Fonte: Introdução à Visão Computacional; 2018. Pg: 2)

- **Processamento de imagens ou Pré-processamento**: Nessa etapa são identificados regiões e objetos de interesse, que são partes ou elementos nas imagens que se quer obter informações, após definida as regiões de interesse a imagem é melhorada, retirando ruídos, retirando imperfeições, além de ser responsável pela rotação da imagem assim como filtragens. Na Figura 10 tem-se a imagem de fichas de poker, onde as fichas são os objetos de interesse, portanto foram realçadas, e o fundo retirado. (BARELLI, 2018)(BACKES e JUNIOR,2016).

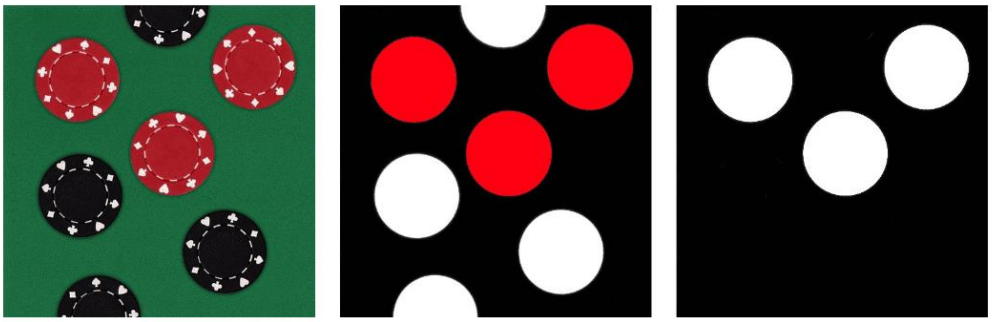
Figura 13 – Exemplo de processamento de imagem



(Fonte: Introdução à Visão Computacional; 2018. Pg: 8)

- **Segmentação**: fase responsável por separar a imagem original dos seus segmentos de interesse. Na Figura 11 mostra as mesmas fichas de poker da fase anterior, mas foi selecionado somente as fichas vermelhas, então houve o pré-processamento, para definir que as fichas vermelhas são os objetos de interesse, e assim somente elas foram evidenciadas, e o restante foi descartado. (BARELLI, 2018)(BACKES e JUNIOR,2016).

Figura 14 – Segmentação de uma imagem



(Fonte: Introdução à Visão Computacional; 2018. Pg: 8)

- **Extração de características/Análise de imagens:** essa etapa consiste em retirar características do objeto de interesse, como área e diâmetro, podem ser utilizadas para diferenciar outros objetos em cena. Essa fase também é responsável por encontrar uma codificação numérica de cada imagem, para a identificação. (BARELLI, 2018)(BACKES e JUNIOR,2016)
- **Reconhecimento de Padrões:** classifica e agrupa em classes as imagens segmentadas de acordo com as suas características. Como na figura 12 pode-se classifica-la em classes como “ficha de poker vermelha” e “objeto circular vermelho”, utilizando seus atributos e baseado em objetos já vistos antes com características similares. (BARELLI, 2018)(BACKES e JUNIOR,2016)

Para o processamento de imagens realizado na visão computacional, alguns elementos fundamentais do processo de obtenção dessas devem ser discutidos. Como foi demonstrado na Figura 9, existe um fluxo de tratamento para as imagens, antes da aplicação dos algoritmos (BARELLI, 2018)(BACKES e JUNIOR,2016).

A aquisição de imagens digitais é feita por sensores, câmeras e scanners. É a primeira etapa do fluxo do sistema de visão computacional. Os sensores dividem a imagem em pixels, que são as unidades fundamentais para a formação de uma imagem, assim cada pixel recebe um número que contém a informação de tonalidade da imagem (BACKES e JUNIOR,2016).

2.5 REDES NEURAIS

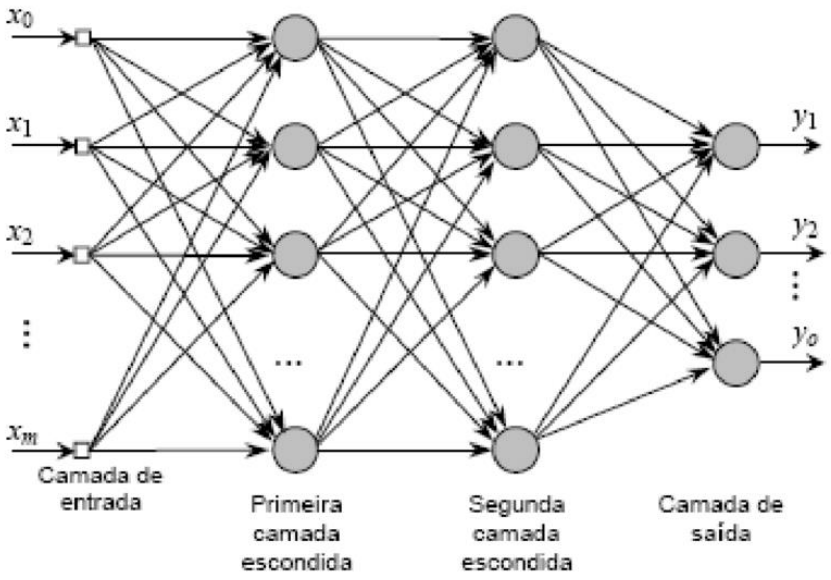
Redes neurais são modelos de estruturas computacionais baseado no funcionamento do cérebro, são um exemplo de um algoritmo de aprendizado de máquina (*Machine Learning*). (CIABURRO, VENKATESWARAN, 2017) (GRANATYR, 2016)

O cérebro consegue processar uma quantidade imensa de dados enviados por todos seus sentidos, em especial a visão. Os neurônios são responsáveis por esse processo, permitindo que um impulso elétrico passe ou não por outro neurônio. (CIABURRO, VENKATESWARAN, 2017)

Baseado nesse conceito foi desenvolvido estruturas de tomadas de decisão para a resolução de problemas e *machine learning*. A rede neural tem o objetivo de gerar uma única saída, a partir de diversos dados de entrada, onde o neurônio é uma unidade central de processamento que opera matematicamente para gerar o dado de saída. A saída do neurônio é dada pela soma ponderada de todas as entradas mais o *bias* daquela entrada. (CIABURRO, VENKATESWARAN, 2017).

Na Figura 15, mostras as camadas de uma rede neural simples, onde temos as entradas (*input*), a camada do meio, ou também chamada de camada oculta (*middle or hidden layer*), e por fim a camada de saída (*output*)

Figura 15- Rede neural multicamadas



(Fonte: Neural Network with R, 2017)

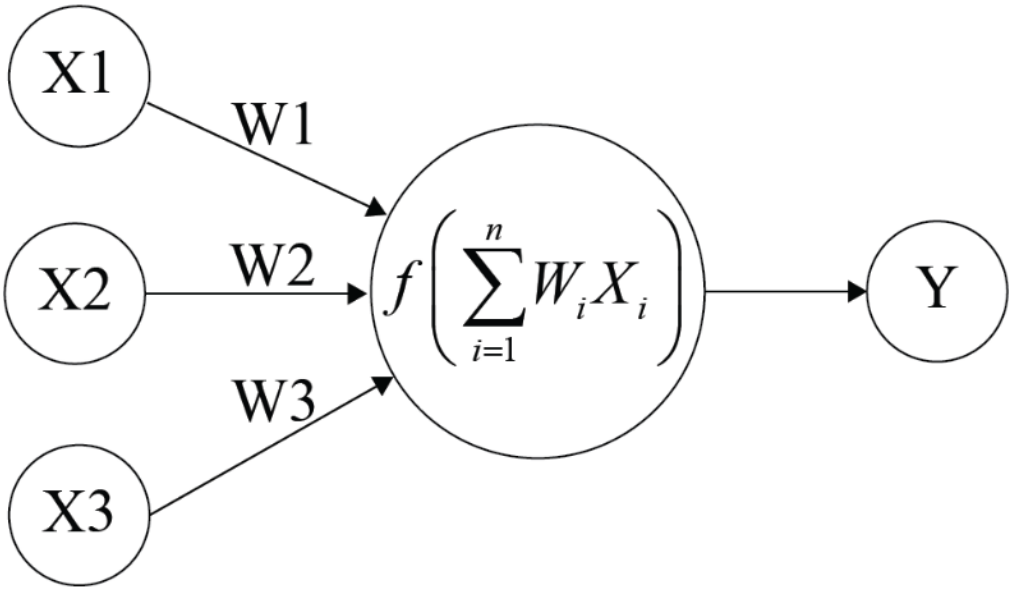
A camada do meio é a responsável por converter os dados de entrada para dado de saída. Essa camada contém pesos, o *bias*, e a funções de ativação. (CIABURRO, VENKATESWARAN, 2017).

A seguir podemos observar o processo de um neurônio através da Equação 1.

$$saída = (pesos * entradas) + bias \quad \text{Equação 1}$$

Uma função é aplicada nessa saída, chamada de função de ativação, a entrada da próxima camada é a saída do neurônio da camada anterior, como mostrada na Figura 16, onde X são as entradas, W são os pesos, e Y é a saída. (CIABURRO, VENKATESWARAN, 2017) (GRANATYR, 2016)

Figura 16– Função de ativação de uma rede neural



(Fonte: Neural Network with R, 2017)

2.5.1 REDES NEURAIS CONVOLUCIONAIS

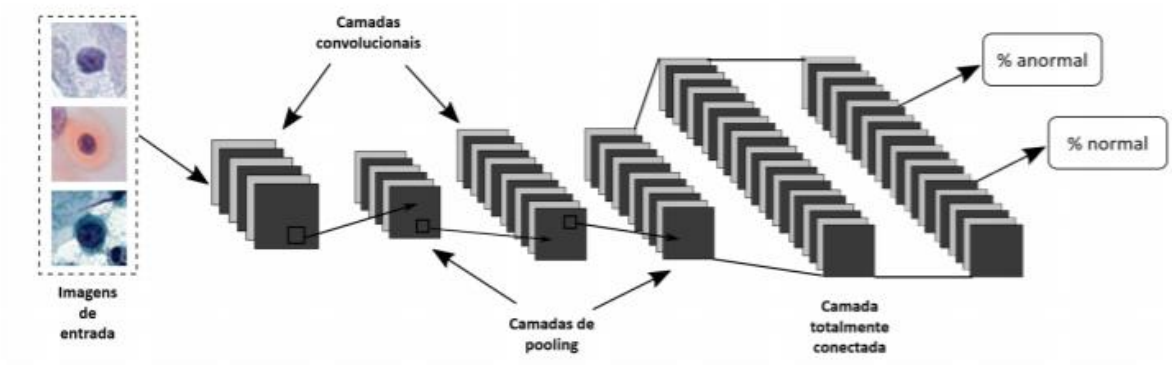
As redes neurais convolucionais (também chamadas de *ConvNets* ou CNN) estão sendo amplamente utilizadas em visão computacional, pois são facilmente treinadas quando se necessita de uma grande quantidade de amostras, e necessita-se classifica-las e determinar a quais das diversas classes existentes essas amostras pertencem. (ARAUJO *et al*, 2017)(VARGAS, PAES, VASCONCELOS. 2016).

As principais vantagens envolvendo as CNN consistem na capacidade de extração das principais características, utilizando de aprendizado de transformação, chamado de *kernels*, assim como a dependência de menos parâmetros de ajustes, como em redes totalmente conectadas. (ARAUJO *et al*, 2017)

Em CNN, as unidades de uma camada não se conectam com todas as unidades das outras camadas, assim utiliza-se de menos pesos para atualização, tornando o aprendizado mais fácil. (ARAUJO *et al*, 2017)

As redes neurais convolucionais atuais compartilham das características de um projeto de CNNs chamada de *LeNet*, proposta por Yann LeCun. Esse projeto ajudou a impulsionar as atividades em *Deep Learning*. Na Figura 17, ilustra a arquitetura de uma *LeNet*, para a classificação de células, onde é dividida em três principais camadas, as convolucionais, de *pooling* e as totalmente conectadas. (ARAUJO *et al*, 2017) (LOPES, 2017)

Figura 17- Arquitetura de uma *LeNet*



(Fonte: Livro Anais - Artigos e Minicursos, 2017)

As camadas convolucionais extraem os atributos das entradas. As camadas de *pooling* reduzem o tamanho dos dados extraídos pelas camadas convolucionais, tornando assim as representações menos suscetíveis as variações na entrada. Já a camada totalmente conectada propaga o sinal, multiplicando ponto a ponto e utilizando uma função de ativação. Na saída obtém-se a probabilidade da imagem de entrada pertencer a alguma das classes que foram treinadas. (ARAUJO *et al*, 2017)

Outras arquiteturas de CNN, que vieram posteriormente ao *LeNet* utilizam basicamente suas mesmas camadas. (ARAUJO *et al*, 2017)

2.5.2 YOLO (YOU ONLY LOOK ONCE)

Quando olhamos uma imagem, imediatamente já conseguimos reconhecer que tipo de objeto é aquele, sua localização e seu comportamento. A visão humana é rápida e possui uma alta acurácia, permitindo assim realizar tarefas complexas como por exemplo dirigir. (REDMON *et. al,*2016).

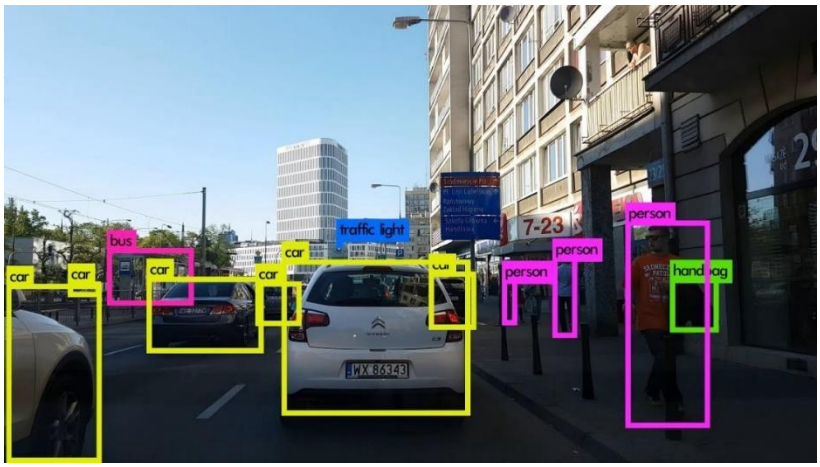
A utilização de algoritmos de detecção permitiu que carros pudessem ser conduzidos por computadores sem a necessidade de sensores específicos para essa tarefa, realizando tarefas em tempo real. (REDMON *et. al*, 2016).

Os sistemas atuais de detecção e classificação de imagens, geram uma caixa delimitadora ao redor do objeto em cena e assim classificam esse objeto, utilizam de pós processamento para refinar as caixas e eliminar qualquer detecção duplicada, necessitando olhar a imagem diversas vezes. Esse sistema é lento e difícil de ser otimizado, pois deve haver um treinamento individual para cada componente. (REDMON *et. al*, 2016)(PONNUSAMY, 2018)

O YOLO (*You Only Look Once* – Você Só Olha uma Vez) reformulou a detecção de objetos, seu nome vem da necessidade de se olhar uma imagem uma única vez para detecção e classificação. Utiliza unicamente um problema de regressão, onde uma única rede convolucional prevê simultaneamente desde a localização da caixa delimitadora e as probabilidades de qual classe pertence o objeto. (REDMON *et. al*, 2016) (PONNUSAMY, 2018).

Na Figura 18, é mostrado um exemplo do funcionamento do YOLO. Onde os objetos identificados são marcados por uma caixa delimitadora colorida, e a classe à qual o objeto é pertencente, é colocado acima da caixa.

Figura 18 – Exemplo de funcionamento YOLO



(Retirado de: https://cdn-images-1.medium.com/max/2000/0*HMacEfECt2PYQOxF.jpg). Acesso em: 24/10/2018

O YOLO é muito rápido, desde o molde de detecção como um problema de regressão, roda-se a rede neural em uma nova imagem no teste para a previsão de detecções, também podendo alcançar o dobro da precisão média de outros sistemas de tempo real. Porém precisa de um esforço maior para a detecção de objetos pequenos, perdendo precisão, mas esse problema foi melhorado em sua versão mais atual. (REDMON *et. al*, 2016)(KATHURIA, 2018)

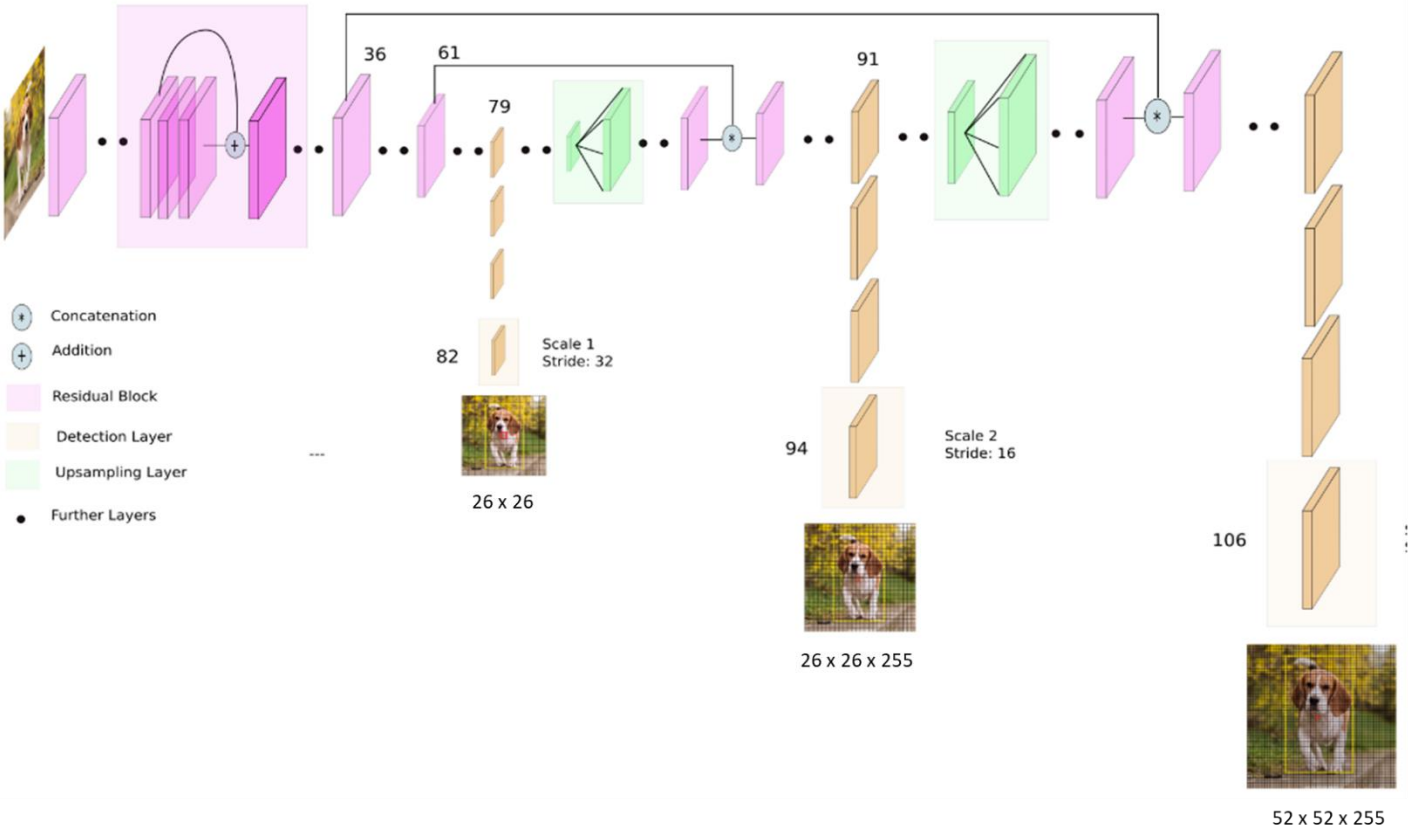
Existem até agora 3 versões do YOLO, a utilizada nesse trabalho foi a v3. O YOLO v3 usa uma variante do *Darknet*, que a princípio tinha 53 camadas, treinadas no *Imagenet* (banco de imagens). Para a detecção foram adicionadas mais 53 camadas, tornando a arquitetura subjacente totalmente convolucional com 106 camadas. (KATHURIA, 2018)

Na Figura 19, mostra a arquitetura da rede YOLO v3. Onde mostra a detecção em três camada distintas. Sua saída final é gerada ao aplicar um *kernel* 1X1, no mapa de recursos de três tamanhos e localizações diferentes. (KATHURIA, 2018)

A forma do kernel de detecção é dada por 1x1x (B x (5+C)), onde B representa a quantidade de caixa delimitadoras que um mapa pode prever. A constante 5 é pelos 4 atributos da caixa, mais um objeto de confiabilidade. Por fim C é a quantidade de classes. (KATHURIA, 2018)

No YOLO v3, adotando os valores de B=3, e C=80, assim temos o kernel resultante é de 1x1x255. (KATHURIA, 2018)

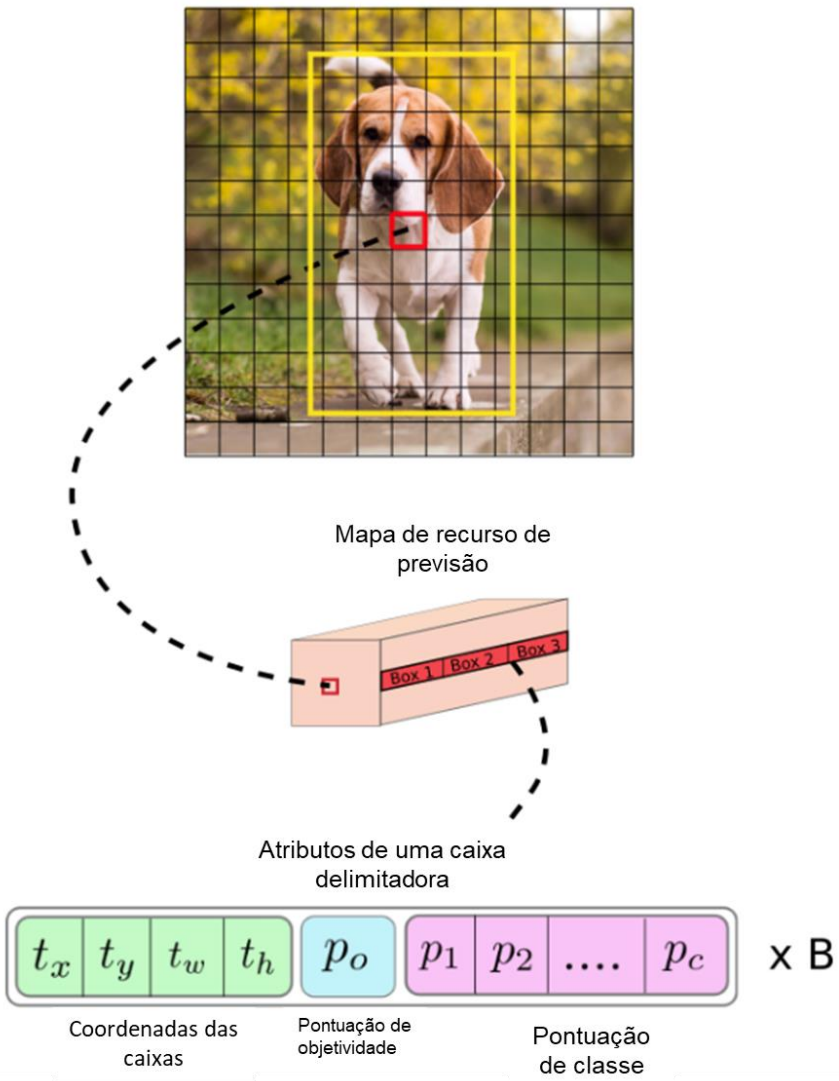
Figura 19 – Arquitetura da rede YOLO v3



(Retirado de: https://cdn-images-1.medium.com/max/2000/0*HMacEfECt2PYQOxF.jpg).
Acesso em: 24/11/2018

Uma camada define-se como a proporção a qual ela diminui a entrada. A Figura 19 e 20 ilustra os exemplos dados. Onde na entrada a imagem de exemplo seu tamanho é 416 X 416. (KATHURIA, 2018).

Figura 20 – Grade de uma imagem



(Adaptado de: https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/). Acesso em: 27/11/2018)

O YOLO utiliza de predição em três escalas, que reduzem as resoluções das dimensões da imagem de entrada, por 32, 16 e 8. A primeira detecção é feita na 82ª camada, onde a imagem é reduzida por meio de amostragem, onde a 81ª possui um passo de 32. Utilizando a imagem exemplo de 416 X 416, o mapa resultante seria de

13 X 13. Utilizando o mapa de kernel o mapa resultante de detecção seria de 13 X 13 X 255, como mostrado na Figura 19. (KATHURIA, 2018)

Na camada 79, o mapa de feições é submetido a camadas convolucionais, depois são amostradas duplamente para dimensões de 26X26, resultando em um mapa de recurso, onde é novamente concatenado com o mapa de feições da 61ª camada. Os mapas de feição são então concatenados e submetidos a mais camadas convolucionais. A segunda detecção então é feita na camada 94, resultando em um mapa de detecção de 26X26X255 (Figura 19). (KATHURIA, 2018).

Novamente, na camada 91 passa por algumas camadas convolucionais, para ser concatenado em profundidade junto ao mapa de feições da camada 36. Camadas convolucionais são adicionadas a anterior. No final das três escalas na camada 106, resulta em um mapa de tamanho 52 X 52 X 255. (Figura 19). (KATHURIA, 2018)

3. METODOLOGIA

Severino (2007) aponta que o objetivo do trabalho de conclusão de curso deve ser fornecer um projeto capaz de por a prova os conhecimentos e habilidades adquiridos ao longo da fase acadêmica de um indivíduo.

Ainda segundo a definição do autor, o objetivo do trabalho, a validação do uso de câmeras RGB-D, para a aplicação de algoritmos de mapeamento e localização simultâneas, comparando seu desempenho em ambientes externos, atende essa proposta.

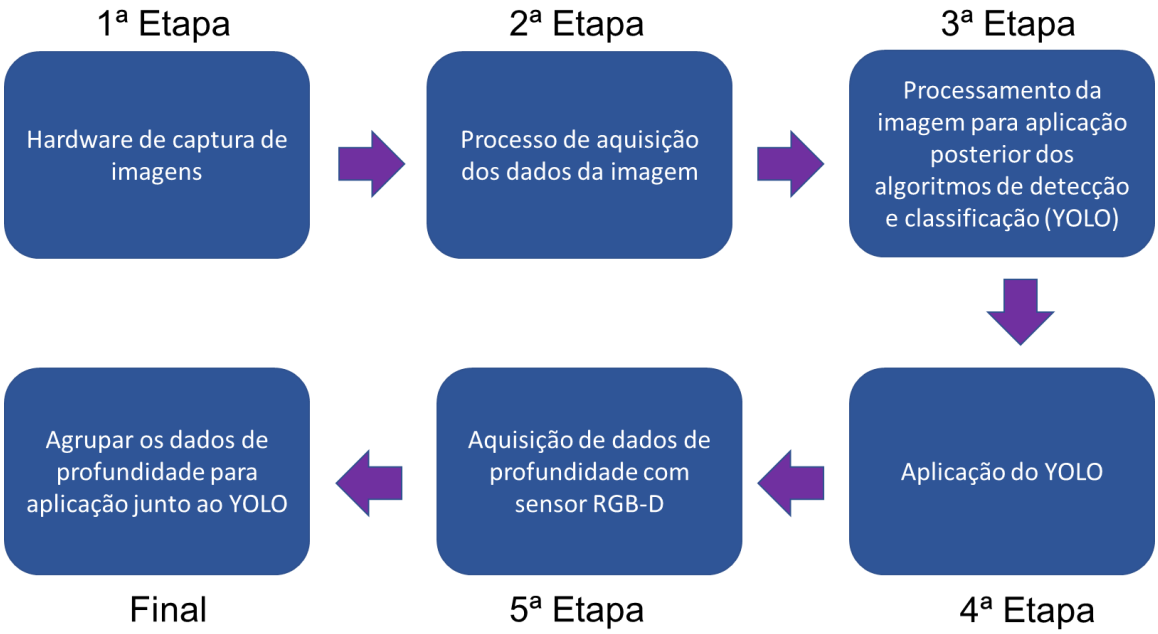
Com o objetivo do trabalho bem definido e com o tema do trabalho direcionado para realizar uma contribuição para o estudo relacionado ao desenvolvimento de veículos autônomos, o conjunto para o desenvolvimento do trabalho estava definido.

Depois dessas definições, a proposta do trabalho foi feita ao professor da disciplina de projetos e, posteriormente ao professor orientador, para que ambos pudessem avaliar a validade do tema proposto em relação as diretrizes bases do curso de Tecnologia em Eletrônica Automotiva.

O desenvolvimento da introdução do trabalho foi realizado levando em consideração o tema proposto e como ele seria problematizado. Gerando o objetivo e a proposta de elaboração do trabalho.

A etapa da fundamentação teórica foi construída buscando referências para elaboração do procedimento metodológico utilizado para a cada uma das etapas do projeto, tais como a escolha do hardware de aquisição de imagens, a escolha dos algoritmos que poderiam ser utilizados para realizar o mapeamento, o processo de aquisição e posterior tratamento das imagens. A Figura 21 apresenta um diagrama de blocos de quais etapas foram abordadas no projeto.

Figura 21 – Fluxo de dados do processamento de informação



(Fonte: Autoria própria)

Para o levantamento bibliográfico, buscou-se um pouco da origem dos veículos autônomos, além de trabalhos que mostram o impacto desses veículos, em especial em suas vantagens associadas à redução de mortes e feridos em acidentes de trânsito. Sem desconsiderar os possíveis obstáculos que a tecnologia enfrenta ou poderá enfrentar. Mostrou-se também os níveis de autonomia de um veículo autônomo, possibilitando diferenciar os veículos conforme a tecnologia presente.

Um veículo autônomo depende de uma capacidade muito importante de se localizar e mapear um ambiente desconhecido. Para que isso ocorra o veículo deve ser dotado de uma capacidade de visão, para o papel de olhos do veículo foi utilizado uma câmera RGB-D.

Uma câmera RGB-D fornece dados de cores e profundidade de cada *pixel* de uma imagem, para este trabalho a câmera usada foi um Kinect da Microsoft de segunda geração.

De nada adiantaria ter a capacidade de ver as imagens e não conseguir entendê-las. Nesse ponto foi introduzida a visão computacional, a câmera sendo os olhos, a visão computacional é o sentido da visão, a capacidade de obter informações so-

bre a imagem captada, poder segmentar conforme o interesse, e classificar a imagem baseada em similaridades. A visão computacional permite identificar pedestres, diferenciando-os de outros veículos e construções.

Na primeira etapa, foi conectado o sensor Kinect v2 ao computador para verificação de compatibilidade, e se havia a ausência de drivers necessários ao uso adequado do sensor, na Figura 18 representado pelo *hardware* de captura de imagem.

Estando tudo adequado as primeiras imagens foram captadas dando conclusão a segunda etapa. Assim sua capacidade de identificação de objetos é testada.

A terceira etapa consistiu na obtenção das primeiras imagens, e o teste do seu processamento testada. Para o algoritmo de processamento, diferentes linguagens podem ser utilizadas junto ao OpenCV. As escolhidas para teste são: C#, linguagem nativa do sensor Kinect, Python e Processing.

Ainda na terceira etapa o sensor Kinect v2 foi ligado, capturando suas imagens e processando elas com os algoritmos de visão computacional, presentes no conjunto de ferramentas da biblioteca OpenCV.

A quarta etapa de testes consistiu em carregar os pesos da API do YOLO para dentro da aplicação com o auxílio da OpenCV. Dessa forma seria possível identificar os elementos presentes na imagem.

Ainda na quarta etapa de testes foi feito o reconhecimento dos objetos e de seu posicionamento nas imagens.

A quinta etapa foi feita a realização de aquisição de dados de profundidade com a câmera RGB-D.

A etapa final agrupa os dados de localização e detecção de imagem junto as informações sobre profundidade. Na Figura 18 referente a agrupar os dados de profundidade e aplicação junto ao YOLO.

Durante este trabalho, as etapas 1 à 4 foram desenvolvidas e são descritas na seção de desenvolvimento, a etapa final ficou sugerida como trabalhos futuros.

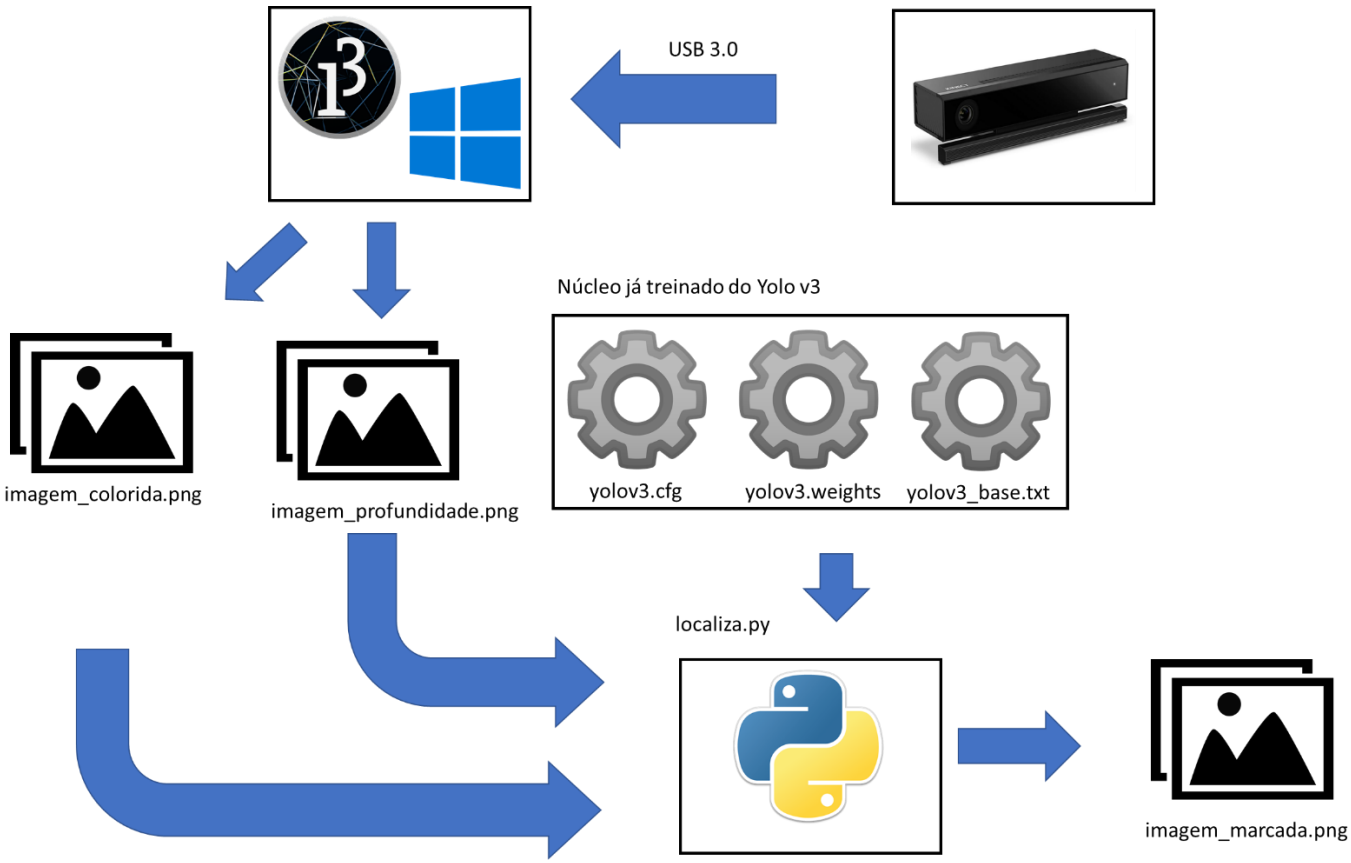
4. DESENVOLVIMENTO

Durante este capítulo foram abordados os procedimentos necessários para o desenvolvimento do projeto. As ferramentas utilizadas e sua integração são aqui descritas, junto de algumas listagens de código essenciais para o entendimento do desenvolvimento proposto.

O código fonte completo da aplicação pode ser encontrado nos Apêndices A e B do trabalho.

A Figura 22, mostra o escopo geral do projeto, onde posteriormente cada etapa será explica em detalhes.

Figura 22 – Esquema do projeto



(Fonte: Autoria Própria)

4.1 PREPARAÇÃO DO AMBIENTE PARA LIGAR O SENSOR

Nessa etapa foram instalados os drivers necessários para a comunicação como Kinect v2 e o computador, foi instalado o KinectSDK v2, fornecida pela própria Microsoft. Além da instalação de outros programas como Python 3.7.0, Processing Kinect Studio, OpenCv, e Sublime Text para edição do código em Python.

O computador utilizado tem Windows 10 *Education* como sistema operacional, uma placa de vídeo dedicada Nvidia GEFORCE GTX. Processador Intel® Core™ i5-7300HQ CPU @ 2.50GHz. Para a conexão do sensor, o computador deve ser dotado de uma porta USB 3.0, portas inferiores não conseguem fazer a comunicação com o sensor. Memória RAM 8,00 GB. Com sistema operacional 64 bits.

4.2 FERRAMENTAS NECESSÁRIAS PARA APLICAÇÃO DOS ALGORITMOS DE VISÃO COMPUTACIONAL

A primeira etapa foi verificar o funcionamento do sensor, para isso foi usado o Kinect Studio, que faz a comunicação com o Kinect diretamente, e permite a gravação de vídeos. Foi verificado se a câmera e o sensor de profundidade estavam funcionando. Constando seus funcionamentos pôde-se começar implementar os códigos de obtenção de imagens.

A principio foi feita tentativas de código em Python, pois essa linguagem era a mais familiar, facilitando a modificação do código e sua implementação. Para a escrita do código em Python foi utilizado o Sublime Text, pois fornece facilidades como marcadores em cores diferentes, permitindo uma melhor visualização do código.

Os códigos em Python encontraram dificuldades de se comunicar com o sensor Kinect, forçando o uso de outras ferramentas. Assim o Processing foi utilizado.

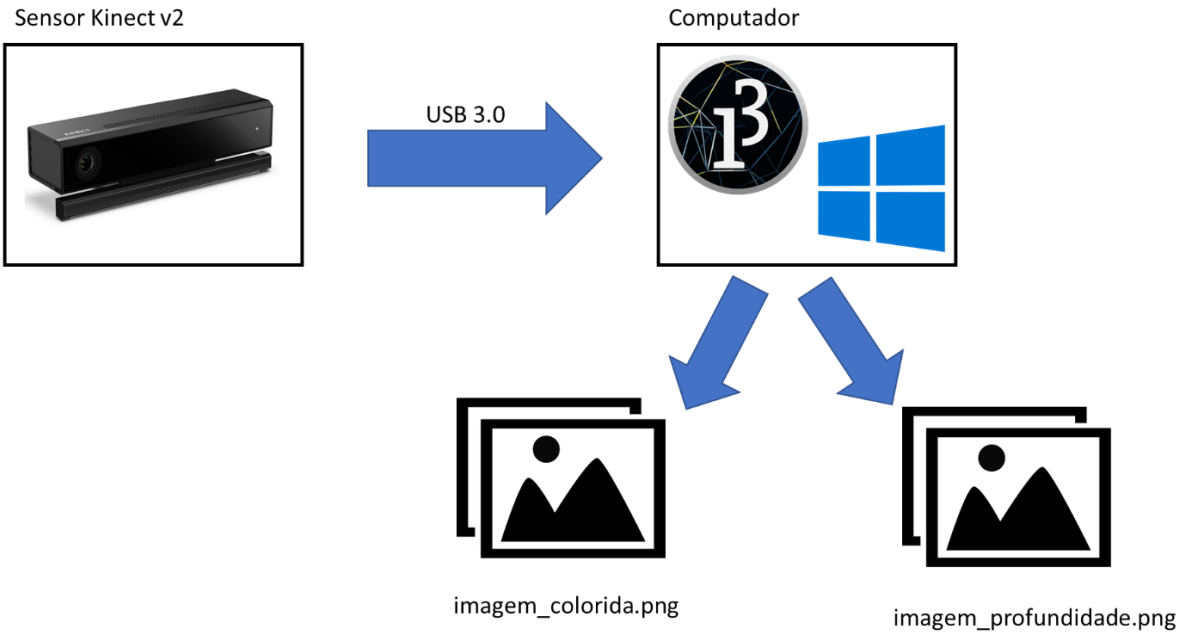
“Processing é um caderno de esboços de software flexível e uma linguagem para aprender a codificar dentro do contexto das artes visuais.” Foi criado para auxiliar no aprendizado em software, promovendo o aprendizado da programação, de modo mais rápido e dinâmico. Na Figura 23, é ilustrado a comunicação do sensor com o Windows 10 e o Processing.

Figura 23- Comunicação Sensor Kinect com Windows 10 e Processing

(Fonte: Autoria Própria)

Com a conexão do Processing e o Kinect, pode-se obter as primeiras imagens, primeiramente estáticas, uma imagem colorida (imagem_colorida.png) e outra imagem de profundidade (imagem_profundidade.png), assim como mostra a Figura 24. Os códigos para a captação das imagens com Processing estão no Apêndice para consulta e implementação.

Figura 24- Obtenção Imagem Colorida e Outra de Profundidade

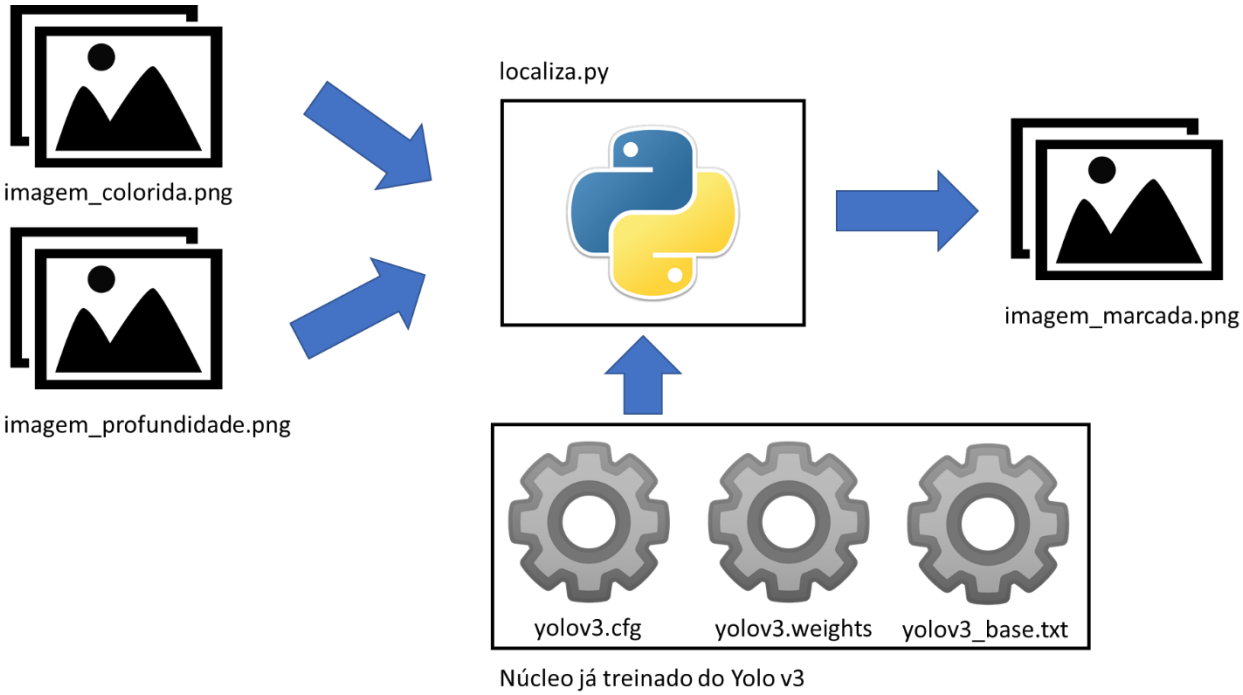


(Fonte: Autoria Própria)

Tendo em posse as imagens de pode-se aplicar os algoritmos de YOLO (yolov3.cfg) mas primeiro deve-se inicializar a rede neural com o treinamento (yolov3_base.txt) e os pesos (yolov3.weights). O código de implementação do YOLO foi escrito em Python A lista de classe e todo o núcleo treinado de YOLO também estão disponíveis no Apêndice.

Ao aplicar o YOLO, o resultado esperado é uma imagem colorida com as detecções, marcações, classificação, e a probabilidade da imagem dentro das caixas limitadoras pertenceram realmente a classe (imagem_marcada.png). E uma imagem de profundidade também com as marcações e classificação Na Figura 25 ilustra a etapa da aplicação do programa.

Figura 25 – Aplicação do YOLO



(Fonte: Autoria Própria)

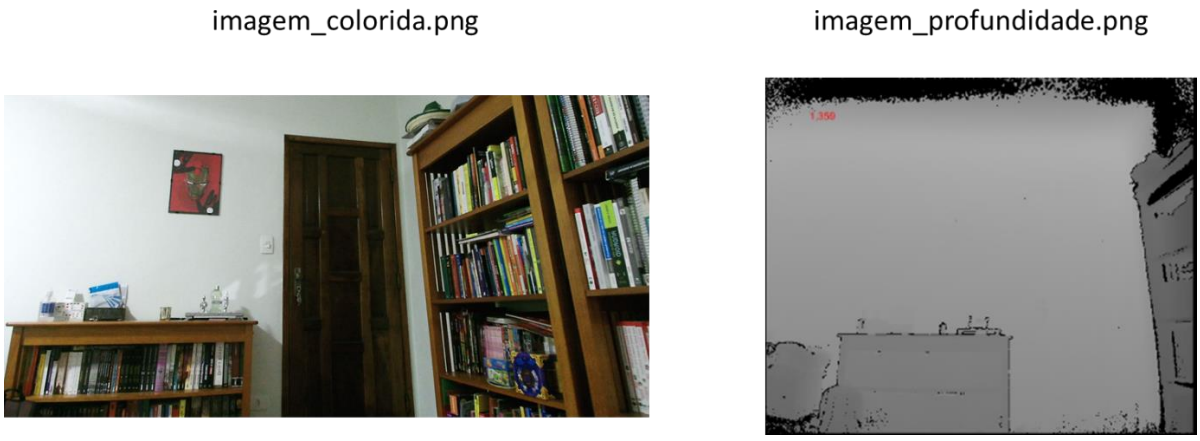
Foram feitas tentativas para aplicação do YOLO em vídeo de tempo real, a princípio utilizando a câmera do próprio computador. Após os testes, foi a aplicado YOLO em vídeo no Kinect, porém a comunicação do código em Python com a câmera não foi possível.

5. RESULTADOS E DISCUSSÃO

Já com todos os *drivers* e programas instalados o primeiro teste feito, para verificação do funcionamento do sensor, deve-se tomar cuidado com o encaixe dos fios, se levemente mal conectados eles não funcionam.

Na próxima etapa de testes, obteve-se a imagem colorida e a imagem de profundidade, todas estáticas. A Figura 26, mostra lado a lado as imagens obtidas pelo sensor.

Figura 26 – Imagens obtidas no primeiro teste com o sensor



(Fonte: Autoria Própria)

Podemos observar que imagem colorida possui um ângulo de abertura maior que a imagem de profundidade. Na imagem de profundidade, quanto mais longe o objeto em cena, mais clara é sua representação.

Em testes posteriores foi feita uma nova amostragem de imagens, para posterior aplicação do YOLO. Nessas imagens existem mais elementos que podem ser classificados, podendo verificar melhor seu funcionamento, conforme visto na Figura 27.

Figura 27 –Amostras de imagens para aplicação do YOLO

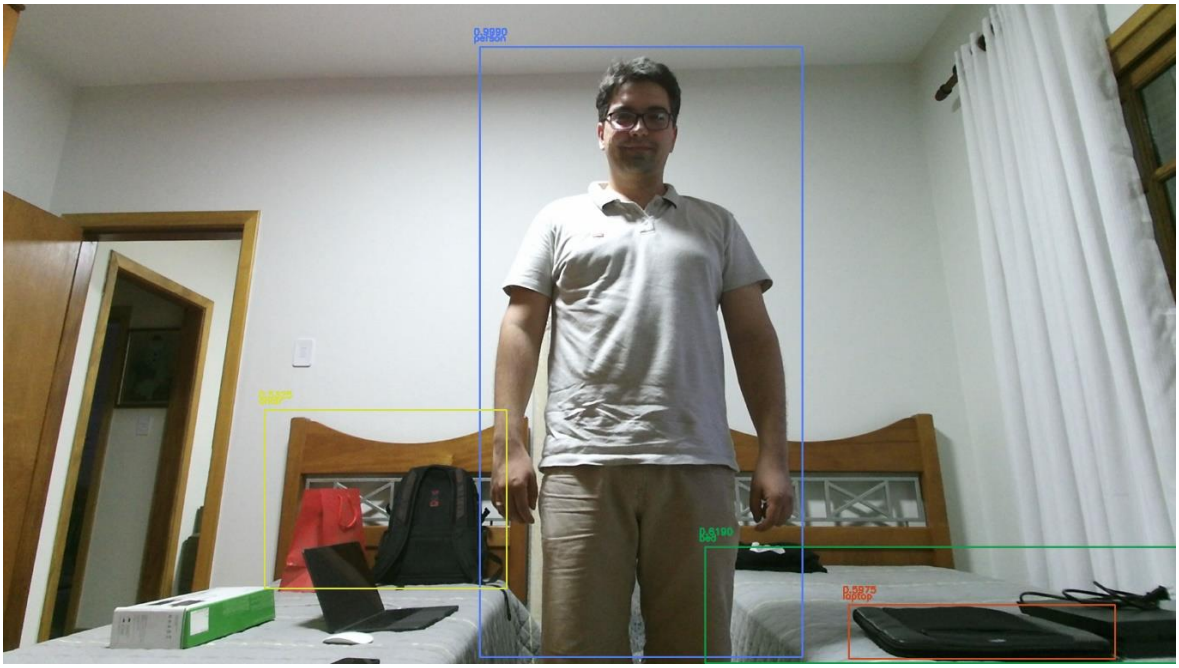
imagem_colorida.png

imagem_profundidade.png

(Fonte: Autoria Própria)

Após a obtenção das imagens de base, pode-se aplicar o algoritmo de YOLO. Resultando na Figura 28.

Figura 28 –Aplicando o YOLO



(Fonte: Autoria Própria)

Podemos ver que o algoritmo conseguiu identificar diversos elementos em cena. No caso do retângulo em azul, onde foi identificado uma pessoa, possui a chance de 99,9% de se tratar de uma pessoa. No retângulo em verde, foi identificado corretamente como uma cama, mas a probabilidades de ser realmente uma cama são de 61,9%.

No caso dos retângulos laranjas e amarelos, o algoritmo identificou erroneamente os elementos. Em laranja classificou como um laptop, com uma probabilidade de 59,7%. Na caixa limitadora amarela, foi identificado uma cadeira com chances de 53,2%. Mostrando que o YOLO não é a prova de erros. Mas ainda assim possuiu um grau de confiabilidade maior que muitos sistemas atuais.

O YOLO mostra dificuldades em identificação de objetos menores, e a posição que o objeto se encontra também influencia a capacidade de detecção. Na imagem havia um laptop presente na cena, porém sua posição, ou o ângulo de abertura da tampa não permitiu que fosse identificado.

Para os testes de detecção em profundidade, o YOLO não conseguiu identificar os objetos em cena, como mostrado na Figura 29.

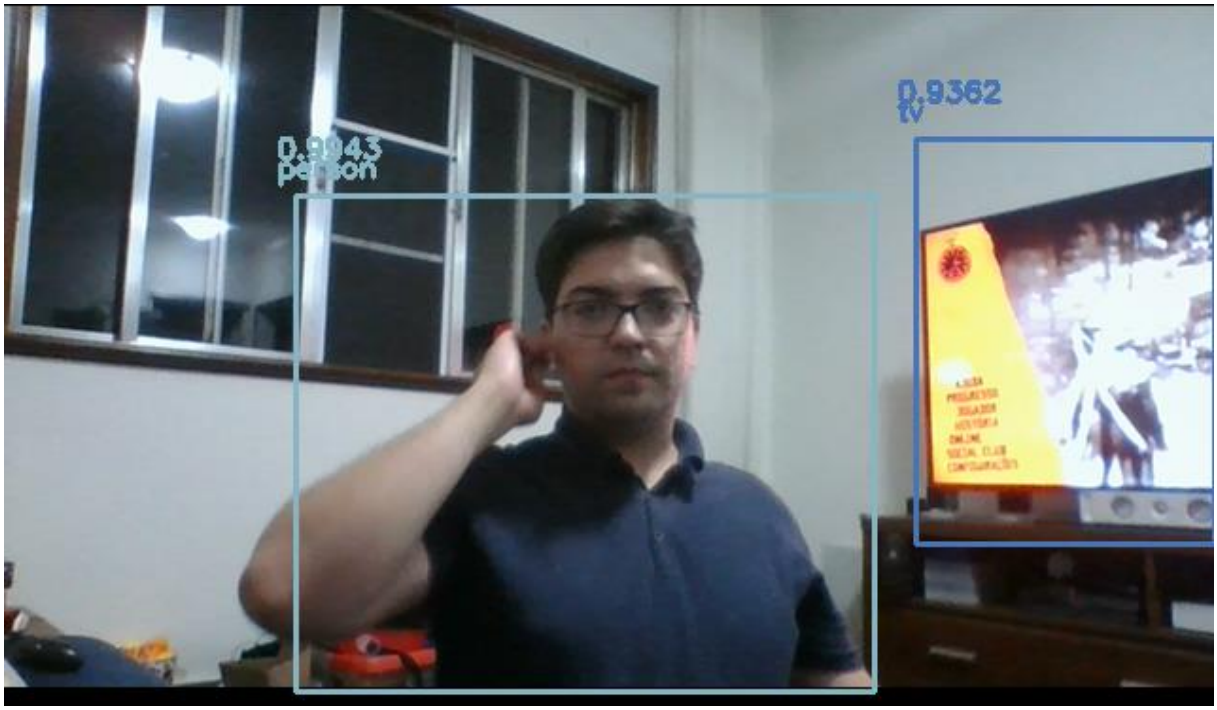
Figura 29 –Aplicando o YOLO em uma imagem de profundidade

(Fonte: Autoria Própria)

Na realização de testes de em vídeo, o YOLO só pode ser aplicado em uma câmera tradicional, como as dos próprios notebooks, a comunicação do sensor Kinect com o código em Python, não foi possível impossibilitando seu uso em vídeo.

Na Figura 30 é mostrado as imagens capturadas pela câmera do próprio computador utilizado. Pode-se observar que a qualidade das imagens é inferior se comparadas com as imagens obtidas pelo sensor Kinect, mas o suficiente para a localização e identificação dos objetos. Com a utilização da *webcam* foi possível faze a aplicação do YOLO com imagens em tempo real.

Figura 30 –Imagem capturada por *webcam*



(Fonte: Autoria própria)

6. CONSIDERAÇÕES FINAIS

Em primeiro teste utilizando a câmera do próprio computador, em uma imagem estática, utilizando a rede YOLO foi possível a identificação e classificação dos objetos em cena. Evoluindo para um teste em tempo real, a rede YOLO conseguiu obter as informações sobre os objetos do vídeo de modo instantâneo. Nessa etapa a linguagem escolhida para a implementação da rede foi Python.

Ao incluir o sensor RGB-D, foi utilizado ferramentas de OpenCV para fazer a utilização da rede YOLO, novamente a priori utilizando uma imagem estática, e posteriormente utilizando uma imagem em tempo real. Onde foi possível fazer as identificações de objetos em cena, além de sua classificação.

Durante os testes com a câmera RGB-D não foi possível fazer sua comunicação com o Python, utilizando o sistema operacional e os recursos disponíveis para

essa função. Do levantamento que foi realizado, foi possível perceber que, em geral, os desenvolvedores utilizando o SO Linux (distribuição Ubuntu) para fazer essa comunicação.

Na etapa de obtenção de dados de profundidade, foi possível a aquisição de uma imagem e seus dados de profundidade com a ferramenta de desenvolvimento padrão fornecida pelo desenvolvedor da câmera RGB-D. Ao tentar ligar a câmera com o sistema YOLO, não foi possível conectar as imagens, pois a câmera funcionava nos programas escritos com Processing, mas não se comunicava com os programas escritos em Python.

Foi possível perceber também que o desempenho do YOLO com o OpenCV no Windows, fica limitado a 10 fps, mesmo o computador possuindo uma placa de vídeo dedicada. Também foi notado que para realizar o aumento dessa taxa de aquisição de imagens, a troca do sistema operacional seria necessária.

Como sugestão para trabalhos futuros, recomenda-se a utilização do sistema operacional Linux, ao invés da utilização de Windows. Vale também a tentativa de utilização de uma linguagem de programação diferente de Python, além de comprovadamente o algoritmo funcionar em câmeras digitais e webcams.

7.PROPOSTAS FUTURAS

Como sugestão para trabalhos futuros, recomenda-se a utilização do sistema operacional diferente ao Windows, como o Linux. Vale também a tentativa de utilização de uma linguagem de programação diferente de Python, como o C#, por ser linguagem nativa ao sensor Kinect.

A utilização de uma câmera RGB-D diferente ao Kinect, assim como câmeras de vídeo e webcams poderiam ser utilizadas, por já se mostrarem eficientes na aplicação do YOLO.

A integração do sistema de reconhecimento junto a um veículo autônomo poderia ser feita também, verificando a possibilidade de uso do sistema YOLO em conjunto a uma plataforma autônoma em tempo real.

REFERÊNCIAS

ACEITUNO, Javier Hernández, et al**. Using Kinect on an autonomous vehicle for outdoors obstacle detection**, 2016.

ARAÚJO, Flávio H. D.; CARNEIRO, Allan C. Carneiro; SILVA, Romuere R. V.; MEDEIROS, Fátima N. S.; USHIZIMA, Daniela M. **Redes Neurais Convolucionais com Tensorflow: Teoria e Prática.** 1º ed. Piauí. Escola Regional de Informática do Piauí, 2017.

BACKES, André Ricardo; JUNIOR, Jarbas Joaci de Mesquita Sá. **Introdução à Visão Computacional Usando MATLAB®**. 1º.ed. Rio de Janeiro: Alta Books Editora, 2016.

BARELLI, Felipe. **Introdução à Visão Computacional. Uma abordagem prática com Python e OpenCV**. 1º.ed. São Paulo: Casa do Código, 2018.

BIMBRAW, Keshav. **Autonomous Cars: Present and Future. A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology**, 2015.

BUTKIEWICZ, Thomas. **Low-cost Coastal Mapping using Kinect v2 Time-of-Flight Cameras**, 2014.

CARDOSO, Gabriel Schade. Microsoft Kinect: **Criando aplicações interativas com o Microsoft Kinect.** 1º.ed. São Paulo: Casa do Código, 2017.

KATHURIA, Ayoosh. What's new in YOLO v3? **Towards Data Science**, 2018. Disponível em: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. Acesso em: 29 de out.de 2018.

LITMAN, Todd**. Autonomous Vehicle Implementation Predictions Implications for Transport Planning**, 2018.

LITOMISKY, Krystof. **Consumer RGB-D Cameras and their Applications**, 2012.

LIU, Yi, et al. **The Design of a Fully Autonomous Robot System for Urban Search and Rescue**, 2016.

LOPES, Uilian Kenedi. **Redes Neurais Convolucionais Aplicadas ao Diagnóstico de Tuberculose por Meio de Imagens Radiológicas.** 2017.

PONNUSAMY, Arun. YOLO Object Detection with OpenCV and Python. **Arun Ponnusamy,** 2018. Disponível em: https://www.arunponnusamy.com/yolo-object-detection-opencv-python.html. Acesso em: 30 de out de 2018.

REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. **You Only Look Once: Unified, Real-Time Object Detection**, 2016.

SEVERINO, Antônio Joaquim. **Metodologia do Trabalho Científico**. 23º.ed. São Paulo: Cortez Editora, 2007.

UROOJ, Sana; FEROZ, Irum; AHMAD, Dr. Nadeem. **Systematic Literature Review on User Interfaces of Autonomous Cars: Liabilities and Responsibilities**, 2018.

VARGAS, Ana Carolina Gomes; PAES, Aline; VASCONCELOS, Cristina Nader. **Um Estudo sobre Redes Neurais Convolucionais e sua Aplicação em Detecção de Pedestres**, 2016.

WASENMÜLLER, Oliver; STRICKER, Didier. **Comparison of Kinect v1 and v2 Depth Images in Terms of Accuracy and Precision**, 2016.

YAMANE, Katsu; SMART, William D.; FORLIZZI, Jodi. **How to Form Robot Law—a Case Study of Autonomous Cars**, 2017.

ZENNARO, S. et al. **Performance Evaluation of the 1st and 2nd Generation Kinect for Multimedia Applications**, 2015.

APENDICE A – Código fonte em Processing para receber os dados do Kinect V2.

```
1.  //Programa para pegar os dados de profundida e imagem com cor e encerrar a execução
2.  import KinectPV2.*;
3.
4.  KinectPV2 kinect;
5.
6.  void setup() {
7.    size(1920, 1080);
8.
9.    kinect = new KinectPV2(this);
10.   kinect.enableColorImg(true);
11.   kinect.enableDepthImg(true);
12.
13.   kinect.init();
14.   delay(3000);
15.
16.   //Roda a l[ogica apenas uma vez
17.   background(0);
18.
19.   //obtain the color image from the kinect v2
20.   PImage imagem;
21.   imagem = kinect.getColorImage();
22.   imagem.save("cor.png");
23.
24.   imagem = kinect.getDepthImage();
25.   imagem.save("fundo.png");
26.
27.   fill(255, 0, 0);
28.   text(frameRate, 50, 50);
29. }
30.
31. void draw() {
32.   exit();
33. }
```

APENDICE B – Código fonte em Python para marcar os elementos presents das imagens utilizando a rede YOLO V3.

```
1.  # import required packages
2.  import cv2
3.  import time
4.  import numpy as np
5.
6.
7.
8.  # read class names from text file
9.  classes = None
10. with open('yolov3_base.txt', 'r') as f:
11.     classes = [line.strip() for line in f.readlines()]
12.
13. # generate different colors for different classes
14. COLORS = np.random.uniform(0, 255, size=(len(classes), 3))
15.
16. # read pre-trained model and config file
17. net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')
18.
19. #Funcoes para manipular os dados
20. # function to get the output layer names
21. # in the architecture
22. def get_output_layers(net):
23.
24.     layer_names = net.getLayerNames()
25.
26.     output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
27.
28.     return output_layers
29.
30. # function to draw bounding box on the detected object with class name
31. def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
32.     label = str(classes[class_id])
33.
34.     color = COLORS[class_id]
35.
36.     cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)
37.
38.     cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
39.     cv2.putText(img, "%.4f" % (confidence), (x-10,y-
    20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
40.
41. def classificar_e_marcar(nome_image1, nome_image2):
42.     imagens = [nome_image1, nome_image2]
43.     for nome in imagens:
44.         # read input image
45.         image = cv2.imread(nome+".png")
46.         # image = cv2.imread("foto_1.jpg")
47.         Width = image.shape[1]
48.         Height = image.shape[0]
49.         scale = 0.00392
50.         # create input blob
51.         blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)
52.         #blob = cv2.resize(image, (416,416), cv2.INTER_LINEAR)
53.         # set input blob for the network
54.         net.setInput(blob)
55.
56.         # run inference through the network
57.         # and gather predictions from output layers
```

```
58.         outs = net.forward(get_output_layers(net))
59.
60.         # initialization
61.         class_ids = []
62.         confidences = []
63.         boxes = []
64.         conf_threshold = 0.5
65.         nms_threshold = 0.4
66.
67.         # for each detetion from each output layer
68.         # get the confidence, class id, bounding box params
69.         # and ignore weak detections (confidence < 0.5)
70.         for out in outs:
71.             for detection in out:
72.                 scores = detection[5:]
73.                 class_id = np.argmax(scores)
74.                 confidence = scores[class_id]
75.                 if confidence > 0.5:
76.                     center_x = int(detection[0] * Width)
77.                     center_y = int(detection[1] * Height)
78.                     w = int(detection[2] * Width)
79.                     h = int(detection[3] * Height)
80.                     x = center_x - w / 2
81.                     y = center_y - h / 2
82.                     class_ids.append(class_id)
83.                     confidences.append(float(confidence))
84.                     boxes.append([x, y, w, h])
85.
86.         # apply non-max suppression
87.         indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
88.
89.         # go through the detections remaining
90.         # after nms and draw bounding box
91.         for i in indices:
92.             i = i[0]
93.             box = boxes[i]
94.             x = box[0]
95.             y = box[1]
96.             w = box[2]
97.             h = box[3]
98.             # print("Achei: ",classes[class_ids[i]], " Chance: %.4f" % confidences[i])
99.             draw_bounding_box(image, class_ids[i], confidences[i], round(x), round(y), round
    (x+w), round(y+h))
100.
101.                 # display output image
102.                 # img = cv2.resize(image, (800,800))
103.                 # cv2.imshow("object detection", img)
104.                 cv2.imwrite(nome+"_marcada.png", image)
105.
106.         classificar_e_marcar("imagem_colorida", "imagem_profundidade")
```

APENDICE C – Lista de classes identificaveis pelo YOLO.

1	person	21	bottle	41	elephant	61	bed
2	bicycle	22	wine glass	42	bear	62	dining table
3	car	23	cup	43	zebra	63	toilet
4	motorcycle	24	fork	44	giraffe	64	tv
5	airplane	25	knife	45	backpack	65	laptop
6	bus	26	spoon	46	umbrella	66	mouse
7	train	27	bowl	47	handbag	67	remote
8	truck	28	banana	48	tie	68	keyboard
9	boat	29	apple	49	suitcase	69	cell phone
10	traffic light	30	sandwich	50	frisbee	70	microwave
11	fire hydrant	31	orange	51	skis	71	oven
12	stop sign	32	broccoli	52	snowboard	72	toaster
13	parking meter	33	carrot	53	sports ball	73	sink
14	bench	34	hot dog	54	kite	74	refrigerator
15	bird	35	pizza	55	baseball bat	75	book
16	cat	36	donut	56	baseball glove	76	clock
17	dog	37	cake	57	skateboard	77	vase
18	horse	38	chair	58	surfboard	78	scissors
19	sheep	39	couch	59	tennis racket	79	teddy bear
20	cow	40	potted plant	60	toothbrush	80	hair drier