UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

HENRIQUE MALLMANN GRÄBIN

ODOMETRIA VISUAL MONOCULAR:
Estudo de caso para veículos terrestres

Porto Alegre

2019

HENRIQUE MALLMANN GRÄBIN

ODOMETRIA VISUAL MONOCULAR:
Estudo de caso para veículos terrestres

Pojeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como requisito parcial para Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Ronaldo Husemann

Porto Alegre

2019

HENRIQUE MALLMANN GRÄBIN

ODOMETRIA VISUAL MONOCULAR:
Estudo de caso para veículos terrestres

Este projeto foi julgado adequado para fazer jus aos créditos da atividade de Projeto de Diplomação do Curso de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Aprovado em ___/___/_____

BANCA EXAMINADORA

Prof. Dr. Altamiro Amadeu Susin
UFRGS

Prof. Dr. Tiago Roberto Balen
UFRGS

Prof. Dr. Ronaldo Husemann
Orientador - UFRGS

Agradecimentos

Agradeço à minha família, meus pais Cláudia e Carlos e meu irmão Vicente que sempre me apoiaram incondicionalmente e forneceram todos os meios para realização dos meus objetivos. Sem vocês eu não seria quem sou hoje.

Ao professor Ronaldo Husemann por ter dedicado tempo considerável lendo o texto, sugerindo modificações e indicando diferentes possibilidades de caminhos a serem explorados.

À Universidade Federal do Rio Grande do Sul e ao Governo Federal, que me proporcionaram estudo superior de qualidade e oportunidade de bolsa de estudo no exterior.

À École Centrale de Lille, que me recebeu calorosamente e foi fundamental na minha formação acadêmica e desenvolvimento pessoal.

À todas as pessoas que encontrei nos últimos sete anos e fizeram parte da minha jornada acadêmica até hoje.

"If you're not failing,
you're not pushing your limits,
and if you're not pushing your limits,
you're not maximizing your potential"
(Ray Dalio, Principles: Life and Work)

Resumo

O presente trabalho tem por objetivo estudar a implementação de um algoritmo de odometria visual monocular, que consiste em estimar a trajetória percorrida por uma câmera utilizando imagens sequenciais. O desenvolvimento desta área de pesquisa adquiriu relevância a partir de 2004 quando a NASA integrou esta tecnologia em seus *rovers* enviados a Marte. Hoje, tendo em vista a aplicação destes métodos para veículos autônomos, se tornam ainda mais relevantes. Neste trabalho são realizados três experimentos, cujos objetivos são: Investigar o funcionamento e escolha do algoritmo de *Lucas-Kanade* para seguimento de pontos (parte do algoritmo de odometria visual); Verificar o funcionamento do algoritmo de odometria visual completo em uma base de dados teste padrão (utilizou-se a base de dados *KITTI*); Verificar o funcionamento do algoritmo de odometria visual em uma trajetória própria. Para cada experimento, um estudo dos parâmetros ótimos para cada algoritmo, que dependem fundamentalmente da base de dados, foi realizado. Para o primeiro experimento, foi verificado que a utilização do algoritmo de LK através da biblioteca *OpenCV* é mais eficiente que uma implementação própria, pois para imagens com 100 pontos a serem seguidos atingiu-se uma taxa de acerto média de 86,6% e tempo médio de execução de 5,9 ms. O segundo experimento, com os dados da base de dados *KITTI*, para percursos de distância 3,72 km, 1,70 km, 5,06 km e 2,206 km (índice *KITTI*: 00, 09, 02 e 05) teve como resultado erros médios de 7,47 m, 7,47 m, 10,03 m e 6,33 m respectivamente. Para o percurso com dados próprios adquiridos por meio de de uma câmera *GoPro*, com 1,846 km de distância, o erro médio obtido foi de 28,57 m. Adicionalmente, estudou-se a possibilidade de execução do algoritmo de odometria visual em um micro-computador *Raspberry Pi 3 B+*. Porém, foi constatado que este consegue executar apenas 1,17 imagens por segundo com o código realizado, não sendo viável a estimação do percurso em tempo real.

Palavras-chave: Odometria Visual. Monocular. Lucas-Kanade. Estimação de Movimento.

Abstract

The present paper aims to study the implementation of a monocular visual odometry algorithm, which consists of estimating the motion of a camera using sequential images. The development of this research area gained relevance in 2004, when NASA integrated this technology into its Mars Rovers. Nowadays, considering the application of these methods into autonomous vehicles, they have become still more relevant. In this work three experiments are executed. Their objectives are to: Investigate the operation and choice of the *Lucas-Kanade* algorithm for feature tracking (part of the visual odometry pipeline); Verify the operation of the full visual odometry pipeline in a standardized dataset (the *KITTI* dataset was chosen); Verify the operation of the full visual odometry pipeline in a custom trajectory, acquired with a sports camera. For each experiment, a study of the optimal parameters for each algorithm, which depend fundamentally on the dataset, is done. For the first experiment, it was verified that the usage of the LK algorithm from the *OpenCV* library is more efficient than an own implementation, given that for images with 100 points to be tracked the average accuracy was 86,6% and average execution time 5,9 ms. The second experiment, with the *KITTI* dataset, for courses with distances of 3,72 km, 1,70 km, 5,06 km and 2,21 km (*KITTI* index: 00, 09, 02 e 05), resulted in average distance estimation error 7,47 m, 7,47 m, 10,03 m and 6,33 m respectively. For the 1,846 km course with own data, acquired with a *GoPro*, the average error was 28,57 m. Aditionally, the possibility of execution of the visual odometry algorithm by a microcomputer *Raspberry Pi 3 B+* was verified. However, it was found that it could not execute more than 1,17 frames per second with the implemented code, not being sufficient for real time motion estimation.

Keywords: Visual Odometry. Monocular. Lucas-Kanade. Motion Estimation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

OV	Odometria Visual
LK	Lucas-Kanade
OCV	OpenCV
ST	Shi-Tomasi
RPi	Raspberry Pi
IMU	Inertial Measurement Unit
VLC	VLC Media Player
GDL	Graus de Liberdade
BBB	BeagleBone Black

Sumário

1 Introdução

Na última década, grandes empresas, como Tesla, Bosch, Apple, Audi, Baidu, Intel e Lyft vêm trazendo avanço tecnológico em tecnologias de navegação para carros autônomos, que conseguem trafegar com grande confiabilidade em diversas estradas. Um dos sistemas necessários para navegação autônoma consiste em conhecer a posição atual do veículo de forma confiável. Uma forma de fazê-lo consiste em utilizar dispositivos GPS. Em várias situações práticas o sinal GPS pode não estar disponível, por exemplo em túneis, partes densas de uma grande cidade, lugares muito afastados ou regiões nas quais o dado é de baixa exatidão. Nestas situações, o veículo precisa conhecer a sua posição corrente e mapear o caminho percorrido de outras maneiras. Um problema semelhante pode ser encontrado em aplicações com drones, cujos sistemas de localização se baseiam majoritariamente em localização GPS e têm dificuldades para navegar ou pousar em situações nas quais esta informação está indisponível (YANG et al., 2018).

Neste contexto, o presente trabalho visa investigar métodos de estimação de trajetória através de câmeras, que permitem diminuir a dependência de veículos autônomos do sinal GPS. Considerando que a informação de posição e direção do veículo é um fator de grande importância para navegação autônoma, o desenvolvimento destes métodos se torna fundamental. O nome dado a eles é odometria visual. O termo odometria tem origem no grego, *hodos* significando "jornada" e *metron*, "medida". O termo criado, se refere então à variação da translação e orientação e foi cunhado por Nistér (2004). A área de pesquisa foi explorada majoritariamente pela NASA entre os anos 1980 e 2000 (AQEL et al., 2016) ao integrar um dispositivo de odometria visual no veículo Opportunity enviado a Marte (CHENG; MAIMONE, 2005). De maneira geral, odometria visual consiste essencialmente em estimar a posição relativa de um agente em relação ao ponto de partida utilizando informação proveninte de uma câmera carregada pelo mesmo. A principal vantagem deste método em relação à odometria clássica de medição de rotação das rodas é que o método baseado em imagens não é sensível a derrapagens.

Três objetivos distintos foram definidos para realização neste trabalho. O primeiro consiste em investigar o funcionamento do algoritmo de *Lucas-Kanade*, que é uma parte do algoritmo completo de odometria visual, e comparar o desempenho de uma implementação própria com o desempenho de uma implementação de biblioteca especializada em processamento de imagens. O segundo consiste em escolher e validar uma metodologia de odometria visual testando o algoritmo completo em uma base de dados frequentemente utilizada para *benchmark* em pesquisas nesta área. O terceiro consiste em adquirir uma base dados própria e verificar se a abordagem escolhida continua funcional. Por fim, verifica-se se é possível executar a aplicação em tempo real em micro-computador de baixo custo.

Inicialmente realiza-se um estudo teórico a respeito do funcionamento dos métodos de odometria visual e dos diferentes algoritmos que podem ser utilizados. Em seguida, é definida uma sequência de algoritmos a serem utilizados para solução do problema e implementação em linguagem de programação *Python* (Python Software Foundation, 2016). Um dos algoritmos utilizados é o *Lucas-Kanade* para seguimento de pontos presentes em duas imagens consecutivas. O primeiro experimento realizado foi a implementação deste algoritmo, que é então comparada com a implementação da biblioteca *OpenCV* (BRADSKI, 2000) através de análise de acerto e de tempo de execução. Através desta, foi encontrado que a taxa de acerto média com a função do *OpenCV* é superior à da implementação própria.

A segunda série de experimentos realizados consistiu na implementação completa da sequência de blocos algorítmicos para odometria visual, em uma base de dados de alta qualidade denominada *KITTI* desenvolvida pelo *Karlsruhe Institute of Technology* (URTASUN; LENZ; RAQUEL, 2012). Neste momento, diferentes possibilidades de parâmetros para os algoritmos foram avaliados, como, por exemplo, número de pontos característicos seguidos por par de imagens, tamanho dos padrões do *Lucas-Kanade* e do detector de cantos. Estes algoritmos foram então testados em 11 percursos. Por exemplo, para percursos de 3,72 km, 1,7 km, 5,06 km e 2,21 km foram encontrados erros médios de respectivamente 7,47 m, 7,47 m, 10,03 m e 6,33 m.

Na terceira série de experimentos foi testada a metodologia estudada em um caso de aplicação real. Para isso, dados foram obtidos com uma câmera esportiva *GoPro* acoplada em um carro que trafegava em um bairro residencial da cidade de Porto Alegre. Para que a solução proposta funcionasse de maneira satisfatória, foi necessário utilizar também dados de velocidade provenientes da unidade de medidas inerciais da câmera. O percurso realizado foi de 1,846 km e o erro médio obtido foi de 28,57 m.

Por fim, desejou-se examinar experimentalmente se o hardware escolhido para execução do algoritmo possui capacidade computacional suficiente para execução em tempo real do programa desenvolvido. Para isso, o número de imagens por segundo é calculado para diferentes combinações de parâmetros. Com o hardware escolhido, o número máximo de imagens por segundo atingidas foi de 1,17, não sendo suficiente para execução em tempo real. Assim, foram sugeridas alterações a serem feitas em trabalhos futuros para melhora da velocidade de cálculo.

2 Fundamentação Teórica

2.1 Fundamentos de Odometria Visual

O problema a ser estudado consiste em encontrar o caminho percorrido por um veículo que registra o percurso através de uma câmera no modo vídeo. Assim, a partir deste vídeo, imagens podem ser obtidas de forma sequencial e com frequência constante. Para cada par de imagens analisadas (I_k e I_{k-1}) de uma sequência de imagens, o objetivo da odometria visual é encontrar uma transformação de corpo rígido que faz a correspondência de píxeis da imagem I_{k-1} nos correspondentes da imagem I_k. O movimento de rotação da câmera pode ser descrito por uma matriz de rotação $R_{k,k-1}$ de tamanho 3x3 e por um vetor de translação $t_{k,k-1}$ de tamanho 3x1. A matriz de rotação pertence ao grupo de rotação 3D composto por todas as rotações sobre a origem em um espaço euclidiano tridimensional. Conforme a notação estabelecida por Scaramuzza e Fraundorfer (2011a), a transformação pode ser definida através de uma matriz pertencente a um espaço de dimensões 4x4, conforme as equações 2.1 e 2.2, composta pelas matrizes R e t. Por consequência, $\mathbf{0}$ é uma matriz de valor nulos 1x3 e $\mathbf{1}$ é 1x1. A adição desta linha inferior permite que as transformações possam ser compostas simplesmente através de multiplicação de matrizes, conforme indicado na equação 2.2. Os índices k e $k-1$ indicam as imagens para as quais a transformação se refere.

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

Deseja-se determinar o caminho completo percorrido pelo veículo. Assim, basta calcular a matriz de transformação, denominada T, para cada par consecutivo de frames. A trajetória completa realizada pode ser estimada acumulando todas as transformações até o momento k. A posição do observador em relação à origem, para cada instante de tempo k, é dada pelo conjunto $C_{0:k} = C_0, \ldots, C_k$. Nesta notação, a posição da imagem k é dada pela posição anterior simplesmente multiplicada pela transformação entre o último par de frames, ou seja, $C_k = C_{k-1}T_{k,k-1}$. O cálculo de $R_{k,0}$ (matriz de rotação atual em relação à referência) e $t_{k,0}$ (posição atual em relação à referência) pode ser atualizado imagem por imagem simplesmente reescrevendo os termos da multiplicação de matrizes conforme a equação 2.2.

$$C_{k,0} = \begin{bmatrix} R_{k,0} & t_{k,0} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{k-1,0} & t_{k-1,0} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

Calculando o produto matricial e igualando à definição de $C_{k,0}$:

$$R_{k,0} = R_{k-1,0} \cdot R_{k,k-1} \tag{2.3}$$

$$t_{k,0} = R_{k-1,0} \cdot t_{k,k-1} + t_{k-1,0} \tag{2.4}$$

Dentro do ramo de pesquisa de odometria visual, o primeiro divisor entre as diferentes abordagens disponíveis é a opção de captura das imagens com duas câmeras (métodos chamados binoculares ou *stereo*) ou com uma câmera (monoculares). Segundo Scaramuzza e Fraundorfer (2011a), devido às diferenças de metodologia associada a cada abordagem, elas vêm evoluindo praticamente como linhas de pesquisa independentes. A principal diferença entre os dois métodos é que no caso monocular a estimação do movimento (que é fundamentalmente tridimensional) deve ser feita apenas com dados bidimensionais (imagens). Uma vantagem dos métodos binoculares é a facilidade de triangulação para pontos relativamente próximos. Por esta razão, historicamente, a maioria da pesquisa desenvolvida nesta área foi realizada no modo binocular. Porém, a odometria visual monocular se torna essencial nos casos em que os objetos se encontram a uma distância muito maior das câmeras em relação à distância entre elas. Neste caso, o problema se reduz essencialmente ao caso monocular. Além disso, quando se deseja embarcar uma solução em microcontrolador de baixo custo, a complexidade do algoritmo em modo monocular pode se tornar uma vantagem pois não há necessidade de recalibração repetitiva e evita-se a etapa de associação das duas imagens do modo binocular.

Dentro da área monocular, pode-se separar os métodos de odometria visual em três grandes grupos conforme Szeliski (2011): 1) métodos que utilizam pontos característicos (*features*) em uma imagem; 2) métodos que utilizam informação de variação de intensidade luminosa (chamados de *appearance-based*); 3) métodos híbridos.

Os métodos que utilizam pontos característicos foram desenvolvidos inicialmente por Nistér (2004) para estimação da estrutura da posição. Estes métodos baseiam-se fundamentalmente em descritores de regiões características que são detectadas independentemente e comparadas para cada par de imagens. Assim, a estimação de movimento pode ser realizada encontrando uma função que descreve o movimento dos pontos conforme a sequência de imagens avança. Alguns métodos que utilizam ideias baseadas neste princípio são: SIFT, SURF, ORD e BRISK (PODDAR; KOTTATH; KARAR, 2018), que consistem fundamentalmente em definir descritores de regiões.

As técnicas *appearance-based* consistem fundamentalmente em verificar a variação luminosa na imagem como um todo ou em sub-regiões. Por exemplo, Yu, Pradalier e Zong (2011) implementa uma solução na qual para cada duas imagens consecutivas de uma

câmera, regiões da primeira imagem são detectadas e procuradas com vários ângulos de rotação diferentes para identificação da região na imagem posterior.

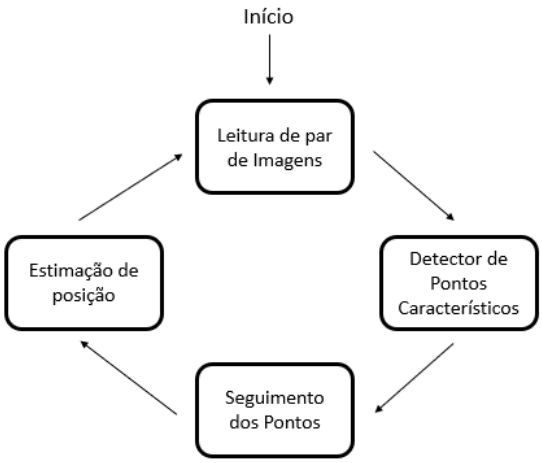
As abordagens híbridas consistem fundamentalmente em incluir características de ambas as outras. Isso pode ser realizado, por exemplo, utilizando detectores de pontos característicos para definição das regiões que devem ser seguidas, e em seguida um algoritmo que procure por estas regiões na imagem posterior.

Como pode-se perceber, ambas abordagens possíveis para o problema exigem que haja pontos ou regiões facilmente identificáveis ou variações luminosas na imagem. Então é natural que para que estes métodos sejam aplicados, há necessidade de variação de textura do ambiente.

2.2 Estrutura Algorítmica de um Sistema de Odometria Visual Monocular

A solução proposta por Nistér (2004) divide a sequência de algoritmos necessários para solução do problema em três etapas. Estas são: detecção de pontos característicos (*features*); seguimento (*tracking*) destes pontos; estimação de posição. Conforme a sequência de frames do trajeto é calculada, os algoritmos são executados de maneira sequencial conforme ilustrado na figura 1. Apesar de haver mais de uma forma de realizar cada uma das tarefas indicadas, o objetivo geral de cada bloco permanece o mesmo. Nas três seções seguintes, analisa-se quais algoritmos constituem cada uma destas etapas.

Figura 1 – Etapas de um Algoritmo de Odometria Visual



Fonte: Desenvolvido pelo autor

2.2.1 Detecção de Pontos Característicos

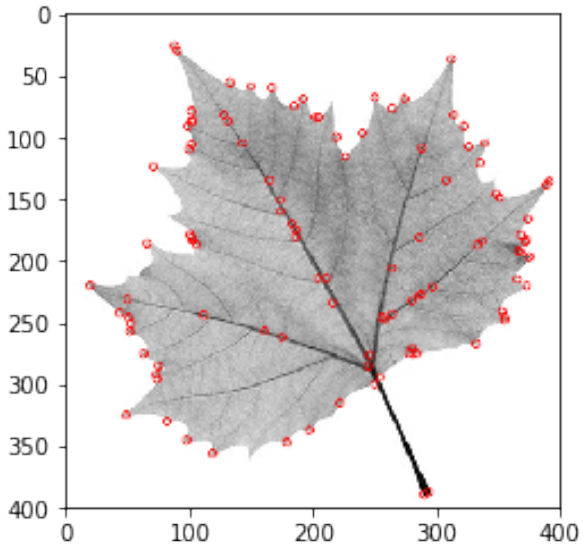
Uma etapa importante consiste em encontrar pontos específicos que possuem características relevantes ao seu redor, que possibilitem que sejam encontrados na imagem seguinte. Há várias maneiras de definir pontos que possuam características que permitam diferenciá-los dos demais, já que uma imagem possui informações de brilho, cor e textura. Na literatura, pode-se encontrar detectores como, por exemplo, de Harris e Stephens (1988), de Tomasi (1991), *FAST*, *SIFT* e *SURF* (SCARAMUZZA; FRAUNDORFER, 2011a). Os três primeiros desta lista são chamados de detectores de cantos pois procuram por intersecções de dois ou mais lados, enquanto os últimos dois são chamados pela literatura de "detectores de bolhas" (prefere-se utilizar o nome em inglês *blob-detectors*). Estes se distinguem dos detectores de cantos pois não procuram necessariamente por interesecções de lados, mas sim regiões que se distinguem de sua redondeza pela sua intensidade ou textura. Segundo Scaramuzza e Fraundorfer (2011a), detectores de cantos são mais rápidos porém menos distintivos quando comparados aos chamados *blob-detectors*.

Em geral, o procedimento de detecção de cantos consiste em duas etapas distintintas. Na primeira, aplica-se na imagem uma transformação matemática com objetivo de acentuar as variações de luminosidade da imagem com as características de interesse conforme o detector escolhido. Na segunda etapa, procura-se por regiões com valores de máximo ou mínimo da transformação realizada.

Não há uma maneira universal de se determinar qual é o melhor detector pois a própria definição de ponto característico é ambígua e dependente da aplicação escolhida. Dessa forma, o detector de *Shi-Tomasi* (TOMASI, 1991) é particularmente interessante pois foi desenvolvido por construção para se ter performance ótima conjuntamente com o algoritmo de *Lucas-Kanade* (LUCAS; KANADE, 1981), de forma que um ponto é definido como "bom" caso ele seja facilmente seguido por este algoritmo.

O algoritmo de Tomasi (1991) pode ser facilmente integrado na aplicação através de sua implementação na biblioteca *OpenCV*. Na figura 2, pode-se visualizar, através das marcações circulares na imagem, o tipo de ponto que é tipicamente encontrado utilizando o algoritmo de *Shi-Tomasi*.

Figura 2 – Aplicação do Algoritmo de Shi-Tomasi



Fonte: Desenvolvido pelo autor

2.2.2 Algoritmo de *Lucas-Kanade*

Outra etapa fundamental no processo de encontrar o movimento relativo entre o observador e a cena consiste no mapeamento de píxeis entre uma imagem $n-1$ e outra imagem n (*tracking* dos pontos). Conforme Scaramuzza e Fraundorfer (2011b), há duas abordagens frequentemente utilizadas para realizar tal tarefa. A primeira consiste em encontrar regiões características na primeira imagem (que contém diferenças de luminosidade grandes ou que podem ser fácilmente distiguíveis de outras regiões) e procurar estas regiões na imagem seguinte. Esta abordagem chama-se de *feature tracking* (seguimento de pontos). Este nome vem do fato de seguirmos pontos da primeira imagem para a segunda. A segunda abordagem consiste em primeiramente encontrar regiões características em ambas imagens e em seguida encontrar transformações que mapeiem os pontos corretamente. Esta última abordagem chamamos de *feature matching* (casamento de pontos), pois fazemos a correspondência entre pontos relevantes encontrados em ambas as imagens.

Nota-se que busca-se sempre mapear regiões quadradas ao redor de um píxel central. Um píxel sozinho possui apenas informações de luminosidade, encontrar a sua posição na imagem subsequente se mostraria uma tarefa árdua já que, conforme o observador se movimenta, podem haver grandes diferenças de luminosidade da imagem como um todo, além de oclusão e reflexão.

Conforme Scaramuzza e Fraundorfer (2011b) e Poddar, Kottath e Karar (2018), historicamente, os primeiros estudos feitos para realizar esta tarefa utilizaram casamento de pontos. Estes consistem fundamentalmente em utilizar uma medida de similaridade, como, por exemplo, soma de mínimos quadrados e correlação normalizada entre duas

regiões que se deseja associar. Porém, nas últimas décadas se obteve melhores resultados utilizando abordagens de seguimento de pontos piramidais. Segundo Pereira (2018), estes algoritmos tendem a apresentar melhor performance e menos valores atípicos quando comparados com casamento de pontos em situações de deslocamento de poucos píxeis (imagens consecutivas semelhantes). Porém, em situações, como curvas acentuadas ou quando o observador se desloca em velocidade elevada, pode haver movimento relativo grande de um ponto em relação à sua posição na imagem e este pode ser um ponto fraco desta abordagem.

Segundo Scaramuzza e Fraundorfer (2011b), quando há grandes deslocamentos nos píxeis das imagens o algoritmo com melhor performance é o *Kanade-Lucas-Tomasi*. Este, foi desenvolvido por Carlo Tomasi, Takeo Kanade e Bruce Lucas nos artigos de 1981 (LUCAS; KANADE, 1981) e 1991 (TOMASI, 1991). Neles, são descritas as ideias centrais por trás do algoritmo e do detector de cantos ótimo para o funcionamento do mesmo, que consiste essencialmente em utilizar o seguidos de *Lucas-Kanade* juntamente com o detector de cantos de *Shi-Tomasi*.

A melhor referência no que se refere a detalhes de implementação do algoritmo de *Kanade-Lucas-Tomasi* são as notas de Bouguet (2001). Devido à clareza da notação, a seguir utiliza-se leve alteração daquela utilizada por ele.

O problema consiste fundamentalmente em encontrar um vetor $\mathbf{d}$ para cada região escolhida (ponto característico) que mapeia um píxel da primeira imagem em um píxel da imagem seguinte. O objetivo é que o vetor $\mathbf{d}$ minimize uma função $\epsilon(\mathbf{d})$ que calcula a similaridade entre as regiões ao redor dos pontos escolhidos nas duas imagens, onde $I(x, y)$ é um ponto na primeira imagem e $J(x, y)$ na segunda. A função (equação 2.5) consiste na soma dos quadrados das diferenças de luminosidade entra as imagens para cada par de píxeis dentro de uma janela de integração de tamanho $2w_y + 1$ e $2w_x + 1$. Na equação 2.5, a primeira soma se refere a percorrer as posições da coordenada x, centralizada em u_x e entre os limites $u_x - w_x$ e $u_x + w_x$. A segunda soma se refere às posições da coordenada y, centralizada em u_y e entre os limites $u_y - w_y$ e $u_y + w_y$, de forma que todos os pontos dentro do padrão da janela definida sejam percorridos e sua diferença quadrática seja calculada.

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x, y) - J(x + d_x, y + d_y))^2 \tag{2.5}$$

Se tratando de um problema de minimização, espera-se que para $\mathbf{d}$ mínimo a sua derivada seja nula, ou seja:

$$\frac{\partial \epsilon}{\partial d} = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{2.6}$$

Derivando a equação da função de erro:

$$\frac{\partial \epsilon}{\partial \mathbf{d}} = -2 \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x,y) - J(x+d_x, y+d_y)) \cdot \begin{bmatrix} \frac{\partial J}{\partial x} & \frac{\partial J}{\partial y} \end{bmatrix} \tag{2.7}$$

Utilizando expansão de Taylor de primeira ordem ao redor do ponto $\begin{bmatrix} 0 & 0 \end{bmatrix}$ temos:

$$\frac{\partial \epsilon}{\partial \mathbf{d}} \approx -2 \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x,y) - J(x,y) - \begin{bmatrix} \frac{\partial J}{\partial x} & \frac{\partial J}{\partial y} \end{bmatrix} \mathbf{d}) \cdot \begin{bmatrix} \frac{\partial J}{\partial x} & \frac{\partial J}{\partial y} \end{bmatrix} \tag{2.8}$$

Nota-se que $\frac{\partial J}{\partial x}$ e $\frac{\partial J}{\partial y}$ são essencialmente as derivadas da imagem J nas direções x e y. Neste momento, prefere-se calcular as derivadas espaciais diretamente da primeira imagem. Elas podem ser obtidas facilmente através de filtros Sobel (BOUGUET, 2001). Vamos definir uma imagem I convoluída por um filtro Sobel em x e y como respectivamente I_x e I_y.

Fazendo as seguintes definições:

$$\nabla I \approx \begin{bmatrix} I_x \\ I_y \end{bmatrix} \tag{2.9}$$

$$\delta I(x,y) \doteq I(x,y) - J(x,y) \tag{2.10}$$

Assim, a equação 2.8 pode ser reescrita matricialmente por:

$$\frac{1}{2}\frac{\partial \epsilon}{\partial \mathbf{d}} = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (\nabla I^T \mathbf{d} - \delta I) \nabla I^T \tag{2.11}$$

$$\frac{1}{2}\frac{\partial \epsilon}{\partial \mathbf{d}} \approx \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} \left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \mathbf{d} - \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix} \right) \tag{2.12}$$

Pode-se definir duas matrizes G e b que representam as equações de soma simplificadamente:

$$G = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.13}$$

$$b = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix} \tag{2.14}$$

Finalmente:

$$\frac{1}{2}\left[\frac{\partial \epsilon}{\partial \mathbf{d}}\right]^T \approx G\mathbf{d} - b \tag{2.15}$$

$$\mathbf{d} = G^{-1}b \tag{2.16}$$

A equação encontrada é valida somente na região de validade da aproximação da expansão de Taylor, ou seja, para deslocamento pequenos de píxeis. Dessa forma, para que se convirja para a posição correta são necessárias várias iterações do algoritmo. O loop de iteração pode ser saído a partir do momento que a função de erro for suficientemente pequena ou o número máximo de iterações seja alcançado.

O equacionamento descrito se refere a apenas uma iteração do algoritmo. Na prática, é utilizado um esquema piramidal cujo objetivo é tornar o algoritmo robusto a diferença de escala, ou seja, quando pontos se afastam ou aproximam de forma a mudar o seu tamanho relativo. Fundamentalmente, este esquema consiste em primeiramente diminuir a resolução da imagem para que seja possível encontrar os pontos quando o deslocamento tenha sido muito grande ou saído da janela de integração. Os detalhes de programação referente à implementação piramidal podem ser encontradas em Bouguet (2001).

Um detalhe fundamental que não é frequentemente explorado é a necessidade de realizar todos os cálculos a nível de subpíxel. Na grande maioria das iterações do algoritmo de *Lucas-Kanade* a mudança de estimativa da posição do píxel é uma quantidade inferior ao deslocamento de um píxel. Ou seja, caso não haja resolução nesta ordem o algoritmo não terá efeito algum. Para resolver esta questão, sugere-se a utilização de um modelo de interpolação bilinear simples (BOUGUET, 2001) para tratar deste problema que é descrito pelas equações a seguir.

$$x = x_0 + \alpha_x \tag{2.17}$$

$$y = y_0 + \alpha_y \tag{2.18}$$

Nestas equações, os valores com índice 0 correpondem a valores inteiros e os com índice x ou y pela parte restante (< 1). Assim, cada subpíxel da imagem I(x,y) pode ser calculado através da seguinte equação:

$$\begin{aligned} I(x,y) =& (1-\alpha_x)(1-\alpha_y)I(x_0,y_0) + \alpha_x(1-\alpha_y)I(x_0+1,y_0) + \\ & (1-\alpha_x)\alpha_x I(x_0,y_0+1) + \alpha_x\alpha_y I(x_0+1,y_0+1) \end{aligned} \tag{2.19}$$

Assim, a imagem passa de um domínio discreto, onde apenas valores inteiros eram admitidos, para um domínio contínuo, onde qualquer valor dentro dos limites da imagem pode ser obtido.

2.2.3 Estimação de Movimento

A etapa restante a ser analisada, e também etapa principal de um algoritmo de odometria visual, consiste na estimação do movimento relativo entre duas imagens consecutivas. Esta etapa pode ser realizada através de diferentes abordagens conforme especificado por Scaramuzza e Fraundorfer (2011a):

– 2-D para 2-D: Os pontos correspondentes de ambas imagens são dados nas coordenadas da imagem.

– 3-D para 3-D: Ambos pontos são dados em coordenadas tridimensionais através de triangulação.

– 3-D para 2-D: Os pontos da imagem anterior (f_{k-1}) são dados em coordenadas tridimensionais (obtidas através de triangulação de duas imagens anteriores, f_{k-1} e f_{k-2}) enquanto que os pontos da imagem corrente (f_k) é dado em coordenadas bidimensionais.

De acordo com a Nistér (2004), as duas abordagens que apresentam melhores resultados são 2-D para 2-D e 3-D para 2-D. No caso 3-D para 2-D, considerando odometria monocular, uma etapa adicional é necessária que consiste na utilização de três imagens consecutivas e triangulação. Quando se trabalha com uma câmera binocular, a triangulação pode ser realizada diretamente. Conforme Scaramuzza e Fraundorfer (2011a), para casos monoculares, a abordagem 2-D para 2-D é preferíval pois evita a etapa de triangulação. Porém, essa opção apresenta problemas na estimação da escala absoluta, conforme pode ser visto na subseção seguinte.

2.2.3.1 Estimação de Movimento 2-D para 2-D

Deseja-se encontrar duas matrizes que descrevam o movimento entre um par de imagens. O movimento de translação pode ser descrito por um vetor $t_k = [t_x t_y, t_z]^T$ onde k indica o índice da imagem que está sendo considerada. O movimento de rotação é descrito por uma matriz 3 X 3 R_k. Tanto t_k quanto R_k podem ser estimados por uma matriz chamada de matriz essencial E. O conceito desta matriz foi introduzido em 1981 em um artigo de Longuet-Higgins (1981).

$$E_k \approx \hat{t}_k R_k \tag{2.20}$$

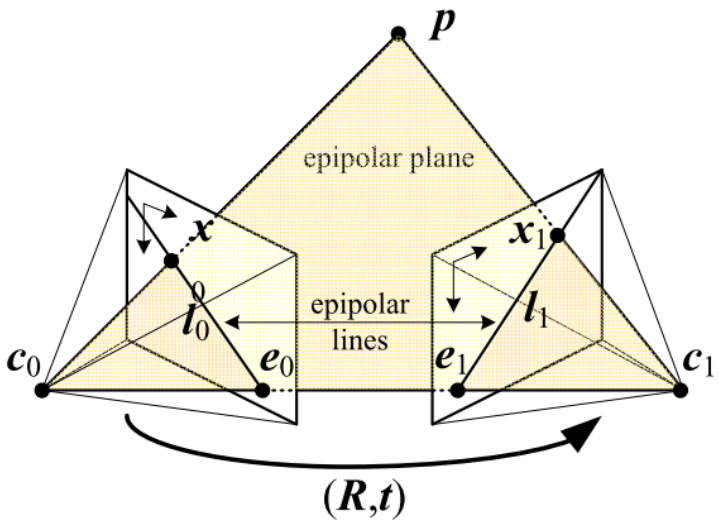
Onde:

$$\hat{t}_k = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{2.21}$$

A matriz essencial é importante pois através dela pode-se obter as matrizes que descrevem o movimento relativo. Ou seja, pode-se passar de um conjunto de pontos correspondentes nas imagens I_{k-1} e I_k para a matriz essencial E; em seguida pode-se passar desta para as matrizes R e t.

A metodologia para estimação da matriz essencial consiste em considerar as restrições sob uma ótica de geometria epipolar (SZELISKI, 2011). Na figura 3 tem-se uma ilustração para o caso de duas imagens. Considera-se duas imagens I_0 e I_1 referentes respectivamente aos pontos de observação c_0 e c_1 que possuem em seu campo de visão o ponto p. Estas imagens podem ser interpretadas como duas câmeras diferentes observando o mesmo objeto; ou uma câmera em movimento que observa a mesma cena em dois momentos distintos do tempo.

Figura 3 – Geometria Epipolar



Fonte: (SZELISKI, 2011)

Estas restrições são descritas pela equação 2.22, onde $\tilde{p}'$ representa os pontos característicos na imagem k e $\tilde{p}$ os da imagem $k-1$.

$$\tilde{p}' E \tilde{p} = 0 \tag{2.22}$$

O sinal til indica que as coordenadas são homogêneas (ou seja, possuem três valores, e.g. $\tilde{p} = [\tilde{u}, \tilde{v}, 1]$). Considerando estas restrições e a natureza homogênea das matrizes,

pode-se perceber que a multiplicação destas grandezas por um escalar não altera o resultado. Ou seja, a matriz E fica com um fator multiplicativo não determinado. Este fato acaba originando uma das maiores dificuldades na prática da odometria visual monocular pois a obtenção deste valor escalar se torna uma tarefa árdua.

A matriz essencial pode ser calculada utilizando uma quantidade n de pontos característicos. Há duas implementações frequentemente utilizadas, a de Nistér (2004) que utiliza 5 pontos e a de Longuet-Higgins (1981) que utiliza 8 pontos. Ambos algoritmos funcionam de forma iterativa, assim é natural considerar que o de 5 pontos é mais rápido conforme pode ser observado em estudo realizado em Scaramuzza e Fraundorfer (2011a).

A restrição obtida através da geometria epipolar nos permite obter várias equações. Seguindo a dedução do artigo de Nistér (2004), pode-se reescrever a equação 2.22 como $\tilde{q}^T\tilde{E} = 0$ conforme indicado nas equações a seguir. O índice de q indica uma das três dimensões, enquanto que os índices de E indicam um dos nove valores da matriz.

$$\tilde{q} = [p_1p'_1\ p_2p'_1\ p_3p'_1\ p_1p'_2\ p_2p'_2\ p_3p'_2\ p_1p'_3\ p_2p'_3\ p_3p'_3]^T \tag{2.23}$$

$$\tilde{E} = [E_{11}\ E_{12}\ E_{13}\ E_{21}\ E_{22}\ E_{23}\ E_{31}\ E_{32}\ E_{33}]^T \tag{2.24}$$

Assim, utilizando cinco pontos, obtém-se uma matriz 5x9. Em seguida são calculados quatro vetores X, Y, Z e W que compõem o núcleo da matriz. Esse cálculo pode ser realizado através de decomposição em valores singulares ou através de fatoração QR conforme especificado na literatura (NISTÉR, 2004). Assim, a matriz essencial é obtida através da equação 2.25, onde os valores de x, y, z e w são escalares.

$$E = xX + yY + zZ + wW \tag{2.25}$$

Visto que um GDL é perdido, um dos escalares não pode ser determinado, assim, assume-se w=1. Os detalhes de implementação podem ser encontrados em Nistér (2004). Na prática, este algoritmo é implementado de forma eficiente na biblioteca OpenCV.

2.2.3.2 RANSAC

O algoritmo de Nistér (2004) é utilizado em um *framework* RANSAC (*Random Sample Consensus*). Este *framework* consiste em uma abordagem matemática probabilística que pode ser aplicada a vários tipos de problemas. Este modelo é interessante para este problema em particular pois ele é robusto a erros povenientes de valores atípicos (*outliers*). No nosso problema, os valores atípicos se encontram nos pontos associados incorretamente pelo algoritmo de *Lucas-Kanade*. Fundamentalmente, escolhe-se aleatoriamente um subgrupo de pontos e calcula-se a a matriz essencial para estes pontos. Em seguida, a

validade da matriz encontrada é verificada calculando o erro para todos os outros pontos não utilizados. Os pontos que são validados pelo modelo são adicionados a um grupo *inlier*. O algoritmo é repetido iterativamente e por fim o grupo de pontos com o maior número de *inliers* é escolhido e um modelo é construído em cima destes pontos.

As matrizes R e t podem ser extraídas diretamente de E através de decomposição por valores singulares. Em geral há quatro possibilidades de pares R e t que satisfazem as restrições impostas pela geometria epipolar. Triangulando um ponto em ambas imagens pode-se determinar o par que representa a configuração correta (SCARAMUZZA; FRAUNDORFER, 2011a), (NISTÉR, 2004).

2.2.3.3 Cálculo do Fator Escalar

Conforme foi visto, as matrizes que compõem as retrições de geometria epipolar são homogêneas, assim, ao computar a matriz essencial um grau de liberdade é perdido, ficando-se apenas com 5 GDL (DINGFU; DAI; LI, 2016). O grau de liberdade perdido pode ser interpretado como um fator de escala que multiplica t. Para duas imagens consecutivas, este escalar representa a distância percorrida entre elas. Sabendo a frequência de captura da câmera ou o tempo entre a captura de duas imagens, este escalar pode ser obtido caso soubermos a velocidade do veículo.

Para aplicações de odometria binocular este fator escalar pode ser computado triangulando pontos. A geometria da montagem das câmeras permite triangulação de pontos 3D que possibilitam o cálculo da escala. Para o caso de odometria visual monocular, tendo apenas uma imagem para cada posição, não é possível determinar a escala diretamente. Segundo Scaramuzza e Fraundorfer (2011a), um método alternativo consiste em utilizar no mínimo três imagens consecutivos. Para cada par de imagens calcula-se pontos tridimensionais que em seguida são utilizados para se calcular a sua distância relativa. Este método, porém, depende de seguir pontos para múltiplos frames consecutivos, o que pode ser um problema dependendo da frequência de captura da câmera ou da velocidade do veículo.

Para solucionar este problema, várias soluções que utilizam informações de objetos na imagem foram propostas. Por exemplo, Dingfu, Dai e Li (2016) menciona literatura que utiliza a altura média dos pedestres, sensores GPS, giroscópios ou velocímetro. Soluções que envolvem detecção de objetos aumentam muito a complexidade do software e o custo computacional, além de dependerem da existência de objetos que seguem as premissas estabelecidas no algoritmo, sendo pouco robustas. A solução proposta por (DINGFU; DAI; LI, 2016) utiliza informação da altura da câmera instalada no carro juntamente com estimação de uma matriz de homografia que determina o plano diretamente a frente do veículo. Dentre as soluções para o caso monocular que utilizam apenas dois frames esta parece ser a mais robusta e melhor candidata para estudos que utilizam unicamente dados

de câmera monocular.

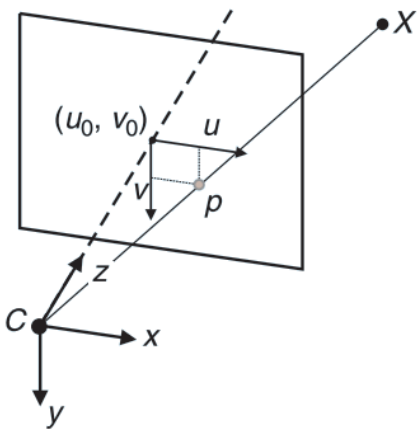
Um método interessante foi desenvolvido por Pereira (2018). Neste trabalho, o autor estima a constante escalar utilizando informação da altura da câmera e utiliza pontos no chão que são escolhidos cuidadosamente através de uma função de custo que considera a região do ponto em questão na imagem, a qualidade de seguimento do ponto entre diferentes frames e a altura estimada em relação ao chão. Outra solução prática para o problema é a utilização de um odômetro para obtenção de informação a respeito da velocidade estimada em cada imagem.

2.3 Modelo da Câmera

Técnicas de odometria visual podem ser aplicadas para diferentes modelos de câmeras. Pode-se utilizar, por exemplo, câmeras omnidirecionais, que possuem um campo de visão maior, ou câmeras estenopeicas (*pinhole*, buraco de alfinete). Para se trabalhar com as imagens obtidas através de uma câmera é necessário conhecer o modelo que mapeia os pontos tridimensionais do mundo real para uma projeção bidimensional da imagem. Assim sendo, usualmente torna-se mais simples trabalhar com câmeras estenopeicas.

O modelo destas câmeras consiste em considerar que a imagem é formada pela intersecção da luz que passa pelo centro da lente com o plano focal, conforme representação na figura 4. Segundo Szeliski (2011), as relações entre um ponto tridimensional e a sua projeção no plano da imagem podem ser descritas por uma matriz de calibração K definida pela equação 2.26.

Figura 4 – Representação da Câmera Estenopeica (*Pinhole*)



Fonte: (SCARAMUZZA; FRAUNDORFER, 2011a)

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

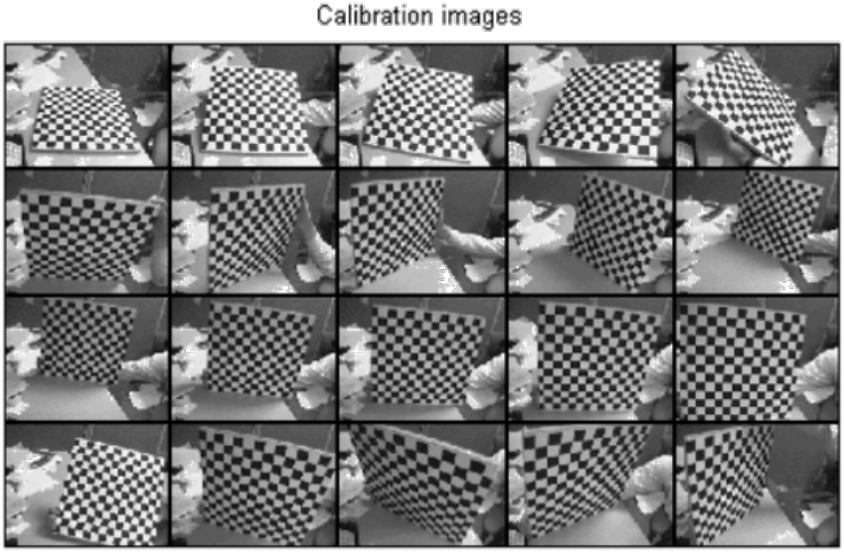
Neste modelo, f representa a distância focal, considerada comum em ambas as direções x e y. O centro óptico, expresso em coordenadas de píxeis é dado por (c_x, c_y). O processo de obtenção destes parâmetros é chamado de calibração da cãmera. Finalmente, um ponto $X = [x, y, z]^T$ é transformado em um ponto $p = [u, v, 1]^T$ homogêneo da seguinte forma:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.27}$$

2.4 Calibração da Câmera

O processo de calibração da câmera consiste em determinar através de imagens (normalmente padrões xadrez) os valores das variáveis intrísecas (conforme representado na matriz K apresentada anteriormente) e extrínsecas, como a distância entre duas câmeras no caso binocular. A principal ferramenta que facilita a calibração da câmera é a *Camera Calibration Toolbox for Matlab*, desenvolvida em Bouguet (2015). Essa ferramenta foi traduzida para *C* e integrada no *OpenCV*, cujas funções podem ser chamadas através de *Python*. Para calibrar uma câmera basta tirar fotos de um padrão xadrez em diversos ângulos diferentes conforme especificado na documentação da biblioteca. Na figura 5 encontram-se exemplos de imagens utilizadas no processo de calibração de uma câmera.

Figura 5 – Exemplo de Imagens de Calibração



Fonte: Bouguet (2015)

3 Metodologia

O objetivo definido consiste em estimar a trajetória percorrida por um veículo utilizando uma câmera instalada no mesmo. Em seguida, verificar se é viável integrar a solução programada em um micro-computador de baixo custo para execução em tempo real. Para atingí-lo, três etapas intermediárias foram definidas.

A primeira etapa, consistiu na definição e validação individual dos algoritmos envolvidos no sistema. Para isso, foi realizada notadamente, uma implementação do algoritmo de Lucas-Kanade, para verificação de sua exatidão, tempo de execução e comparação com uma implementação do *OpenCV*.

Na segunda etapa, a programação e desenvolvimento de um conjunto de algoritmos envolvidos em um sistema de odometria visual monocular são realizados. Neste experimento, uma base de dados padrão foi utilizada, de sorte que o funcionamento do algoritmo pudesse ser verificado.

A terceira etapa consistiu na aquisição de um percurso próprio através de uma câmera esportiva. Após a realização da gravação, as imagens foram tratadas de forma que os resultados provenientes do algoritmo de odometria visual pudessem ser comparados com o percurso de fato percorrido.

No restante deste capítulo, os algoritmos, softwares e hardwares utilizados para os experimentos de cada etapa são descritos.

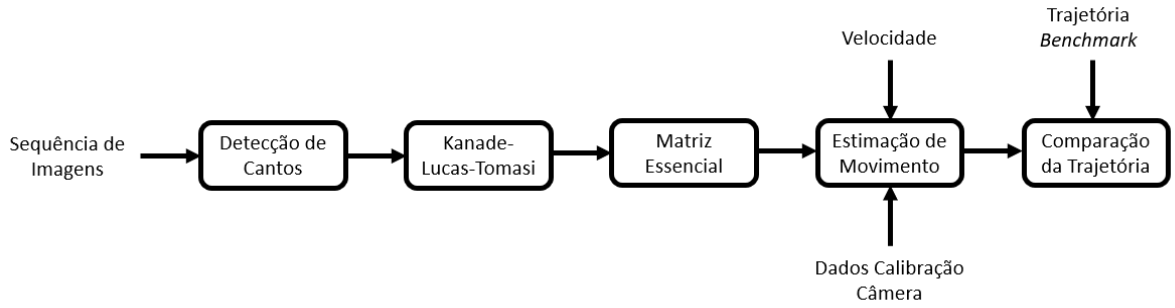
3.1 Definição de Algoritmos

Em um primeiro momento os algoritmos de cada etapa foram estudados conforme descrito no capítulo de revisão bibliográfica. Em um segundo momento partiu-se para a implementação dos mesmos. O algoritmo de Lucas-Kanade é implementado integralmente e uma comparação com a performance da implementação do *OpenCV* é realizada. Outros algoritmos importantes, como detecção de cantos de *Shi-tomasi* e estimação de movimento através de cálculo da matriz essencial são trazidos diretamente da biblioteca *OpenCV*.

A figura 6, mostra a sequência dos algoritmos utilizados assim como os dados de entrada. Estes são, essencialmente, uma sequência de imagens, a velocidade, posição do veículo e dados de calibração da câmera. Utiliza-se o detector de cantos de *Shi-Tomasi* para detecção de pontos característicos facilmente identificáveis em uma imagem. Posteriormente, utiliza-se o algoritmo de *Lucas-Kanade* para que pontos importantes de uma imagem sejam encontrados na imagem seguinte da sequência. Em seguida o cálculo da matriz essencial é realizado, esta matriz é importante pois através dela pode ser calculada a matriz de

rotação R e o vetor de translação t. Para a etapa de estimação de movimento (cálculo de R e t) os dados de calibração da câmera são necessários, que consistem na distância focal da câmera e no centro geométrico da imagem, e são informações constantes para cada câmera.

Figura 6 – Diagrama de Etapas e Algoritmos Utilizados



Fonte: Diagrama realizado pelo autor

Devido à natureza monocular do problema, conforme descrito no capítulo de fundamentação teórica, a determinação de um fator multiplicativo do vetor de translação homogêneo é necessária. Na metodologia escolhida, foi assumido que se tem a informação de velocidade do veículo. Na prática isso consiste em utilização de acelerômetros ou de um odômetro. Para teste da solução com uma base de dados padrão, a velocidade pode ser estimada através da distância euclidiana calculada a partir dos dados GPS fornecidos. Ou seja, apesar de se ter como objetivo determinar a localização do veículo através de odometria visual, os dados de GPS são utilizados neste caso como fonte da velocidade para que se possa validar o funcionamento dos algoritmos.

Cada um dos algoritmos da figura 6 possui diversas possibilidades de parâmetros a serem definidos, como por exemplo, o tamanho do ponto característico a ser detectado e o número de pontos mínimos que devem estar presentes em uma imagem. A definição destes é feita através de um estudo estatístico de desvios padrões, número de *outliers* e variância de cada parâmetro através de modelo *One-Way ANOVA*. A métrica que deve ser minimizada é o erro médio em um dado percurso.

3.2 Aquisição de Dados Próprios

Bases de dados padrões de qualidade que são muito úteis para avaliação e comparação de algorítmos. Estas são realizadas normalmente em condições ótimas, em cidades com asfalto excelente e pouco tráfego de veículos, especialmente no sentido contrário. Porém, estas características podem ser um ponto fraco quando se deseja avaliar a viabilidade de

um sistema que opere em um carro que trafega em condições normais de tráfego como, por exemplo, em uma metrópole como Porto Alegre.

Algumas situações normais e inevitáveis pouco presentes em datasets padrões podem prejudicar significativamente a validação do sistema em ambientes reais, como por exemplo: Oclusão quando há parada em semáforo; Carros trafegando em sentido contrário; Vias não asfaltadas; Carros trafegando em sentido perpendicular ao de movimento do observador na proximidade de esquinas. Além do mais, os equipamentos utilizados para obtenção de bases de dados de alta qualidade também são de alto custo. Logo, seria desejável obter uma base de dados justa de forma mais barata e mantendo a funcionabilidade do sistema. Estes fatores justificam a necessidade de se adquirir dados próprios.

Alguns critérios podem ser estabelecidos para obtenção de um trajeto experimental de qualidade. Primeiramente, deseja-se que todo o percurso seja asfaltado, pois um pavimento de má qualidade pode trazer vibrações para a câmera que tragam ruído e dificultem o funcionamento do algoritmo. Deseja-se também que não haja grandes diferenças de nível, visto que não pretende-se estudar variações de altitude. É preferível que não haja engarrafamentos nas vias. Assim, escolheu-se um percurso no bairro residencial Jardim Botânico em Porto Alegre. O percurso inteiro é asfaltado e o teste deve ser realizado em um final de semana para evitar tráfego excessivo de veículos. O percurso escolhido pode ser visualizado na figura 7. Nota-se também que há duas placas de PARE e cinco semáforos no percurso, observação que se mostrará relevante mais tarde.

Figura 7 – Ilustração do Percurso Definido

Fonte: Extraído do GoogleMaps e modificado pelo Autor

3.3 Hardware Utilizado

3.3.1 Computador para Simulação

Para teste dos algoritmos, um computador com sistema operacional *Windows* é utilizado. Este possui um processador *i7-4710HQ*, *2,5 GHz*, frequência turbo 3,5 GHz, Cache 6 Mb, quad-core e até 8 threads simultâneas. A memória RAM instalada é de 16,0 Gb. Visto que pode haver um número razoável (dezenas ou até mesmo centenas) de combinações de parâmetros que podem ser investigadas para várias possibilidades de trajetos, o uso de um computador capaz de executar vários threads é uma característica importante para redução do tempo de execução.

3.3.2 Câmera para Aquisição de Imagens

A câmera escolhida para coleta de dados foi uma câmera *GoPro Hero 5 Black* (INC., 2016) ilustrada na figura 8. Esta câmera foi escolhida por incluir captura de dados de velocidade e de posição através de giroscópios e acelerômetros operando a aproximadamente 400Hz e posição GPS a 18Hz. A configuração da câmera para captura de dados foi *30fps*, campo de visão *Linear*, resolução 1920 por 1080 píxeis, razão 16:9. O campo de visão

escolhido corresponde a uma visão com leve *zoom* no centro da imagem que tem como objetivo remover distorções causadas pela lente tipo *olho de peixe*. Porém, esta escolha tem como consequência a redução significativa do campo de visão em relação a datasets padrões que utilizam imagens formato *wide*.

Figura 8 – Imagem da GoPro Hero 5 Black

Fonte: Desenvolvido pelo Autor

Para aquisição da trajetória, a câmera foi fixada em cima de um carro utilizando um suporte padrão da marca *GoPro*. O modo vídeo foi acionado e o motorista do veículo percorre o trajeto definido. A câmera é fixada no teto do carro logo após o para-brisas, centralizada com distância igual em relação às portas laterais. O ângulo em relação ao plano horizontal é ajustado de forma que se possa visualizar ao máximo o chão imediatamente a frente ao carro, porém de forma que o capô do carro não seja visível. A maior diferença em relação às câmeras utilizadas em datasets profissionais se encontra nas laterais pois o campo de visão da *GoPro* no modo *Linear* é reduzido. Este pode ser um fator que dificulte o cálculo da rotação em curvas.

O video é salvo no cartão de memória no formato *.mp4*. Para utilização dos dados obtidos no algoritmo, é necessário convertê-lo em uma sequência de frames. Para isso, o software livre *VLC Media Player* é ser utilizado.

3.3.3 Escolha de Computador para Execução

Um computador de placa única é um computador no qual todos os componentes estão integrados em uma mesma placa. O principal objetivo de sua utilização é verificar a possibilidade de execução da solução em sistema embarcado. Deseja-se estudar se o sistema de odometria visual escolhido poderia ser executado em tempo real. Então, o

ponto de validação desta seção consiste na análise do tempo de processamento das imagens na plataforma escolhida.

Os critérios para seleção da melhor plataforma são essencialmente custo, capacidade de processamento e facilidade de implementação dos algoritmo. Uma possibilidade a ser analisada é o *Raspberry Pi Model 3 B+*, que no momento do início deste trabalho era a versão mais atual da marca. Este RPi é um computador de placa única BCM2837B0 (System on Chip), Quadcore *Cortex-A53*, 64-bit, 1.4GHz e 1Gb de memória LPDDR2 SDRAM.

Outra plataforma considerada é o *BeagleBone Black*. Este também é de placa única possui processador AM335x 1Ghz ARM Cortex-A8, 512Mb DDR3 RAM, 2 microcontroladores PRU 32-bit e memória flash 4Gb eMMC. Assim como o RPi, este chip também é compatível com Debian e Ubuntu. Esta plataforma possui vantagens interessantes, como maior número de GPIOs, interfaces PWM, portas seriais, conversores A/D e não haver necessidade de cartão SD. A memória interna deste processador poderia ser uma grande vantagem em relação ao seu concorrente para este trabalho pois a execução do algoritmo depende da leitura de imagens salvas na memória. Porém, os bancos de dados deste trabalho são volumosos e não caberiam na memória interna no BeagleBone. Outro incoveniente é que este dispositivo está disponível no Brasil a preço significativamente mais elevado.

Na tabela 1, pode-se encontrar um breve resumo das informações mais relevantes para escolha da plataforma neste trabalho.

Tabela 1 – Comparação das Plataformas RPi e BBB

Percurso	BeagleBone Black	Raspberry 3 B+
Tipo de SOC	TI AM33x	Broadcom BCM2837B0
Tipo de Core	Cortex A8	Cortex A53
Número de Cores	1	4
Clock da CPU	1 GHz	1.4GHz
RAM	512 Mb DDR3	1Gb LPDDR2 SDRAM

Fonte: Produzido pelo autor, dados de (BEAGLEBOARD, 2013) e (Raspberry Pi, 2018)

Apesar de o BeagleBone Black possuir algumas vantagens em relação ao RPi, considerou-se que para os critérios definidos neste trabalho, os quais não exigem utilização de GPIOs, o custo superior do BeagleBone Black, o número inferior de núcleos e frequência inferior, a plataforma que melhor atende os requisitos seria um *Raspberry Pi 3 B+*. Além das especificações básicas da placa mencionadas acima, foi adquirido um cartão miniSD Sandisk Ultra 32 GB com velocidade de leitura de 80Mb/s e dissipadores de calor.

3.4 Software Utilizado

3.4.1 Linguagem de Programação

Para programação da solução foi escolhida a linguagem de programção *Python*. Esta escolha se deu devido ao fato de a linguagem possuir um grande número de bibliotecas de tratamento de imagem e computação científica. Para este trabalho, as duas bibliotecas mais utilizadas são *OpenCV* (BRADSKI, 2000) e *numpy* (WALT; COLBERT; VAROQUAUX, 2011). Ambas permitem integrar funções programadas em *C* ou *C++* ao script em *Python*, que podem ser chamadas em alto nível e são executadas eficientemente. A biblioteca *OpenCV* é particularmente útil pois grande parte dos algoritmos de visão computacional utilizados neste trabalho possuem funções prontas extremamente otimizadas. Como por exemplo, o algoritmo de *Lucas-Kanade*, o detector de cantos de *Shi-Tomasi* e funções para calibração de câmeras.

3.4.2 Calibração da Câmera

Para estimação da matriz essencial é fundamental que se saiba os parâmetros intrínsecos da câmera. Assim, torna-se necessário obter os parâmetros da câmera utilizada para coleta de dados. As bibliotecas *OpenCV* nas linguagens *Python* e *C++* possuem funções que calculam os parâmetros que são desejados. Para isso, é necessário imprimir um padrão xadrez e fixá-lo a uma superfície de papel rígido de forma que não haja distorção em função de dobras ou ondulações no papel. Em seguida, vinte fotos do padrão completo são tiradas em diferentes posições e rotações. Por fim, estas imagens são registradas no computador e um *script* escrito em linguagem *Python* calcula a matriz contendo a distância focal da câmera e os centros ópticos.

3.4.3 Dados GPS e Acelerômetro

Câmeras esportivas acopladas a um carro podem ser utilizadas para captura de dados. Estas, integram unidades de medidas inerciais (inertial measurement units, IMUs) que medem por exemplo aceleração e informação GPS. A câmera escolhida foi uma câmera *GoPro* que é especificada na seção seguinte. Para extração de *metadata* com posição GPS e velocidade do veículo a partir do arquivo *.mp4* proveniente da câmera o software *DashWare* (GoPro Inc, 2017), versão 1.9.1, é utilizado. Um arquivo *.csv* é gerado que possui dados não tratados de vários dos sensores da câmera, como giroscópio de 3 eixos, acelerômetro de 3 eixos e GPS além de dados consolidados como velocidade e distância percorrida para cada intervalo de tempo de 50ms.

4 Resultados

4.1 Base de dados KITTI

A suíte KITTI *Vision Benchmark Suite*, criada pelo *Karlsruhe Institute of Technology* em parceria com o *Toyota Institute of Technology at Chicago* (URTASUN; LENZ; RAQUEL, 2012) disponibiliza gratuitamente imagens para desenvolvimento e teste de algoritmos de odometria visual, detecção e seguimento de objetos. As imagens são de alta resolução e há dados GPS disponíveis para comparação de resultados. Estas bases de dados foram obtidas utilizando um veículo, equipado com diversos sensores, que trafega nas ruas da cidade de Karlsruhe na Alemanha e em algumas rodovias da proximidade.

A utilização dos dados fornecidos pela base KITTI é útil pois permite que se teste os algoritmos em ambiente com dados obtidos a partir de equipamentos de qualidade. Assim evita-se problemas relacionados com baixa qualidade de imagem ou trepidação e permite que os pesquisadores da área avaliem seus algorítmos em uma base de dados padronizada. Na figura 9, tem-se um exemplo de primeira imagem retirada do primeiro percurso da base KITTI.

Figura 9 – Imagem Ilustrativa do Primeiro Percurso da Base de Dados KITTI

Fonte: KITTI Vision Benchmark Suite

Para análise de desempenho de odometria são fornecidas imagens coloridas e monocromáticas de duas câmeras de resolução 1392x512 píxeis. São disponibilizados para testes 11 trajetórias diferentes, com características distintas, como, por exemplo trajetórias em rodovias, no centro da cidade, com grande número de curvas, trajetórias retas ou com elevação. Além disso há dados de laser *Velodyne* [1], dados de calibração das câmeras e a posição do veículo obtida através de um sistema de localização de precisão que combina

[1] Velodyne LiDAR é uma empresa conhecida notamente por seu destaque e experiência na fabricação de dispositivos de medida de distância através de tecnologia laser.

GPS e uma unidade de medidas inerciais. O erro de localização em céu aberto do sistema é estimado em até 5cm segundo os autores (URTASUN; LENZ; RAQUEL, 2012).

Esta base de dados é utilizada nesse trabalho para validação e teste dos algoritmos desenvolvidos. Para o problema estudado, é utilizado apenas dados de uma câmera monocromática e dados da posição do veículo a cada imagem.

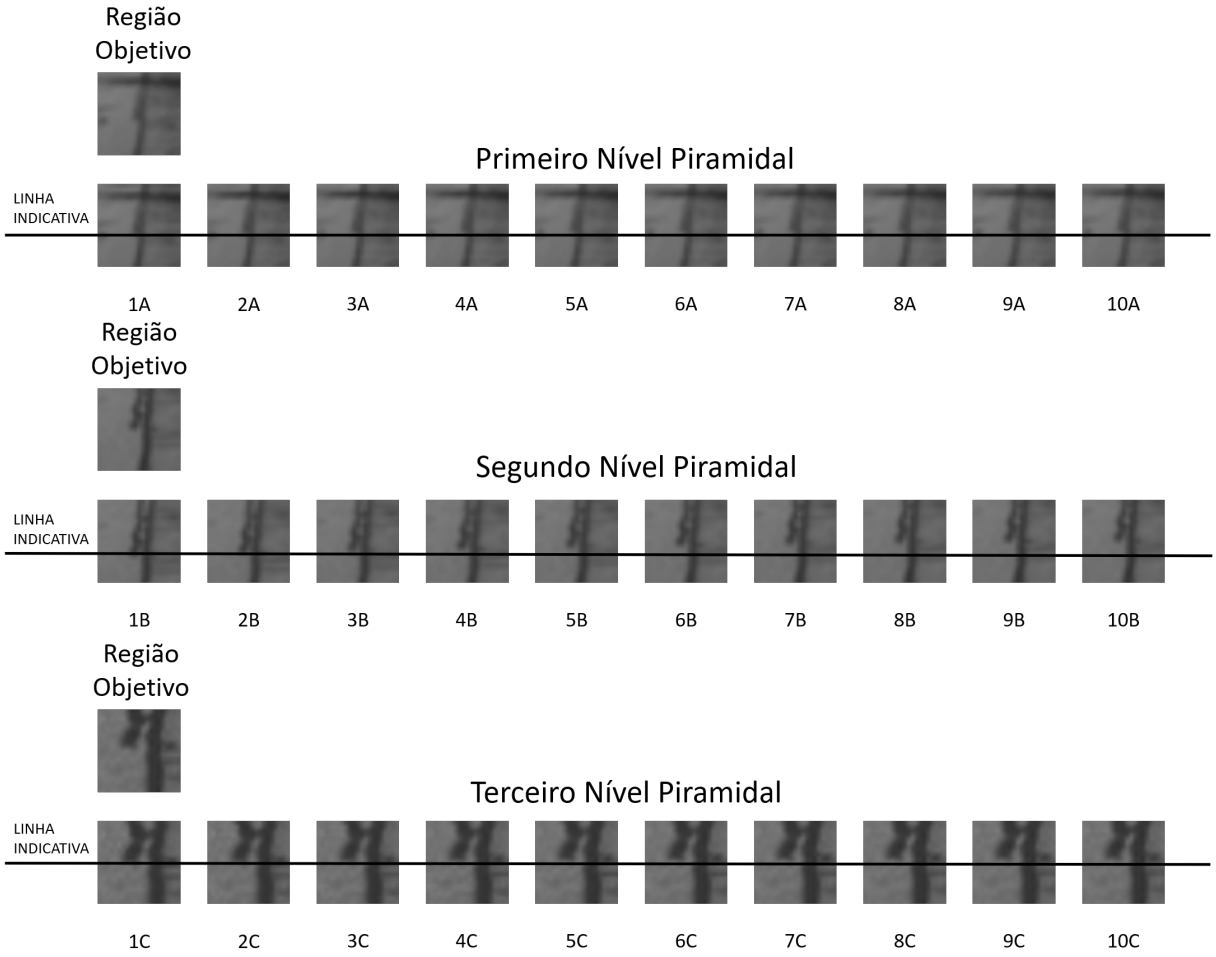
4.2 Resultados do Algoritmo de Lucas-Kanade

O algoritmo de Lucas-Kanade consiste essencialmente em encontrar a localização de uma região de uma imagem de índice $n-1$ em outra imagem de índice n. Isso é realizado encontrando o ponto central das regiões ao redor destes pontos, de forma que se minimize a diferença de brilho entre elas. Neste contexto uma abordagem piramidal foi implementada seguindo o algortimo descrito por Bouguet (2001). A principal ideia de uma implementação piramidal vem do fato de querer-se um algoritmo que não seja variante à escala do padrão detectado. Ou seja, quando há grandes movimentos relativos entre uma imagem e outra, o ponto que se deseja seguir pode mudar de dimensões ou se encontrar fora da região de busca do algoritmo. Para resolver este problema, considera-se inicialmente uma região de busca maior e diminui-se a resolução do padrão desejado.

O algoritmo é inicializado com duas imagens, $n-1$ e n, e pontos centrais que definem regiões da imagem $n-1$ que se deseja encontrar na imagem n. O processo de busca de cada região da primeira imagem na segunda imagem é inicializado no píxel central da região na primeira imagem. Em seguida, iterações são realizadas com objetivo de ajustar a posição da região na segunda imagem e diminuir a diferença de brilho entre elas, calculado conforme indicado pela equação 2.5. O deslocamento da região na imagem n em relação à imagem $n-1$ é calculado conforme a equação 2.16.

Na figura 10 pode-se visualizar a progressão do algoritmo conforme a localização de um ponto é buscada. Utilizando um esquema piramidal de três níveis, cada uma das linhas de imagens A, B e C representa as iterações de um nível piramidal. Parte-se do ponto da primeira imagem diminuindo a resolução de forma que a região possa ser vista mais "de longe". Nesta figura, pode-se visualizar a progressão do algoritmo para cada nível conforme as iterações, indicadas por um número, são realizadas. A linha horizontal é uma linha indicativa com o objetivo de ajudar a visualização das diferenças entre cada iteração.

Figura 10 – Iterações do Lucas-Kanade Piramidal



Fonte: Desenvolvido pelo autor

Primeiro Nível Piramidal: Parte-se, na imagem 1A, das coordenadas no qual o padrão objetivo é encontrado na imagem anterior. Procura-se, em resolução reduzida duas vezes (cada dimensão da imagem possui um quarto dos píxeis), a região objetivo na nova imagem. A procura consiste em um processo iterativo, composto neste exemplo por dez iterações indicadas por um número e uma letra, partindo na primeira iteração 1A e terminando na 10A. Nota-se na figura 10 que conforme as iterações avançam elas deslocam sensivelmente a região observada para baixo, de forma que seja mais semelhante à região objetivo da imagem anterior, diminuindo a função de erro definida.

Segundo Nível Piramidal: Esta camada consiste nas imagens com resolução reduzida uma vez (metade das dimensões da imagem original), iterações definidas com índices de 1B até 10B. Parte-se do ponto central encontrado ao final das iterações da primeira camada (10A). Conforme as iterações passam, nota-se que a região examinada desloca-se claramente para baixo. Neste exemplo, esta é a camada que teve o maior deslocamento entre as três, como pode-se visualizar pela grande diferença entre as imagens 1B e 10B.

Terceiro Nível Piramidal: Esta camada é tratada na resolução original da imagem. Compara-se o ponto buscado diretamente com a região ao redor do ponto encontrado ao final do processo iterativo da segunda camada (10B). Nota-se que no exemplo da figura 10 a posição encontrada no fim da segunda camada já é bem próxima da posição desejada, assim, há pouco deslocamento a ser realizado na camada final.

Ao final dos processos iterativos, encontra-se a nova posição do padrão buscado. Cada iteração dentro do processo piramidal permite uma melhora da localização da região buscada na ordem de fração de píxeis. Este fato é o que justifica a necessidade de cálculo subpixel conforme desenvolvimento teórico realizado.

4.3 Implementação Própria do Lucas-Kanade

Nesta parte do trabalho, tem-se por objetivo desenvolver uma implementação própria funcional. O código desenvolvido foi programado em linguagem *Python*. Este, consiste essencialmente nas equações de 2.5 a 2.16 programadas utilizando a biblioteca de cálculo científico *numpy*.

Para ilustrar o funcionamento do algoritmo, um ponto da primeira imagem do primeiro percurso da base *KITTI* foi escolhido manualmente utilizando como único critério que haja alguma forma ou padrão distinguível na região ao redor deste ponto. A função do algoritmo é encontrar na imagem seguinte a posição do píxel central do padrão escolhido. Na figura 11, pode-se visualizar o ponto para qual o método é aplicado. A imagem da esquerda é a primeira imagem da sequência, com um círculo indicando a região escolhida. A imagem da direita é uma imagem posterior[2], onde um círculo indica a localização encontrada para o padrão correspondente nesta imagem.

[2] Foi escolhida a terceira imagem da sequência pois o movimento em relação à primeira é maior, o que amplifica o deslocamento e facilita a comparação e visualização.

Figura 11 – Resultado do Lucas-Kanade para um Ponto

Fonte: Desenvolvido pelo autor

4.4 Comparação do Método de Lucas-Kanade

Afim de verificar a coerência do algoritmo próprio, decidiu-se compará-lo à implementação da biblioteca *OpenCV*. Para fazê-lo, a função *calcOpticalFlowPyrLK* deve ser utilizada, com parâmetro de profundidade piramidal definido em 3 níveis para que esteja comparável com a implementação própria.

Na presente aplicação o algoritmo de LK é utilizado sobre vários pontos para um par de imagens. Por exemplo, pode-se visualizar na figura 12, uma imagem na qual ambas as implementações são comparadas. Nesta comparação, duas imagens consecutivas são processadas. Na primeira delas, são identificados 100 pontos utilizando o detector de *Shi-Tomasi*. Na segunda, é executado o algoritmo próprio e o da biblioteca *OpenCV*. Assim, é possível comparar a taxa de acertos e o tempo de execução de cada um dos algoritmos. Visto que a implementação própria foi desenvolvida com três níveis piramidais, a função importada foi configurada também para três níveis de forma que a comparação seja justa.

Figura 12 – Comparação Algoritmo Próprio x OpenCV

Fonte: Desenvolvido pelo autor

Para comparar o desempenho de ambas implementações, foi planejado um experimento com dez pares distintos de imagens de diferentes percursos. Na primeira imagem de cada um destes pares foi executado o detector de *Shi-Tomasi* para encontrar 100 pontos. Na segunda imagem de cada par, tanto o algoritmo próprio quanto o da biblioteca importada foram executados. As duas características mais relevantes para este algoritmo são tempo de execução e taxa de acerto. Assim, estes parâmetros foram anotados. A taxa de acerto foi analisada manualmente para ambos algoritmos, verificando se o ponto encontrado para a segunda imagem está coerente com o respectivo ponto encontrado na primeira imagem. O par de figuras analisado em cada célula foi escolhido sob condição que os pontos encontrados na primeira imagem estejam distribuídos de forma que seja possível diferenciá-los (não estejam próximos o suficiente para haver sobreposição de suas regiões respectivas). Na tabela 2 tem-se os valores encontrados.

Tabela 2 – Resultados experimentais de comparação entre algoritmo próprio e biblioteca OpenCV.

Número	Percurso	Imagens	T Exec. Próprio [s]	Acertos Próprio	T Exec. OCV	Acertos OCV
1	0	0-1	13,78s	62/100	6ms	97/100
2	1	100-101	19,18s	62/100	5ms	66/100
3	3	1-2	13,00s	54/97	6ms	94/97
4	4	0-1	16,81s	54/100	5ms	73/99
5	5	0-1	21,57s	98/100	8ms	100/100
6	6	0-1	17,03s	63/100	5ms	70/99
7	7	10-11	17,35s	89/100	7ms	89/100
8	8	0-1	21,66s	59/100	6ms	97/100
9	10	100-101	10,05s	74/99	5ms	79/99
10	10	0-1	19,31s	69/100	6ms	96/100

Fonte: Desenvolvido pelo autor

Ao comparar a performance com a implementação do *OpenCV*, percebe-se que este segundo é mais exato pois sua taxa de acerto média no experimento realizado foi de $86,6\%$, enquanto que a taxa de acerto média da implementação própria foi de $68,6\%$. A biblioteca *OpenCV* é programada em linguagem *C/C++* e permite cálculo da ordem de mil vezes mais rápidos. Além disso, a implementação deles utiliza processamento paralelo aproveitando todos os núcleos do computador o que torna a execução do algoritmo diversas vezes mais rápida.

Testou-se realizar uma implementação fazendo os cálculos apenas com números inteiros, com o objetivo de reduzir o tempo de processamento e a complexidade adicional proveniente da utilização de números *float*. Porém, observou-se que o algoritmo não conseguiu convergir visto que a contribuição de cada iteração é inferior a um píxel, evitando a convergência do algoritmo. Logo, não faz sentido fazer uma comparação de performance neste caso.

Nota-se também, que a função carregada do *OpenCV* possui a funcionalidade de excluir o ponto quando ele não é encontrado ou é considerado fora da imagem. Tal funcionalidade não foi implementada no algoritmo próprio.

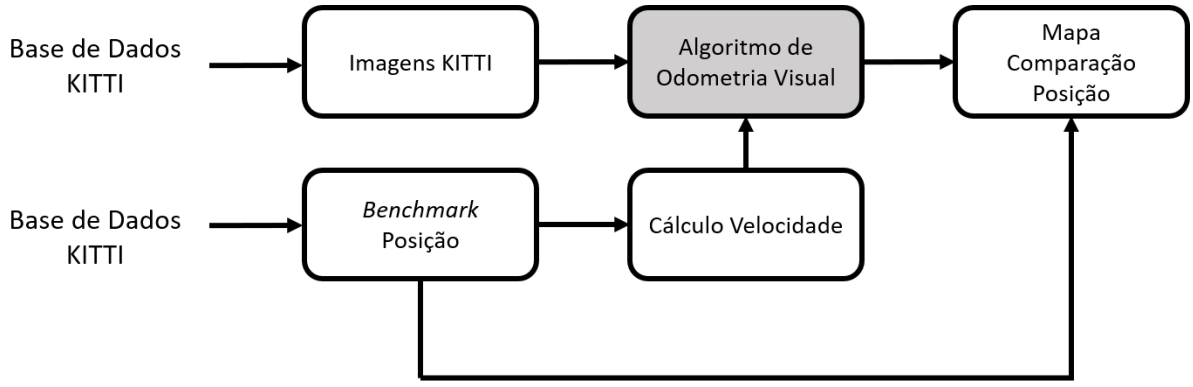
O desempenho do algoritmo próprio, principalmente no quesito tempo de execução poderia ser melhorado rearranjando a ordem na qual os cálculos são realizado, vetorizando operações matriciais e utilizando a biblioteca *multiprocessing* para utilização de múltiplos núcleos em paralelo. Porém, dada a existência de uma solução software livre, não haveria sentido em fazer este retrabalho visto que é apenas uma parte da sequência de algoritmos utilizados para estimação de trajetória via odometria visual. Com base nestas observações, decidiu-se que para a aplicação do presente trabalho, seria preferível utilizar para o algoritmo de *Lucas-Kanade* a implementação do OpenCV.

4.5 Odometria Visual com dados KITTI

Nesta seção, verifica-se os resultados obtidos com a implementação da *pipeline* completa de algoritmos necessários para estimação da trajetória via odometria visual monocular. Na figura 13 encontra-se um esquema indicando as etapas utilizadas para estimação da posição. O bloco "Algoritmo de Odometria Visual" refere-se ao conjunto de algoritmos conforme indicado na figura 6.

Para validação do funcionamento dos algoritmos, a base de dados KITTI é utilizada pois permite a comparação direta da resposta obtida através de posição GPS de alta qualidade. Conforme pode ser visto na figura 13, o algoritmo de odometria visual possui essencialmente duas entradas, um par de imagens por iteração e a respectiva velocidade na imagem correspondente. Como a base de dados *KITTI* não possui dados de velocidade, utiliza-se a posição entre duas imagens e calcula-se a distância euclidiana entre dois pontos para determinar o fator escalar que multiplica a matriz de translação t.

Figura 13 – Organigrama de Fluxo de Informações Solução KITTI



Fonte: Desenvolvido pelo autor

Inicialmente, a escolha dos pontos que deverão ser seguidos é feita através do detector desenvolvido por SHI e Tomasi (1994). Assim, utiliza-se uma função do *OpenCV* que realiza esta conta. Segundo esse mesmo autor, esta escolha é ótima pois este algoritmo de seleção de pontos é construído com base no funcionamento do *Lucas-Kanade*. Nesta etapa, o número de pontos a serem seguidos a cada imagem é determinado. Os parâmetros fundamentais para configuração desta função são: 1) número máximo de pontos a serem encontrados ordenados pela qualidade do ponto; 2) nível de qualidade, parâmetro filtrando a qualidade dos pontos encontrados em relação ao melhor ponto da imagem; 3) tamanho do bloco utilizado para cálculo da derivada da imagem durante a execução do algoritmo.

Para seguimento dos pontos conforme as imagens passam, utiliza-se o algoritmo de *Lucas-Kanade* do *OpenCV* pois é mais eficiente computacionalmente e mais exato que a versão própria desenvolvida. Assim utilizou-se a função *calcOpticalFlowPyrLK*. Esta possui como parâmetros: 1) o tamanho da janela de pesquisa em cada nível piramidal; 2)

número de camadas piramidais a serem utilizadas; 3) critérios de para, como o número máximo de iterações por nível piramidal, erro mínimo para continuação das iterações.

Após ter-se encontrado a posição dos pontos na imagem posterior, calcula-se a matriz essencial. Esta função possui como parâmetros: 1) método de estimação, neste trabalho utilizado *RANSAC*; 2) intervalo de confiança desejado no qual a matriz estimada está correta. Em seguida a função *recoverPose* é executada, que permite que as matrizes de rotação (R) e de translação (t) sejam encontradas.

4.5.1 Primeiro Experimento - Percurso Único

Conforme visto no capítulo de metodologia, os algoritmos escolhidos possuem vários parâmetros diferentes que podem ser definidos.

Para parâmetro nível de qualidade do algoritmo de *Shi-Tomasi* pode ser utilizado o valor padrão de $0,01$ pois nas imagens estudadas haverá um número muito mais elevado de pontos detectados, que são disponibilizados em ordem de qualidade, em relação ao número de pontos desejados.

No algoritmo de seguimento de pontos de *Lucas-Kanade*, a profundidade da pirâmide é configurada para 5 níveis piramidais, o que permite encontrar pontos bem distantes de sua posição original (cada dimensão da imagem é reduzida quatro vezes no nível mais alto) e não prejudica o tempo de execução, pois se o erro for pequeno a iteração não é realizada.

Dessa forma, os parâmetros mais interessantes para serem analisados são: Tamanho da janela utilizada para o algorítmo de *Lucas-Kanade*; tamanho do detector de *Shi-Tomasi*; número de pontos detectados e número de pontos mínimo que gera redefinição de novos pontos. Há um grande número de combinações possíveis considerando estas variáveis.

Para o cálculo da matriz essencial também é necessário indicar os dados intrínsicos da câmera utilizada para gravação. Utilizou-se os parâmetros fornecidos pelo KITTI, que são: distância focal 707.1 e ponto central (601.88 e 183.11).

Em um primeiro experimento, decidiu-se aplicar para uma dada trajetória várias cobinações diferentes de parâmetros, de forma que os que fossem claramente muito inferiores pudessem ser descartados. Dessa forma, três parâmetros foram variados e o percurso inteiro é calculado para cada um deles. Em cada uma das iterações o erro médio e o erro máximo são calculados.

- Tamanho da Janela do Lucas-Kanade: 13, 15, 17, 19, 21 px
- Tamanho do detector de *Shi-Tomasi*: 3, 5, 10 px
- Número de pontos do detector ao recalcular e número mínimo para acionamento de recálculo: (2000/1600), (1500/1200), (1000/800), (500/400) pontos

No total, são 60 possibilidades de combinações visto que pode haver interação entre as mesmas. Na tabela 3 podem ser encontrados os resultados obtidos, para que estes não se tornem demasiadamente longos, as combinações com Detector de Shi-Tomasi tamanho 10 ou tamanho da janela do Lucas-Kanade 13 foram retirados desta tabela. Os resultados do experimento podem ser encontrados no Apêndice A.

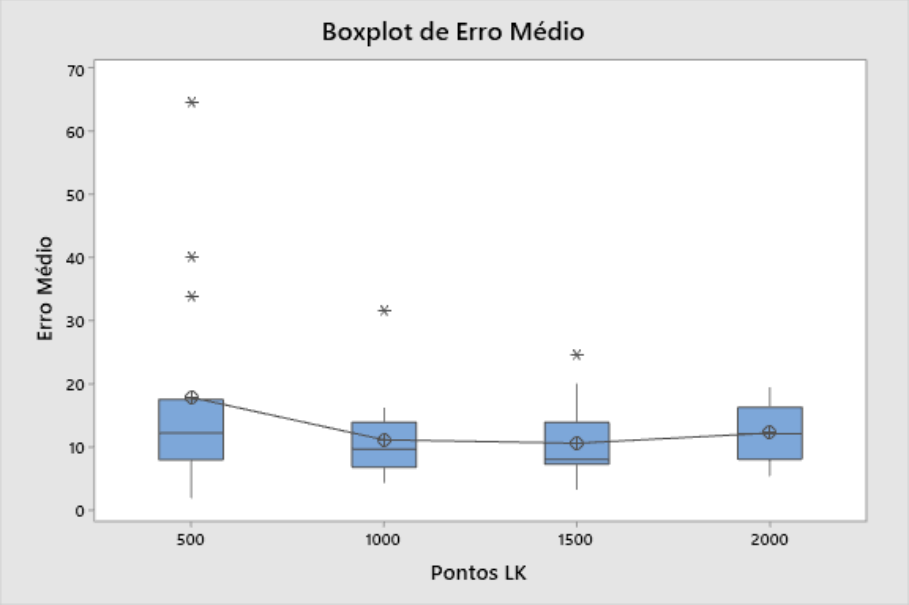
Tabela 3 – Resultados Experimentais Percurso Único

Tempo [s]	Janela ST [px]	Janela LK [px]	Pontos LK	Erro Máximo [m]	Erro Médio [m]
87.96	3	15	2000	17.77	8.86
81.74	3	15	1500	20.82	11.02
65.86	3	15	1000	15.89	8.31
49.87	3	15	500	33.52	13.59
90.87	5	15	2000	35.61	16.34
89.89	5	15	1500	5.49	3.34
63.52	5	15	1000	8.98	4.46
50.32	5	15	500	4.71	2.03
88.61	3	17	2000	51.90	12.11
73.84	3	17	1500	27.98	7.31
60.98	3	17	1000	20.43	12.18
57.23	3	17	500	80.11	40.07
96.73	5	17	2000	18.56	9.57
76.56	5	17	1500	24.32	7.75
61.14	5	17	1000	35.19	10.66
49.87	5	17	500	41.05	11.16
142.52	3	19	2000	19.69	8.10
118.89	3	19	1500	18.81	10.51
120.08	3	19	1000	30.27	16.14
54.67	3	19	500	14.60	7.34
96.29	5	19	2000	15.17	6.09
80.92	5	19	1500	24.91	8.07
66.89	5	19	1000	11.11	5.69
51.35	5	19	500	36.91	9.70
94.14	3	21	2000	46.38	15.25
79.82	3	21	1500	25.67	7.31
65.01	3	21	1000	25.33	15.79
51.69	3	21	500	122.53	64.76
101.22	5	21	2000	40.49	19.12
85.66	5	21	1500	27.20	8.15
68.39	5	21	1000	19.29	8.42
57.24	5	21	500	33.52	14.01

Fonte: Desenvolvido pelo autor

Com base nestes dados experimentais, tem-se como objetivo reduzir a quantidade de parâmetros a serem estudados. Para melhor ilustrar a relação de cada variável com o erro médio utilizam-se diagramas *Boxplot*. Por exemplo, na figura 14, pode-se visualizar a relação do erro médio com o número de pontos seguintos pelo algoritmo de *Lucas-Kanade*. A região dentro da caixa correponde a valores dentro dos dois quartis intermediários, e a linha horizontal à mediana. Os pontos marcados por asteríscos correspondem a valores atípicos enquanto que as linhas verticais indicam o *range* dos dados à exceção destes valores atípicos.

Figura 14 – Boxplot do Erro Médio em Função do Número de Pontos Seguidos



Fonte: Desenvolvido pelo Autor

Das quatro possibilidades de valores analisados para o número de pontos seguidos, decidiu-se, após análise da figura 14 e da tabela 4, excluir da região de análise do próximo experimento o nível correspondente a 500 pontos. Este, apresentou número elevado de valores atípicos, além de grande variabilidade do erro, e média e mediana alta em comparação com as outras possibilidades.

Tabela 4 – Resultados do Boxplot para Pontos LK

Pontos LK	Média	Desvio Padrão	*Outliers*
500	17,90	16,33	3
1000	11,18	6,68	1
1500	10,63	5,60	1
2000	12,23	4,64	0

Fonte: Desenvolvido pelo autor

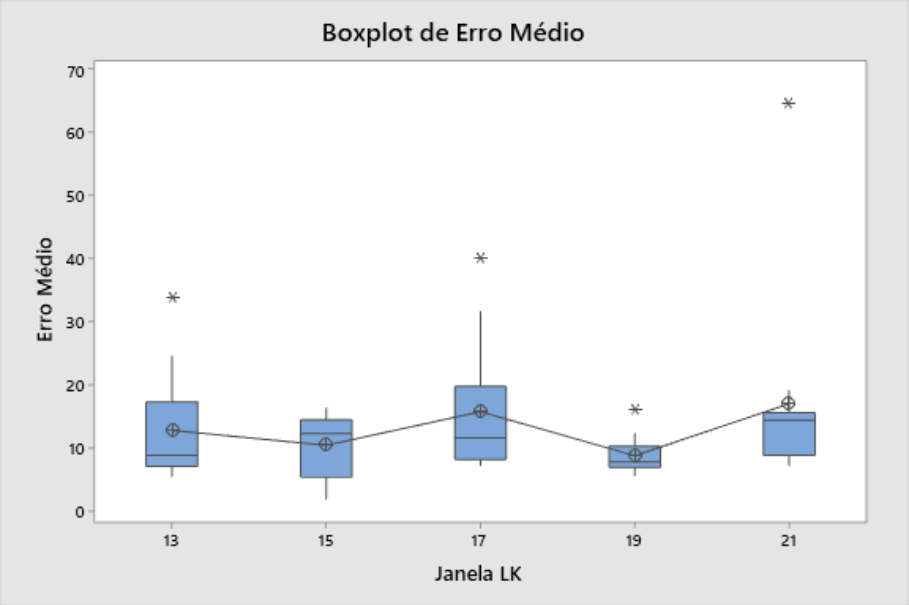
Pode-se perceber, através da análise da figura 15 e da tabela 5, que os níveis 13, 17, e 21 possuem *outliers*. Além disso, eles possuem variabilidade alta e valores médios superiores em relação aos níveis 15 a 19. Assim, decidiu-se examinar apenas estes dois últimos níveis no próximo experimento.

Tabela 5 – Resultados do Boxplot para Janela LK

Janela LK	Média	Desvio Padrão	*Outliers*
13	12,84	8,75	1
15	10,47	4,98	0
17	15,79	10,37	1
19	8,818	2,97	1
21	17,05	15,46	1

Fonte: Desenvolvido pelo autor

Figura 15 – Boxplot do Erro Médio em Função do Tamanho do Seguidor de LK



Fonte: Desenvolvido pelo Autor

O *boxplot* para a variável que controla o tamanho da janela de detecção de pontos do algoritmo de *Shi-Tomasi* também foi realizado. Porém, não foram tiradas conclusão significativas deste, que manteve os três níveis de análise para o próximo experimento.

Com base nestas análises foi decidido realizar o próximo experimento, abrangendo todos os percursos, com os seguintes parâmetros e níveis: tamanho da janela do LK: 15 e 19 px; tamanho do detector: 3, 5 e 10 px, número de pontos: 2000/1600, 1500/1200, 1000/800.

4.5.2 Segundo Experimento - Múltiplos Percursos

Em um segundo experimento, as combinações dos parâmetros candidatos são testados para todos os percursos disponíveis na base de dados *KITTI*. Considerando que há 11 percursos, e 2 parâmetros com 3 níveis, e um parâmetro com 2 níveis, tem-se 198 testes a serem realizados. Em cada uma das iterações, o erro médio e o erro máximo são

calculados. O tempo total de execução dos testes foi de aproximadamente 5h30min. Na tabela 6 podem ser encontrados os resultados dos testes, filtrados para os percursos de número 0, 2, 5 e 9, pois estes são mais longos e apresentam mais curvas, para as janelas de *Shi-Tomasi* de 3 e 5 px e para os pontos de 1500 e 2000. Os resultados para os outros percursos podem ser encontrados no Apêndice A.

Tabela 6 – Resultados Experimentais Múltiplos Percursos

Percurso	Tempo [s]	Janela ST [px]	Janela LK [px]	Pnts LK	Erro Máx. [m]	Erro Méd. [m]
0	238.93	3	15	2000	22.42	7.34
0	257.49	10	15	2000	17.81	7.24
0	231.21	3	19	2000	41.81	13.67
0	244.23	10	19	2000	14.39	7.47
0	221.30	3	15	1500	15.70	8.30
0	255.93	10	15	1500	28.86	9.57
0	208.07	3	19	1500	48.43	21.27
0	224.57	10	19	1500	14.90	5.35
2	268.02	3	15	2000	28.57	12.40
2	304.53	10	15	2000	40.88	14.11
2	241.65	3	19	2000	39.87	25.18
2	279.69	10	19	2000	24.02	10.04
2	221.88	3	15	1500	55.19	22.00
2	254.30	10	15	1500	22.54	11.13
2	207.54	3	19	1500	42.86	21.89
2	273.32	10	19	1500	15.67	7.14
4	17.41	3	15	2000	2.34	1.23
4	14.01	10	15	2000	6.94	3.15
4	14.01	3	19	2000	4.14	2.15
4	14.22	10	19	2000	5.92	2.17
4	11.26	3	15	1500	2.45	1.37
4	13.91	10	15	1500	6.94	3.15
4	11.59	3	19	1500	2.71	1.55
4	14.42	10	19	1500	5.92	2.17
9	89.81	3	15	2000	17.77	8.86
9	87.32	10	15	2000	28.77	14.69
9	87.30	3	19	2000	19.69	8.10
9	96.04	10	19	2000	15.46	7.47
9	69.93	3	15	1500	20.82	11.02
9	86.87	10	15	1500	26.95	13.93
9	70.09	3	19	1500	18.81	10.51
9	92.95	10	19	1500	17.11	7.58

Fonte: Desenvolvido pelo autor

Com os resultados adquiridos, resta escolher um modelo para definição da escolha dos parâmetros que minimizam o erro. Para isso, decidiu-se utilizar análise ANOVA, que consiste fundamentalmente em verificar se há influência significativa de parâmetros em uma determinada variável dependente. Para realizar esta tarefa, foi utilizado o software *MiniTab* e um modelo *One-Way ANOVA* para cada variável a ser estudada.

Primeiramente analisou-se a influência do parâmetro tamanho da janela do algoritmo Lucas-Kanade. Sendo assim, faz-se uma análise da variância cuja variável de

resposta é o erro médio. Foi obtido que para esta variável, o valor de F calculado [3] é de 0,03. Visto que o valor de F encontrado é muito baixo, não se consegue rejeitar a hipótese nula de igualdade de variâncias. Sendo assim, não é possível concluir que as médias são significativamente diferentes.

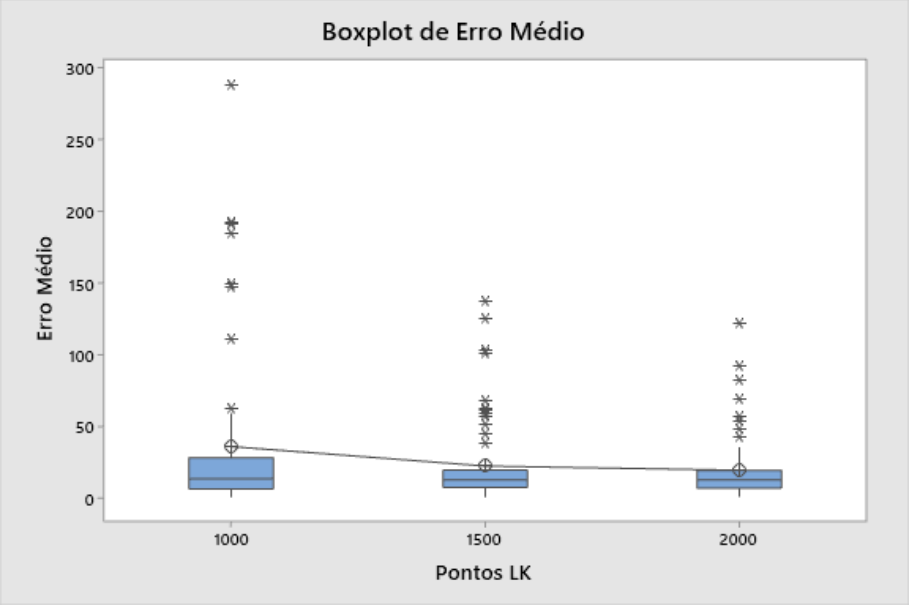
Em seguida, analisou-se a importância da variável que controla o número de pontos a serem seguidos pelo algoritmo de *Lucas-Kanade*. Obteve-se para este parâmetro o valor de F calculado de 3,18. Utilizando uma tabela de Fischer com $\alpha = 0,05$, 2 graus de liberdade no parâmetro estudado e 195 no erro, pôde-se concluir, com 95% de certeza que este parâmetro é significativo para o erro médio. Através do *boxplot* da figura 16, pode-se visualizar que o nível de menor desvio padrão é o de 2000 pontos.

Tabela 7 – Resultados do Boxplot para Pontos LK

Janela LK	Média	Desvio Padrão	*Outliers*
1000	35,97	58,68	8
1500	22,44	29,05	12
2000	19,51	22,73	8

Fonte: Desenvolvido pelo autor

Figura 16 – Boxplot do Erro Médio em Função do Número de Pontos Seguidos



Fonte: Desenvolvido pelo Autor

Por fim, verifica-se a importância da varíavel que controla o tamanho do detector de pontos *Shi-Tomasi*. O valor obtido para F calculado foi de 0,84, considerando 2 graus

[3] F se refere à distribuição de Fischer, comumente utilizada em testes estatísticos como a análise de variância.

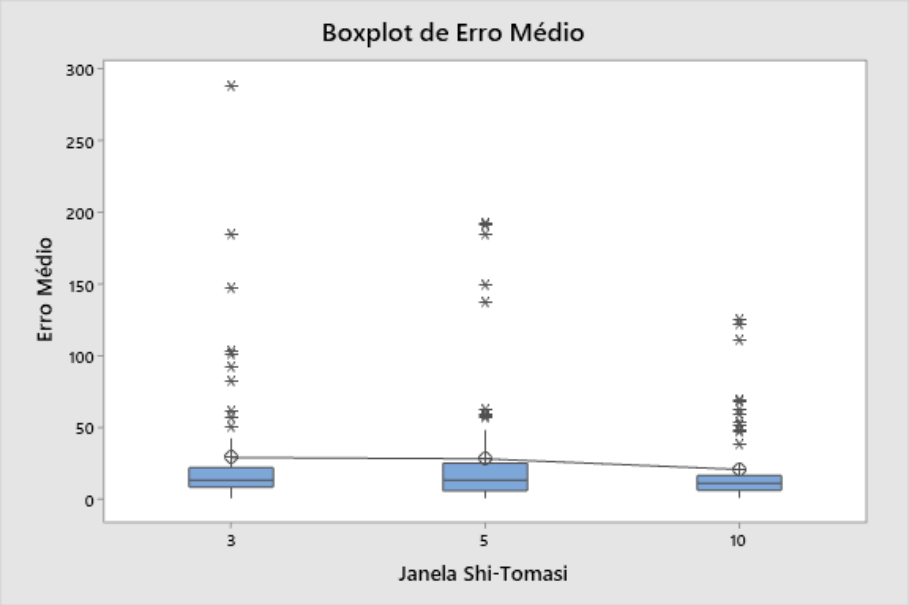
de liberdade desta variável e 195 do erro. Desta forma, não foi possível rejeitar a hipótese nula, indicando que não se pode afirmar que há diferença significativa entre suas médias. Na figura 17, apesar de os diferentes níveis terem médias parecidas, estes possuem valores atípicos de diferentes magnitudes, fazendo com que a janela de tamanho 10 px tenha o menor desvio padrão. Dessa forma, utilizar-se-á esta informação para a tomada de decisão do modelo escolhido.

Tabela 8 – Resultados do Boxplot para Janela Shi-Tomasi

Janela LK	Média	Desvio Padrão	*Outliers*
3	29,06	47,14	11
5	28,13	44,11	8
10	20,73	27,40	12

Fonte: Desenvolvido pelo autor

Figura 17 – Boxplot do Erro Médio em Função do Tamanho da Janela de ST



Fonte: Desenvolvido pelo Autor

4.5.3 Definição de Parâmetros para Dados KITTI

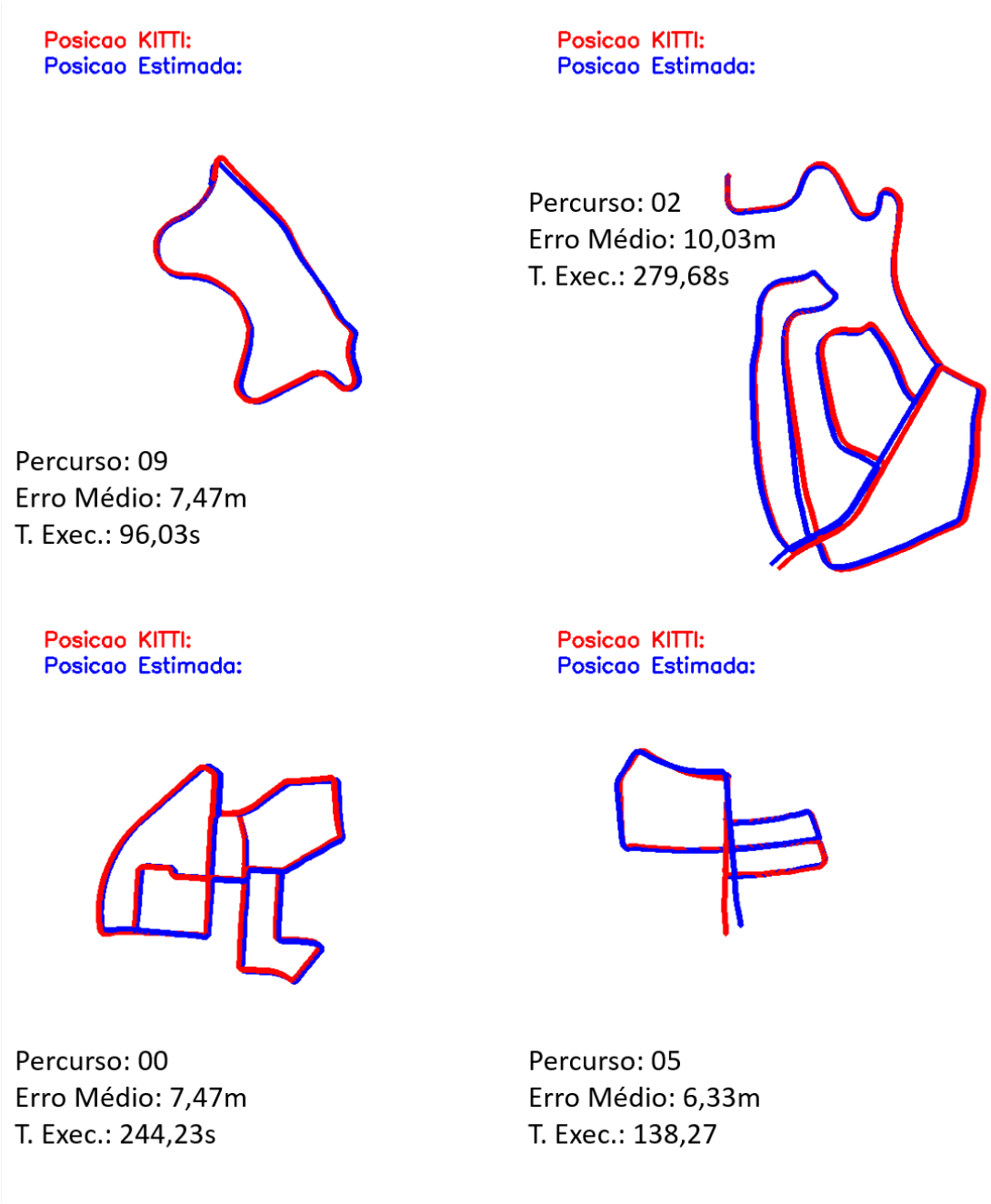
A partir dos resultados obtidos nas duas subseções anteriores, uma escolha de parâmetro para execução do algoritmo deve ser realizada. Assim, foi mostrado estatisticamente que a escolha mais adequada para o número de pontos do algoritmo de *Lucas-Kanade* é de 2000 pontos. Esta escolha é natural pois conforme mais pontos são seguidos menores são as chances de um erro de cálculo ou um valor atípico estragar o estimação do movimento de forma global. Para o tamanho da janela de *Lucas-Kanade* a análise de variância indicou que não há diferença significativa entre os valores de 15 e 19px. Porém, visto que um valor precisa ser obrigatóriamente definido, foi escolhido 19px, pois este apresentou o

menor desvio padrão conjuntamente com a menor média. Para o tamanho do detector de *Shi-Tomasi*, apesar de não haver demonstração estatística, a interpretação da figura 17 leva a crer que os valores atípicos para 10px tendem a ser menos significantes reduzindo o desvio padrão, assim escolheu-se este valor.

Os testes foram realizados nos percursos sem levar em consideração o número de frames, ou seja, a distância total percorrida pelo veículo. Visto que a solução escolhida possui um problema de acúmulo de erro (erros no início do trajeto influenciam todos os próximos valores) esta característica pode ser significante. Por exemplo, em um trajeto longo, erros cometidos no início penalizarão severamente o erro médio do trajeto, enquanto que erros graves no fim do trajeto serão diluídos ao calcular o erro médio.

Na figura 18, pode-se visualizar quatro percursos disponibilizados pelo dataset KITTI juntamente com o correspondente caminho encontrado através da metodologia descrita. Estes percursos foram escolhidos pois estão entre os mais longos da base de dados, assim tornam-se mais interessantes a visualizar.

Figura 18 – Resultados Odometria Visual KITTI



Fonte: Desenvolvido pelo autor

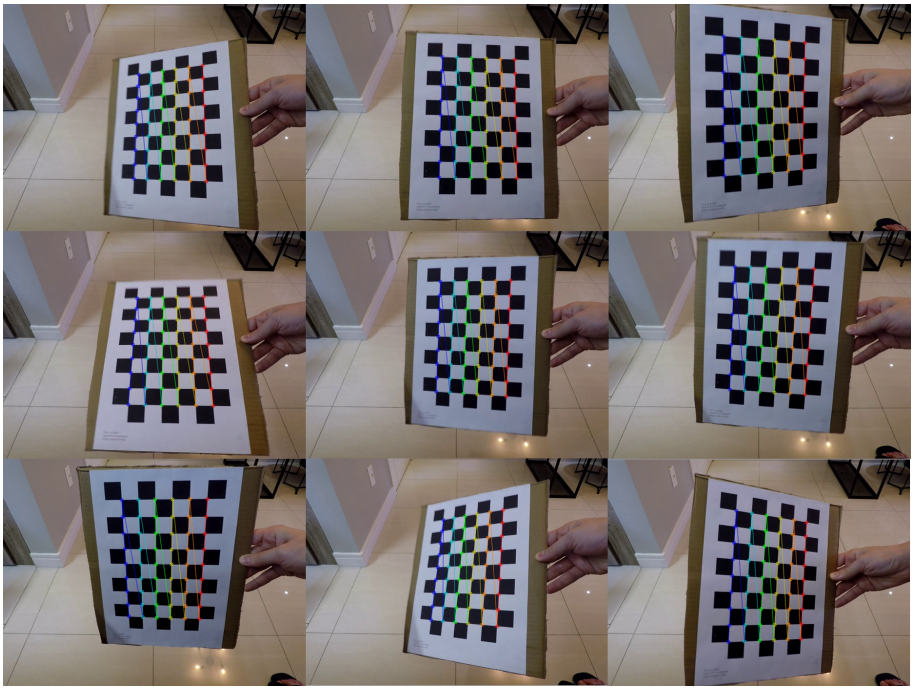
4.6 Odometria Visual com Dados Próprios

Para obtenção de dados experimentais foi utilizado uma câmera *GoPro Hero 5 Black* conforme descrito no capítulo de metodologia. O modo de gravação *linear* foi escolhido pois é o único que retira as distorções geradas pela lente tipo "olho de peixe". Assim, a gravação foi realizada a $30 fps$, *Full HD* (1920 x 1080) pois esta é a menor resolução disponível para este modo de filmagem.

4.6.1 Calibração da Câmera

Para se calcular a matriz R e o vetor t a partir da matriz essencial, é necessário conhecer os parâmetros da câmera. Para isso calibra-se a *GoPro* utilizando um padrão xadrez 9x6 conforme figura 19. Um vídeo de trinta segundos foi obtido com o padrão em movimento em diversos ângulos. Em seguida, para cada segundo do video foi obtida uma imagem com o auxílio do software *VLC* utilizando a configuração *scene filter*. Assim, foi possível utilizar um script (Intel Corporation; Willow Garage; ITSEEZ, 2013) de obtenção dos parâmetros padrão da biblioteca *OpenCV* para extimeção dos parâmetros intrínsicos. Os valores encontrados são distância focal de 702 e centro de imagem (642, 355).

Figura 19 – Imagens Utilizadas para Calibração

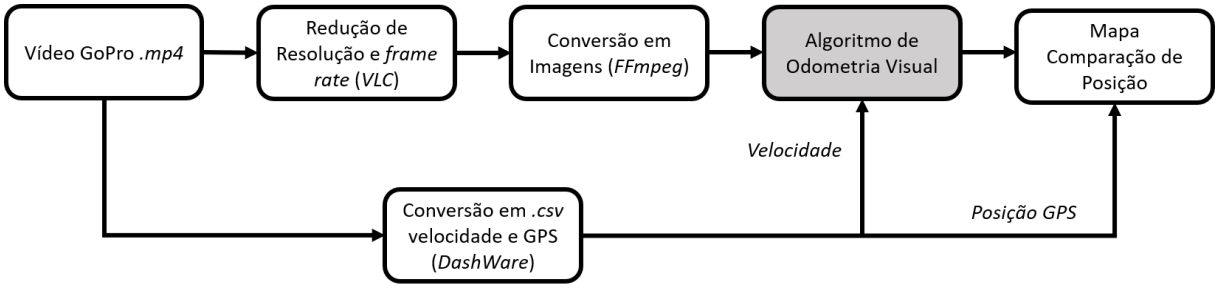


Fonte: Desenvolvido pelo autor

4.6.2 Descrição das Etapas e Procedimentos

Na figura 20, pode-se visualizar um diagrama indicando o fluxo de informações entre as diferentes etapas, dispositivos e softwares. Imagens provenientes da câmera são transformadas, lidas pelo algoritmo de odometria visual e, por fim, compõem um mapa comparativo. O algoritmo de odometria visual tem como entrada uma sequência de imagens e uma informação de velocidade associada. Para se obter a sequência de imagens, primeiramente, o arquivo *.mp4* original é reduzido para resolução 720p (1280x720) com o auxilio do software *VLC* e o número de imagens por segundo é reduzido de 30 fps para 5 fps.

Figura 20 – Esquemas Dados Próprios



Fonte: Desenvolvido pelo autor

Foi realizada uma tentativa de conversão do vídeo em imagens *.png* através da ferramenta *scene filter* do *VLC*. Porém, este apresentou resultados incoerentes, com número de imagens não consistentes com o número de imagens totais do vídeo. Então, para conseguir-se obter a totalidade das imagens de forma confiável, utilizou-se o software *FFmpeg* (FABRICE, 2019) através de linha de comando (versão git-2019-10-26-1054752). Na figura 21, tem-se um exemplo de uma imagem do vídeo gravado juntamente com os pontos identificados neste momento.

Figura 21 – Imagem Típica Dataset Próprio

Fonte: Desenvolvido pelo autor

A obtenção dos dados de velocidade é realizada diretamente do video original com o auxilio do software *DashWare*, que gera um arquivo *.csv* com as informações dos sensores da IMU da câmera. Neste arquivo, tem-se, por exemplo os seguintes campos de dados: Aceleração em três direções; Latitude; Longitude; Elevação; Elevação Média; Velocidade em *Mph* e *Km/h*, entre outros. Estes dados são indicados pela câmera em intervalos de 50ms. Deste arquivo, utiliza-se a informação da velocidade em m/s obtida através de acelerômetros e a posição obtida através de um GPS.

Inicialmente, experimentos foram realizados utilizando diferentes parâmetros testados no dataset KITTI, com o objetivo de ser ter uma intuição do funcionamento do algoritmo nessa nova base de dados adquirida. Logo nos primeiros testes, percebeu-se que em momentos nos quais o veículo parava (semáforos e placas de pare) a velocidade medida pela IMU da câmera se aproximava de zero. Porém, frequentemente haviam valores de velocidade residual (maiores que zero). Também havia movimento residual medido na matriz de rotação, pois apesar do veículo estar em repouso, há movimento na imagem, como veículos, pessoas e vento na vegetação. Dessa forma, ao parar o carro, a matriz de rotação era modificada por sinais aleatórios e o veículo, ao voltar a estar em movimento, partia em uma nova direção sem sentido.

Para correção deste problema, a abordagem escolhida foi implementar um limite mínimo de velocidade no qual o veículo é considerado em movimento, de 0,1m/s. Caso a velocidade do veículo seja inferior ao valor estabelecido, a matriz de rotação não é atualizada. Além disso, é necessário que a informação de velocidade do veículo esteja completamente sincronizada com as imagens. A razão pela qual foi necessário utilizar o software *FFmpeg* (FABRICE, 2019) para obtenção das imagens é que ele garante que todos as imagens do vídeo sejam extraídos, assim pode-se considerar que as imagens do vídeo a *5 fps* estejam aproximadamente uniformemente espaçados temporalmente. Uma primeira tentativa com o software *VLC* foi realizada, porém, este salva as imagens conforme o vídeo é exibido na tela, de forma que múltiplos frames fossem perdidos durante o processo.

Para conversão das coordenadas GPS em coordenadas que possam ser comparadas com os resultados do algoritmo foi necessário convertê-las em metros. Visto que a direção inicial do algoritmo de estimação de movimento é vertical para baixo (referência), independentemente da direção em relação aos pontos cardinais da Terra, é necessário rotacionar as coordenadas dos dados GPS. A transformação das coordenadas para metros foi realizada calculando a distância longitudinal e latitudinal por grau conforme Snyder (1987). Assim, fazendo a diferença em relação ao ponto de partida pode-se saber quantos metros foram percorridos em uma determinada direção.

$$metros\ por\ grau\ longitude = 111.412,84 \cos\left(\frac{latitude * \pi}{360}\right) \tag{4.1}$$

$$\begin{aligned} metros\ por\ grau\ latitude = 111.132,95 - 559,82 \cos\left(2\frac{latitude * \pi}{360}\right) \\ + 1,175 \cos\left(4\frac{latitude * \pi}{360}\right) \end{aligned} \tag{4.2}$$

4.6.3 Definição de Parâmetros para Dados Próprios

Visto que a base de imagens própria apresenta características distintas da base de dados *KITTI*, como por exemplo resolução e razão entre comprimento e largura, é natural pensar que pode haver diferença entre os melhores parâmetros de performance para cada caso. Assim, justifica-se a realização de um estudo para analisar qual combinação de parâmetros é ótima. Para isso, foram variados os mesmos três parâmetros estudados para a solução do *KITTI*: tamanho da janela do algoritmo de *Shi-tomasi*, janela do algoritmo de *Lucas-Kanade* e número de pontos deste último. Na tabela 9 podem ser encontrados alguns dos resultados obtidos.

Tabela 9 – Resultados Experimentais com Dados Próprios

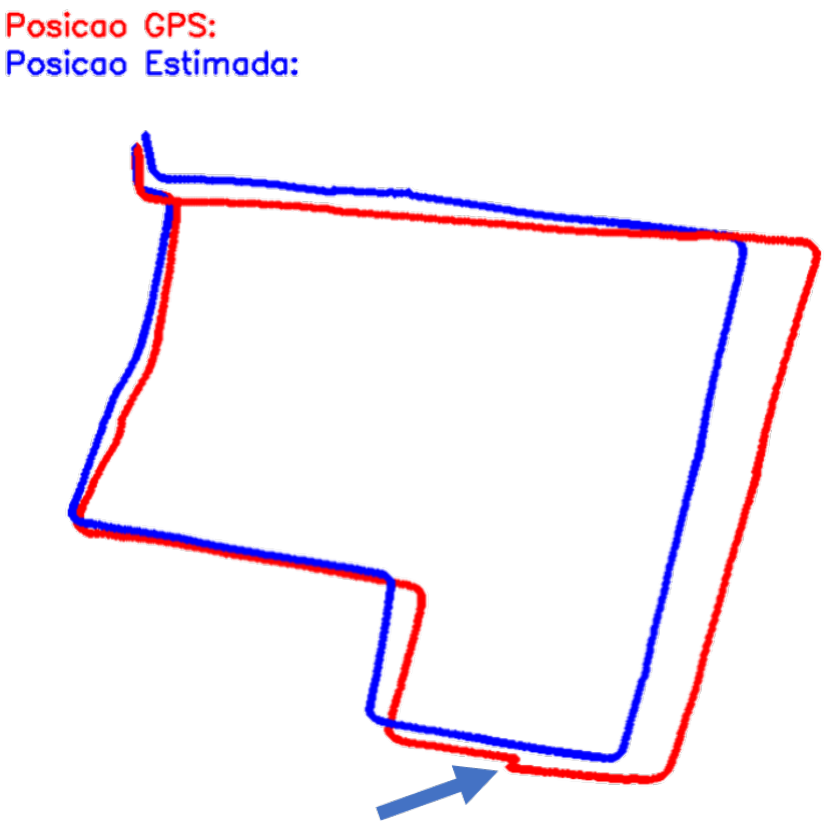
Tempo [s]	Janela ST [px]	Janela LK [px]	Pontos LK	Erro Máximo [m]	Erro Médio [m]
260.41	5	15	2000	75.18	42.10
271.82	3	17	2000	108.56	61.59
276.41	5	17	2000	72.84	42.03
268.51	5	19	2000	88.10	48.94
260.23	5	17	1500	103.83	58.59
264.15	3	19	1500	281.44	47.51
252.01	5	19	1500	143.89	45.97
252.24	3	21	2000	58.07	28.67
258.47	5	21	2000	70.52	42.15

Fonte: Desenvolvido pelo autor

Para visualização do percurso percorrido, uma imagem do algoritmo de odometria visual em comparação à posição GPS é disposto em uma mesma imagem conforme a figura 22. Os parâmetros escolhidos para esta figura foram os que minimizaram o erro médio considerando as possibilidades estudadas na tabela 9, ou seja, janela de Shi-Tomasi 3, janela Lucas-Kanade 21 e número de pontos 2000. O percurso foi percorrido pelo veículo no sentido horário do trajeto da imagem. É relevante observar que o percurso estimado terminou aproximadamente no mesmo ponto de partida, ou seja, não houve um momento específico que produziu um grande erro ângular cumulativo, que pudesse deixar todo o resto do trajeto inconsistente.

Observa-se, principalmente na lateral direita da figura, que a distância percorrida segundo os dados de GPS é maior em relação à posição estimada pelo algoritmo de odometria visual. Este fato poderia ser atribuído em parte a dados de velocidade de baixa qualidade. Os acelerômetros podem ter registrado um dado de velocidade que não corresponde à distância de fato percorrida pelo veículo.

Figura 22 – Comparação Resultados Dataset Próprio



Fonte: Desenvolvido pelo autor

Pode-se perceber na parte inferior da figura, indicada por uma seta, que em certo momento os dados GPS indicam um deslocamento lateral do veículo, o que é inconsistente com a realidade. Esse erro de medida da posição GPS aconteceu quando o veículo estava parado diante de um semáforo. Estacionário durante alguns segundos, o GPS recalculou a sua posição diversas vezes e foi corrigindo o valor encontrado, dando a impressão que o veículo estava em movimento.

Por fim, todas as bibliotecas utilizadas foram instaladas em um micro-computador *Raspberry Pi 3 B+* para verificação do tempo de execução. O algoritmo foi executado corretamente, apresentando resultado identico àquele da figura 22. Assim, sabe-se que as versões disponíveis em diferentes plataformas são compatíveis. Porém, o cálculo do percurso inteiro levou 1404 segundos para ser calculado. O número completo de imagens do percurso é de 1652. Nesta configuração, foi apenas possível atingir 1,17 imagens por segundo, enquanto que a câmera captura e 5 imagens por segundo. Ou seja, não seria possível executar o algoritmo em tempo real com os recursos computacionais definidos.

5 Conclusão

O objetivo deste trabalho foi investigar o funcionamento e implementação de algoritmos de odometria visual monocular para aplicação na estimação de posição e percurso de veículos em movimento. Em um primeiro momento, um estudo teórico da bibliografia foi realizado no qual diferentes etapas do algoritmo foram estudadas individualmente. Foi visto que pode-se dividir o conjunto de algoritmos em quatro etapas: captura de imagens par a par; definição de pontos característicos relevantes (*features*); seguimento destes pontos na imagem seguinte (*feature tracking*); estimação do movimento relativo através da matriz de rotação R e vetor de translação t.

Em um primeiro momento decidiu-se implementar o algoritmo de *Lucas-Kanade* (para seguimento de pontos) com desenvolvimento próprio. Em seguida, comparou-se a velocidade de execução e qualidade dos pontos encontrados com o algoritmo disponível em software livre da biblioteca *OpenCV*. Foi visto então, que a função desta biblioteca possui velocidade da ordem de mil vezes superior pois utiliza processamento paralelo, executa em baixo nível, além de os resultados serem mais exatos. Utilizando imagens da base de dados *KITTI* foi visto que o algoritmo próprio conseguiu seguir 68,6% dos pontos detectados, enquanto o *OpenCV* conseguiu seguir 86,6%. Considerando estas informações, decidiu-se que seria mais apropriado utilizar esta biblioteca.

Para validação da sequência de algoritmos escolhida, utilizou-se a base de dados *KITTI*. Para isso, foram utilizados algoritmos de detecção de cantos (*Shi-Tomasi*), seguimento de pontos (*Lucas-Kanade*), estimação da matriz essencial e matrizes de movimento relativo R e t trazidos da biblioteca *OpenCV*. Devido à natureza monocular do problema e das matrizes homogêneas envolvidas, a matriz de translação precisa ser multiplicada por um escalar a ser determinado, que, nesse caso, corresponde à distância percorrida pelo veículo entre os frames em questão. Como a base de dados *KITTI* não fornece informação de velocidade, esta foi simulada calculando a distância euclidiana percorrida entre cada par de frames através de dados GPS. O algoritmo desenvolvido foi executado para as 11 sequências de imagens disponíveis nesta base de dados. A comparação do percurso percorrido com o estimado foi realizada através de cálculo de erro máximo e erro médio. Pôde-se verificar que a abordagem escolhida apresenta resultados satisfatórios mesmo nos percursos mais longos. Por exemplo, para distâncias de 3,72 km, 1,7 km, 5,06 km e 2,21 km foram encontrados erros médios de respectivamente 7,47m, 7,47m, 10,03m e 6,33m

A principal contribuição deste trabalho foi a validação do algoritmo escolhido com uma aquisição de dados de uma trajetória própria. Estes dados foram obtidos utilizando uma câmera esportiva da marca *GoPro* acoplada em um veículo que trafega em bairro

residencial da cidade de Porto Alegre. A execução do algoritmo nestes dados é interessante pois permite verificar a possibilidade de executá-lo utilizando equipamento de baixo custo. Além disso, os dados obtidos possuem características mais compatíveis com situações reais de tráfego de veículos quando comparada com a base de dados *KITTI*. Os dados obtidos possuem situações diversas não consideradas anteriormente, como carros se movimentando em sentido contrário ou perpendicular e reflexão do sol na câmera. Para tratar o problema da necessidade de informação da velocidade do veículo, foram utilizadas medidas de velocidade obtidas a partir da unidade inercial da câmera. A duração total da aquisição própria foi de 5min 30s e a distância percorrida de 1,846 km. O erro médio obtido neste experimento foi de 28,57 m.

O trajeto escolhido foi percorrido várias vezes com diversas configurações de resolução, razão e frequência de captura. A configuração final escolhida foi vídeo Full HD, modo linear, 30 fps, visto que este modo evita ter que tratar a imagem para remover a distorção da câmera gerada pela lente estilo "olho de peixe". Para calibração da câmera e obtenção dos seus parâmetros intrínsecos foi gravado um vídeo de 30 segundos nos quais 30 imagens com ângulos diferentes foram obtidas. O padrão utilizado para calibração foi o clássico padrão xadrez 9x6. Visto que o algoritmo desenvolvido recebe como entrada imagens, foi necessário converter o vídeo gravado. Para isso, foi utilizado o software livre *VLC Studio* para redução da qualidade do vídeo *.mp4* para *720p* e *5fps* e, em seguida, foi utilizado o software em linha de comando *FFmpeg* para conversão do vídeo em 1542 imagens.

A última etapa do trabalho consistiu em executar o algoritmo desenvolvido em um *Raspberry Pi 3B+*. Apesar deste possuir todas as biblitecas necessárias para execução do código desenvolvido, foi constantado que é apenas capaz de executar aproximadamente 1,17 imagens por segundo, estando bastante abaixo da velocidade necessária. Dessa forma, para que o sistema possa funcionar em tempo real seria necessário utilizar hardware embarcado no carro com maior capacidade computacional ou reduzir a complexidade do algoritmo programando-o em linguagem de mais baixo nível, como por exemplo C. Em trabalhos futuros, caso o objetivo do pesquisador seja execução em tempo real, sugere-se que o algoritmo seja programado em C++ utilizando a biblioteca *OpenCV*, pois esta já é muito otimizada, e que um micro-computador capaz de efetuar cálculos mais rápidos seja escolhido.

A maioria dos esforços foram dedicados procurando por parâmetros que poderiam diminuir o erro médio do percurso. Apesar destes serem importantes e terem diminuído significativamente o erro médio da solução para a base de dados KITTI, agir unicamente sobre estes parâmetros pode não ser suficiente. Ao implementar a solução nos dados próprios, percebeu-se que a base de dados KITTI é particularmente bem condicionada para testes com estes algoritmos pois as dimensões da imagem, orientação da câmera,

resolução da imagem e condições de tráfego são mais adequados e não semelhantes à realidade da aquisição própria. Percebeu-se que, dada as configurações distintas da câmera utilizada neste trabalho em relação à câmera da base KITTI, os pontos detectados se distribuem de maneira diferente, estando mais concentrados principalmente em árvores (pontos ruins pois são concentrados e se mexem devido ao vento), e pouco em objetos facilmente distinguíveis e estáticos no chão. Dessa forma, a importância da localização dos pontos na imagem pode ter sido subestimada e sugere-se que trabalhos futuros se concertrem neste aspecto.

Referências

AQEL, M. O. et al. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, Springer International Publishing, v. 5, n. 1, 2016. ISSN 21931801. Citado na página 12.

BEAGLEBOARD. *BeagleBone Black WebSite*. 2013. Disponível em: <https://beagleboard.org/black>. Citado na página 34.

BOUGUET, J. Pyramidal implementation of the affine lucas kanade feature tracker. *Intel Corporation*, v. 1, n. 2, p. 1–9, 2001. Citado 4 vezes nas páginas 19, 20, 21 e 37.

BOUGUET, J.-Y. *Camera Calibration Toolbox for Matlab*. 2015. Disponível em: <http://www.vision.caltech.edu/bouguetj/>. Citado 2 vezes nas páginas 27 e 28.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Citado 2 vezes nas páginas 13 e 35.

CHENG, Y.; MAIMONE, M. Visual Odometry on the Mars Exploration Rovers. *2005 IEEE International Conference on Systems, Man and Cybernetics*, 2005. Citado na página 12.

DINGFU, Z.; DAI, Y.; LI, H. Reliable scale estimation and correction for monocular Visual Odometry. *IEEE Intelligent Vehicles Symposium, Proceedings*, IEEE, v. 2016-Augus, n. Iv, p. 490–495, 2016. Citado na página 25.

FABRICE, B. *FFmpeg Documentation*. 2019. Disponível em: <https://ffmpeg.org/documentation.html>. Citado 2 vezes nas páginas 54 e 55.

GoPro Inc. *DashWare*. 2017. Disponível em: <http://www.dashware.net/>. Citado na página 35.

HARRIS, C.; STEPHENS, M. A Combined Corner and Edge Detector. p. 147–151, 1988. ISSN 09639292. Citado na página 17.

INC., G. *GoPro Hero 5 Black User Manual*. 2016. Disponível em: <https://gopro.com/content/dam/help/hero5-black/manuals/>. Citado na página 32.

Intel Corporation; Willow Garage; ITSEEZ. *Camera Calibration - OpenCV Documentation*. 2013. Disponível em: <https://opencv-python-tutroals.readthedocs.io/en/latest/py{_}tutorials/py{_}calib3d/py{_}calibration/py{_}cali>. Citado na página 53.

LONGUET-HIGGINS, H. A Computer Algorithm for reconstructing a scene from two projections. *Nature*, v. 293, p. 133–135, 1981. Citado 2 vezes nas páginas 22 e 24.

LUCAS, B. D.; KANADE, T. An iterative image registration technique with and application to stereo vision. In: *DARPA Image Understand Workshop*. Pittsburgh: [s.n.], 1981. p. 121–130. Citado 2 vezes nas páginas 17 e 19.

NISTÉR, D. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 26, n. 6, p. 756–770, 2004. ISSN 01628828. Citado 6 vezes nas páginas 12, 15, 16, 22, 24 e 25.

PEREIRA, F. I. *High Precision Monocular Visual Odometry*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 2018. Citado 2 vezes nas páginas 19 e 26.

PODDAR, S.; KOTTATH, R.; KARAR, V. Evolution of Visual Odometry Techniques. 2018. Disponível em: <http://arxiv.org/abs/1804.11142>. Citado 2 vezes nas páginas 15 e 18.

Python Software Foundation. *Python 3.6 Language Reference*. 2016. Disponível em: <https://docs.python.org/3.6/reference>. Citado na página 13.

Raspberry Pi. *Raspberry PI 3B+ Product Brief*. 2018. Disponível em: <https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>. Citado na página 34.

SCARAMUZZA, D.; FRAUNDORFER, F. Visual odometry: Part I - The First 30 Years and Fundamentals. *IEEE Robotics and Automation Magazine*, v. 18, n. 4, p. 80–92, 2011. ISSN 10709932. Citado 7 vezes nas páginas 14, 15, 17, 22, 24, 25 e 26.

SCARAMUZZA, D.; FRAUNDORFER, F. Visual Odometry: Part II - Matching, Robustness, and Applications. *IEEE Robotics & Automation Magazine*, v. 18, n. 4, p. 80–92, 2011. ISSN 1070-9932. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6096039>. Citado 2 vezes nas páginas 18 e 19.

SHI, J.; TOMASI, C. Good Features to Track. *IEEE Conference on Computer Vision and Pattern Recognition*, 1994. ISSN 13646826. Citado na página 43.

SNYDER, J. P. *Map Projections - A Working Manual*. [S.l.: s.n.], 1987. ISSN 00369225. Citado na página 55.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. [S.l.: s.n.], 2011. 1–25 p. Citado 3 vezes nas páginas 15, 23 e 26.

TOMASI, C. Detection and Tracking of Point Features Technical Report CMU-CS-91-132. *Image Rochester NY*, v. 91, n. April, p. 1–22, 1991. ISSN 00313203. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.5899{&}rep=rep1{&}ty>. Citado 2 vezes nas páginas 17 e 19.

URTASUN, A. G.; LENZ, P.; RAQUEL. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2012. Citado 3 vezes nas páginas 13, 36 e 37.

WALT, S. van der; COLBERT, S. C.; VAROQUAUX, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, v. 13, n. 2, p. 22–30, 2011. Disponível em: <https://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37>. Citado na página 35.

YANG, T. et al. Monocular vision SLAM-based UAV autonomous landing in emergencies and unknown environments. *Electronics (Switzerland)*, v. 7, n. 5, 2018. ISSN 20799292. Citado na página 12.

YU, Y.; PRADALIER, C.; ZONG, G. Appearance-based monocular visual odometry for ground vehicles. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, IEEE, p. 862–867, 2011. Citado na página 15.

APÊNDICE A – Complemento de Dados Experimentais

Tabela 10 – Dados complementares da tabela 3

Tempo [s]	Janela ST [px]	Janela LK [px]	Pontos LK	Erro Máximo [m]	Erro Médio [m]
87.39	3	13	2000	30.77	16.55
78.02	3	13	1500	49.67	24.50
60.73	3	13	1000	26.33	9.70
53.01	3	13	500	59.56	33.81
98.26	5	13	2000	16.12	5.54
80.62	5	13	1500	24.52	6.86
63.52	5	13	1000	15.97	7.97
50.40	5	13	500	33.56	17.58
101.63	10	13	2000	41.98	9.92
89.67	10	13	1500	31.15	7.92
73.79	10	13	1000	19.66	5.65
52.57	10	13	500	12.12	8.04
91.94	10	15	2000	28.77	14.69
89.88	10	15	1500	26.95	13.93
74.40	10	15	1000	26.74	13.96
56.03	10	15	500	28.92	15.07
102.89	10	17	2000	39.37	19.33
101.46	10	17	1500	39.99	20.00
83.44	10	17	1000	70.20	31.60
64.32	10	17	500	22.96	7.76
99.99	10	19	2000	15.46	7.47
90.84	10	19	1500	17.11	7.58
83.98	10	19	1000	11.58	6.85
89.91	10	19	500	22.84	12.27
97.83	10	21	2000	30.54	14.93
117.50	10	21	1500	30.87	15.26
79.19	10	21	1000	23.40	10.34
58.16	10	21	500	38.10	11.32

Fonte: Produzido pelo autor

Tabela 11 – Dados complementares da tabela 6

Percurso	Tempo [s]	Janela ST [px]	Janela LK [px]	Pnts LK	Erro Máx. [m]	Erro Méd. [m]
0	257.98	5	15	2000	24.91	13.28
0	248.24	5	19	2000	38.26	12.73
0	198.45	5	15	1500	52.01	15.60
0	223.91	5	19	1500	32.37	11.90
0	176.44	5	15	1000	529.87	192.51
0	171.78	5	19	1000	524.01	190.87
0	184.09	3	15	1000	50.59	16.12
0	197.70	10	15	1000	30.43	8.58
0	156.64	3	19	1000	530.89	184.42
0	205.21	10	19	1000	38.52	13.11
1	61.59	5	15	2000	135.56	47.70
1	58.91	5	19	2000	143.80	56.67
1	62.18	5	15	1500	297.31	136.92
1	55.70	5	19	1500	124.25	58.93
1	52.35	5	15	1000	372.04	184.06
1	47.87	5	19	1000	334.69	149.46
1	68.00	3	15	2000	244.24	81.99
1	54.23	10	15	2000	279.40	121.39
1	59.31	3	19	2000	257.24	91.55
1	54.17	10	19	2000	129.10	68.86
1	64.88	3	15	1500	252.25	101.17
1	58.96	10	15	1500	285.06	124.91
1	54.23	3	19	1500	184.65	102.45
1	53.33	10	19	1500	128.52	68.08
1	47.61	3	15	1000	615.34	288.05
1	51.75	10	15	1000	184.25	62.41
1	44.22	3	19	1000	370.36	146.66
1	55.50	10	19	1000	183.44	110.66
2	252.55	5	15	2000	52.40	25.63
2	281.06	5	19	2000	31.73	16.12
2	210.48	5	15	1500	34.10	16.62
2	218.51	5	19	1500	41.01	17.62
2	177.84	5	15	1000	63.92	24.98
2	176.55	5	19	1000	28.38	15.98
2	190.85	3	15	1000	52.85	16.72
2	203.85	10	15	1000	30.05	10.53
2	167.26	3	19	1000	120.18	50.53
2	211.25	10	19	1000	30.83	21.49
3	36.83	5	15	2000	2.42	1.29
3	45.06	5	19	2000	4.14	2.51
3	31.85	5	15	1500	7.01	3.36
3	36.03	5	19	1500	5.34	2.96
3	26.30	5	15	1000	8.27	3.64
3	29.12	5	19	1000	7.24	2.95
3	45.85	3	15	2000	4.18	1.78
3	44.56	10	15	2000	4.91	1.52
3	37.18	3	19	2000	5.55	1.91
3	42.51	10	19	2000	13.72	6.20
3	30.96	3	15	1500	4.11	1.97
3	37.32	10	15	1500	10.48	3.74
3	31.46	3	19	1500	13.15	6.06
3	41.12	10	19	1500	11.90	4.78
3	29.51	3	15	1000	3.45	1.87
3	28.65	10	15	1000	16.18	5.67
3	26.01	3	19	1000	7.96	3.97
3	29.54	10	19	1000	5.41	2.14

Fonte: Produzido pelo autor

Tabela 12 – Dados complementares da tabela 6

Percurso	Tempo [s]	Janela ST [px]	Janela LK [px]	Pnts LK	Erro Máx. [m]	Erro Méd. [m]
4	17.29	5	15	2000	6.11	2.56
4	16.11	5	19	2000	2.93	1.25
4	12.73	5	15	1500	5.49	2.72
4	13.23	5	19	1500	1.54	1.19
4	9.59	5	15	1000	7.62	2.98
4	10.88	5	19	1000	3.01	2.05
4	10.47	3	15	1000	1.74	1.15
4	11.95	10	15	1000	9.34	3.87
4	9.48	3	19	1000	5.72	2.29
4	14.41	10	19	1000	2.98	1.37
5	143.23	5	15	2000	26.33	12.24
5	145.56	5	19	2000	45.75	14.56
5	118.49	5	15	1500	52.73	18.04
5	125.88	5	19	1500	69.86	23.58
5	100.70	5	15	1000	30.10	15.30
5	104.86	5	19	1000	50.70	16.03
5	158.93	3	15	2000	63.30	22.93
5	135.72	10	15	2000	25.73	10.96
5	144.44	3	19	2000	30.53	12.77
5	138.27	10	19	2000	39.71	6.33
5	114.85	3	15	1500	40.46	19.00
5	135.76	10	15	1500	29.47	8.22
5	114.64	3	19	1500	21.73	9.55
5	159.00	10	19	1500	33.64	11.75
5	107.04	3	15	1000	23.95	11.34
5	113.37	10	15	1000	28.45	11.92
5	90.42	3	19	1000	42.23	10.51
5	120.32	10	19	1000	34.15	5.71
6	61.60	5	15	2000	24.69	9.32
6	63.14	5	19	2000	22.18	5.78
6	51.61	5	15	1500	32.88	10.30
6	55.35	5	19	1500	32.29	8.48
6	43.46	5	15	1000	52.06	16.89
6	45.14	5	19	1000	17.84	9.18
6	65.64	3	15	2000	64.78	12.85
6	55.59	10	15	2000	35.24	10.10
6	64.91	3	19	2000	39.00	10.48
6	56.36	10	19	2000	7.98	2.77
6	48.23	3	15	1500	39.28	13.22
6	55.90	10	15	1500	35.24	10.10
6	48.31	3	19	1500	30.12	14.79
6	65.31	10	19	1500	7.98	2.77
6	53.95	3	15	1000	36.20	9.28
6	51.57	10	15	1000	26.11	7.93
6	39.72	3	19	1000	17.83	6.11
6	56.95	10	19	1000	18.09	6.59
7	57.86	5	15	2000	41.67	13.28
7	57.61	5	19	2000	41.15	12.21
7	48.86	5	15	1500	39.46	12.84
7	46.36	5	19	1500	34.97	13.90
7	35.49	5	15	1000	44.07	15.00
7	39.25	5	19	1000	33.26	11.39
7	61.76	3	15	2000	34.46	15.16
7	52.29	10	15	2000	46.62	12.91
7	61.13	3	19	2000	39.02	17.29
7	52.76	10	19	2000	50.40	14.58

Fonte: Produzido pelo autor

Tabela 13 – Dados complementares da tabela 6

Percurso	Tempo [s]	Janela ST [px]	Janela LK [px]	Pnts LK	Erro Máx. [m]	Erro Méd. [m]
7	43.75	3	15	1500	44.98	15.79
7	55.45	10	15	1500	47.49	13.23
7	43.91	3	19	1500	29.87	12.70
7	59.95	10	19	1500	48.77	13.78
7	48.94	3	15	1000	30.34	10.33
7	43.96	10	15	1000	45.99	14.39
7	35.04	3	19	1000	58.31	16.44
7	47.57	10	19	1000	47.97	13.87
8	210.30	5	15	2000	56.74	28.76
8	216.58	5	19	2000	70.03	33.46
8	175.55	5	15	1500	83.59	44.08
8	176.37	5	19	1500	142.91	62.10
8	140.45	5	15	1000	73.51	37.23
8	140.14	5	19	1000	131.58	57.94
8	218.68	3	15	2000	102.39	42.10
8	211.51	10	15	2000	110.93	48.33
8	207.85	3	19	2000	66.25	35.12
8	220.07	10	19	2000	112.40	53.07
8	163.08	3	15	1500	140.39	61.17
8	201.99	10	15	1500	82.40	38.48
8	165.40	3	19	1500	122.40	56.59
8	228.68	10	19	1500	110.64	51.44
8	136.85	3	15	1000	136.00	56.34
8	154.43	10	15	1000	141.07	58.70
8	133.57	3	19	1000	94.66	40.13
8	158.66	10	19	1000	90.76	46.90
9	97.20	5	15	2000	35.61	16.34
9	89.98	5	19	2000	15.17	6.09
9	74.27	5	15	1500	5.49	3.34
9	80.78	5	19	1500	24.91	8.07
9	63.88	5	15	1000	8.98	4.46
9	61.12	5	19	1000	11.11	5.69
9	58.47	3	15	1000	15.89	8.31
9	68.78	10	15	1000	26.74	13.96
9	57.35	3	19	1000	30.27	16.14
9	72.21	10	19	1000	11.58	6.85
10	73.93	5	15	2000	31.18	13.36
10	69.74	5	19	2000	58.05	28.46
10	59.38	5	15	1500	37.10	18.24
10	65.47	5	19	1500	25.23	11.59
10	50.34	5	15	1000	27.83	13.20
10	52.55	5	19	1000	18.98	8.69
10	71.87	3	15	2000	30.08	15.05
10	71.94	10	15	2000	36.77	18.10
10	70.47	3	19	2000	36.32	16.10
10	85.96	10	19	2000	33.33	16.30
10	60.24	3	15	1500	21.28	10.25
10	75.78	10	15	1500	31.08	15.36
10	57.13	3	19	1500	28.47	13.55
10	77.15	10	19	1500	32.57	15.84
10	49.90	3	15	1000	20.19	10.16
10	53.88	10	15	1000	39.75	18.54
10	49.74	3	19	1000	45.81	18.72
10	57.74	10	19	1000	45.57	23.01

Fonte: Produzido pelo autor