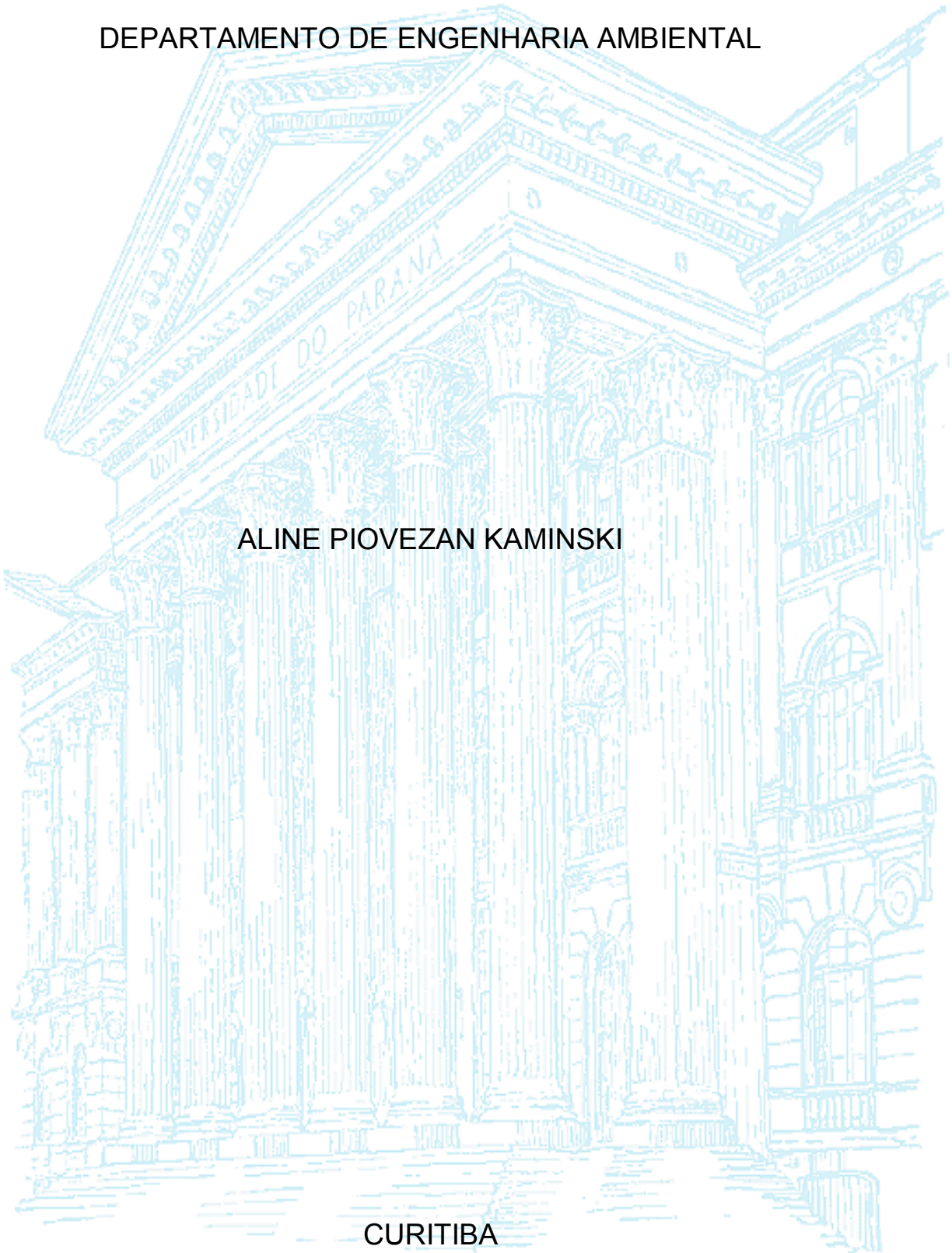
UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA AMBIENTAL



ALINE PIOVEZAN KAMINSKI

CURITIBA
2016

ALINE PIOVEZAN KAMINSKI

EXPERIMENTOS EM MICROCOSMO E ANÁLISE DE VÍDEOS VIA MÉTODOS COMPUTACIONAIS, ESTUDO DE MOVIMENTAÇÕES E INTERAÇÕES ECOLÓGICAS

ENGENHARIA AMBIENTAL

Trabalho de Conclusão de Curso de Graduação em Engenharia Ambiental, Setor de Tecnologia, Universidade Federal do Paraná, como requisito parcial a obtenção do diploma em Bacharel em Engenharia Ambiental.

Orientador: Professor Doutor Emilio Graciliano Ferreira Mercuri

TERMO DE APROVAÇÃO PROJETO FINAL II

ALINE PIOVEZAN KAMISNKI

Trabalho de Conclusão de Curso de Graduação em Engenharia Ambiental, Setor de Tecnologia, Universidade Federal do Paraná, como requisito parcial a obtenção do diploma em Bacharel em Engenharia Ambiental. Avaliado pela banca:

_____
Prof. Dr. Emilio Graciliano Ferreira Mercuri
Orientador - Setor de Tecnologia da Universidade Federal, UFPR.

_____
Prof. Dr. Jean Ricardo Simões Vitule
Setor de Tecnologia da Universidade Federal, UFPR

_____
Prof. Dr. Tobias Bernward Bleninger
Setor de Tecnologia da Universidade Federal, UFPR

_____
M.Sc. Vanessa Maria Ribeiro
Laboratório de Ecologia e Conservação (LEC) da Universidade Federal, UFPR

Curitiba, 20 dezembro de 2016.

RESUMO

Neste estudo foi realizado um experimento em microcosmo, baseado no trabalho de Ribeiro et al (2013), e foram desenvolvidos dois algoritmos para a análise dos vídeos com o enfoque nas interações entre presas e predador. As filmagens foram realizadas em ambiente controlado com cinco presas (*Tilapia Rendalli*) e um predador (*Micropterus salmoides*). A utilização de ferramentais computacionais é amplamente utilizada em diversos campos da engenharia, mas ainda necessita de avanços na área de estudo de ecologia. Foram comparados dois algoritmos distintos com o objetivo de obter a velocidade do predador e sua trajetória ao longo do tempo. Verificando a proporcionalidade entre o primeiro algoritmo e o método de coleta de dados visuais para frames subsequentes do terceiro dia de filmagem obtém R^2 0,0954, e para o segundo algoritmo e o método visual obtém R^2 0,076, mostrando que atualmente o primeiro algoritmo representa melhor a realidade. Porém ainda existem melhorias a serem feitas no algoritmo escolhido, redução dos erros, bem como o estudo de outras frentes do projeto como foco na presa. Ainda é possível a utilização de outras ferramentas computacionais como o *Particle Tracking Velocimetry* – PTV do programa MatLab, e a aplicação do conceito de movimento de Levy.

Palavras-chave: *Tilapia Rendalli*, *Micropterus salmoides,* predador, presa, algoritmo computacional, velocidade.

ABSTRACT

In this study, a microcosm experiment was carried out, based on the work of Ribeiro et al (2013), and two algorithms were developed for the video analysis with the focus on the interactions between prey and predator. Filming was performed in a controlled environment with five preys (Tilapia Rendalli) and one predator (Micropterus salmoides). The use of computacional tools is widely used in several fields of engineering, but still needs advances in the field of ecology. Two distinct algorithms were compared in order to obtain the predator speed and its trajectory over time. Checking the proportionality between the first algorithm and the visual data collection method, for subsequent frames of the third day of filming, obtains R^2 0.0954, and for the second algorithm and the visual method obtains R^2 0.076, showing that the first algorithm currently represents better the reality. However, there are still improvements to be made in the chosen algorithm, reduction of errors, as well as the study of other project fronts, for example focus on prey. It is still possible to use other computational tools such as MatLab's Particle Tracking Velocimetry (PTV) and the Levy Flights.

AGRADECIMENTOS

À meu orientador Emilio Mercuri por me manter motivada e sempre cobrando por resultados melhores, e por estar sempre disponível para responder todas as minhas infinitas duvidas nos programas computacionais.

À aluna de doutorado Vanessa Maria Ribeiro e Gustavo Yamassaki, por me ensinar muito sobre ecologia e experimentos ecológicos.

À minha família, por me ajudar nas horas difíceis e sempre acreditar no meu potencial.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

I – intensidade luminosa

∂ – derivada parcial

d – derivada total

x – coordenada horizontal do sistema cartesiano

y – coordenada vertical do sistema cartesiano

t – tempo

i – vetor unitário na direção x

j – vetor unitário na direção y

∇ - Nabla, operador diferencial de gradiente

Δ - variação

v – velocidade

vel_x, v_x – componente horizontal da velocidade

vel_y, v_y – componente da velocidade na direção vertical

$P^{t+\Delta t}{}_x$ – posição na direção x no tempo posterior

$P^{t+\Delta t}{}_y$ – posição na direção y no tempo posterior

$P^t{}_x$ – posição na direção x no tempo atual

$P^t{}_y$ – posição na direção y no tempo atual

V – módulo da velocidade

$salto$ – módulo do deslocamento

$Erro_{id}$ – erro na identificação

NP_{algo} – número de peixes encontrados pelo algoritmo

$p_{1,x}$ – posição horizontal do peixe no instante posterior

$p_{0,x}$ – posição horizontal do peixe no instante atual

$p_{1,y}$ – posição vertical do peixe no instante seguinte

$p_{0,y}$ – posição vertical do peixe no instante atual

R^2 – coeficiente de correlação linear (Pearson)

V_{MED} – velocidade média

n – quantidade de valores

σ^2 – variância

$\widehat{V_i}$ – velocidade estimada pela função de tendência

DESVQ – desvio quadrático

EQM – erro médio quadrático

$\sum_{i=0}^{n} V_i$ – somatório dos valores de velocidade entre o intervalo $i = 0$ e n

1 INTRODUÇÃO

A visão computacional é uma área de pesquisa que ganhou força após 1970, quando os computadores ganharam velocidade e memória suficiente para processar uma grande quantidade de imagens e dados (Szeliski, 2010). O conceito primordial da visão computacional é o estudo de como capacitar os computadores a interpretar e processar imagens e vídeos da mesma forma que os seres humanos são capazes de perceber o ambiente ao seu redor. Aplicações dessa tecnologia estão sendo desenvolvidas em diversos campos da ciência: como o estudo de ruídos em motores, uso em câmeras de segurança (Rosebrock, 2015), e na detecção de movimentos sutis ao olho humano (Rubinstein, 2014).

No presente projeto, a visão computacional foi aplicada para analisar imagens de vídeos em aquários com duas espécies de peixes (predador: *Micropterus salmoides*, presa: *Tilapia Rendalli)*. Segundo Matai (2012), as técnicas de reconhecimento automático e classificação são benéficas para contagem de peixes para descrever associações entre o animal e seu habitat e para monitorar ecossistemas. Em seu trabalho, Matai et al (2012) determina a localização do peixe no frame da filmagem, o tamanho do indivíduo (comprimento, altura) e por fim diferencia o indivíduo do seu arredor. Outro trabalho nessa área de pesquisa é de Spampinato (2008), o qual utiliza vídeos filmados próximo às águas do banco de corais Ken-Ding em Taiwan.

Como Sih et al (2010) explica em seu artigo, a hipótese da ingenuidade ecológica sugere que quando não houver uma história evolutiva conectando duas espécies, haverá menor chance de reconhecimento do predador. No artigo de Paolucci et al (2013), é comparada a inserção de predadores invasores em ambientes fechados (lagos e rios ou ilhas) ou abertos (oceanos ou continentes). Nesse estudo foi analisada a influência dos dois tipos de ambientes e, contrariando a hipótese da ingenuidade ecológica, não houve influencia da origem biogeográfica das espécies no impacto ecológico. Os dois ambientes apresentaram fragilidade, o que não era de se esperar, já que em sistemas fechados existe menos contato com espécies diferentes, o que em hipótese os tornaria mais frágeis em comparação aos sistemas abertos.

2 CONTEXTUALIZAÇÃO DO TRABALHO

Os grupos de pesquisa do Laboratório de Computação e Tecnologia em Engenharia Ambiental (LACTEA) e do Laboratório de Ecologia e Conservação (LEC) juntaram esforços para o desenvolvimento de um software capaz de auxiliar em experimentos de ecologia, desde contagens de indivíduos até avaliações comportamentais. O objetivo final desse trabalho é contribuir na elaboração desse software, que auxilie nas observações ecológicas em ambientes controlados, deixando essas análises mais eficientes e confiáveis.

Segundo Ribeiro (2013), as atividades humanas são as mais prejudiciais na perda de biodiversidade do planeta. Efeitos do impacto antrópico podem ser observados em mudanças climáticas, poluição das águas e atmosfera, inclusão de espécies invasoras (não nativas), entre vários outros. A biosfera como um todo se encontra em risco, entretanto os ambientes de água doce, por serem fechados possuem maior vulnerabilidade (Paolucci et al, 2013). Em seu estudo, Ribeiro propõe que a modificação ou perda de habitat é a atividade humana que mais deprecia a diversidade biológica.

O predador que foi utilizado em nosso estudo, *M. salmoides*, foi introduzido no Brasil por ser uma espécie de interesse na pesca esportiva. Em sua ocorrência natural, do rio St. Lawrence nos Estados Unidos até o norte do México, os indivíduos chegam até 97 centímetros de comprimento com até 10 quilos (Lasenby, 2000).

No Brasil, esta espécie foi introduzida em 1922 no Estado de Minas Gerais, posteriormente se espalhando pela região sul e sudeste, inclusive estando presente em reservatórios na região de Curitiba (Godoy,1954 *apud* Ribeiro 2013). Ainda o fato do *M. salmoides* ser um predador de topo de cadeia (Maezono e Miyashita 2003; Trumpickas et al. 2011), e sem outras espécies para controlar o número de seus indivíduos, podendo atingir porte superior a predadores nativos, tendo o provável aumento da competição entre eles (Heidinger 1975 *apud* Ribeiro 2013).

International Union for Conservation of Nature - IUCN (União internacional de conservação da natureza) nos anos 2000 publicou uma lista com as cem piores espécies mais invasoras do mundo, e *M. salmoides* encontra-se nela. Aumentando

ainda mais a necessidades de estudos mais aprofundados sobre essa espécie, e mais uma inspiração para o presente trabalho.

3 OBJETIVOS GERAIS

Desenvolver um experimento em microcosmo da interação entre predador (*M. salmoides*) e a presa (*T. Rendali*). Testar e aprimorar um software livre para análises de imagens ecológicas usando como modelo experimentos para testar comportamento e ingenuidade ecológica

3.1 Objetivos específicos

a) Aperfeiçoar um desenho amostral para facilitar a visualização padronizada das imagens e comportamentos;

b) Realizar experimentos com presas e predadores;

c) Desenvolver um código para estudo da movimentação de peixes em vídeos;

d) Detectar por meio do programa computacional a presa e o predador;

e) Caracterizar o padrão de movimentação da presa e do predador;

f) Identificar posições (x, y) de cada indivíduo (peixe) no aquário;

g) Identificar velocidades de cada peixe no aquário;

h) Mostrar a eficiência do algoritmo para a detecção dos animais;

i) Validar a metodologia por comparação visual;

j) Validar a metodologia com uso de análises estatísticas.

4 REVISÃO BIBLIOGRÁFICA

O trabalho de Matai (2012) trata sobre o uso de programas computacionais para reconhecimento de peixes. Segundo ele, com esse processo automatizado, é possível fazer a contagem de peixes, bem como de outros organismos com uma maior rapidez, podendo aumentar as possibilidades de identificação de associação entre habitats e seres vivos, ou também o monitoramento de ecossistemas. Em seu estudo, Matai divide a automação computacional em duas partes, a primeira sendo a diferenciação do indivíduo do restante do ecossistema, e a segunda seria a avaliação da imagem pelo programa computacional. Matai apresenta ainda algumas metodologias para detecção de peixes, e os seus processos consistem na identificação do peixe no frame estudado (posição x, y) e completa diferenciação com o fundo (ecossistema). A primeira metodologia estudada para detecção de peixes, utilizando duas espécies *Chaetodontidae* e *Sebastes rubrivinctus*, foi a análise de filmagem com peixes (positiva) e sem peixes (negativa) para a validação do algoritmo (Viola, 2004). A segunda metodologia foi utilizada para reconhecimento das espécies de peixes, usando *Principal Component Analysis (*Análise de Componentes Principais - ACP) e *Scale Invariant Feature Transform* (Transformação de escala com característica invariável - SIFT). A ACP é uma metodologia que compara imagens conhecidas com o alvo não conhecido. Já a transformada SIFT compara duas imagens, e encontra pontos em comum entre elas. A partir da análise desses algoritmos, os pesquisadores observaram uma grande variância na qualidade das respostas obtidas, e concluíram que algumas espécies geram melhores resultados que outras.

Spampinato (2008) estudou a detecção, o caminho e a contagem de peixes a partir de vídeos obtidos nas águas subtropicais do recife de coral perto de Ken-Ding, Taiwan. Como os vídeos desse trabalho foram feitos em águas abertas, ou seja, no ecossistema do recife de coral, tiveram que ser feitas algumas modificações em comparação com trabalhos feitos em ambientes controlados. Seu sistema de processamento de vídeo consiste em análise da textura do vídeo, detecção do peixe e por último, o acompanhamento do caminho percorrido. A detecção do peixe é baseada na combinação de dois algoritmos independentes. O caminho foi estudado a partir do

algoritmo de combinação de cores chamado *CamShift,* o qual gera médias da distribuição pela iteração da direção com maior crescimento. No estudo da textura da imagem, Spampinato avalia o brilho, a suavidade e a coloração. O brilho e a suavidade são obtidos a partir da escala de cinza presente no histograma dessa imagem. A coloração é classificada como verde ou não verde, a partir da contagem de pixels no frame que ultrapassam 128. A próxima etapa de detecção do peixe foi descrita a partir do *Adaptive Gaussian Mixture Model*, com isso a detecção de peixes com rápida e/ou baixa movimentação foi satisfatória. A última etapa, a de rastreamento, é baseada na utilização de dois algoritmos, um para comparar formas, e outro para comparações no histograma. Para finalizar o estudo são realizadas contabilizações de peixes (área detectada como sendo peixe), sendo que as que têm menos de três frames de duração são ignoradas, consideradas erros. Para realizar o teste da metodologia proposta, Spampinato analisou 20 vídeos diferentes, com aproximadamente 1 minuto de duração cada (300 frames).

Palucci (2013) estudou as interações entre espécies nativas e invasoras, bem como sua interação com o ecossistema. Segundo seu estudo, a distribuição de efeitos de interação ecológica positivas, negativa e neutra em presas nativas, é bastante diferente se levado em consideração a origem do consumidor. Ao estudar consumidores invasores, os efeitos negativos são maiores, e os positivos menores do que o esperado. Palucci corrobora a hipótese de que a ingenuidade ecológica da biota nativa facilita sua supressão por predadores invasores em comparação com inimigos nativos. Um resultado não esperado obtido pelo estudo foi que não houve diferença de impacto de consumidores invasores em sistema insular ou sistemas abertos.

Os alimentos não estão distribuídos igualmente ao redor do globo, então os animais precisam sair à procura. E os que possuírem a melhor estratégia de forrageamento, terão maior sucesso em sua busca. E para a análise da movimentação do predador em busca de alimento (hipótese de forrageamento) um conceito muito utilizado é o de *Levy Flights*, “movimento” de Levy, um conceito matemático que se baseia nos saltos entre o ponto de início e chegada (Buchanan, 2008).

Esse conceito pode ser aplicado em difusões, turbulência, etc. No âmbito de forrageamento animal, quanto maior o voo ou pulo, maior o espaço percorrido pelo

animal afirma Buchanan, e maior a sua chance de encontrar alimento. Ao estudarmos os deslocamentos locais, pequenas distâncias, os mesmos parecem aleatórios, mas ao estudar maiores distâncias observamos a presença do conceito de movimento de Levy.

Um experimento realizado com um total de 168 análises de desempenho para 75 espécies de peixes, com o objetivo de caracterizar a natação dos peixes com padrões de movimentação, idade e tamanho dos peixes, e prever impacto nos peixes a partir de atividades humanas alterando a hidrodinâmica do curso d´água. O desempenho de natação dos peixes está ligada a sua velocidade e tempo até a fadiga. O autor classifica a natação em três classes. A *Sustained swimming* (natação sustentada) é caracterizada pela velocidade mantida por mais de 200 minutos, sem chegar a fadiga. A *Prolonged swimming* (natação prolongada) pode ser mantida entre 20 segundos e 200 minutos e cessa com fadiga. E por último*, Burst speed* (velocidade de estouro), a qual é caracterizada pela velocidade mais elevada e pode ser mantida por até 20 segundos. E outra característica importante para o estudo da performance de natação dos peixes é a quantidade de dependências físicas e biológicas. Ainda o formato do corpo, função muscular, entre outras. A velocidade absoluta de natação aumenta com o tamanho do peixe, e o seu desempenho é dependente da temperatura da água (Wolter, 2003).

A motivação para movimentação é dividida em três categorias, a alimentação, reprodução ou fugir de predadores. Ainda dependendo de vários fatores, incluindo clima, temperatura, concentração de ferormônios, densidade de outro organismo no mesmo local, entre outros. Esses fatores podem modificar a velocidade e/ou trajetória, mas não o porquê do inicio da movimentação. Caracterizando os encontros biológicos como um processo de reação difusiva, transporte, ou reativa (interação), e alimentação e reprodução não representam comportamentos lineares, pois duas presas não trazem o dobro de benefícios se comparado a uma (Gandhimohan, 2011).

A dificuldade de análises ecológicas é comum entre muitos pesquisadores dessa área, e vários estão estudando maneiras de automatizá-las, para que seja possível um estudo mais rápido, com mais variáveis e uma maior confiabilidade nos dados, como mostrado pelos inúmeros trabalhos citados anteriormente. O uso de um programa computacional é aceito e usado por muitos, mas ainda é complexo para o seu uso

generalizado. Com isso o objetivo do nosso trabalho, como já dito, é de testar um código simples para essas análises em ambientes controlados, como aquários, por exemplo.

5 MATERIAIS E MÉTODOS

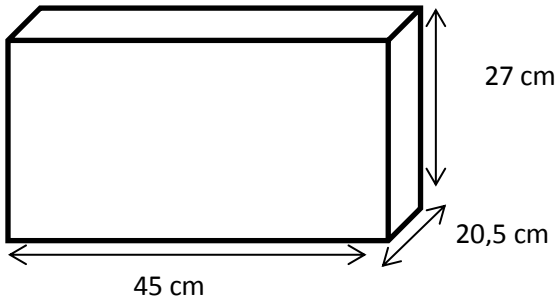
5.1 Modelo experimental

Iniciamos com a construção do modelo experimental e realização de vários testes com o aquário, para que pudéssemos verificar todas as variáveis intrínsecas do nosso problema. O arranjo final utilizado nas filmagens foram aquário revestido internamente com EVA (Etil Vinil Acetato) preto - para redução de reflexo dos peixes - e isolamento deste aquário do restante do laboratório com caixa coberta com cartolina preta com um furo na frente para o encaixe da máquina fotográfica (para redução de reflexo da câmera), e com bomba de oxigênio ligada antes do início das filmagens, como representado nas Figuras 1 e 7.

Figura 1: Aquário revestido com EVA preto

FONTE: O autor 2016

Figura 2: Esquema comprimento x altura x profundidade do aquário utilizado nas filmagens



FONTE: O autor 2016

Após os testes para encontrar o arranjo que menos interferisse na filmagem, iniciamos a segunda fase, a filmagem propriamente dita. Para as filmagens utilizamos uma Canon EOS 500D com lente de 18 a 55 milímetros. Esta fase consiste em 3 etapas: aclimatação dos espécimes (mostrado nas Figuras 3 e 4), filmagem das presas e por último, inclusão do predador e segunda parte da filmagem. A partir desses vídeos estudamos com o auxílio do programa computacional de detecção de imagens a movimentação dos indivíduos.

A realização da aclimatação é para diminuir o stress dos indivíduos e para captar em câmera seu estado normal, ou mais próximo do seu estado natural possível. Inicialmente eles foram colocados em baldes com água de seu aquário anterior, para ir se misturando aos poucos com a água do novo aquário. Isso é feito para que não ocorra um choque de mudança de temperatura. Depois de alguns minutos eles foram retirados do balde para dentro do aquário, os quais ficaram por aproximadamente cinco horas, aclimatação propriamente dita (Mendes, 1983).

Figura 3: primeira fase aclimatação para os indivíduos antes da filmagem

FONTE: O autor 2016

Figura 4: segunda e última fase de aclimatação antes da filmagem

FONTE: O autor 2016

As espécies utilizadas no experimento foram o *Micropterus salmoides* (Nome popular: Black bass), de originário da América do Norte, como o predador, e *Tilapia rendalli* (Nome Popular: Tilápia), de origem Africana, como presa. A alimentação das presas consistiu em ração para peixe (despejada a cada 48 horas), e para o predador um indivíduo *T. Rendalli* pequena a cada 48 horas.

Figura 5: *T. Rendalli*

Figura 6: *M. salmoides*

FONTE: O autor 2016

Todos os espécimes utilizados no experimento foram coletados no reservatório do Passaúna com tarrafas de malha 2,5 (entre nós). As coletas dos exemplares foram feitas em um único dia, entre os meses de março e abril de 2016, não sendo animais

de cultivos.

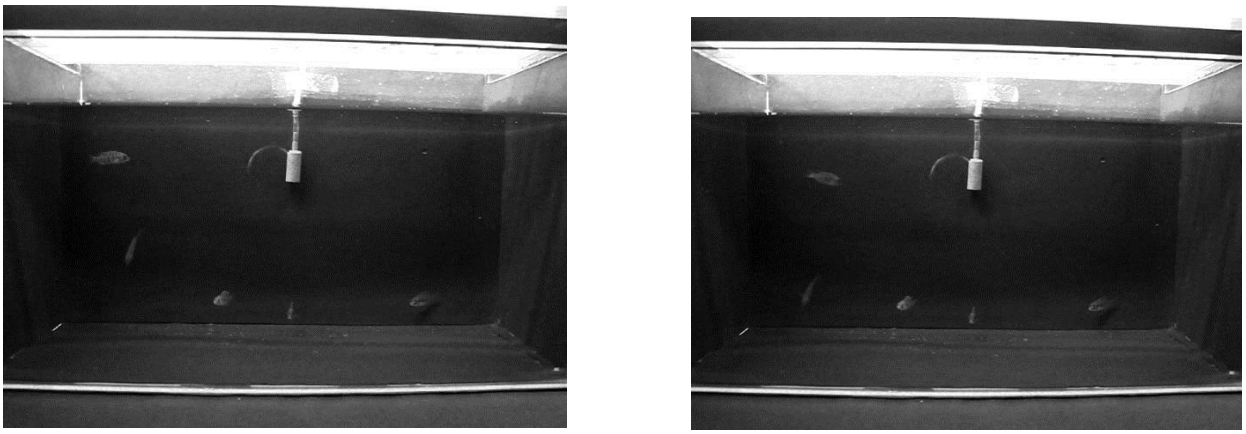
Figura 7: montagem para filmagem (sem cartolina preta na caixa)

FONTE: O autor 2016

Para iniciar o processo de filmagem, primeiro foi necessário deixar as presas para aclimatação por aproximadamente cinco horas (Figura 5), no aquário em que foi feita a filmagem. Após esse período de tempo os indivíduos foram filmados por 15 minutos, e logo em seguida foi inserido o predador no aquário e filmado por mais 15 minutos. Esse processo foi realizado três vezes, com indivíduos diferentes para evitar diferenciação individual, e observar padrões das espécies.

A Figura 8 abaixo é uma foto de uma das filmagens com as cinco presas no aquário, para ilustrar o experimento.

Figura 8: foto em preto e branco das cinco presas (dois frames subsequentes)



FONTE: O autor 2016

A tabela descrita abaixo apresenta o comprimento padrão das presas e dos predadores. O comprimento padrão é a distância do focinho até o final da coluna vertebral (ou início da nadadeira caudal) do peixe.

Tabela 1: comprimento médio *T. Rendalli* usados nos experimentos.

Presas – *T. Rendalli*					
Nº réplica	**Comprimento padrão (cm)**	**Nº réplica**	**Comprimento padrão (cm)**	**Nº réplica**	**Comprimento padrão (cm)**
1	2,8	2	2,8	3	2,8
1	3,0	2	2,8	3	3,0
1	3,0	2	2,9	3	3,1
1	3,1	2	3,0	3	3,2
1	3,1	2	3,2	3	3,2
Média 3,0					

FONTE: O autor 2016

Tabela 2: comprimento médio *M. salmoides* usados nos experimentos.

Predador – *M. salmoides*	
Nº réplica	**Comprimento padrão (cm)**
1	6,9
2	7,0
3	6,7
Média 6,9	

FONTE: O autor 2016

A partir dessa tabela, observa-se que o desvio padrão é pequeno, não havendo uma grande diferenciação no comprimento dos indivíduos utilizados no experimento.

A análise dos dados, vídeos, obtidos a partir das filmagens (três réplicas de 30 minutos cada) foram feitas por duas ferramentas distintas: o algoritmo computacional desenvolvido e modificado a partir de um código de detecção de ocupação de câmera de segurança (Rosebrock, 2015); e o outro, o algoritmo desenvolvido por Bruce D. Lucas e Takeo Kanade, chamado de Lucas Kanade (*OpenCV: Optical Flow*). Porém, antes da análise dos dados feitos pelos dois programas computacionais, é necessário um pré-processamento dos vídeos obtidos.

5.2 Pré-processamento das imagens:

Para a formação da imagem nula, ou seja, sem peixes presentes no aquário, para ser usada como primeiro frame, é necessário obter fragmentos dos vídeos. Essa conversão dos vídeos com presas e predador (dia 3, 4, e 5) para sequência de imagens é realizada a cada 15 frames, com o programa em Python gerar_imagens.py (Anexo 3) e suas rotinas common.py e common.pyc.

Para a formação do fundo do aquário sem peixes utilizamos o programa *Gimp*. Por fim cortamos os vídeos modificados (com fundo sem peixes como primeiro frame) com o aplicativo *Lightworks*, retirando partes desnecessárias e enquadrando o vídeo, vide figura explicativa abaixo. Posteriormente o vídeo é processado com os programas em Python (Anexos 1 e 2).

Figura 9: representação de um frame do vídeo após trabalho no *Lightworks.*

FONTE: O autor 2016

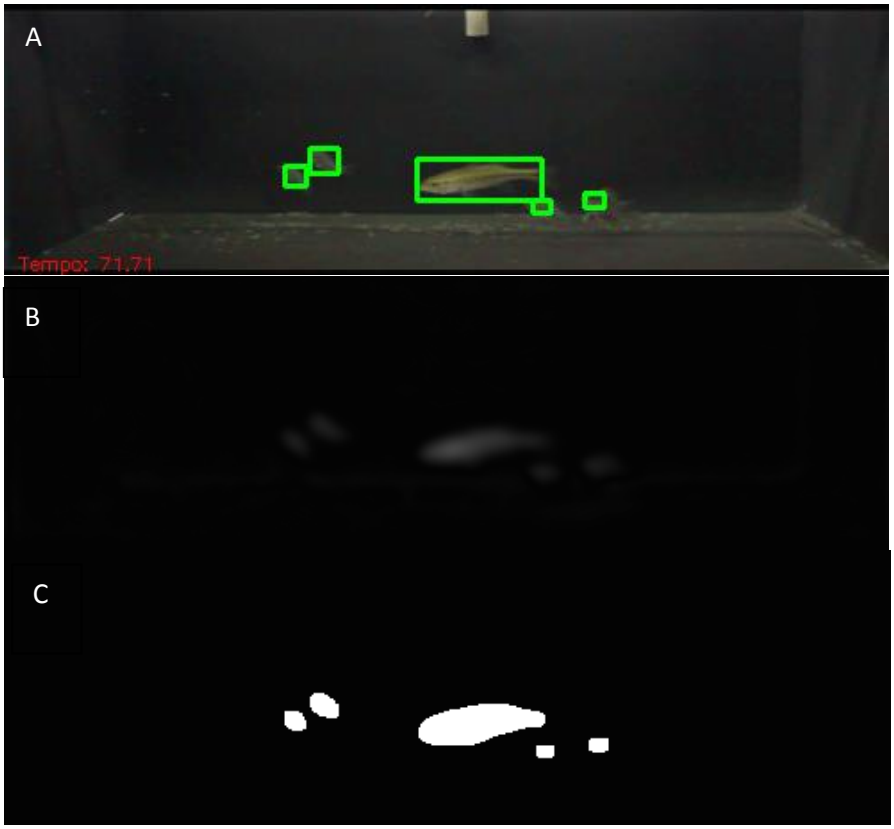
5.3 Algoritmo I (*Motion detector* modificado)

O algoritmo computacional desenvolvido por Rosebrock (2015) tem como função a detecção de ocupação (distribuição em uma dada área) em vídeos de câmeras de segurança de edifícios, gravados ou ao vivo. O código está disponível em seu site pyimagesearch na linguagem computacional Python.

Inicialmente Rosebrock utiliza as seguintes bibliotecas do Python no seu código: argparse, datetime, imutills, time, e cv2. No próximo passo ele constrói o argumento e o caminho para percorrer no computador até encontrar o vídeo requerido. O primeiro frame da filmagem é "Nulo", ou seja, supõe-se que o primeiro instante da filmagem é o local sem nenhum indivíduo dentro ("*Unoccupied*"). No nosso caso, é o aquário sem peixes, como mostrada na Figura 10 abaixo.

Em seguida o vídeo é convertido para um tamanho fixo de 500 pixels e para uma escala de cinza, exemplo na figura central abaixo. Outro vídeo é criado modificando o contorno dos objetos (*Gaussian Blur*) para ter uma região média de pixels de 11 por 11.

Figura 10: Imagem retirada da análise com o programa computacional. A - retângulo de reconhecimento. B- escala de cinza, e C - imagem com *Gaussian Blur.*



FONTE: O autor 2016

O primeiro frame já caracterizado e convertido para os tipos de imagens necessários, o computador pode compará-lo com os frames seguintes. Essa comparação é feita com base no primeiro frame (fundo do aquário sem peixes), modificando o status do ambiente para ocupado ou desocupado.

Os resultados obtidos pelo programa apresentam-se em formato de texto, com o frame, a contagem dos peixes, a posição x e y de cada um deles, área do retângulo encontrado, velocidades na direção horizontal e vertical, e por fim o módulo da

velocidade. As velocidades são obtidas a partir do dado de deslocamento, não sendo uma obtenção direta.

O cálculo das velocidades está apresentado abaixo, e como o aquário possui 0,45 metros de largura e a imagem possui 500 pixels, podemos converter a velocidade de pixels por frame para metros por segundo. Como a cada segundo obtemos 30 frames (30 *frames per second*), utilizamos a variação do tempo como sendo 1 frame ou 1/30 segundos.

$$v = \frac{\Delta x}{\Delta t} \tag{1}$$

Velocidade na direção x:

$$vel_x = \left(P^{t+\Delta t}{}_x - P^t{}_x\right)30\left(\frac{0{,}45}{500}\right) \tag{2}$$

$$vel_x = \left(P^{t+\Delta t}{}_x - P^t{}_x\right)0{,}027 \tag{3}$$

sendo que $\mathrm{P^{t+\Delta t}_i}$ representa a posição do peixe no instante seguinte $(\mathrm{t} + \Delta\mathrm{t})$ e $\mathrm{P^t_i}$ a posição no instante atual na direção x ou y.

A velocidade na direção y é descrita pelas equações:

$$vel_y = \frac{\left(P^{t+\Delta t}{}_y - P^t{}_y\right)}{\frac{500}{0{,}45}}30 \tag{4}$$

$$vel_y = \left(P^{t+\Delta t}{}_y - P^t{}_y\right)0{,}027 \tag{5}$$

Ao fazermos o módulo das velocidades nas direções x e y, obtemos:

$$V = \sqrt{vel_x{}^2 + vel_y{}^2} \tag{6}$$

E assim como calculamos o módulo da velocidade, podemos calcular o deslocamento do predador. O multiplicador $\frac{0{,}45}{500}$ também é utilizado nessa conta, para transformar de pixel para metros.

Deslocamento na direção x:

$$\Delta x = \left(P^{t+\Delta t}{}_x - P^t{}_x\right)\frac{0{,}45}{500} \quad (7)$$

Deslocamento na direção y:

$$\Delta y = \left(P^{t+\Delta t}{}_y - P^t{}_y\right)\frac{0{,}45}{500} \quad (8)$$

Módulo do deslocamento:

$$salto = \sqrt{(\Delta x^2 + \Delta y^2)} \quad (9)$$

Outra variável que podemos estudar é a do erro da contagem do predador no aquário obtida pelo algoritmo (*Motion detector*), se comparado com a realidade da quantidade de peixes, utilizando a formula:

$$Erro_{id} = |1 - NP_{algo}| \quad (10)$$

sendo que $Erro_{id}$ representa o erro em módulo na identificação do número de peixes pelo algoritmo, NP_{algo} o número de peixes identificado pelo algoritmo, e 1 a quantidade de predadores que estão no aquário.

A partir de todos esses resultados é possível criar gráficos de ocupação do predador no aquário no tempo estudado, usando o programa código-figura-predador-v01.py (Anexo 4), e de velocidade por tempo. Após a obtenção desses dados, o próximo passo é a análise estatística da eficiência do método (para testes do programa computacional).

5.4 Algoritmo II (Lucas-Kanade)

No segundo algoritmo utilizado, o frame subsequente é comparado com o frame anterior, não necessariamente sendo o primeiro. A avaliação das imagens, como é explicado na página de internet do *OpenCV* (*Open Source Computer Vision*), é feita a partir da equação do fluxo óptico, o qual compara o padrão de movimentação do objeto em questão, em frames sequenciais.

A utilização da equação do fluxo óptico é baseada em expansões locais em séries de Taylor a partir de um sinal da imagem. Algumas hipóteses simplificadoras são aplicadas nas derivadas: a superfície do objeto filmado é plano, iluminação incidente uniforme ($\frac{\mathrm{dI}}{\mathrm{dt}} = 0$) e sua reflectância praticamente não varia e não apresenta descontinuidades, portanto é diferenciável.

A equação da variação total do brilho:

$$dI = \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{11}$$

sendo que $\mathrm{I}(\mathrm{x}, \mathrm{y}, \mathrm{t})$ representando a intensidade luminosa.

Dividindo a equação anterior (11) por dt e utilizando a relação $\frac{\mathrm{dI}}{\mathrm{dt}} = 0$ obtemos:

$$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \tag{12}$$

A equação do fluxo óptico:

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = \left(\frac{\partial I}{\partial x}i + \frac{\partial I}{\partial y}j\right).\left(\frac{\partial x}{\partial t}i + \frac{\partial y}{\partial t}j\right) = 0 \tag{13}$$

E em sua versão compacta:

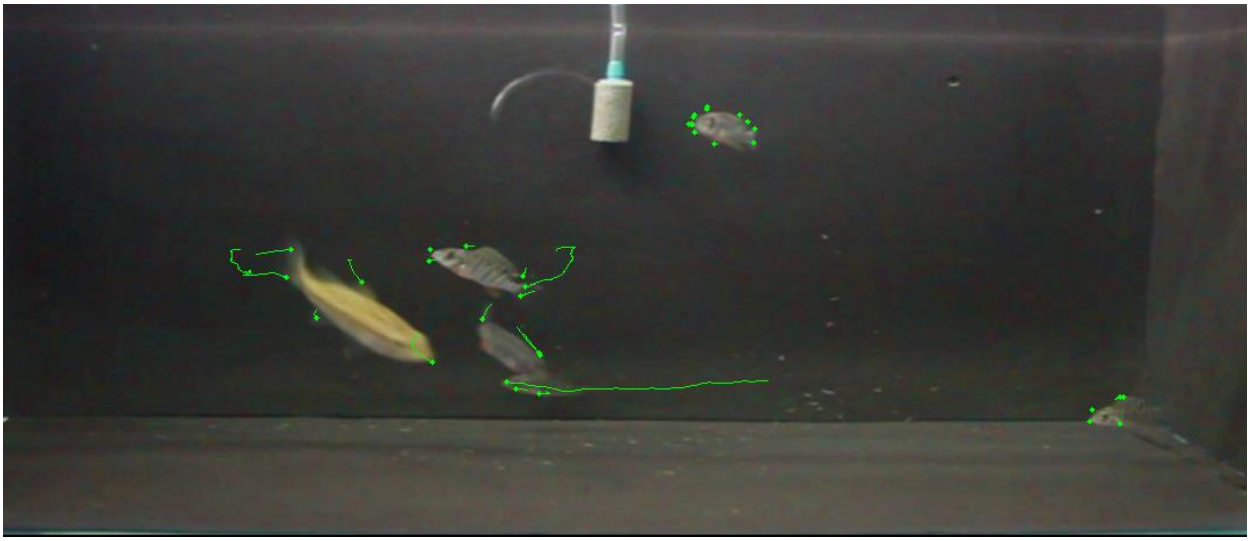
$$\nabla I.v + \frac{dI}{dt} = 0 \tag{14}$$

Para resolver a equação (13) é necessário aplicá-la em uma vizinhança da imagem. Se calcularmos em uma região de 5x5, obtemos 15 equações e somente duas

variáveis (velocidade na direção x e y), e resolvemos pelo método dos mínimos quadrados, utilizando a transposta da matriz.

Com o resultado do programa obtemos um arquivo em texto com: as posições nos dois instantes (atual e instante seguinte), frame, contagem, erro, velocidades na direção x e y, e módulo da velocidade. E em tempo real do andamento do programa computacional, em tela, vemos o filme com os pontos detectados nos espécimes, bem como sua trajetória (aparecendo por um tempo predeterminado), como mostrado na Figura 11 a seguir.

Figura 11: imagem retirada da análise do programa computacional II



FONTE: O autor 2016

Código original do método II está disponível online, e o código modificado para necessidade do projeto no Anexo 2. O cálculo das velocidades está apresentado abaixo, e como explicado anteriormente utilizamos as conversões de $\frac{1}{30}$ segundos e $\frac{0,45}{500}$ metros.

$$v = \frac{\Delta x}{\Delta t} \qquad (15)$$

Velocidade na direção x:

$$v_x = (p_{1,x} - p_{0,x})30\left(\frac{0{,}45}{500}\right) \tag{16}$$

$$v_x = (p_{1,x} - p_{0,x})0{,}027 \tag{17}$$

sendo que $p_{1,x}$ representa a posição x do peixe no instante seguinte e $p_{0,x}$ a posição no instante atual na direção x

A velocidade na direção y é descrita pelas equações:

$$v_y = \frac{(p_{1,y} - p_{0,y})}{\frac{500}{0{,}45}}30 \tag{18}$$

$$v_y(p_{1,y} - p_{0,y})0{,}027 \tag{19}$$

sendo que $p_{1,y}$ representa a posição y do peixe no instante seguinte e $p_{0,y}$ a posição no instante atual na direção y.

Ao calcularmos o módulo das velocidades nas direções x e y, obtemos:

$$V = \sqrt{{v_x}^2 + {v_y}^2} \tag{20}$$

5.5 Métodos estatísticos

Com os dados obtidos pelos dois métodos computacionais, previamente explicados, devemos utilizar métricas para o estudo dos mesmos. Dentre vários disponíveis, foi utilizado a velocidade média, variância, coeficiente de determinação –

R^2, M*ean square error* - MSE (erro médio quadrático - EQM) e o desvio quadrático que estão descritos abaixo.

Cálculo da velocidade média dos dados de velocidade:

$$V_{MED} = \frac{1}{n}\sum_{i=1}^{n} V_i \tag{21}$$

sendo que V_i representa a velocidade obtida para cada frame do vídeo, e n o tamanho da amostra.

Cálculo da variância dos dados de velocidade:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (V_i - V_{MED})^2 \tag{22}$$

Cálculo do coeficiente de determinação:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(V_i - \hat{V}_i\right)^2}{\sum_{i=1}^{n} (V_i - V_{MED})^2} \tag{23}$$

endo que $\hat{V}_i$ representa é o valor estimado e V_i o valor observado.

Cálculo do EQM dos dados de velocidade:

$$EQM = \frac{1}{n}\sum_{i=1}^{n} \left(\hat{V}_i - V_i\right)^2 \tag{24}$$

Cálculo do desvio quadrático:

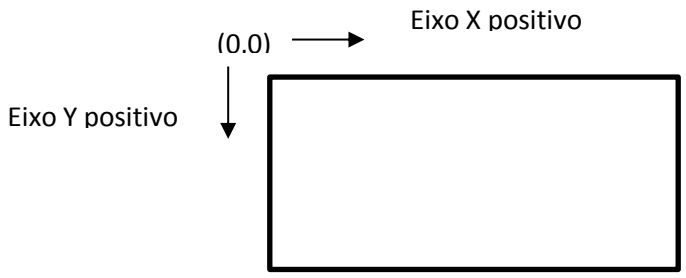
$$DESVQ = \sum_{i=1}^{n} (V_i - V_{MED})^2 \tag{25}$$

6 RESULTADOS

6.1 Algoritmo I (*Motion detector* modificado)

Após o processamento dos vídeos pelo programa em Python, anteriormente explicado (disponível no Anexo 1), obtemos como saída um arquivo de texto com: tempo, frame, contagem dos peixes, posição x e y do retângulo (mostrado na Figura 12) de contorno do peixe, área desse retângulo, velocidades nas direções x e y e por último o módulo das velocidades.

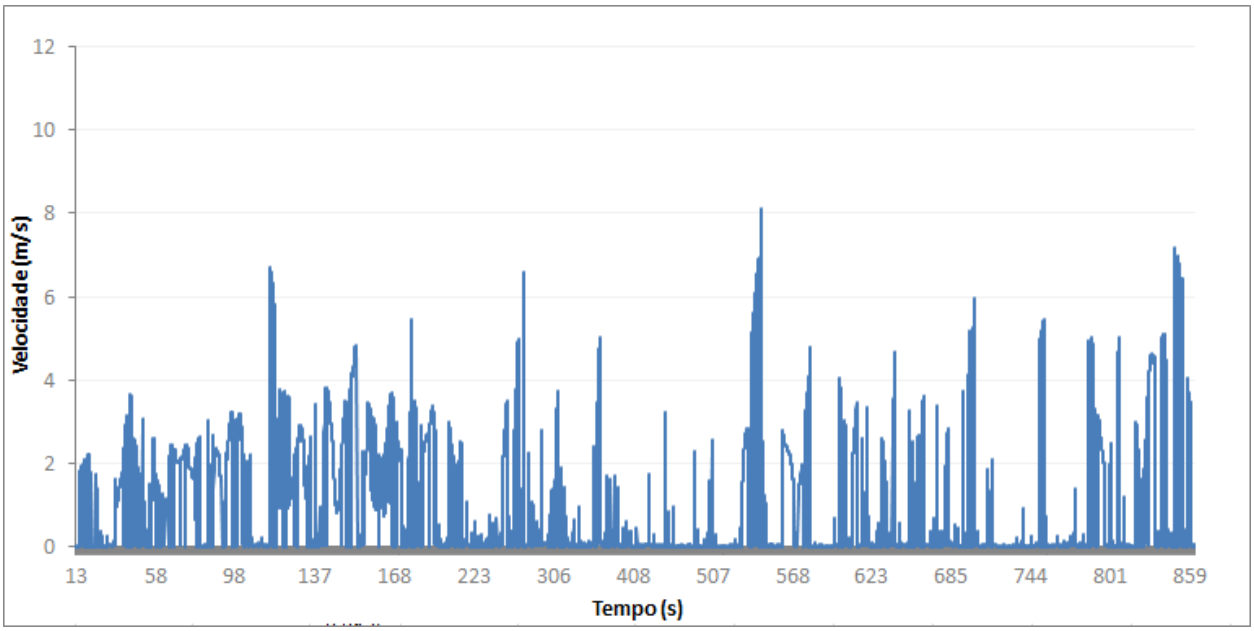
Figura 12: esquema da direção dos eixos escolhidos no aquário visto de frente.



FONTE: O autor 2016

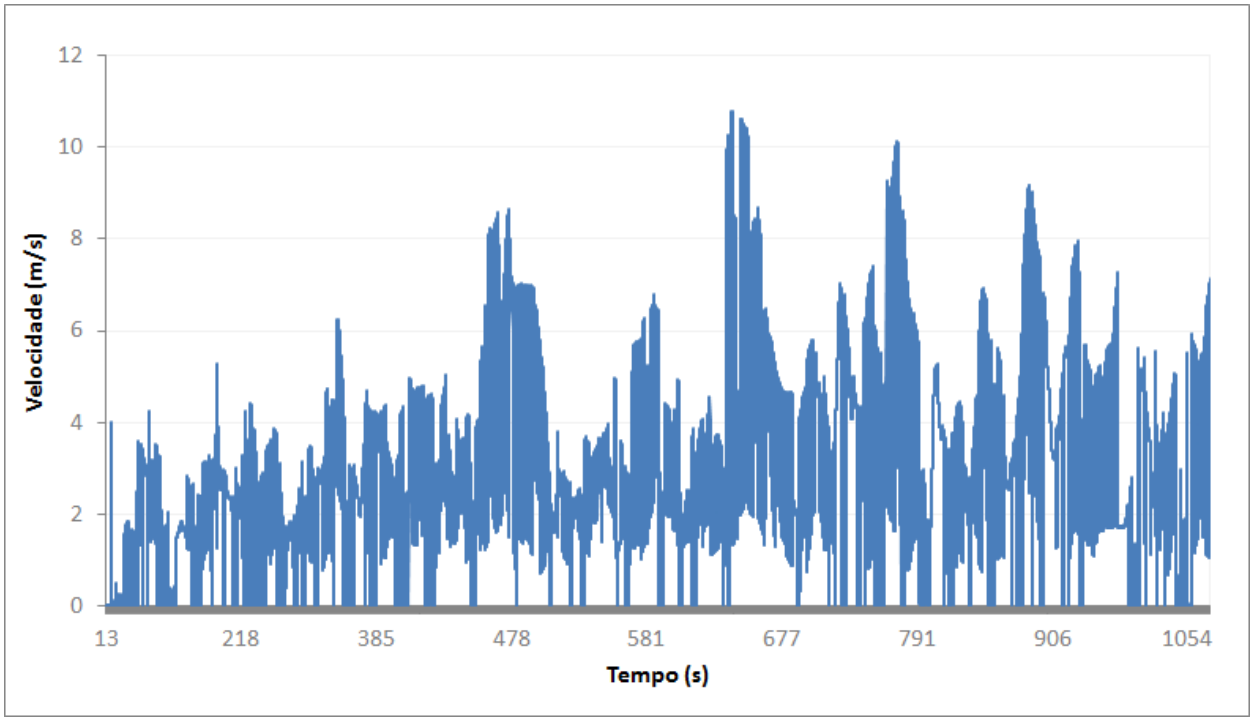
Com esse arquivo de texto podemos reproduzir gráficos de módulo da velocidade por tempo. Nos gráficos a seguir apresentamos os dados produzidos de velocidade por tempo para o predador nos dias 3, 4 e 5 de filmagem. E na sequência, mostrando a média de velocidade do predador para cada segundo em cada dia.

Figura 13: velocidade por tempo – Dia 3 Predador



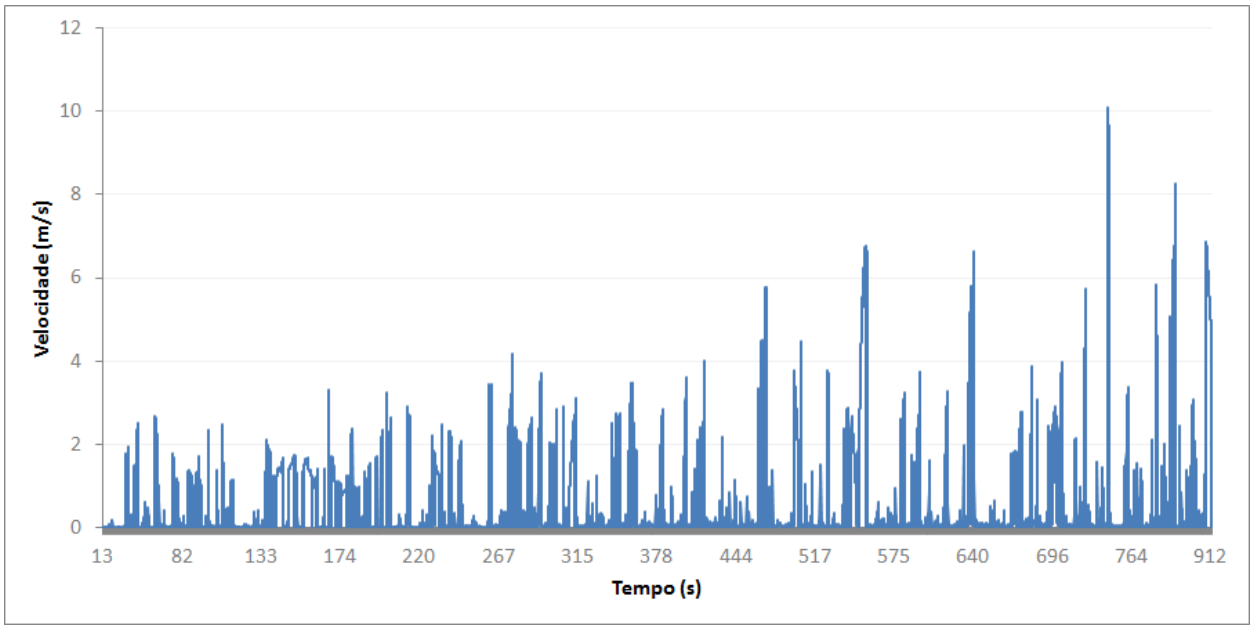
FONTE: O autor 2016

Figura 14: velocidade por tempo – Dia 4 Predador



FONTE: O autor 2016

Figura 15: velocidade por tempo – Dia 5 Predador

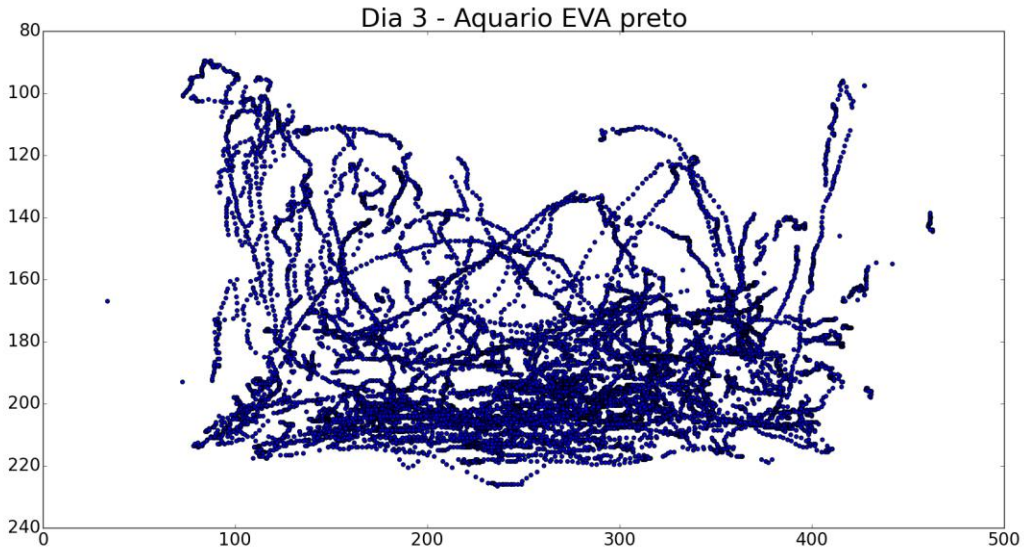


FONTE: O autor 2016

Na totalidade dos gráficos apresentados anteriormente existe uma grande variação nos valores de velocidade, como mostrado na tabela de parâmetros estatísticos (Tabelas 4 e 5). Isso é possível porque como observamos nos vídeos há uma mudança repentina na velocidade tanto das presas quanto do predador. Isso se deve a fuga das presas com a aproximação do predador (aumento de velocidade das presas, evitação), aproximação do predador às presas (busca de alimento, forrageamento), entre outras interações.

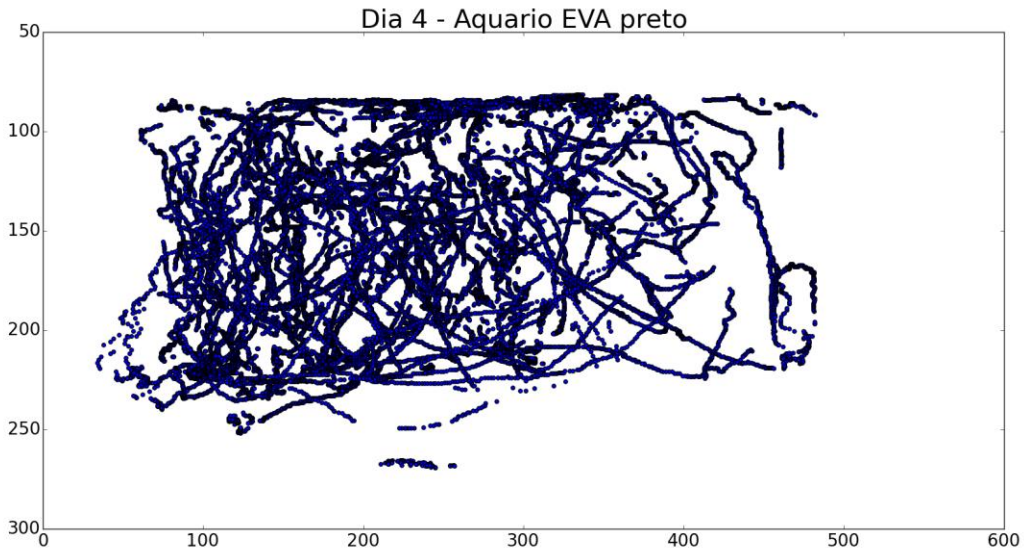
Com o arquivo de texto do predador podemos obter um gráfico de ocupação no aquário, inicialmente com a totalidade da filmagem (de 300 a 800 segundos aproximadamente, dependendo do dia de filmagem).

Figura 16: gráfico da ocupação do predador no dia 3 de filmagem



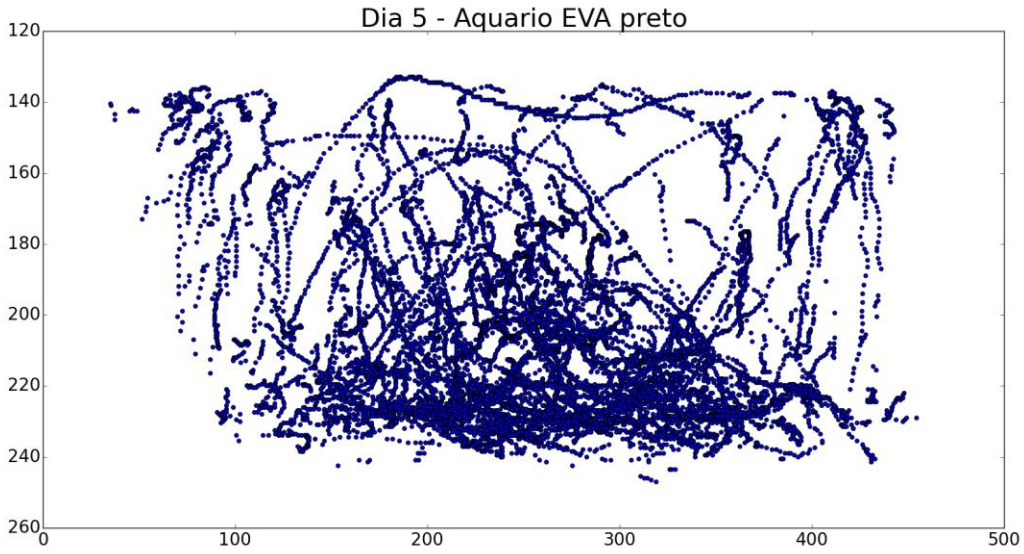
FONTE: O autor 2016

Figura 17: gráfico da ocupação do predador no dia 4 de filmagem



FONTE: O autor 2016

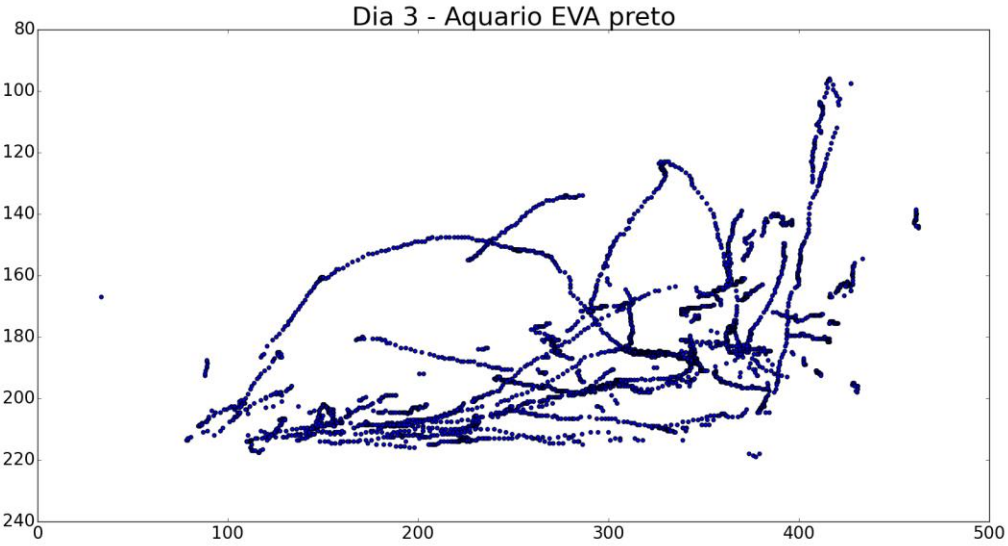
Figura 18: gráfico da ocupação do predador no dia 5 de filmagem



FONTE: O autor 2016

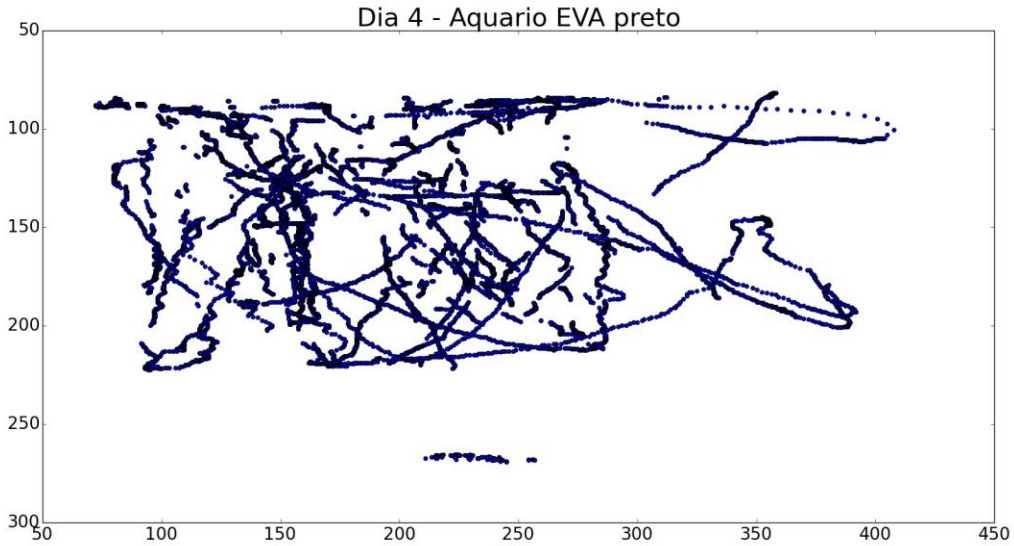
Agora com uma redução no intervalo de plotagem (de 300 a 599 segundos) para ser possível o estudo das movimentações, obtemos:

Figura 19: gráfico da ocupação do predador no dia 3 de filmagem (menor duração)



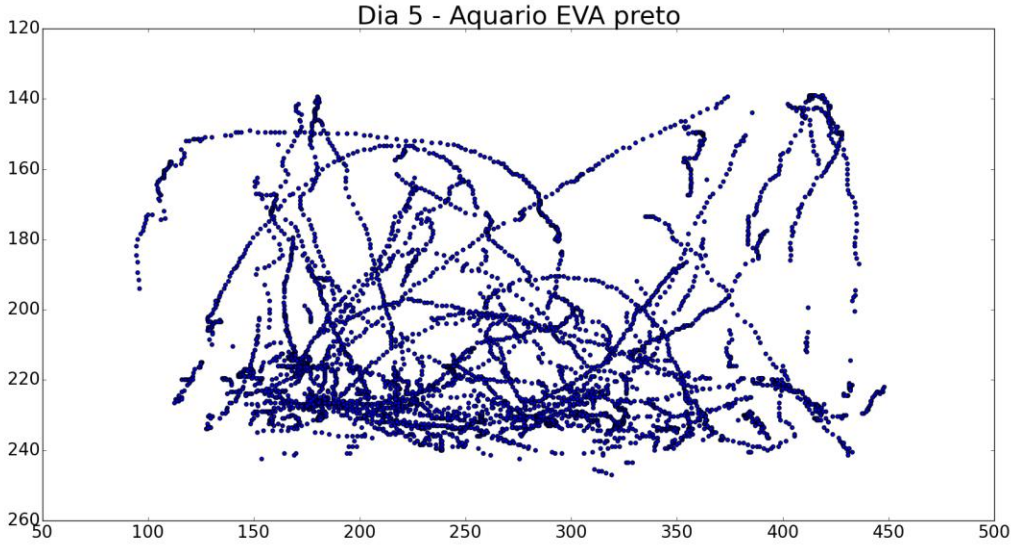
FONTE: O autor 2016

Figura 20: gráfico da ocupação do predador no dia 4 de filmagem (menor duração)



FONTE: O autor 2016

Figura 21: gráfico da ocupação do predador no dia 5 de filmagem (menor duração)



FONTE: O autor 2016

Com os dados de erros existentes, podemos fazer algumas análises estatísticas, como mostrado na tabela comparativa abaixo.

Tabela 3: Parâmetros estatísticos relacionados aos erros de contagem de predador, método I

	Dia 3	**Dia 4**	**Dia 5**
Média do erro da contagem do predador	0,1679	0,7948	0,0899
Variância do Erro de contagem predador	0,1622	0,7979	0,0823

FONTE: O autor 2016

Com todas essas séries de dados de velocidade do predador e das presas e velocidades médias, podemos montar uma tabela comparativa das médias gerais por dia de filmagem e variância dos dados. Assim, podemos observar se alguma semelhança existe.

Tabela 4: médias gerais por dia do predador, método I

	Dia 3	**Dia 4**	**Dia 5**
V_{MED} **predador (m/s)**	0,9436	3,0855	0,0135
σ^2_{VEL} **predador (m/s)**	1,984	4,0333	1,0832
V_{MAX} **predador (m/s)**	8,0932	10,7818	10,0863
Acertos na contagem do predador	84,17%	45,81%	91%

FONTE: O autor 2016

Tabela 5: médias gerais por dia das presas, método I

	Dia 3	Dia 4	Dia 5
V_{MED} **presas - com predador (m/s)**	2,2672	2,0073	1,7532
V_{MAX} **presas – com predador (m/s)**	12,1194	11,0347	11,7616
σ^2_{VEL} **presas (com predador presente)**	5,0796	4,1997	5,3074
V_{MED} **presas – sem predador (m/s)**	2,5904	2,5875	2,667
V_{MAX} **presas – sem predador (m/s)**	13,6804	10,6759	12,367
σ^2_{VEL} **presas (sem predador)**	3,5848	3,5189	3,982

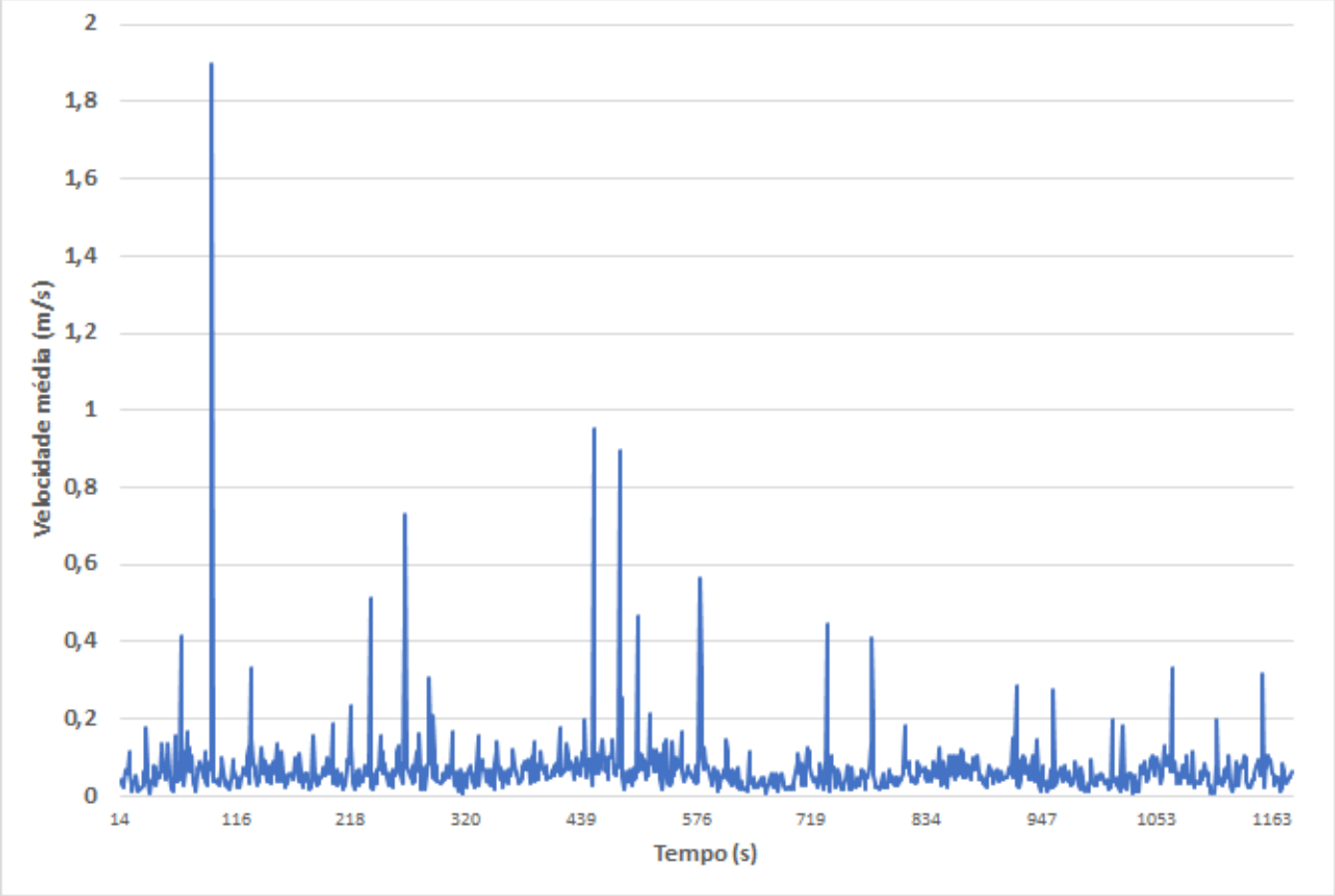
FONTE: O autor 2016

As velocidades mínimas tanto das presas como do predador não foram mostradas porque não foi possível distinguir a existência de um erro no cálculo ou a menor velocidade encontrada possui valor de 0,0135 m/s. Esse valor foi obtido como velocidade mínima em todos os cálculos.

6.2 Algoritmo II (Lucas-Kanade + *Motion detector* modificado)

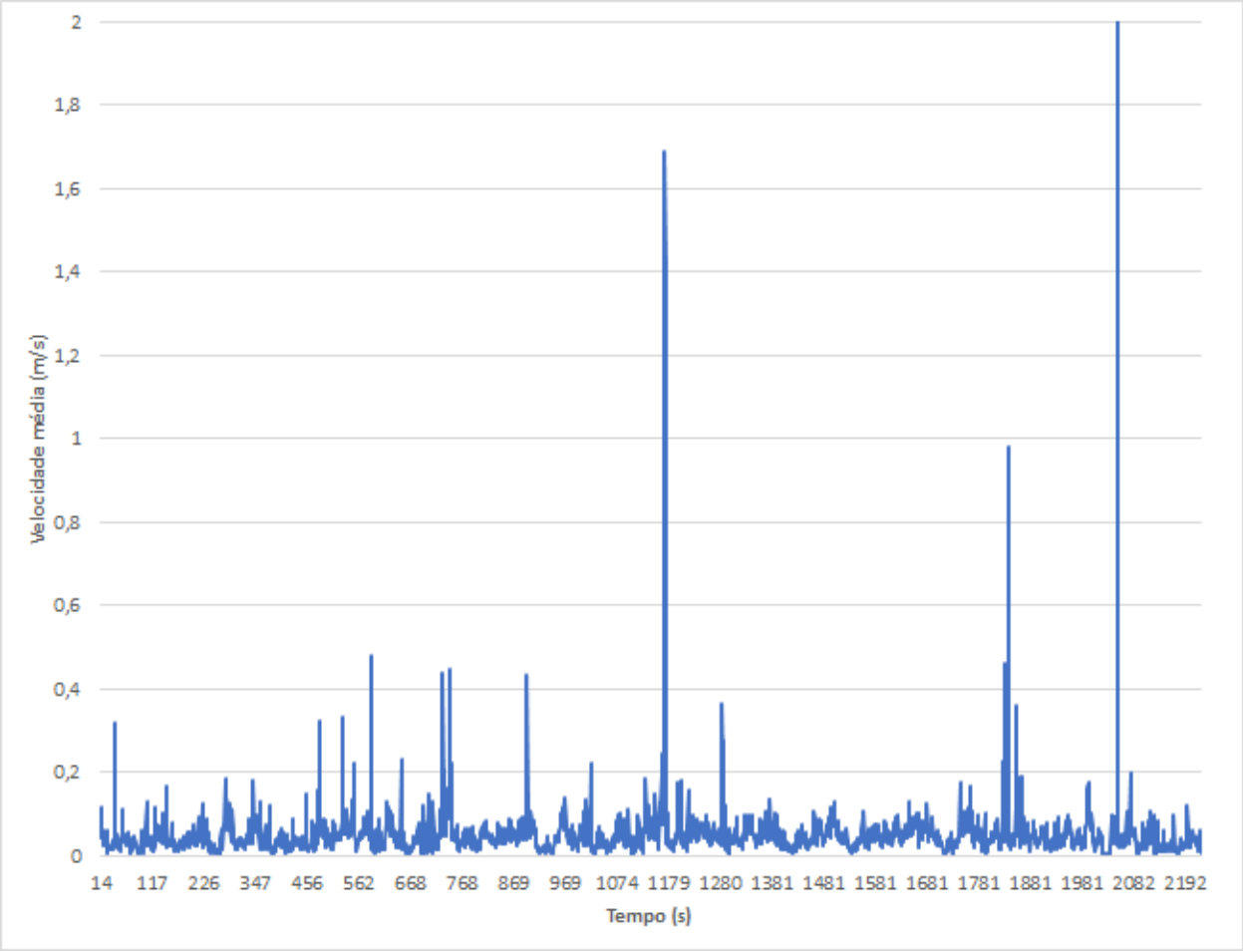
A partir do código computacional desenvolvido por Lucas e Kanade, inserimos o código *Motion Detector* modificado dentro de sua rotina de cálculo. E após o processamento dos vídeos pelo programa em Python, anteriormente explicado (disponível no Anexo 2), obtemos como saída um arquivo de texto que depois de trabalhado obtemos a velocidade para o tempo decorrido. Com a utilização do programa já mencionado para gerar imagem, o qual também calcula a média de velocidades para cada segundo, disponível no Anexo 3, obtemos um arquivo de texto com as velocidades médias para cada segundo.

Figura 22: velocidade média por tempo – Dia 3 Predador



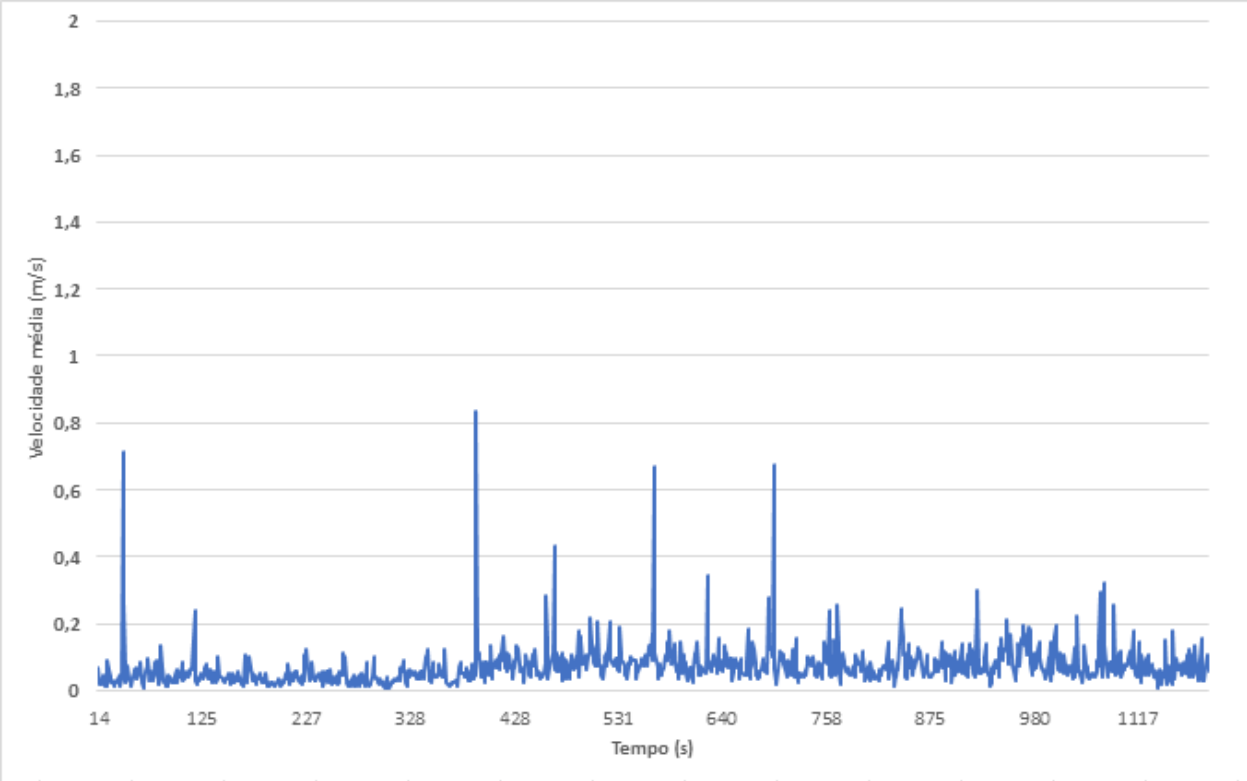
FONTE: O autor 2016

Figura 23: velocidade média por tempo – Dia 4 Predador



FONTE: O autor 2016

Figura 24: velocidade média por tempo – Dia 5 Predador



FONTE: O autor 2016

Na Tabela 6 a seguir estão os valores da velocidade média e da variância dos dados para cada dia, obtidos a partir do método II (Lucas-Kanade + *motion detector* modificado).

Tabela 6: médias gerais por dia, método II

	Dia 3	**Dia 4**	**Dia 5**
V_{MED} **predador (m/s)**	0,054	0,0412	0,06
V_{MAX} **predador (m/s)**	1,8974	4,413	0,8319
V_{MIN} **predador (m/s)**	0,0016	0,002	0,0066
σ^2_{VEL} **predador (m/s)**	0,0079	0,0133	0,0036

FONTE: O autor 2016

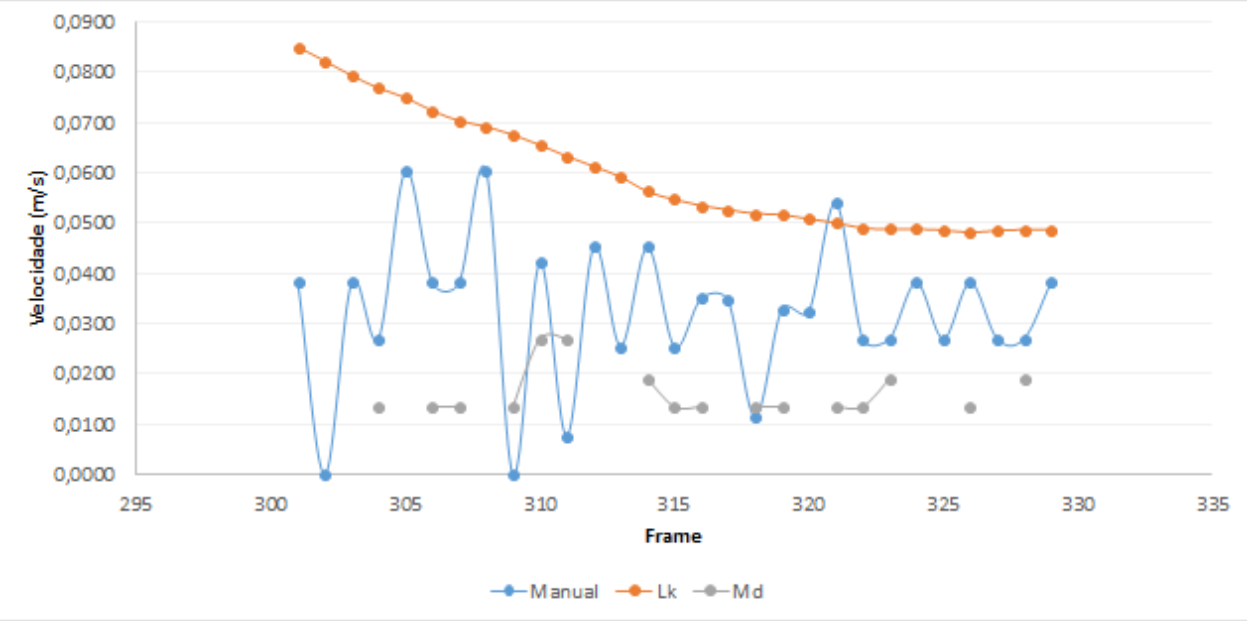
6.3 Método Visual (manual)

Uma maneira escolhida para a avaliação da confiabilidade dos métodos foi a observação frame a frame visualmente das posições (x, y) do predador, com o auxílio da ferramenta *Gimp*. Escolhemos quatro conjuntos dos frames sequenciais do dia 3 para analisar, a primeira de 299 a 329 (fase de aclimatação, início da filmagem), outra entre 533 e 559 (fase de forrageamento, primeira investida dada pelo predador realizada aos 18 segundos de filmagem ou próxima ao frame 540), entre os frames 689 e 721, e o último conjunto do frame 1019 ao 1058. A partir destes dados das posições, podemos calcular o deslocamento entre cada frame, e posteriormente obter a velocidade calculada com os mesmos passos do Algoritmo I – mostrado anteriormente. E os resultados das posições, deslocamentos e da velocidade para cada frame esta plotada na tabela presente no Anexo 5.

Na tabela presente no Anexo 6 comparamos as velocidades obtidas pelo Método Visual, a partir do Algoritmo I (Md) e Algoritmo II (Lk). O cálculo de velocidade do predador a partir do algoritmo I (*Motion detector* modificado) não é obtido para todos os frames, pois só é calculado quando é encontrado um retângulo com área entre 800 e 5000 pixels.

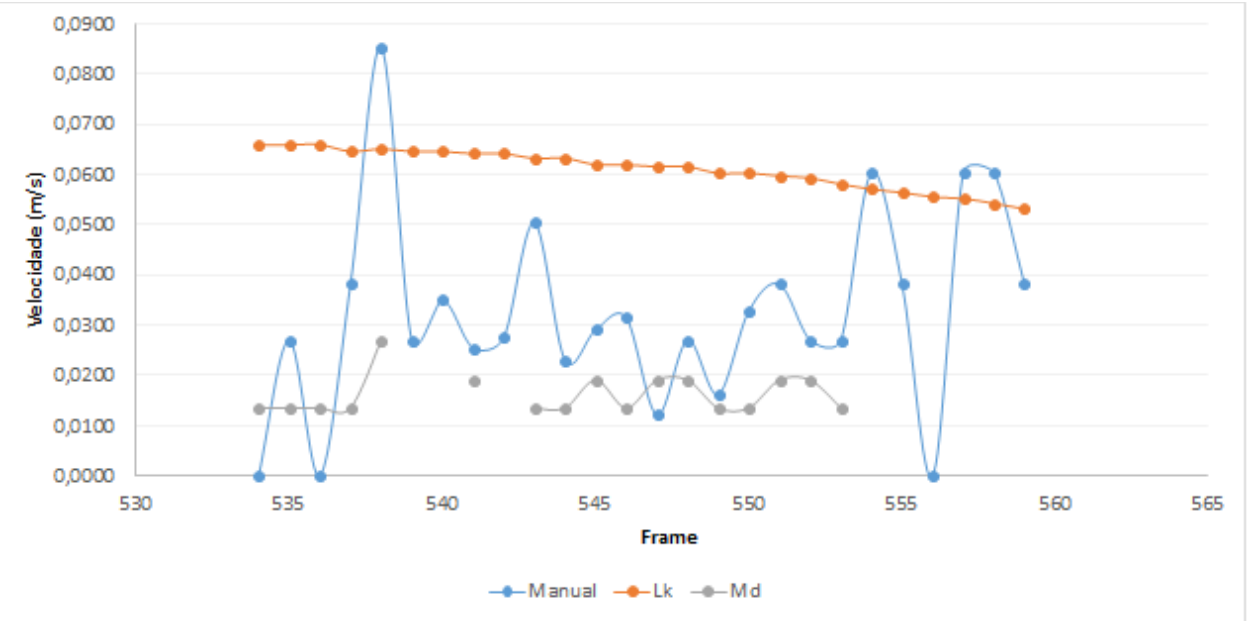
O próximo passo para verificar os erros dos códigos foi a plotagem dos gráficos de velocidade por frame para os três métodos escolhidos, nas Figuras 25 a 28 apresentadas abaixo. Os frames apresentados são os mesmo que foram anteriormente mostrados nos Anexos 5 e 6.

Figura 25: Comparação entre métodos para intervalo de 301 a 329 frames



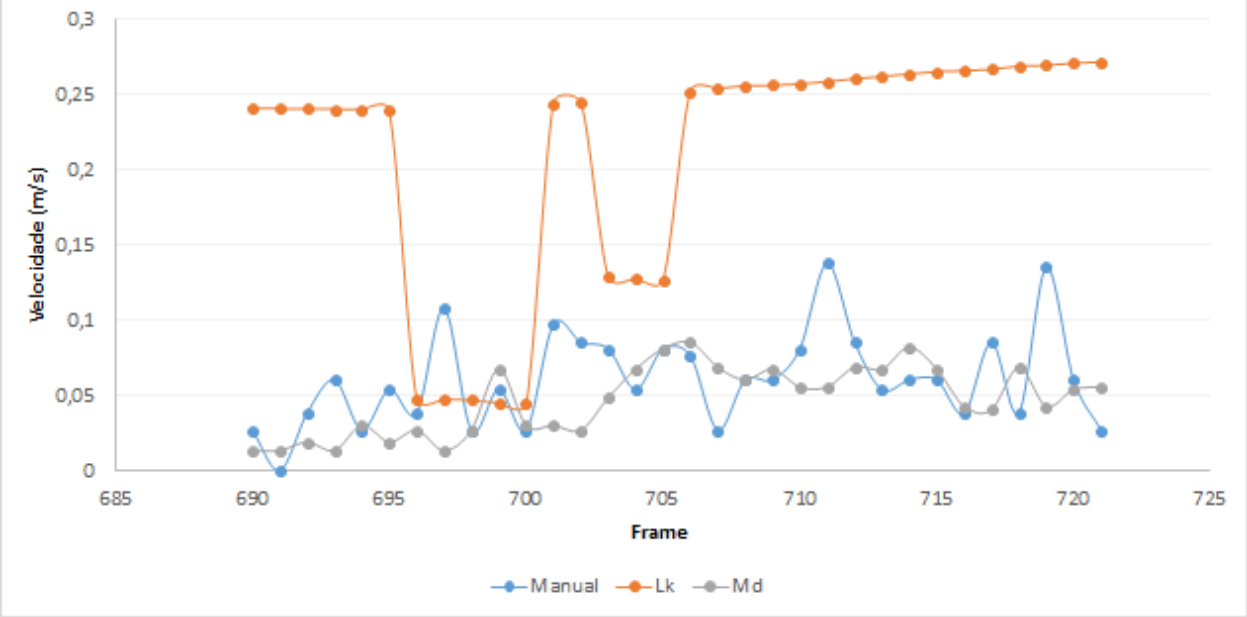
FONTE: O autor 2016

Figura 26: Comparação entre métodos para intervalo de 534 a 559 frames



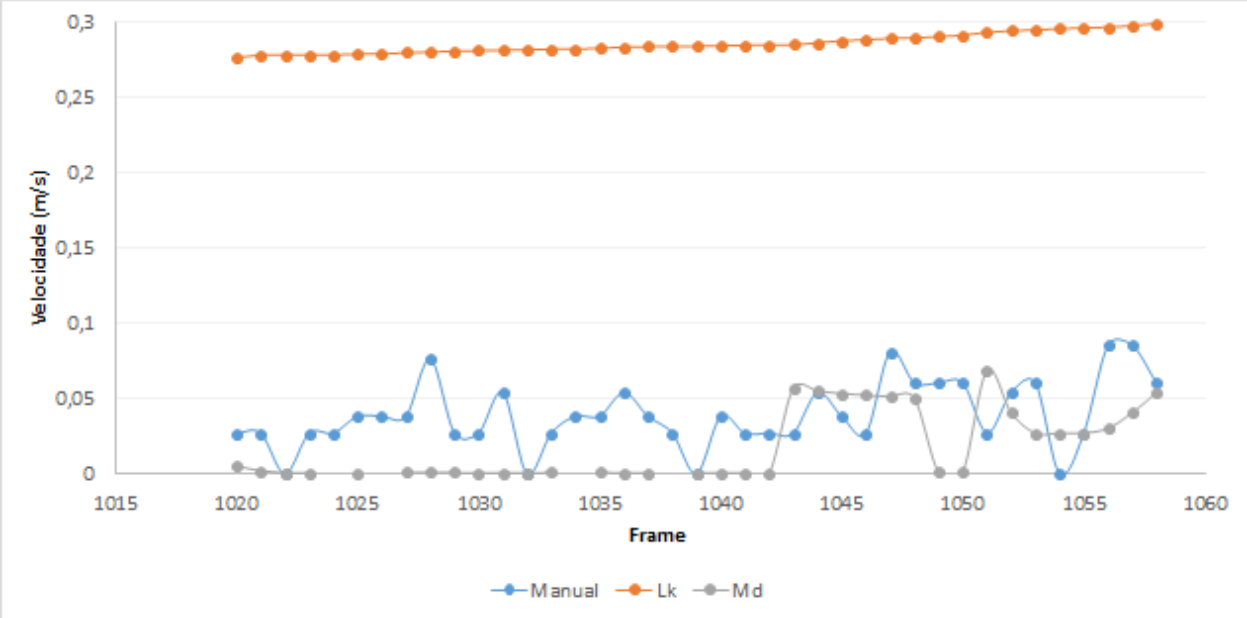
FONTE: O autor 2016

Figura 27: Comparação entre métodos para intervalo de 689 a 721 frames



FONTE: O autor 2016

Figura 28: Comparação entre métodos para intervalo de 1019 a 1058 frames



FONTE: O autor 2016

7 DISCUSSÃO

A partir dos dador apresentados nas Figuras 25 a 28, visualmente percebemos que o Algoritmo I, apresentado pela linha em cinza, representa melhor o que foi obtido pelo Método Visual, representado pela linha em azul, se comparado com os dados do Algoritmo II, linha em laranja. Outros métodos estatísticos foram utilizados para corroborar essa hipótese. Na tabela a seguir estão apresentados os valores de desvio quadrático (DESVQ) e erro quadrático médio (EQM) dos dois algoritmos quando comparados ao Método Visual.

Tabela 7: Comparação entre o método visual e cada um dos dois algoritmos desenvolvidos e cálculo de desvio quadrático e erro quadrático médio

	Método Visual e Algoritmo I	**Método Visual e Algoritmo II**
DESVQ	2,613	2,537
EQM	0,001	0,027

FONTE: O autor 2016

Os desvios quadráticos obtidos são bem próximos, uma diferença de apenas 0,0754. Já a comparação feita com os dados obtidos de erro médio quadrático é possível afirmar que há uma maior semelhança entre o Método Visual e o Algoritmo I, pois o seu valor é menor se comparado com o obtido pelo Algoritmo II.

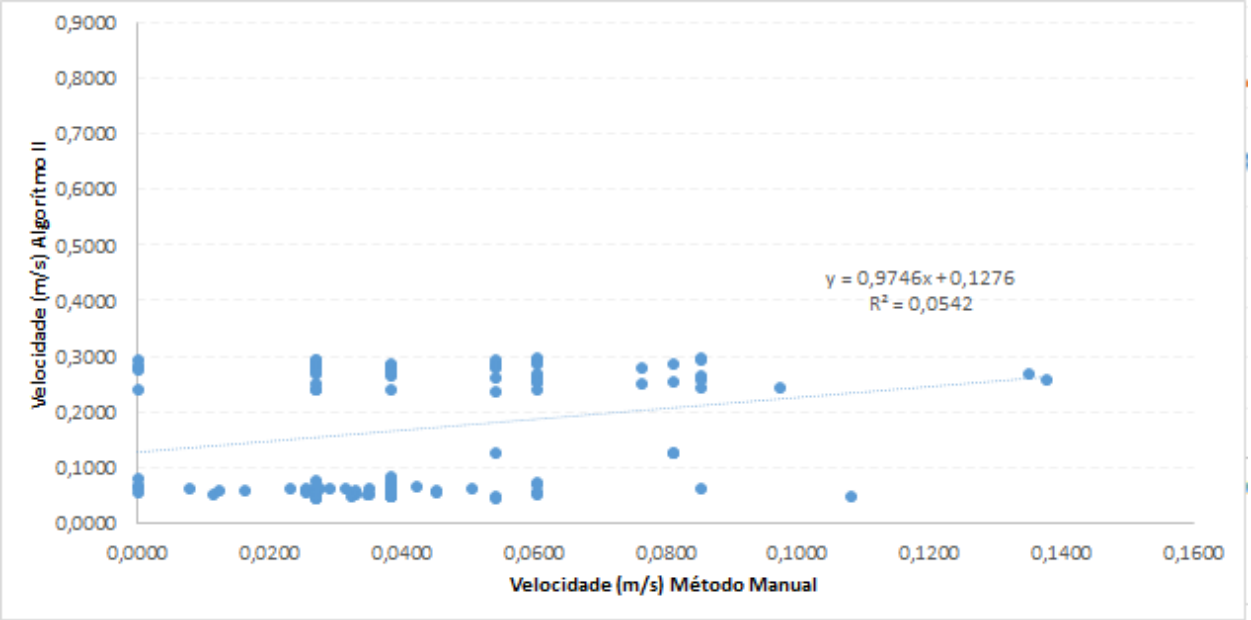
Uma terceira maneira para verificar qual dos dois algoritmos desenvolvidos representam melhor a realidade é o cálculo do R^2 (coeficiente de correlação de Pearson), o qual mostra uma possível correlação existente entre os dados plotados.

Tabela 8: valores de velocidades comparadas nos três métodos

	Método Visual x Algoritmo I	**Método Visual x Algoritmo II**
R^2	0,0005	0,0047

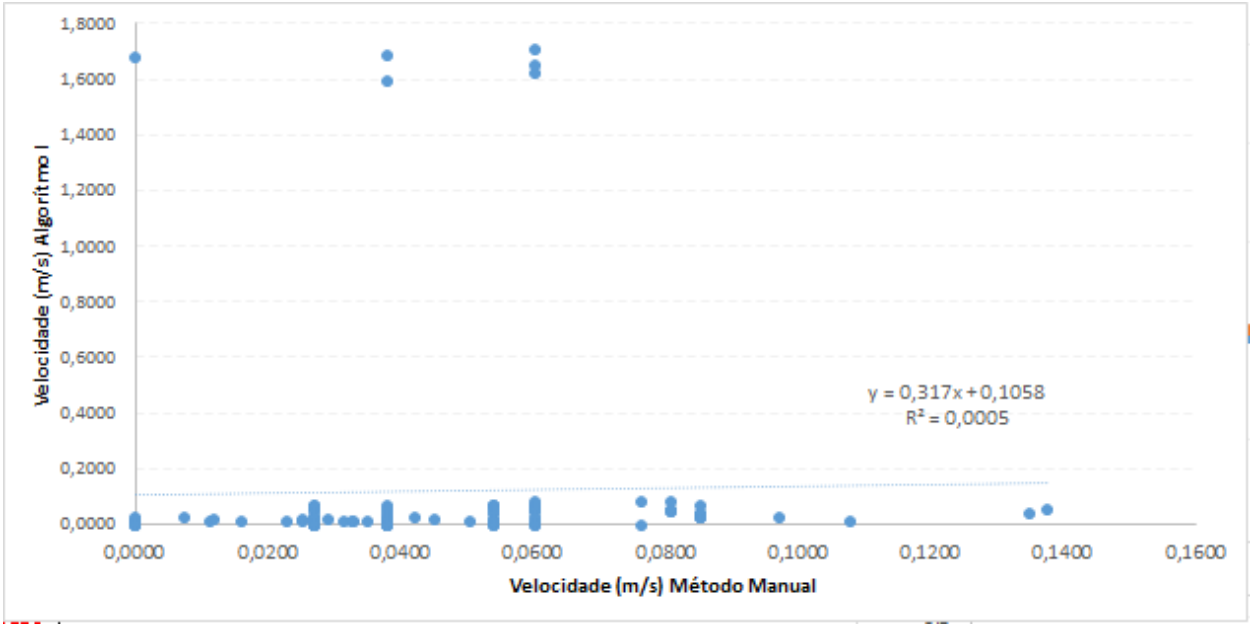
FONTE: O autor 2016

Figura 29: Comparação entre Método visual e Algoritmo II



FONTE: O autor 2016

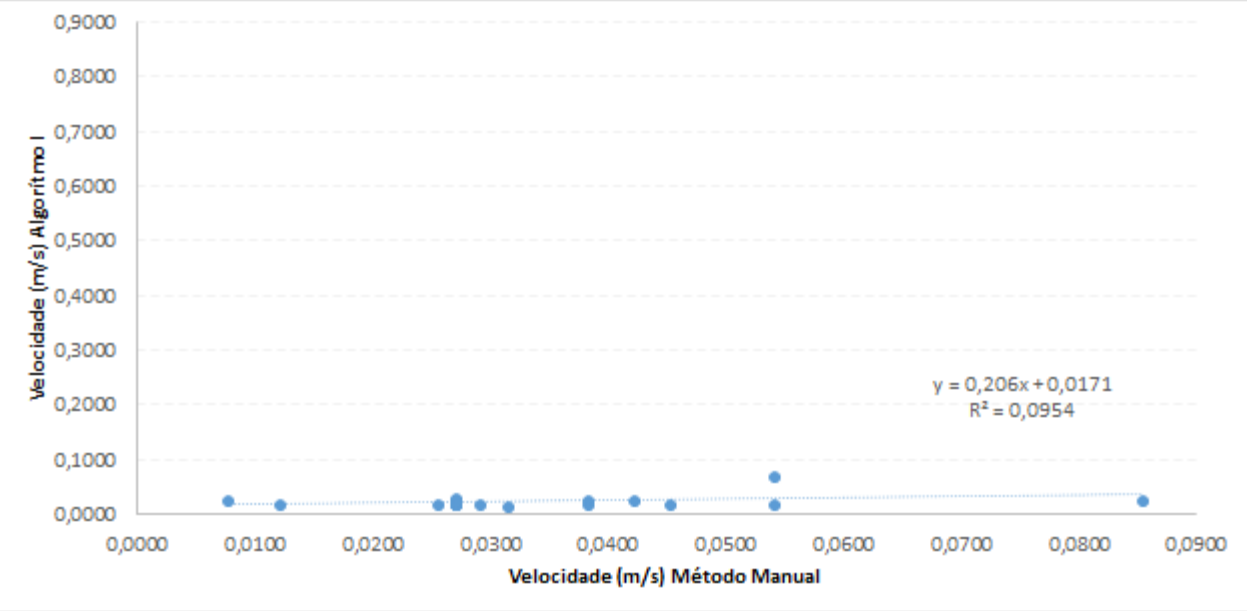
Figura 30: Comparação entre Método visual e Algoritmo I



FONTE: O autor 2016

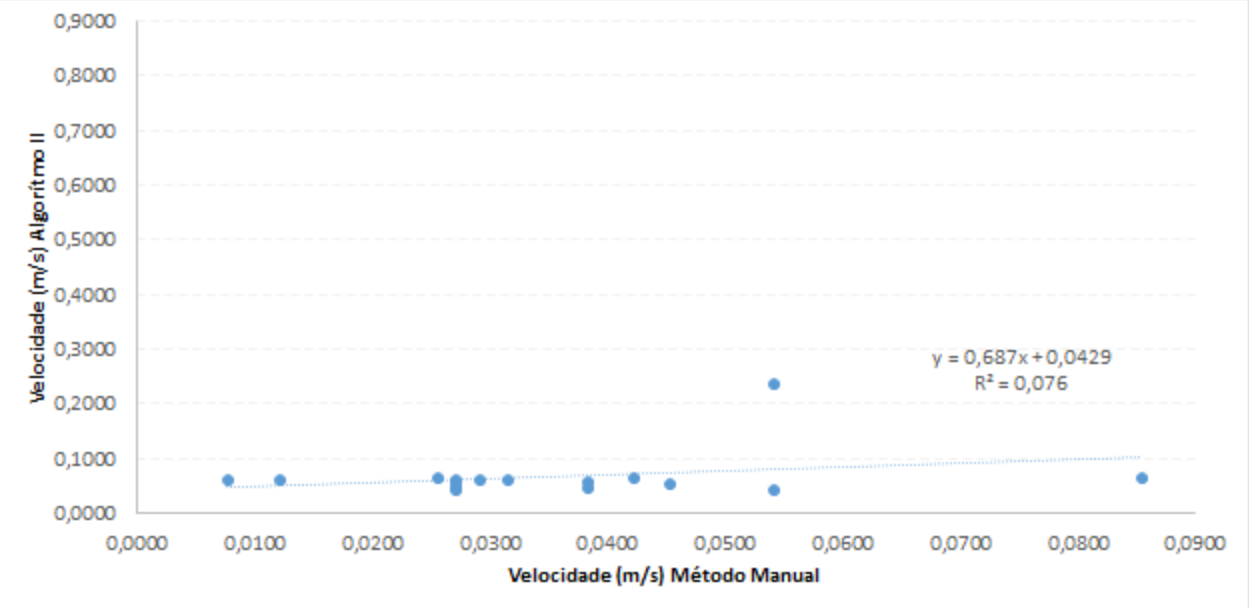
Na série de dados de velocidade obtido pelo Algoritmo I, como já explicado anteriormente, a velocidade mínima é sempre 0,0135. Até o momento não é possível distinguir quais desses valores são erro ou não. Portanto foram retirados os dados dos frames com essas velocidade para o cálculo do coeficiente de correlação, e os resultados são apresentados a seguir.

Figura 31: Comparação entre Método visual e Algoritmo I, modificado



FONTE: O autor 2016

Figura 32: Comparação entre Método visual e Algoritmo II, modificado



FONTE: O autor 2016

Tabela 9: valores de velocidades comparadas nos três métodos, modificados

	Método Visual x Algoritmo I	**Método Visual x Algoritmo II**
R^2	0,0954	0,076

FONTE: O autor 2016

Novamente, a comparação feita a partir do Método Visual com o algoritmo I forneceu um valor maior de R^2 se comparado com o obtido pelo algoritmo II, considerando os valores analisados após o trabalho com os pontos de erro no gráfico. O valor do coeficiente de correlação varia entre 0 e 1, sendo que quanto mais perto de 1 maior é a correlação. A Tabela 11, abaixo, resume os valores obtidos nas comparações entre os métodos.

Tabela 10: Resumo dos valores obtidos nas comparações entre métodos

	Método Visual com Algoritmo I	**Método Visual com Algoritmo II**
R^2	0,0954	0,076
DESVQ	2,613	2,537
EQM	0,001	0,027

FONTE: O autor 2016

Para compararmos as velocidades com dados reais, buscamos na internet na base de dados do *Fishbase* por mais informações sobre o *M. salmoides.* Os dados encontrados estão resumidos na tabela abaixo.

Tabela 11: dados de velocidade do *M. salmoides* da literatura

Velocidade (m/s)	**(Comprimento/s)**	**Modo de natação**	**Tipo de comprimento**	**Comprimento (cm)**
0,88	4,1	*Sustained* (sustentado)	TL	21

FONTE: *Fishbase*

Podemos comparar a velocidade de 0,88 m/s obtida da literatura, com os dados de velocidade média em cada dia para os dois algoritmos.

Tabela 12: velocidades médias obtidas por cada método em cada dia

	Dia 3	**Dia 4**	**Dia 5**
V_{MED} **predador (m/s) – Algoritmo I**	0,9436	3,0855	0,135
V_{MED} **predador (m/s) – Algoritmo II**	0,054	0,0412	0,06

FONTE: O autor 2016

Utilizando todas as métricas mostradas de coeficiente de correlação, erro médio, velocidade média e desvio quadrático, até o momento avaliamos o Algoritmo I como o que melhor representa a realidade, apesar de ainda conter muitos erros. Além disso, ainda é necessário verificar os erros na obtenção das velocidades mínimas.

8 CONCLUSÃO

Conclui-se que, foi possível aplicar e aperfeiçoar um design de experimento para obtenção de dados amostrais, facilitando a visualização e filmagem do comportamento e movimentação dos peixes estudados. Além disso, foi desenvolvido um programa de computador para detecção da presa e do predador e a sua diferenciação, possibilitando a identificação das posições de cada indivíduo, suas velocidades e, erros embutidos em cada análise computacional.

Aperfeiçoamos um desenho do experimento que facilitasse a visualização dos peixes, seus padrões de movimentações e interações no vídeo. Essas filmagens dos experimentos foram realizadas em três dias distintos (no primeiro semestre de 2016). Após todo o aparato para a filmagem já escolhido e montado (posição da câmera e aquário, cartolina, etc), e com os testes já realizados, foram utilizados quinze indivíduos como presas e três como predadores.

Desenvolvemos o código para o estudo das movimentações dos peixes filmados no ambiente controlado montado, e detectamos por meio desse algoritmo a presa e o predador nos instantes filmados, as posições dos mesmos e suas velocidades.

Mostramos a eficiência dos algoritmos desenvolvidos por meio de cálculos do R^2 na comparação de velocidades obtidas por algoritmos distintos (I e II), e porcentagem de acertos na contagem dos predadores.

A validação da metodologia foi feita a partir da comparação dos dados obtidos dos algoritmos e método visual, o qual se caracteriza pela obtenção visual da posição do centróide do predador em cada frame. Em conjunto com a comparação visual, foi realizada a análise estatística dos dados obtidos.

No presente trabalho focamos no estudo da movimentação do predador, com a utilização do programa computacional em Python *Motion Detector* (compara qualquer frame com o primeiro), e comparação com o programa desenvolvido por Lucas Kanade (compara o frame seguinte com o anterior).

8.1 Próximos passos

Deve-se continuar com os testes e tentar minimizar os erros no cálculo da velocidade e desenvolver métricas para caracterizar comportamentos funcionais distintos. Podem ser estudadas nesse projeto o padrão de movimentação para o comportamento de evitação das presas com a utilização de outras plataformas como, por exemplo, o código em MatLab *Particle Tracking Velocimetry – PTV*, bem como outros conceitos de probabilidade. Ao aumentar o número de diferentes códigos e programas utilizados para medir velocidade e deslocamento, conseguimos aumentar as comparações entre dados, e, portanto, temos mais confiabilidade nos dados obtidos.

Utilizando o conceito do movimento de Levy buscávamos verificar se pelo fato dos experimentos terem sido gravados em um ambiente confinado, como a teoria afirma, não teria sido representativo para a movimentação do predador. Mas, para podermos calculá-lo, necessitamos primeiro de um algoritmo com um bom funcionamento (alta confiabilidade dos dados).

Para verificarmos se as distribuições de probabilidades dos nossos experimentos se enquadram no conceito de movimento de Levy, podemos utilizar um código computacional disponibilizado no *Jupyter Notebook* pelo MIT (Alstott, 2013). Neste código obtemos os valores de alfa e sigma a partir do módulo do deslocamento do predador em cada dia, que representam o quanto os nossos dados se assemelham a uma distribuição *Heavy Tailed* (calda longa).

A outra ferramenta que podemos utilizar é do software MatLab. Nessa ferramenta, é possível calcular e visualizar a distribuição de velocidades por meio de sequência de imagens, utilizando método Lagrangeano para descrever os padrões de velocidades.

9 REFERÊNCIAS BIBLIOGRÁFICAS

Alstott J. (2013) Powerlaw: a python package for analysis of Heavy Taled distribution. Disponivel em: <http://journals.plos.org/plosone/article/asset?id=10.1371/journal.pone.0085777.PDF> Primeiro acesso em: 10 de outubro.

Buchanan M. (2008) Ecological modelling: The mathematical mirror to animal nature. Disponível em: http://www.nature.com/news/2008/080604/pdf/453714a.pdf; primeiro acesso em 31 de outubro.

Fishbase. Disponível em: http://fishbase.org Primeiro acesso em: 24 de setembro.

Gandhimohan M., Marcos G, Ernesto P., Eugene S., Cambridge Press (2011) THE PHYSICS OF FORAGING: An Introduction to Random Searches and Biological Encounters.

Gimp, disponível em: https://www.gimp.org; primeiro acesso em 06 de setembro.

International Union for Conservation of Nature (IUCN) 100 of the world's worst invasive alien species: a selection from the Global Invasive Species Database. Disponível em: https://portals.iucn.org/library/sites/library/files/documents/2000-126.pdf; primeiro acesso em 28 de julho.

Jupyter notebook. Disponivel em: <http://jupyter.org>. Primeiro acesso em: 17 de outubro.

Lasenby T. A., Kerr S. J. (2000) Bass transfers and stocking: An annotated bibliography and literature review.

Lightworks, disponível em: < https://www.lwks.com>; primeiro acesso em 07 de setembro.

Maezono Y, Miyashita T (2003) Community-level impacts induced by introduced largemouth bass and bluegill in farm ponds in Japan.

Matai J., R. Kastner, G. R. Cutter, D. A. Demer. University of California, San Diego, (2012) Automated Techniques for Detection and Recognition of Fishes using Computer vision Algorithms.

Mendes G. N. (1983) Estudo sobre aclimatação de alevinos de tainha (*Mugil curema* Valenciennes, 1836) à agua doce.

OpenCV: Optical Flow. Disponível em: <http://docs.opencv.org/trunk/d7/d8b/tutorial_py_lucas_kanade.html> Primeiro acesso em: 10 de novembro.

Paolucci E. M., H. J. MacIsaac, A. Ricciardi (2013) Origin matters - alien consumers inflict greater damage on prey populations than do native con copy.

Ribeiro V. (2013) *Micropterus salmoides*, Um predador introduzido em um reservatório neotropical: composição da dieta, táticas reprodutivas e métodos de captura.

Rosebrock A. (2015) Código computacional para câmera de segurança, Disponível em: <http://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/>; primeiro acesso em 19 de março.

Rubinstein M. (2014) See invisible motion, hear silent sounds. TED talk novembro 2014. Disponível em: <http://www.ted.com/talks/michael_rubinstein_see_invisible_motion_hear_silent_sounds_cool_creepy_we_can_t_decide>

Sih A., D. Bolnick, B. Luttbeg, J. Orrock, S. Peacor, L. Pintor, E. Preisser, J. Rehage and J. Vonesh (2010) Predator–prey naïveté, antipredator behavior, and the ecology of predator invasions.

Spampinato C., Y. Chen-Burguer, G. Nadaraja, R. Fisher (2008) Detecting, Tracking and couting fish in low quality unconstrained underwater videos.

Szeliski R. (2010) Computer Vision: Algorithms and Applications.

Trumpickas J, Mandrack NE, Ricciardi A (2011) Nearshore fish assemblages associated with introduced predatory fishes in lakes.

Viola, P., and M. Jones (2004) Robust real-time object detection.

Wolter C., Arlinghaus R. (2003) Burst and critical swimming speeds of fish and their ecological relevance in waterways.

Zaburdaev V. (2015) Levy Walks. Disponivel em: <https://arxiv.org/abs/1410.5100>; primeiro acesso em: 01 de novembro.

ANEXO 1 - Código *Motion Detector* modificado

```
# USAGE
# python motion_detector.py
# python motion_detector.py --video videos/example_01.mp4

# import the necessary packages
import argparse, datetime, imutils, time
import cv2
import re, os, math
import numpy as np

# Initialize variables
startTime = time.time()
timeElapsed = 0.0
start = time.clock()

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=10, help="minimum area size")
args = vars(ap.parse_args())

#-----------------------------------------
# Arquivos de entrada e saida
#-----------------------------------------
file = open("predador.txt", "w")
file2 = open("presas.txt", "w")
file3 = open("erropred.txt", "w")
file4 = open("erropres.txt", "w")
#fin = open('tempos.txt','rt');

linhas = 13

Tempo = []
predador = []
presa = []


# funcao para escrever nos resultados
def imprime_predador(tempo,frame,conta,x,y,a,vx,vy,vv,dx,dy,d):
    file.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">3.0f}",frame) + ' '  + str.format("{0:"
">3.0f}",conta) + ' ' + str.format("{0:" ">8.1f}", x)+ ' ' + str.format("{0:" ">8.2f}", y) + ' ' + str.format("{0:"
">8.2f}", a) + ' ' + str.format("{0:" ">10.6f}", vx) + ' ' + str.format("{0:" ">10.6f}", vy) +' ' + str.format("{0:"
">10.6f}", vv) + ' ' + str.format("{0:" ">8.6f}", dx) + ' ' + str.format("{0:" ">8.6f}", dy) +' ' + str.format("{0:"
">8.6f}", d) + ' \n ')

# funcao para escrever nos resultados
def imprime_presas(tempo,frame,conta,x,y,a,vx,vy,vv):
    file2.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">3.0f}",frame) + ' '  + str.format("{0:"
">3.0f}",conta) + ' ' + str.format("{0:" ">8.1f}", x)+ ' ' + str.format("{0:" ">8.2f}", y) + ' ' + str.format("{0:"
">8.2f}", a) + ' ' + str.format("{0:" ">10.6f}", vx) + ' ' + str.format("{0:" ">10.6f}", vy) +' ' + str.format("{0:"
">10.6f}", vv) +  ' \n ')
```

```
# funcao para escrever nos resultados
def imprime_erropred(tempo,erro):
    file3.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">10.6f}",erro) +  ' \n ')

# funcao para escrever nos resultados
def imprime_erropres(tempo,erro):
    file4.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">10.6f}",erro) +  ' \n ')



file.write(str.format(' '))
file2.write(str.format(' '))
file3.write(str.format(' '))
file4.write(str.format(' '))

# if the video argument is None, then we are reading from webcam
if args.get("video", None) is None:
        camera = cv2.VideoCapture(0)
        time.sleep(0.25)

# otherwise, we are reading from a video file
else:
        camera = cv2.VideoCapture(args["video"])


frameWidth = int(camera.get(cv2.cv.CV_CAP_PROP_FRAME_WIDTH))
frameHeight = int(camera.get(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT))


# initialize the first frame in the video stream
firstFrame = None

# inicializa variaveis
i = 0

posicao_x0Pred = 0
posicao_y0Pred = 0

posicao_x0Pres = 0
posicao_y0Pres = 0

# loop over the frames of the video
while True:
        # inicializa contagem de peixes
        contagempeixes = 0
        contagempredador = 0
        i = i + 1

        # grab the current frame and initialize the occupied/unoccupied
        # text
        (grabbed, frame) = camera.read()
        text = "Desocupado"

        # if the frame could not be grabbed, then we have reached the end
        # of the video
        if not grabbed:
```

```
        break

# resize the frame, convert it to grayscale, and blur it
frame = imutils.resize(frame, width=500)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (21, 21), 0)

# if the first frame is None, initialize it
if firstFrame is None:
        firstFrame = gray
        continue

# compute the absolute difference between the current frame and
# first frame
frameDelta = cv2.absdiff(firstFrame, gray)
thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

# dilate the thresholded image to fill in holes, then find contours
# on thresholded image
thresh = cv2.dilate(thresh, None, iterations=2)
(cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)

posicao_x = np.zeros(20)
posicao_y = np.zeros(20)
area_ok   = np.zeros(20)

# loop over the contours
for c in cnts:
        # if the contour is too small, ignore it
        if cv2.contourArea(c) < args["min_area"]:
                continue

        # compute the bounding box for the contour, draw it on the frame,
        # and update the text
        (x, y, w, h) = cv2.boundingRect(c)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        text = "Ocupado"

        centroidex = (x + (x + w))/2.0
        centroidey = (y + (y + h))/2.0
        area = w*h
        contagempeixes = contagempeixes + 1

        posicao_x[contagempeixes] = centroidex
        posicao_y[contagempeixes] = centroidey
        area_ok[contagempeixes]   = area

        if (area > 800 and area < 5000):

          delta_xPred = (centroidex - posicao_x0Pred)*(0.45/500)
          delta_yPred = (centroidey - posicao_y0Pred)*(0.45/500)
          saltoPred   = (delta_xPred**2+delta_yPred**2)**(0.5)

   posicao_x0Pred = centroidex
        posicao_y0Pred = centroidey
```

```
                    # Velocidades do Predador
                    velxPred = delta_xPred*30 # Velocidade x (m/s) para uma largura de 45 cm
                    velyPred = delta_yPred*30 # Velocidade y (m/s) para uma largura de 45 cm
                    velPred  = (velxPred**2+velyPred**2)**(0.5)

                    contagempredador = contagempredador + 1

imprime_predador(elapsed,i,contagempredador,centroidex,centroidey,area,velxPred,velyPred,velPred,delta_xPred,
delta_yPred, saltoPred)
                    imprime_erropred(elapsed,abs(1-contagempredador))
                   elif (area < 800):

                    delta_xPres = (centroidex - posicao_x0Pres)*(0.45/500)
                    delta_yPres = (centroidey - posicao_y0Pres)*(0.45/500)
                    saltoPres   = (delta_xPres**2+delta_yPres**2)**(0.5)

            posicao_x0Pres = centroidex
                    posicao_y0Pres = centroidey

                    # Velocidades do Presa
                    velxPres = delta_xPres*30 # Velocidade x (m/s) para uma largura de 45 cm
                    velyPres = delta_yPres*30 # Velocidade y (m/s) para uma largura de 45 cm
                    velPres  = (velxPres**2+velyPres**2)**(0.5)

                    imprime_presas(elapsed,i,contagempeixes,centroidex,centroidey,area,velxPres,velyPres,velPres)
                    #imprime_erropres(elapsed,abs(5-contagempeixes)) arrumar o codigo


        # draw the text and timestamp on the frame
        cv2.putText(frame, "Estatus: {}".format(text), (10, 20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
        #cv2.putText(frame, datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"),
        #           (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

        # mostra o tempo do video
    elapsed = (time.clock() - start)
    #print("Elapsed time: ", elapsed, " seconds")
    cv2.putText(frame, 'Tempo: {:.2f}'.format(elapsed), (10, frame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

    # Gera imagem do frame da contagem visual da Vanessa
    #if int(elapsed) in Tempo:
    #   print('ok - entrou')
    #   vis = frame.copy() # vis >> show image and points
    #   cv2.line(vis,(158,0),(158,169),(255,0,0),1)
    #   cv2.line(vis,(334,0),(334,169),(255,0,0),1)
    #   cv2.putText(vis, 'Tempo: {:.2f}'.format(elapsed), (10, frame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)
    #   cv2.imwrite('imagens/image'+str(i)+'.png',vis)
    #   imprime_arquivo2(i,contagempeixes,centroidex,centroidey)
    #   imprime_histograma(elapsed, histograma_TOTAL1,histograma_TOTAL2,histograma_TOTAL3)

    #break

        # show the frame and record if the user presses a key
```

```
            cv2.imshow("Security Feed", frame)
            #cv2.imshow("Thresh", thresh)
            #cv2.imshow("Frame Delta", frameDelta)
            key = cv2.waitKey(1) & 0xFF

            # if the `q` key is pressed, break from the lop
            if key == ord("q"):
                        break

# cleanup the camera and close any open windows
camera.release()
cv2.destroyAllWindows()
```

ANEXO 2 – Código Lucas-Kanade modificado

```
#!/usr/bin/env python

'''
Lucas-Kanade tracker
====================

Lucas-Kanade sparse optical flow demo. Uses goodFeaturesToTrack
for track initialization and back-tracking for match verification
between frames.

Usage
-----
python lk_mot-detect.py [<video_source>]


Keys
----
ESC - exit
'''
import pylab
import numpy as np
import cv2
import video
import sys
from common import anorm2, draw_str
from time import clock
from matplotlib import pyplot as plt

# import the necessary packages
import argparse, datetime, imutils, time
import re, os, math


# Initialize variables
startTime = time.time()
timeElapsed = 0.0
start = time.clock()

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=100, help="minimum area size")
args = vars(ap.parse_args())

#-----------------------------------------
# Arquivos de entrada e saida
#-----------------------------------------
file = open("predador.txt", "w")
file2 = open("presas.txt", "w")
file3 = open("erropred.txt", "w")
file4 = open("erropres.txt", "w")
#fin = open('tempos.txt','rt');

linhas = 13
```

```
Tempo = []
predador = []
presa = []


# funcao para escrever nos resultados
def imprime_predador(tempo,frame,conta,x,y,a,vx,vy,vv,dx,dy,d):
    file.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">3.0f}",frame) + ' '  + str.format("{0:"
">3.0f}",conta) + ' ' + str.format("{0:" ">8.1f}", x)+ ' ' + str.format("{0:" ">8.2f}", y) + ' ' + str.format("{0:"
">8.2f}", a) + ' ' + str.format("{0:" ">8.2f}", vx) + ' ' + str.format("{0:" ">8.2f}", vy) +' ' + str.format("{0:" ">8.2f}",
vv) + ' ' + str.format("{0:" ">8.6f}", dx) + ' ' + str.format("{0:" ">8.6f}", dy) +' ' + str.format("{0:" ">8.6f}", d) + ' \n
')

# funcao para escrever nos resultados
def imprime_presas(tempo,frame,conta,x,y,a,vx,vy,vv):
    file2.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">3.0f}",frame) + ' '  + str.format("{0:"
">3.0f}",conta) + ' ' + str.format("{0:" ">8.1f}", x)+ ' ' + str.format("{0:" ">8.2f}", y) + ' ' + str.format("{0:"
">8.2f}", a) + ' ' + str.format("{0:" ">8.2f}", vx) + ' ' + str.format("{0:" ">8.2f}", vy) +' ' + str.format("{0:" ">8.2f}",
vv) +  ' \n ')

# funcao para escrever nos resultados
def imprime_erropred(tempo,erro):
    file3.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">10.6f}",erro) +  ' \n ')

# funcao para escrever nos resultados
def imprime_erropres(tempo,erro):
    file4.write(str.format(str.format("{0:" ">10.6f}", tempo)+ ' ' + "{0:" ">10.6f}",erro) +  ' \n ')



file.write(str.format(' '))
file2.write(str.format(' '))
file3.write(str.format(' '))
file4.write(str.format(' '))


lk_params = dict( winSize  = (4, 4),
                  maxLevel = 300,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

feature_params = dict( maxCorners = 50,
                       qualityLevel = 0.2,
                       minDistance = 2,
                       blockSize = 7 )

class App:
    def __init__(self, video_src): # Variaveis
        self.track_len = 100
        self.detect_interval = 1
        self.tracks = [] # keep track of the current frame
        self.cam = video.create_capture(video_src)
        self.frame_idx = 0

    def run(self):
        # initialize the first frame in the video stream
```

```
        firstFrame = None

            # inicializa variaveis
            i = 0


        frameme = [25000]
        frameimage = [1049,1050,1051,1052,1053,1054,1055,1056,1057,1058] # frames que eu quero tirar imagens
        while True:
                # inicializa contagem de peixes
                contagempeixes = 0
                i = i + 1

           ret, frame = self.cam.read() #captura frame-by-frame
           frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # convertendo cada frame para cinza

           # resize the frame, convert it to grayscale, and blur it
           frame = imutils.resize(frame, width=500)
           gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
           gray = cv2.GaussianBlur(gray, (21, 21), 0)

           # if the first frame is None, initialize it
           if firstFrame is None:
              firstFrame = gray
              continue

           # compute the absolute difference between the current frame and
           # first frame
           frameDelta = cv2.absdiff(firstFrame, gray)
           thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

           # dilate the thresholded image to fill in holes, then find contours
           # on thresholded image
           thresh = cv2.dilate(thresh, None, iterations=2)
           (cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)
# inicio

                posicao_x = np.zeros(20)
                posicao_y = np.zeros(20)
                area_ok   = np.zeros(20)

                posicao_x0 = 0
                posicao_y0 = 0


                # loop over the contours
                for c in cnts:
                        # if the contour is too small, ignore it
                        if cv2.contourArea(c) < args["min_area"]:
                                continue

                        # compute the bounding box for the contour, draw it on the frame,
                        # and update the text
                        (xn, yn, w, h) = cv2.boundingRect(c)
                        cv2.rectangle(frame, (xn, yn), (xn + w, yn + h), (0, 255, 0), 2)
```

```
                    text = "Ocupado"

                    centroidex = (xn + (xn + w))/2.0
                    centroidey = (yn + (yn + h))/2.0
                    area = w*h
                    contagempeixes = contagempeixes + 1

                    posicao_x[contagempeixes] = centroidex
                    posicao_y[contagempeixes] = centroidey
                    area_ok[contagempeixes]   = area

                    # Velocidades do Predador
                    velx = (centroidex - posicao_x0)*(0.45/500)*30 # Velocidade x (m/s) para uma largura de 45
cm
                    vely = (centroidey - posicao_y0)*(0.45/500)*30 # Velocidade y (m/s) para uma largura de 45
cm
                    vel  = (velx**2+vely**2)**(0.5)

                    delta_x = (centroidex - posicao_x0)*(0.45/500)
                    delta_y = (centroidey - posicao_y0)*(0.45/500)
                    salto   = (delta_x**2+delta_y**2)**(0.5)

                    if (area > 800 and area < 5000):

                      imprime_predador(elapsed,i,contagempeixes,centroidex,centroidey,area,velx,vely,vel,delta_x,
delta_y, salto)
                  imprime_erropred(elapsed,abs(1-contagempeixes))

#inicio2


            frame_gray = thresh

            vis = frame.copy() # vis >> show image and points
            #if self.frame_idx in frameimage:
            #    cv2.imwrite('image'+str(self.frame_idx)+'.png',vis)
            if len(self.tracks) > 0:# process moving points       # len Return the length (the number of items) of an
object
                # para length > 0
                img0, img1 = self.prev_gray, frame_gray
                p0 = np.float32([tr[-1] for tr in self.tracks]).reshape(-1, 1, 2) # Reshape to fit input format
                p1, st, err = cv2.calcOpticalFlowPyrLK(img0, img1, p0, None, **lk_params) # calculate optical
flow/ handle large displacements
                p0r, st, err = cv2.calcOpticalFlowPyrLK(img1, img0, p1, None, **lk_params)
                d = abs(p0-p0r).reshape(-1, 2).max(-1) #abs >> retorna valor absoluto
                good = d < 1
                new_tracks = []
                     #if self.frame_idx in frameimage :
                print self.frame_idx
                file10.write(' frame index: '+ str(self.frame_idx) + ' contagem de pontos = ' + str(len(self.tracks)) +
'\n')  # frame atual + qtde objetos
                file10.write('cont     p0_x     p1_x     p0_y     p1_y   status   erro       v_x        v_y         v  \n ')
                conta = 0

                for cont in range(len(p0)):
                        if st[cont][0] == 1.0:
```

```
                        conta = conta + 1
                                    v_x=((p1[cont][0][0]-p0[cont][0][0]) *(0.45/500)*30)
                                    v_y=((p1[cont][0][1]-p0[cont][0][1]) *(0.45/500)*30)
                        v=((v_x)**(2.0) + (v_y)**(2.0))**(0.5)

                        imprime_arquivo(cont,conta,st,p0,p1,err,v_x,v_y,v)
                                            if (p0[cont][0][0] < xn + w and p0[cont][0][1] < yn+h and
p0[cont][0][0] > xn and p0[cont][0][1] > yn and err[cont][0] < 0.99 and v > 0.0000001):
                                imprime_arquivo2(elapsed,cont,conta,p0,p1,err,v_x,v_y,v)

                    for tr, (x, y), good_flag in zip(self.tracks, p1.reshape(-1, 2), good):
                        if not good_flag:
                            continue
                        tr.append((x, y))
                        if len(tr) > self.track_len:
                            del tr[0]
                        new_tracks.append(tr)
                        cv2.circle(vis, (x,y), 2, (0, 255, 0), -1)
                    self.tracks = new_tracks
                    cv2.polylines(vis, [np.int32(tr) for tr in self.tracks], False, (0, 255, 0))
                    draw_str(vis, (20, 20), 'track count: %d' % len(self.tracks))

                    # inicio - histograma - parte nova
                    #if self.frame_idx in frameimage :
                        #f = pylab.figure()
                        #ax = f.add_axes([0.1, 0.1, 0.8, 0.8])
                        #ax.bar(x, y, align='center')
                        #ax.set_xticks(x)
                        #ax.set_xticklabels(['quadrante 1', 'quadrante 2', 'quadrante 3'])
                        #pylab.savefig('graph'+str(self.frame_idx)+'.png', bbox_inches='tight') # desenha histograma
                    # fim - histograma - parte nova

                                if self.frame_idx in frameme: # programa para no frame
                        sys.exit()
                        #break

                if self.frame_idx % self.detect_interval == 0: # perform edge detection  #  %0 -->???
                    mask = np.zeros_like(frame_gray)
                    mask[:] = 255
                    for x, y in [np.int32(tr[-1]) for tr in self.tracks]:
                        cv2.circle(mask, (x, y), 5, 0, -1)
                    p = cv2.goodFeaturesToTrack(frame_gray, mask = mask, **feature_params) #
goodFeaturesToTrack >>  Detect corners (ShiTomasi)
                    if p is not None:
                        for x, y in np.float32(p).reshape(-1, 2):
                            self.tracks.append([(x, y)])

                if self.frame_idx in frameimage:
                    cv2.imwrite('image'+str(self.frame_idx)+'.png',vis)

                #self.frame_idx += 1
                self.prev_gray = frame_gray
                cv2.imshow('lktrack', vis) # display the resulting frame
                ch = 0xFF & cv2.waitKey(1) # Esc key (ascii number 27)
                if ch == 27:
                    break
```

```
#fim2
                        elif (area < 800):
                          imprime_presas(elapsed,i,contagempeixes,centroidex,centroidey,area,velx,vely,vel)
                          #imprime_erropres(elapsed,abs(5-contagempeixes)) arrumar o codigo

                    posicao_x0 = centroidex
                        posicao_y0 = centroidey

            self.frame_idx += 1

                # draw the text and timestamp on the frame
                #cv2.putText(frame, "Estatus: {}".format(text), (10, 20),
                        #cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

                # mostra o tempo do video
            elapsed = (time.clock() - start)
            #print("Elapsed time: ", elapsed, " seconds")
            #cv2.putText(frame, 'Tempo: {:.2f}'.format(elapsed),(10, frame.shape[0]-
10),cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

# final

def imprime_arquivo(cont,conta,st,p0,p1,err,v_x,v_y,v):
    file10.write(str.format("{0:" ">3.0f}",conta) + ' ' + str.format("{0:" ">8.1f}", p0[cont][0][0])+ ' ' + str.format("{0:"
">8.2f}", p1[cont][0][0]) + ' ' + str.format("{0:" ">8.2f}",p0[cont][0][1])  + ' ' + str.format("{0:"
">8.2f}",p1[cont][0][1]) + ' ' + str.format("{0:" ">6.0f}",st[cont][0]) + ' ' +   str.format("{0:" ">8.2f}",err[cont][0]) +
'  ' + str.format("{0:" ">10.5f}",v_x) + '  ' + str.format("{0:" ">10.5f}",v_y) + '  ' + str.format("{0:" ">8.2f}",v) +  ' \n
')

def imprime_arquivo2(tempocor,cont,conta,p0,p1,err,v_x,v_y,v):
    file11.write(str.format("{0:" ">3.4f}",tempocor) + ' ' +str.format("{0:" ">3.0f}",conta) + ' ' + str.format("{0:"
">8.1f}", p0[cont][0][0])+ ' ' + str.format("{0:" ">8.2f}", p1[cont][0][0]) + ' ' + str.format("{0:"
">8.2f}",p0[cont][0][1])  + ' ' + str.format("{0:" ">8.2f}",p1[cont][0][1]) + ' ' +str.format("{0:" ">8.2f}",err[cont][0])
+ '  ' + str.format("{0:" ">12.8f}",v_x) + '  ' + str.format("{0:" ">12.8f}",v_y) + '  ' + str.format("{0:" ">12.8f}",v) +
' \n ')


def main():
    import sys
    try: video_src = sys.argv[2]  # Procura o argumento depois da linha >>> python lk_track2.py [argumento 1]
[argumento 2] ..
    except: video_src = 0

    print __doc__
    App(video_src).run()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    file10  = open("coordenadas.txt", "w")
    file11  = open("velocidades-lk-md.txt", "w")
    main()
```

ANEXO 3 – Código modificado de geração de imagens

```
#!/usr/bin/env python
'''
Gera imagens
====================
'''

import pylab
import numpy as np
import cv2
import video
from common import anorm2, draw_str
from time import clock
from matplotlib import pyplot as plt
lk_params = dict( winSize  = (4, 4),
                  maxLevel = 300,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
feature_params = dict( maxCorners = 50,
                       qualityLevel = 0.2,
                       minDistance = 2,
                       blockSize = 7 )
class App:def __init__(self, video_src): # Variaveis

        self.track_len = 10
        self.detect_interval = 10
        self.tracks = [] # keep track of the current frame
        self.cam = video.create_capture(video_src)
        self.frame_idx = 0
def run(self): while True:

ret, frame = self.cam.read() #captura frame-by-frame frame_gray = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY) # convertendo cada frame para cinza vis = frame.copy() # vis >> show image
and points if self.frame_idx % 15 == 0: # para gerar imagens a cada 15 frames

cv2.imwrite('image'+str(self.frame_idx)+'.png',vis) self.frame_idx += 1 self.prev_gray =
frame_gray cv2.imshow('lktrack', vis) # display the resulting frame ch = 0xFF & cv2.waitKey(1) #
Esc key (ascii number 27) if ch == 27:

break

try: video_src = sys.argv[1] # Procura o argumento depois da linha >>> python lk_track2.py
[argumento 1] [argumento 2] ..

except: video_src = 0

print __doc__ App(video_src).run() cv2.destroyAllWindows()

if __name__ == '__main__': file = open("coordenadas.txt", "w") main()
```

ANEXO 4 – Gerar gráfico de ocupação do predador no aquário

```
import re, os, math
        from pydoc import help
        #from scipy.stats.stats import pearsonr
        import matplotlib.pyplot as plt
        import matplotlib
        from pylab import *

        matplotlib.rc('xtick', labelsize=20)
        matplotlib.rc('ytick', labelsize=20)


        #-----------------------------------------
        # Arquivos de entrada e saida
        #-----------------------------------------
        file = open("predador.txt", "r")
        file2 = open("des-vel-media-predador.txt", "w")

        # funcao para escrever nos resultados
        def imprime_vel(t,desloc,v):
            file2.write(str.format("{0:" ">10.1f}", t)+ '       ' + str.format("{0:" ">8.6f}",
        desloc) +  '       ' + str.format("{0:" ">8.6f}", v) +  ' \n ')

        frame   = []
        #conta   = []
        x       = []
        y       = []
        area    = []
        #w       = []
        #h       = []
        vel     = []

        tempo_atual = 13
        vel_media = 0
        desloc   = 0
        contador = 1

        for line in file:
           campo = re.split(r'[;/:\s]\s*', line)
           print(campo)
           if len(campo) > 5:
             segundo = float(campo[1])
             frame.append(float(campo[2]))
             x.append(float(campo[4]))
             y.append(float(campo[5]))
             area.append(float(campo[6]))
             vel.append(float(campo[9]))

             #print(int(segundo), tempo_atual)
             if (int(segundo) == tempo_atual):
               #print('Primeiro if')
               vel_media = vel_media + float(campo[9])
               desloc = desloc + float(campo[12])
               contador = contador + 1
               tempo_atual = int(segundo)


             if (int(segundo) > tempo_atual):
               #print('Segundo if')
               if (contador != 0):
                 vel_media = vel_media/contador
                  imprime_vel(int(segundo),desloc,vel_media)
                  vel_media = float(campo[9])
                  desloc    = float(campo[12])
                  contador  = 1
                  tempo_atual = int(segundo)
```

```
print("Ok")
plt.figure(figsize=(20, 10)) # This increases resolution
plt.scatter(x,y) #.invert_yaxis()



# Linhas
#l1_x = [155, 155]
#l1_y = [0,225]
#l1_y = [500,500-169]
#l2_x = [348,348]
#l2_y = [0,225]
#l2_y = [500,500-169]

#plt.plot(l1_x,l1_y, lw=5, color='black') #.invert_yaxis()
#plt.plot(l2_x,l2_y, lw=5, color='black') #.invert_yaxis()

#plt.ylim((50,225))
#plt.xlim((0,500))
# Reverse the y-axis
ax = gca()
ax.set_ylim(ax.get_ylim()[::-1])

title('Dia 3 - Aquario EVA preto', fontsize=30)
#plt.ylabel('Average Price')
#plt.xlabel('Mark-up', fontsize=25)
#plt.grid(True)


#Exo = dict(fc="white", ec="black", lw=3)
#t = ax.text(80, 60, "Exotico", ha="center", va="center", rotation=0,
#            size=40,
#            bbox=Exo)
#Neu = dict(fc="white", ec="black", lw=2)
#t = ax.text(250, 60, "Neutro", ha="center", va="center", rotation=0,
#            size=40,
#            bbox=Neu)
#Nat = dict(fc="white", ec="black", lw=2)
#t = ax.text(430, 60, "Nativo", ha="center", va="center", rotation=0,
#            size=40,
#            bbox=Nat)


plt.savefig('fig-aquario3_2.png')
#plt.xticks(paramValues)
```

ANEXO 5 – valores observados frame a frame visualmente

Frame	X	Y	delta x (m)	delta y (m)	V
299	184	124			
300	184	124	0	0	0
301	185	125	0,0009	0,0009	0,0382
302	185	125	0	0	0
303	184	124	-0,0009	-0,0009	0,0382
304	184	125	0	0,0009	0,027
305	186	124	0,0018	-0,0009	0,0604
306	185	125	-0,0009	0,0009	0,0382
307	186	124	0,0009	-0,0009	0,0382
308	187	126	0,0009	0,0018	0,0604
309	187	126	0	0	0
310	185,8	125	-0,00108	-0,0009	0,0422
311	186	125,2	0,00018	0,00018	0,0076
312	186,5	126,8	0,00045	0,00144	0,0453
313	186	126	-0,00045	-0,00072	0,0255
314	187,2	126,5	0,00108	0,00045	0,0351
315	187,5	126,2	0,00027	-0,00027	0,0115
316	186,8	127,2	-0,00063	0,0009	0,0329

317	186	128,2	-0,00072	0,0009	0,0346
318	186,5	127,5	0,00045	-0,00063	0,0232
319	185,8	127	-0,00063	-0,00045	0,0232
320	187	127	0,00108	0	0,0324
321	189	127	0,0018	0	0,054
322	188	127	-0,0009	0	0,027
323	187	127	-0,0009	0	0,027
324	188	128	0,0009	0,0009	0,0382
325	188	127	0	-0,0009	0,027
326	189	128	0,0009	0,0009	0,0382
327	188	128	-0,0009	0	0,027
328	188	127	0	-0,0009	0,027
329	189	128	0,0009	0,0009	0,0382
533	157	159			
534	157	159	0	0	0
535	157	160	0	0,0009	0,027
536	157	160	0	0	0
537	156	159	-0,0009	-0,0009	0,0382
538	157	162	0,0009	0,0027	0,0854
539	158	162	0,0009	0	0,027

540	156,8	161,5	-0,00108	-0,00045	0,0351
541	156	161	-0,00072	-0,00045	0,0254
542	156,2	162	0,00018	0,0009	0,0275
543	158	162,5	0,00162	0,00045	0,0504
544	157,2	162,8	-0,00072	0,00027	0,0231
545	158,2	163,2	0,0009	0,00036	0,0291
546	159,2	163,8	0,0009	0,00054	0,03149
547	159	164,2	-0,00018	0,00036	0,01207
548	159,8	164,8	0,00072	0,00054	0,027
549	159,8	164,2	0	-0,00054	0,0162
550	160	163	0,00018	-0,00108	0,03285
551	159	164	-0,0009	0,0009	0,0382
552	160	164	0,0009	0	0,027
553	161	164	0,0009	0	0,027
554	163	165	0,0018	0,0009	0,0604
555	164	166	0,0009	0,0009	0,0382
556	164	166	0	0	0
557	166	167	0,0018	0,0009	0,0604
558	164	166	-0,0018	-0,0009	0,0604
559	165	167	0,0009	0,0009	0,0382

689	173	204			
690	173	203	0	-0,0009	0,027
691	173	203	0	0	0
692	172	204	-0,0009	0,0009	0,0382
693	170	203	-0,0018	-0,0009	0,06037
694	171	203	0,0009	0	0,027
695	173	203	0,0018	0	0,054
696	174	204	0,0009	0,0009	0,0382
697	178	204	0,0036	0	0,108
698	177	204	-0,0009	0	0,027
699	179	204	0,0018	0	0,054
700	179	203	0	-0,0009	0,027
701	182	205	0,0027	0,0018	0,0974
702	185	206	0,0027	0,0009	0,0854
703	188	206	0,0027	0	0,081
704	190	206	0,0018	0	0,054
705	193	206	0,0027	0	0,081
706	195	208	0,0018	0,0018	0,0764
707	195	207	0	-0,0009	0,027
708	197	208	0,0018	0,0009	0,0604

709	199	207	0,0018	-0,0009	0,0604
710	202	207	0,0027	0	0,081
711	207	208	0,0045	0,0009	0,1377
712	210	207	0,0027	-0,0009	0,0854
713	212	207	0,0018	0	0,054
714	214	208	0,0018	0,0009	0,0604
715	216	207	0,0018	-0,0009	0,0604
716	215	208	-0,0009	0,0009	0,0382
717	218	207	0,0027	-0,0009	0,0854
718	219	208	0,0009	0,0009	0,0382
719	223	205	0,0036	-0,0027	0,135
720	225	206	0,0018	0,0009	0,0604
721	226	206	0,0009	0	0,027
1019	219	218			
1020	219	219	0	0,0009	0,027
1021	219	218	0	-0,0009	0,027
1022	219	218	0	0	0
1023	220	218	0,0009	0	0,027
1024	219	218	-0,0009	0	0,027
1025	220	219	0,0009	0,0009	0,0382

1026	221	218	0,0009	-0,0009	0,0382
1027	222	219	0,0009	0,0009	0,0382
1028	224	217	0,0018	-0,0018	0,0764
1029	224	218	0	0,0009	0,027
1030	225	218	0,0009	0	0,027
1031	227	218	0,0018	0	0,054
1032	227	218	0	0	0
1033	228	218	0,0009	0	0,027
1034	229	219	0,0009	0,0009	0,0382
1035	230	218	0,0009	-0,0009	0,0382
1036	228	218	-0,0018	0	0,054
1037	229	219	0,0009	0,0009	0,0382
1038	229	218	0	-0,0009	0,027
1039	229	218	0	0	0
1040	230	219	0,0009	0,0009	0,0382
1041	230	218	0	-0,0009	0,027
1042	231	218	0,0009	0	0,027
1043	232	218	0,0009	0	0,027
1044	234	218	0,0018	0	0,054
1045	235	217	0,0009	-0,0009	0,0382
1046	235	218	0	0,0009	0,027

1047	238	218	0,0027	0	0,081
1048	240	219	0,0018	0,0009	0,0604
1049	242	218	0,0018	-0,0009	0,0604
1050	244	219	0,0018	0,0009	0,0604
1051	245	219	0,0009	0	0,027
1052	247	219	0,0018	0	0,054
1053	249	218	0,0018	-0,0009	0,0604
1054	249	218	0	0	0
1055	250	218	0,0009	0	0,027
1056	253	217	0,0027	-0,0009	0,0854
1057	256	218	0,0027	0,0009	0,0854
1058	258	219	0,0018	0,0009	0,0604

FONTE: O autor 2016

ANEXO 6 – Valores de velocidades comparadas nos três métodos

Frame	Velocidade (m/s) Método manual	Velocidade (m/s) Algoritmo II	Velocidade (m/s) Algoritmo I
301	0,0382	0,0851	
302	0,0000	0,0823	
303	0,0382	0,0794	
304	0,0270	0,0769	0,0135
305	0,0604	0,0750	
306	0,0382	0,0724	0,0135
307	0,0382	0,0704	0,0135
308	0,0604	0,0692	
309	0,0000	0,0676	0,0135
310	0,0422	0,0655	0,0270
311	0,0076	0,0633	0,0270
312	0,0453	0,0613	
313	0,0255	0,0592	
314	0,0453	0,0563	0,0191
315	0,0255	0,0547	0,0135
316	0,0351	0,0535	0,0135
317	0,0346	0,0527	
318	0,0115	0,0519	0,0135
319	0,0330	0,0517	0,0135
320	0,0324	0,0509	
321	0,0540	0,0501	0,0135
322	0,0270	0,0490	0,0135
323	0,0270	0,0489	0,0191
324	0,0382	0,0489	
325	0,0270	0,0486	
326	0,0382	0,0482	0,0135
327	0,0270	0,0486	

328	0,0270	0,0487	0,0191
329	0,0382	0,0487	
534	0,0000	0,0659	0,0135
535	0,0270	0,0659	0,0135
536	0,0000	0,0660	0,0135
537	0,0382	0,0646	0,0135
538	0,0854	0,0651	0,0270
539	0,0270	0,0646	
540	0,0351	0,0646	
541	0,0255	0,0642	0,0191
542	0,0275	0,0642	
543	0,0504	0,0632	0,0135
544	0,0231	0,0633	0,0135
545	0,0291	0,0620	0,0191
546	0,0315	0,0620	0,0135
547	0,0121	0,0616	0,0191
548	0,0270	0,0616	0,0191
549	0,0162	0,0603	0,0135
550	0,0328	0,0603	0,0135
551	0,0382	0,0597	0,0191
552	0,0270	0,0593	0,0191
553	0,0270	0,0581	0,0135
554	0,0604	0,0571	1,7124
555	0,0382	0,0564	1,6897
556	0,0000	0,0556	1,6830
557	0,0604	0,0552	1,6568
558	0,0604	0,0542	1,6273
559	0,0382	0,0532	1,5947
690	0,0270	0,2408	0,0135
691	0,0000	0,2405	0,0135
692	0,0382	0,2404	0,0191

693	0,0604	0,2402	0,0135
694	0,0270	0,2399	0,0302
695	0,0540	0,2393	0,0191
696	0,0382	0,0477	0,0270
697	0,1080	0,0477	0,0135
698	0,0270	0,0469	0,0270
699	0,0540	0,0451	0,0675
700	0,0270	0,0443	0,0302
701	0,0973	0,2441	0,0302
702	0,0854	0,2446	0,0270
703	0,0810	0,1287	0,0487
704	0,0540	0,1278	0,0675
705	0,0810	0,1266	0,0810
706	0,0764	0,2520	0,0854
707	0,0270	0,2538	0,0688
708	0,0604	0,2557	0,0604
709	0,0604	0,2559	0,0675
710	0,0810	0,2569	0,0557
711	0,1377	0,2584	0,0557
712	0,0854	0,2603	0,0688
713	0,0540	0,2619	0,0675
714	0,0604	0,2635	0,0821
715	0,0604	0,2651	0,0675
716	0,0382	0,2657	0,0427
717	0,0854	0,2667	0,0405
718	0,0382	0,2686	0,0688
719	0,1350	0,2693	0,0427
720	0,0604	0,2706	0,0540
721	0,0270	0,2716	0,0557
1020	0,0270	0,2762	0,0051
1021	0,0270	0,2781	0,0020

1022	0,0000	0,2781	0,0006
1023	0,0270	0,2781	0,0006
1024	0,0270	0,2781	
1025	0,0382	0,2787	0,0006
1026	0,0382	0,2787	
1027	0,0382	0,2797	0,0010
1028	0,0764	0,2800	0,0010
1029	0,0270	0,2807	0,0009
1030	0,0270	0,2810	0,0005
1031	0,0540	0,2813	0,0005
1032	0,0000	0,2816	0,0005
1033	0,0270	0,2820	0,0010
1034	0,0382	0,2820	
1035	0,0382	0,2826	0,0009
1036	0,0540	0,2832	0,0006
1037	0,0382	0,2836	0,0005
1038	0,0270	0,2836	
1039	0,0000	0,2839	0,0005
1040	0,0382	0,2842	0,0005
1041	0,0270	0,2845	0,0005
1042	0,0270	0,2849	0,0005
1043	0,0270	0,2849	0,0569
1044	0,0540	0,2862	0,0551
1045	0,0382	0,2875	0,0532
1046	0,0270	0,2882	0,0522
1047	0,0810	0,2891	0,0514
1048	0,0604	0,2895	0,0504
1049	0,0604	0,2901	0,0009
1050	0,0604	0,2911	0,0014
1051	0,0270	0,2931	0,0688
1052	0,0540	0,2941	0,0405

1053	0,0604	0,2948	0,0270
1054	0,0000	0,2955	0,0270
1055	0,0270	0,2961	0,0270
1056	0,0854	0,2965	0,0302
1057	0,0854	0,2975	0,0405
1058	0,0604	0,2989	0,0540

Algoritmo I – Motion Detector modificado

Algoritmo II – Lucas-Kanade + Motion Detector modificado

FONTE: O autor 2016