

UNIVERSIDADE DO OESTE DE SANTA CATARINA – UNOESC
ÁREA DAS CIÊNCIAS EXATAS

CURSO DE CIÊNCIA DA COMPUTAÇÃO

JEFFERSON SILVIO MEIRELES DE MATOS

DESENVOLVIMENTO DE SOFTWARE E SIMULAÇÃO DE SEMÁFORO
INTELIGENTE

Videira - SC

2019

JEFFERSON SILVIO MEIRELES DE MATOS

DESENVOLVIMENTO DE SOFTWARE E SIMULAÇÃO DE SEMÁFORO
INTELIGENTE

Trabalho de conclusão de curso
apresentado ao Curso de Ciência da
Computação, Área das Ciências Exatas,
da Universidade do Oeste de Santa
Catarina, *campus* Videira.

Orientador: Prof. MSc. Herculano Haymussi De Biasi

Videira - SC

2019

JEFFERSON SILVIO MEIRELES DE MATOS

DESENVOLVIMENTO DE SOFTWARE E SIMULAÇÃO DE SEMÁFORO
INTELIGENTE

Trabalho de conclusão de curso
apresentado à Universidade do Oeste de
Santa Catarina, como requisito parcial à
obtenção do grau de Bacharel em Ciência
da Computação.

Aprovada em de de 2019.

BANCA EXAMINADORA

Prof. Dr. ...

Universidade do Oeste de Santa Catarina

Prof. Dr. ...

Universidade do Oeste de Santa Catarina

Prof. Dr. ...

Universidade do Oeste de Santa Catarina

Dedico este trabalho à minha mãe e esposa, fonte de meus conhecimentos e saber. Graças a elas, tornei-me uma pessoa capaz de lutar e se adaptar, para que meus sonhos e objetivos fossem sempre alcançados, sem jamais desanimar. Considero-me forte porque elas são fortes.

AGRADECIMENTOS

Agradeço à toda a minha família, que dentro de seus limites, forneceram todo o apoio necessário em minha formação acadêmica. Em especial à minha esposa Josiane Lopes Pasquali que sempre esteve junto a mim nesta importante etapa. Ao meu orientador Herculano Haymussi De Biasi por partilhar seu conhecimento e fomentar minha busca pelo aprimoramento pessoal, ao meu coordenador Fabiano de Oliveira Wonzoski que sempre se manteve a disposição para todo e qualquer problema. Agradeço também a todos os meus colegas por todo companheirismo e conversa que tivemos nos últimos anos. A todos os professores envolvidos e engajados pelo conhecimento. Um agradecimento em especial para as empresas Google, YouTube, Udemmy que com certeza facilitaram em muito o meu aprendizado e por último e não menos importante à Unoesc Videira que fez deste sonho realidade.

Podemos vender nosso tempo, mas não podemos comprá-lo de volta.

(FERNANDO PESSOA)

RESUMO

Com a atual evolução do mundo a importância que o transporte tem em nossas vidas nos faz dependentes na utilização de sistemas como o semáforo, o semáforo é um sistema antigo e utilizado em todo o mundo com o mesmo padrão (verde passa, amarelo atenção e vermelho para), o objetivo deste sistema é otimizar o tempo de espera em um semáforo utilizando técnicas de simulação para comparar o sistema atual com o proposto, modificando o sistema atual que é utilizado com tempo fixo por um sistema que se adapta ao fluxo de veículos. Desenvolveu-se um software utilizando *threads* e após isto utilizou-se o software SUMO para simular e comparar o sistema semafórico atual com o proposto. Para que a simulação apresentasse um resultado próximo ao real foram utilizados dados como (quantidade de carros, motos, ônibus, caminhões), fornecidas pelo IBGE e dados semafóricos colhidos diretamente no local (tempo de verde, amarelo, vermelho). A simulação foi executada em 86.400 segundos, equivalente a 24 horas. Os resultados mostraram que o sistema proposto é mais eficaz pois teve redução de 58,8% em relação ao tempo de espera do sistema atual, passando de 20h 48min (sistema atual) para 8h 38min (sistema proposto), com estas informações é possível afirmar que o sistema proposto é eficiente na otimização de tempo.

Palavras-Chave: Semáforo. Simulação. Otimização.

ABSTRACT

With the current evolution of the world the importance that transport has in our lives makes us dependent on the use of systems such as traffic lights, the traffic light is old system and used all over the world with the same pattern (green to go, yellow to attention and red to stop), the objective of this system is to optimize the waiting time in a traffic light using simulation techniques to compare the current system with the proposed one, modifying the current system that is used with fixed time by a system that adapts to the flow of vehicles. The software was developed using threads so that the system obtained the optimization of the time and after the software SUMO was used to simulate and compare the current system with the proposed, so that the simulation presented a result close to the real one were used data such as (quantity of cars, motorcycles, buses, trucks), provided by IBGE and semaphore data collected directly in the place (time of green, yellow, red). The simulation was performed in 86.400 seconds, equivalent to 24 hours. The results showed that the proposed system is more efficient because it had a reduction of 58.8% in relation to the waiting time of the current system, going from 20h 48min (current system) to 8h 38min (proposed system), with this information it is possible to affirm that the proposed system is efficient in time optimization.

Key-words: *Traffic light. Simulation. Optimization.*

LISTA DE ILUSTRAÇÕES

Figura 1: Mapa Senac Caçador.....	23
Figura 2: Composição do arquivo TLS	29
Figura 3: Exemplo de semáforo com estado atual "GrGr"	31
Figura 4: Exemplo de configuração tLogic <i>actuated</i>	32
Figura 5: Exemplo de configuração tLogic <i>delay_based</i>	33
Figura 6: Exemplo de configuração WAUT	34
Figura 7: Diagrama de classe.....	38
Figura 8: Diagrama de atividade	39
Figura 9: Método <i>cycle</i>	40
Figura 10: Parte do código onde ocorre a mudança dos estados	41
Figura 11: Método usado para colocar o sistema em espera.....	41
Figura 12: Métodos usados para calcular os tempos de espera	42
Figura 13: Método <i>manager</i>	43
Figura 14: Variável <i>position</i>	43
Figura 15: Mapa Viposa, Caçador – SC.....	44
Figura 16: Comandos utilizados NETCONVERT	44
Figura 17: Configuração do arquivo typemap.xml	45
Figura 18: Comandos utilizados POLYCONVERT	45
Figura 19: Comandos utilizados RANDOMTRIPS classe <i>Motorcycle</i>	46
Figura 20: Comandos utilizados RANDOMTRIPS classe <i>Passenger</i>	46
Figura 21: Comandos utilizados RANDOMTRIPS classe <i>Truck</i>	46
Figura 22: Comandos utilizados RANDOMTRIPS classe <i>Bus</i>	47
Figura 23: Configuração arquivo <i>tls_rep.xml</i>	47
Figura 24: SUMO-GUI id semáforo	48
Figura 25: Arquivo <i>cacador.sumocfg</i>	49
Figura 26: SUMO-GUI mapa de Caçador – SC.....	49
Figura 27: tLogic 2390821305 sistema atual.....	50
Figura 28: Sistema semaforico atual	51
Figura 29: tLogic 2390821305 sistema proposto.....	51
Figura 30: Sistema semaforico proposto.....	52
Figura 31: Comparação dos sistemas.....	52

LISTA DE QUADROS

Quadro 1: Opções utilizadas na ferramenta NETCONVERT	25
Quadro 2: Opções utilizadas na ferramenta POLYCONVERT	25
Quadro 3: Opções utilizadas na ferramenta RANDONTRIPS	26
Quadro 4: Opções utilizadas no <i>software</i> SUMO	27
Quadro 5: Composição do arquivo TLS	28
Quadro 6: Cores de sinal.....	29
Quadro 7: Opções WAUT.....	34
Quadro 8: Valores utilizados no método <i>cycle</i>	41
Quadro 9: Composição arquivo tls_rep.xml	48

LISTA DE EQUAÇÕES

(1).....	19
(2).....	20
(3).....	21
(4).....	21
(5).....	22
(6).....	22

LISTA DE ABREVIATURAS E SIGLAS

APIs	<i>Application Program Interface</i>
BPMN	<i>Business Process Model and Notation</i>
GSM	<i>Global System for Mobile Communication</i>
IPC	<i>Inter Process Communication</i>
SUMO	<i>Simulation of Urban Mobility</i>
SysML	<i>System Modelling Language</i>
TLS	<i>Traffic Lights</i>
WAUT	<i>Wochenschaltautomatik</i>

SUMÁRIO

1 INTRODUÇÃO	15
1.1 OBJETIVOS	16
1.1.1 Objetivos Específicos	16
1.2 JUSTIFICATIVA	16
1.3 METODOLOGIA	17
2 FUNDAMENTAÇÃO TEÓRICA	18
2.1 FLUXO DE TRÁFEGO	18
2.2 SEMÁFORO	18
2.2.1 Sinal Verde	19
2.2.2 Sinal Amarelo	22
2.2.3 Sinal Vermelho	22
2.3 OPENSTREETMAP	23
2.4 SUMO	23
2.4.1 NETCONVERT	25
2.4.2 POLYCONVERT	25
2.4.3 RANDONTRIPS	26
2.4.4 SUMO	26
2.4.5 SUMO-GUI	27
2.4.6 Traffic Lights	27
2.4.7 Semáforo Baseado em Intervalos de Tempo (<i>Actuated</i>)	31
2.4.8 Semáforo Baseado em Perda de Tempo (<i>Delay_based</i>)	32
2.5 WAUT	33
2.6 JAVA	34
2.6.1 Processos e <i>Threads</i>	34
3 DESENVOLVIMENTO	37
3.1 MODELAGEM DO SISTEMA	37

3.2 PROJETO	39
3.2.1 TrafficLight.....	40
3.2.2 ColorTrafficLight	41
3.2.3 TrafficLightManager	42
3.3 SIMULAÇÃO	43
4 RESULTADOS.....	50
4.1 SIMULAÇÃO COM SISTEMA ATUAL.....	50
4.2 SIMULAÇÃO COM SISTEMA PROPOSTO	51
5 CONCLUSÃO	53
5.1 PRINCIPAIS VANTAGENS	53
5.2 PONTOS FRACOS	53
6 TRABALHOS FUTUROS.....	54
REFERÊNCIAS.....	55

1 INTRODUÇÃO

Diante de uma era tão tecnológica, sempre visando resolver problemas antigos como, encontrar curas para doenças, descobrir inteligência extraterrestre, etc. convém observar que todo ser humano já se deparou com um problema peculiar que é “a falta de tempo”.

Tempo é um, se não o principal problema em que a humanidade vive hoje. Quem nunca teve que correr contra o relógio por algum motivo? Não teve tempo de abraçar um ente querido enquanto podia? Como ter tempo para resolver a falta de tempo? Quantas horas um dia precisaria ter? Todas estas perguntas podem ser resumidas em apenas uma. Como otimizar o tempo? Existem várias técnicas e ferramentas disponíveis hoje em dia, mas mais uma vez o tempo seria um empecilho então como mudar ao redor e otimizar o tempo, sem que ninguém precise aprender nada ou fazer algo de diferente?

Imaginem uma encruzilhada onde um semáforo de quatro tempos funciona normalmente em x unidades de tempo para cada lado, nas quais motoristas e pedestres devem respeitar as cores (vermelha, amarelo e verde) que determinam se podem prosseguir ou não. Surge uma simples pergunta “Porque estou parado aqui se não há fluxo em nenhuma das vias?”. Para o motorista a resposta poderia ser “para não ser autuado com multa como prevê o artigo 208 do Código de Trânsito Brasileiro” e para o pedestre simplesmente para se manter seguro.

Simplesmente pode-se desviar o semáforo contornando-o, ou talvez em vez disto reduzir o tempo do semáforo ou talvez o retirar do local, porque não.

Agora com as informações e padrões do trânsito pode-se elaborar um algoritmo que aprenda com o fluxo e que saiba quais os horários, dias e pontos onde o tráfego é maior.

A proposta deste estudo é analisar o fluxo de tráfego de um ponto estratégico ou uma cidade e coletar dados que mostrem em números o tempo total de parada em um semáforo. Com estes dados levantados, simulações de ambiente foram executadas e um algoritmo foi desenvolvido com intuito de estudar e aprender a otimizar o tempo levando em consideração as informações coletadas (simulações, reais) e também parâmetros como tempo, horário, entre outros.

1.1 OBJETIVOS

O presente trabalho tem como principal objetivo elaborar um software que permita a otimização do tempo de um ciclo de um semáforo.

1.1.1 Objetivos Específicos

São objetivos específicos deste trabalho:

- Analisar normas e regulamentações de trânsitos;
- Descrever o funcionamento do método atual;
- Desenvolver o software de acordo com os requisitos elencados;
- Testar o sistema atual e o proposto por meios de simulações de ambiente real.

1.2 JUSTIFICATIVA

Este trabalho justifica-se tendo em vista como principal objetivo a otimização do tempo o estudo deve levar em consideração a situação em muitas vezes não há fluxo em algumas das vias e mesmo assim deve-se esperar em um semáforo. Em muitos casos uma espera demasiada pode causar ansiedade.

“A ação de dirigir compreende um processo de influência mútua das funções psicológicas e cognitivas do homem. Diversos fatores estão vinculados à condução de um veículo, desde memória e atenção, até a tomada de decisões frente às situações comuns no trânsito como tráfego de pedestres e veículos, além de estímulos relacionados a fatores emocionais, que por sua vez influenciam o comportamento e a forma como o indivíduo conduz o veículo.” (LAÍS DE MEDEIROS, CRISTINA VASCONCELOS, *et al.*, 2018)

Afirma-se que “pelo menos 48% dos paulistanos gastam pelo menos 2 horas por dia no trânsito para realizar suas atividades diárias como por exemplo, trabalhar, estudar, ir ao supermercado, entre outros.” (IBOPE, 2015)

O ganho para a sociedade com o estudo é de extrema importância, como por exemplo ter mais tempo de lazer, estudos, trabalho e até mesmo melhorar sua saúde como aponta a pesquisa.

“Quanto tempo você demora para ir ou voltar do **trabalho**? De acordo com pesquisa da empresa britânica *VitalityHealth*, especializada em seguro médico privado, pessoas que passam **horas no trânsito**, seja no volante ou no transporte público, estão mais propensas ao **stress** e à **depressão**, além de enfrentar problemas no **sono** e na **produtividade**. O estudo foi feito em parceria com a Universidade de Cambridge.” (VEJA, 2017)

Este problema se torna tão comum que não é associado a uma dor de cabeça ou dores nas costas, mas é um problema sério e deve ser tratado como tal.

1.3 METODOLOGIA

Devido à natureza do problema, a primeira etapa realizada foi o estudo e análise das normas e regulamentações de trânsito. Tendo adquirido o conhecimento necessário sobre o assunto a ser abordado, iniciou-se o estudo das possíveis tecnologias adotadas no processo de desenvolvimento do projeto.

Enfim foi desenvolvido o software que controla o ciclo de um semáforo de forma a atender às especificidades do problema, como o tempo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo estão descritos os conceitos básicos de um semáforo, o software de simulação utilizado e a linguagem de programação aplicada.

2.1 FLUXO DE TRÁFEGO

O fluxo de tráfego também conhecido como volume de tráfego é o número de veículos que passam por determinado local em um dado de intervalo de tempo. (SILVA, 2007)

O controle e a operação semafórica podem ser realizados por diferentes tipos e estratégias sendo elas tempo fixo e atuado. (DENATRAN, 2014)

O tempo fixo basicamente utiliza planos semafóricos obtidos com base em dados de tráfego disponíveis. O controle pode ser feito baseado em um plano semafórico, ou em planos específicos.

O tipo atuado é principalmente dividido em semiatuado e totalmente atuado.

No tipo semiatuado é dada a preferência ao verde sempre para as vias principais, sendo modificada quando houver indicação de presença de veículos por detectores estrategicamente posicionados.

O tipo totalmente atuado ocorre no monitoramento do tráfego utilizando sensores em todas as direções, podendo ser alterado os tempos de estágios em tempo real.

2.2 SEMÁFORO

O semáforo é um subsistema de sinalização composto de indicações luminosas (VERDE, AMARELO e VERMELHO) acionadas ou intermitentes por meio de sistema eletrônico ou eletromecânico, com finalidade de transmitir mensagens aos usuários seja sobre o direito de passagem ou para alertar sobre situações de risco iminente.

O semáforo é uma invenção antiga, sendo que sua primeira instalação ocorreu em um cruzamento de Westminster em 1868. Muito tempo se passou e não houve modificações expressivas em seu funcionamento.

Seu funcionamento com duas ou três luzes teve início em 1914, em Cleveland com o intuito de organizar o trânsito para evitar acidentes. Funcionando de tal forma até hoje, seu sistema é utilizado em todo o mundo e em um mesmo padrão: verde para seguir, vermelho para parar, amarelo para atenção. (ALECRIM, 2016)

A finalidade do semáforo é transmitir aos usuários informações sobre o direito de passagem em interseções e/ou seções de via onde o espaço é disputado por dois ou mais movimentos conflitantes, ou advertir sobre a presença de situações na via que possam comprometer a segurança dos usuários (DENATRAN, 2014).

São dois os tipos de semáforos:

- O semáforo de regulamentação: Tem a função de controlar o trânsito através de sinalização luminosa, alternando a liberação das vias para veículos ou pedestres.
- O semáforo de advertência: Tem a função de alertar sobre a existência de obstáculo ou situação perigosa, devendo o condutor reduzir a velocidade e redobrando a atenção.

2.2.1 Sinal Verde

O sinal verde é o estágio em que o condutor ou pedestre obtém o direito de passagem.

O verde efetivo é o tempo de verde que é efetivamente utilizado pelo fluxo do grupo de movimento, obtido pela equação (DENATRAN, 2014),

$$t_{v,ef,t,i} = p_i \times t_c \quad (1)$$

Em que:

$t_{v,efet,i}$ = tempo de verde efetivo do estágio i , em segundos;

t_c = tempo de ciclo, em segundos;

p_i = fração de verde requerida para o estágio i .

O verde real é o tempo em que o estágio permanece verde em um ciclo, obtida pela equação (DENATRAN, 2014),

$$t_{v,real} = t_{v,efet} - t_{ent} + t_{pin} + t_{pfn} \quad (2)$$

Em que:

$t_{v,real}$ = tempo de verde real, em segundos;

$t_{v,efet}$ = tempo de verde efetivo, em segundos;

t_{ent} = tempo de entreverdes, em segundos;

t_{pin} = tempo perdido no início, em segundos;

t_{pfn} = tempo perdido no final, em segundos.

O tempo de verde efetivo para pedestres é igual ao tempo de verde real, obtido pela equação (2).

O entreverdes é o fim do intervalo de verde somado ao vermelho geral, com o propósito de evitar acidentes entre os usuários que estão perdendo seu direito de passagem e aqueles que vão passar a adquiri-lo no estágio seguinte. No caso dos veículos, compõe-se do intervalo de amarelo seguido do intervalo de vermelho geral, já para os pedestres consiste no intervalo de vermelho intermitente seguido do intervalo de vermelho geral.

O entreverdes para veículos se dá pela equação (DENATRAN, 2014),

$$t_{ent} = t_{pr} + \frac{v}{2(a_{ad} \pm ig)} + \frac{d_2 + c}{v} \quad (3)$$

Em que,

t_{ent} = tempo de entreverdes para o grupo focal de veículos em segundos;

t_{pr} = tempo de percepção e reação do condutor, em segundos;

v = velocidade do veículo, em m/s;

a_{ad} = máxima taxa de frenagem admissível em via plana, em m/s²;

i = inclinação da via na aproximação, sendo (+) em rampas ascendentes e (-) em rampas descendentes (m/m);

g = aceleração da gravidade (9,8 m/s²);

d_2 = extensão da trajetória do veículo entre a linha de retenção e o término da área de conflito, em metros;

c = comprimento do veículo, em metros.

O entreverdes para pedestres se dá pela equação (DENATRAN, 2014),

$$t_{ent} = t_{pr} + \frac{l}{v_p} \quad (4)$$

Em que:

t_{ent} = tempo de entreverdes para o grupo focal de veículos em segundos;

t_{pr} = tempo de percepção e reação do condutor, em segundos;

l = extensão da travessia, em metros;

v_p = velocidade do pedestre, em m/s.

2.2.2 Sinal Amarelo

O sinal amarelo é o tempo que se dá entre o fim do estágio verde até o início do estágio vermelho. É o estágio de atenção, obtido pela equação (DENATRAN, 2014),

$$t_{am} = t_{pr} + \frac{v}{2(a_{ad} \pm ig)} \quad (5)$$

Em que,

t_{am} = tempo de amarelo em segundos;

t_{pr} = tempo de percepção e reação do condutor, em segundos;

v = velocidade do veículo, em m/s;

a_{ad} = máxima taxa de frenagem admissível em via plana, em m/s²;

i = inclinação da via na aproximação, sendo (+) em rampas ascendentes e (-) em rampas descendentes (m/m);

g = aceleração da gravidade (9,8 m/s²).

2.2.3 Sinal Vermelho

O sinal vermelho é o estágio proibitivo no qual o condutor ou pedestre perde o direito de passagem, sendo o mais perigoso.

O vermelho geral é o intervalo de tempo do estágio amarelo até o início do estágio verde.

Dá-se pela equação (DENATRAN, 2014),

$$t_{vg} = \frac{d_2 + c}{v} \quad (6)$$

Em que,

t_{vg} = tempo de vermelho geral em segundos;

v = velocidade do veículo, em m/s;

d_2 = extensão da trajetória do veículo entre a linha de retenção e o término da área de conflito, em metros;

c = comprimento do veículo, em metros.

2.3 OPENSTREETMAP

OpenStreetMap é um mapa livre e editável do mundo. É construído por uma comunidade de mapeadores que contribuem e mantêm dados sobre estradas, trilhas, cafés, estações ferroviárias e muito mais, em todo o mundo (© OPENSTREETMAP CONTRIBUTORS, 2018).

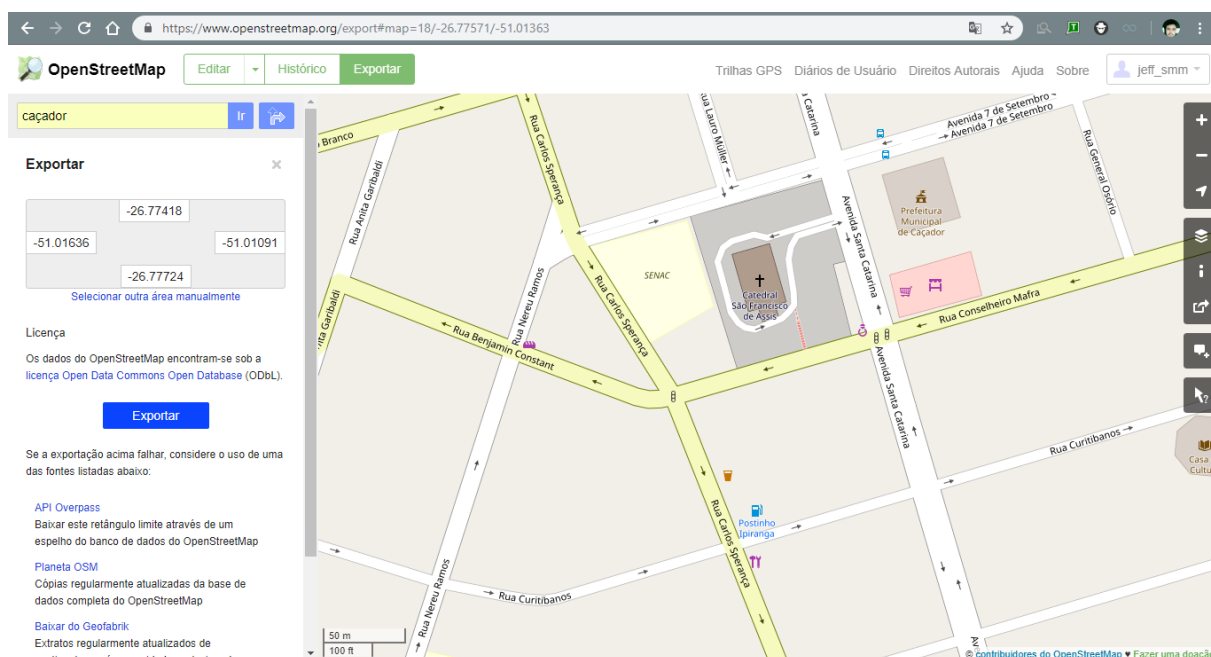


Figura 1: Mapa Senac Caçador

Fonte: (© OPENSTREETMAP CONTRIBUTORS, 2018).

2.4 SUMO

O SUMO é uma suíte de simulação de tráfego livre e aberta que está disponível desde 2001 (DLR, 2018). O SUMO permite a modelagem de sistemas de tráfego

intermodal, incluindo veículos rodoviários, transporte público e pedestres. Incluído no SUMO estão ferramentas de suporte que lidam com tarefas como localização de rotas, visualização, importação de rede e cálculo de emissões. O SUMO pode ser aprimorado com modelos personalizados e fornece várias APIs para controlar remotamente a simulação.

O SUMO tem sido usado em vários projetos para responder a uma grande variedade de questões de pesquisa como:

- Avaliação do desempenho dos semáforos, incluindo a avaliação de algoritmos modernos até a avaliação de planos de cronograma semanais.
- A escolha da rota do veículo, incluindo o desenvolvimento de novos métodos, a avaliação do roteamento ecologicamente consciente com base na emissão de poluentes e investigações sobre influências em toda a rede da escolha de rotas autônomas.
- Fornecimento de previsões de tráfego para as autoridades da cidade de Colônia durante a visita do papa em 2005 e durante a Copa do Mundo de Futebol de 2006.
- Apoio o comportamento simulado de telefonia em veículos para avaliar o desempenho da vigilância de tráfego baseada em GSM.
- É amplamente utilizado pela comunidade que desenvolve os protocolos V2X¹ para fornecer traços realistas de veículos e para avaliar aplicativos em um simulador de rede.

Algumas das ferramentas (DLR, 2018) existentes no pacote SUMO são:

- NETCONVERT;
- POLYCONVERT;
- RANDONTRIPS;
- SUMO;
- SUMO-GUI.

¹ O V2X é um conjunto de protocolos que permite a transmissão de dados de um veículo, para qualquer sistema que tenha o poder de afetar tal veículo ou vice-versa.

2.4.1 NETCONVERT

NETCONVERT é uma ferramenta do pacote SUMO usada para importar redes de estradas digitais de diferentes fontes e gerar redes rodoviárias, que podem ser interpretadas por outras ferramentas do pacote (DLR, 2018). O Quadro 1 mostra as opções.

Quadro 1: Opções utilizadas na ferramenta NETCONVERT

Opção	Descrição	Valor
--osm-files	Informa uma entrada OSM XML.	nome_arquivo.osm
-o	Informa a saída em NET XML.	nome_arquivo.net.xml
--output.street-names	Inclui os nomes das ruas.	true
--output.original-names	Mantem os nomes originais das ruas.	true
--no-turnarounds	Desabilita a criação de retornos.	
--tls.join	Agrupa nós próximos em um mesmo semáforo.	
--junctions.join	Agrupa junções próximas.	
--tls.guess-signals	Controla semáforos próximos em um mesmo controle.	
--osm.elevation	Leva em consideração se existe elevações.	

Fonte: (DLR, 2018)

2.4.2 POLYCONVERT

POLYCONVERT é uma ferramenta do pacote SUMO usada para importar formas geométricas (polígonos ou pontos de interesse) de diferentes fontes, convertendo-as em uma representação que pode ser visualizada usando SUMO-GUI (DLR, 2018). O Quadro 2 mostra as opções utilizadas.

Quadro 2: Opções utilizadas na ferramenta POLYCONVERT

Opção	Descrição	Valor
--net-files	Informa uma entrada NET XML	nome_arquivo.net.xml
-osm-files	Informa uma entrada OSM XML	nome_arquivo.osm
--type-file	Inclui os nomes das ruas	typemap.xml
-o	Informa a saída em POLY XML	nome_arquivo.poly.xml

Fonte: (DLR, 2018)

2.4.3 RANDONTRIPS

RANDONTRIPS é uma ferramenta do pacote SUMO usada para gerar viagens aleatórias (DLR, 2018). O Quadro 3 exibe as opções utilizadas.

Quadro 3: Opções utilizadas na ferramenta RANDONTRIPS

Opção	Descrição	Valor
-n	Informa uma entrada NET XML	nome_arquivo.net.xml
-e	Tempo em Segundos	86400
-l	Probabilidade de borda de peso por comprimento	

Fonte: (DLR, 2018)

2.4.4 SUMO

O software SUMO realiza a simulação de fluxo de tráfego microscópico, contínuo e discreto no tempo (DLR, 2018).

Possui como entradas obrigatórias:

- Uma rede rodoviária (net.xml);
- Um conjunto de rotas (rou.xml).

Entradas opcionais:

- Arquivos adicionais como semáforos, sinais de velocidade detectores de saída, etc.
- Arquivos de saída como relatórios.

A Tabela 4 mostra as opções:

Quadro 4: Opções utilizadas no *software* SUMO

Opção	Descrição	Valor
-n	Informa uma entrada NET XML.	nome_arquivo.net.xml
-r	Informa uma entrada ROU XML.	nome_arquivo.rou.xml
-a	Informa uma entrada adicional.	nome_arquivo.poly.xml
--tripinfo-output	Informa uma saída.	Trip_info.xml
--tripinfo-output.write-unfinished	Veículos que ainda estão em percurso após o tempo de simulação se encerrar serão incluídos no relatório.	True
-e	Informa duração da simulação em segundos.	86400
-l	Informa um arquivo de saída.	Trips_report.xml
--duration-log.statistics	Habilita dados detalhados dos veículos como (comprimento de rota, duração da viagem, tempo perdido...).	True
--no-step-log	Desativa a saída do console da etapa de simulação atual.	True

Fonte: (DLR, 2018).

2.4.5 SUMO-GUI

SUMO-GUI é uma ferramenta do pacote SUMO que fornece uma interface gráfica para o usuário (DLR, 2018).

2.4.6 Traffic Lights

Normalmente, a ferramenta NETCONVERT gera semáforos e programas para junções durante o cálculo das redes. Ainda assim, esses programas computados muitas vezes diferem daqueles encontrados na realidade. Para alimentar a simulação com programas reais de semáforos, é possível executar o SUMO / SUMO-GUI com definições adicionais do arquivo TLS (DLR, 2018).

Um arquivo TLS é composto pelos ciclos semaforicos (tempos, padrões, etc.).

Para alterar a estrutura de um arquivo TLS pode-se carregar novas definições para semáforos como parte de um arquivo adicional.

Opções de TLS (Quadro 5):

Quadro 5: Composição do arquivo TLS

Elemento	Atributo	Tipo de valor	Descrição
tlLogic	id	id (string)	Esse deve ser um código de semáforo existente no arquivo .net.xml. Normalmente, o id de um semáforo é idêntico ao id da <i>junction</i> .
	type	enum (static, actuated, delay_based)	O tipo de semáforo (durações de fase fixa (<i>static</i>), prolongamento de fase com base em intervalos de tempo entre veículos (<i>actuated</i>) ou na perda de tempo acumulada de veículos na fila (<i>delay_based</i>)).
	programID	id (string)	Este deve ser um novo nome de programa para o id do semáforo.
	offset	int	O deslocamento inicial do tempo do programa.
phase	duration	time (int)	A duração da fase.
	State	Lista de signal states	Os sinais de semáforo para esta fase.
	minDur	time (int)	A duração mínima da fase ao usar o tipo <i>actuated</i> .
	maxDur	time (int)	A duração mínima da fase ao usar o tipo <i>actuated</i> .

Fonte: (DLR, 2018).

A estrutura de um arquivo TLS é gerada automaticamente pela ferramenta NETCONVERT, ou através de um arquivo adicional (Figura 2).

```

<additional>
  <tlLogic id="0" programID="my_program" offset="0" type="static">
    <phase duration="31" state="GGggrrrrGGggrrrr"/>
    <phase duration="5" state="yyggrrrryyggrrrr"/>
    <phase duration="6" state="rrGGrrrrrrGGrrrr"/>
    <phase duration="5" state="rryyrrrrrryyrrrr"/>
    <phase duration="31" state="rrrrGGggrrrrGGgg"/>
    <phase duration="5" state="rrrryyggrrrryygg"/>
    <phase duration="6" state="rrrrrrGGrrrrrrGG"/>
    <phase duration="5" state="rrrrrryyrrrrrryy"/>
  </tlLogic>
</additional>

```






Figura 2: Composição do arquivo TLS



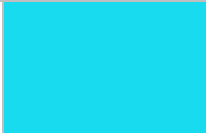
Fonte: (DLR, 2018)

Cada caractere dentro do estado de uma fase descreve o estado de um sinal do semáforo. Uma única faixa pode conter vários sinais - por exemplo, um para veículos virando à esquerda e outro para veículos que se movem em linha reta. Isso significa que um sinal não controla pistas, mas *links* - cada um conectando uma pista que está entrando em uma junção e uma que está saindo desta mesma junção (DLR, 2018).

As seguintes cores de sinal são usadas (Quadro 6),

Quadro 6: Cores de sinal

Personagem	Cor da GUI	Descrição
r		Os veículos devem parar.
Y		Os veículos começarão a desacelerar se estiverem longe da junção, caso contrário eles passarão.
g		Sem prioridade - os veículos podem passar a junção se nenhum veículo usar um maior fluxo antecipado, caso contrário eles desaceleram para deixar passar.
G		Prioridade - veículos podem passar a junção.
s		"Seta verde para a direita" requer parada - os veículos podem passar pela junção se nenhum veículo usar um fluxo de inimigos priorizado mais alto.

u		“Vermelho + luz amarela” para um sinal, pode ser usado para indicar a próxima fase verde, mas os veículos ainda não podem dirigir.
o		O sinal "off-blinking" é desligado, a luz intermitente indica que os veículos podem passar.
O		Sinal é desligado, os veículos têm o direito de passagem.

Fonte: (DLR, 2018).

No SUMO, uma dependência de um para n entre sinais e *links* é implementada, isso significa que cada sinal pode controlar mais de um *link* - embora as redes geradas pelo NETCONVERT usem normalmente um sinal por *link*. Um semáforo também pode controlar as faixas que entram em diferentes cruzamentos. As informações sobre qual *link* é controlado por qual sinal de semáforo podem ser obtidos com a opção "*show link tls index*" nas configurações de visualização do SUMO-GUI ou a partir do atributo *linkIndex* dos elementos *<connection>* no arquivo .net.xml (DLR, 2018).

A Figura 3 mostra um exemplo de semáforo com estado atual “GrGr”. A letra mais à esquerda “G” codifica a luz verde para o *link* 0, seguida de “r” para o *link* 1, “G” para o *link* 2 e “r” para o *link* 3. Os números de *link* são ativados via configurações SUMO-GUI ativando a opção “*show link tls index*” (DLR, 2018).

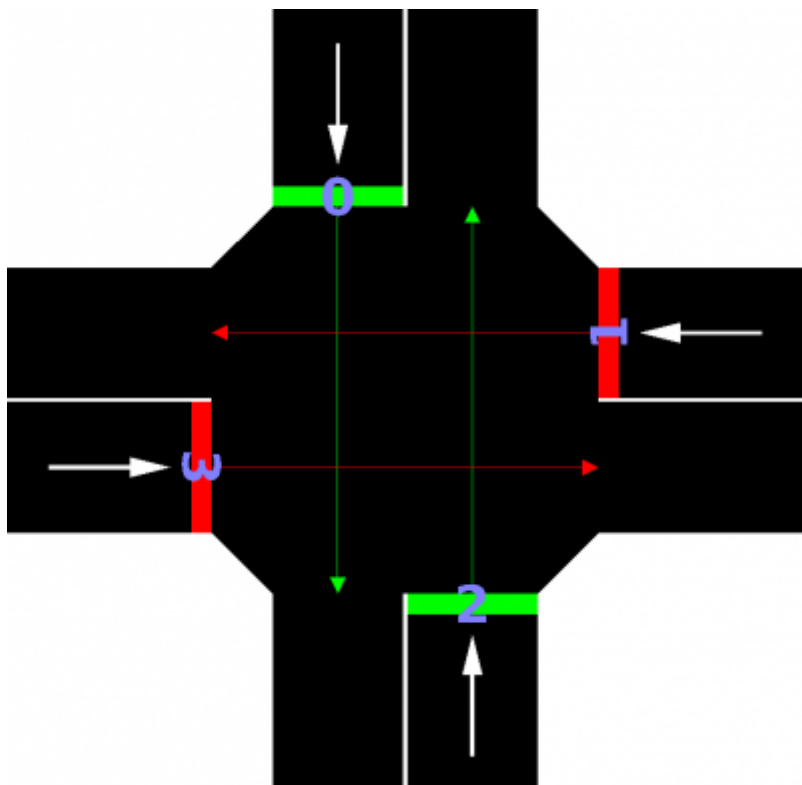


Figura 3: Exemplo de semáforo com estado atual "GrGr"

Fonte: (DLR, 2018).

Para semáforos que controlam uma única interseção, os índices gerados pelo NETCONVERT são numerados em um padrão no sentido horário. Travessias de pedestres são sempre atribuídas no final, também no sentido horário.

Se os semáforos são unidos de modo que um único programa controle múltiplas interseções, a ordenação para cada interseção permanece a mesma, mas os índices são aumentados de acordo com a ordem das junções controladas no arquivo de entrada.

2.4.7 Semáforo Baseado em Intervalos de Tempo (*Actuated*)

O SUMO suporta controle de tráfego atuado baseado em lacunas. Este esquema de controle é comum na Alemanha e funciona prolongando as fases de tráfego sempre que um fluxo contínuo de tráfego é detectado. Ele muda para a próxima fase depois de detectar um intervalo de tempo suficiente entre os veículos sucessivos. Isso permite uma melhor distribuição do tempo verde entre as fases e também afeta a duração do ciclo em resposta às condições dinâmicas de tráfego.

Para usar esse tipo de controle, o elemento `<tlLogic>` precisa receber o atributo `type = "actuated"` como ilustra a Figura 4. Também requer o uso dos atributos de fase `minDur` e `maxDur` em vez de duração para definir o intervalo permitido de durações de tempo para cada fase (se esses valores forem iguais ou somente a duração for dada, essa fase terá duração constante). Parâmetros extras podem ser usados para configurar o algoritmo de controle adicional (DLR, 2018).

```
<tlLogic id="0" programID="my_program" offset="0" type="actuated">
  <param key="max-gap" value="3.0"/>
  <param key="detector-gap" value="2.0"/>
  <param key="show-detectors" value="false"/>
  <param key="file" value="NULL"/>
  <param key="freq" value="300"/>

  <phase duration="31" minDur="5" maxDur="45" state="GGggrrrrGGggrrrr"/>
  ...
</tlLogic>
```

Figura 4: Exemplo de configuração `tlLogic actuated`

Fonte: (DLR, 2018).

O valor de `max-gap` descreve o intervalo de tempo máximo entre o veículo sucessivo que fará com que a fase atual seja prolongada (dentro dos limites). O valor `detector-gap` determina a distância do tempo entre o detector (gerado automaticamente) e a linha de parada em segundos (a cada velocidade máxima de cada pista). Se `show-detectors` estiver definido como `"true"`, os detectores gerados serão mostrados em SUMO-GUI. Os parâmetros `file` e `freq` têm o mesmo significado que para os detectores de `loop` de indução regulares. Os valores dos exemplos são os valores padrão para esses parâmetros (DLR, 2018).

2.4.8 Semáforo Baseado em Perda de Tempo (*Delay_based*)

Semelhante ao controle por intervalos de tempo entre veículos, um prolongamento de fase também pode ser desencadeado pela presença de veículos com perda de tempo (DLR, 2018). Um TLS com este tipo de atuação pode ser definido da seguinte forma como ilustra a Figura 5:


```

<tlLogic id="0" programID="my_program" offset="0" type="delay_based">
  <param key="detectorRange" value="100" />
  <param key="minTimeLoss" value="1" />
  <param key="file" value="NULL"/>
  <param key="freq" value="300"/>
  <param key="show-detectors" value="false"/>

  <phase duration="31" minDur="5" maxDur="45" state="GGggrrrrGGggrrrr"/>
  ...
</tlLogic>

```

Figura 5: Exemplo de configuração tlLogic delay_based

Fonte: (DLR, 2018).

Aqui, o *detectorRange* especifica a faixa de detecção a montante em metros medidos a partir da linha de parada. Por padrão (se o parâmetro for deixado indefinido), supõe-se que o detector E2 subjacente cubra completamente as primeiras pistas de aproximação. A perda de tempo de um veículo é acumulada assim que entra na faixa do detector. Se a perda de tempo acumulada exceder o valor de *minTimeLoss* (o padrão atual é um segundo), um prolongamento da fase verde correspondente é solicitado, se estiver ativo. A perda de tempo instantânea de um veículo é definida como $1 - v/v_{max}$, onde v é a velocidade atual e v_{max} a velocidade máxima permitida (DLR, 2018).

2.5 WAUT

Na prática as empresas costumam usar diferentes programas durante o dia e talvez também durante a semana e nos fins de semana. É possível carregar uma definição de tempos de comutação entre os programas usando uma configuração WAUT (DLR, 2018).

Dado um TLS que conhece quatro programas Figura 6 - dois para dias de semana e dois para fins de semana, das 22:00 às 6:00 o plano noturno deve ser usado e das 6:00 às 22:00 o plano do dia. Assim como programas já definidos, nomeados "*weekday_night*", "*weekday_day*", "*weekend_night*", "*weekend_day*". Para descrever o processo de troca, deve-se descrever o interruptor em um primeiro momento, supondo que a simulação seja executada da segunda-feira 00:00 (0s) até segunda-feira 00:00 (604.800s).

O Quadro 7 mostra as opções da ferramenta WAUT:

Quadro 7: Opções WAUT

Elemento	Atributo	Tipo de valor	Descrição
WAUT	id	id (string)	Nome do WAUT definido.
	refTime	int	Um tempo de referência que é usado como offset para os tempos de troca dados posteriormente (em segundos de simulação).
	starProg	id (string)	O programa que será usado no início da simulação.
WAUTSWITCH	time	Int	A hora em que a troca será realizada.
	to	id (string)	O nome do programa que os tfs designados devem mudar.

Fonte: (DLR, 2018).

```
<WAUT refTime="0" id="myWAUT" startProg="weekday_night">
  <wautSwitch time="21600" to="weekday_day"/>    <!-- monday, 6.00 -->
  <wautSwitch time="79200" to="weekday_night"/>  <!-- monday, 22.00 -->
  <wautSwitch time="108000" to="weekday_day"/>    <!-- tuesday, 6.00 -->
  ... further weekdays ...
  <wautSwitch time="453600" to="weekend_day"/>    <!-- saturday, 6.00 -->
  ... the weekend days ...
</WAUT>
```

Figura 6: Exemplo de configuração WAUT

Fonte: (DLR, 2018).

2.6 JAVA

A linguagem de programação Java™ é uma linguagem orientada a objetos concorrente, fortemente tipada e de uso geral. Normalmente, é compilada para um código intermediário *bytecode* e no formato binário definidos na especificação da Java *Virtual Machine* (ORACLE, 2018).

2.6.1 Processos e *Threads*

Na programação concorrente existem duas unidades básicas de execução: processos e *threads*. Na linguagem de programação Java, a programação

concorrente está principalmente relacionada a *threads*. No entanto, os processos também são importantes.

Um sistema de computador normalmente tem muitos processos e *threads* ativos. Isso é válido mesmo em sistemas que possuem apenas um único núcleo de execução e, portanto, possuem apenas uma *thread* em execução em um determinado momento. O tempo de processamento de um único núcleo é compartilhado entre processos e o tempo em que cada processo/*thread* for executado é chamado *time slice* (fatia de tempo) (ORACLE, 2017).

2.6.1.1 Processos

Um processo tem um ambiente de execução independente. Um processo geralmente possui um conjunto completo e privado de recursos básicos de tempo de execução, em particular, cada processo tem seu próprio espaço de memória.

Os processos geralmente são vistos como sinônimos de programas ou aplicativos. No entanto, o que o usuário vê como um único aplicativo pode, na verdade, ser um conjunto de processos cooperativos. Para facilitar a comunicação entre os processos, a maioria dos sistemas operacionais suporta recursos de IPC (Comunicação Entre Processos), como *pipes* e *soquetes*. O IPC é usado não apenas para comunicação entre processos no mesmo sistema, mas processos em sistemas diferentes (ORACLE, 2017).

A maioria das implementações da máquina virtual Java é executada como um único processo.

2.6.1.2 *Threads*

Threads são algumas vezes chamados de processos leves. Ambos os processos e *threads* fornecem um ambiente de execução, mas a criação de uma nova *thread* requer menos recursos do que a criação de um novo processo.

Threads existem dentro de um processo - cada processo tem pelo menos uma *thread*. *Threads* compartilham os recursos do processo, incluindo memória e arquivos abertos. Isso contribui para uma comunicação eficiente, mas potencialmente problemática.

A execução *multithread* é um recurso essencial da plataforma Java. Cada aplicativo tem pelo menos uma *thread* - ou várias, se contar *threads* "do sistema" que fazem funções como gerenciamento de memória e manipulação de sinais. Mas, do ponto de vista do programador de aplicativos, começa com apenas uma *thread*, chamada de *thread* principal (ORACLE, 2017).

3 DESENVOLVIMENTO

O projeto se baseia no desenvolvimento de um software que otimize o tempo de espera em um semáforo, e o compare através de simulação com o método atual.

Os métodos utilizados para desenvolvimento do sistema podem ser divididos em três etapas.

- Modelagem do sistema;
- Desenvolvimento do software de otimização;
- Execução das simulações.

3.1 MODELAGEM DO SISTEMA

A modelagem do sistema foi construída utilizando a ferramenta SysML que é uma linguagem de modelagem gráfica de propósito geral usada para especificar, analisar, projetar e verificar sistemas complexos que podem incluir hardware, software, informações, procedimentos e instalações (OMG, 2019).

A arquitetura do sistema pode ser visualizada através do diagrama de classe (Figura 7), que tem por objetivo descrever os tipos de objeto e seus relacionamentos, a perspectiva de implementação foi a utilizada neste modelo.

A classe *TrafficLightManager* é responsável por gerenciar quatro semáforos e organizá-los em sentido horário. Também compete ao gerenciador determinar qual semáforo tem a preferência de ativação. A classe *TrafficLight* é responsável por manter um ciclo entre os estados, sendo composta por três objetos da classe *ColorTrafficLight* (Verde, Amarelo e Vermelho) e por um objeto da classe *Sensor*.



Figura 7: Diagrama de classe

Fonte: Autor

Para representar o comportamento do semáforo foi utilizado o diagrama de estados também conhecido como diagrama de máquina de estados ilustrado na Figura 8. Ele tem como objetivo principal mostrar os estados e seus comportamentos mostrando os possíveis estados de um objeto e as transições responsáveis pelas suas mudanças de estado.

O processo é iniciado com a solicitação de permissão. Quando concedida é avaliado se o sensor está ocioso ou não. Se o sensor está ativo, passa para o processo de **siga**. Em seguida passa para o processo de **atenção** e finalmente para o processo de **pare** permanecendo em um ciclo.

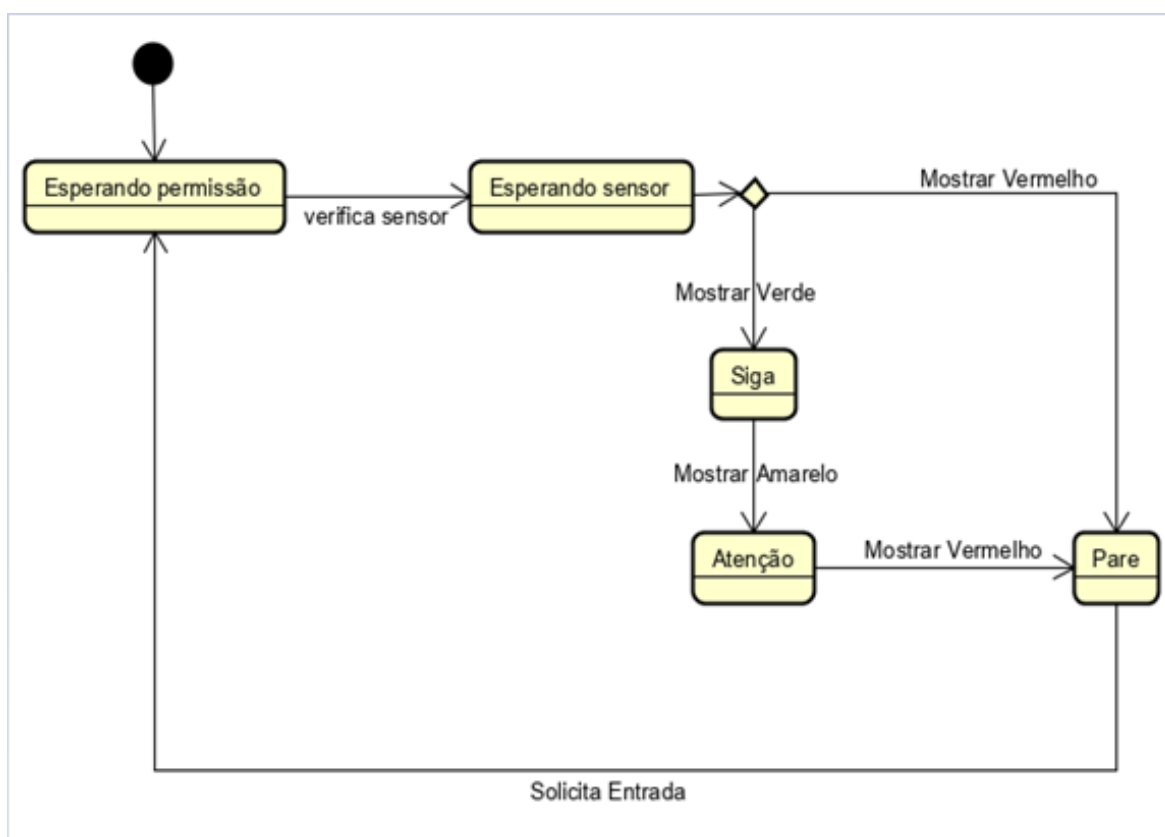


Figura 8: Diagrama de atividade

Fonte: Autor

3.2 PROJETO

O software do projeto foi desenvolvido utilizando a linguagem de programação Java e implementado em três classes.

- *TrafficLight*
- *ColorTrafficLight*
- *TrafficLightManager*

3.2.1 TrafficLight

A classe *TrafficLight* é responsável pela composição do semáforo. É nela que está o método *cycle*, responsável pelo ciclo do semáforo. A Figura 9 ilustra o código utilizado.

```
public void cycle() {
    for (Component child : panelTrafficLight.getComponents()) {
        if (child instanceof JButton) {
            if (currentState == 0) {
                ((JButton) child).setIcon(new ImageIcon(path + "green.png"));
                currentState = 1;
                rest(colorTrafficLight.getTimeGreen());
            }
            if (currentState == 1) {
                ((JButton) child).setIcon(new ImageIcon(path + "yellow.png"));
                currentState = 2;
                rest(colorTrafficLight.getTimeYellow());
            }
            if (currentState == 2) {
                ((JButton) child).setIcon(new ImageIcon(path + "red.png"));
                currentState = 0;
                rest(colorTrafficLight.getTimeRed());
            }
        }
    }
}
```

Figura 9: Método *cycle*

Fonte: Autor

O método *cycle* executa um *loop* no semáforo alterando o valor do estado. Cada estado é responsável por uma cor. No Quadro 8 pode-se observar os valores de cor de cada estado.

Quadro 8: Valores utilizados no método *cycle*

Valor	Cor	Descrição
0	Verde	O número zero é atribuído para o estado verde.
1	Amarelo	O número um é atribuído para o estado amarelo.
2	Vermelho	O número dois é atribuído para o estado vermelho.

Fonte: Autor

Quando o método executa a condição *if* o estado é alterado para a próxima etapa como ilustra a Figura 10, e assim sucessivamente até o ciclo ser encerrado.

```

if (currentState == 0) {
    ((JButton) child).setIcon(new ImageIcon(path + "green.png"));
    currentState = 1;
    rest(colorTrafficLight.getTimeGreen());
}

```

Figura 10: Parte do código onde ocorre a mudança dos estados

Fonte: Autor

O método *rest* é responsável pelo tempo (em segundos) que cada estado permanecerá em espera. Para implementá-lo foi utilizado o método *sleep* do pacote *Thread*. O método recebe como parâmetro o número de segundos ilustrado na Figura 11.

```

public void rest(int second) {
    try {
        Thread.sleep(second * 1000);
    } catch (InterruptedException e) {
        System.out.println(e);
    }
}

```

Figura 11: Método usado para colocar o sistema em espera

Fonte: Autor

3.2.2 ColorTrafficLight

A classe *ColorTrafficLight* é responsável por calcular o tempo de espera de cada estado. É nela que estão contidas as regras de negócios, ou seja, a codificação

das formulas que calculam o tempo dos estados verde, amarelo e vermelho como ilustra a Figura 12.

```
public int getTimeGreen() {
    if (i == 0) {
        return tpr + (v / (2 * (a_ad - g))) + (d2 + c) / v;
    }
    return tpr + (v / (2 * (a_ad + g))) + (d2 + c) / v;
}

public int getTimeYellow() {
    if (i == 0) {
        return tpr + (v / (2 * (a_ad - g)));
    }
    return tpr + (v / (2 * (a_ad + g)));
}

public int getTimeRed() {
    return (d2 + c) / v;
}
```

Figura 12: Métodos usados para calcular os tempos de espera

Fonte: Autor

Todas as fórmulas utilizadas estão discriminadas no capítulo 2.

3.2.3 TrafficLightManager

A classe *TrafficLightManager* é a responsável por gerenciar todo o sistema. Nela pode-se destacar o método *manager* (Figura 13). Seu funcionamento ocorre através de um laço que interage em cima dos quatro semáforos do gerenciador criando uma *Thread* para cada semáforo. A *Thread* executa uma condição *if* que verifica se o sensor está habilitado. Se a verificação retornar *true* ele executa o método *cycle*. Após ele incrementa a variável *position* (Figura 14) fazendo com que todos os semáforos participem. Neste mesmo método é verificado se o valor da variável *position* é igual ao número de semáforos se retornar *true* o valor da variável é alterado para zero.

```

public void manager() {
    for (int i = 0; i < numberTrafficLight; i++) {
        threads.add(new Thread(tg, listTrafficLight.get(i).getName()) {
            @Override
            public void run() {
                while (true) {
                    if (listTrafficLight.get(position).getSensor().isSelected()) {
                        listTrafficLight.get(position).printStatus();
                        listTrafficLight.get(position).cycle();
                    }
                    position++;
                    if (position == numberTrafficLight) {
                        position = 0;
                    }
                }
            }
        });
    }
}

```

Figura 13: Método *manager*

Fonte: Autor

```

position++;
if (position == numberTrafficLight) {
    position = 0;
}

```

Figura 14: Variável *position*

Fonte: Autor

3.3 SIMULAÇÃO

Para a etapa de simulação foi escolhida a ferramenta SUMO cujas características atendem aos requisitos elencados. Para que fosse possível realizar a simulação as seguintes etapas tiveram de ser executadas:

- Definição de uma rede rodoviária (NETCONVERT);
- Criação de formas geométricas (POLYCONVERT);
- Desenvolvimento de um conjunto de rotas (RANDOMTRIPS).

Para a utilização da ferramenta NETCONVERT foi preciso extrair um local específico do mapa do OpenStreetMap (Figura 15). A área foi escolhida por possuir um semáforo com as condições ideais para a simulação. Para utilizar o mapa é preciso

exportá-lo em um arquivo compatível com o simulador, neste caso em formato OSM. Este arquivo foi nomeado de cacador.osm.

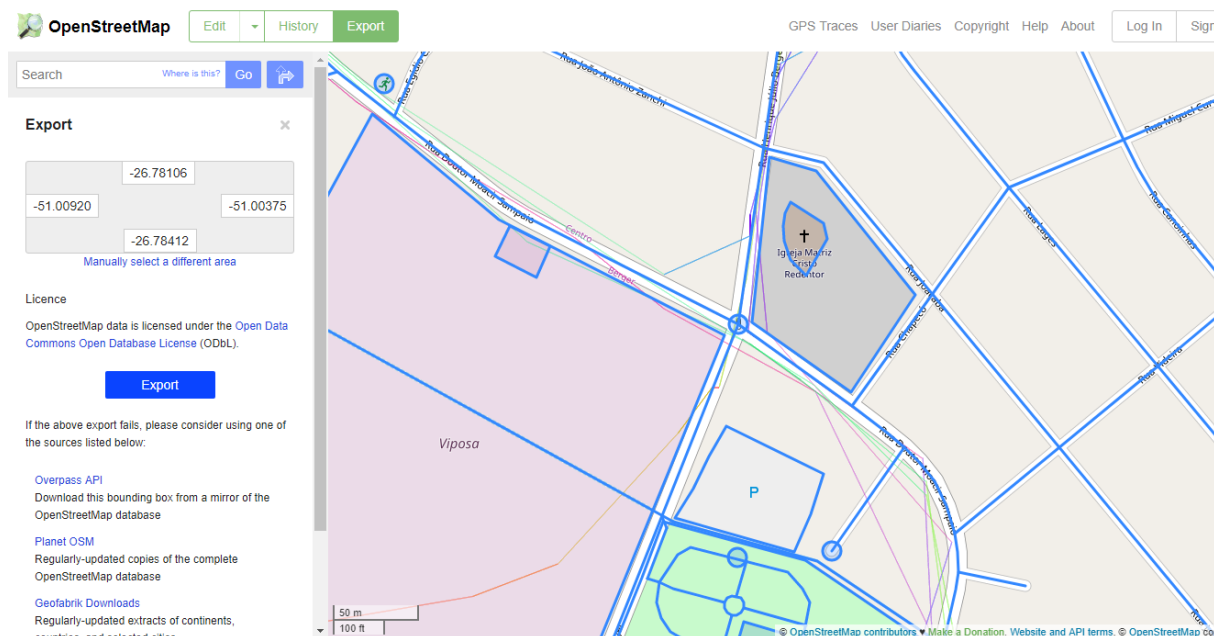


Figura 15: Mapa Viposa, Caçador – SC

Fonte: Autor

Com o arquivo obtido foi utilizado o comando NETCONVERT (Figura 16), com os seguintes parâmetros:

```
netconvert --osm-files cacador.osm -o cacador.net.xml --ptstop-output
cacador_stops.add.xml --ptline-output cacador_ptlines.xml --output.street-names true --
output.original-names true --no-turnarounds --proj.utm true --tls.discard-simple true --
tls.join true --tls.guess-signals true --tls.default-type actuated --geometry.remove
true --ramps.guess true --roundabouts.guess true --junctions.join true --
junctions.corner-detail 5 --crossings.guess true --osm.elevation
```

Figura 16: Comandos utilizados NETCONVERT

Fonte: Autor

Este comando retorna um arquivo convertido no formato que o simulador utiliza. Neste caso o arquivo retornado foi viposa_cacador.net.xml. Este arquivo contém informações como semáforos, ruas, encruzilhadas, entre outras.

Para a utilização da ferramenta POLYCONVERT é preciso configurar um arquivo com nome de *typemap.xml*. As configurações deste arquivo foram importadas do próprio desenvolvedor e mostradas na Figura 17. Este arquivo é utilizado para melhorar as formas gráficas.

Importing additional Polygons (Buildings, Water, etc.)

OSM-data not only contains the road network but also a wide range of additional polygons such as buildings and rivers. These polygons can be imported using [POLYCONVERT](#) and then added to a [sumo-gui](#) -configuration.

To interpret the OSM-data an additional *typemap*-file is required (the example below is identical to `<SUMO_HOME>/data/typemap/osmPolyconvert.typ.xml`):

```
<polygonTypes>
  <polygonType id="waterway"          name="water"          color=".71,.82,.82" layer="-4"/>
  <polygonType id="natural"           name="natural"       color=".55,.77,.42" layer="-4"/>
  <polygonType id="natural.water"      name="water"         color=".71,.82,.82" layer="-4"/>
  <polygonType id="natural.wetland"    name="water"         color=".71,.82,.82" layer="-4"/>
  <polygonType id="natural.wood"       name="forest"        color=".55,.77,.42" layer="-4"/>
  <polygonType id="natural.land"       name="land"          color=".98,.87,.46" layer="-4"/>

  <polygonType id="landuse"            name="landuse"       color=".76,.76,.51" layer="-3"/>
  <polygonType id="landuse.forest"     name="forest"        color=".55,.77,.42" layer="-3"/>
  <polygonType id="landuse.park"       name="park"          color=".81,.96,.79" layer="-3"/>
  <polygonType id="landuse.residential" name="residential"   color=".92,.92,.89" layer="-3"/>
  <polygonType id="landuse.commercial" name="commercial"    color=".82,.82,.80" layer="-3"/>
  <polygonType id="landuse.industrial" name="industrial"    color=".82,.82,.80" layer="-3"/>
  <polygonType id="landuse.military"   name="military"      color=".60,.60,.36" layer="-3"/>
  <polygonType id="landuse.farm"       name="farm"          color=".95,.95,.80" layer="-3"/>
  <polygonType id="landuse.greenfield" name="farm"          color=".95,.95,.80" layer="-3"/>
  <polygonType id="landuse.village_green" name="farm"          color=".95,.95,.80" layer="-3"/>

  <polygonType id="tourism"            name="tourism"       color=".81,.96,.79" layer="-2"/>
  <polygonType id="military"           name="military"      color=".60,.60,.36" layer="-2"/>
  <polygonType id="sport"              name="sport"         color=".31,.90,.49" layer="-2"/>
  <polygonType id="leisure"           name="leisure"       color=".81,.96,.79" layer="-2"/>
  <polygonType id="leisure.park"       name="tourism"       color=".81,.96,.79" layer="-2"/>
  <polygonType id="aeroway"           name="aeroway"       color=".50,.50,.50" layer="-2"/>
  <polygonType id="aerialway"          name="aerialway"     color=".20,.20,.20" layer="-2"/>

  <polygonType id="shop"               name="shop"          color=".93,.78,1.0" layer="-1"/>
  <polygonType id="historic"           name="historic"      color=".50,1.0,.50" layer="-1"/>
  <polygonType id="man_made"           name="building"      color="1.0,.90,.90" layer="-1"/>
</polygonTypes>
```

Figura 17: Configuração do arquivo typemap.xml

Fonte: (DLR, 2018)

Após sua configuração são utilizados os seguintes comandos (Figura 18).

```
polyconvert --net-file cacador.net.xml --osm-files cacador.osm --
osm.keep-full-type true --type-file typemap.xml -o cacador.poly.xml
--poi-layer-offset 5
```

Figura 18: Comandos utilizados POLYCONVERT

Fonte: Autor

Estes comandos retornam um arquivo denominado de cacador.poly.xml.

O próximo passo foi executar os comandos para criação de uma rota aleatória, para isto foi utilizada a ferramenta RANDOMTRIPS. Esta ferramenta permite que se

gere uma rota aleatória para cada veículo. Para isto ela utiliza o arquivo `cacador.net.xml` e gera quatro classes, uma para cada forma de veículo:

- *Motorcycle* (Figura 19).
- *Passenger* (Figura 20).
- *Truck* (Figura 21).
- *Bus* (Figura 22).

```
randomTrips.py -n cacador.net.xml --fringe-factor 1 -r cacador.motorcycle.rou.xml -o  
cacador.motorcycle.trips.xml -e 6828 --vehicle-class motorcycle --vclass motorcycle  
--prefix moto --max-distance 1200 --trip-attributes speedDev="0.1" departLane=  
"best" --validate
```

Figura 19: Comandos utilizados RANDOMTRIPS classe *Motorcycle*

Fonte: Autor

```
randomTrips.py -n cacador.net.xml --fringe-factor 1 -r cacador.passenger.rou.xml -o  
cacador.passenger.trips.xml -e 30197 --vehicle-class passenger --vclass passenger --  
prefix veh --min-distance 300 --trip-attributes speedDev="0.1" departLane="best  
" --validate
```

Figura 20: Comandos utilizados RANDOMTRIPS classe *Passenger*

Fonte: Autor

```
randomTrips.py -n cacador.net.xml --fringe-factor 1 -r cacador.truck.rou.xml -o  
cacador.truck.trips.xml -e 8757 --vehicle-class truck --vclass truck --prefix truck --  
min-distance 600 --trip-attributes speedDev="0.1" departLane="best" --validate
```

Figura 21: Comandos utilizados RANDOMTRIPS classe *Truck*

Fonte: Autor

```
randomTrips.py -n cacador.net.xml --fringe-factor 1 -r cacador.bus.rou.xml -o
cacador.bus.trips.xml -e 920 --vehicle-class bus --vclass bus --prefix bus --min-
distance 600 --trip-attributes departLane=\"best\" --validate
```

Figura 22: Comandos utilizados RANDOMTRIPS classe *Bus*

Fonte: Autor

Os valores utilizados na ferramenta RANDOMTRIPS, referentes à quantidade de veículos, foram obtidos através de consulta ao IBGE (IBGE, 2016).

Para que a simulação gerasse informações mais detalhadas sobre o semáforo em questão, foi criado um arquivo adicional denominado de `tls_rep.xml`, como ilustra a Figura 23.

```
<additional>

    <timedEvent type = "SaveTLSSwitchTimes"
        source = "2390821305" dest = "tls_reportProposto.xml" />

</additional>
```

Figura 23: Configuração arquivo `tls_rep.xml`

Fonte: Autor

A saída deste arquivo contém todas as informações do semáforo escolhido. Neste caso o semáforo foi escolhido por representar as condições ideais para a simulação. As opções podem ser verificadas no Quadro 9, a opção *source* teve seu valor foi extraído com ajuda da ferramenta SUMO-GUI, conforme pode ser verificado na Figura 24.

Quadro 9: Composição arquivo tls_rep.xml

Elemento	Atributo	Tipo de valor	Descrição
timedEvent	type	Enum (<i>SaveTLSStates</i> , <i>SaveTLSSwitchTimes</i> , <i>SaveTLSSwitchStates</i>)	Tipo do gatilho do evento.
	source	Id (string)	ID de um tls.
	dest	File name (string)	Arquivo para salvar estado.

Fonte: (DLR, 2018).

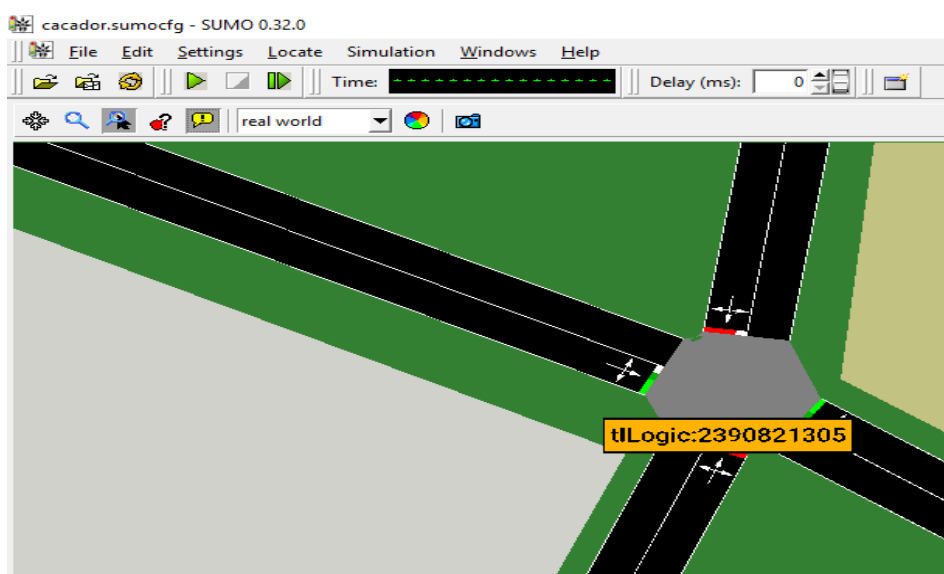


Figura 24: SUMO-GUI id semáforo

Fonte: Autor

A configuração do arquivo cacador.sumocfg conforme ilustra a Figura 25 dá-se através da junção dos arquivos:

- cacador.net.xml;
- cacador.passenger.trip.xml;
- cacador.truck.trip.xml;
- cacador.bus.trip.xml;
- cacador.motorcycle.trip.xml;
- cacador.poly.xml;
- cacador_stop.add.xml;
- tls_rep.xml.

Este arquivo funciona como um gerenciador e é ele que faz a chamada da simulação.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">

  <input>
    <net-file value="cacador.net.xml"/>
    <route-files value="cacador.passenger.trips.xml,cacador.truck.trips.xml,
      cacador.bus.trips.xml,cacador.motorcycle.trips.xml"/>
    <additional-files value="cacador.poly.xml,cacador_stops.add.xml,tls_rep.xml"/>
  </input>

  <processing>
    <ignore-route-errors value="true"/>
  </processing>

  <routing>
    <device.rerouting.adaptation-steps value="180"/>
  </routing>

  <output>
    <tripinfo-output value="C:\Users\Jefferson\Desktop\SUMO\Tripnfol.xml"/>
  </output>
</configuration>
```

Figura 25: Arquivo cacador.sumocfg

Fonte: Autor.

Por fim a ferramenta SUMO-GUI é utilizada passando para ela o arquivo cacador.sumocfg Figura 26.

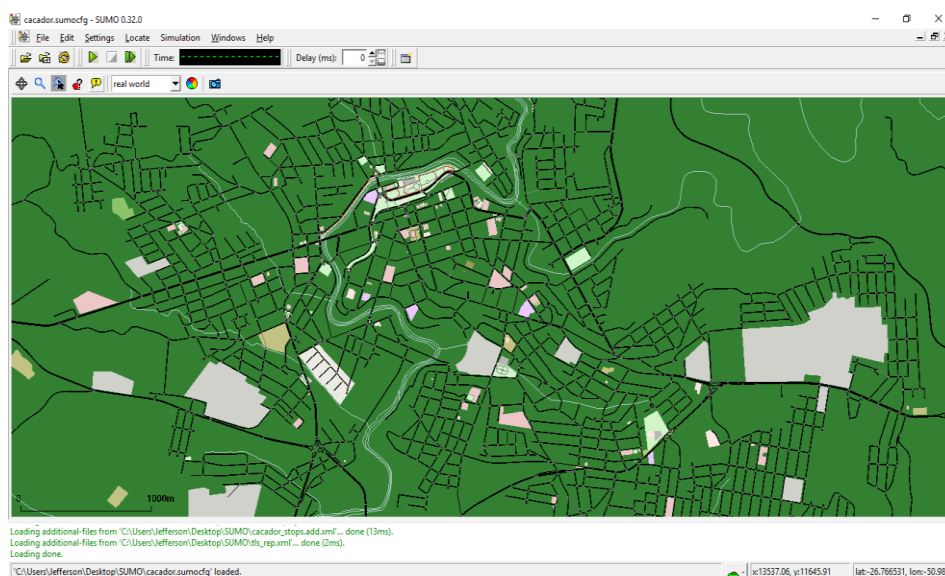


Figura 26: SUMO-GUI mapa de Caçador – SC

Fonte: Autor.

4 RESULTADOS

Neste capítulo é apresentada a simulação com os dados produzidos pelo *software*.

4.1 SIMULAÇÃO COM SISTEMA ATUAL

A simulação com o sistema semafórico atual utiliza os valores de vinte segundos para verde, três segundos para amarelo, com tipo *static* utilizando o padrão de oito fases conforme ilustra a Figura 27. Cada fase representa o estado atual de cada semáforo. O modelo em questão é representado por quatro semáforos.

Na primeira fase, a duração é de vinte segundos para verde no primeiro semáforo e vermelho nos restantes. Para a segunda fase o primeiro semáforo é alterado para amarelo com duração de três segundos os demais se mantêm em vermelho, e na terceira fase o primeiro semáforo fica vermelho e o próximo se inicia em verde se mantendo em um *loop*.

```
<tlLogic id="2390821305" type="static" programID="0" offset="0">
  <phase duration="20" state="gggrrrrrrrrr"/>
  <phase duration="3" state="yyyrrrrrrrrr"/>
  <phase duration="20" state="rrrggrrrrrrr"/>
  <phase duration="3" state="rrryyyrrrrrr"/>
  <phase duration="20" state="rrrrrrggrrrr"/>
  <phase duration="3" state="rrrrrryyrrrr"/>
  <phase duration="20" state="rrrrrrrrrggg"/>
  <phase duration="3" state="rrrrrrrrryyy"/>
</tlLogic>
```

Figura 27: tlLogic 2390821305 sistema atual

Fonte: Autor

Após executar uma simulação de 86400 segundos, equivalente a 24 horas, obtiveram-se os seguintes resultados: em 20h 48min (87%) do tempo o semáforo executou o estado de verde e em 3h 07min (13%) o estado amarelo. Todos os valores utilizados foram retirados do relatório gerado pela simulação com nome de *tls_reportAtual.xml*.

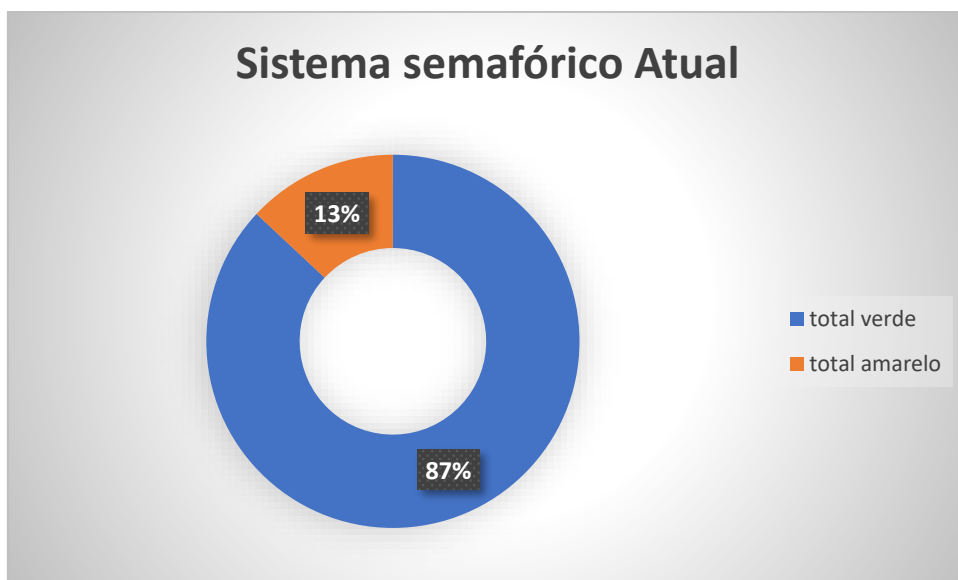


Figura 28: Sistema semafórico atual

Fonte: Autor

4.2 SIMULAÇÃO COM SISTEMA PROPOSTO

Para a simulação com o sistema semafórico proposto utilizou-se os valores escolhidos aleatoriamente pelo autor de vinte segundos para verde com duração mínima de cinco segundos e máxima de sessenta segundos, cinco segundos para amarelo, com tipo *actuated* utilizando o padrão de oito fases conforme ilustra a Figura 28. Cada fase representa o estado atual de cada semáforo, o modelo em questão é representado por quatro semáforos.

```
<tlLogic id="2390821305" type="actuated" programID="0" offset="0">
  <phase duration="20" state="gggrrrrrrrrrr" minDur='5' maxDur='60' />
  <phase duration="5" state="yyyrrrrrrrrrr" />
  <phase duration="20" state="rrrgggrrrrrr" minDur='5' maxDur='60' />
  <phase duration="5" state="rrrryyyrrrrrr" />
  <phase duration="20" state="rrrrrrgggrrrr" minDur='5' maxDur='60' />
  <phase duration="5" state="rrrrrryyyrrrr" />
  <phase duration="20" state="rrrrrrrrrggg" minDur='5' maxDur='60' />
  <phase duration="5" state="rrrrrrrryyy" />
</tlLogic>
```

Figura 29: tlLogic 2390821305 sistema proposto

Fonte: Autor

Após executar uma simulação de 86.400 segundos, equivalente a 24 horas, obteve-se os seguintes resultados, em 8h 38min (50%) do tempo o semáforo atual em estado de verde e nos 8h 42min (50%) em amarelo.

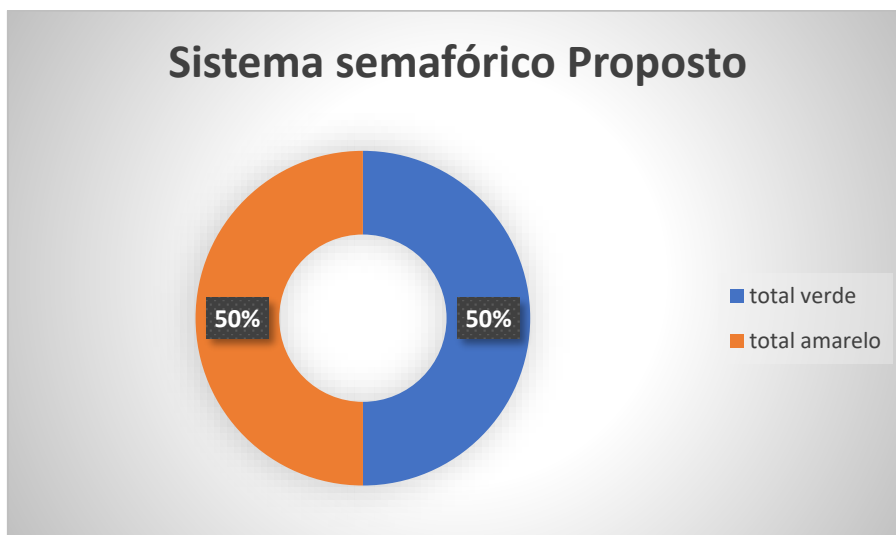


Figura 30: Sistema semafórico proposto

Fonte: Autor

Através das simulações realizadas comparou-se os resultados e chegou-se às seguintes conclusões: como ilustra a Figura 31, o sistema proposto teve redução significativa no tempo de espera (estado vermelho) em relação ao sistema atual. O sistema atual fornece tempo de verde em excesso, o que implica em espera excessiva e desnecessária das demais partes.

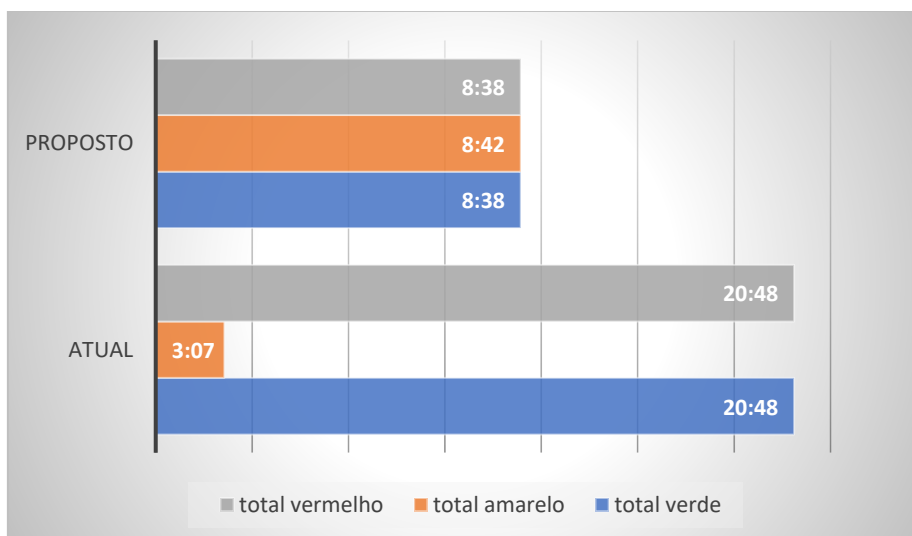


Figura 31: Comparação dos sistemas

Fonte: Autor

5 CONCLUSÃO

5.1 PRINCIPAIS VANTAGENS

Tendo em mãos os dados conclui-se que a principal vantagem em adotar o sistema proposto se dá na otimização do tempo do usuário. Sabendo que o sistema atual é antigo e defasado, analisando seus dados verifica-se um tempo de verde alto, porém o tempo de vermelho é o mesmo, o que significa tempo de verde sem uso e veículos em espera. No sistema proposto o valor chega a 58,88% de diferença ao sistema atual, significando menos tempo de espera enquanto que no sistema atual tem-se 20h 48min de tempo parado, no proposto totalizou apenas 8h 38min.

O sistema proposto atende ao requisito de otimização.

5.2 PONTOS FRACOS

O principal ponto a ser apontado é a falta de tempo para a aplicação de técnicas voltadas à visão computacional no sistema.

6 TRABALHOS FUTUROS

Como ações em potenciais, para trabalhos futuros pode-se apontar o aprimoramento no software para estender a utilização.

Outro fator importante a ser mencionado é referente a aplicação de visão computacional e possivelmente tratando de um sistema independente que gerencie e tome decisões como quanto ao sentido das vias, e comunicação com os veículos.

Por fim, outro trabalho futuro seria o desenvolvimento de uma interface mais amigável com possibilidade de análise em tempo real com sugestões de novas implementações nas vias urbanas como ruas, semáforos entre outras.

REFERÊNCIAS

- © OPENSTREETMAP CONTRIBUTORS. OPENSTREETMAP. **OPENSTREETMAP**, 2018. Disponível em: <<https://www.openstreetmap.org/about>>. Acesso em: 19 ago. 2018.
- ALECRIM, E. tecnoblog.net. **tecnoblog**, 2016. Disponível em: <<https://tecnoblog.net/199957/semaforos-realmente-inteligentes/>>. Acesso em: 04 abr. 2018.
- DENATRAN. Denatran. **Denatran**, 2014. Disponível em: <[https://www.denatran.gov.br/images/Educacao/Publicacoes/Manual_VOL_V_\(2\).pdf](https://www.denatran.gov.br/images/Educacao/Publicacoes/Manual_VOL_V_(2).pdf)>. Acesso em: 18 ago. 2018.
- DLR. **Institute of Transportation Systems**, 2018. Disponível em: <https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/>. Acesso em: 18 ago. 2018.
- DLR, S. SUMO. **SUMO SIMULATION OF URBAN MOBILITY**, 2018. Disponível em: <http://sumo.dlr.de/wiki/SUMO_User_Documentation>. Acesso em: 18 ago. 2018.
- IBGE. IBGE. **IBGE**, 2016. Disponível em: <<https://cidades.ibge.gov.br/brasil/sc/cacador/pesquisa/22/28120>>. Acesso em: 30 maio 2019.
- IBOPE. **IBOPE**, 28 AGOSTO 2015. Disponível em: <<http://www.ibopeinteligencia.com/noticias-e-pesquisas/paulistanos-gastam-em-media-2h38min-no-transito-para-realizar-suas-atividades-diarias/>>. Acesso em: 15 mar. 2018.
- LAÍS DE MEDEIROS, É. et al. ESTRESSE E COMPORTAMENTOS DE RISCO NO TRÂNSITO. **Temas em Saúde**, João Pessoa, v. 18, n. 1, p. 31-50, Abril 2018. ISSN 2447-2131.
- MING, S. H. cetesp. **cetesp**, 2010. Disponível em: <<http://www.cetesp.com.br/media/20797/nt212.pdf>>. Acesso em: 05 maio 2018.
- OMG. OMG SysML. **OMG Systems Modeling Language**, 2019. Disponível em: <<http://www.omgsysml.org/what-is-sysml.htm>>. Acesso em: 07 maio 2019.
- ORACLE. THE JAVA TUTORIALS. **ORACLE JAVA DOCUMENTATION**, 2017. Disponível em: <<https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html>>. Acesso em: 02 abr. 2019.
- ORACLE. ORACLE. **JAVA PROGRAMING LANGUAGE**, 2018. Disponível em: <<https://docs.oracle.com/javase/7/docs/technotes/guides/language/>>. Acesso em: 02 abr. 2019.
- PESSOA, F. frases famosas. **frases famosas**. Disponível em: <<https://www.frasesfamosas.com.br/frase/fernando-pessoa-podemos-vender-nosso-tempo-mas-nao/>>. Acesso em: 10 set. 2018.
- SILVA, P. C. M. D. www.sinaldetransito. **sinaldetransito**, julho 2007. Disponível em: <<http://www.sinaldetransito.com.br/artigos/teoria-do-fluxo-de-traffic.pdf>>. Acesso em: 25 abr. 2018.
- SUMO. SUMO. **SUMO DLR**, out. 2018. Disponível em: <<http://www.sumo.dlr.de/wiki/Networks/Import/OpenStreetMap>>. Acesso em: 15 jul. 2018.
- VEJA. Mundo Educação. **Veja**, 23 Maio 2017. Disponível em: <<https://veja.abril.com.br/saude/muito-tempo-no-transito-aumenta-o-risco-de-depressao/>>. Acesso em: 15 mar. 2018.

VILANOVA, L. sinaldetransito. **site de sinal de transito**, 03 jan. 2008. Disponível em: <<http://www.sinaldetransito.com.br/artigos/dimensionamento.pdf>>. Acesso em: 06 maio 2018.