UNIVERSIDADE FEDERAL DO PARANÁ

ALYSSON JHOVERT MALKO DE FREITAS
MARCOS YAMASAKI JUNIOR

DESENVOLVIMENTO E IMPLEMENTAÇÃO EM HARDWARE DE UM PROJETO DE CONTROLE DE TRÂNSITO BASEADO EM APRENDIZAGEM POR REFORÇO

CURITIBA
2017

ALYSSON JHOVERT MALKO DE FREITAS
MARCOS YAMASAKI JUNIOR

DESENVOLVIMENTO E IMPLEMENTAÇÃO EM HARDWARE DE UM PROJETO DE CONTROLE DE TRÂNSITO BASEADO EM APRENDIZAGEM POR REFORÇO

Trabalho de conclusão de curso apresentado ao Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados da Universidade Federal do Paraná como requisito à obtenção do título do grau de bacharelado em Engenharia Elétrica.

Orientadora: Prof. Dra. Sibilla B. da Luz França

CURITIBA
2017

TERMO DE APROVAÇÃO

ALYSSON JHOVERT MALKO DE FREITAS
MARCOS YAMASAKI JUNIOR

DESENVOLVIMENTO E IMPLEMENTAÇÃO EM HARDWARE DE UM PROJETO DE CONTROLE DE TRÂNSITO BASEADO EM APRENDIZAGEM POR REFORÇO

Trabalho de conclusão de curso apresentado como requisito parcial para à obtenção do grau de Engenheiro Eletricista com Ênfase em Sistemas Eletrônicos Embarcados, do Setor de Tecnologia da Universidade Federal do Paraná, pela seguinte banca examinadora:

_____

Profª Dra. Sibilla Batista da Luz França
Orientadora – Departamento de Engenharia Elétrica, UFPR

_____

Prof. Dr. Leandro dos Santos Coelho
Departamento de Engenharia Elétrica, UFPR

_____

Prof. Dr. André Bellin Mariano
Departamento de Engenharia Elétrica, UFPR

Curitiba, 5 de Dezembro de 2017.

Dedicamos este trabalho a nossa orientadora, por seus ensinamentos, paciência e confiança ao longo das supervisões dadas, e a nós mesmos pelo esforço realizado em conjunto e a coragem para tentar resolver desafios complexos em nossa sociedade.

AGRADECIMENTOS

Agradecemos primeiramente as nossas mães, que tiveram que se dedicar a criar seus filhos sem o amparo paterno e sempre nos ajudou nos momentos de cansaço e frustração durante toda a graduação. Agradecemos também a todos os professores que nos passaram o conhecimento necessário para compreender melhor o Universo que nos cerca e como modifica-lo para o bem da sociedade. Em especial gostaríamos de agradecer a professora Dra. Sibilla Batista da Luz França, nossa orientadora, pelo esforço dedicado em nos guiar não apenas neste projeto, mas durante toda a graduação, a professora Dra. Giselle Ferrari, pelos ensinamentos práticos e orientações profissionais, ao professor Dr. Leandro Coelho que sempre nos foi uma referência como pessoa e profissional, ao professor Dr. André Bellin Mariano que nos mostrou a importância em aplicar os conhecimentos técnicos aprendidos no curso para resolver os problemas da nossa sociedade, e ao professor Dr. João da Silva Dias, pelas incontáveis horas corrigindo nossa redação e fornecendo ideias para o projeto.

“Saber muito não lhe torna inteligente.
A inteligência se traduz na forma que você
recolhe, julga, maneja e, sobretudo, onde e
como aplica esta informação.”

Carl Sagan

RESUMO

O congestionamento do trânsito e as emissões de gases poluentes são os maiores problemas enfrentados nos dias atuais para a mobilidade urbana. Atualmente o brasileiro gasta em média 114 minutos nos deslocamentos diários. Portanto, o desenvolvimento de projetos de semáforos inteligentes é interessante para todos, pois tem como objetivo minimizar o congestionamento de veículos em intersecções, diminuindo o tempo de espera do motorista e por consequência, aumentando o fluxo de carros, desafogando a cidade de maneira inteligente, porque o sistema está sempre aprendendo a otimizar o trânsito. Este projeto tem por objetivo definir um método de aprendizado de máquina mais adequado para realizar o controle do ciclo de um semáforo de trânsito, desenvolver e simular em *software* o algoritmo de controle escolhido e implementar módulos do sistema em *hardware* FPGA (*Field Programmable Gate Array*) para validar os resultados obtidos em simulação. O aprendizado de máquina é uma das áreas da inteligência artificial mais relevante e que já conta com diversos algoritmos computacionais elaborados ao longo dos últimos anos. Um dos avanços importantes no aprendizado por reforço (*Reinforcement Learning*) foi o desenvolvimento de um algoritmo de controle conhecido como *Q-Learning*, onde o aprendizado é alcançado aplicando-se ações a partir de estados, onde é possível obter recompensa por uma boa ação e métodos de política para escolha da ação com maior retorno positivo. Este projeto apresenta duas abordagens para a melhor implementação do *Q-Learning*. A primeira abordagem consiste na elaboração de um exemplo para encontrar o melhor caminho em um labirinto e a segunda abordagem no desenvolvimento do método *Q-Learning* para controle de trânsito. Para execução deste projeto utilizou-se a ferramenta computacional MATLAB da empresa *Mathworks* para o desenvolvimento do algoritmo e simulações, e sua extensão HDL (*Hardware Description Language*) Coder para geração do código em VHDL (*Verilog Hardware Description Languge*). Nesta aplicação em *hardware* usou-se como objetivo uma FPGA Virtex 5 VLX50T. Foram obtidos os resultados esperados com as implementações em *hardware* das duas abordagens propostas. A primeira abordagem serviu de apoio e auxílio para uma futura implementação da segunda, mostrando como é mais simples começar com um problema menor e posteriormente implementar um algoritmo mais complexo. O FPGA foi utilizado pela possibilidade de prototipagem rápida e flexível. Uma abordagem de arquitetura paralela seria uma boa extensão deste projeto, assim como uma implementação de um contador volumétrico de veículo através de imagens em *hardware*.

Palavras-chave: *Q-Learning*. *Field Programmable Gate Array*. Trânsito. Controle. Inteligência Artificial. Veículos. Congestionamento.

ABSTRACT

Vehicle Traffic congestion and pollutant emissions are the biggest problems facing urban mobility today. Currently the Brazilians spend an average of 114 minutes in daily commutes. Therefore, the development of intelligent traffic light control projects is interesting for everyone, as it aims to minimize congestion of vehicles at intersections, reducing the driver's waiting time and consequently increasing the flow of cars, intelligently venting the city, because the system is always learning to optimize traffic. The objective of this project is to define a machine learning method that is most suitable to control the cycle of a traffic light, to develop and simulate the chosen control algorithm in software, and to implement system modules in Field Programmable Gate Array (FPGA) hardware, to validate the results obtained in simulation. Machine learning is one of the most relevant areas of artificial intelligence, and it already has several computational algorithms developed over the last few years. One of the most important advances in Reinforcement Learning was the development of a control algorithm known as *Q-Learning*, where learning is achieved by applying actions from a state, where it's possible to receive rewards by a good action and use policy methods to choose the action with most valuable action. This project presents two approaches for the best implementation of *Q-Learning*. The first approach consists of elaborating an example to find the best path in a labyrinth and the second approach in the development of the *Q-Learning* method for traffic control. For the execution of this project we used the MATLAB of Mathworks computational tool for the development of the algorithm and simulations, and its HDL Coder extension for VHDL code generation. In this hardware application, a Virtex 5 LX50T FPGA was used. The expected results were obtained with the hardware implementations of the two proposed approaches. The first approach served as support and assistance for the future implementation of the second, showing how it is simpler to start with a smaller problem and then implement a more complex algorithm. The FPGA was used for the possibility of fast and flexible prototyping. A parallel architecture approach would be a good extension of this design, as well as an implementation of a vehicle volumetric counter through hardware imaging.

Key-words: *Q-Learning*. Field Programmable Gate Array. Traffic. Control. Artificial Intelligence. Vehicle. Traffic Jam.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASIC	-	A*pplication-Specific Integrated Circuit*
CCO	-	Centro de Controle Operacional
CFTVs	-	Circuito Fechado de TVs
FIRJAN	-	Federação das Indústrias do Rio de Janeiro
FPGA	-	*Field Programmable Gate Array*
HDL	-	*Hardware Description Language*
LABs	-	*Logic Arrays Blocks*
LE	-	*Logic Elements*
LQF	-	*Longest Queue First*
LUTs	-	*Lookup Tables*
PTV	-	*Planung Transport Verkehr*
QLC	-	*Q-Learning Control*
RL	-	*Reinforcement Learning*
SRAM	-	*Static Random-Access Memory*
SUMO	-	*Simulation of Urban MObility*
TD	-	*Temporal Difference*
UDOT	-	*Utah Department of Transportation*
URBS	-	*Urbanização de Curitiba S/A*
VHDL	-	*Verilog Hardware Description Language*
VISSIM	-	*Verkehr In Städten SIMulationsmodell*
XOR	-	*Exclusive Or*

LISTA DE SÍMBOLOS

© - copyright

@ - arroba

® - marca registrada

Σ - somatório de números

Π - produtório de números

SUMÁRIO

1 INTRODUÇÃO

A mobilidade urbana é um dos temas com foco permanente de discussões no setor público e fora dele dado o caos que vem se transformando o trânsito de cargas e pessoas nas grandes cidades brasileiras. Dois fatores contribuíram decisivamente com a crise de mobilidade urbana vivenciada atualmente no mundo. Nos últimos 30 anos, a taxa de crescimento da população urbana no Brasil passou de 80 milhões em 1980, para 153 milhões de habitantes em 2008 (dados do IBGE – Instituto Brasileiro de Geografia e Estatística – e PNAD – Pesquisa Nacional por Amostra de Domicílio), e o crescimento do poder econômico dos consumidores nos últimos anos, ampliou o índice de veículos nas cidades.

Segundo o relatório "*Estado das Cidades da América Latina e Caribe"*, 80% da população latino-americana vive em centros urbanos e 14% (cerca de 65 milhões) habita metrópoles como São Paulo e Cidade do México (UN-HABITAT, 2012). Ocorre que esse aumento contínuo da população urbana não foi acompanhado de políticas de urbanização e infraestrutura que resolvessem questões como moradia e transporte (UOL – ATUALIDADES, 2012).

O congestionamento do trânsito e as emissões de gases poluentes são os maiores problemas enfrentados nos dias atuais para a mobilidade urbana. Segundo um estudo feito em 2015, pela FIRJAN (Federação das Indústrias do Rio de Janeiro) os prejuízos estão na casa dos 111 bilhões de reais anuais. O estudo revela que o brasileiro gasta em média 114 minutos nos deslocamentos diários. O Rio de Janeiro é a cidade que perde mais tempo no trânsito, com 141 minutos por dia em média. São Paulo está em segundo lugar com 132 minutos (FIRJAN, 2015).

A tendência é aumentar a frota de veículos nas vias públicas, aumentando o tempo de espera no trânsito. É visível que o Brasil está carente de uma solução para os congestionamentos no trânsito. Uma das soluções apresentadas em outros países, como nos Estados Unidos da América, em Utah (BARRY, 2014), utiliza-se um sistema inteligente de semáforos que se adapta de acordo com as condições do tráfego. O próprio sistema reprograma os tempos dos sinais verde e vermelho dependendo da situação do fluxo de veículo, fluxo de pedestres, tempo de espera, entre outros fatores, adaptando dinamicamente seu sistema (BARRY, 2014).

Existem sistemas sendo desenvolvidos no Brasil, mas em caráter experimental, como na cidade de Mogi das Cruzes, em São Paulo. Algumas ruas

obtiveram semáforos inteligentes que tem o controle adaptativo (PREFEITURA DE MOGI DAS CRUZES, 2015).

Portanto, o desenvolvimento de projetos de semáforos inteligentes é interessante para todos, pois tem como objetivo minimizar o congestionamento de veículos em interseções, diminuindo o tempo de espera do motorista e por consequência, aumentando o fluxo de carros, desafogando a cidade de maneira inteligente, porque o sistema está sempre aprendendo a otimizar o trânsito.

Em suma este projeto tem por finalidade implementar um algoritmo de inteligência artificial por métodos de aprendizado por reforço em FPGA (*Field Programmable Gate Array*) para um possível controle de semáforos de trânsito instalados em vias públicas, utilizando principalmente a ferramenta computacionais MATLAB® para desenvolvimento e simulações. O projeto irá se ater ao ambiente computacional, estendendo-se a uma co-simulação para validar os resultados obtidos via simulação e a ocupação em *hardware*.

1.1 JUSTIFICATIVA

Os sistemas de controle de trânsito com tempo fixo, não prevê a mudança de fluxo de veículos em horários diferentes ao que o sistema espera, o qual pode resultar em grandes congestionamentos nas cidades. O controle deve se adaptar as demandas do trânsito, para minimizar o tempo de espera de cada veículo. Entre as técnicas para minimizar o tempo de espera no trânsito por meio da mudança do tempo dos semáforos, as que se baseiam nos algoritmos de inteligência artificial apresentam resultados com menores tempos de espera em relação a outras técnicas. E entre as soluções com algoritmos de inteligência artificial, o método de aprendizado por reforço *Q-Learning* apresenta diversas referências com uma maior eficiência em comparação a outras soluções. Portanto, o método de aprendizagem por reforço *Q-Learning* aplicado à um sistema de controle de trânsito indica alta relevância econômica, social e ambiental, pois a diminuição do tempo de espera no trânsito resulta em grande economia em gastos públicos e na diminuição das emissões de gases poluentes dos veículos.

1.2 OBJETIVOS

O objetivo geral é desenvolver e implementar em um FPGA (*Field Programmable Gate Array)* blocos de um projeto de controle inteligente de semáforos de trânsito, utilizando método de Aprendizagem de Máquina.

1.2.1 Objetivos Específicos

Os objetivos específicos deste trabalho são os seguintes:

a) Definir o método de aprendizado de máquina mais adequado para realizar o controle do tempo dos semáforos;
b) desenvolver, e simular em *software* MATLAB® o algoritmo de controle inteligente escolhido;
c) refinar o algoritmo com parâmetros de aprendizagem buscando melhorias na eficiência;
d) analisar e concluir a viabilidade do algoritmo em aplicações reais;
e) implementar o método escolhido em *hardware* FPGA;
f) definir um ambiente para simulação de trânsito;
g) comparar os resultados obtidos em aplicações com tempo fixo.

2 REVISÃO DE LITERATURA

Os semáforos de trânsito atuais que normalmente são encontrados nas ruas funcionam de maneira limitada, pois são programados antecipadamente apenas baseados na análise histórica de tráfego do local. São utilizadas fórmulas matemáticas baseadas em fatores como o fluxo de carros por minuto para descobrir o ciclo ideal de cada cruzamento. Cada ciclo corresponde ao tempo que leva para completar os sinais de verde, amarelo e vermelho. Outros fatores considerados são: largura da rua, número de carros estacionados e presença de lombadas ou valas (PREFEITURA DE MOGI DAS CRUZES, 2015).

Os sistemas são reprogramados conforme a necessidade, sendo que alguns locais podem variar de três a cinco anos para a manutenção. Os ajustes são feitos nos tempos de sinal verde e sinal vermelho, dependendo do tráfego do local. Em corredores, vias que percorrem distâncias com diversos cruzamentos, busca-se a “onda verde”, que idealmente o veículo percorre todo o percurso sem parar em nenhum sinaleiro (PREFEITURA DE MOGI DAS CRUZES, 2015).

No Brasil, há sistemas de trânsito sendo desenvolvidos para otimizar o fluxo de veículos nas cidades, como aplicado Mogi das Cruzes no estado de São Paulo, que em certas interseções do município há monitoramento do tráfego local por câmeras instaladas nos semáforos, detectando o volume de carros. Com as informações obtidas pelas câmeras, o semáforo inteligente adapta os tempos de sinal verde e vermelho, reduzindo o tempo de espera do motorista (PREFEITURA DE MOGI DAS CRUZES, 2015).

Na cidade de Curitiba, Paraná, foi inaugurado em 2012 o CCO (Centro de Controle Operacional) que faz parte da empresa URBS (Urbanização de Curitiba) que controla o sistema de transporte público na cidade, tem como objetivo monitorar o trânsito intervindo quando necessário dependendo das circunstâncias. De acordo com o site da URBS, o sistema utiliza CFTVs (Circuitos Fechados de TV), PMVs (Painéis de Mensagens Variáveis), conectividade por fibra ótica e outros equipamentos e *softwares* específicos que estão instalados em diversos pontos da cidade. No entanto, a intervenção nos semáforos é feita manualmente por funcionários da empresa.

O sistema mais avançado de semáforos de trânsito em grande escala se localiza no estado de Utah nos Estados Unidos da América segundo Keith Barry, do site Citylab (2014). Os engenheiros de tráfego utilizam dados históricos de tráfego do estado para criar planos de sinalização para otimizar o “tempo de sinal verde”. Para melhorar a fluidez do trânsito utilizam sensores para detectar veículos em espera no sinaleiro, adotando diferentes estratégias para diferentes momentos do dia. Enquanto, em sistemas atuais de muitos lugares, os ajustes são revisados entre três a cinco anos, no *Utah Department of Transportation* (UDOT) os ajustes podem ser feitos em qualquer semáforo em todo o estado dentro de trinta segundos.

A chance de encontrar um sinal vermelho em Utah, segundo Lee Davidson do jornal The Salt Lake Tribune (2013), é de um em quatro, precisamente, 28%, ou seja, 72% de chance de não parar em um sinaleiro (BARRY, 2014).

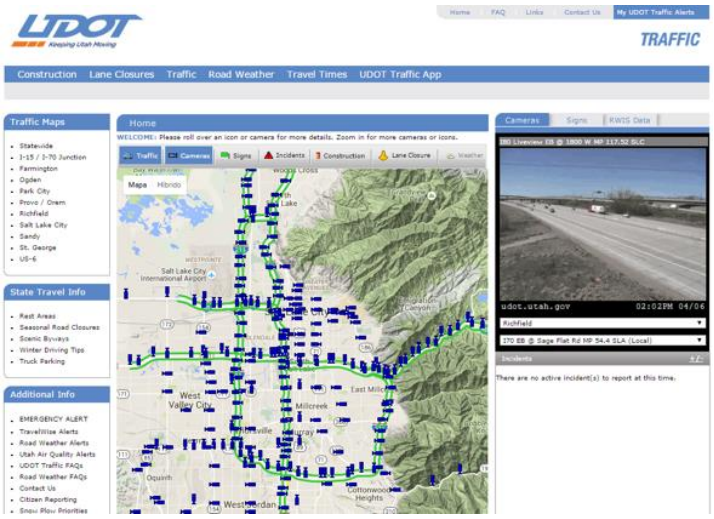
A comunicação dos semáforos é feita por uma rede de fibra ótica instalada por empresas parceiras. Os sensores utilizados que foram informados são câmeras de circuito fechado (BARRY, 2014).

Toda esta infraestrutura tem um custo alto, no entanto, o investimento tem uma razão de retorno de quarenta por um – não é surpreendente considerando que os gastos com congestionamentos custem mais de 120 bilhões de dólares por ano para os EUA, segundo Keith Barry do portal Citylab (2014).

Basicamente, o sistema utilizado em Utah, coleta dados do trânsito ao longo do tempo e otimiza dinamicamente o tempo de sinal verde e vermelho. Os ajustes no tempo são realizados em tempo real, o próprio sistema se adapta para melhorar a fluidez do tráfego de veículos.

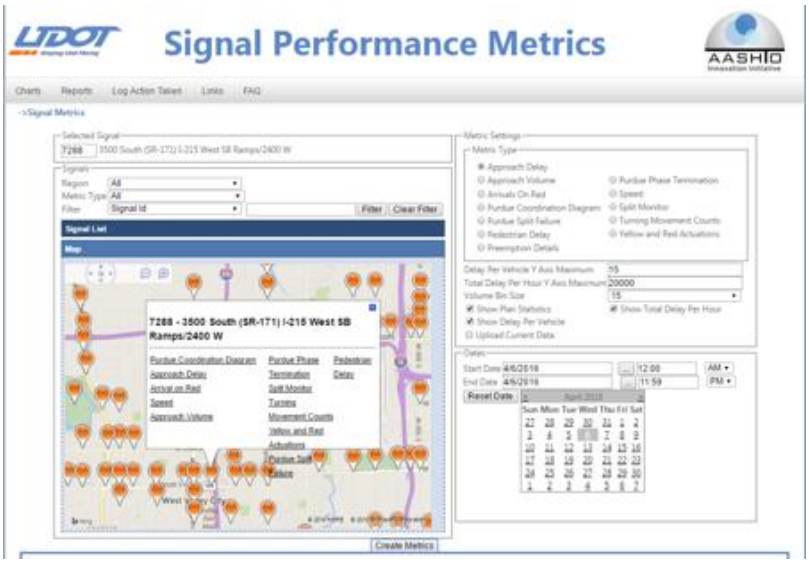
É possível verificar no site do UDOT *Traffic* (*Utah Department of Transportation*) as condições do tráfego no estado, navegando no mapa. É possível visualizar as imagens das câmeras instaladas nas vias, verificar acidentes, sinalizações, construções e condições do tempo. Também está aberto a comunidade, os dados de trânsito no estado, podendo escolher no mapa a interseção desejada e visualizar uma série de informações obtidas pelo sistema inteligente de semáforos de Utah (UDOT, 2016), como volume de veículos, tempo de espera dos veículos, tempo de espera dos pedestres, velocidade, entre outros, ilustrado na FIGURA 1 e na FIGURA 2. Inclusive, é possível gerar os gráficos dos dados de tráfego da interseção desejada, podendo escolher o horário e o dia. As informações são apresentadas em tempo real.

FIGURA 1 – SITE DE UDOT TRAFFIC (UTAH DEPARTMENT OF TRANSPORTATION)



FONTE: http://udottraffic.utah.gov/ (Acesso em 02/04/2016).

FIGURA 2 – SITE DE UDOT TRAFFIC (UTAH DEPARTMENT OF TRANSPORTATION) SIGNAL PERFORMANCE METRICS



FONTE: http://udottraffic.utah.gov/signalperformancemetrics/ (Acesso em 02/04/2016).

Uma alternativa barata, é a solução de Stephen Smith, professor do *Robotics Institute da Carnegie Mellon University* em Pittsburgh. A solução utiliza-se de uma rede de semáforos de trânsito inteligentes (SURTRAC; FINAL REPORT, 2014).

Ao invés de utilizar pessoas para monitorar e atuar no trânsito, os novos semáforos usam radares e câmeras para detectar o tráfego, e algoritmos sofisticados para aplicar instantaneamente ajustes no sistema baseado nas

condições do ambiente em tempo real. Segundo Smith (2014), a cada interseção é construído um plano para otimizar o tráfego local, e quando isso é feito, ele se comunica com os outros semáforos seguintes. O sistema conta com 49 *smart signals* (PITTSBURGH POST-GAZETTE, 2014).

A redução do tempo ao longo de corredores foi de 26 %, o tempo de espera caiu para 41% e as emissões dos veículos caíram 21% (SURTRAC; FINAL REPORT, 2014).

Já em Cambridge, Carolina Osorio do MIT (*Massachusetts Institute of Technology*), professora assistente de engenharia ambiental e civil, criou um elaborado modelo de computador de tráfego urbano que prediz como os motoristas irão se comportar, incluindo como eles irão reagir com as mudanças dos padrões do semáforo (BARRY; CITYLAB, 2014).

O algoritmo de Osorio pode programar automaticamente os sinais do semáforo, em particulares situações que a cidade prioriza, como por exemplo, aumento de fluxo de pedestres (BARRY, 2014).

No Brasil, ainda não há sistemas de otimização de tráfego em grande escala sendo implantados. As possíveis soluções ainda estão em fase experimental, como em São Paulo e Mogi das Cruzes. Em Curitiba, não há oficialmente semáforos inteligentes que se adaptam automaticamente dependendo das condições de trânsito, o que motiva uma implantação de um sistema inteligente experimental.

2.1 O APRENDIZADO DE MÁQUINA

O aprendizado é “qualquer processo no qual um sistema melhora seu desempenho através da experiência” (SIMON, 1954). Na natureza do aprendizado, pode-se inicialmente definir que o ser humano aprende interagindo com o seu ambiente. Quando uma criança brinca, mexe seus braços, ou pensa sobre o que está fazendo, não existe alguém que a esteja ensinando isso de maneira explicita, mas existe uma conexão sensorial e motora dela com o seu ambiente. Treinar este tipo de conexão gera um perfeito conceito sobre causa e efeito, as consequências de suas atitudes, e o que fazer, na ordem correta, para atingir algum objetivo. Quando alguém tenta aprender a dirigir ou aprende como estabelecer uma conversa com outra pessoa, fica claro como seu ambiente responde ao que é feito, e o ser humano sempre busca influenciar o que acontece através do seu comportamento.

Aprender através de interações é uma ideia fundamental que permeia todos as teorias de aprendizado e inteligência durante muito tempo (SUTTON, 1998).

O Aprendizado de Máquina é uma área da Inteligência Artificial relevante e já conta com diversos algoritmos computacionais elaborados ao longo dos últimos anos, abordagens como *Decision Tree Learning, Deep Learning, Support Vector Machines, Clustering, Reinforcement Learning, Genetic Algorithm, Bayesian networks* e entre outros. O objetivo do aprendizado de máquina pode ser descrito como programar computadores para aprender um determinado comportamento, dado exemplos ou observações (MITCHELL, 1997).

Uma das maneiras que um animal adquire comportamentos mais complexos é pelo aprendizado buscando obter recompensas e evitando punições. A teoria pelo aprendizado por reforço (*Reinforcement Learning*) é um modelo computacional deste tipo de aprendizado (WATKINS, 1989). O método de *Reinforcement Learning (RL)*, aprendizado pelo reforço, é uma das teorias de aprendizado para máquinas focadas em interações com o meio do que outras abordagens (BARTO, 1998).

2.2 O MÉTODO DE REINFORCEMENT LEARNING

O RL pode ser descrito como aprender o que fazer, como mapear situações para ações, de modo a maximizar um sinal numérico de recompensa. Não é dito para aquele que estará aprendendo quais ações tomar, como na maioria das formas de aprendizados de máquina, ao invés disto ele deve descobrir quais ações geram maior recompensa ao tenta-las. Nos mais desafiadores casos, ações podem afetar não só a recompensa imediata, mas podem interferir na próxima situação e assim todas as próximas recompensas (BARTO, 1998). Essas duas características, tentativa-e-erro e recompensa atrasada são duas das mais importantes características do método de RL, e mostram como isto pode melhorar o desempenho do problema proposto, controle de tráfego em tempo real.

O RL é definido não pela caracterização dos seus métodos de aprendizado, mas por caracterizar um problema de aprendizado (BARTO, 1998). Dependendo do contexto um método que se encaixa bem para resolver um problema, pode ser considerado um método de *RL*, a ideia básica é simplesmente capturar os aspectos mais importantes do problema real diante de um agente que está aprendendo a interagir com o seu ambiente para alcançar um objetivo (BARTO,1998).

O agente que está aprendendo deve ser capaz de saber o estado do ambiente em que se encontra e conseguir, através de ações, alterar o seu ambiente. O agente também deve ter um objetivo ou objetivos que tenham relação com o estado do ambiente. A formulação tende a incluir estes três aspectos: sensação, ação e objetivo, em suas mais simples formas (BARTO, 1998).

O *RL* é diferente de um método supervisionado (*supervised learning)*, onde as informações dos estados do ambiente são fornecidas ao agente de aprendizado, como reconhecimento de padrões estatísticos e redes neurais artificiais (BARTO,1998). O aprendizado supervisionado pode ser entendido como aprender por exemplos providos de um supervisor externo. Este é um importante tipo de aprendizado, mas sozinho não é adequado para o aprendizado pela interação com o ambiente, e não se aplicaria adequadamente a este projeto. Em um problema de interação muitas vezes não se consegue adquirir exemplos do comportamento desejado, que mostre quais as situações que o agente que está aprendendo deve interagir. Em um território desconhecido o agente que está aprendendo deve ser capaz de aprender pela sua própria experiência, e o método de RF oferece as ferramentas necessárias para este objetivo (SUTTON, 1998).

2.3 DESAFIOS DO MÉTODO

Um dos maiores desafios existentes no *RL*, e não em outros métodos de aprendizado é o equilíbrio entre exploração e aproveitamento. Para obter muita recompensa, o agente que está aprendendo deve ter preferência por ações que ele experimentou no passado e que foram efetivas produzindo recompensas. Mas para descobrir essas ações ele precisa experimentar ações que ainda não testou antes. O agente deve aproveitar o que ele já conhece para conseguir recompensas, mas também deve explorar para fazer melhores ações no futuro. O dilema é este, nem exploração ou aproveitamento podem ser usados exclusivamente, sem que haja uma falha na tarefa (BARTO, 1998).

O agente deve tentar uma variedade de ações e progressivamente favorecer aquelas que parecem a melhor opção. Na área da estocástica, cada ação deve ser tentada muitas vezes para obter uma estimativa confiável e a recompensa esperada. É interessante destacar que esse problema de balanceamento entre exploração e

aproveitamento não aparece no método de aprendizado supervisionado (BARTO, 1998).

O RL considera explicitamente todo o problema que um agente vai enfrentar para alcançar um certo objetivo interagindo com um ambiente desconhecido. Isto contrasta com os muitos outros métodos, que consideram seus subproblemas sem se importar como isso se encaixa em uma escala maior. O método de RL começa com um ambiente completo, interativo e um agente que sempre busca algum objetivo. Todo o agente de aprendizado tem um objetivo claro, podem sentir aspectos do seu ambiente, e escolher ações que influenciam o seu ambiente. Além disso, é normalmente assumido desde o princípio que o agente vai operar, apesar da incerteza significativa do ambiente que enfrenta (BARTO, 1998).

2.4 ELEMENTOS BÁSICOS DO REINFORCEMENT LEARNING

Formalmente um agente de RL encontra um problema de decisão de Markov (PDM), que possui quatro componentes básicos: estados, ações, e distribuições de transição e de recompensa. Como este método é dinâmico, ou seja, terá um aumento exponencial de ações durante o tempo (BELMANN, 1957), ele pode ser adaptado em quatro elementos: *a policy* (a política), *a reward function* (função de recompensa), *a value function* (função de valor) e um modelo de ambiente (BARTO, 1998).

2.4.1 A Política

A política define a maneira do agente de aprendizagem se comportar em um determinado ambiente. A política é um mapeamento de estados encontrados no ambiente para ações a serem tomadas nesses estados. Em alguns casos, a política pode ser uma função ou pesquisa de tabela simples, enquanto que em outros, pode envolver extensa computação, como um processo de busca. A política é o núcleo de um agente de aprendizagem, no sentido de que só ele é suficiente para determinar o comportamento (BARTO, 1998).

2.4.2 A Função De Recompensa

Uma função de recompensa define a meta em um problema de *RL*. Ou seja, ela mapeia cada estado observado do ambiente para um único número, uma recompensa, indicando o desejo intrínseco daquele estado. O único objetivo de um agente de aprendizagem é maximizar a recompensa total, que ele recebe em longo prazo. A função de recompensa define quais são os bons e maus eventos para o agente. A função de recompensa não deve ser alterada pelo agente. No entanto, pode servir como uma base para a alteração da política. Por exemplo, se uma ação selecionada pela política resulta em uma baixa recompensa, no futuro a política pode ser alterada para selecionar outra ação para melhorar a recompensa obtida. (BARTO, 1998).

2.4.3 A Função De Valor

Considerando que uma função de recompensa indica o que é bom imediatamente, uma função valor especifica indica o que é bom no longo prazo. De maneira geral, o valor de um estado é a quantidade total de recompensa que um agente pode acumular no futuro. As recompensas determinam o desejo imediato, intrínseco de estados do ambiente, já valores indicam o desejo de longo prazo dos estados, tendo em conta os estados que são propensos a seguir, e as recompensas disponíveis nesses estados. Por exemplo, um estado pode sempre produzir uma baixa recompensa, mas ainda tem um alto valor, pois é regularmente seguido por outros estados que produzem altas recompensas. Ou o inverso pode ser verdadeiro. Recompensas estão em primeiro lugar, enquanto que os valores, como previsões de recompensas, estão em segundo. Sem recompensas não poderia haver valores, e o único objetivo de estimar valores é conseguir mais recompensas. As opções de ações são feitas com base em juízos de valor. Geralmente buscam-se ações que provocam estados de maior valor, não mais alta recompensa, porque essas ações vão obter a maior quantidade de recompensa no longo prazo. Na tomada de decisão e planejamento, a quantidade de valor é aquela com a qual deve-se ficar mais preocupado. Recompensas são basicamente, dadas diretamente pelo ambiente, mas os valores devem ser estimados e reestimados a partir das sequências de observações que um agente faz ao longo de toda sua vida útil. É possível dizer que

o componente mais importante dos algoritmos de aprendizagem de reforço é um método eficiente para estimar valores (BARTO, 1998).

2.4.4 O Modelo De Ambiente

O quarto e último elemento é o modelo de ambiente. Pode ser entendido como algo que imita o comportamento do ambiente. Por exemplo, dado um estado e ação, o modelo pode prever o próximo estado resultante e próxima recompensa (BARTO, 1998).

2.5 APRENDIZAGEM DE DIFEREÇA TEMPORAL

O método de diferença temporal de aprendizagem (TD) é uma combinação de ideias de Monte Carlo e de programação dinâmica ideias (DP). Como métodos de Monte Carlo, o método TD pode aprender diretamente de experiência crua, sem um modelo de dinâmica do ambiente (SUTTON, 1998).

2.6 MÉTODO DE APRENDIZAGEM *Q-LEARNING*

Um dos avanços importantes no aprendizado por reforço foi o desenvolvimento de um algoritmo de controle conhecido como *Q-Learning* (WATKINS, 1989). A sua forma mais simples, um episódio de *Q-Learning*, é definido pela equação 2.1, em que a cada episódio o algoritmo incrementa valores em uma tabela de recompensas que servirá para a tomada de decisão do agente de aprendizado. Em outras palavras toda vez que o agente de aprendizado toma uma decisão escolhendo uma ação a ser tomada, é levado em consideração o valor da recompensa e o novo estado que vai depender da ação escolhida atualmente e da anterior, assim o valor de $\mathrm{Q(s,a)_i}$ é atualizado.

$$Q(s,a)_i \leftarrow Q(1-\alpha).Q(s,a)_{i-1} + \alpha[R(s,a)_i + \gamma \underbrace{max}_{a} Q(s',a')] \tag{2.1}$$

Onde Q é:

$s = Estado\ atual;$
$a = a\c{c}\tilde{a}o\ tomada\ no\ estado\ atual;$
$s' = pr\acute{o}ximo\ estado;$

$a' = ação\ tomada\ no\ próximo\ estado;$
$\alpha = taxa\ de\ aprendizado;$
$\gamma = fator\ de\ desconto.$

Para facilitar o entendimento a forma procedural do algoritmo de *Q-Learning* é mostrada no Quadro 1.

QUADRO 1 - FORMA PROCEDURAL ALGORITMO *Q-LEARNING*

Inicie $Q(s,a)$ arbitrariamente
Repita (para cada episódio):
    Inicie s
    Repita (para cada passo do episódio)
    Escolha um a de um s usando a política derivada de Q
    Tome a ação a, observe r, s'

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \underbrace{max}_{a} Q(s',a') - Q(s,a)]$$

$$s \leftarrow s'$$

até que s chegue ao fim.

FONTE: SUTTON, R., S.; BARTO, A., G. (1998).

Da equação 3.1 é possível analisar que $Q(1-\alpha).Q(s,a)_{i-1}$ é o valor antigo, $R(s,a)_i$ é o valor de recompensa, $maxQ(s',a')$ pode ser entendido com uma estimativa do melhor valor futuro.

A essência básica de *Q-Learning* é que o agente de aprendizado tem uma representação dos estados ***s*** do ambiente e possíveis ações nesses estados ***a***, e aprende o valor de cada uma dessas ações em cada um desses estados. Intuitivamente, este valor, *Q*, é referido como o valor de estado-ação. Assim, em *Q-Learning* é necessário começar definindo todos os seus valores de estado de ação para zero e o agente vai ao redor explorar o espaço de estado-ação. Depois de tentar uma ação em um estado, é avaliado o estado que levou a ação ***a***. Se esta ação levou a um resultado indesejável, é reduzido o valor *Q* (ou peso) para que outras ações tenham um valor maior e sejam escolhidas na próxima vez que o agente estiver nesse estado. Da mesma forma, se há uma recompensa por tomar uma ação particular, o peso dessa ação para esse estado é aumentado, assim o agente de aprendizado é mais propenso a escolhê-la novamente na próxima vez que você estiver nesse estado (BARTO, 1998).

2.7 APLICAÇÕES DE *Q-LEARNING* PARA CONTROLE DE SEMÁFOROS

A maioria dos métodos de controle de tráfego precisa-se de um modelo pré-definido de fluxo de tráfego para que tenham um curto tempo de resposta para predizer futuras condições. No método *Q-Learning* não é necessário nenhum tipo de modelo do ambiente e uma relação entre as possíveis ações, estados, e ambientes são aprendidos por interações com o ambiente. Neste contexto, foram Thorpe e Anderson (1996) que estudaram pela primeira vez métodos de *Reinforcement Learning* para controle de tráfego (WIERING e VREEKEN, et al., 2004).

Thorpe (1996) aplicou o método de SARSA (SUTTON, 1996) em um problema de controle de tráfego e avaliou o desempenho em três diferentes representações de um estado específico, ele usou algoritmos de redes neurais para estimar o valor de reforço. No seu estudo, estados eram definidos pelo número e posição dos veículos em todas as direções terminando em uma interseção. Uma ação para cada estado foi definida para mudar as cores de um semáforo de vermelho para verde e vice-versa. Essas características foram combinadas de três diferentes maneiras. A primeira delas foi chamada de "*Vehicle count*". Nesta abordagem Thorpe (1996) construiu dez partições baseadas no número de carros considerando todos os pares de combinação destas dez partições nas direções de leste a oeste e de norte a sul e também considerando dois possíveis modos para os semáforos, 200 (10x10x2) estados eram entradas para o agente de aprendizado.

Na segunda representação ou "distância fixa", Thorpe (1996) dividiu cada faixa em intervalos de 34 metros, o que gerou quatro partições em cada faixa. Um bit "ocupado" é definido para mostrar a ausência ou presença de veículos em cada partição, o que causa a existência de oito componentes para toda faixa oeste-leste ou norte-sul e um componente para a luz do semáforo. No total, existia um vetor de nove componentes para a entrada da rede neural. A terceira representação era a de "distância variável", de maneira similar a segunda representação, mas com uma distância variável para cada partição. As distâncias eram definidas em 15, 34, 67 e 122 metros, e novamente existem quatro partições em cada faixa e para o semáforo. O agente de aprendizado nesta representação tem nove componentes de entrada como na distância fixa. Thorpe (1996) definiu a recompensa para $r = -1$ em cada passo no percurso parra atingir o objetivo. A avaliação é feita em uma rede 4x4, e durante a avaliação o melhor resultado para simular os passos para limpar os

veículos do ambiente pertenciam às partições variáveis, e para o caso do tempo mínimo de viagem que teve o melhor resultado.

Wiering (2000) propôs uma transição de modelos para estimar o tempo de espera para os semáforos verdes e vermelhos em cada interseção. Ele aplicou um agente múltiplo de aprendizado para controlar os semáforos. Seu método era centrado no veículo, cada carro estima seu próprio tempo de espera e se comunica com o semáforo mais próximo. Para a definição de estado ele considerou a orientação e o posicionamento do veículo em uma fila, e o seu endereço de destino. A ação foi definida para mudar entre a luz vermelha e verde, a para a função de recompensa, se o carro permanece na mesma posição r = 1 e de outra maneira r = 0. Neste sistema o objetivo era minimizar o tempo total de espera, e a aprender a função de atribuição para estimar o tempo médio de todos os veículos. Durante seus experimentos, ambas as comunicações, local e global, eram levadas em consideração para chegar a uma melhor decisão no controle de semáforos.

Abdulhai (2003) aplicou *Q-Learning* como um controlador de semáforos de trânsito. Ele executou um experimento para uma interseção isolada, mas com alguns traços do método de agente múltiplo. Neste caso de interseção única, estados são comprimentos de filas em quatro abordagens fazendo uma conexão com a interseção e o tempo decorrido. Ações são definidas como uma extensão das luzes verdes e vermelhas ou uma mudança para a próxima a ação. Neste caso, a recompensa é considerada como uma penalidade e é o tempo total de atraso entre duas decisões sucessivas pelos veículos na fila formadas atrás da luz de parada de quatro abordagens conectando uma interseção. Além disso, uma função de potência foi usada para aproximar o equilíbrio do tamanho da fila alterando a recompensa, que é diretamente proporcional ao tamanho da fila em cada passo “*s*”. Isto é útil para que o agente de aprendizado não fique indiferente em uma fila muito grande, muito pequena ou ainda filas de tamanho igual. Para o caso de múltiplas interseções alguns outros estados como a divisão entre duas intercessões podem ser adicionados, e a recompensa seria ponderada pela soma de todas as interseções, considerando a maior recompensa para a estrada principal. Abdulhai (2003) mostrou que o *Reinforcement Learning* e especificamente o *Q-Learning* são uma promissora abordagem para um controle de semáforos. O seu resultado para uma interseção única mostrou que o *Q-Learning* superou o controle de variáveis de trafego pré-

definidas, e superou por pouco ou se iguala ao controlador pré-programado para situações de fluxo constante e uniformes.

Em Wunderlich et al. (2008), estabeleceram uma nova visão, o LQF (*Longest Queue First*), com um algoritmo de semáforos por agendamento para uma interseção isolada. O algoritmo LQF foi desenvolvido para um problema de controle de sinais e os conceitos foram empregados a partir do campo de comutação de pacotes em redes de computadores. Este método utiliza um algoritmo de união dos pesos máximos para minimizar o tamanho das filas em cada passo e levava ao menor atraso médio de um veículo através do cruzamento. Foi constatado que o LQF era estável e com um bom desempenho em variados cenários de controle de tráfego. Eles decidiram aplicar o LQF em uma rede de interseção múltipla em seu próximo estudo (AREL et al., 2010). Em uma rede de interseção múltipla uma decisão de agendamento de fase em uma única interseção afetaria muito as condições de tráfego em sua vizinhança de interseção e aplicando o LQF a tarefa se torna ainda mais difícil. Nesta pesquisa de *Reinforcement Learning*, é usada para dar a capacidade de ter controle distribuído conforme necessário para a programação de múltiplos cruzamentos. Na verdade, eles introduziram um novo uso do sistema de múltiplo agente e a estrutura do *Reinforcement Learning* para obter uma política de controle de tráfego eficiente.

Alguns outros trabalhos que Abdulhai teve contribuição que falam sobre o controle de tráfego são Abdi et al. (2012, 2013), Tantawy et al. (2013). Em Tantawy (2013), foi proposto um controle adaptativo de tráfego que emprega uma abordagem de aprendizado por reforço em múltiplos agentes. Cada controlador (agente) era responsável pelo controle do tempo de um semáforo em uma única junção de tráfego. Ele propôs dois modelos distintos: modo independente (1), onde cada controlador de interseção trabalha independentemente de outros agentes; e o modo integrado (2) onde cada controlador coordenava as ações com interseções vizinhas. Ele testou um modelo com uma rede de 59 interseções em uma parte da cidade de Toronto, Canadá, na parte da manhã em um horário de grande movimento. Os resultados mostraram uma redução no tempo de espera médio de 27% no modo 1, e de 39% no modo 2.

O grande desafio de todos os controles com *Q-Learning Control* (QLC) é administrar o enorme número de espaços de estado-ações. Uma das soluções para reduzir o número de espaços é categorizando possíveis estados em grupos. Além

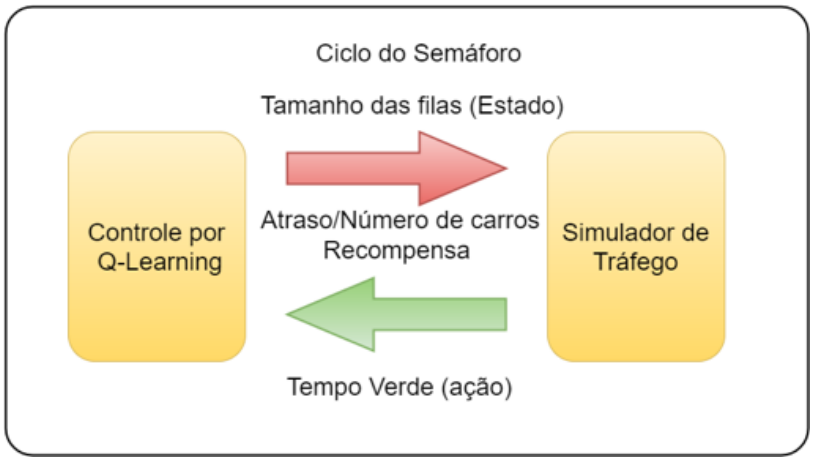
disso, esta abordagem aumenta a taxa de aprendizado, limitando o número de estados para o número de grupos diminuindo a precisão do sistema. A maioria das propostas de QLC atribui a uma ação o tempo verde do semáforo. Geralmente o tempo de espera é um período fixado, que pode se repetir até chegar em um *treshhold* máximo. Este período fixo pode diminuir a eficiência do sistema. Preparar informação suficiente para treinar o sistema pode gerar vários problemas para o QLC. *Q-Learning* sem treino suficiente não conseguem convergir para um resultado otimizado. No entanto, o *Q-Learning* é um método eficaz para adquirir um aprendizado em tempo real e é possível melhorar o seu desempenho conforme adapta-se a novas situações.

2.8 PROJETO DE UM QLC

Um modo de projetar um QLC pode ser feito em tabelas (ARAGHI, 2015). Estados são formados a partir da média do tamanho das filas.

O processo de interação entre o QLC e um Simulador de tráfego pode ser vista na FIGURA 3. Os tamanhos das filas que formam o ambiente são enviados ao QLC e é proposto um tempo verde para cada semáforo. Os tempos propostos de semáforos verdes são selecionados pela lista de ações pré-definidas do método *Q-Learning*.

FIGURA 3 - O PROCESSO DE INTERAÇÃO ENTRE O QLC E UM SIMULADOR DE TRÁFEGO



FONTE: Adaptado de ARAGHI, S. et al. (2015)

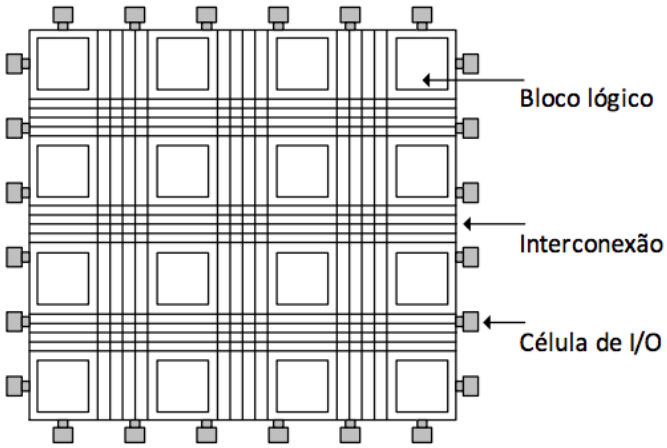
O QLC é método tabular de *Q-Learning* e existe um número limitado de conjunto de ações neste controle. Um conjunto de ações é uma combinação de tempos verdes no semáforo em cada fase. O tempo cíclico é variável e baseado na demanda do tráfego.

A recompensa é definida como inversamente proporcional ao tempo de atraso médio ao final de cada ciclo para todas as vias que se interceptam. Isso significa que existe um valor maior para casos com um menor tempo médio de atraso.

2.9 COMPUTAÇÃO RECONFIGURÁVEL

A computação reconfigurável tem como objetivo combinar o desempenho do *hardware* com a flexibilidade do *software*. A principal diferença entre os circuitos do tipo ASIC e os circuitos reconfiguráveis, como os FPGAs, é justamente a possibilidade de adaptar o *hardware* através da reprogramação, não sendo necessária a fabricação de um novo dispositivo a cada mudança na arquitetura do circuito integrado. Se comparados com microprocessadores tradicionais, nos FPGAs é possível desenvolver e embarcar uma arquitetura com a capacidade de fazer alterações no próprio fluxo de dados e no fluxo de controle. Num FPGA, a implementação de milhões de operações com recursos distribuí- dos é realizável, fazendo com que sejam mais rápidos (SILVA, 2016).

FIGURA 4 - ESTRUTURA INTERNA FPGA



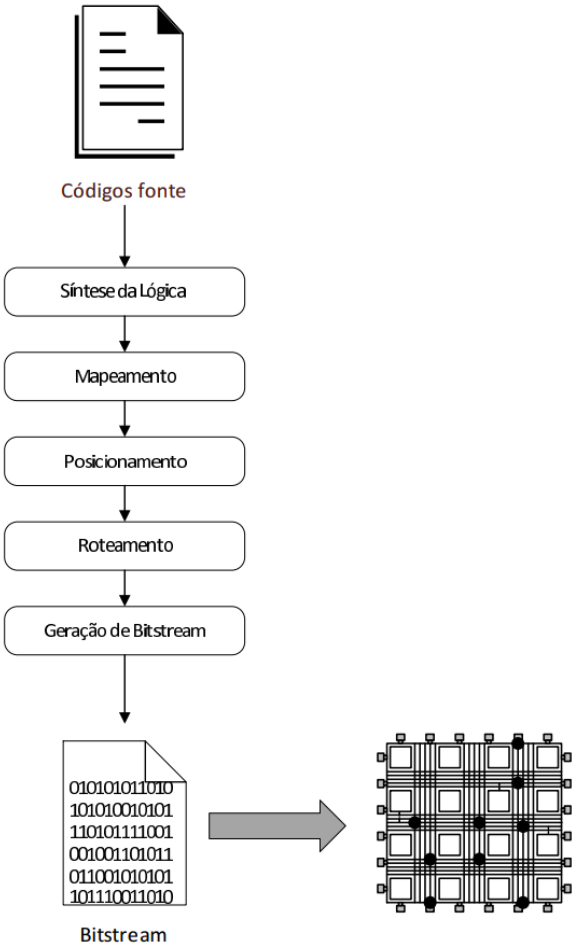
FONTE: SILVA LUCILEIDE M. D. et al (2016)

Um FPGA (ou *Field Programmable Gate Array*) é um circuito integrado desenhado para ser configurado pelo usuário após a sua fabricação. A FIGURA 4 ilustra a estrutura interna do FPGA, que é constituído por um conjunto de blocos lógicos embarcados numa estrutura de roteamento flexível para fazer sua conexão e células de entrada e saída, onde é possível programar blocos lógicos e suas interligações correspondentes de maneira a implementar a aplicação desejada (SILVA, 2016).

O FPGA é formado por elementos lógicos que podem ser configurados para formar circuitos específicos e implementar algoritmos em *hardware* afim de alcançar melhoras expressivas na performance.

O FPGA é majoritariamente composto por LABs (*Logic Arrays Blocks*) dispostos em matrizes conectadas por estruturas programáveis de roteamento (blocos de interconexão). Cada LAB contém vários LEs (*Logic Element*) que são blocos lógicos formados por LUTs (Lookup Tables), Flip-Flops ou registradores, e alguns circuitos adicionais como, por exemplo, *carry logic*, para prover uma maior funcionalidade ou flexibilidade. As LUTs são constituídas por uma árvore de multiplexadores que tem como entrada uma matriz de elementos de memória. Dependendo do tipo de dado escrito no elemento de memória durante sua configuração, o elemento lógico pode realizar qualquer tipo de função lógica combinacional desejada. Desde as complexas até a simples portas AND ou XOR. Já o registrador (ou Flip-Flop) permite que o elemento lógico realize funções lógicas sequenciais. As LUTs, blocos de interconexão, e todas as demais funções programáveis do FPGA são controladas por bits de controle das SRAMs. O FPGA precisa ser configurado antes de ser utilizado. Significa dizer que os dados precisam ser escritos nas SRAMs para que seja programada sua funcionalidade. Como as SRAMs são regraváveis, os FPGAs podem ser programados para se adaptar a diferentes tipos de aplicação (TANG 2016). A criação de um circuito baseado em FPGA é um processo de criação de um fluxo de dados em formato binário (*bitstream*) que será carregado no dispositivo, como pode ser observado na FIGURA 5.

FIGURA 5 - FLUXO DE DESENVOLVIMENTO EM FPGA



FONTE: SILVA LUCILEIDE M. D. et al (2016)

Existem ferramentas para realizar esta configuração, desenvolvidas pelos fabricantes dos FPGAs, sendo a configuração do FPGA geralmente especificada por uma linguagem de descrição de *hardware* (*Hardware Description Language - HDL*). A partir do HDL é feita a síntese lógica que converte um código comportamental de alto nível em portas lógicas. Em seguida a tecnologia de mapeamento separa as portas em grupos que melhor correspondem aos recursos lógicos do FPGA. O posicionamento (*placement*) atribui os agrupamentos lógicos para blocos lógicos específicos e o roteamento determina os recursos de interconexão que irão transportar os sinais. Finalmente com a geração do *bitstream*, um arquivo binário que define todos os pontos de programação do FPGA, os blocos lógicos e de roteamento são configurados apropriadamente (SILVA, 2016).

2.10 HDL CODER

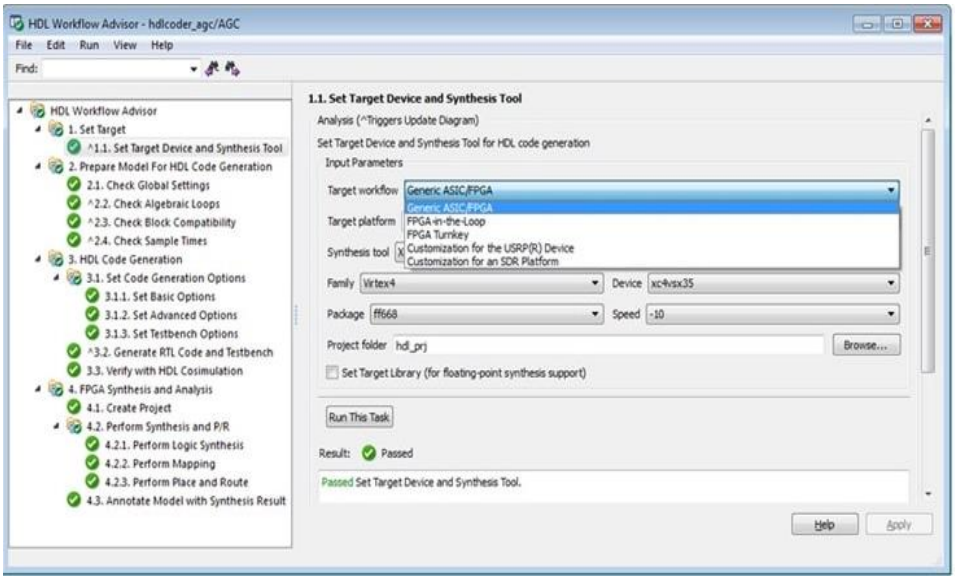
HDL Coder® é uma toolbox do MATLAB®, que realiza a geração de códigos sintetizáveis de VHDL e Verilog de funções do MATLAB® e modelos do Simulink®, resultando em um código mais genérico, que pode ser implementado para design de FPGAs e ASCIs (MATHWORKS, 2017).

O HDL Coder® oferece um orientador de fluxo de trabalho chamado Workflow Advisor, que automatiza a programação de FPGAs. É possível controlar a arquitetura do HDL e a sua implementação, destacar o caminho crítico e gerar estatísticas de estimação de consumo de recurso de *hardware*. Ele oferece cerca de 200 blocos do Simulink®, e com isso é possível construir a comunicação necessária para se conectar com o *hardware*, criar o processamento lógico de sinais como modelo de arquivo Simulink® apenas arrastando e jogando blocos no ambiente de desenvolvimento. É também possível converter números do tipo ponto-flutuante em ponto-fixo.

As funções escritas em código MATLAB® também podem ser integradas no design e para ver os resultados via simulação, pode ser usado diversas ferramentas do MATLAB®, incluindo scopes, displays, etc. Os blocos do Simulink® podem ser utilizados para gerar sinais de teste para o design. A FIGURA 6 mostra um pouco da interface do Workflow Advisor e a FIGURA 7 mostra o fluxo de trabalho com o HDL Coder® (NUTAQ, 2016).

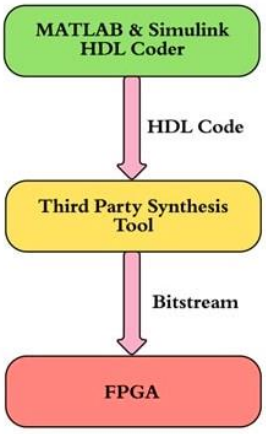
O HDL Coder® também gera *test benches* para verificar o design do HDL, no entanto, é preciso utilizar um *toolbox* adicional chamada HDL Verifier. Já se precisar gerar o código HDL sintetizado e gerar o *bitstream* para implementação em FPGA, é requerido utilizar ferramentas como o Xilinx ISE e Vivado ou Altera Quartus. Depois de sintetizado e gerado o *bitstream* na FPGA, é possível continuar utilizando o Workflow Advisor e realizando as etapas de integração em um mesmo ambiente (Simulink®) para todos os processos. Uma alternativa para geração do *bitstream* é utilizar a ferramenta HDL Verifier, que também possibilita a cossimulação e o *FPGA-in-the-loop* com as placas da Xilinx e Altera, com isso apenas com as ferramentas oferecidas pelo MATLAB® e Simulink®, é possível implementar os sistemas e modelos diretamente sem a necessidade de utilizar *softwares* terceiros (MathWorks, 2017).

FIGURA 6 – *WORKFLOW ADVISOR*



FONTE: NUTAQ (2017)

FIGURA 7 – FLUXO DE TRABALHO PARA IMPLEMENTAÇÃO EM *HARDWARE* FPGA



FONTE: NUTAQ (2017)

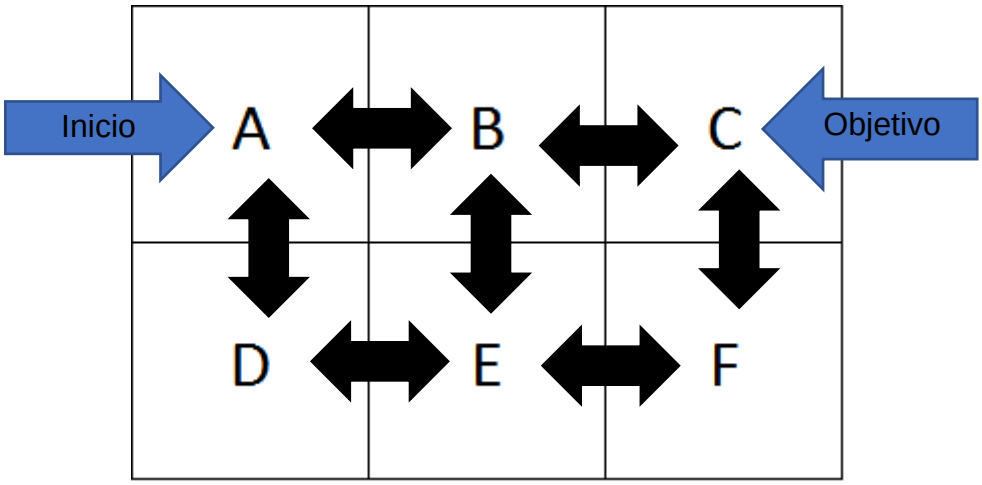
3 MATERIAIS E MÉTODOS

Foram desenvolvidas duas abordagens para a melhor implementação do *Q-Learning:* A primeira abordagem foi elaborar um exemplo para encontrar o melhor caminho e a segunda abordagem foi o desenvolvimento do método *Q-Learning* para controle de trânsito.

3.1 EXEMPLO DE UTILIZAÇÃO DO *Q-LEARNING*

Nesta abordagem, foi realizada uma implementação no *software* MATLAB® para solução de um problema de “melhor caminho”, com o uso do método de *Q-Learning*. O problema consiste em sair do ponto A e encontrar o melhor caminho até o ponto C, seguindo as normas impostas em cada ponto, ou seja, de A apenas pode-se ir para B e D, de B pode-se ir para E, C e A, e assim por diante, conforme apresentado na FIGURA 8.

FIGURA 8 – PROBLEMA PARA ENCONTRAR O MELHOR CAMINHO



FONTE: dos autores

Este problema foi escolhido, pois, demonstra um problema básico de aprendizado de máquina e que ajuda no entendimento do método de *Q-Learning* demonstrando todos os pontos básicos importantes para implementação de um projeto mais complexo.

Para começar definiu-se valores de recompensa para cada ação em cada estado, os estados serão os pontos A, B, C, D, E e F, e as ações serão os caminhos que os agente pode tomar, por exemplo, de A ir para B. Os valores das ações estão descritos na TABELA 1.

TABELA 1 – ESTADOS E AÇÕES DO PROBLEMA PROPOSTO

Estado	Ações	Recompensas
A	Ir para B	0
	Ir para D	0
B	Ir para A	0
	Ir para C	100
	Ir para E	0
D	Ir para A	0
	Ir para E	0
E	Ir para B	0
	Ir para D	0
	Ir para F	0
F	Ir para C	100
	Ir para E	0

Fonte: dos autores

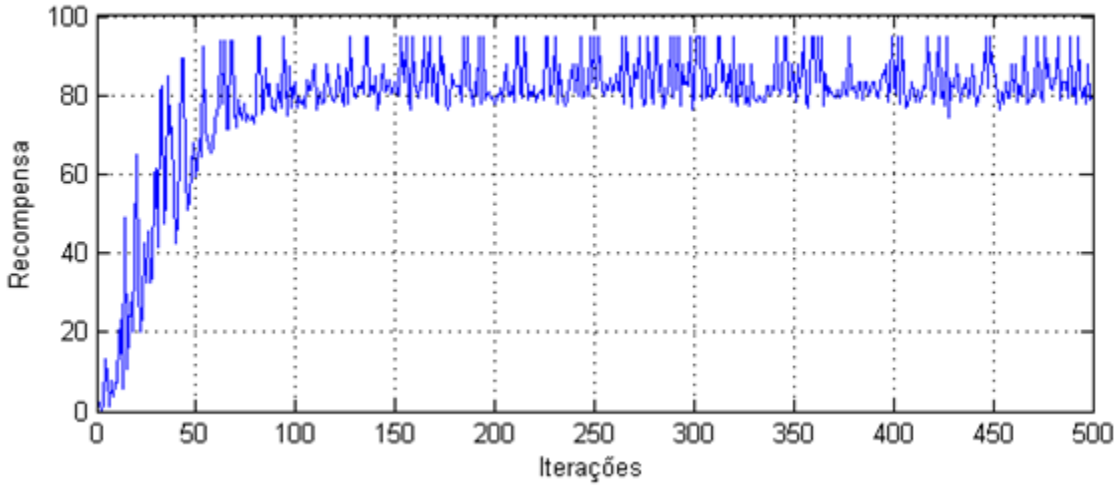
Apenas serão dados valores de recompensa altos quando a próxima ação for ir para C. Isto significa que o agente escolherá caminhos que forneçam maiores recompensas, e estas informações são guardadas em uma matriz que irá sendo “treinada” conforme o número de iterações aumenta.

O coeficiente de aprendizado define a velocidade de aprendizado e convergência de um valor alto recompensa e o fator de desconto define o quanto o agente de aprendizado irá explorar o ambiente.

A FIGURA 9 mostra o resultado de uma simulação com o treino de 500 iterações, onde o agente deveria sair do ponto A e ir ao ponto C em cada iteração, e a cada vez que fizer o processo atualizar a tabela de recompensas (Tabela Q). É possível ver o ganho de recompensa do agente ao longo das iterações, e é interessante observar que mesmo quando o sistema identifica um “melhor caminho”, ou seja, quando o gráfico tende a estabilizar com um valor máximo de recompensa, ele decai devido ao fator de desconto, ou fator de exploração aplicado. Por isto o que se tem após aproximadamente 100 iterações é uma faixa de melhores

recompensas para aquele sistema de conjunto de estados e ações. Assim fica fácil notar porque este método é eficiente com dados que são atualizados em tempo real, visto que ele sempre explora novos caminhos buscando uma nova melhor solução.

FIGURA 9 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA O MELHOR CAMINHO COM PARÂMETROS $\alpha = 0{,}1\ e\ \gamma = 0{,}9$ DURANTE 500 ITERAÇÕES



Fonte: dos autores

O gráfico de aprendizado, na FIGURA 9, está com a seguinte configuração:

- $\alpha = 0{,}1 - \ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}9 - \ fator\ de\ desconto$;
- $iterações = 500$.

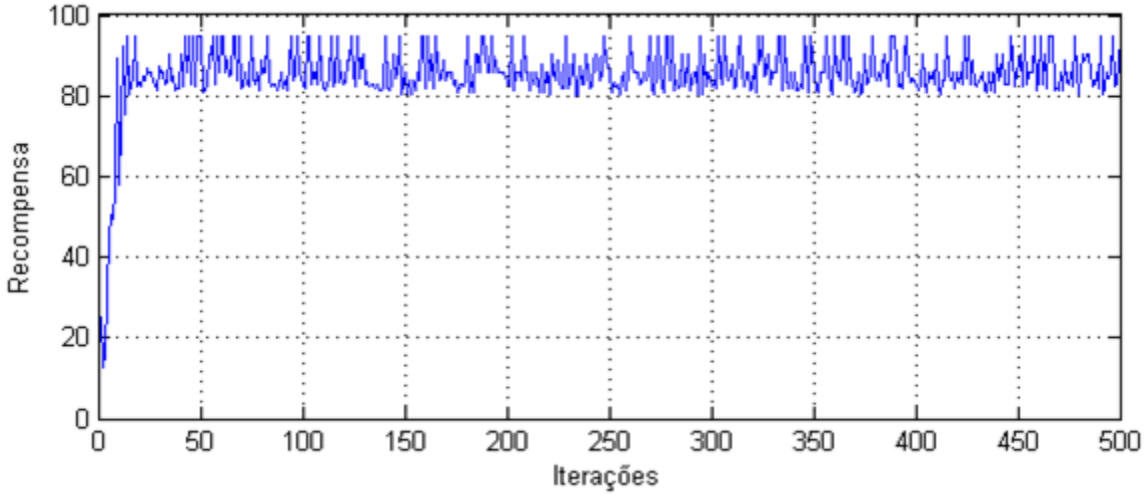
Nesta configuração, o gráfico de aprendizado se comportou adequadamente, convergindo o aprendizado em torno de 100 iterações.

Alterar o valor do coeficiente de aprendizado faz com que o sistema estabilize mais rapidamente, o que nesta aplicação onde o ambiente não muda com frequência, tende a ser mais perto do valor unitário para melhores resultados. Já o fator de desconto estimula a exploração e conforme ele diminuiu mais instável fica o valor de recompensa do agente de aprendizado do sistema.

O gráfico de aprendizado, na FIGURA 10, está com a seguinte configuração:

- $\alpha = 0{,}5 - \ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}9 - \ fator\ de\ desconto$;
- $iterações = 500$.

FIGURA 10 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA O MELHOR CAMINHO COM PARÂMETROS $\alpha = 0{,}5\ e\ \gamma = 0{,}9$ DURANTE 500 ITERAÇÕES

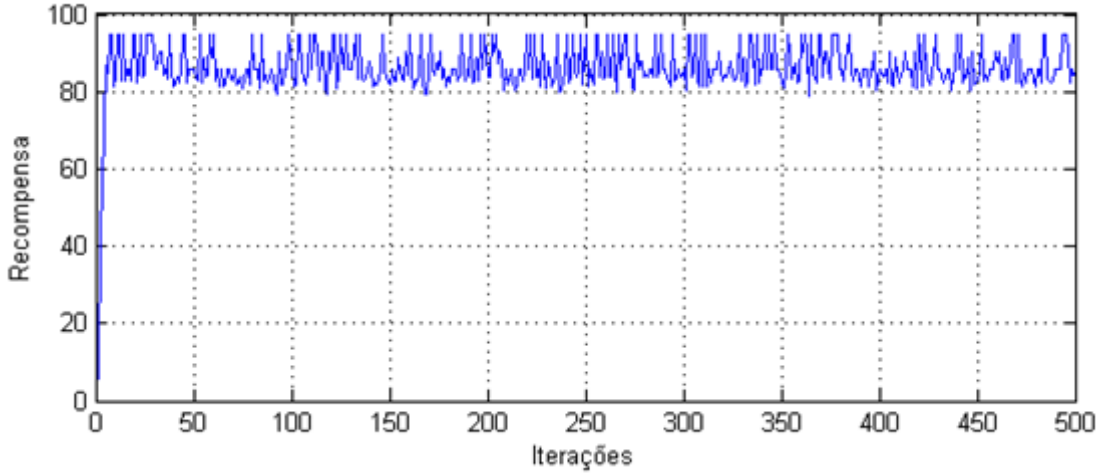


FONTE: dos autores

O gráfico de aprendizado, na FIGURA 11 está com a seguinte configuração:

- $\alpha = 0{,}9 -\ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}9 -\ fator\ de\ desconto$;
- $iterações = 500$.

FIGURA 11 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA O MELHOR CAMINHO COM PARÂMETROS $\alpha = 0{,}9\ e\ \gamma = 0{,}9$ DURANTE 500 ITERAÇÕES



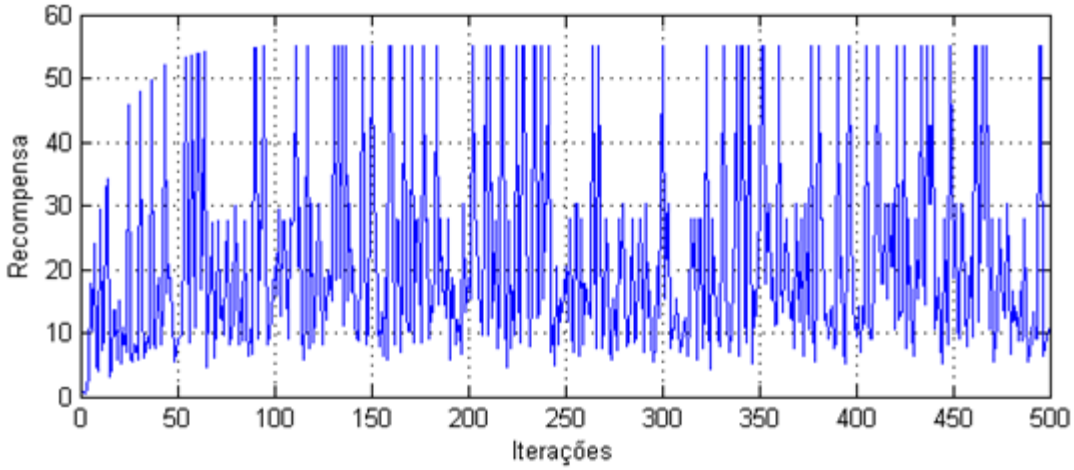
Fonte: dos autores

O gráfico de aprendizado, na FIGURA 12, está com a seguinte configuração:

- $\alpha = 0{,}1 -\ coeficiente\ de\ aprendizado$;

- $\gamma = 0{,}1 - \ fator\ de\ desconto$;
- $itera\tilde{c}\tilde{o}es = 500$.

FIGURA 12 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA O MELHOR CAMINHO COM PARÂMETROS $\alpha = 0{,}1\ e\ \gamma = 0{,}1$ DURANTE 500 ITERAÇÕES

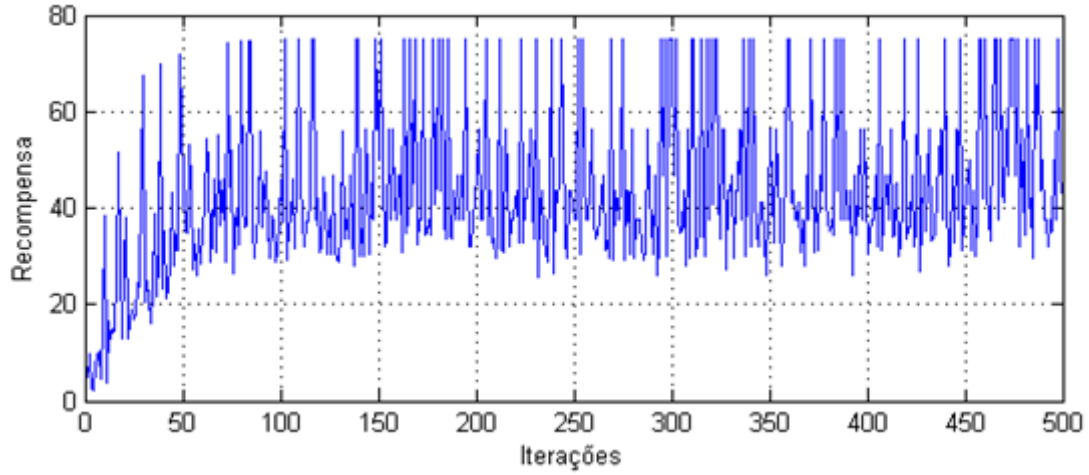


Fonte: dos autores

O gráfico de aprendizado, na FIGURA 13, está com a seguinte configuração:

- $\alpha = 0{,}1 - \ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}5 - \ fator\ de\ desconto$.

FIGURA 13 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA O MELHOR CAMINHO COM PARÂMETROS $\alpha = 0{,}1\ e\ \gamma = 0{,}5$ DURANTE 500 ITERAÇÕES

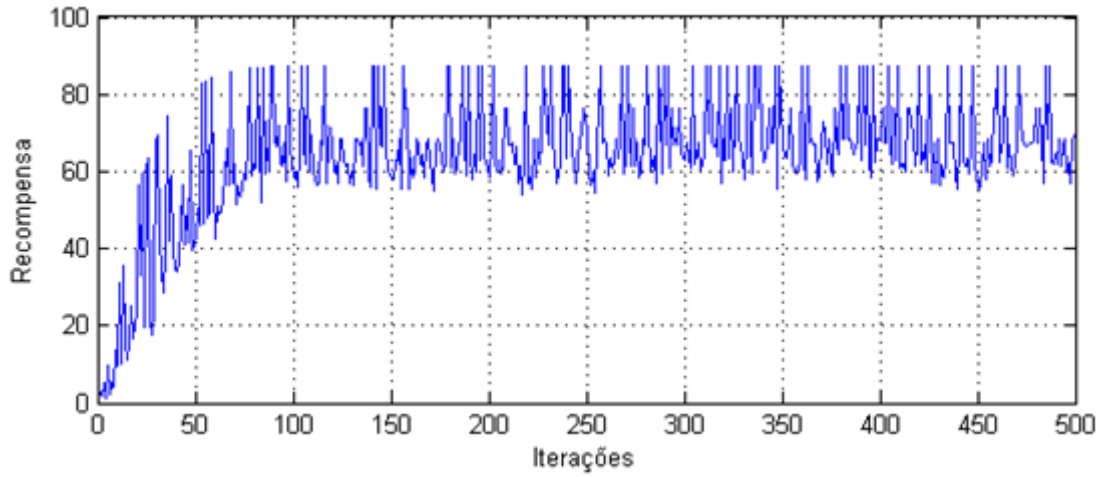


Fonte: dos autores

O gráfico de aprendizado, na FIGURA 14, está com a seguinte configuração:

- $\alpha = 0{,}1 - \ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}75 - \ fator\ de\ desconto$;
- $iterações = 500$.

FIGURA 14 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA O MELHOR CAMINHO COM PARÂMETROS $\alpha = 0{,}1\ e\ \gamma = 0{,}75$ DURANTE 500 ITERAÇÕES



FONTE: dos autores

3.2 DESENVOLVIMENTO DO MÉTODO *Q-LEARNING* PARA CONTROLE DE TRÂNSITO

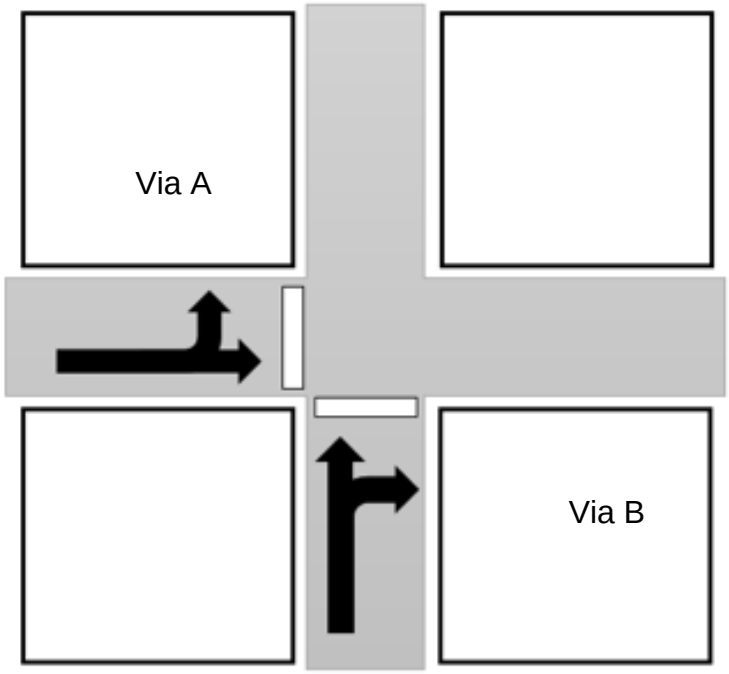
O primeiro passo para implementação do *Q-Learning* em um controle de semáforos é a definição dos parâmetros básicos do algoritmo, ou seja, qual será o valor de recompensa, os tipos de ação que poderão ser tomadas e o ambiente de operação.

A função de recompensa foi definida dependendo do tamanho da fila de veículos a cada ciclo do semáforo. Cada ciclo representa a soma do tempo que um semáforo permanece verde com o tempo que ele permanece vermelho. Os estados são o número total de veículos em cada via da esquina e o ciclo do semáforo. As ações foram definidas como a mudança de tempo em que um semáforo permanece em verde e vermelho, dentro de um ciclo.

3.2.1 Definição do Ambiente de simulação

Foi necessário a criação de um ambiente de simulação para testes, criado no *software* MATLAB®, sendo necessário definir todos os parâmetros do ambiente onde o *Q-Learning* será implementado, como número de semáforos, tamanho da quadra, direção onde os veículos podem seguir, o tamanho médio de um veículo, a distância média entre dois veículos e a aceleração média do veículo, quantidade de veículos que entram em cada via em um ciclo do semáforo.

FIGURA 15 - AMBIENTE DE SIMULAÇÃO



FONTE: dos autores

A simulação foi feita com apenas uma esquina, para simplificar a implementação, e apenas dois semáforos, a FIGURA 15 mostra uma representação desta esquina, indicando os possíveis sentidos que os carros em cada via podem seguir. Cada quadra tem um tamanho de 100 metros por 100 metros. O modelo contém duas vias, na via A os veículos podem apenas continuar em frente ou virar para esquerda, e na via B os veículos podem seguir em frente ou virar para esquerda. As duas vias contêm apenas uma mão, ou seja, um único sentido de tráfego.

3.2.2 A física do ambiente de simulação

Embora simples este ambiente deva obedecer algumas regras físicas, para que os dados adquiridos na simulação tenham certa validade quando comparados a um modelo real.

A TABELA 2 mostra as informações necessárias para a simulação baseado na posição em que o veículo está na fila de espera em uma das vias. O tamanho médio dos veículos foi definido através de uma média do comprimento de uma lista de veículos mais populares que circulam no Brasil, que pode ser visto na tabela 3. A aceleração foi definida pela média de aceleração de 0 a 27,78 m/s (100 quilômetros por hora) destes veículos consultando as informações técnicas fornecidas pelos fabricantes.

TABELA 2 – DADOS NECESSÁRIOS PARA A SIMULAÇÃO

POSIÇÃO	DISTANCIA DO SEMÁFORO (m)	TEMPO ATÉ ATINGIR 40km/h (s)	TEMPO ATÉ O SEMÁFORO (s)	DISTÂNCIA PERCORRIDA ACELERANDO	ATRASO (s)	DISTANCIA RESTANTE APÓS 40km/h	TEMPO ATÉ O SEMÁFORO a 40km/h(s)	TEMPO ACUMULADO (s)
1	6,34	5,6	2,53	6,34	1,5	0	0	4,03
2	12,68		3,37	12,68	3	0	0	6,37
3	19,02		4,21	19,02	4,5	0	0	8,71
4	25,36		4,91	25,36	6	0	0	10,91
5	31,7		5,52	31,7	7,5	0	0	13,02
6	38,04		6,07	31,11	9	6,933	6,224	21,3
7	44,38		6,58	31,11	10,5	13,273	6,794	23,87
8	50,72		7,05	31,11	12	19,613	7,365	26,41
9	57,06		7,49	31,11	13,5	25,953	7,936	28,92
10	63,4		7,9	31,11	15	32,293	8,506	31,41
11	69,74		8,3	31,11	16,5	38,633	9,077	33,87
12	76,08		8,67	31,11	18	44,973	9,648	36,32
13	82,42		9,04	31,11	19,5	51,313	10,218	38,75
14	88,76		9,38	31,11	21	57,653	10,789	41,17
15	95,1		9,72	31,11	22,5	63,993	11,36	43,58

FONTE: dos autores.

A construção dos dados da tabela 2 foi feita a partir das equações básicas de mecânica clássica newtoniana. Desta forma foi possível estimar uma velocidade e aceleração dos veículos de forma ideal, o que para esta etapa do desenvolvimento será suficiente para uma análise básica do sistema. Nesta simulação a velocidade máxima permitida será de 11,11 m/s (40 quilômetros por hora), velocidade comum em vias urbanas no Brasil. E o tempo máximo de um ciclo do semáforo (tempo de sinal verde mais o tempo de sinal vermelho será de 60 segundos), o que significa que quando um dos semáforos estiver com 35 segundos verde o outro estará com 25 segundos, e assim por diante.

TABELA 3 – DADOS DE TAMANHO E ACELERAÇÃO DE ALGUNS CARROS POPULARES DO BRASIL

Marca	Modelo	Comprimento (m)	Largura (m)	Aceleração 0-100 km/h (s)
Chevrolet	Celta Life 1.0	3,788	1,626	14,3
	Astra Adv 2.0 8V	4,199	1,709	9,8
	Classic LS 1.0	4,152	1,608	13,6
	Onix LTZ 1.4	3,93	1,608	13,6
	Cobalt LTZ 1.4	4,481	1,735	11,5
	S10 LTZ 2.4	5,347	1,882	11,9
Fiat	Novo Uno Way 1.0	3,77	1,656	15,8
	Palio Att 1.0	3,875	1,67	15
	Nova Strada Adv 1.8	4,471	1,74	10,3
	Siena EL 1.0	4,155	1,639	16,1
	Stilo Att 1.8	4,253	1,756	10,7
	Fiorino Furgão 1.3	4,183	1,622	14
Volkswagen	Gol Seleção 1.0	3,899	1,656	13
	Kombi Std 1.4	4,505	1,72	16,1
	Voyage 1.0	4,23	1,656	13,3
	Amarok 4x4 CD 2.0	5,254	1,944	13,7
	Fox 1.0	3,823	1,64	14,2
	Saveiro 1.6	4,493	1,708	10,5
	Médias	4,22	1,69	13,19

FONTE: CARROS NA WEB (2016)

Com estes dados é possível definir a quantidade máxima de carros passam em um determinado tempo em que o semáforo permaneça no sinal verde, levando em consideração: a posição do veículo na fila, a sua aceleração até atingir a velocidade máxima de 11,11 m/s, e o tempo de atraso para o início da aceleração do veículo, dado a posição na fila. A última coluna da TABELA 3 mostra os tempos finais necessários para um carro em determinada posição ultrapassar o semáforo, por exemplo, com 40 segundos de tempo no sinal verde é possível atravessar até 13 carros.

Uma simulação mais precisa teria que levar em consideração a aceleração variada de cada veículo, que alteraria o fluxo de veículos em cada ciclo. Poderia considerar também possíveis acidentes e falhas nos veículos, pedestres atravessando a rua em locais não adequados, veículos não autorizados para a via e a possibilidade de trocas de faixas, caso houvesse mais faixas. E o número de veículos que entram em uma via será adquirido no futuro através de processamento de imagens, mas para esta simulação estes valores foram definidos previamente.

3.2.3 Aplicação do algoritmo

O método de *Q-Learning* deve aprender e reaprender para fornecer a melhor escolha ao sistema de controle. Deste modo foi necessário implantar uma função de ganho de recompensa que atenda a um objetivo específico, neste caso diminuir a fila de carros em cada via. O algoritmo monta uma tabela com a recompensa ganha em cada escolha que tomou, dado um certo estado e as possíveis ações deste estado. Como foi visto no exemplo para encontrar um melhor caminho, o algoritmo testa caminhos possíveis randomicamente, mas apenas quando alcança um certo objetivo ele ganha recompensa, assim o *Q-Learning* descobre um “melhor caminho” devido a recompensa ganha quando atingiu sua meta. Cabe então a política do algoritmo selecionar este caminho e fornecer ao sistema.

Para melhorar o fluxo de trânsito o algoritmo deve ser capaz de modificar o tempo dos semáforos para melhor se adaptar ao número de veículos, por exemplo, se cada uma das via tem um fluxo de saída máximo de 9 veículos, em 30 segundos de tempo verde no semáforo, quando mais veículos entraram em uma via e/ou menos em outra ele deve alterar o tempo para 15 segundos verde para a via de menor tráfego e 45 segundos para a de maior, isto deve ser feito em tempo real para

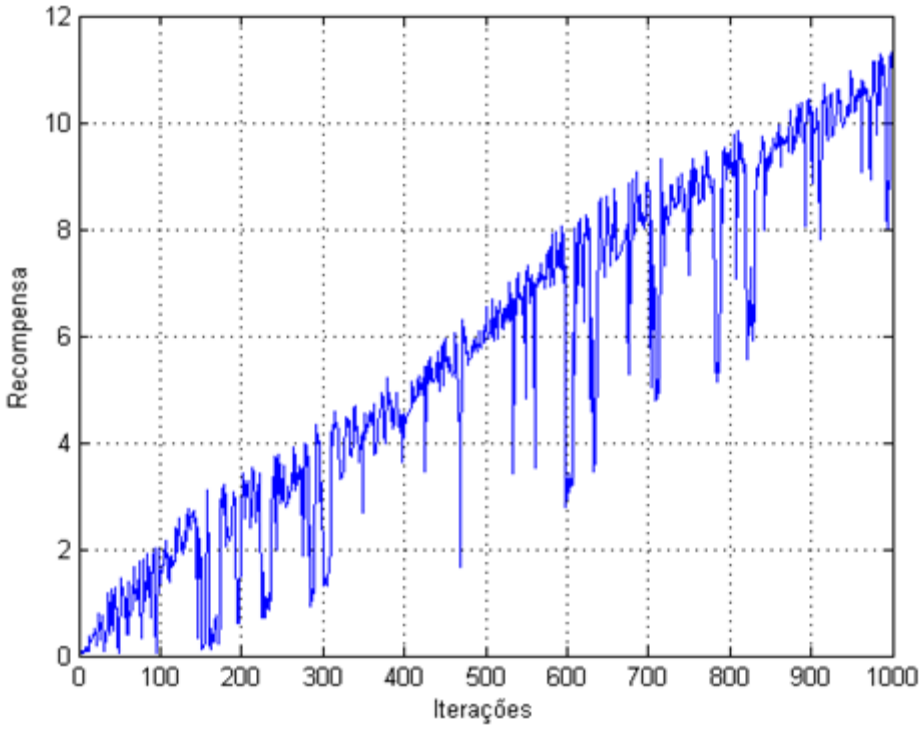
que ele aprenda qual é o melhor tempo para cada semáforo. Ou seja, um semáforo de tempo fixo 30 segundos verde e 30 segundos vermelho será eficiente quando as duas vias têm o mesmo fluxo, mas quando ambas se alternam é necessário um aprendizado em tempo real para otimizar o fluxo.

3.3 RESULTADOS PRELIMINARES DE UM CONTROLE DE TRÂNSTIO UTILIZANDO *Q-LEARNING*

Feita a implementação em MATLAB®, com a entrada de veículos randômica os parâmetros de aprendizado e fator de desconto devem ser avaliados. A FIGURA 16 mostra o gráfico de aprendizado com a seguinte configuração:

- $\alpha = 0{,}1 - \ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}98 - \ fator\ de\ desconto$;
- $iterações = 1000$.

FIGURA 16 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA TRÂNSITO COM PARÂMETROS $\alpha = 0{,}1\ e\ \gamma = 0{,}98$ DURANTE 1000 ITERAÇÕES

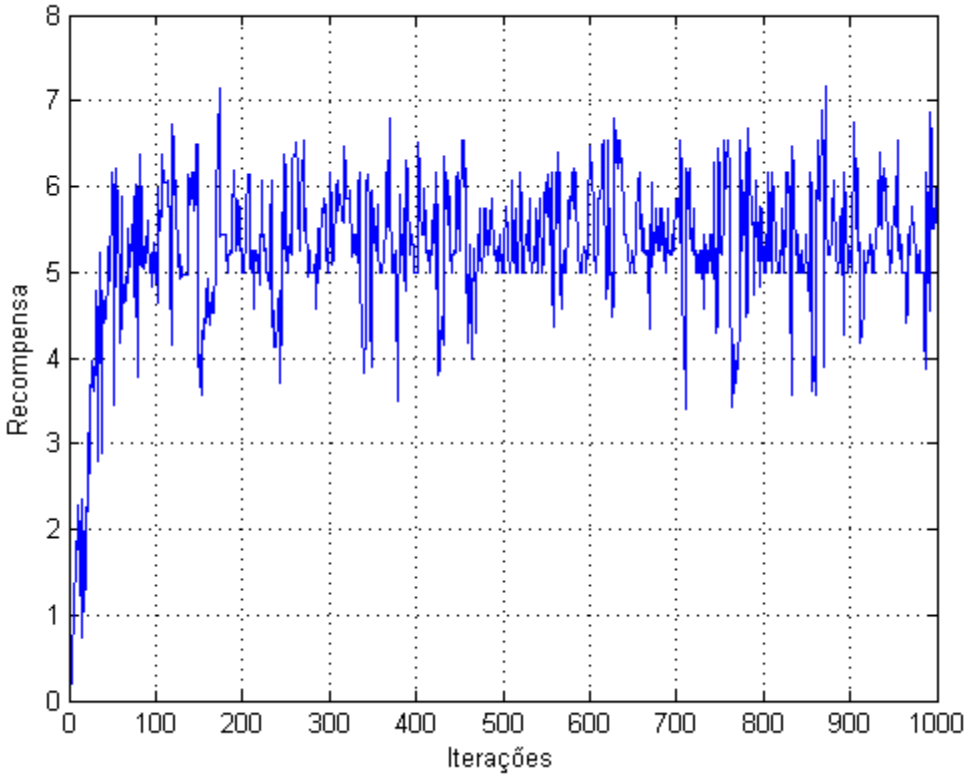


Fonte: dos autores

É possível ver que nesta configuração o sistema não converge para uma região de estabilidade de recompensa. A FIGURA 17 mostra o gráfico de aprendizado com os seguintes parâmetros:

- $\alpha = 0{,}9 - \ coeficiente\ de\ aprendizado$;
- $\gamma = 0{,}9 - \ fator\ de\ desconto$;
- $itera\tilde{c}\tilde{o}es = 1000$.

FIGURA 17 - GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA TRÂNSITO COM PARÂMETROS $\alpha = 0{,}9\ e\ \gamma = 0{,}9$ DURANTE 1000 ITERAÇÕES



FONTE: dos autores

Na FIGURA 17 constate-se que há convergência com os valores de α e γ para uma recompensa estável, mas existe uma oscilação muito alta devido ao fator de desconto.

Nesta configuração o ganho de recompensa está mais estável e pode ser usado para um controle efetivo. Em outras palavras, o sistema atingiu uma região de ganho de recompensas que não oscila muito, isto se deve ao equilíbrio entre um bom fator de desconto e de um coeficiente de aprendizado, ou seja, o sistema busca

novas combinações de tempos para os semáforos, mas nunca deixar que a recompensa abaixe demais.

Uma das configurações mais distintas que este modelo pode adquirir seria de 10 segundos verde em um semáforo e 50 segundos verde no outro, desta forma em comparação com um sistema fixo de 30 segundos, caso exista apenas um veículo em uma das vias, e na outra um fluxo mais intenso, o ganho seria de até 60% no fluxo de veículos neste ambiente de simulação.

3.4 IMPLEMENTAÇÃO EM *HARDWARE* FPGA

Para o desenvolvimento da implementação de *Q-Learning* em *hardware* FPGA (*Field Programmable Gate Array*), utilizou-se a ferramenta *HDL Coder* do MATLAB®, que possibilita geração de códigos VHDL a partir de modelos MATLAB® e Simulink®.

Como o desenvolvimento do método *Q-Learning* para controle de trânsito já foi desenvolvido para o ambiente MATLAB®, é plausível aplicar o *HDL Coder* na implementação em *hardware*. Para a elaboração do algoritmo no ambiente Simulink®, é necessário criar uma nova topologia e consequentemente, um novo algoritmo, no entanto, é similar ao desenvolvimento anterior em ambiente MATLAB®.

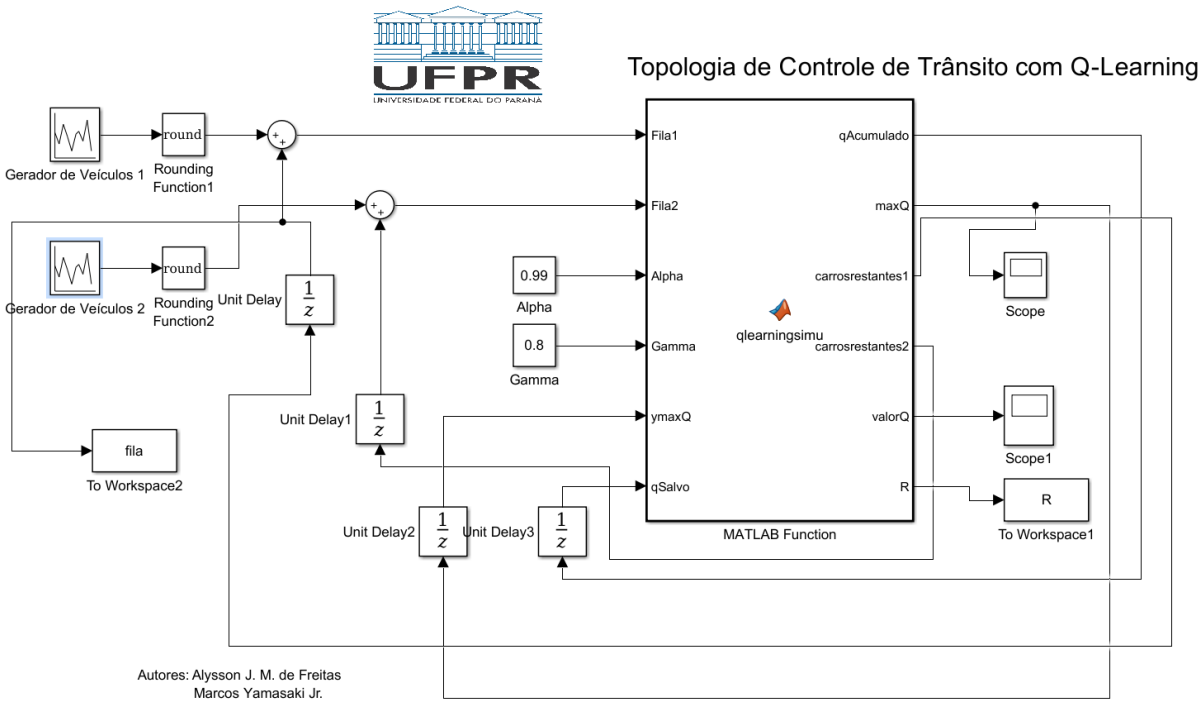
Foi considerado o uso do *System Generator*, ferramenta do Xilinx ISE Design Suite (*System Edition*) e do Xilinx Vivado HL (*System Edition*), que também facilita a geração do código VHDL para FPGA a partir de modelos MATLAB® e Simulink®, limitado a fabricante Xilinx. Entretanto, nos testes de implementação, foram detectadas diversas falhas de integração e comunicação com a placa de desenvolvimento FPGA da Xilinx.

Adicionalmente, é necessário utilizar a ferramenta *HDL Verifier*, também pertencente ao MATLAB®, para realizar a cossimulação com a placa FPGA e a geração do *bitstream* sintetizado para ser implementado diretamente sem a necessidade do uso de ferramentas terceiras que, portanto, seria o *software* de *design* do fabricante da placa.

3.5 MODELAGEM DO SISTEMA DE CONTROLE

Uma das etapas necessárias para implementação em *hardware* é o desenvolvimento de uma topologia em forma de diagrama de blocos do algoritmo implementado, a FIGURA 18 mostra a topologia desenvolvida dentro da ferramenta Simulink® do MATLAB®. A topologia proposta representa uma iteração do algoritmo e é formada pelos seguintes blocos: Gerador de Veículos, Constantes, Bloco principal e Unidades de Atraso.

FIGURA 18 – TOPOLOGIA DE CONTROLE DE TRÂNSITO COM *Q-LEARNING*



Fonte: dos autores.

O bloco gerador de veículos consiste em um gerador pseudoaleatório de valores que variam entre 1 e 15 (número de carros que podem entrar na via). Será necessário modelar este bloco com base em dados estatísticos reais de trânsito para um resultado mais coeso.

As constantes Alpha (constante de aprendizado), Gamma (fator de desconto) são necessárias para o funcionamento do método como explanado anteriormente, e a constante **I** é necessária apenas para a primeira iteração do sistema onde os valores de realimentação do sistema ainda são nulos.

Os blocos de atraso são necessários para qualquer aplicação em tempo real, visto que o sistema não pode ser realimentado sem ter processado a sua saída primeiramente.

O bloco principal é composto pelas seguintes entradas: Fila1, que representa a fila de veículos na primeira via; Fila2, que representa a fila de veículos na segunda via; Alpha, o coeficiente de aprendizado; *Gamma*, o fator de desconto; *ymaxQ*, que representa o valor máximo obtido pelo sistema e deve ser realimentado e atualizado a cada iteração do sistema; e *qSalvo*, que é necessário para acumular os valores de recompensa obtidos ao longo do funcionamento do sistema. Nas saídas do bloco principal tem-se: *qAcumulado* que representa os valores de recompensa acumulados; *maxQ*, o valor máximo de recompensa obtidos no sistema; e os carros restantes na via 1 e via 2 que são somados aos carros que chegam a fila.

TABELA 4 – AÇÕES DO SISTEMA

Ciclo de 60 segundos	Tempo Verde Semáforo da primeira via	Tempo Verde Semáforo da segunda via
Ação 1	0	60
Ação 2	5	55
Ação 3	10	50
Ação 4	15	45
Ação 5	20	40
Ação 6	25	35
Ação 7	30	30
Ação 8	35	25
Ação 9	40	20
Ação 10	45	15
Ação 11	50	10
Ação 12	55	5
Ação 13	60	0

Fonte: dos autores.

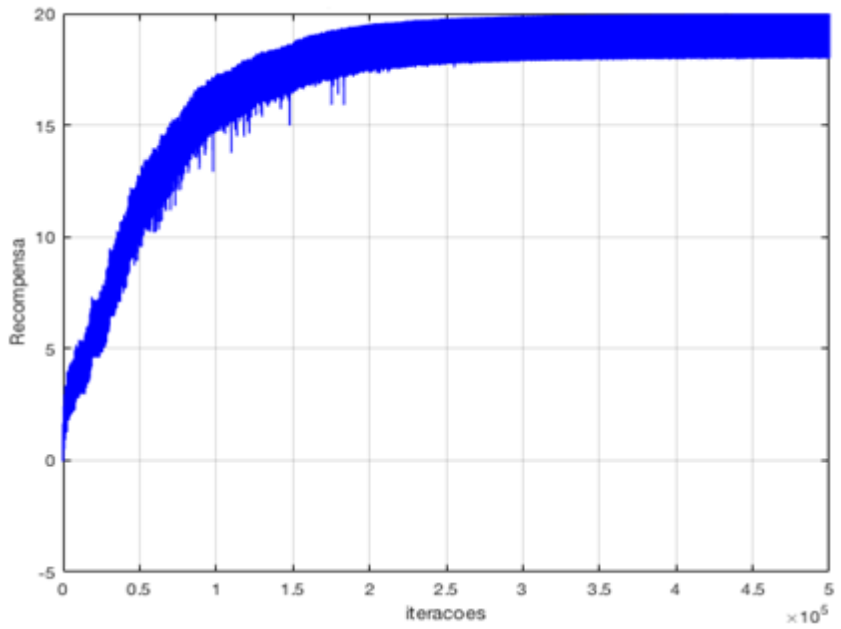
O algoritmo foi refinado e tem como estados o tamanho das filas em cada via, e como ações os tempos dos semáforos, no entanto foi necessário limitar o escopo nesta etapa permitindo um máximo de 15 carros em cada via e uma mudança de 5 segundos no tempo do semáforo a cada iteração, ou seja, se o ciclo sinal verde-sinal vermelho for definido em 60 segundos, os tempos dos semáforos apenas poderão ser alterados para múltiplos de 5, e o par de tempos em verde dos dois semáforos

deverá sempre fechar o ciclo de 60 segundos, por exemplo [5 55], [10 50], [15 45], etc. Sendo assim definido um número máximo de 13 ações, conforme a TABELA 4.

A função de recompensa também foi modificada fornecendo valores ponderados dependendo do tamanho da fila de cada via.

Desta forma a tabela Q deste sistema que é formada pelos estados e ações possíveis do ambiente será uma matriz (15x15x13). Mesmo limitando estes parâmetros um número muito grande de iterações é necessário para que o sistema aprenda e preencha todos os valores da tabela Q. E para que houvesse uma convergência em valores ótimos de recompensa para tomada de decisão foram necessárias 200.000 iterações, como é possível ver na FIGURA 19. Portanto, será necessário definir um critério de parada para o sistema assim que um valor satisfatório seja atingido para tomar a decisão de troca de tempo dos semáforos.

FIGURA 19 – GRÁFICO DE APRENDIZADO *Q-LEARNING* PARA SEMÁFORO



FONTE: dos autores.

Com este passo desenvolvido será possível realizar alguns testes de implementação em *hardware* FPGA, levando em consideração as restrições do *hardware*, como capacidade de memória, conversão de ponto flutuante para ponto fixo, tempo de processamento, etc.

3.6 MODELAGEM DE UM ALGORTIMO EXEMPLO DE *Q-LEARNING*

Para implementação em *hardware* FPGA utilizando o *HDL Coder* no Simulink®, foi necessário desenvolver um algoritmo exemplo com complexidade baixa para validar a metodologia de implementação. O modelo mostrado de Controle de Trânsito com *Q-Learning* ainda não está adequado e apropriadamente configurado para ser implementado em *hardware*, pois se deve atender a arquitetura de um *hardware* e seu método de funcionamento, como por exemplo, definir os pontos-fixos (*fixed-points*) de cada dado, seja de entrada ou de saída, para a FPGA reservar adequadamente espaço na memória para estes dados.

Para realizar esta adequação antes de tudo, deve-se realizar a simulação do modelo no Simulink® e a simulação do modelo em código VHDL em um ambiente externo, como o ModelSim da Mentor. Essas simulações podem ser chamadas de cossimulação, onde é realizado simulações em ambientes diferentes, com códigos em linguagens diferentes com o mesmo modelo do projeto. Portanto, foi realizada a cossimulação do algoritmo exemplo *Q-Learning* no ambiente Simulink® utilizando o *add-on HDL Coder* do próprio MATLAB®, para implementação em *hardware* FPGA. A cossimulação então significa simular o projeto em duas etapas de maneira simultânea, ou seja, uma simulação utilizando o Simulink® e outra simulação utilizando um simulador externo, como o ModelSim HDL Simulator da Mentor. Desta maneira, é possível validar o código VHDL gerado pelo *HDL Coder* no projeto produzido no Simulink®.

3.6.1 *Softwares* necessários

Os *softwares* utilizados para gerar o código VHDL e para realizar a cossimulação com as respectivas versões utilizadas são os seguintes:

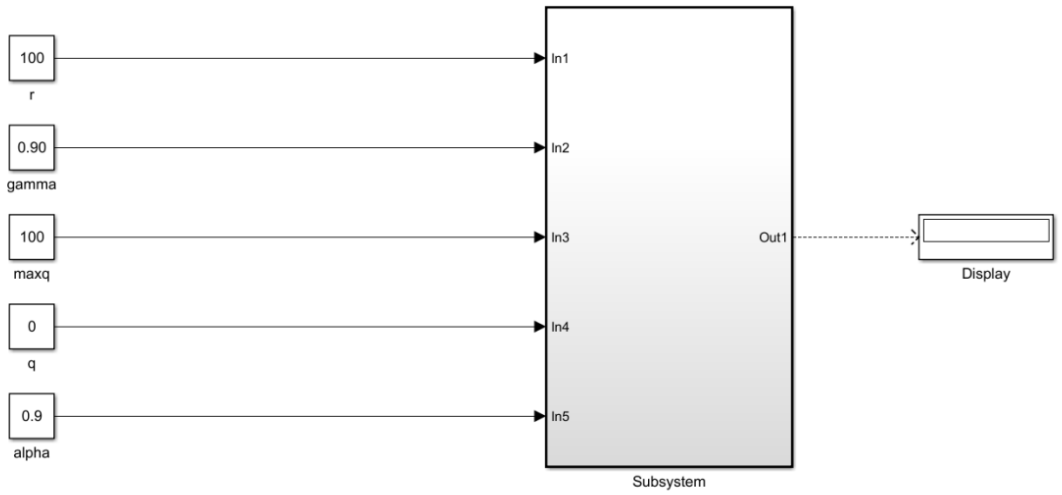
- MATLAB® R2016b;
- Simulink® Version 8.8 (R2016b);
- HDL Coder™ (R2016b);
- ModelSim® PE Student Edition HDL Simulator 10.4a (rev 2015.03) – Mentor®;
- ISE Design Suite v14.7 (Full license version) – Xilinx, Inc.;
- Microsoft Windows 10 Pro (10.0.15064 Build 15063).

Para o funcionamento correto do ISE Design Suite v14.7 com o Windows 10, foi necessário realizar alguns procedimentos extras como substituir alguns arquivos *.dll* nos diretórios do ISE.

3.6.2 Gerando Código VHDL

Para gerar o código VHDL pelo Simulink®, é necessário ter um projeto já preparado para um *hardware* FPGA. Para ser possível compilar esse projeto e passar pelo *HDL Workflow Advisor* (assistente do HDL Coder® para geração do projeto no formato VHDL e entre outros), deve-se primeiro atentar-se aos tipos de dados que serão utilizados no bloco (ou função) que será gerado em VHDL. Portanto é necessário tomar alguns cuidados na hora de implementar o projeto, como definir os dados de entrada e saída como ponto-fixo (*fixed-point*), utilizar atrasos (*delays ou registradores*) e não utilizar funções e *toolbox* do MATLAB® que não tenham suporte com o HDL Coder®. A FIGURA 20 é um exemplo de implementação que será utilizada, e os dados de entrada e de saída estão configurados para *fixed-point* em diferentes configurações de valores máximo, mínimo e quantidade de casas decimais. Para mais detalhes deste processo, é possível solicitar aos autores um tutorial detalhado sobre como realizar o processo de cossimulação.

FIGURA 20 – EXEMPLO DE IMPLEMENTAÇÃO NO SIMULINK PARA GERAÇÃO DO PROJETO EM VHDL UTILIZANDO A TOOLBOX HDL CODER



Fonte: dos autores.

3.7 SIMULADOR DE TRÂNSITO

Para evoluir no projeto se fez necessária a inclusão de um simulador de veículos para validar o controlador proposto. No entanto qualquer simulador completo exige uma demanda de tempo alta de estudo devido a complexidade dos componentes envolvidos. Embora fossem necessários resultados nesta etapa do projeto, isto só seria possível com a inserção de um simulador ao mesmo. Este simulador deveria ser compatível direta ou indiretamente com o ambiente de programação utilizado, ser preferencialmente gratuito e de fácil aprendizado. Os seguintes simuladores foram encontrados:

3.7.1 PTV VISSIM

O *software* VISSIM é produzido pela empresa PTV é um dos melhores aplicativos disponíveis no mercado para projetos envolvendo veículos. Ele oferece ambiente de simulação 3D e 2D, várias extensões com *softwares* de outros fabricantes, contudo sua licença é paga e não oferece nenhuma opção para avaliação. Desta forma, não foi possível utilizar este programa em nenhum dos testes de desenvolvimento.

i

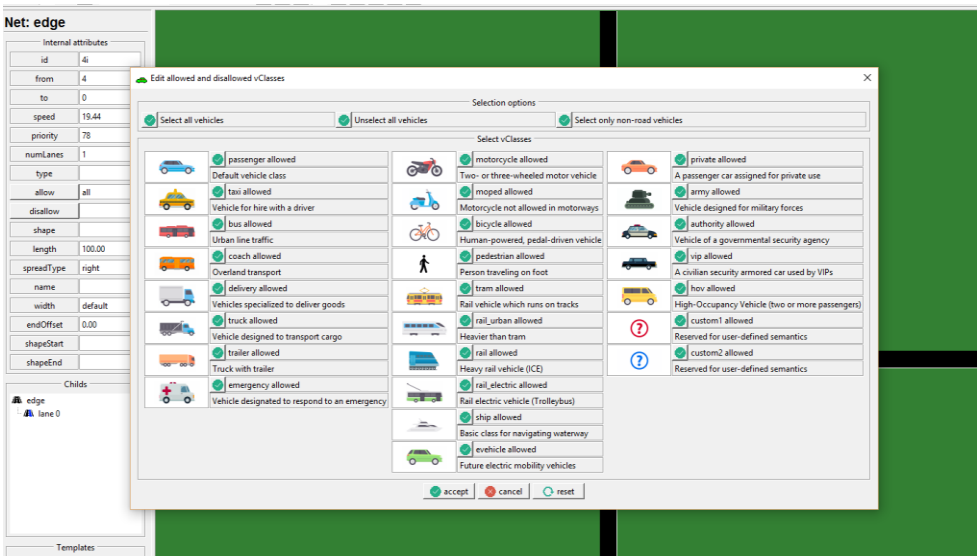
3.7.2 ARENA

O ARENA é um *software* fornecido pela empresa PARAGON. Ele se destaca por conseguir modelar quase qualquer tipo de sistema de maneira, mas simplificada, com a utilização de fluxogramas por exemplo. No entanto ele não possui nenhuma extensão para o MATLAB® de maneira direta e montar um sistema de tráfego não era um dos focos principais do aplicativo o que também cria uma barreira no desenvolvimento, visto que além do tempo de aprendizagem gasto no *software*, também seria necessário um tempo para desenvolver um ambiente de simulação. Sua licença é paga, mas oferece versões para estudantes e de avalição por tempo limitado.

3.7.3 SUMO

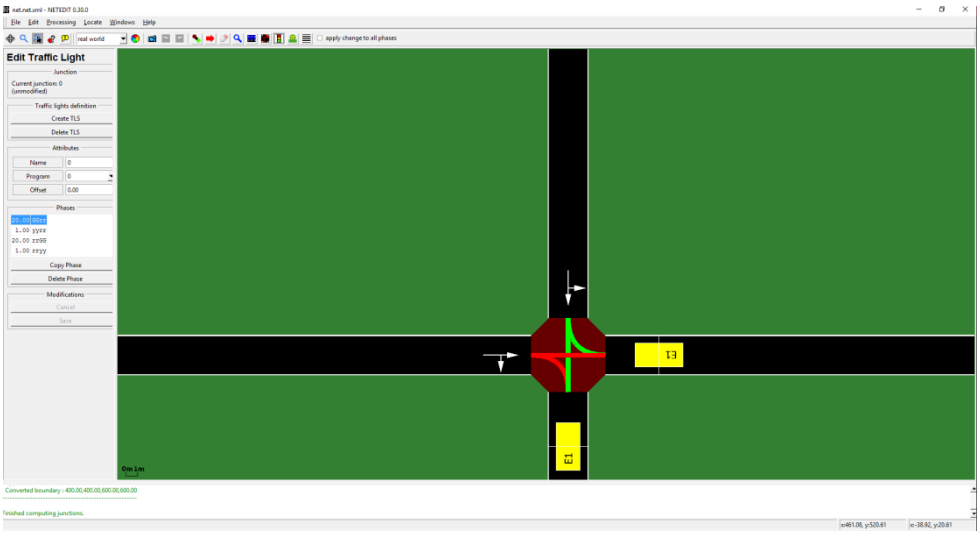
O SUMO (*Simulation of Urban Mobility*) é um *software* livre, altamente portável, e com simulação em nível microscópico. O SUMO oferece várias ferramentas avançadas de simulação com diversas possibilidades de customização. Algumas podem ser vistas na FIGURA 21, como adição de carros, motos, ônibus limitação de velocidade da via.

FIGURA 21– TELA DE CONFIGURAÇÃO DO SUMO



Fonte: dos autores.

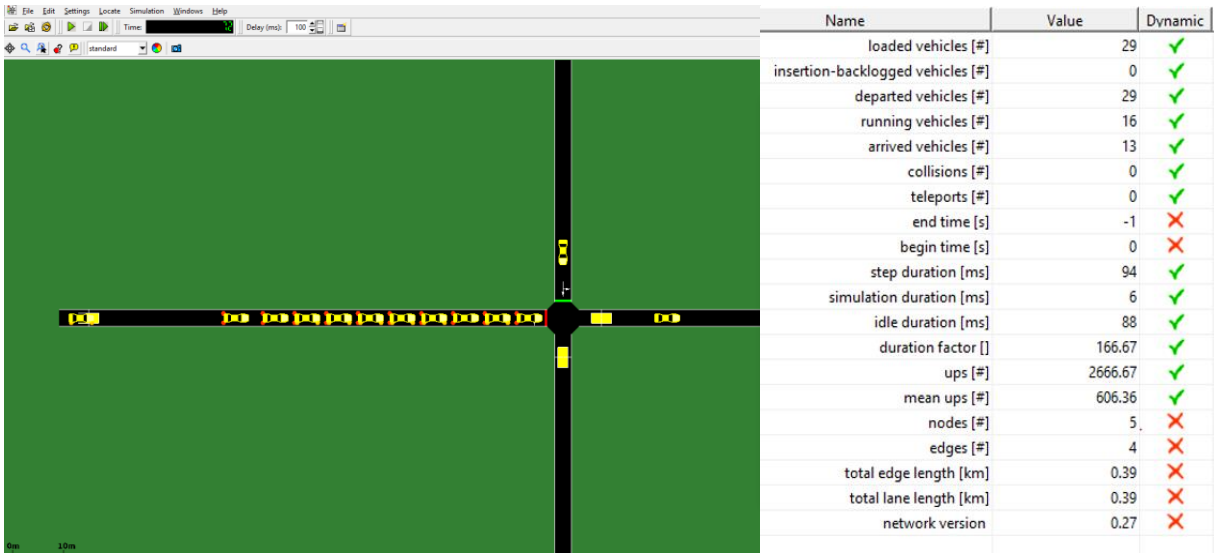
FIGURA 22 - TELA DE EDIÇÃO DAS VIA DO SUMO



Fonte: dos autores.

O SUMO oferece uma representação em 2D, onde é possível montar qualquer modelo de vias e cruzamentos como mostra a FIGURA 22. Também é possível programar os tempos dos semáforos presentes na simulação de maneira individual. É possível ainda verificar informações como o número de veículos na via e veículos que deixaram a via, e simular até mesmo colisões caso os tempos dos semáforos esteja muito pequeno e a velocidade máxima da via muito alta como pode ser visto na FIGURA 23.

FIGURA 23 - TELA DE DADOS DURANTE A SIMULAÇÃO DO SUMO



Fonte: dos autores.

Quanto a interação com o MATLAB®, o SUMO por ser uma ferramenta aberta não oferece nenhum tipo de suporte oficial, no entanto, é disponibilizado uma ferramenta chamada TraCI (Traffic Control Interface) que oferece uma interface de controle via portas remotas, com seu protocolo de comunicação específico.

TraCI4Matlab é uma API (*Application Programming Interface*) desenvolvido no MATLAB® por uma comunidade de desenvolvedores independentes que permite a comunicação entre qualquer aplicação feita nessa linguagem e o simulador de tráfego SUMO. As funções que compreendem o TraCI4Matlab implementam a aplicação de nível de protocolo TraCI (Traffic Control Interface), no qual é desenvolvido sobre o protocolo TCP/IP, e a aplicação é desenvolvida no MATLAB®, que é o cliente, pode acessar e modificar o ambiente de simulação provida pelo servidor (SUMO). O TraCI4Matlab permite o controlar os objetos do simulador, como os seus veículos, semáforos, junções ou esquinas, customização das vias, e entre

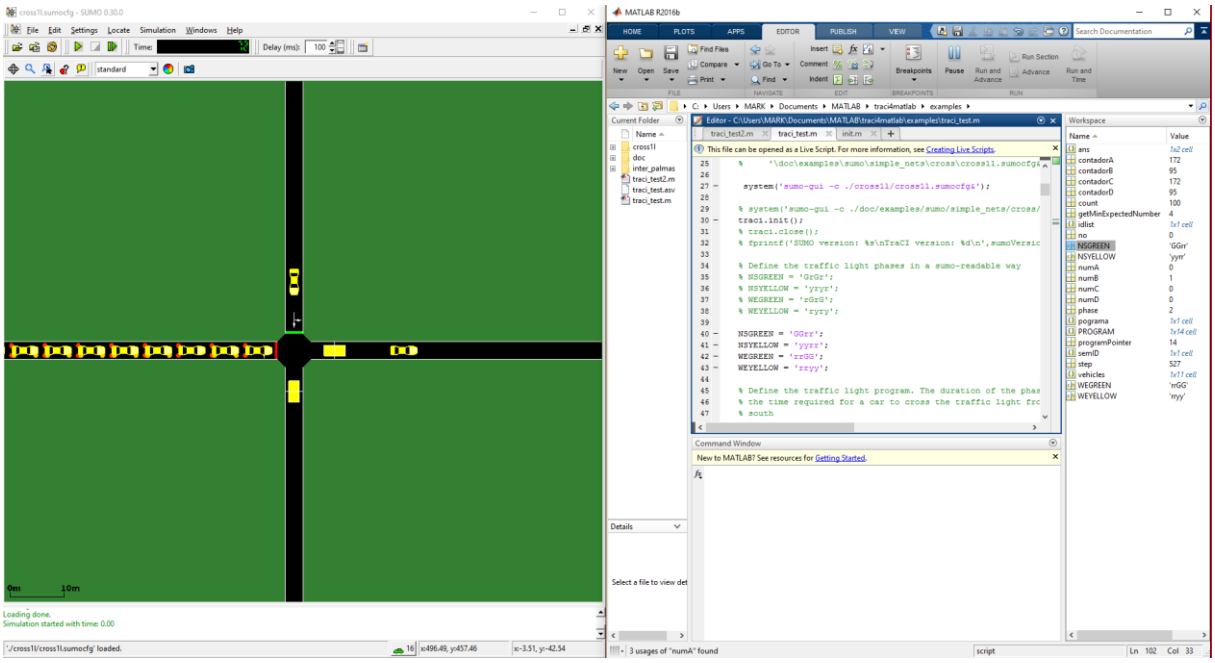
outras funcionalidades, permitindo soluções como semáforos com controles adaptativos, dinamização das rotas definidas e entre outros.

Para a utilização do TraCI4Matlab, é necessário utilizar uma versão do MATLAB 2011b ou superior, realizar a instalação do *software* de simulação SUMO e configurar de acordo com o que o *software* sempre reconheça a sua pasta raiz e por fim instalar e codificar o TraCI4Matlab. O prévio conhecimento no simulador SUMO e no ambiente MATLAB® auxilia todo o desenvolvimento do sistema, pois o TraCI4Matlab é apenas uma API que realiza a comunicação entre as duas plataformas. É necessário realizar a configuração para que o SUMO execute como um servidor e o MATLAB® comunique-se com ele como cliente, podendo ser o servidor em rede local, com número de porta específica. Os detalhes de configuração são muito importantes para o funcionamento correto da aplicação. Após a criação do servidor é necessário criar uma aplicação no MATLAB® para inicializar a comunicação com o servidor, onde nesta aplicação deve conter os comandos necessários para a inicialização do servidor, definição de parâmetros e um laço assegurar os passos da simulação do SUMO. Além disso, é necessário criar previamente um ambiente de simulação no *software* SUMO, contendo vias (*edges, lanes*), junções, e *routes* que contém os dados dos veículos que trafegarão a via, tais como tamanho, distância mínima, quantidade, velocidade, aceleração, período, ponto de início.

A integração com o MATLAB® é feita utilizando comandos da API, onde os dados do simulador podem ser obtidos, além de ser possível enviar comandos para o simulador para alteração de tempo de semáforo, por exemplo, utilizando funções já preestabelecidas. A FIGURA 24 mostra a comunicação entre o MATLAB® e o SUMO em tempo real.

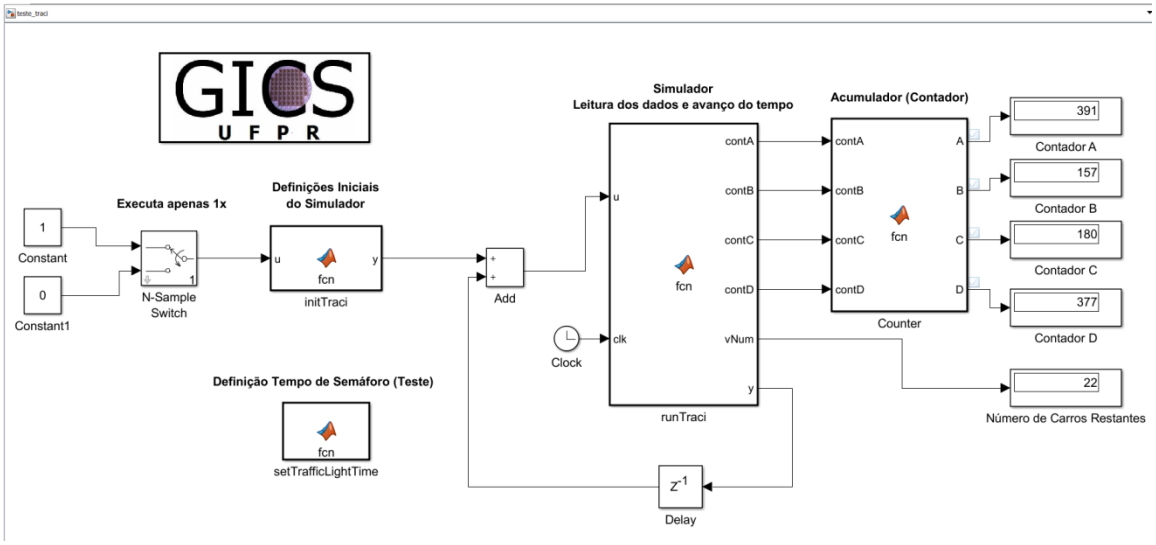
A dificuldade é a sincronização dos dados obtidos enquanto a simulação ocorre, pois é necessário conhecer a biblioteca de funções, analisar cada função para obter a informação desejada, bem como desenvolver corretamente o ambiente de simulação para não haver erros ou *bugs*. Como a simulação ocorre em passos de um segundo, é necessário analisar a cada segundo os dados recebidos e para modificar, por exemplo, o estado dos semáforos, deve-se utilizar os comandos corretos com os argumentos indicados pelo programa. Como o programa é dinâmico e está sempre em estado de mudança, é necessário sincronizar as modificações de acordo com o estado atual.

FIGURA 24 - MATLAB E SUMO SINCRONIZADOS



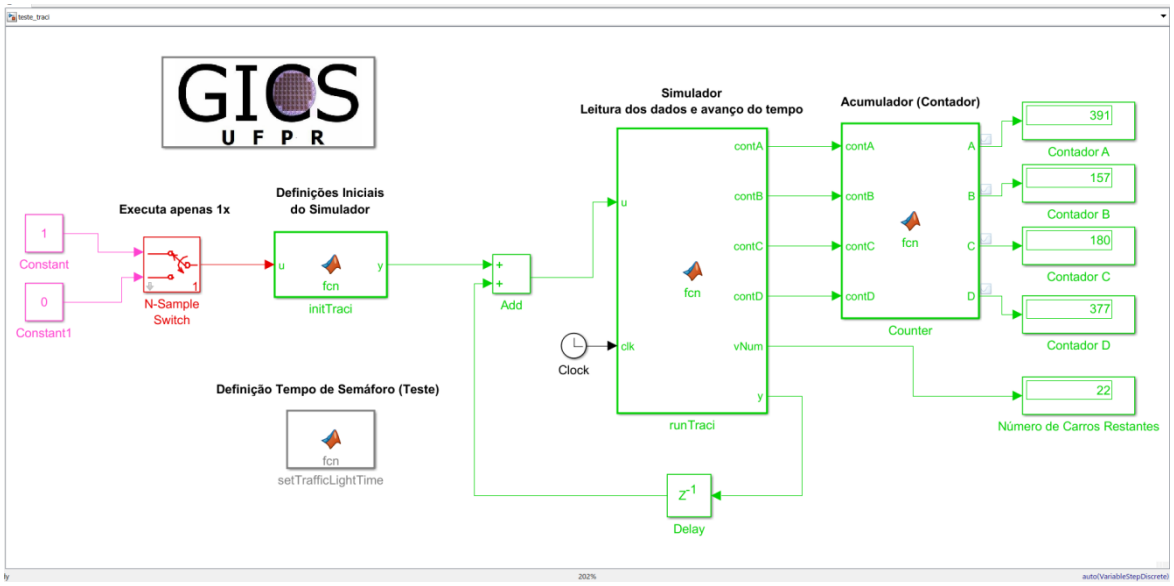
Fonte: dos autores.

FIGURA 25 – SIMULINK – IMPLEMENTAÇÃO EM *SOFTWARE* COM SIMULADOR DE TRÂNSITO



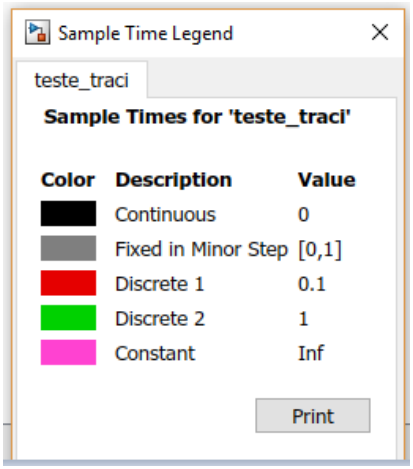
Fonte: dos autores.

FIGURA 26 – SIMULINK – IMPLEMENTAÇÃO EM *SOFTWARE* COM SIMULADOR DE TRÂNSITO COM TEMPO DE AMOSTRAGEM



Fonte: dos autores.

FIGURA 27 – LEGENDA PARA O TEMPO DE AMOSTRAGEM DO MODELO SIMULINK



Fonte: dos autores.

Na FIGURA 25 é exibido a implementação em *software* Simulink® com Simulador de Trânsito SUMO. Os primeiros blocos a partir da esquerda para direita, *Constant, Constant 1 e N-Sample Switch* são blocos auxiliares que são utilizados na inicialização do simulador, como o simulador utiliza a API (*Application Programming Interface*) TraCI (*Traffic Control Interface*), é necessário iniciar o servidor para interface de comunicação com o ambiente Simulink/Matlab, e também iniciar o simulador SUMO. Para isso, o bloco *initTraci* tem as configurações para as

definições iniciais do programa, e ele espera o valor 1 na entrada, onde após a primeira iteração do Simulink® a chave *N-Sample Switch* comutará para o valor 0, bloqueando o funcionamento do bloco *initTraci* nas próximas iterações.

O bloco *initTraci* na primeira instância, envia para a sua saída o valor 1, realizando a soma com o valor 0 do bloco principal *runTraci (Simulador),* onde o resultado é 1. Com isso, é possível o funcionamento do bloco principal *runTraci* (Simulador), onde será realizado os passos de tempo para execução do Simulador. O Simulink® irá aguardar até o Simulador for inicializado e executado. Após a primeira iteração, o bloco principal *runTraci* tem como saída o valor 1, onde realizará a soma com o valor 0 da saída do bloco inicial *initTraci*, no qual já não há mais necessidade para o programa.

Com isso, o sistema entre em modo *loop*, onde o bloco principal está sendo retroalimentado por ele mesmo, até o término da simulação. Para armazenar os dados da simulação, dentro do bloco principal *runTraci*, há a leitura da contagem de veículos para cada laço magnético (sensor) instalado nas vias do simulador, no início das vias e após o semáforo, totalizando em 4 sensores. Também tem como saída o valor atual do número restante de carros nas vias. Contudo, para armazenar e acumular os valores da contagem de veículos foi necessário utilizar outro bloco, o *Counter*. Nele é feito o armazenamento dos veículos contados, acumulando ao longo do tempo, assim foi possível testar duas funcionalidades: variável com memória e exibição do valor armazenado.

Para a mudança do tempo do semáforo, foi criado um bloco individual e isolado, onde atua diretamente no simulador, sem a necessidade de ligações externas, já que a simulação é apenas em *software*. Idealmente, a mudança do semáforo seria feita pelo bloco controlador *Q-Learning*, onde determinaria aleatoriamente a ação (tempo do semáforo) a cada ciclo de semáforo (sinal verde-vermelho-verde). No entanto, para efeitos de testes de implementação, o tempo do semáforo é comutado a cada ciclo de semáforo, iniciando com o tempo 10-10 segundos definido nas definições iniciais do bloco *initTraci* e variando entre 10- 60 segundos, 60-10 segundos. Com isso foi possível testar duas funcionalidades: mudança efetiva do semáforo no simulador pelo Simulink® e identificar o ciclo do semáforo.

Na FIGURA 27 é exibida a legenda dos tempos de amostragem, onde os blocos de cor:

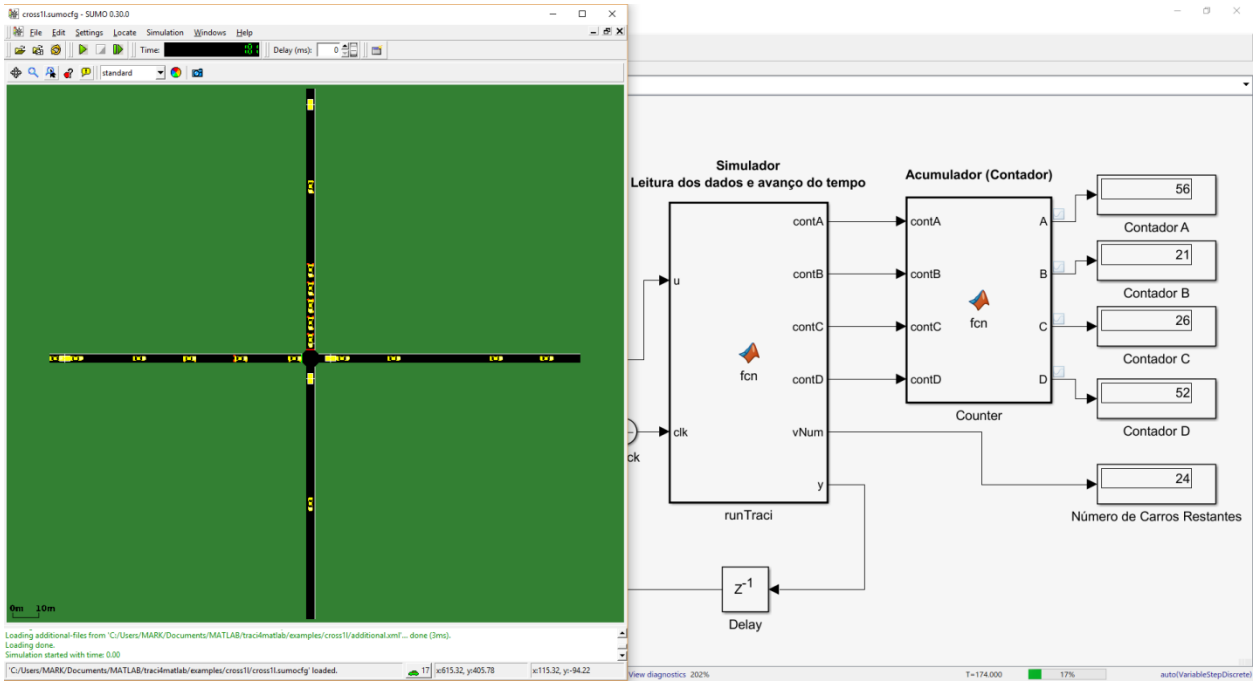
- Rosa representa valor constante, com tempo de amostragem infinito;
- Verde representa valor discreto, com tempo de amostragem 1;
- Vermelho representa valor discreto, com tempo de amostragem 0,1;
- Cinza representa valor de ponto-fixo, com tempo de amostragem 1;
- Preto representa valor contínuo, com tempo de amostragem 0 (não presente na simulação).

Na FIGURA 28 é exibido o Simulador em execução em conjunto com o Simulink®. A simulação foi definida em 1000 iterações, onde haverá em torno de 1000 veículos passando pela na intersecção. O resultado final da contagem é exibido na FIGURA 25, nos displays *Contador A, Contador B, Contador C e Contador D.*

Os blocos *initTraci, runTraci e Counter* são funções do MATLAB®, onde dentro deles há a execução de um código de extensão ".m" que funciona de maneira pontual no Simulink®. São códigos feitos no ambiente MATLAB® que tem o funcionamento diferente do Simulink®, onde a arquitetura do modelo influencia no desempenho e efetividade do programa. Foi necessário utilizar algumas funções especiais do Simulink® como o *coder.extrinsic()* para habilitar o código MATLAB® dentro da simulação.

Essa implementação teve como objetivo verificar as funcionalidades e limitações que o Simulink® pode oferecer na implementação em *hardware* FPGA. O bloco controlador *Q-Learning* será o tomador de decisões global do programa, portanto, foi necessário verificar os pontos que poderiam ser limitantes como armazenagem do valor da variável na memória, atuação do Simulink® no Simulador SUMO, leitura dos dados do Simulador e funcionamento em *loop* entre o Simulink® e o Simulador.

FIGURA 28 – SIMULADOR DE TRÂNSITO COM ATUAÇÃO DO SIMULINK NO TEMPO DO SEMÁFORO



Fonte: dos autores.

4 APRESENTAÇÃO DOS RESULTADOS

Para obter os resultados de comparação com o controle de tempo fixo foi necessário antes especificar e melhorar a função de recompensa do algoritmo.

A função de recompensa foi dividida em duas partes, a primeira analisa a razão de veículos que passaram por cada via, e fornece um valor numérico de recompensa dependendo da razão de carros que saiu em cada via conforme a TABELA 5. E a segunda parte analisa a melhor ação possível a ser tomada naquele estado, dando prioridade para a fila com maior tamanho.

TABELA 5 – ANÁLISE DAS RAZÕES DE SAÍDA DOS VEÍCULOS

Razão de quantos veículos passaram							Recompensa
Razão 1	Razão 2	Media	Variancia	Desvio Padrao	Total	Total(%)	R (0 a 10)
100%	100%	100,0%	0,0%	0,0%	30	100,0%	10
80%	80%	80,0%	0,0%	0,0%	24	80,0%	4
60%	60%	60,0%	0,0%	0,0%	18	60,0%	1
40%	40%	40,0%	0,0%	0,0%	12	40,0%	0
20%	20%	20,0%	0,0%	0,0%	6	20,0%	0
0%	0%	0,0%	0,0%	0,0%	0	0,0%	0
100%	80%	90,0%	2,0%	10,0%	27	90,0%	6
80%	60%	70,0%	2,0%	10,0%	21	70,0%	2
60%	40%	50,0%	2,0%	10,0%	15	50,0%	0
50%	50%	50,0%	0,0%	0,0%	15	50,0%	0
100%	0%	50,0%	50,0%	50,0%	15	50,0%	0
100%	20%	60,0%	32,0%	40,0%	18	60,0%	0
100%	40%	70,0%	18,0%	30,0%	21	70,0%	0
100%	50%	75,0%	12,5%	25,0%	22,5	75,0%	0
100%	60%	80,0%	8,0%	20,0%	24	80,0%	1
100%	70%	85,0%	4,5%	15,0%	25,5	85,0%	2
100%	80%	90,0%	2,0%	10,0%	27	90,0%	6
100%	90%	95,0%	0,5%	5,0%	28,5	95,0%	9

FONTE: dos autores

É interessante notar por mieo de uma análise da TABELA 5, que os valores de recompensa são fornecidos levando em consideração a variância e o desvio padrão da razão de veículos. Para complementar esta função foi adicionado um complemento para que o sistema receba recompensa quando tomar a melhor decisão possível daquele estado dando prioridade para a fila de maior tamanho. Isso foi feito analisando a proporção de veículos em cada via, para que mesmo quando

uma ação tomada não tenha uma saída muito grande de veículos, mas foi a melhor possível para aquele estado receba um valor de recompensa. A TABELA 6 mostra os valores das proporções encontradas, para um ciclo de 40 segundo de semáforo, e no máximo 15 carros em cada via.

TABELA 6 – ANÁLISE DAS RAZÕES DOS TEMPOS SEMAFÓRICOS

Tempo Verde na Via 1	Tempo Verde na Via 2	Razão
5	35	0,14
10	30	0,33
15	25	0,60
20	20	1,00
25	15	1,67
30	10	3,00
35	5	7,00

FONTE: dos autores

A TABELA 7 e TABELA 8 mostram as proporções de alguns possíveis estados, e o melhor tempo para estas proporções. Os melhores tempos para cada proporção foram definidos através de uma comparação das tabelas 6, 7 e 8. Foi desenvolvido um algoritmo de recompensa baseado no tamanho das filas de veículos das duas vias, via 1 e via 2, onde a proporção do tamanho da fila define a prioridade. Portanto, se uma das vias tem maior prioridade (maior fila), então o tempo verde deve ser maior para ela, ou menor, para o caso de uma menor prioridade em comparação a outra via.

TABELA 7 – ANÁLISE DAS RAZÕES DA VIA 1

Quantidade de Veículos na Via 1	Quantidade de Veículos na Via 2	Razão	Melhor estado possível (tempo verde em segundos)
15	15	1,00	20
15	14	1,07	
15	13	1,15	
15	12	1,25	
15	11	1,36	
15	10	1,50	
15	9	1,67	25
15	8	1,88	
15	7	2,14	
15	6	2,50	
15	5	3,00	30
15	4	3,75	
15	3	5,00	
15	2	7,50	35
15	1	15,00	
15	0	0,00	

FONTE: dos autores

TABELA 8 – ANÁLISE DAS RAZÕES DA VIA 2

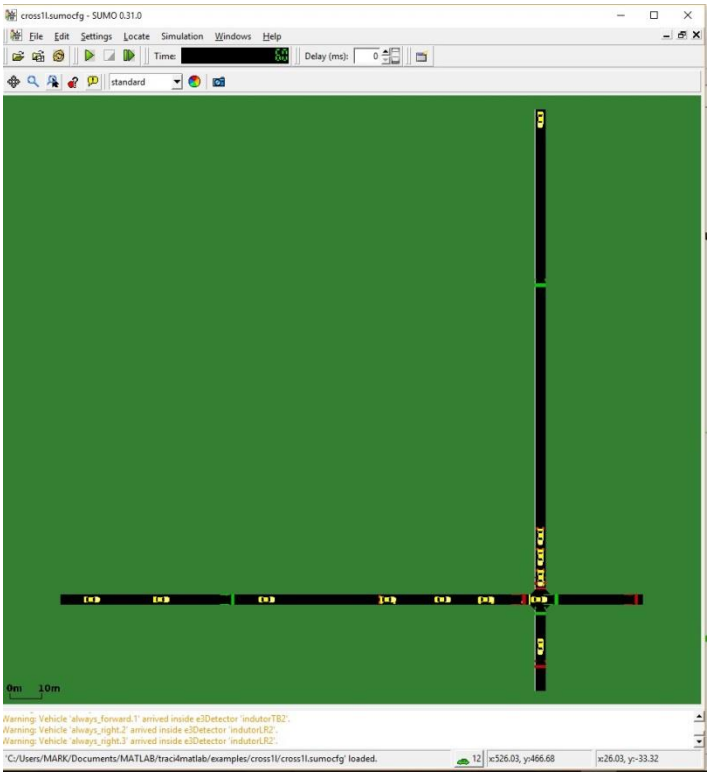
Quantidade de Veículos na Via 1	Quantidade de Veículos na Via 2	Razão	Melhor estado possível (tempo aberto em segundos)
15	15	1,00	20
14	15	0,93	
13	15	0,87	
12	15	0,80	
11	15	0,73	
10	15	0,67	
9	15	0,60	15
8	15	0,53	
7	15	0,47	
6	15	0,40	
5	15	0,33	10
4	15	0,27	
3	15	0,20	
2	15	0,13	5
1	15	0,07	
0	15	0,00	

FONTE: dos autores

Com a função de recompensa definida foram realizados os testes de comparação com tempo fixo. O ambiente de comparação foi montado utilizando o *software* SUMO mostrado na FIGURA 29, e definido com os seguintes parâmetros:

- *Tamanho médio dos veículos*: 5m;
- *Aceleração média dos veículos:* 1 metro por segundo;
- *Tamanho da via*: 100 metros;
- *Número máximo de veículos em cada via:* 15;
- *Tempo de simulação (simulador):* 8 horas;
- *Tempo de simulação (Real):* 22 minutos.

FIGURA 29 – SUMO AMBIENTE DE SIMULAÇÃO CONSTRUÍDO



Fonte: dos autores.

A distribuição de entrada de veículos foi feita da seguinte forma: foram definidas probabilidades de entrada de veículo por segundo, ou seja, se esta probabilidade estiver definida em 20%, por exemplo, significa que há 20% de chance de entrar um veículo a cada segundo nas vias.

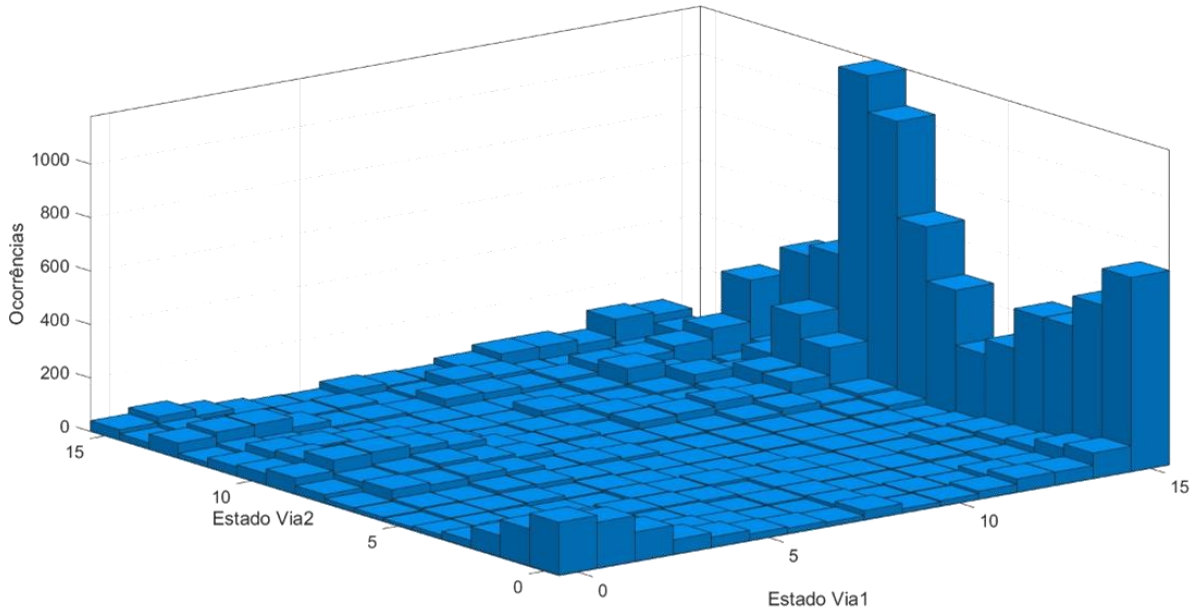
TABELA 9 – PROBABILIDADES DE ENTRADA DE VEÍCULOS

Horas	Distribuição Via1	Distribuição Via 2
1	20%	10%
2	10%	20%
3	5%	20%
4	5%	5%
5	18%	20%
6	6%	18%
7	2%	5%
8	16%	10%

FONTE: dos autores

Como pode ser vista na TABELA 9 a distribuição dos carros foi variada para experimentar diversos tipos de ambiente, e validar a os resultados do método aplicado. A FIGURA 30 mostra a distribuição dos estados encontrados durante a simulação. Neste ambiente não foram encontrados todos os estados possíveis, já que para isso seria necessário um número expressivo de tempo de simulação e probabilidades diversificadas de entrada de veículos.

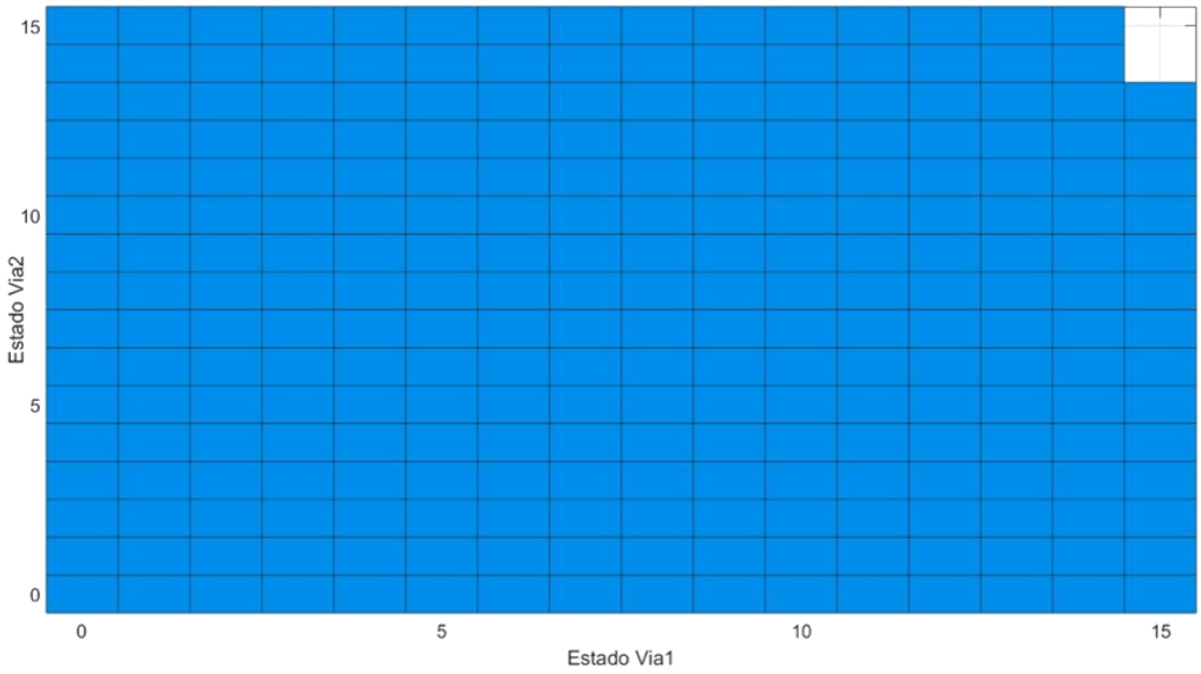
FIGURA 30 – HISTOGRAMA DOS ESTADOS ENCONTRADOS DURANTE O TREINO



Fonte: dos autores.

Foi aplicado dois método de tempo fixo, um foi fixado em 20 segundos de tempo semafórico verde para cada via, fechando um ciclo de 40 segundos, e outro foi tempos fixos variados de acordo com a definição da modelagem do ambiente, que se baseia na probabilidade da demanda de veículos de cada via . Já no método *Q-Learning* foi utilizada uma tabela Q treinada como uma política baseada no algoritmo de prioridade para a fila mais longa ao longo de 20 horas, ou 12 dias dentro do ambiente de simulação com probabilidades de entradas de veículos distintos. Este tempo de treino ainda não preencheu toda a tabela Q, já que nem todos os estados possíveis foram encontrados, como mostra a FIGURA 30. No entanto, para validar este resultado, considerou-se suficiente. Para cada ambiente diferente, seria necessário um valor diferente de tempo para treinar o sistema, analisando também o quanto este ambiente varia durante este tempo de análise. A FIGURA 31 mostra uma visão 2D dos estados que foram treinados no algoritmo de *Q-Learning* e a FIGURA 32 mostra o gráfico de aprendizado do sistema durante o treinamento.

FIGURA 31 – VISÃO 2D DOS ESTADOS ENCONTRADOS DURANTE O TREINO



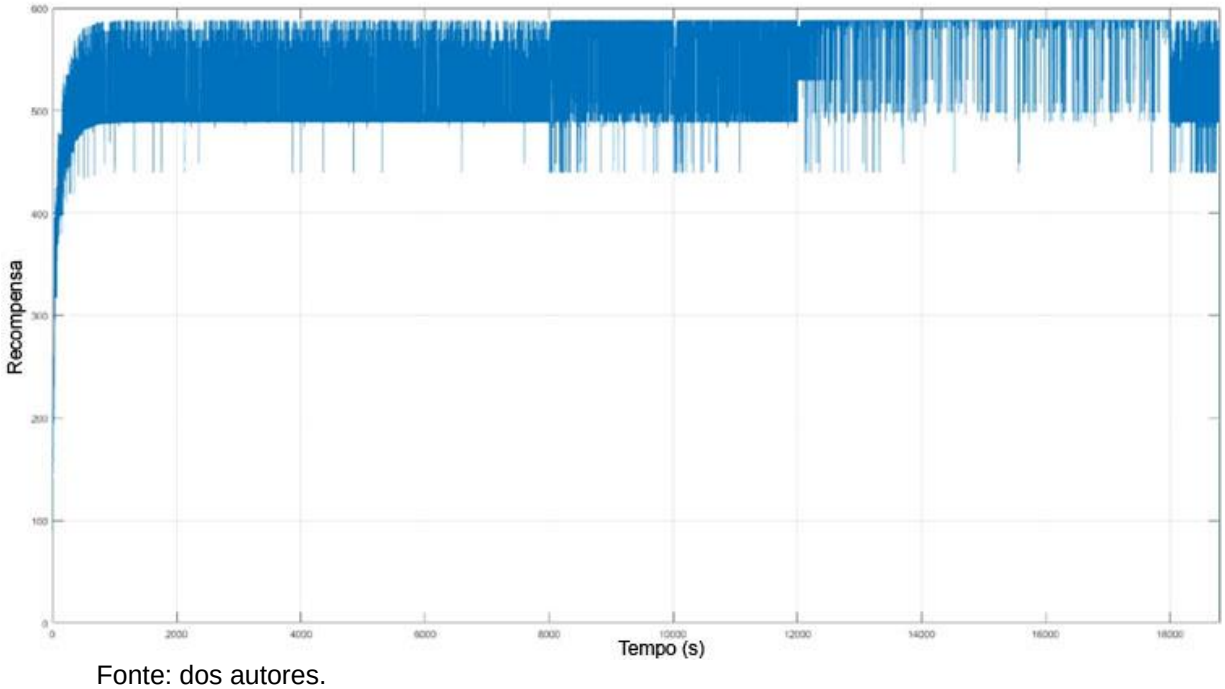
Fonte: dos autores.

Seria possível ainda implementar uma política de controle, para que o algoritmo tenha uma fase de exploração e outra de aproveitamento, desta forma

este algoritmo poderia ser utilizado ininterruptamente em uma esquina. Onde primeiramente haveria uma fase de exploração que o sistema tomaria ações aleatórias, e ganharia recompensas por estas ações, em seguida começaria uma fase de aproveitamento, tomando as melhores decisões conhecidas, aproveitando a tabela Q e ainda ganhando recompensa. Um método para realizar esta política é a chamada *Greedy Policy,* que simplesmente consiste em utilizar um coeficiente *épsilon*, que seria a probabilidade de exploração, iniciando sempre em 0,9 e através do número de iterações ele deve ser diminuído, resultando em um sistema que quanto mais iterações fossem realizadas, menor seria a probabilidade de exploração e maior a de aproveitamento do sistema.

Para esta comparação foi definido o coeficiente *épsilon* para 0, ou seja, apenas aproveitando os valores da tabela Q sem realizar nenhuma exploração. E para realizar uma comparação com tempo fixo foi necessário usar o mesmo ambiente, e não levar em consideração o tempo de aprendizado e ações tomadas durante este tempo.

FIGURA 32 – GRÁFICO DE APRENDIZADO DURANTE O TREINAMENTO



Fonte: dos autores.

Os resultados da comparação podem ser vistos na TABELA 10 e na FIGURA 33. Houve uma diminuição do tempo de espera em 7335% (-397,9 segundos) no tempo fixo e 3178% (169,27 segundos) no tempo fixo variado, o que significa que

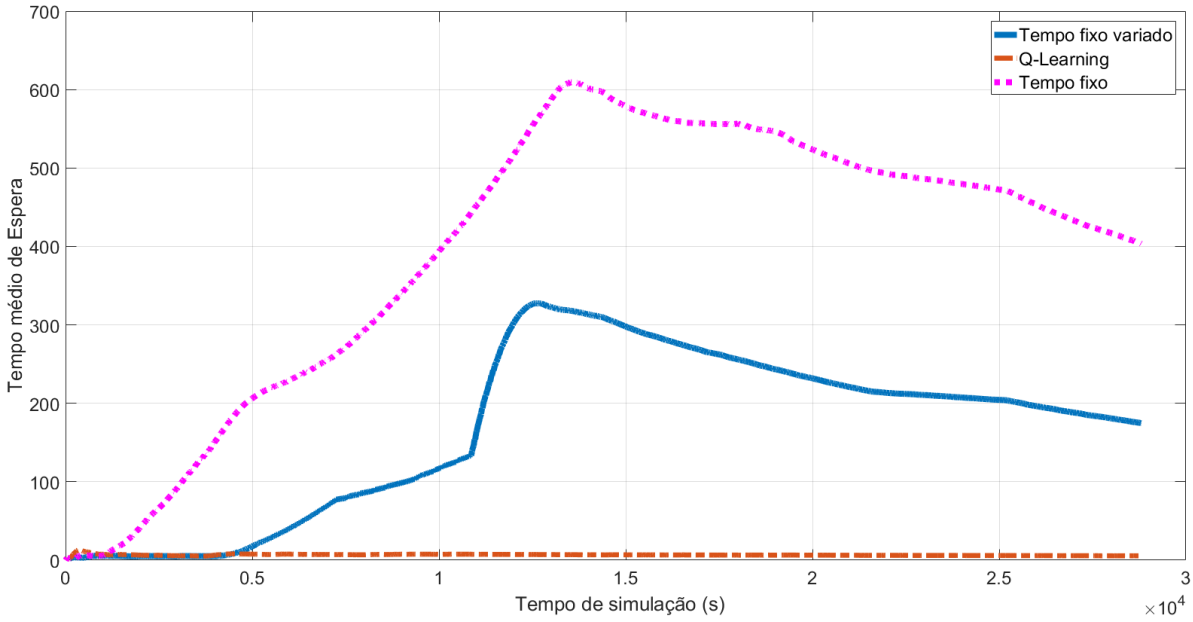
um motorista no ambiente controlado por tempo fixo teria aproximadamente um tempo de espera médio de 403,4 segundos no tempo fixo, e 174,77 segundos no tempo fixo variado, resultando em 8 e 4 ciclos de semáforo, enquanto no ambiente controlado pelo método QLC o tempo médio de espera é aproximadamente para 5,5 segundos, então o motorista esperaria em média um ciclo de semáforo para sair da intersecção.

TABELA 10 – COMPARAÇÃO COM TEMPO-FIXO

Método de Controle	Número de veículos que saíaram do ambiente	Tempo médio de espera(s)	Diferença
Tempo Fixo	6222	403,4	7335%
Tempo Fixo Var.	6232	174,77	3178%
Q-Learning	6098	5,5	

FONTE: dos autores

FIGURA 33 – COMPARAÇÃO ENTRE TEMPO-FIXO, TEMPO-FIXO VARIADO E *Q-LEARNING*



FONTE: dos autores

Isto representa uma melhora muito significativa, no entanto, algumas observações devem ser levadas em consideração. O ambiente foi modelado de forma a ser dinâmico, caso ele tivesse entradas constantes de veículos iguais nas duas vias, o tempo médio de espera dos dois métodos provavelmente estariam mais próximos. Para uma comparação com a realidade seria necessário modelar um

ambiente existente analisando o fluxo médio de veículos durante um período de tempo e também o tempo fixo de controle semafórico utilizado neste ambiente. Como este não era o objetivo desta pesquisa, esta atividade não foi realizada, mas é uma possibilidade de extensão do projeto.

Uma outra comparação que pode ser realizada é a de emissões. Um veículo em movimento emite menos poluentes ao ambiente, desta forma um controle inteligente de trânsito, onde os veículos ficam menos tempo esperando e mais tempo andando pode impactar nos valores de emissões. O próprio SUMO oferece ferramentas para análise de emissões, podendo ser adquiridos os valores de emissões como por exemplo: CO2, CO, HC, PMx e NOx.Para as simulações de tempo-fixo e do método de *Q-Learning* feitas, as emissões são mostradas na TABELA 11.

TABELA 11 – COMPARAÇÃO DE EMISSÕES

Método de Controle	Tempo de Simulação	Emissões CO2 Totais(mg)	Emissões CO Totais(mg)	Emissões HC Totais(mg)	Emissões PMX Totais(mg)	Emissões de NOX(mg)
Tempo-Fixo	8 horas	8,17E+08	3,31E+07	1,70E+05	1,62E+04	3,51E+05
Q-Learning	8 horas	5,66E+08	1,84E+07	9,69E+04	1,03E+04	2,37E+05
	Diferença	−144%	−180%	−175%	−157%	−148%

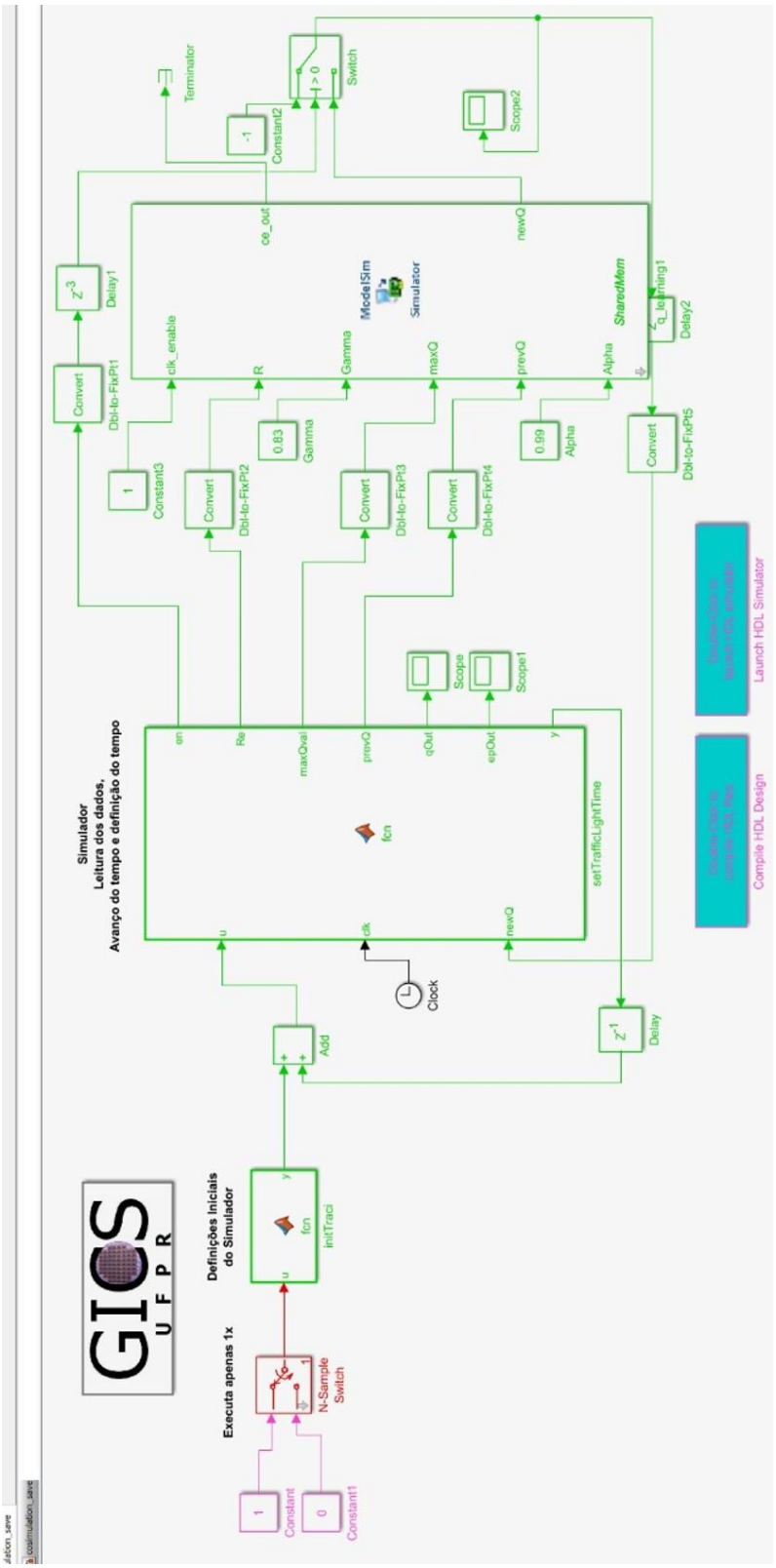
FONTE: dos autores

Estes dados refletem apenas uma esquina simples, o que significa que se essa aplicação for ampliada para uma cidade inteira os valores de emissão poderiam diminuir significativamente. Um trabalho futuro poderia ser realizado analisando o impacto destas emissões na saúde dos indivíduos que estão expostos no trânsito, agregando ainda mais valor à pesquisa realizada.

Para simulação do algoritmo em *hardware* realizou-se uma cossimulação dos blocos do sistema. Desta maneira, o comportamento da FPGA é emulado, permitindo assim analisar os resultados obtidos em *hardware* dentro do ambiente de simulação. Para isso, foram criados blocos simplificados, gerados códigos em VHDL, realizados os *Test Benchs*, e cossimulados através da metodologia explanada no capítulo de materiais e métodos. Não foi possível realizar a implementação FPGA-in-the-loop devido a problemas de comunicação entre o kit Genesys (contendo a FPGA) e a ferramenta de desenvolvimento.

A cossimulação foi realizada e seus valores de saída validados. O sistema montado para realizá-la é mostrado na FIGURA 34.

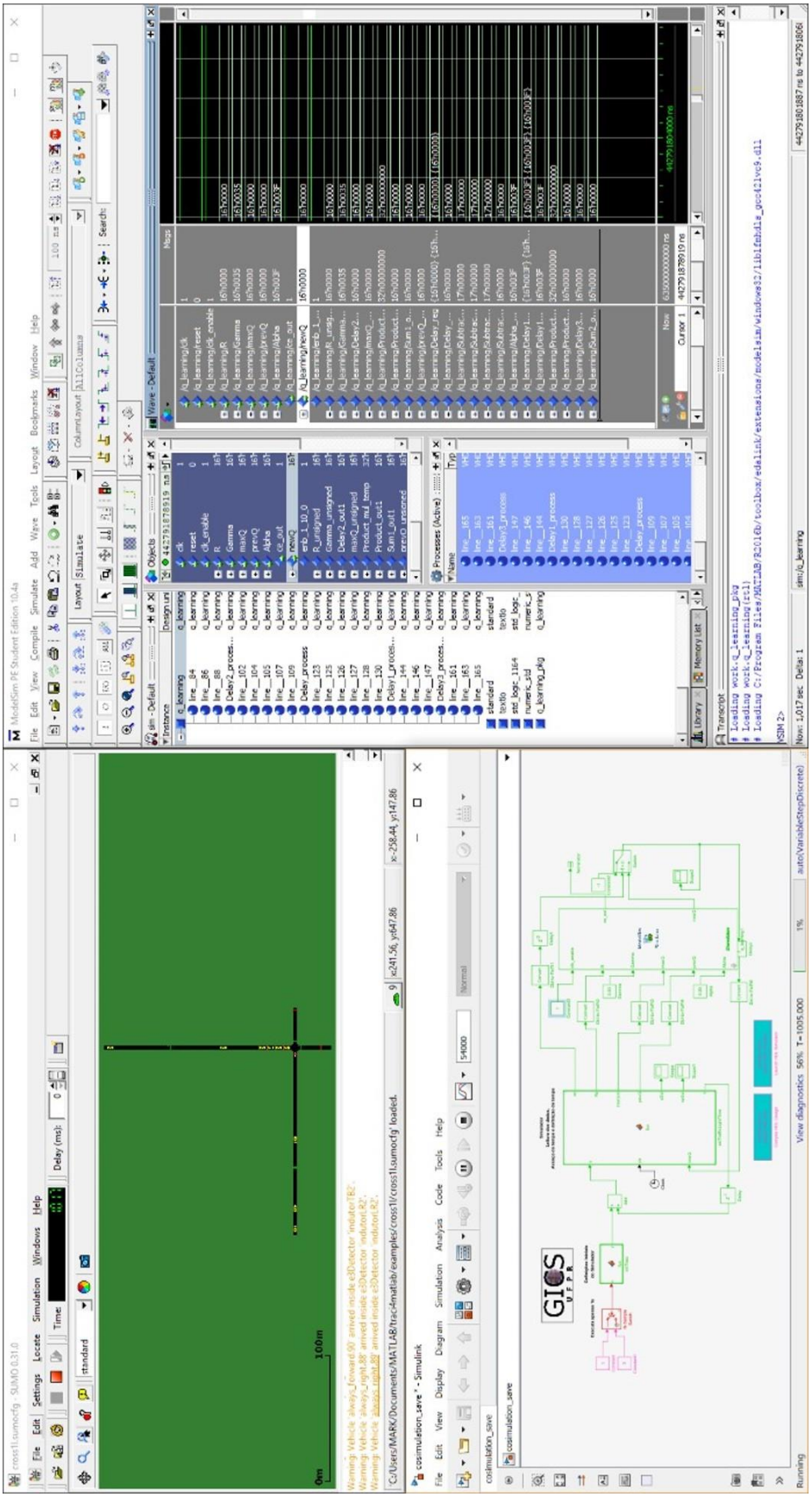
FIGURA 34 – SISTEMA DE COSSIMULAÇÃO



Fonte: dos autores.

Como visto no capitulo de Materiais e Métodos esta etapa exige uma atenção para a conversão de dados do tipo flutuante para ponto fixo e a configuração de clock do *hardware*. A saída do sistema com o simulador rodando junto com o sistema em cossimulação é mostrada na FIGURA 35. A ocupação de recursos lógicos foi baixa, já que a placa que estávamos especificando era robusta, uma Virtex 5 modelo VLX50T. Foi utilizado um total de 112 de 28000 *slices* de registradores e 48 LUTs de 28000 disponíveis, isto totaliza uma ocupação de 1% da capacidade desta placa.

FIGURA 35 – SISTEMA DE COSSIMULAÇÃO COM SIMULADOR



Fonte: dos autores.

5 CONSIDERAÇÕES FINAIS

A Inteligência Artificial é uma importante ferramenta computacional, e que pode ser aplicada nas mais variadas áreas. Este projeto buscou desenvolver um algoritmo de aprendizado de máquina que se encaixasse nos problemas atuais de trânsito de nossa região. Diminuir o tempo de espera, melhorar o fluxo no trânsito, diminuir o número de poluentes em nosso meio ambiente, minimizar o custo do transporte coletivo, são algumas das vantagens que esta pesquisa pode proporcionar a sociedade. Conforme revisado neste documento fica claro que o Brasil ainda está atrasado nesta área, mas se for oferecida uma solução de baixo custo e fácil implementação o mercado pode mostrar interesse. Não existe um consenso geral para definir melhor maneira computacional de controlar semáforos de trânsito, o *Q-Learning* mostra um caminho, já que o agente de aprendizado consegue tomar decisões com base na sua experiência e pode sempre aprender com novas interações com o ambiente.

Como visto na revisão bibliográfica este algoritmo já vem sendo construído e adaptado por mais de 20 anos (WATKINS, 1989) e como foi mostrado já existem várias implementações no controle de trânsito. Algumas abordagens resultaram em uma forma sólida e eficaz de controle, outras nem tanto.

Um dos maiores desafios neste projeto foi a utilização de técnicas computacionais e metodologias de desenvolvimento que não foram analisadas durante a graduação, o que resultou em um trabalho de integração de conhecimentos já aprendidos além de um projeto de pesquisa de metodologias e técnicas de aprendizado. A função de recompensa do sistema, por exemplo, poderia ser feita em um trabalho a parte, já que é alta a complexidade da formulação de uma função adequada e que valorize o sistema de forma igualitária e eficiente.

Foi mostrado também uma metodologia que abstrai a maioria dos conhecimentos necessários para realizar uma implementação em *hardware* diretamente do *software* de desenvolvimento MATLAB®, mesmo assim esta tarefa ainda demanda um tempo considerável, mas abre a possibilidade de expandir a implementação realizada para o controle completo criado em uma futura pesquisa.

Em suma este projeto mostrou uma maneira simples de implementação do algoritmo de *Q-Learning* no controle do fluxo de veículos, mas a falta de um

ambiente com mais complexidade e informações, evitou uma comparação igualitária do método discutido com o método de tempo fixo. No entanto, em um ambiente dinâmico é possível afirmar que este método utilizado e implementado apresenta reduções significativas no tempo de espera médio dos veículos.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

O projeto tem diversas possibilidades de extensão, como modelagem do ambiente para obter uma melhor aproximação do mundo real, com geração de resultados mais precisos e aplicáveis na prática. Uma melhoria que poderia ser aplicada seria utilizar uma intersecção mais complexa, com mais vias e mais estados de semáforo. Também seria interessante utilizar demandas de veículos de modelos já consolidados que têm uma exploração mais ampla de estados ou ainda a utilização de modelos reais.

Na literatura mais recente (GENDERS e RAZAVI, 2016), apontam que o método *Deep Reinforcement Learning* tem resultados mais eficazes e precisos, com possibilidade de ter menos esforço nas análises e parametrizações no qual é geralmente feito caso a caso para cada tipo de ambiente modelado.

A aplicação do *Deep Reinforcement Learning* seria o ideal, com resultados mais significativos que traria um aprendizado de uma política mais justa em relação ao tempo de espera dos veículos nas intersecções. Também é possível melhorar a definição de estados, com aplicação de pesos para diferentes tipos de veículos, bem como modelagem do ambiente para pedestres, ciclistas, motocicletas, ônibus, caminhões e entre outros tipos de automóveis que tornariam o ambiente mais próximo da realidade.

Em relação a viabilidade de processamento paralelo ao projeto, como a utilização do *hardware* FPGA, seria interessante se houvesse a aplicação do método *Deep Reinforcement Learning*, pois ele utiliza-se da rede neural convolucional, inspirado nos processos biológicos do córtex visual, no qual a tendência é abstrair informações das imagens que não são tão relevantes e foca-se apenas nas informações essenciais para o cálculo do resultado. Com isso, o processamento paralelo se tornaria eficiente, pois sistemas com processamento de imagens, ou processamento de matrizes, geralmente não precisam ser necessariamente processadas sequencialmente. Outra possibilidade seria aplicar diversos ambientes

iguais em simulações paralelas, que seria capaz de gerar múltiplos resultados de diferentes estados e ações, no entanto haveria um esforço extra para integrar estes resultados.

Outra viabilidade para utilização do processamento paralelo do *hardware* FPGA seria utilizá-lo para contagem de veículos através de processamento de imagens, no qual seriam obtidas através de câmeras instaladas nas vias públicas e que poderiam ser uma alternativa barata para o caso de utilização de CIs (circuitos integrados). Um dos grandes motivos de não existir aplicações de controle de trânsito aplicados nos semáforos nas vias públicas é pelo alto custo de instalação de sensores e controladores de baixo custo.

No decorrer do projeto ouve um desenvolvimento inicial para a contagem de veículos utilizando processamento de imagens, mas que não coube citar nesta pesquisa, pois o projeto focou-se no desenvolvimento do controle de tráfego de veículos em conjunto com *hardware* FPGA.

REFERÊNCIAS

LIVROS

CROWLEY, J.; CHRISTENSEN, H.; CHEHIKIAN, A. Vision as process. Berlin: Springer-Verlag, 1995.

JOHNSON, S. Stephen Johnson on **Digital Photography**. Oreilly, 2006.

SUTTON, RICHARD S.; BARTO, ANDREW G. ***Reinforcement Learning*: An Introduction**. (1998).

THORPE, T. L., & ANDERSON, C. W. **Traffic light control using SARSA with three state representations. Technical Report**. IBM Corporation. (1996).

WATKINS, C. e DAYAN, P. (1992). ***Q-Learning*. Machine Learning,** v.3/4, n.8, p.279-292.

ARTIGOS DE PERIÓDICOS

ABDI, J., MOSHIRI, B., ABDULHAI, B., & SEDIGH, A. **Short-term traffic flow forecasting: Parametric and nonparametric approaches via emotional temporal difference learning**. Neural Computing and Applications (2013) v.23, n.1, p.141–159

ABDOOS, M., MOZAYANI, N., e BAZZAN, A. L. Holonic multi-agent system for traffic signals control. Engineering Applications of Artificial Intelligence, v.26, n.2, p.575–1587, 2013.

ABDULHAI, B., PRINGLE, R., e KARAKOULAS, G. J. **Reinforcement learning for true adaptive traffic signal control.** Journal of Transportation Engineering, v.129, n.3, p.278–285, 2003.

ARAGHI, S.; KHOSRAVI, A.; CREIGHTON, D. **A review on computational intelligence methods for controlling traffic signal timing**. Expert Systems with Applications, v. 42, n. 3, p. 1538-1550, 2015.

AREL, I., LIU, C., URBANIK, T., e KOHLS, A. **Reinforcement learning-based multi-agent system for network traffic signal control**. IET Intelligent Transport Systems, v.4, n.2, p.128–135, 2010.

BALAJI, P., GERMAN, X., e SRINIVASAN, D. **Urban traffic signal control using reinforcement learning agents**. IET Intelligent Transport Systems, v.4, n.3, p.177–188, 2010.

BARTO, A., G., e MAHADEVAN, S. **Recent advances in hierarchical reinforcement learning.** Discrete Event Dynamic Systems, v.13, n.4, p.341–379, 2003.

BAXTER, J., BARTLETT, P. L., e WEAVER, L. **Experiments with infinite-horizon, policy-gradient estimation.** Journal of Artificial Intelligence Research, v.15 p.351–381, 2001.

BROCKFELD, E., BARLOVIC, R., SCHADSCHNEIDER, A., e SCHRECKENBERG, M. **Optimizing traffic lights in a cellular automaton model for city traffic.** Physical Review E, v.64, n.5, p.56-132, 2001.

CHIN, Y. K., BOLONG, N., KIRING, A., YANG, S. S., e TEO, K. T. K. ***Q-Learning* based traffic optimization in management of signal timing plan.** International Journal of Simulation, Systems, Science & Technology, v.12, n.3, p.29–35, 2011.

DEGRIS, T., PILARSKI, P. M., e SUTTON, R. S. **Model-free reinforcement learning with continuous action in practice.** In American Control Conference (ACC), 2012, p.2177–2182. IEEE, 2012.

DENG, L. **A tutorial survey of architectures, algorithms, and applications for deep learning** – ERRATUM. APSIPA Transactions on Signal and Information Processing, v.3, 2014.

DUGGAN, M., DUGGAN, J., HOWLEY, E., e BARRETT, E. **An autonomous network aware vm migration strategy in cloud data centres**. In Cloud and Autonomic Computing (ICCAC), 2016 International Conference on, p.24–32. IEEE, 2016.

DUGGAN, M., FLESK, K., DUGGAN, J., HOWLEY, E., e BARRETT, E. **A reinforcement learning approach for dynamic selection of virtual machines in cloud data centers.** In Sixth International Conference on Innovating Computing Technology. IEEE, 2016.

EL-TANTAWY, S., ABDULHAI, B., & ABDELGAWAD, H. **Multiagent *Reinforcement Learning* for integrated network of adaptive traffic signal controllers** (MARLINATSC): Methodology and large-scale application on downtown Toronto. IEEE Transactions on Intelligent Transportation Systems, p. 14, 1140–1150. (2013).

EL-TANTAWY, S., ABDULHAI, B., e ABDELGAWAD, H.. **Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown toronto.** IEEE Transactions on Intelligent Transportation Systems, 14(3):1140–1150, 2013.

GHAZANFARI, B. e MOZAYANI, N. **Enhancing nash *Q-Learning* and team *Q-Learning* mechanisms by using bottlenecks**. Journal of Intelligent & Fuzzy Systems, v.26, n.6, p.2771–2783, 2014.

GHAZANFARI, B. e MOZAYANI, N. **Extracting bottlenecks for reinforcement learning agent by holonic concept clustering and attentional functions.** Expert Systems with Applications, v.54, p.61–77, 2016.

KRAJZEWICZ, D., ERDMANN, J., BEHRISCH, M., e BIEKER, L. **Recent development and applications of sumo-simulation of urban mobility.** International Journal on Advances in Systems and Measurements, 5(3&4), 2012.

LANGE, S. e RIEDMILLER, M. **Deep auto-encoder neural networks in reinforcement learning.** In Neural Networks (IJCNN), The 2010 International Joint Conference on Barcelona, Spain, IEEE, p.1–8, 2010.

LECUN, Y., BOTTOU, L., BENGIO, Y., e HAFFNER, P. **Gradient-based learning applied to document recognition.** Proceedings of the IEEE, 86(11):2278–2324, 1998.

LI, L. e WEN, D. **Parallel systems for traffic control: A rethinking.** IEEE Transactions on Intelligent Transportation Systems, v.17, n.4, p.1179–1182, 2016.

LI, L. e WEN, D., e YAO, D. **A survey of traffic control with vehicular communications.** IEEE Transactions on Intelligent Transportation Systems, v.15, n.1, p.425–432, 2014.

LI, L., LV, Y., e WANG, F., Y. **Traffic signal timing via deep reinforcement learning.** IEEE/CAA Journal of Automatica Sinica, v.3, n.3, p.247–254, 2016.

LIFENG H. E, Y. C.; SUZUKI, K. A **run-based two-scan labeling algorithm**. IEEE Transaction on Image Processing, v.17, p.749 – 756, 2008.

LIN, L.-J. **Reinforcement learning for robots using neural networks.** PhD thesis, Fujitsu Laboratories Ltd, 1993.

MANNION P., SHAKSHUKI E., DUGGAN J., HOWLEY E., **Parallel Reinforcement Learning for Traffic Signal Control**, v.52, 2015, p. 956-961.

MNIH, V., BADIA, A.P., MIRZA, M., GRAVES, A., LILLICRAP, T., HARLEY, T., SILVER, D.; KAVUKCUOGLU, K. (2016). **Asynchronous Methods for Deep Reinforcement Learning**. Proceedings of The 33rd International Conference on Machine Learning, in PMLR, New York, USA, v.48, p.1928-1937.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL. **Human-level control through deep reinforcement learning.** Nature, v.518, n.7540, p.529–533, 2015.

MOUSAVI, S. S., GHAZANFARI, B., MOZAYANI, N., e JAHED-MOTLAGH, M. R. **Automatic abstraction controller in reinforcement learning agent via automata**. Applied Soft Computing, v.25, p.118–128, 2014.

MOUSAVI, S. S., SCHUKAT, M., e HOWLEY, E. **Deep reinforcement learning: An overview**. Intelligent Systems Conference, Springer, Cham, Switzerland, 2016.

MOUSAVI, S., SCHUKAT, M. AND HOWLEY, E. (2017). Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems*, 11(7), pp.417-423.

PRASHANTH, L. e BHATNAGAR, S**. Reinforcement learning with function approximation for traffic signal control.** IEEE Transactions on Intelligent Transportation Systems, 12(2):412–421, 2011.

RICHARD E. WOODS, R. C. G. **Processamento de imagens digitais**, 3a Edição Pearson, 2010.

RITCHER, S. **Traffic light scheduling using policy-gradient reinforcement learning.** In The International Conference on Automated Planning and Scheduling, ICAPS, Providence, Rhode Island, USA, 2007.

SANTOS, R. **Introdução ao processamento de imagens digitais em Java com aplicações em ciências espaciais**, 2009.

SCHULMAN, J., HEESS, N., WEBER, T., e ABBEEL, P. **Gradient estimation using stochastic computation graphs.** In Advances in Neural Information Processing Systems, pages 3528–3536, 2015.

SILVA LUCILEIDE M. D., TORQUATO MATHEUS F., FERNANDES MARCELO A. C., **Proposta de arquitetura em *hardware* para FPGA da técnica *Q-Learning* de Aprendizagem por Reforço**, XIII Encontro Nacional de Inteligência Artificial e Computacional, SBC ENIAC (2016).

SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, PANNEERSHELVAM, I., V., LANCTOT, M., ET AL. **Mastering the game of go with deep neural networks and tree search**. Nature, v.529, n.7587, p.484–489, 2016.

SUTTON, R. S. e BARTO, A. G. **Introduction to Reinforcement Learning**. MIT Press, Cambridge, MA, 1998.

SUTTON, R. S., MCALLESTER, D. A., SINGH, S. P., MANSOUR, Y., ET AL. **Policy gradient methods for reinforcement learning with function approximation.** In NIPS'99 Proceedings of the 12th International Conference on Neural Information Processing Systems, v,99, p.1057-1063, Denver, CO, USA,1999.

TSITSIKLIS, J. N., VAN ROY, B., ET AL. **An analysis of temporal-difference learning with function approximation.** IEEE transactions on automatic control, v.42, n.5, p.674–690, 1997.

VAN DER POL, E. e OLIEHOEK, F. A. **Coordinated deep reinforcement learners for traffic light control.** In NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems on Barcelona, Spain, 2016.

WIERING, M. A. (2000). **Multi-agent reinforcement learning for traffic light control**. In Langley, P. (Ed.), Proceedings of the Seventeenth International Conference on Machine Learning (ICML'2000), Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, pp. 1151–1158.

WIERSTRA, D., FORSTER, A., PETERS, J., e SCHMIDHUBER, J. **Recurrent policy gradients.** Logic Journal of IGPL, v.18, n.5, p.620–634, 2010.

WILLIAMS, R. J**. Simple statistical gradient-following algorithms for connectionist reinforcement learning**. Machine learning, v.8, n.3-4, p.229–256, 1992.

WUNDERLICH, R., LIU, C., ELHANANY, I., & URBANIK, T. **A novel signal-scheduling algorithm with quality-of-service provisioning for an isolated intersection.** IEEE Transactions on Intelligent Transportation Systems**,** p. 9, 536–547. (2008).

XU, X., ZUO, L., e HUANG, Z. **Reinforcement learning algorithms with function approximation: Recent advances and applications**. Information Sciences, v.261, p.1–31, 2014.

YANG, JUN, WANG, Y. S. A. L. Z. Z. B. X. J. **Feature fusion for vehicle detection and tracking with low angle cameras**. Applications of Computer Vision (WACV), 2011 IEEE, p. 382– 388, 2011.

YIT, K. C., NURMIN, B., AROLAND, K., SOO S. Y., KENNETH T. K. T. ***Q-Learning* Based Traffic Optimization in Management of Signal Timing Plan**. International Journal Of Simulation: Systems, Science & Technology, JUNE (2011).

NORMAS TÉCNICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724**: Informação e documentação — Trabalhos acadêmicos — Apresentação. 2011.

UFPR, Orientação Para Normalização De Trabalhos Acadêmicos. Disponível em < http://www.portal.ufpr.br/normalizacao.html>. Acesso em: data (1 abr. 2016).

DOCUMENTOS CONSULTADOS ONLINE

CAMERITE. Camerite. **Câmeras Públicas**, 17 Setembro 2016. Disponivel em: <https://camerite.com/cameras/594/curitiba/parana/avenida-visconde-de-guarapuava-n-1877>. Acesso em: 16 Setembro 2016.

FEITOSA, Francisco Coelho Citó. **Um estudo prático para contagem volumétrica automática de veículos usando Visão Computacional**. 2012. 140 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Goiás, Goiânia, 2012. Disponível em: Acesso em: 17 nov. 2015.

FIRJAN. "O custo dos deslocamentos nas principais áreas urbanas do Brasil (2015). Disponível em: http://www.firjan.com.br/lumis/portal/file/fileDownload.jsp?fileId=2C908A8F4F8A7DD3014FB26C8F3D26FE&inline=1 Acesso em: data (10 mai. 2016).

GAO, J., SHEN, Y., LIU, J., ITO, M., SHIRATORI, N. **Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network**. Published 2017 in ArXiv – Disponível em: < https://arxiv.org/pdf/1705.02755.pdf>. Acesso em: data (20 nov. 2017).

GENDERS, W. e RAZAVI, S. **Using a deep reinforcement learning agent for traffic signal control.** arXiv preprint arXiv:1611.01142, 2016 – Disponível em < https://arxiv.org/pdf/1611.01142.pdf>. Acesso em: data (20 nov. 2017).

JON SCHMITZ. **City expands use of high-tech traffic signals**. Pittsburgh Post-Gazette (2013). Disponível em <http://www.post-gazette.com/news/transportation/2014/05/02/City-expands-use-of-high-tech-traffic-signals/stories/201405020123>. Acesso em: data (02 abr. 2016).

JOSÉ RENATO SALATIEL. **Mobilidade urbana: Como solucionar o problema do trânsito nas metrópoles**. UOL – Atualidades, 2012. Disponível em: <http://vestibular.uol.com.br/resumo-das-disciplinas/atualidades/mobilidade-urbana-como-solucionar-o-problema-do-transito-nas-metropoles.htm> Acesso em: data (2 abr. 2016).

KEITH BARRY. **The Traffic Lights of Tomorrow Will Actively Manage Congestion**. Citylab (2014). Disponível em: <http://www.citylab.com/commute/2014/09/the-traffic-lights-of-tomorrow-will-actively-manage-congestion/379950/> Acesso em: data (1 abr. 2016).

KINGMA, D. e ADAM, J. BA.: **A method for stochastic optimization.** arXiv preprint arXiv:1412.6980, 2014. Disponível em <https://arxiv.org/pdf/1412.6980.pdf>. Acesso em: data (20 nov. 2017).

LEE DAVIDSON. **Odds of hitting a red light in Utah? Just 1-in-4**. The Salt Lake Tribune (2013). Disponível em: <http://archive.sltrib.com/story.php?ref=/sltrib/politics/57276984-90/traffic-percent-system-state.html.csp> Acesso em: data (1 abr. 2016).

MATHWORKS. HDL Coder. **Generate Verilog and VHDL code for FPGA and ASIC designs**, 2017. Disponivel em: <https://www.mathworks.com/products/hdl-coder.html>. Acesso em: 20 Março 2017.

MATHWORKS. Third-Party Products & Services. **Xilinx System Generator for DSP**, 2017. Disponivel em: <https://www.mathworks.com/products/connections/product_detail/product_35567.html>. Acesso em: 20 Março 2017.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., e RIEDMILLER, M. **Playing atari with deep reinforcement learning.** arXiv preprint arXiv:1312.5602, 2013. Disponível em < https://arxiv.org/pdf/1312.5602.pdf>. Acesso em: data (20 nov. 2017).

NUTAQ. HDL Coder and Xilinx System Generator. **An Overview of MATLAB HDL Coder and Xilinx System Generator**, 2016. Disponivel em: <https://www.nutaq.com/matlab-hdl-coder-xilinx-system-generator>. Acesso em: 20 Março 2017.

OPENCV. Open Source Computer Vision. **Background Substraction**, 2015. Disponivel em: <http://docs.opencv.org/3.1.0/db/d5c/tutorial_py_bg_subtraction.html>. Acesso em: 18 Setembro 2016.

PREFEITURA DE MOGI DAS CRUZES. **“Semáforo inteligente” utiliza tecnologia para garantir fluidez e segurança ao trânsito** (2015). Disponível em: <http://www.mogidascruzes.sp.gov.br/comunicacao/noticia.php?id=8977> Acesso em: data (02 abr. 2016).

STAUFFER, C.; GRIMSON, W. **Learning patterns of activity using real time tracking**, 2000. 747-767.

STAUFFER, C.; GRIMSON, W. The Artificial Intelligence Laboratory. **Adaptive background mixture models for real-time tracking**, Cambridge, MA 02139, USA, 1999. 246-252.

STEPHEN SMITH, GREG BARLOW, XIAO-FENG XIE, DR. DEBORAH D. STINE, DR. ENES HOSGOR. **SURTRAC Smart Traffic Light - Final Report** (2014). Disponível em: <https://www.cmu.edu/epp/people/faculty/course-reports/SURTRAC%20Final%20Report.pdf> Acesso em: data (02 abr. 2016).

SUMO, Simulation of Urban MObility (2017). Disponível em: <http://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki> Acesso em: data (02 nov. 2017).

THE MATHWORKS INC. Documentation. **Fullfile**, 2016. ISSN 5. Disponivel em: <http://www.mathworks.com/help/matlab/ref/fullfile.html?searchHighlight=fullfile>. Acesso em: 17 Setembro 2016.

THE MATHWORKS INC. Documentation. **IM2BW**, 2016. Disponivel em: <http://www.mathworks.com/help/images/ref/im2bw.html>. Acesso em: 17 Setembro 2016.

THE MATHWORKS INC. Documentation. **IMREAD**, 2016. Disponivel em: <http://www.mathworks.com/help/matlab/ref/imread.html?>. Acesso em: 17 Setembro 2016.

THE MATHWORKS INC. Documentation. **Imwrite**, 2016. Disponivel em: <http://www.mathworks.com/help/matlab/ref/imwrite.html>. Acesso em: 17 Setembro 2016.

THE MATHWORKS INC. Documentation. **MEDFILT2**, 2016. Disponivel em: <http://www.mathworks.com/help/images/ref/medfilt2.html>. Acesso em: 17 Setembro 2016.

THE MATHWORKS INC. Documentation. **READ, VideoReader.read**, 2016. ISSN 4. Disponivel em: <http://www.mathworks.com/help/matlab/ref/videoreader.read.html>. Acesso em: 16 Setembro 2016.

THE MATHWORKS INC. Documentation. **RGB2GRAY**, 2016. ISSN 7. Disponivel em: <http://www.mathworks.com/help/matlab/ref/rgb2gray.html>. Acesso em: 17 Setembro 2016.

THE MATHWORKS INC. Documentation. **Struct**, 2016. ISSN 3. Disponivel em: <http://www.mathworks.com/help/matlab/ref/struct.html>. Acesso em: 18 Setembro 2016.

THE MATHWORKS INC. Documentation. **VideoWriter**, 2016. Disponivel em: <http://www.mathworks.com/help/matlab/ref/videowriter.html>. Acesso em: 16 Setembro 2016.

THE MATHWORKS, INC. Documentation. **VideoReader**, 2016. Disponivel em: <http://www.mathworks.com/help/matlab/ref/videoreader.html>. Acesso em: 16 Setembro 2016.

THORPE, T. L. e ANDERSON, C. W. **Traffic light control using sarsa with three state representations**. Technical report, IBM Corporation, Department of Computer Science Colorado State University Fort Collins, Citeseer, 1996. Disponível em < https://pdfs.semanticscholar.org/c3cb/5362c9b4e0c8ab66feaf6dfbc295cd073d2a.pdf >. Acesso em 20 nov. 2017.

UN-HABITAT. **State of Latin American and Caribbean cities** (2012). Disponível em: <http://unhabitat.org/books/state-of-latin-american-and-caribbean-cities-2/> Acesso em: data (10 mai. 2016).

WREN, C. R.; AZARBAYEJANI, T. D.; PENTLAND, A. P. **Realtime tracking of the human body**, 1997. 780-785.

XILINX. User Guide. **System Generator for DSP**, 2012. Disponivel em: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/sysgen_user.pdf>. Acesso em: 20 Março 2017.

YORIFUJI, T., KAWACHI, I., KANEDA, M., TAKAO, S., KASHIMA, S. AND DOI, H. (2011). Diesel vehicle emission and death rates in Tokyo, Japan: A natural experiment. Science of The Total Environment, 409(19), pp.3620-3627.