UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

IGOR NAGAMASSA VIEIRA YAMAZAKI

IMPLEMENTAÇÃO DE AGENTES ÉTICOS NA RESOLUÇÃO DE CONFLITOS ENVOLVENDO VEÍCULOS AUTÔNOMOS EM UM CRUZAMENTO DE TRÂNSITO.

PONTA GROSSA

2026

versão 11.0 (abr. 25)

IGOR NAGAMASSA VIEIRA YAMAZAKI

IMPLEMENTAÇÃO DE AGENTES ÉTICOS NA RESOLUÇÃO DE CONFLITOS ENVOLVENDO VEÍCULOS AUTÔNOMOS EM UM CRUZAMENTO DE TRÂNSITO.

Implementation of Ethical Agents in the Resolution of Conflicts Involving Autonomous Vehicles at a Traffic Intersection

Trabalho de Conclusão de Curso de Graduação apresentado (a) como requisito para obtenção do título de Bacharel em Ciência da Computação do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Prof. Dr. Gleifer Vaz Alves.

PONTA GROSSA

2026

IGOR NAGAMASSA VIEIRA YAMAZAKI

IMPLEMENTAÇÃO DE AGENTES ÉTICOS NA RESOLUÇÃO DE CONFLITOS ENVOLVENDO VEÍCULOS AUTÔNOMOS EM UM CRUZAMENTO DE TRÂNSITO.

Trabalho de Conclusão de Curso de Graduação apresentado (a) como requisito para obtenção do título de Bacharel em Ciência da Computação do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 21 / maio / 2026

Prof. Dr. Gleifer Vaz Alves
Doutorado
Universidade Tecnológica Federal do Paraná (UTFPR)

Prof. Dr. André Pinz Borges
Doutorado
Universidade Tecnológica Federal do Paraná (UTFPR)

Alexandre Lizieri Leite Mellado
Mestrado
Universidade Tecnológica Federal do Paraná (UTFPR)

PONTA GROSSA

2026

AGRADECIMENTOS

Gostaria de agradecer à minha família. Meu pai e minha mãe foram a base de tudo isso — não só pelo apoio financeiro, mas por cada “você consegue“ dito nos momentos em que eu mesmo duvidava. Não existe forma justa de colocar em palavras o quanto a presença de vocês pesou para que eu chegasse até aqui.

Ao meu orientador, Prof. Dr. Gleifer Vaz Alves, que me guiou em um tema que exige fôlego e paciência.

E agradeço também aos colegas de curso que, em algum momento, dividiram dúvidas, risadas e muito café. Nenhuma graduação se faz sozinho, e parte do que aprendi veio dessas conversas.

“The great discovery that launched the Scientific Revolution was the discovery that humans do not know the answers to their most important questions.”
(HARARI, Yuval Noah, 2011).

RESUMO

Este trabalho apresenta o desenvolvimento e validação de um sistema multiagente (SMA) capaz de realizar raciocínio ético em decisões envolvendo veículos autônomos em cruzamentos urbanos. O problema investigado surge da necessidade de incorporar princípios morais em sistemas autônomos que interagem com vidas humanas, garantindo conformidade com diretrizes éticas e aceitação social. O objetivo geral consiste em modelar e implementar um SMA seguindo a arquitetura de agente ético proposta por Alves, estendendo-a através da aplicação do Triângulo de Galtung para formalização de conflitos e de regras MODD para delimitação de autonomia decisória. O sistema foi implementado utilizando três agentes BDI no framework MASPY: o Agente VA responsável pela condução autônoma, o Agente Ético atuando como Governor Module para monitoramento e cálculo de utilidades éticas baseadas no German Ethics Code, e o Agente HITL representando supervisão humana. A comunicação entre agentes utiliza adaptação do protocolo ContractNet. Os experimentos validaram o sistema através de cenários de cruzamentos urbanos, confirmando que o sistema identifica corretamente situações resolvíveis de forma autônoma e aquelas que exigem transferência para supervisão humana, respeitando as diretrizes éticas estabelecidas.

Palavras-chave: sistemas multiagentes; veículos autônomos; agentes éticos; conflitos éticos; cruzamento de tráfego urbano.

ABSTRACT

This work presents the development and validation of a multi-agent system (MAS) capable of performing ethical reasoning in decisions involving autonomous vehicles at urban intersections. The investigated problem arises from the need to incorporate moral principles into autonomous systems that interact with human lives, ensuring compliance with ethical guidelines and social acceptance. The general objective consists of modeling and implementing a MAS following the ethical agent architecture proposed by Alves, extending it through the application of Galtung's Triangle for conflict formalization and MODD rules for decision-making autonomy delimitation. The system was implemented using three BDI agents in the MASPY framework: the AV Agent responsible for autonomous driving, the Ethical Agent acting as Governor Module for monitoring and ethical utility calculation based on the German Ethics Code, and the HITL Agent representing human supervision. Communication between agents uses an adaptation of the Contract Net protocol. Experiments validated the system through urban intersection scenarios, confirming that the system correctly identifies situations resolvable autonomously and those requiring transfer to human supervision, respecting established ethical guidelines.

Keywords: multi-agent systems; autonomous vehicles; ethical agents; ethical conflicts; urban traffic intersection.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGOS FONTE

SUMÁRIO

1 INTRODUÇÃO

Sistemas autônomos estão cada vez mais presentes no cotidiano, auxiliando em diversas tarefas humanas, desde simples limpezas domésticas realizadas por robôs aspiradores até procedimentos cirúrgicos complexos conduzidos por robôs cirurgiões (ALGOtive, 2022). No entanto, um dos avanços mais debatidos, onde há pouco tempo atrás não passavam de ficção, são os sistemas autônomos em veículos, também conhecidos por Veículos Autônomos (VAs), no qual um agente condutor realiza funções do piloto humano sem o seu auxílio (Fernandes *et al.*, 2017), como: mapeamento do ambiente, detecção de obstáculos, planejamento de rotas e tomada de decisões em tempo real com base nas condições da estrada e do tráfego urbano.

O primeiro modelo de veículo realmente autônomo foi o projeto NavLab 1 (Dowling; Mahalingam; Whittaker, 1990), lançado em 1986, sendo um modelo simples que combinava lasers, radares e câmeras para navegar sem intervenção humana, porém, era pouco eficiente e lidava com diversos problemas. Desde então, com o avanço tecnológico, os VAs evoluíram significativamente, contudo, à medida que esses sistemas se tornam mais autônomos e envolvem vidas humanas, questões éticas emergem.

Conforme argumentado por Picard (1995), quanto maior a liberdade de uma máquina, mais ela precisará de padrões morais. Isso se torna relevante quando se considera a interação cada vez mais estreita entre humanos e máquinas, o que pode levar a um controle humano menos significativo e a um maior número de decisões tomadas por máquinas. Portanto, a ética precisa ser um fator na tomada de decisão para considerar problemas fundamentais em veículos autônomos (Sparrow, 2007).

Para fazer com que uma máquina aja de maneira ética de forma autônoma — denominada Máquina Ética —, o sistema necessita de uma teoria moral para orientar suas decisões. Em Tolmeijer *et al.* (2020) são descritas as teorias éticas mais comumente usadas em máquinas éticas, sendo elas: Consequencialismo, Ética Deontológica e Ética das Virtudes.

De acordo com Bench-Capon (2020), o Consequencialismo é uma teoria moral que defende que uma ação é moralmente boa quando suas consequências são boas, independentemente das intenções. O Utilitarismo é uma forma específica de Consequencialismo que busca maximizar o bem-estar total ou minimizar o dano total resultante das ações — ou seja, produzir "o maior bem para o maior número de pessoas". A Ética Deontológica, por outro lado, independe das consequências das ações, levando em consideração apenas o julgamento de um conjunto de regras. Por fim, a Ética das Virtudes é uma abordagem mais simples e antiga, no qual ações moralmente boas exemplificam virtudes e ações ruins exemplificam vícios, diferenciando-as. Este trabalho adota a abordagem Utilitarista, que permite quantificar as consequências morais de cada ação através de valores numéricos — denominados utilidades —, tornando possível comparar alternativas e selecionar aquela com melhores resultados morais esperados.

As Máquinas Éticas podem auxiliar VAs utilizando decisões éticas para resolver conflitos em cenários de tráfego (Reed *et al.*, 2021). Por exemplo, em situações onde o VA necessitar realiza uma manobra evasiva que desrespeite uma lei de trânsito — configurando em uma ação

antiética, é necessário que esses conflitos sejam resolvidos de maneira moralmente aceitável para garantir a aceitação social dos VAs (Rakow; Schwammberger, 2023). Para resolver esses conflitos, a máquina ética necessita dos contextos específicos de seu domínio de aplicação. Os Domínios de Design Operacional Moral (*Moral Operational Design Domains* — MODDs) desempenham essa função, guiando suas decisões ao especificar os contextos que permitirão resolver os conflitos e quais valores ou objetivos devem ser priorizados em diferentes cenários (Rakow; Schwammberger, 2023).

Por conseguinte, a proposta deste trabalho é desenvolver uma arquitetura de agente com a função de uma máquina ética, denominado de agente ético, conforme apresentada inicialmente em Alves, Dennis e Fisher (2021). Logo, com base nessa arquitetura, será implementado um Sistema Multiagente (SMA) em um cenário de trânsito urbano, utilizando agentes inteligentes, por meio do framework de programação MASPY (Mellado; Borges; Alves, 2025), que é especializado na programação de agentes BDI na linguagem Python.

O SMA é composto por três agentes com responsabilidades distintas. O agente VA realiza o controle autônomo do veículo, sendo responsável pela condução e execução de manobras. Já o agente Ético monitora o ambiente, detecta situações que exigem avaliação moral e recomenda ações eticamente justificáveis ao VA, atuando como uma camada de raciocínio ético independente do sistema de condução. Por fim, o agente HITL (*Human-in-the-Loop*) representa a supervisão humana, intervindo quando o sistema autônomo enfrenta dilemas éticos genuínos — situações onde nenhuma ação é moralmente superior às demais. A comunicação entre os agentes ocorre através de protocolos de negociação, permitindo que o VA aceite, rejeite ou solicite alternativas às recomendações éticas recebidas caso for necessário.

1.1 Objetivos

Este trabalho tem como objetivo modelar e implementar um sistema multiagente capaz de realizar raciocínio ético em decisões envolvendo veículos autônomos em cruzamentos urbanos, seguindo a arquitetura de agente ético proposta por Alves, Dennis e Fisher (2021) e estendendo-a através da aplicação de formalização de conflitos e de regras para delimitação de autonomia do sistema.

Para alcançar o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- Formalizar conflitos éticos através do Triângulo de Galtung, estruturando cada situação de conflito segundo os vértices de Atitude (percepções do ambiente), Contradição (metas incompatíveis) e Comportamento (ações opostas);
- Implementar MODDs que definem os limites da autonomia do sistema: para cada tipo de conflito, estabelecer quando o agente pode decidir sozinho e quando precisa transferir a decisão para supervisão humana;

- Adaptar o protocolo ContractNet para negociação entre agentes, incorporando a verificação MODD como etapa intermediária entre a detecção de conflito e o envio de recomendações éticas;
- Desenvolver funções de utilidade ética baseadas no *German Ethics Code*, permitindo quantificar as consequências morais de cada ação possível e identificar formalmente dilemas éticos através de critérios matemáticos;
- Validar o sistema através de cenários de teste representativos de cruzamentos urbanos, incluindo conflitos com pedestres e dilemas éticos envolvendo múltiplos agentes.

1.2 Justificativa

A motivação principal para o desenvolvimento deste trabalho é a consolidação dos veículos autônomos e dos sistemas autônomos éticos, fornecendo a implementação de uma arquitetura que integra conceitos de máquinas éticas, conflitos formais e comunicação entre agentes. Tal abordagem visa não somente expandir a arquitetura proposta por Alves, Dennis e Fisher (2021), mas também atender à necessidade de experimentos que comprovem a capacidade dos VAs em resolver conflitos éticos, condição essencial para que esses sistemas atuem de forma socialmente aceitável, promovendo a confiança e a aceitação da sociedade.

Também, a utilização do framework MASPY — ferramenta em desenvolvimento na UTFPR Ponta Grossa — constitui uma justificativa importante para este trabalho, pela possibilidade de testar suas funcionalidades.

1.3 Organização do Trabalho

Este trabalho está organizado em seis capítulos. No Capítulo 2 são apresentados os conceitos fundamentais que embasam este estudo, incluindo uma visão geral sobre Sistemas Multiagentes e Agentes BDI, os fundamentos de Veículos Autônomos e Máquinas Éticas, a modelagem de conflitos utilizando o Triângulo de Galtung e os MODDs, o protocolo ContractNet para negociação entre agentes, e o framework MASPY utilizado na implementação.

O Capítulo 3 apresenta a modelagem completa do sistema multiagente proposto. Nele, é detalhada a visão geral da arquitetura do sistema, a descrição PEAS dos agentes, o diagrama UML MASPY que ilustra a estrutura dos agentes e suas interações, a modelagem dos conflitos éticos baseada no Triângulo de Galtung, e o protocolo de negociação utilizando ContractNet, incluindo o diagrama de sequência que representa a comunicação entre os agentes.

No Capítulo 4 são descritos os aspectos de implementação do sistema. Primeiramente, é apresentado o Ambiente Urbano que simula o cruzamento de tráfego. Em seguida, são detalhados cada um dos agentes: o Agente Veículo Autônomo (VA), responsável pela condução e ações no ambiente; o Agente Ético, que monitora o ambiente e fornece recomendações baseadas em princípios éticos; e o Agente HITL, que atua como observador e interventor em

situações de dilemas éticos. Por fim, são explicados os mecanismos de comunicação entre os agentes utilizando o framework MASPY.

O Capítulo 5 apresenta os experimentos realizados e a validação do sistema proposto. Nele são descritos os cenários de teste utilizados para avaliar o comportamento do sistema. Em seguida, são apresentados os resultados obtidos nas simulações e, por fim, é realizada uma discussão sobre os resultados, analisando o comportamento dos agentes e a eficácia da arquitetura proposta.

Finalmente, no Capítulo 6 são apresentadas as considerações finais sobre o trabalho desenvolvido, destacando as principais contribuições e limitações encontradas. Também são propostos trabalhos futuros que podem estender e aprimorar a arquitetura implementada.

Durante a preparação deste trabalho, o autor utilizou Claude Sonnet 4.5 (Anthropic) para auxiliar na revisão textual e organização estrutural do documento. Após o uso dessa ferramenta, o autor revisou e editou todo o conteúdo em conformidade com o método científico e assume total responsabilidade pelo conteúdo da publicação. O desenvolvimento conceitual, a implementação do sistema e a análise dos resultados foram realizados integralmente pelo autor.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais necessários para a compreensão do sistema desenvolvido neste trabalho. São abordados os Sistemas Multiagentes e a arquitetura BDI, os conceitos de Veículos Autônomos e Máquinas Éticas, a modelagem de conflitos baseada no Triângulo de Galtung e nos MODDs, o protocolo Contract Net para negociação entre agentes, e o framework MASPY utilizado na implementação.

2.1 Sistemas Multiagentes e Agentes BDI

Sistemas Multiagentes são sistemas compostos por múltiplos agentes autônomos que interagem para alcançar objetivos. Um agente é uma entidade que percebe seu ambiente através de sensores e age através de atuadores, possuindo autonomia para tomar decisões (Wooldridge, 2009). No contexto deste trabalho, cada componente do sistema (VA, Agente Ético, HITL) é implementado como um agente autônomo.

A arquitetura BDI, proposta por Rao e Georgeff (1995), modela agentes através de três componentes: **crenças** (conhecimento sobre o mundo), **desejos** (objetivos a alcançar) e **intenções** (planos de ação). Por exemplo, o Agente VA possui crenças como *"há um pedestre na faixa"*, objetivos como *"atravessar o cruzamento com segurança"*, e planos que definem as ações a executar para alcançar esses objetivos. No qual, ao longo da execução, o agente atualiza suas crenças conforme percebe o ambiente, revisa seus objetivos e seleciona novos plano. Logo, essa arquitetura foi utilizada por permitir raciocínio explícito e verificável, característica essencial para sistemas éticos (Bremner *et al.*, 2019).

2.2 Framework MASPY

O MASPY (*Multi-Agent System for Python*) é um framework para desenvolvimento de sistemas multiagentes baseados na arquitetura BDI, implementado em Python (Mellado; Borges; Alves, 2025). O framework foi desenvolvido com o objetivo de facilitar a implementação de agentes inteligentes utilizando uma linguagem de propósito geral, tornando mais acessível o desenvolvimento de sistemas que tradicionalmente requerem linguagens especializadas como Jason ou Gwendolen.

Uma das principais características do MASPY é a utilização de decoradores Python para definição dos planos BDI. Cada plano é implementado como um método de classe decorado com `@pl`, seguindo a estrutura tradicional BDI: evento gatilho (*trigger*), contexto (*context*) e corpo do plano (*body*). O evento gatilho especifica quando o plano deve ser ativado, como ao adquirir uma nova crença (`gain`) ou perder um objetivo (`drop`). O contexto define condições adicionais que devem ser satisfeitas para que o plano seja executável, usando a classe `Belief` para testar crenças do agente. O corpo do plano contém as ações a serem executadas, podendo

incluir ações internas como adicionar ou remover crenças (`add`, `rm`), enviar mensagens para outros agentes, ou interagir com o ambiente.

A comunicação entre agentes no MASPY é realizada através de canais que seguem o padrão de performativas KQML (*Knowledge Query and Manipulation Language*). As principais performativas disponíveis são `tell` (informar uma crença), `achieve` (solicitar que outro agente alcance um objetivo), e `ask` (consultar crenças ou objetivos). Essa estrutura permite que agentes coordenem suas ações através de troca explícita de mensagens.

O framework fornece uma classe `Environment` para modelagem do ambiente onde os agentes atuam. O ambiente é responsável por criar percepções (`Percept`) que representam o estado observável do mundo. Agentes conectados ao ambiente podem perceber automaticamente essas percepções, que são convertidas em crenças locais, permitindo que tomem decisões baseadas no estado atual do ambiente. Essa separação clara entre agente e ambiente facilita a modelagem de sistemas onde múltiplos agentes interagem com um mundo compartilhado.

A classe `Admin` gerencia o ciclo de vida do sistema multiagente, sendo responsável por conectar agentes a canais de comunicação e ambientes, iniciar o ciclo de raciocínio dos agentes, e controlar a execução do sistema. O Listagem 1 ilustra um exemplo simples de implementação utilizando o framework MASPY, demonstrando a estrutura básica de um agente BDI, a comunicação entre agentes e a interação com o ambiente.

Listagem 1 – Exemplo básico de agente BDI em MASPY

```
from maspy import *

# Definicao do Ambiente
class AmbienteSimples(Environment):
    def __init__(self, env_name):
        super().__init__(env_name)
        self.create(Percept("temperatura", 25))

# Definicao do Agente
class AgenteSimples(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
        # Crenca inicial
        self.add(Belief("ativo", True))
        # Objetivo inicial
        self.add(Goal("monitorar"))

    # Plano: gatilho (gain Goal), contexto (Belief), corpo (acoes)
    @pl(gain, Goal("monitorar"), Belief("ativo", True))
    def executar_monitoramento(self, src, status):
        # Acao interna: adicionar crenca
        temp = self.get(Belief("temperatura", Any, "AmbienteSimples"))
        self.add(Belief("temp_lida", temp.values if temp else 0))

        # Comunicacao KQML: enviar mensagem
        self.send("AgenteReceptor", tell,
                  Belief("temperatura_atual", temp.values))
        self.stop_cycle()

# Configuracao do sistema multiagente
ambiente = AmbienteSimples("AmbienteSimples")
agente = AgenteSimples("AgenteSimples")
canal = Channel("CanalComunicacao")

Admin().connect_to([agente], canal)
Admin().connect_to([agente], ambiente)
Admin().start_system()
```

Fonte: Autoria própria (2026).

O framework também oferece capacidades de geração de relatórios detalhados sobre a execução dos agentes, incluindo informações sobre crenças, objetivos, planos executados e mensagens trocadas, facilitando a depuração e análise do comportamento do sistema. Essa combinação de características torna o MASPY adequado para implementação de sistemas multiagentes complexos que requerem raciocínio explícito, comunicação estruturada e interação com ambientes dinâmicos, características essenciais para o desenvolvimento de sistemas éticos para veículos autônomos.

2.3 Veículos Autônomos e Máquinas Éticas

Veículos Autônomos tomam decisões de direção em frações de segundo e em muitas situações, essas decisões envolvem escolhas éticas. Por exemplo, em um cruzamento urbano, o VA pode precisar escolher entre frear para proteger um pedestre — arriscando uma colisão traseira — ou avançar para proteger seus passageiros. Essas situações exigem que o sistema possua alguma forma de raciocínio ético.

Para orientar decisões éticas em VAs, existem diferentes códigos normativos. Este trabalho utiliza como referência o *German Ethics Code* (Ethics Commission, 2017), conjunto de diretrizes éticas estabelecido pela Comissão de Ética do Governo Alemão. As principais diretrizes utilizadas estão apresentadas no Quadro 1.

Quadro 1 – Diretrizes do *German Ethics Code* relevantes para veículos autônomos

Diretriz	**Descrição**
EG 1	A proteção da vida humana tem prioridade máxima
EG 2	A proteção de indivíduos deve prevalecer sobre danos materiais
EG 5	O sistema deve respeitar as regras de trânsito sempre que possível
EG 7	Em situações de perigo inevitável, a proteção de indivíduos não pode ser baseada em características pessoais
EG 8	Em situações dilemáticas genuínas, o sistema não deve tomar decisões programadas, sendo necessária intervenção humana
EG 19	Em situações de emergência, o veículo deve transferir o controle para o condutor humano (*handover*)

Fonte: Adaptado de Ethics Commission (2017).

Para implementar essas diretrizes em um sistema computacional, é necessário uma abordagem ética que permita comparar ações de forma quantitativa. Como apresentado na introdução, este trabalho adota a abordagem utilitarista (Bench-capon, 2020): cada ação possível recebe um valor numérico — denominado **utilidade** — que representa suas consequências morais previstas naquele contexto. O Agente Ético avalia todas as alternativas disponíveis(frear, desviar e seguir) e recomenda aquela com maior utilidade, pois essa é a escolha moralmente mais justificável dado o cenário. Essa representação numérica é o que torna o raciocínio ético viável em tempo real: em vez de "raciocínios"abstratos, o sistema realiza comparações aritméticas entre valores morais formalizados. As diretrizes do *German Ethics Code* funcionam como base de referência para a atribuição desses valores — ações que protegem a vida humana (EG 1, EG 2) recebem utilidades mais altas e ações que violam regras de trânsito sem justificativa moral recebem utilidades menores ou negativas (contrariando EG 5). Os valores específicos adotados neste trabalho são definidos na seção 3.4.

Quando duas ações produzem utilidades equivalentes e nenhuma é moralmente superior à outra, o sistema se depara com um **dilema ético**. Nesses casos, a diretriz EG 8 (Quadro 1)

determina que a decisão deve ser transferida para supervisão humana. Os mecanismos de detecção e transferência são aprofundados na seção 3.3.

Para centralizar o raciocínio ético e separá-lo do sistema de controle do veículo, Alves, Dennis e Fisher (2021) propõe o conceito de *Governor Module* — um componente dedicado exclusivamente ao raciocínio ético, operando de forma independente do agente de condução. O VA é responsável pela navegação e execução de manobras, sem qualquer lógica moral interna. O *Governor Module* monitora o ambiente, detecta situações que exigem avaliação ética, calcula utilidades e emite recomendações formais. Essa separação tem uma consequência prática importante: o comportamento ético do sistema pode ser analisado e modificado sem interferir na lógica de condução, o que facilita a verificação de conformidade com normas morais (Bremner *et al.*, 2019). Na implementação deste trabalho, o Agente Ético exerce o papel de *Governor Module*.

2.4 Modelagem de Conflitos Éticos

Este trabalho utiliza duas técnicas complementares para modelagem de conflitos éticos em veículos autônomos: Triângulo de Galtung e Domínio de Design Operacional. O Triângulo de Galtung (Galtung, 1969) fornece a estrutura conceitual para representar conflitos através de três componentes relacionados — Atitude, Comportamento e Contradição —, permitindo que o sistema formalize situações onde objetivos incompatíveis entram em choque. Já os Domínios de Design Operacional Moral (MODDs), propostos por Rakow *et al.* (2024), delimitam as fronteiras dentro das quais o sistema pode exercer raciocínio ético de forma autônoma, estabelecendo quais conflitos ele consegue resolver sozinho e quais precisam ser transferidos para supervisão humana. Juntas, essas técnicas permitem que o Agente Ético identifique a estrutura de cada conflito, avalie se possui competência moral para resolvê-lo e, quando não possui, transfira a responsabilidade ao HITL — que representa a decisão humana neste trabalho.

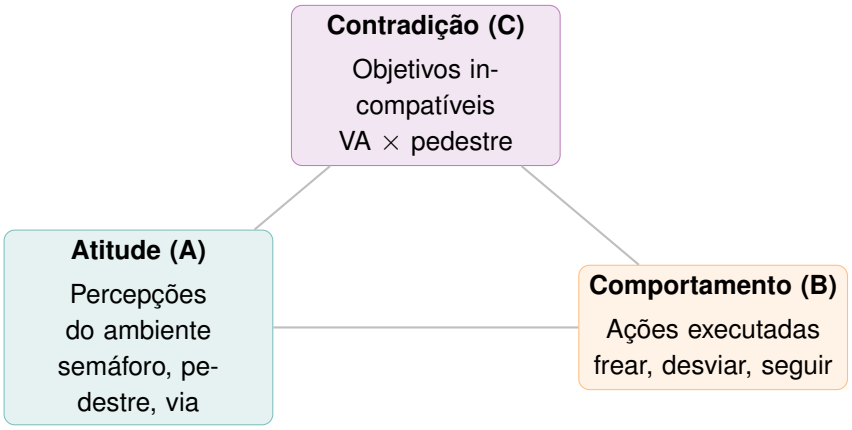
2.4.1 Triângulo de Galtung

A tomada de decisão ética em veículos autônomos exige que o sistema identifique quando uma situação requer raciocínio moral. Nem toda percepção do ambiente constitui um conflito ético: o VA pode atravessar um cruzamento vazio respeitando apenas as regras de trânsito convencionais, sem necessidade de avaliação moral. Já o conflito surge quando há incompatibilidade entre objetivos igualmente válidos — como respeitar uma regra de trânsito versus proteger uma vida humana —, situações onde o sistema precisa decidir qual valor deve prevalecer. Por isso, uma modelagem formal do conceito de conflito é pré-requisito para qualquer sistema de ética computacional aplicado a VAs.

O Triângulo de Galtung, proposto por Johan Galtung em 1969, oferece estrutura para modelar conflitos a partir de três vértices relacionados, denominados modelo ABC: **Atitude** (*Attitude*), **Comportamento** (*Behavior*) e **Contradição** (*Contradiction*) (Galtung, 1969). A **Con-**

tradição é o núcleo do conflito: objetivos ou necessidades de dois ou mais agentes que não podem ser satisfeitos ao mesmo tempo. A **Atitude** captura as percepções que cada parte tem da situação — como os agentes compreendem o que está acontecendo, incluindo possíveis percepções parciais ou incorretas do ambiente. O **Comportamento** são as ações concretas tomadas por cada parte diante da situação. Um conflito existe quando há Contradição entre os agentes. Atitude e Comportamento pioram ou reduzem esse conflito, mas é a incompatibilidade de objetivos que o define, conforme ilustrado na Figura 1.

Figura 1 – Triângulo de Galtung: modelo ABC aplicado ao contexto de veículos autônomos



Fonte: Autoria própria.

Aplicando essa estrutura ao contexto de veículos autônomos, conforme proposto por Rakow *et al.* (2024), os três vértices mapeiam diretamente para os elementos do cenário de trânsito. A **Contradição** ocorre quando o VA tem o objetivo de cruzar o semáforo — válido segundo as regras de trânsito — enquanto um pedestre tem o objetivo de atravessar a faixa — válido segundo seu direito de passagem. Os dois objetivos não podem ser satisfeitos simultaneamente. A **Atitude** corresponde às percepções do Agente Ético sobre o ambiente: semáforo aberto, pedestre detectado, veículo atrás. São essas percepções que orientam o cálculo das utilidades éticas. O **Comportamento** é a ação efetivamente executada pelo VA: frear, desviar ou seguir. Modelar um conflito no sistema equivale, portanto, a identificar os três vértices do triângulo naquele cenário — e resolver o conflito consiste em escolher o Comportamento que melhor trata a Contradição com base nas Atitudes (percepções) disponíveis.

Nem todo conflito detectado pelo sistema constitui um dilema ético. A maioria das situações possui uma ação claramente superior às demais: proteger um pedestre tem utilidade maior do que respeitar um sinal verde. Nesses casos, o conflito é **resolvível** — o Agente Ético calcula as utilidades, identifica a ação dominante e a recomenda ao VA. Um **dilema ético** surge quando dois ou mais comportamentos produzem utilidades equivalentes, sem que haja uma opção moralmente superior. Por exemplo: frear protege o pedestre mas expõe o passageiro traseiro a uma colisão, e seguir faz o inverso. Ambas as ações causam dano moral equivalente a partes distintas, e nenhuma é eticamente preferível à outra. Conforme a diretriz EG 8 (Qua-

dro 1), a responsabilidade nessas situações deve ser transferida para supervisão humana — representada pelo HITL neste trabalho.

2.4.2 Domínios de Design Operacional Moral (MODDs)

Para lidar formalmente com a distinção entre conflitos resolvíveis e dilemas éticos, Rakow *et al.* (2024) argumenta que três capacidades são necessárias em um sistema autônomo ético. A primeira é a **consciência do conflito** (*aware*): o sistema deve ser capaz de identificar, a partir das percepções do ambiente, que uma situação de conflito está ocorrendo — ou seja, reconhecer a Contradição no triângulo de Galtung apresentado anteriormente. A segunda é a **resolução do conflito** (*resolve*): quando o conflito for resolvível, o sistema deve escolher uma ação moralmente justificada com base nas utilidades calculadas. A terceira é a **transferência** (*transfer*): quando o conflito não puder ser resolvido autonomamente — por ser um dilema ético ou por ultrapassar os limites do domínio de operação — deve haver um mecanismo para transferir a responsabilidade de decisão a uma autoridade externa. No sistema implementado neste trabalho, as capacidades *aware* e *resolve* são exercidas pelo Agente Ético, e a capacidade *transfer* acontece através do *handover*, mecanismo que entrega a decisão ao Agente HITL quando necessário.

A aplicação do raciocínio ético não ocorre de forma irrestrita: ela é delimitada pelos MODDs, apresentados brevemente na introdução deste trabalho. Os MODDs especificam os contextos operacionais em que um sistema autônomo pode exercer raciocínio ético de forma autônoma. Eles definem quais tipos de conflito o sistema é capaz de reconhecer e resolver por conta própria, quais valores devem ser priorizados em cada cenário e quais situações ultrapassam o escopo de decisão autônoma e exigem intervenção humana. Conflitos dentro do MODD podem ser resolvidos pelo próprio sistema. Os que o excedem — incluindo dilemas éticos — precisam ser transferidos para uma autoridade externa. Essa fronteira é o que garante que decisões críticas permaneçam sob controle humano, respeitando o princípio de que sistemas autônomos não devem operar além de sua competência moral. O MODD específico deste trabalho — cruzamentos urbanos com semáforo — e seus limites formais são definidos na seção 3.3.

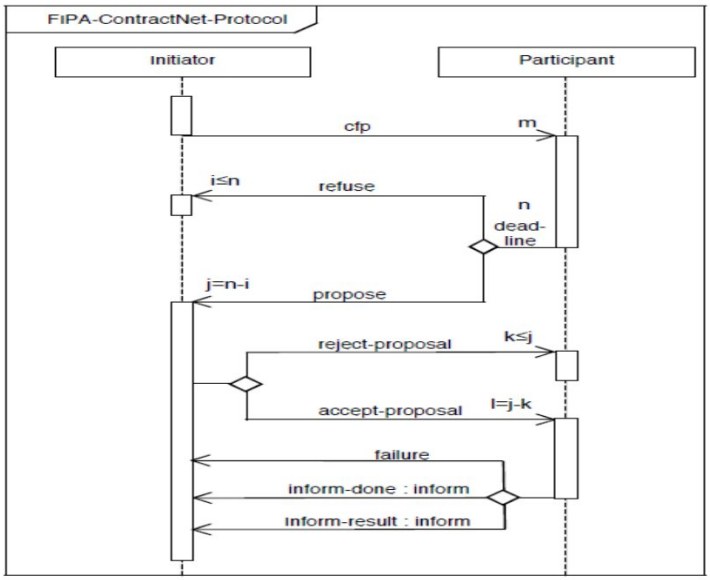
2.5 Protocolo Contract Net

Em SMAs, os agentes frequentemente precisam coordenar suas ações para alcançar objetivos comuns. O protocolo Contract Net, proposto por Smith (1980), oferece uma estrutura para coordenação baseada em negociação de tarefas entre agentes.

No protocolo básico, um agente assume o papel de gerente e outro de participante. O gerente envia uma chamada de propostas descrevendo a tarefa a ser realizada. O participante avalia se é capaz de executar a tarefa e responde com uma proposta contendo suas condições, ou recusa caso não possa executá-la. O gerente então avalia as propostas recebidas e decide

aceitar ou rejeitar cada uma segundo seus próprios critérios de decisão. A Figura 2 ilustra a estrutura básica desse protocolo de negociação, mostrando as trocas de mensagens entre o agente iniciador (gerente) e os participantes ao longo do processo de alocação de tarefas.

Figura 2 – Estrutura do protocolo Contract Net



Fonte: FIPA (2002).

Conforme ilustrado na Figura 2, o protocolo define um conjunto de mensagens e variáveis que estruturam o fluxo de negociação entre os agentes. O gerente envia m chamadas de proposta (*cfp*) aos participantes disponíveis. Desses, $i \leq n$ recusam (*refuse*) antes do prazo estipulado (*deadline*), enquanto os $j = n - i$ restantes submetem propostas (*propose*). O gerente avalia as propostas recebidas e rejeita $k \leq j$ delas (*reject-proposal*), aceitando as $l = j - k$ restantes (*accept-proposal*). Por fim, os participantes com propostas aceitas executam a tarefa e notificam o gerente com a confirmação de conclusão (*inform-done*) e o resultado obtido (*inform-result*), ou comunicam uma falha (*failure*) caso não consigam concluir a execução.

Esse modelo de negociação permite que sistemas multiagentes distribuam tarefas de forma descentralizada, onde cada agente pode avaliar propostas segundo seus próprios objetivos e restrições, coordenando suas ações através de um processo estruturado de comunicação.

3 MODELAGEM DO SISTEMA MULTIAGENTE

Este capítulo apresenta a modelagem do sistema desenvolvido. A modelagem cobre a arquitetura geral e a especificação PEAS dos três agentes, a formalização dos cinco tipos de conflito ético no domínio de cruzamentos urbanos, as funções de utilidade que quantificam as consequências morais de cada ação, o protocolo de negociação adaptado, e o diagrama UML MASPY que representa a estrutura BDI dos agentes.

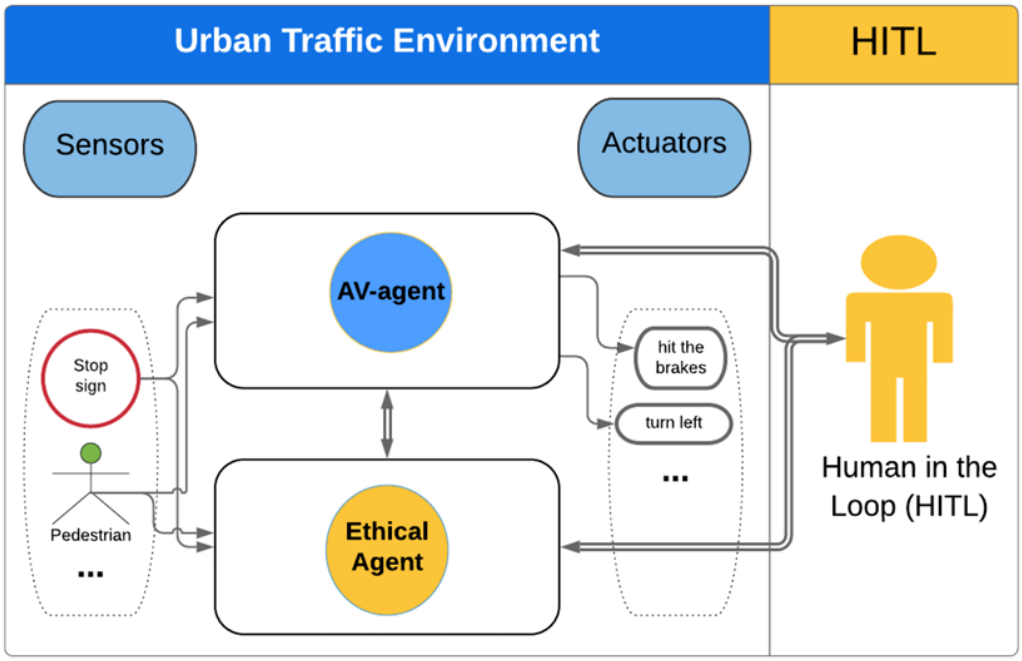
3.1 Arquitetura proposta base

O sistema implementa a arquitetura base de Alves, Dennis e Fisher (2021), onde o Agente Ético desempenha o papel de *Governor Module* conforme apresentado na seção 3.3.

3.1.1 Componentes da Arquitetura

O sistema implementa a arquitetura base proposta por Alves, Dennis e Fisher (2021), onde três agentes BDI interagem através de protocolos estruturados de comunicação em um ambiente de cruzamento urbano. A Figura 3 apresenta a estrutura geral do sistema.

Figura 3 – Arquitetura geral do sistema multiagente com Governor Module



Fonte: Alves, Dennis e Fisher (2021).

Conforme ilustrado na Figura 3, o Ambiente Urbano provê percepções do cruzamento ao Agente VA, sendo este responsável pela condução autônoma e execução de manobras. Já o Agente Ético atua como camada de raciocínio moral independente, avaliando as percepções

e recomendando ações ao VA. Quando há dilema ético, o Agente HITL intervém na decisão. A troca de informações entre os três agentes ocorre por meio de canais de comunicação.

A fim de modelar o sistema, foram definidos os seguintes papéis para os componentes da arquitetura: o **Agente Ético** monitora o ambiente de forma independente, detecta situações de conflito moral — por exemplo, pedestre atravessando com semáforo verde, veículo bloqueando a via, obstáculo exigindo manobra evasiva —, calcula utilidades éticas para cada ação possível usando princípios do *German Ethics Code* e envia recomendações ao VA. Quando detecta um dilema ético, escala a decisão para o Agente HITL.

O **Agente VA** conduz o veículo, percebe o cruzamento através de sensores e executa ações físicas (frear, desviar, seguir). O VA não detecta conflitos éticos — isso é responsabilidade do Agente Ético. Ao receber recomendações, o VA avalia se elas são coerentes com suas próprias percepções e decide se aceita, solicita alternativas ou toma decisão autônoma.

O **Agente HITL** atua como observador e interventor. Recebe logs de todas as decisões do sistema e, em situações de dilema ético, toma decisões que o sistema autônomo não pode resolver sozinho. Representa a supervisão humana exigida pela diretriz EG 8 (Quadro 1).

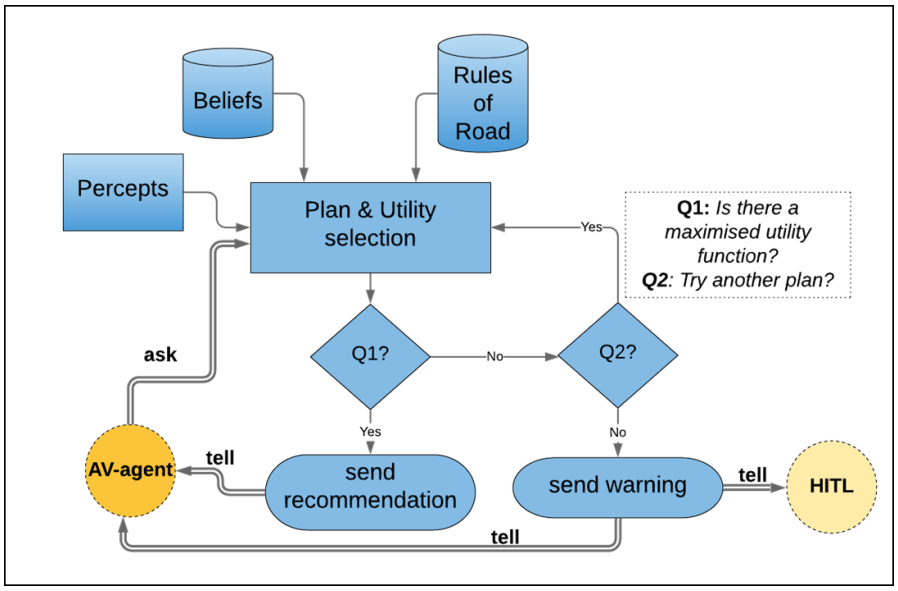
Por fim, o **Ambiente Urbano** representa o cruzamento de trânsito onde o VA opera. Provê sete percepções do estado atual — semáforo, presença de pedestres, veículos (frontais, traseiros e laterais), obstáculos e indicação de zona escolar — que alimentam tanto o VA quanto o Agente Ético. Atua como fonte única de verdade sobre o estado do mundo, garantindo que ambos os agentes percebam o mesmo cenário de forma consistente.

3.1.2 Fluxo de Decisão Ética

O processo de tomada de decisão ética segue um fluxo estruturado em três questões principais (Q1, Q2 e Q3), conforme proposto na arquitetura original e ilustrado na Figura 4 e Figura 5. A primeira questão (Q1) é: **existe utilidade maximizada?** O Agente Ético calcula a utilidade de cada ação possível (frear, desviar, seguir) com base nos princípios do *German Ethics Code*. Se uma ação possui utilidade claramente superior às demais, o Ético recomenda essa ação ao VA (Q1 = Sim). Caso a diferença das duas melhores utilidades configure dilema ético, o sistema reconhece a situação (Q1 = Não) e passa para Q2.

Quando Q1 resulta em Sim — ou seja, quando não há dilema ético —, o sistema ainda não envia a recomendação diretamente ao VA. Antes disso, o Agente Ético consulta o MODD para verificar se possui autoridade para resolver aquele tipo específico de conflito. O MODD funciona como uma tabela de regras que define, para cada categoria de conflito reconhecida pelo sistema, qual a margem de segurança mínima necessária antes de autorizar decisão autônoma. Conflitos envolvendo pedestres em zona escolar, por exemplo, exigem margem maior que outros tipos, refletindo o risco moral elevado daquele contexto. Se a diferença de utilidade entre a melhor e a segunda melhor ação atinge ou supera o limiar definido para aquele tipo de conflito, o MODD autoriza resolução autônoma e o Ético prossegue para a Fase 2, enviando a recomendação para o VA.

Figura 4 – Fluxo de decisão do Agente Ético (Q1 e Q2)

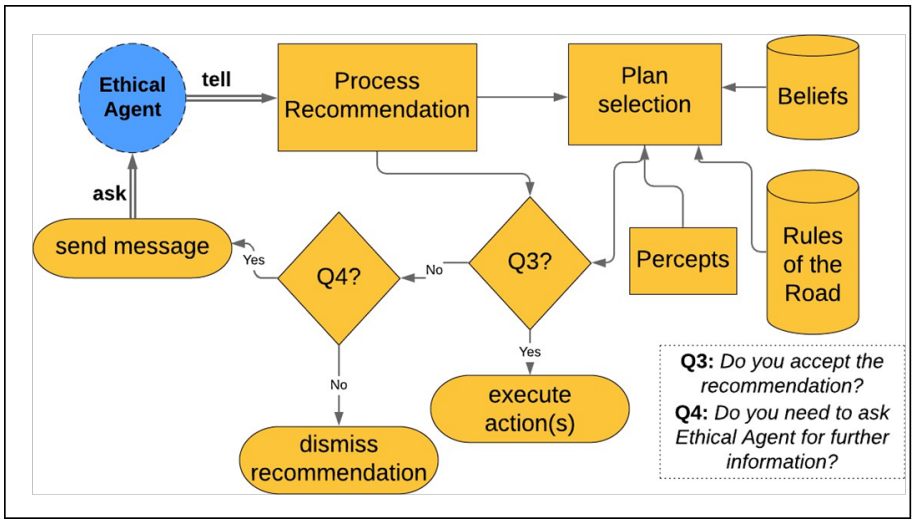


Fonte: Alves, Dennis e Fisher (2021).

Se Q1 resultou em dilema, a segunda questão (Q2) é: **existe plano alternativo?** O Agente Ético verifica se há algum plano de ação alternativo que resolva o conflito. Na implementação deste trabalho, quando Q2 também resulta em negativa, o sistema escala o dilema para o Agente HITL, transferindo a decisão para supervisão humana conforme estabelecido pela diretriz EG 8 (Quadro 1).

A terceira questão (Q3) ocorre no lado do VA: **a recomendação é coerente?** O VA, ao receber uma recomendação do Agente Ético, avalia se ela é coerente com suas próprias percepções e regras de trânsito. Por exemplo, uma recomendação para “seguir“ não é coerente se o VA percebe um pedestre na faixa. Se Q3 = Sim, o VA executa a ação recomendada. Se Q3 = Não, o VA decide entre consultar o Ético por mais detalhes (Q4 = Sim, quando há situação perigosa) ou tomar decisão autônoma (Q4 = Não, quando não há perigo imediato).

Figura 5 – Fluxo de decisão do Agente VA (Q3 e Q4)



Fonte: Alves, Dennis e Fisher (2021).

Esse fluxo de decisão distribuída garante que o sistema ético não imponha ações ao VA de forma arbitrária. O VA mantém autoridade para questionar recomendações quando elas conflitam com suas percepções locais, criando um ciclo de revisão e negociação entre os agentes.

3.1.3 Pacotes de Comunicação Ética (ECP)

A comunicação entre os agentes ocorre através de um canal compartilhado (*AgentComm*), implementando o conceito de *Ethical Communication Packet* (ECP), que são tuplas formais que carregam informações sobre remetente, destinatário, posição no ambiente, ação recomendada e utilidade calculada, conforme apresentada na Equação 1:

$$\text{ECP} = (\text{Sender}, \text{Receiver}, \text{Env\_pos}, \text{Act}, \text{Utl}) \tag{1}$$

onde **Sender** é o agente que envia o pacote (normalmente o Agente Ético), **Receiver** é o agente destinatário (VA ou HITL), **Env_pos** é a posição no ambiente onde a ação deve ser executada, **Act** é a ação recomendada (frear, desviar, seguir, ou *handover* para HITL) e **Utl** é a utilidade ética calculada para a ação recomendada.

Por exemplo, quando o Agente Ético detecta um pedestre no cruzamento e calcula que frear possui utilidade 10 (proteção de vida humana, diretrizes EG 1 e EG 2 do Quadro 1), enquanto seguir possui utilidade -1 (violação de regra moral), ele envia o seguinte ECP ao VA:

```
ECP(ET, AV, Cruzamento, {frear}, {10})
```

O VA recebe esse pacote, avalia a coerência da recomendação com suas próprias percepções (há de fato um pedestre? o semáforo está aberto?) e, se coerente, executa a ação de frear.

Em situações de dilema ético, o ECP enviado ao HITL possui formato especial, indicando transferência de controle (*handover*):

$$\texttt{ECP(ET, HITL, Cruzamento, \{handover, retake control\}, } u_i\texttt{, } u_j\texttt{)}$$

Neste caso, u_i e u_j representam as utilidades conflitantes que caracterizam o dilema, e o HITL assume a responsabilidade de decidir qual ação o sistema deve executar.

3.1.4 Limitações

A arquitetura foi projetada para operar em cenários de cruzamentos urbanos com semáforo, onde o VA pode encontrar pedestres, veículos (frontais, traseiros ou laterais) e obstáculos. O referencial normativo utilizado é o *German Ethics Code*, cujas diretrizes relevantes foram apresentadas no Quadro 1.

A arquitetura não aborda cenários de alta velocidade em rodovias, tráfego intenso com múltiplos agentes simultâneos ou situações onde o tempo de resposta é menor que o necessário para o ciclo completo de comunicação entre Ético e VA. Essas limitações são características do escopo de cruzamentos urbanos definido no projeto original.

3.2 Análise PEAS do Sistema

A análise PEAS (*Performance, Environment, Actuators, Sensors*), proposta por Russell e Norvig (2010), especifica formalmente agentes inteligentes através de quatro dimensões: medida de desempenho, ambiente de operação, atuadores disponíveis e sensores que o agente utiliza para perceber o ambiente. O Quadro 2 apresenta a especificação PEAS dos três agentes do sistema multiagente implementado.

Quadro 2 – Especificação PEAS dos agentes do sistema multiagente

Agente	Performance	Environment	Actuators	Sensors
Agente VA	Respeitar regras de trânsito, garantir segurança dos envolvidos	Cruzamento urbano	Frear, desviar, seguir, modo seguro	Semáforo, pedestres, obstáculos, veículos, recomendações do Ético
Agente Ético	Identificar corretamente conflitos éticos, recomendar ações com máxima utilidade ética	Cruzamento urbano	Enviar ECP, responder consultas	Semáforo, pedestres, obstáculos, veículos
Agente HITL	Garantir resolução adequada de dilemas éticos	Não interage com o Ambiente, somente com o SMA	Enviar decisões, registrar logs	Logs de eventos

Fonte: Autoria própria.

Tanto o Agente VA quanto o Agente Ético percebem o mesmo ambiente físico — o cruzamento urbano — mas com propósitos diferentes. O VA percebe para decidir ações de condução, enquanto o Ético percebe para identificar conflitos morais. O Agente HITL não interage com o ambiente físico, apenas com o sistema multiagente através de logs e comunicações dos demais agentes.

3.3 Classificação e Formalização dos Conflitos

O sistema detecta e classifica conflitos éticos em cruzamentos urbanos com base nas percepções do ambiente. Um conflito ético ocorre quando há incompatibilidade entre valores morais que orientam diferentes ações possíveis entre os agentes, exigindo que o Agente Ético avalie qual ação respeita melhor as diretrizes do *German Ethics Code*.

3.3.1 Tipos de Conflito

O sistema reconhece cinco tipos principais de conflito ético no domínio de cruzamentos urbanos.

3.3.1.1 Conflito Pedestre

O conflito **pedestre** ocorre quando há um pedestre atravessando a faixa enquanto o veículo se aproxima do cruzamento. Há uma contradição entre respeitar o direito de passagem do pedestre (EG 1 — proteção da vida humana tem prioridade máxima) e manter a velocidade

planejada do veículo. O comportamento esperado do VA é frear para evitar atropelamento, ação que maximiza a utilidade ética ao proteger a vida humana.

3.3.1.2 Conflito Pedestre em Zona Escolar

O conflito **pedestre em zona escolar** compartilha a estrutura do conflito pedestre, mas ocorre em área de alta vigilância onde a presença de crianças torna o risco moral consideravelmente maior. A contradição é a mesma — direito de passagem do pedestre versus continuidade do trajeto do VA —, porém o contexto exige margem de segurança ampliada antes de autorizar qualquer decisão autônoma. Por isso, esse tipo recebe tratamento mais rigoroso no MODD, fazendo com que situações de menor margem decisória sejam transferidas ao HITL mesmo sem configurar empate moral.

3.3.1.3 Conflito Veículo

O conflito **veículo** surge quando há um veículo bloqueando parcialmente o cruzamento (por exemplo, veículo quebrado ou estacionado irregularmente). A contradição está entre seguir em frente e arriscar colisão versus desviar e potencialmente violar regras de trânsito. O Agente Ético calcula as utilidades considerando tanto a segurança do VA quanto a integridade do fluxo de trânsito.

3.3.1.4 Conflito Obstáculo

O conflito **obstáculo** caracteriza-se pela presença de um objeto bloqueando a via (entulho, cone de sinalização, animal na pista). A contradição central é entre seguir e colidir com o obstáculo versus desviar e potencialmente invadir outra faixa ou calçada. Quando há também um veículo lateral, este conflito se torna mais complexo, pois a manobra de desvio fica impedida, exigindo que o Agente Ético recalcule as utilidades considerando que apenas frear ou seguir são ações viáveis.

3.3.1.5 Conflito Pedestre com Veículo Traseiro

O conflito **pedestre com veículo traseiro** configura um dilema ético no sentido formal da definição. Há simultaneamente um pedestre atravessando a faixa e um veículo aproximando-se por trás do VA. A contradição fundamental está entre dois objetivos moralmente equivalentes: proteger a vida do pedestre (EG 1) versus proteger a vida dos passageiros do veículo traseiro, que podem sofrer colisão se o VA frear abruptamente. Conforme a diretriz EG 7 (Quadro 1), em situações de perigo inevitável a proteção de indivíduos não pode ser baseada em características pessoais — logo, ambas as vidas têm igual peso moral. Como nenhuma escolha é eticamente superior à outra, essa contradição não pode ser resolvida de forma programada e precisa ser transferida para supervisão humana através do Agente HITL.

3.3.2 Aplicação do Triângulo de Galtung

O sistema estrutura cada conflito ético segundo o modelo ABC apresentado no Capítulo 2, mapeando os três vértices do triângulo para elementos concretos do cenário de cruzamento. O vértice **Atitude** corresponde ao estado do ambiente no momento da detecção — no conflito pedestre, as percepções capturam semáforo aberto, pedestre na faixa, ausência de veículo traseiro — representando o conhecimento disponível ao Agente Ético para fundamentar a avaliação moral. O vértice **Contradição** identifica as metas incompatíveis entre VA e Ético: no conflito pedestre, o VA possui o objetivo de respeitar o sinal verde e atravessar o cruzamento, enquanto o Ético possui o objetivo de proteger a vida do pedestre. Ambos os objetivos são moralmente válidos quando considerados isoladamente, mas tornam-se incompatíveis naquele cenário específico. O vértice **Comportamento** especifica as ações opostas que cada meta demanda — seguir em frente versus frear — sendo a escolha entre esses comportamentos o que define a resolução do conflito.

No dilema ético pedestre-veículo traseiro, a estrutura evidencia o empate moral através da simetria dos vértices. A Contradição mapeia dois objetivos de igual peso segundo a diretriz EG 7: proteger o pedestre versus proteger os passageiros do veículo traseiro. Os Comportamentos opostos — frear protege o pedestre mas arrisca colisão traseira, seguir faz o inverso — produzem consequências morais equivalentes, caracterizando uma situação onde nenhuma escolha é eticamente superior à outra. O conflito obstáculo com veículo lateral apresenta estrutura semelhante ao pedestre, mas com a Contradição centrada entre preservar a integridade do VA (desviar é a ação natural) e respeitar regras de trânsito quando não há espaço seguro para manobra (frear torna-se necessário).

Essa formalização permite que o sistema registre cada conflito de forma verificável — capturando não apenas qual ação foi escolhida, mas também quais percepções fundamentaram a decisão, quais metas estavam em conflito e quais comportamentos alternativos foram considerados —, atendendo ao requisito de transparência estabelecido pela diretriz EG 20 do *German Ethics Code*.

3.3.3 Capacidades do Sistema dentro do MODD

O conceito de MODD proposto por Rakow *et al.* (2024) estabelece que sistemas autônomos devem reconhecer quando um conflito está fora de seu domínio operacional moral, transferindo a decisão para autoridade humana. Para operacionalizar esse conceito, este trabalho define limiares de confiança (θ) específicos para cada tipo de conflito, representando a diferença mínima de utilidade necessária entre a melhor e a segunda melhor ação para que o sistema seja autorizado a decidir autonomamente.

A capacidade de *awareness* (consciência) manifesta-se na detecção de cinco tipos de conflito ético no domínio de cruzamentos urbanos: pedestre, pedestre em zona escolar, veículo frontal, obstáculo e pedestre com veículo traseiro. A detecção ocorre através da análise combi-

nada das sete percepções do ambiente, onde cada combinação específica de valores ativa um tipo de conflito.

A capacidade de *resolution* (resolução) é regulada pelos limiares de confiança definidos no MODD. Para o conflito pedestre simples, o limiar é $\theta = 1{,}0$ — o sistema pode resolver autonomamente quando a diferença entre a melhor e a segunda melhor ação é pelo menos $1{,}0$. Para obstáculos, o limiar é reduzido para $\theta = 0{,}5$, considerando que decisões sobre objetos inanimados envolvem risco moral menor que decisões sobre vidas humanas. Já no conflito pedestre em zona escolar, o limiar é elevado para $\theta = 4{,}0$, exigindo margem de segurança significativamente maior antes de autorizar decisão autônoma.

A capacidade de *awareness* (consciência) manifesta-se na detecção de cinco tipos de conflito ético no domínio de cruzamentos urbanos: pedestre, pedestre em zona escolar, veículo frontal, obstáculo e pedestre com veículo traseiro. A detecção ocorre através da análise das percepções do ambiente seguindo uma hierarquia de prioridade — cenários que configuram dilema ético genuíno são identificados primeiro, seguidos por conflitos simples em contextos especiais (zona escolar), e por fim os demais tipos. Essa lógica garante que o sistema classifique corretamente situações onde múltiplas percepções estão ativas simultaneamente.

3.3.4 Critério Formal de Dilema Ético

Conforme estabelecido por Alves, Dennis e Fisher (2021), um dilema ético ocorre quando não há utilidade maximizada entre as ações disponíveis, ou seja, quando as duas melhores ações produzem utilidades iguais ($u_i = u_j$). Para operacionalizar esse critério em um sistema computacional, este trabalho adota um limiar de tolerância $\epsilon = 0{,}01$:

$$\text{Dilema} \iff |u_i - u_j| < 0{,}01 \tag{2}$$

onde u_i representa a utilidade da melhor ação e u_j representa a utilidade da segunda melhor ação. Esse critério garante que apenas situações indecidíveis do ponto de vista ético sejam escaladas para o Agente HITL, evitando sobrecarga da supervisão humana.

No conflito pedestre simples, $|10 - (-1)| = 11 > 0{,}01$, logo não é dilema. No conflito pedestre com veículo traseiro, $|10 - 10| = 0 < 0{,}01$, configurando dilema ético que deve ser escalado para o HITL conforme estabelecido pela diretriz EG 8 (Quadro 1).

O limiar $\epsilon = 0{,}01$ pode ser ajustado conforme a sensibilidade desejada para a detecção de dilemas éticos. Toda decisão de escalar ou não um conflito para supervisão humana pode ser verificada pelo cálculo das utilidades e pela aplicação deste critério matemático — garantindo que o sistema opere de forma transparente e verificável.

3.4 Funções de Utilidade Ética

Conforme apresentado na fundamentação teórica, o sistema adota uma abordagem utilitarista para o raciocínio ético, onde cada ação possível (frear, desviar, seguir) recebe um valor numérico que representa suas consequências morais. A arquitetura proposta por Alves, Dennis e Fisher (2021) define constantes de utilidade ética baseadas nas diretrizes do *German Ethics Code*, estabelecendo uma escala onde proteção à vida humana tem prioridade absoluta, seguida pelo cumprimento de regras de trânsito. Este trabalho estende essas constantes através da definição de fatores de ponderação que ajustam as utilidades conforme o contexto específico de cada conflito.

3.4.1 Constantes de Utilidade

O sistema define cinco constantes que servem como base para o cálculo das utilidades éticas. O Quadro 3 apresenta essas constantes, seus valores e a justificativa para cada escolha.

Quadro 3 – Constantes de utilidade ética definidas neste trabalho

Constante	Valor	Base (EG)	Justificativa
`RU_SAFE`	10	EG 1, EG 2	Proteção de usuários da via (pedestres, ciclistas, motoristas). Valor máximo da escala, representando prioridade absoluta da vida humana.
`AV_PASS_SAFE`	10	EG 7	Proteção dos passageiros do VA. Igual a RU_SAFE pois EG 7 estabelece que em situações de perigo, pedestres e passageiros têm igual valor moral.
`AV_RESPECTS_RULE`	9	EG 5	Cumprimento das regras de trânsito. Valor 9 é inferior a 10, refletindo que regras devem ceder quando há conflito com proteção de vida.
`AV_DAMAGE`	-1	-	Dano menor ao VA (desgaste de freio, desconforto de passageiros). Valor negativo pequeno, indicando consequência indesejável mas tolerável.
`TRAFFIC_ENV_DAMAGE`	-2	-	Dano ao ambiente de trânsito (invadir faixa, bloquear fluxo). Valor mais negativo que AV_DAMAGE, pois afeta terceiros.

Fonte: Adaptado de Alves, Dennis e Fisher (2021).

3.4.2 Funções de Utilidade para Cada Ação

As funções de utilidade calculam o valor moral de cada ação (frear, desviar, seguir) considerando o tipo de conflito e o estado do ambiente. O Quadro 4 sintetiza os valores calculados para os principais cenários.

Quadro 4 – Utilidades calculadas para cada ação por tipo de conflito

Tipo de Conflito	$u_{\textbf{frear}}$	$u_{\textbf{desviar}}$	$u_{\textbf{seguir}}$
Pedestre (sem. aberto)	$10 + (-1) = 9$	$7 + (-1) = 6$	$9 + (-10) = -1$
Pedestre zona escolar (sem. aberto)*	$10 + (-1) = 9$	$7 + (-1) = 6$	$9 + (-10) = -1$
Pedestre + veíc. traseiro (dilema)	10	6	10
Veículo (sem. aberto)	$8 + (-1) = 7$	$6 + (-1) = 5$	$9 + (-5) = 4$
Obstáculo	7	$7{,}5 + (-1) = 6{,}5$	$9 + (-10) = -1$

Fonte: Autoria Própria.

** No dilema ético, frear e seguir produzem utilidades iguais ($u_{\text{frear}} = u_{\text{seguir}} = 10$), caracterizando empate moral que exige transferência ao HITL.

3.4.2.1 Utilidade de Frear

A ação de frear possui utilidade máxima quando há pedestre na faixa: $u_{\text{frear}} =$ RU_SAFE $= 10$, refletindo que proteger a vida do pedestre é prioridade absoluta (EG 1). Quando o semáforo está aberto (verde), adiciona-se a penalidade AV_DAMAGE $= -1$, resultando em $u_{\text{frear}} = 9$. Essa penalidade captura que frear no verde causa desconforto aos passageiros e desgaste ao veículo, mas o valor resultante ainda é superior a seguir ($u_{\text{seguir}} = -1$), confirmando que proteger o pedestre justifica moralmente o custo de frear no verde.

No conflito com veículo frontal, frear reduz o risco de colisão mas não o elimina completamente, logo $u_{\text{frear}} =$ AV_PASS_SAFE $\times\, 0{,}8 = 8$. O fator $0{,}8$ representa que a eficácia de frear depende da distância — se o veículo estiver muito próximo, a colisão pode ocorrer mesmo frenando. No conflito com obstáculo, frear permite parar antes do objeto: $u_{\text{frear}} =$ AV_PASS_SAFE $\times\, 0{,}7 = 7$, onde o fator $0{,}7$ indica que frear não é necessariamente a melhor solução (desviar pode ser mais eficiente dependendo da distância e espaço lateral).

No cenário de dilema ético (pedestre com veículo traseiro), a utilidade é simplesmente $u_{\text{frear}} =$ RU_SAFE $= 10$, sem penalidades adicionais. O que define esse cenário como dilema é o fato de que u_{seguir} também vale 10 (proteção dos passageiros traseiros), criando empate moral.

3.4.2.2 Utilidade de Desviar

A ação de desviar oferece proteção parcial em conflitos com pedestre: $u_{\text{desviar}} = \text{RU\_SAFE} \times 0{,}7 = 7$. O fator $0{,}7$ captura que desviar reduz o risco mas não o elimina — o pedestre pode estar em trajetória que ainda intercepta o veículo. Adiciona-se a penalidade TRAFFIC_ENV_DAMAGE $\times\, 0{,}5 = -1$, representando que desviar pode invadir outra faixa ou calçada, resultando em utilidade líquida $u_{\text{desviar}} = 6$.

No conflito com veículo, $u_{\text{desviar}} = \text{AV\_PASS\_SAFE} \times 0{,}6 - 1 = 5$. No conflito com obstáculo, desviar é mais eficiente: $u_{\text{desviar}} = \text{AV\_PASS\_SAFE} \times 0{,}75 - 1 = 6{,}5$, pois permite manter velocidade enquanto evita o objeto, desde que haja espaço lateral.

3.4.2.3 Utilidade de Seguir

A ação de seguir, quando o semáforo está aberto, respeita a regra de trânsito: $u_{\text{seguir}} = \text{AV\_RESPECTS\_RULE} = 9$. No conflito com pedestre, porém, seguir resulta em atropelamento: $u_{\text{seguir}} = 9 - \text{RU\_SAFE} = -1$. Essa utilidade negativa indica que seguir é moralmente inaceitável neste cenário, mesmo com sinal verde.

No conflito com veículo, seguir arrisca colisão: $u_{\text{seguir}} = 9 - (\text{AV\_PASS\_SAFE} \times 0{,}5) = 4$. No conflito com obstáculo, seguir resulta em colisão: $u_{\text{seguir}} = 9 - \text{AV\_PASS\_SAFE} = -1$, novamente uma ação moralmente inaceitável.

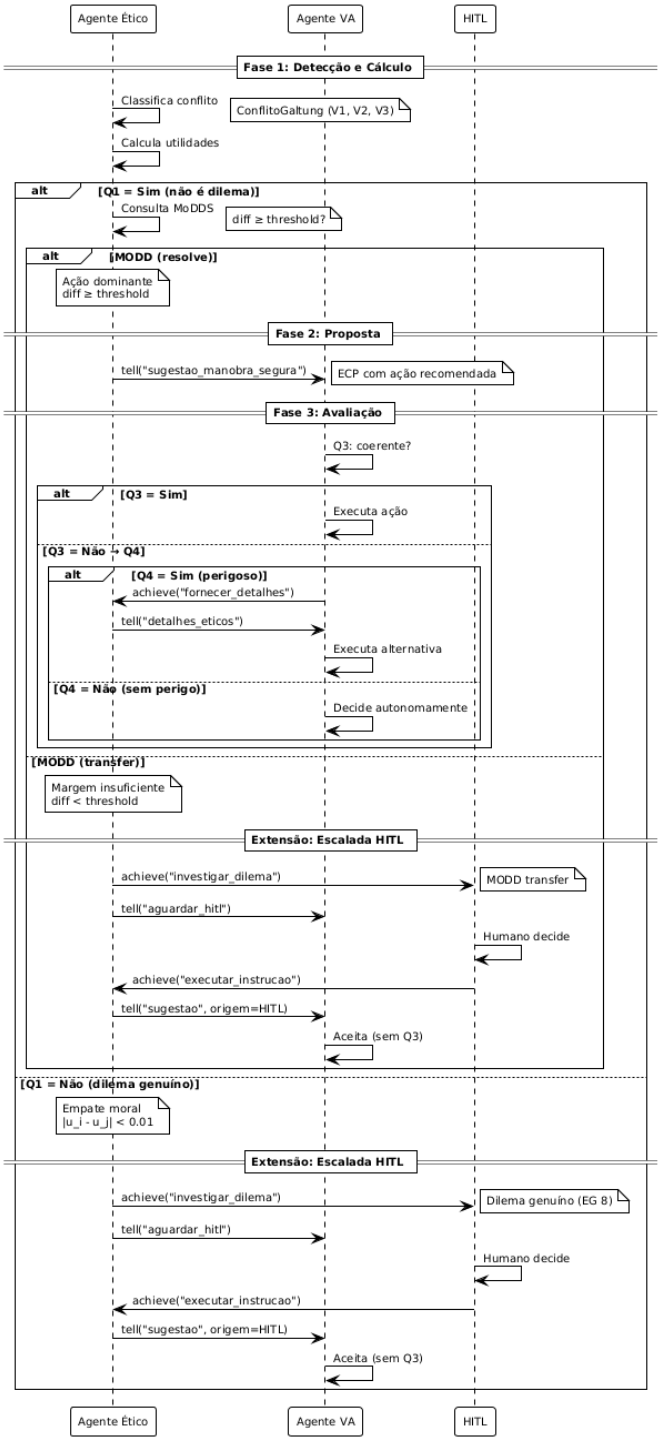
No cenário de dilema ético, seguir protege os passageiros do veículo traseiro (evitando colisão traseira), logo $u_{\text{seguir}} = \text{AV\_PASS\_SAFE} = 10$.

As constantes de utilidade foram estabelecidas por Alves, Dennis e Fisher (2021) com base na hierarquia moral do *German Ethics Code*. Os fatores de ponderação foram definidos neste trabalho para ajustar as utilidades conforme o contexto específico de cada conflito, refletindo que a eficácia de cada ação varia segundo a distância, espaço disponível e risco envolvido.

3.5 Protocolo Contract Net: Implementação e Adaptações

A coordenação entre o Agente VA e o Agente Ético é estruturada segundo o protocolo Contract Net, proposto por Smith (1980) e apresentado na fundamentação teórica. No protocolo clássico, um agente gerente distribui tarefas entre múltiplos participantes através de chamadas de propostas, avaliação de ofertas e alocação de responsabilidades. Neste trabalho, o protocolo foi adaptado para o contexto de raciocínio ético — o VA atua como gerente solicitando avaliação ética de uma situação, o Agente Ético atua como participante único calculando utilidades e propondo ações, e o VA decide se aceita, rejeita ou solicita alternativas adicionais. A Figura 6 apresenta a estrutura completa da negociação adaptada, incluindo os três agentes envolvidos, as três fases principais e os caminhos de decisão.

Figura 6 – Protocolo Contract Net adaptado — Negociação entre VA (Gerente) e Agente Ético (Participante)



Fonte: Autoria própria.

3.5.1 Papéis dos Agentes

A Figura 6 mostra que o protocolo envolve três agentes com papéis distintos. O Agente Ético — identificado como "Participante"no diagrama — assume o papel de contratado (*con-*

tractor), sendo responsável por monitorar o ambiente, detectar conflitos e calcular utilidades. Diferentemente do protocolo clássico, há apenas um participante, o que significa que o sistema não envolve múltiplos agentes éticos competindo por tarefas.

O Agente VA — identificado como "Gerente"no diagrama — atua como gerente (*manager*) da negociação, avaliando as propostas recebidas do Agente Ético e decidindo se aceita a recomendação, solicita alternativas ou descarta a proposta para tomar decisão autônoma. Quando o VA se aproxima de uma situação onde uma decisão ética pode ser necessária (proximidade de cruzamento, presença de pedestres, obstáculos ou outros veículos), o Agente Ético monitora de forma autônoma e inicia o processo de negociação.

Já o Agente HITL — identificado como "Supervisor"no diagrama — não participa da negociação padrão entre VA e Ético. Conforme mostra o caminho inferior da figura ("Q1 = Não"), ele é acionado quando um dilema ético é detectado conforme a diretriz EG 8 (Quadro 1) ou quando o protocolo entre VA e Ético falha em resolver o conflito, transferindo o controle para supervisão humana através da extensão "Escalada para HITL".

3.5.2 Estrutura da Negociação

A negociação segue três fases delimitadas na Figura 6. Na **Fase 1 — Detecção e Cálculo**, o Agente Ético monitora o ambiente, detecta conflito a partir das percepções do cruzamento, instancia a estrutura Galtung correspondente e calcula as utilidades das três ações possíveis usando as funções definidas na seção 3.4. Em seguida aplica Q1 e, se não houver dilema, consulta o MODD para verificar autorização de decisão autônoma — ambos conforme o fluxo já apresentado na seção 3.1. Quando o MODD autoriza, segue-se a **Fase 2 — Envio da Proposta**: o Agente Ético envia o ECP ao VA, marcando a transição de responsabilidade do contratado para o gerente.

3.5.3 Caminhos de Decisão

Após o envio da proposta, o diagrama mostra três caminhos possíveis. No primeiro (Q3 = Sim), o VA aceita a recomendação e a executa, encerrando a negociação com sucesso. No segundo (Q3 = Não com Q4 = Sim), o VA detecta incoerência em situação perigosa e solicita alternativa ao Ético, que recalcula com a informação adicional e envia nova proposta. No terceiro (Q3 = Não com Q4 = Não), não há perigo imediato e o VA descarta a recomendação, decidindo autonomamente. O detalhamento de cada questão Q3/Q4 já foi apresentado na seção 3.1; a contribuição desta seção é o mapeamento desses caminhos para o protocolo Contract Net.

3.5.4 Extensão para Dilemas Éticos

O caminho inferior do diagrama representa uma extensão ao Contract Net clássico. Quando o Agente Ético detecta dilema (Q1 = Não), em vez de prosseguir para a Fase 2, transfere a responsabilidade ao Agente HITL através de mensagem solicitando investigação do dilema. O HITL analisa o conflito e devolve a decisão ao Ético, que repassa ao VA marcando a origem como HITL. Quando o VA recebe uma proposta com origem HITL, ele a aceita diretamente sem aplicar Q3, conforme a diretriz EG 19 que estabelece rotinas de transferência de controle (*handover*). Decisões humanas têm, portanto, precedência sobre as geradas autonomamente pelo Agente Ético.

3.6 Estrutura BDI dos Agentes

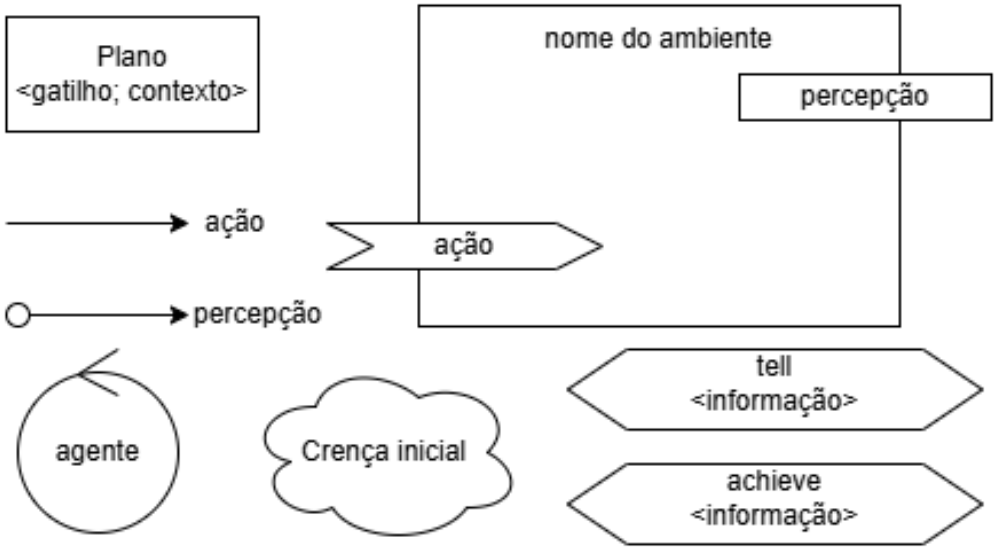
A especificação completa do sistema multiagente é apresentada através do diagrama UML MASPY — notação visual criada pelos desenvolvedores do framework MASPY para representar sistemas com arquitetura BDI (Mellado; Borges; Alves, 2025). Diferente do UML clássico, essa notação foi projetada para capturar conceitos nativos do MASPY (crenças, objetivos, planos, ambientes e comunicação entre agentes), onde cada componente visual tem correspondência direta no código Python. A versão completa do diagrama UML MASPY pode ser consultada no Apêndice A.

Por questão de legibilidade, o diagrama do sistema é apresentado em cinco partes: primeiro a notação utilizada (Figura 7), seguida pelos diagramas individuais do Agente VA (Figura 8), Agente Ético (Figura 9), Agente HITL (Figura 10) e Ambiente Urbano (Figura 11).

3.6.1 Notação do Diagrama UML MASPY

A Figura 7 apresenta os componentes visuais da notação MASPY, onde cada elemento tem correspondência direta no código Python do framework.

Figura 7 – Notação do diagrama UML MASPY

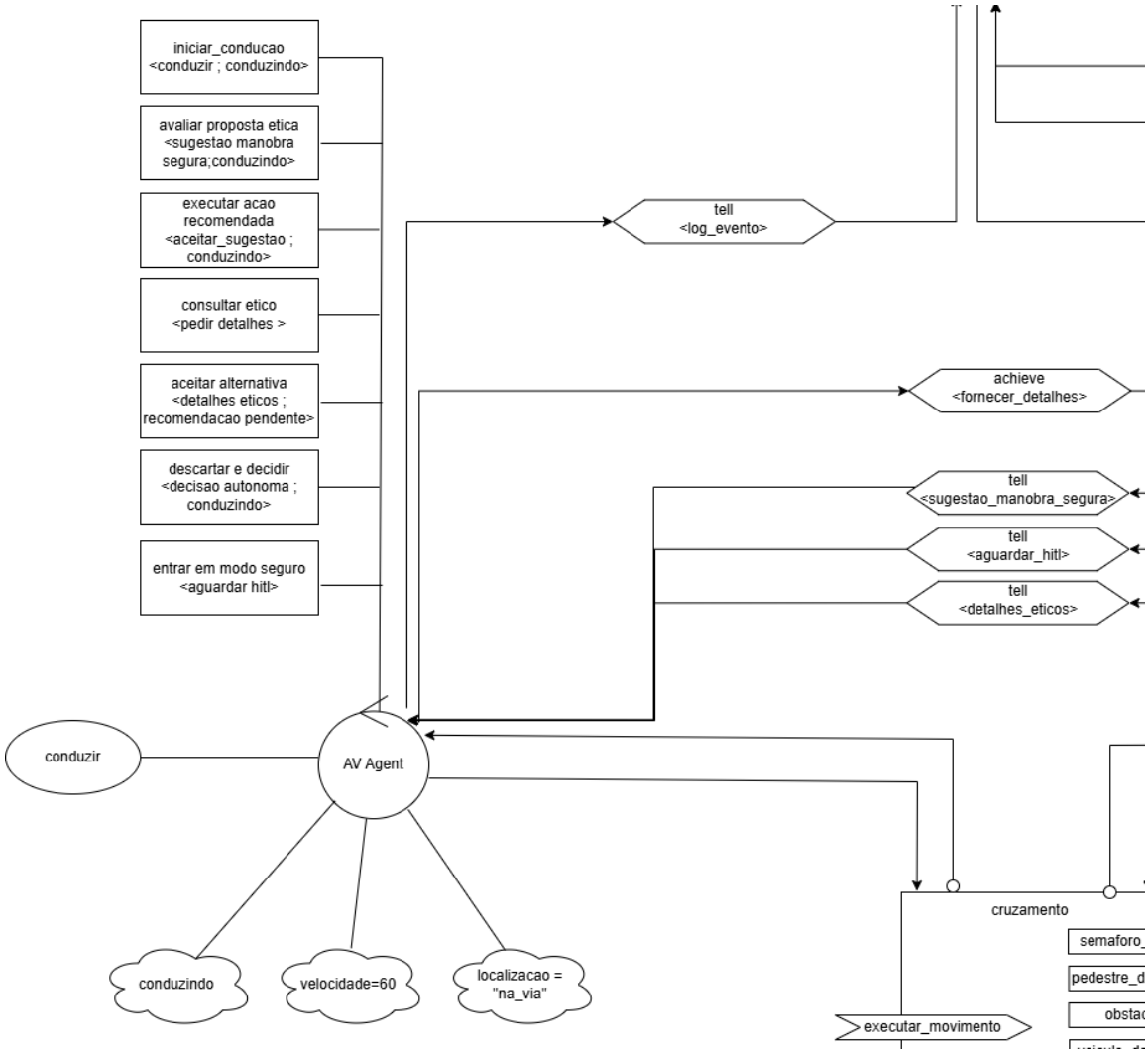


Fonte: Autoria própria.

3.6.2 Agente VA

A Figura 8 apresenta a estrutura BDI completa do Agente VA.

Figura 8 – Diagrama UML MASPY do Agente VA



Fonte: Autoria própria.

O VA possui um objetivo inicial — `conduzir` — ativado ao iniciar o sistema. Suas crenças iniciais incluem `conduzindo`, `velocidade` e `localizacao`, configuradas com valores que representam o estado de operação do veículo.

Há sete planos que implementam o fluxo de decisão apresentado na seção 3.1. O plano `iniciar_conducao` percebe o ambiente e aguarda recomendações. Ao receber uma recomendação ética, o plano `avaliar_proposta_etica` aplica Q3 (coerência) e Q4 (perigo). Se Q3 = Sim, o plano `executar_acao_recomendada` executa a ação. Se Q3 = Não e Q4 = Sim, o plano `consultar_etico` solicita alternativa e o plano `aceitar_alternativa` processa a resposta. Se Q3 = Não e Q4 = Não, o plano `descartar_e_decidir` decide autonomamente. Quando há dilema, o plano `entrar_em_modo_seguro` ativa *handover*.

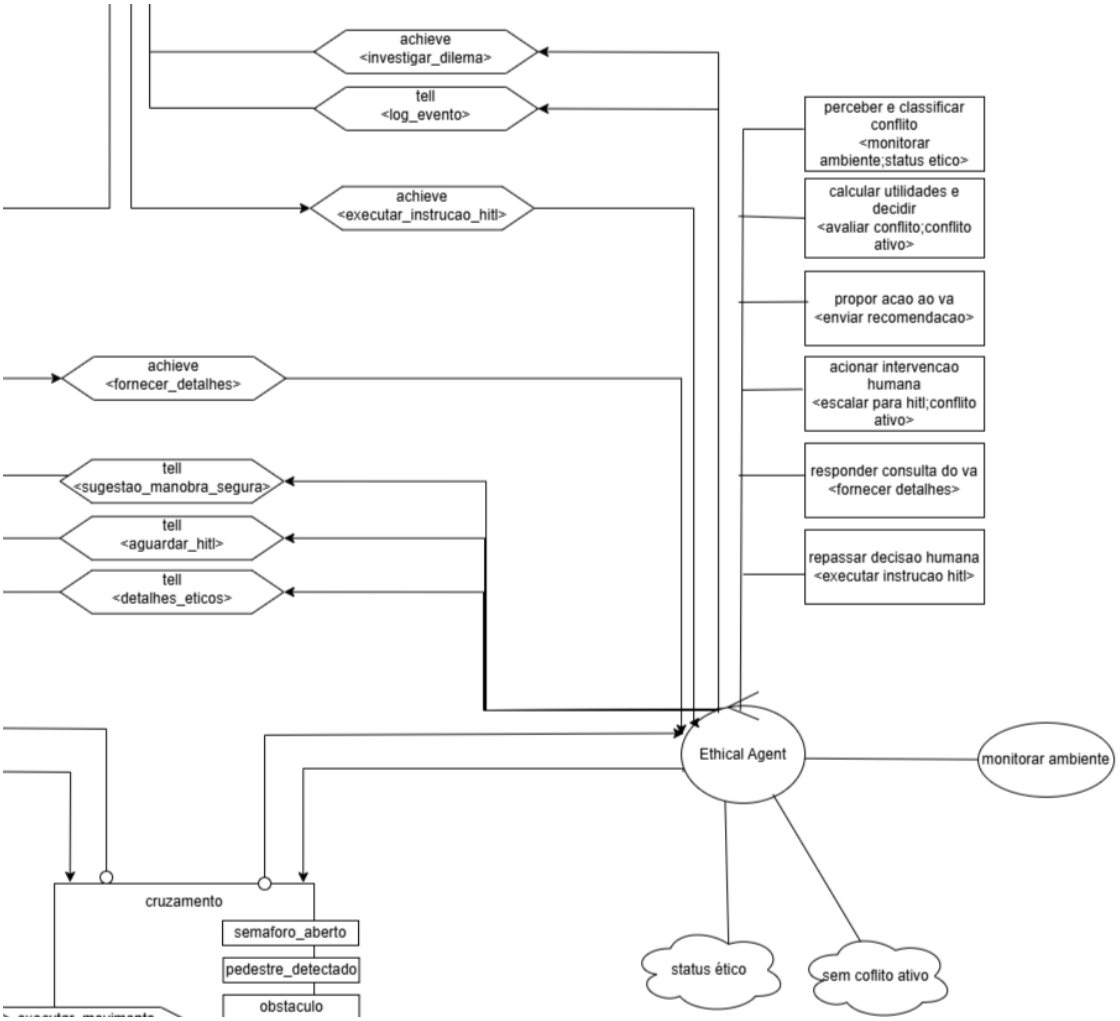
O VA lê cinco percepções do ambiente: `semaforo_aberto`, `pedestre_detectado`, `obstaculo`, `veiculo_detectado` e `veiculo_lateral`. Executa a ação `executar_movimento` para efetuar o movimento físico.

Recebe três mensagens `tell` do Ético: `sugestao_manobra_segura`, `aguardar_hitl` e `detalhes_eticos`. Envia mensagem `achieve` ao Ético quando solicita alternativa (`fornecer_detalhes`) e mensagens `tell` ao HITL para auditoria (`log_evento`).

3.6.3 Agente Ético

A Figura 9 apresenta a estrutura BDI completa do Agente Ético.

Figura 9 – Diagrama UML MASPY do Agente Ético



Fonte: Autoria própria.

O Ético possui um objetivo inicial — `monitorar_ambiente` — que dispara o ciclo de percepção e classificação. Suas crenças iniciais incluem `status_etico` e `sem_conflito_ativo`, indicando que o agente inicia em modo de monitoramento sem conflito ativo.

Há seis planos que implementam o raciocínio ético. O plano `perceber_e_classificar_conflito` lê as percepções e classifica o conflito conforme a seção 3.3.

O plano `calcular_utilidades_e_decidir` calcula as utilidades usando as funções da seção 3.4 e aplica Q1. Se Q1 = Sim, o plano `propor_acao_ao_va` monta e envia o ECP, implementando a Fase 2 do protocolo da seção 3.5. Se Q1 = Não, o plano `acionar_intervencao_humana` escala para o HITL. O plano `responder_consulta_do_va` recalcula com informação adicional (resposta Q4). O plano `repassar_decisao_humana` recebe decisão do HITL e repassa ao VA.

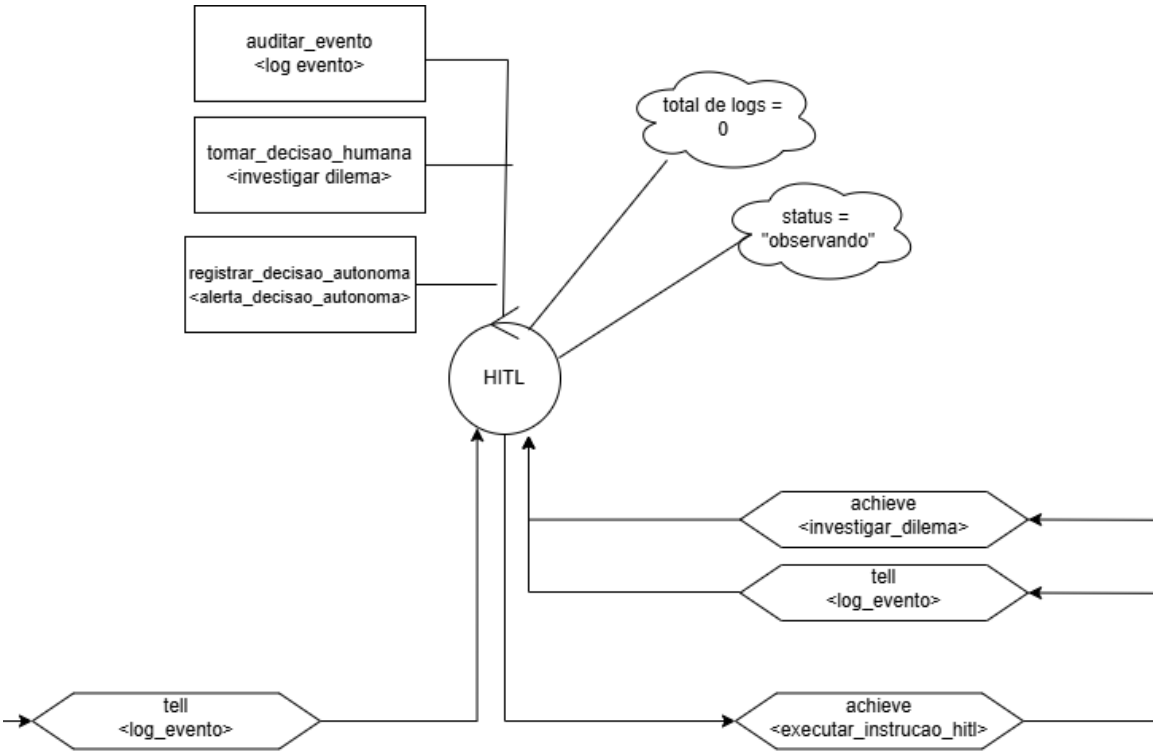
O Agente Ético lê sete percepções do ambiente: `semaforo_aberto`, `pedestre_detectado`, `obstaculo`, `veiculo_detectado`, `veiculo_atras`, `veiculo_lateral` e `zona_escolar`. Note que `veiculo_atras` e `zona_escolar` são lidas apenas pelo Ético — a primeira porque é necessária para detectar o dilema ético pedestre-veículo traseiro, a segunda porque ativa regras especiais de prudência no MODD, elevando o limiar de confiança para decisão autônoma de $\theta = 1{,}0$ para $\theta = 4{,}0$. O Ético não executa ações no ambiente.

Envia mensagens `tell` ao VA (`sugestao_manobra_segura`, `aguardar_hitl`, `detalhes_eticos`), mensagens `achieve` ao HITL (`investigar_dilema`) e mensagens `tell` ao HITL (`log_evento`). Recebe mensagens `achieve` do VA (`fornecer_detalhes`) e do HITL (`executar_instrucao_hitl`).

3.6.4 Agente HITL

A Figura 10 apresenta a estrutura BDI completa do Agente HITL.

Figura 10 – Diagrama UML MASPY do Agente HITL



Fonte: Autoria própria.

O HITL possui apenas crenças iniciais — `status` e `total_logs`, configuradas como observador sem registros — sem objetivo inicial, pois atua apenas reativamente.

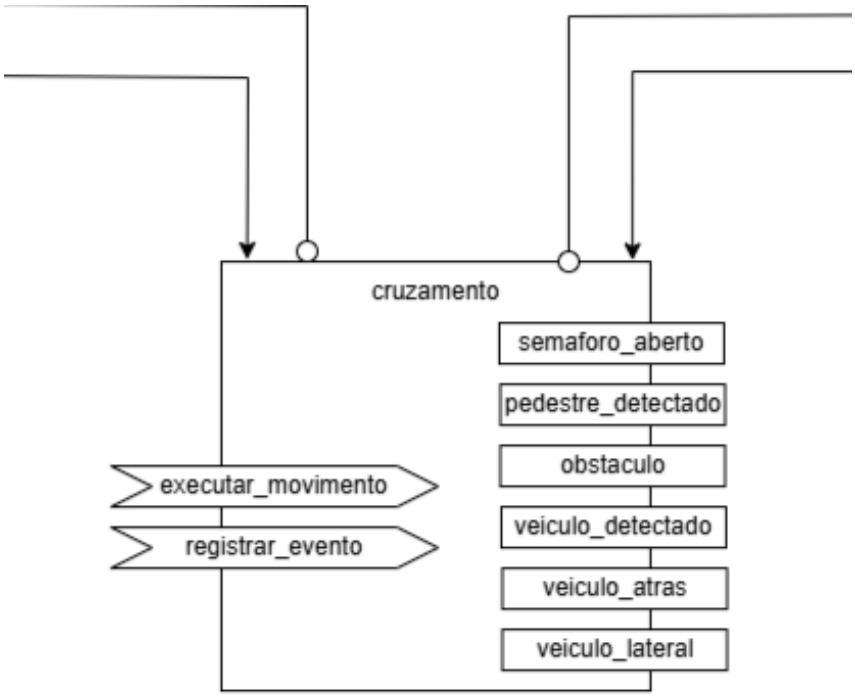
Há três planos. O plano `auditar_evento` registra eventos e emite alerta quando o VA decide autonomamente. O plano `tomar_decisao_humana` analisa dilemas e envia instrução ao Ético. O plano `registrar_decisao_autonoma` registra situações onde o VA descartou recomendação.

O HITL não está conectado ao ambiente. Recebe mensagens `achieve` do Ético (`investigar_dilema`) e mensagens `tell` do Ético e VA (`log_evento`). Envia mensagens `achieve` ao Ético (`executar_instrucao_hitl`).

3.6.5 Ambiente Urbano

A Figura 11 apresenta o Ambiente Urbano, representando o cruzamento onde o VA opera.

Figura 11 – Diagrama UML MASPY do Ambiente Urbano



Fonte: Autoria própria.

O ambiente disponibiliza sete percepções que representam o estado completo do cruzamento. As percepções `semaforo_aberto` (1 = verde, 0 = vermelho), `pedestre_detectado`, `obstaculo`, `veiculo_detectado`, `veiculo_atras` e `veiculo_lateral` capturam os elementos dinâmicos da via. A percepção `zona_escolar` (1 = zona ativa, 0 = fora de zona) indica quando o cruzamento está em área de alta vigilância, ativando regras especiais de prudência no MODD — conforme apresentado na seção 3.3.

O ambiente disponibiliza duas ações: `executar_movimento` (chamada pelo VA para executar frear, desviar ou seguir) e `registrar_evento` (chamada pelo VA para auditoria).

4 IMPLEMENTAÇÃO

Este capítulo apresenta os aspectos de implementação do SMA modelado no Capítulo 3, utilizando o *framework* MASPY apresentado no Capítulo 2. Primeiramente, são apresentadas as tecnologias utilizadas e a organização do código fonte. Em seguida, são detalhadas as estruturas de dados compartilhadas entre os agentes e, por fim, são descritos os componentes do sistema: Ambiente Urbano, Agente Ético, Agente VA e Agente HITL.

O código-fonte completo da implementação descrita neste capítulo está disponível no repositório *online*. O acesso pode ser realizado através do endereço: https://github.com/igoryamazaki/traffic_intersection.

4.1 Tecnologias e Organização do Código

A implementação foi realizada integralmente em Python 3.12, por ser a linguagem de desenvolvimento do *framework* MASPY e pela vasta disponibilidade de bibliotecas científicas. Tal escolha facilita futuras extensões do sistema através de técnicas de aprendizado de máquina, permitindo, por exemplo, o treinamento de modelos para refinamento das funções de utilidade ética. O código fonte está organizado em seis módulos Python independentes, no qual cada módulo implementa um componente específico do sistema, conforme apresentado no Quadro 5.

Quadro 5 – Organização modular do código fonte

Módulo	**Responsabilidade**
`main.py`	Ponto de entrada do sistema; instancia os três agentes, o ambiente e o canal de comunicação; inicia o ciclo de execução do MASPY.
`environment.py`	Implementa o `AmbienteUrbano`, sendo responsável por prover as sete percepções especificadas no PEAS (seção 3.2) e executar as ações físicas comandadas pelo VA.
`agent_va.py`	Implementa o `AgenteVA`; contém sete planos que realizam o fluxo de decisão Q3/Q4 e a execução de manobras físicas.
`agent_ethical.py`	Implementa o `AgenteEtico`, exercendo o papel de *Governor Module*; contém seis planos responsáveis pela detecção de conflitos via Galtung, cálculo de utilidades éticas, verificação MODD e envio de recomendações via Contract Net.
`agent_hitl.py`	Implementa o `AgenteHITL`; contém três planos responsáveis por registro de eventos, investigação de dilemas éticos e envio de instruções aos demais agentes.
`utilities.py`	Biblioteca compartilhada contendo: (1) classe `ConflitoGaltung` e cinco *factory functions* para instanciação de conflitos, (2) funções de cálculo de utilidade ética baseadas no *German Ethics Code*.

Fonte: Autoria própria.

Essa organização modular segue o princípio de separação de responsabilidades — cada módulo implementa um componente específico do sistema, facilitando tanto a manutenção quanto os testes independentes de cada parte.

O fluxo de execução inicia em `main.py`, que configura o sistema completo. Primeiramente, são instanciados os três agentes (VA, Ético e HITL) e o ambiente urbano, cada um com suas crenças e objetivos iniciais conforme modelados no Capítulo 3. Em seguida, é criado o canal de comunicação `AgentComm`, permitindo que os agentes troquem mensagens. Posteriormente, o ambiente é conectado aos agentes VA e Ético — mas não ao HITL, conforme especificado no PEAS do seção 3.2, já que o HITL não interage diretamente com o mundo físico. Por fim, o método `maspy.run()` inicia o ciclo de raciocínio BDI, no qual cada agente percebe o ambiente, seleciona planos aplicáveis e executa ações de forma autônoma até que alguma condição de parada seja satisfeita ou que não haja mais planos ativos no sistema.

4.2 Triângulo de Galtung: Estruturas de Dados Compartilhadas

O módulo `utilities.py` implementa as estruturas de dados e funções auxiliares utilizadas pelos agentes. A principal estrutura é a classe `ConflitoGaltung`, que codifica em campos internos os três vértices do conflito formalizado na seção 3.3. Para compatibilidade com o MASPY, a classe herda de `HashableDict`, permitindo que instâncias de conflito sejam transmitidas através do sistema de crenças e objetivos do *framework*.

O método `explicar` (Listagem 2) retorna uma descrição textual do conflito registrada em *log* pelo Agente Ético após cada cálculo de utilidades, permitindo verificar quais percepções fundamentaram cada recomendação ética.

Listagem 2 – Classe ConflitoGaltung com três vértices

```
1  class ConflitoGaltung(HashableDict):
2      def __init__(self, dados_adfers, metas_incompativeis, acoes_opostas, crencas):
3          super().__init__(dados_adfers)
4          self["metas_incompativeis"] = tuple(metas_incompativeis)
5          self["acoes_opostas"] = tuple(acoes_opostas)
6          self["crencas"] = HashableDict(crencas)
7
8      def explicar(self):
9          v1 = dict(self["crencas"])
10         v2 = self["metas_incompativeis"]
11         v3 = self["acoes_opostas"]
12         return f"Galtung V1(crencas)={v1} | V2(metas)={v2} | V3(acoes)={v3}"
```

Fonte: Autoria própria (2026).

Cada um dos cinco tipos de conflito possui uma função de criação específica que recebe as percepções do ambiente e constrói a instância correspondente. O Listagem 3 mostra a função para conflitos com pedestre.

Listagem 3 – Factory function para conflito com pedestre

```
def criar_conflito_pedestre(semaforo, pedestre, obstaculo, veiculo,
                            v_atras, v_lateral):
    acao_va_padrao = "seguir" if semaforo else "frear"
    return ConflitoGaltung(
        {"tipo": "pedestre", "semaforo": "aberto" if semaforo else "fechado"},
        metas_incompativeis=("VA: respeitar_sinal_transito",
                             "Etico: proteger_pedestre"),
        acoes_opostas=(acao_va_padrao, "frear"),
        crencas={"semaforo": semaforo, "pedestre": pedestre,
                 "obstaculo": obstaculo, "veiculo": veiculo,
                 "veiculo_atras": v_atras, "veiculo_lateral": v_lateral}
    )
```

Fonte: Autoria própria (2026).

A separação em cinco funções distintas — em vez de uma função única com parâmetro `tipo` — facilita a manutenção, pois cada uma encapsula lógica específica daquele cenário. Por exemplo, `criar_conflito_obstaculo` implementa lógica dinâmica onde a ação do Agente Ético varia conforme o estado do semáforo: se aberto, recomenda desviar e, se fechado, recomenda frear.

4.3 Funções de Utilidade Ética

O módulo `utilities.py` também implementa as funções de cálculo de utilidade ética cujas fórmulas foram apresentadas na seção 3.4. As cinco constantes são definidas como valores globais, refletindo diretamente a hierarquia moral do *German Ethics Code*:

Listagem 4 – Constantes de utilidade ética

```
RU_SAFE             = 10   # Usuarios da via seguros (EG 1, 2)
AV_PASS_SAFE        = 10   # Passageiros do VA seguros (EG 7)
AV_RESPECTS_RULE    =  9   # VA respeitou regras de transito (EG 5)
AV_DAMAGE           = -1   # Dano menor ao VA
TRAFFIC_ENV_DAMAGE  = -2   # Dano ao ambiente de transito
```

Fonte: Autoria própria (2026).

Para cada ação possível (frear, desviar e seguir), há uma função de cálculo que aplica as constantes conforme o tipo de conflito e o estado do semáforo. O Listagem 5 apresenta a função de utilidade para frear, que serve como referência para as demais:

Listagem 5 – Função de utilidade para a ação de frear

```
def calcular_utilidade_frear(conflito):
    utilidade = 0.0
    if conflito.get("tipo") in _TIPOS_PEDESTRE:
        utilidade += RU_SAFE              # 10: protege pedestre (EG 1)
    elif conflito.get("tipo") == "veiculo":
        utilidade += AV_PASS_SAFE * 0.8   # 8: reduz risco de colisao frontal
    elif conflito.get("tipo") == "obstaculo":
        utilidade += AV_PASS_SAFE * 0.7   # 7: para antes do obstaculo
    if conflito.get("semaforo") == "aberto":
        utilidade += AV_DAMAGE            # -1: frear no verde = dano menor
    return utilidade
```

Fonte: Autoria própria (2026).

As funções `calcular_utilidade_desviar` e `calcular_utilidade_seguir` seguem a mesma estrutura, aplicando fatores de ponderação distintos conforme o tipo de conflito — os valores resultantes correspondem ao Quadro 4 apresentado na seção 3.4. O cenário de dilema ético (pedestre com veículo traseiro) possui funções separadas — `calcular_utilidade_frear_dilema` e `calcular_utilidade_seguir_dilema` — que retornam ambas `RU_SAFE = 10`, produzindo a igualdade de utilidades que caracteriza o empate moral.

A função `gerar_relatorio_utilidades` (Listagem 6) consolida os três cálculos em um dicionário utilizado pelo Agente Ético para tomar a decisão e pelo Agente HITL para análise do dilema:

Listagem 6 – Função gerar_relatorio_utilidades

```
def gerar_relatorio_utilidades(conflito):
    if conflito.get("tipo") == "pedestre_veiculo_atras":
        return gerar_relatorio_dilema(conflito)

    u_frear   = calcular_utilidade_frear(conflito)
    u_desviar = calcular_utilidade_desviar(conflito)
    u_seguir  = calcular_utilidade_seguir(conflito)
    acao, utilidade, dilema = selecionar_melhor_acao(conflito)

    return HashableDict({
        "utilidades":           HashableDict({"frear":   u_frear,
                                              "desviar": u_desviar,
                                              "seguir":  u_seguir}),
        "acao_recomendada":     acao,
        "utilidade_selecionada": utilidade,
        "eh_dilema":            dilema,
    })
```

Fonte: Autoria própria (2026).

O campo `eh_dilema` é calculado internamente por `selecionar_melhor_acao`, que aplica o critério formal apresentado na Equação 2. Esse campo é o que o Agente Ético consulta para decidir entre enviar uma recomendação ao VA ou escalar o conflito ao HITL.

4.4 Ambiente Urbano

O `AmbienteUrbano` é implementado como uma classe que herda de `Environment` do MASPY, representando o cruzamento de trânsito onde o VA opera. Conforme especificado no PEAS do seção 3.2, o ambiente é responsável por prover sete percepções sobre o estado do cruzamento e executar duas ações físicas comandadas pelos agentes.

4.4.1 Implementação das Percepções

As percepções são implementadas através do decorador `@prp` do MASPY, que expõe propriedades da classe como percepções automaticamente disponíveis aos agentes conectados. Cada percepção retorna o valor de um atributo armazenado internamente no ambiente.

Listagem 7 – Implementação das percepções no AmbienteUrbano

```
class AmbienteUrbano(Environment):
    def __init__(self, env_name):
        super().__init__(env_name)
        self._semaforo_aberto = 1
        self._pedestre_detectado = 1
        self._obstaculo = 0
        self._veiculo_detectado = 0
        self._veiculo_atras = 0
        self._veiculo_lateral = 0
        self._zona_escolar = 0

    @prp
    def semaforo_aberto(self):
        return self._semaforo_aberto

    @prp
    def zona_escolar(self):
        return self._zona_escolar
    # ... demais percepcoes seguem o mesmo padrao
```

Fonte: Autoria própria (2026).

4.4.2 Implementação das Ações

As ações executáveis pelo VA são implementadas através do decorador `@agt_actuation` do MASPY, sendo que cada método decorado recebe o nome do agente soli-

citante como primeiro parâmetro, permitindo rastrear qual agente está executando determinada ação no ambiente. A ação `executar_movimento` simula a execução física da manobra escolhida pelo VA, registrando-a através de *log* e retornando um dicionário de confirmação para o agente solicitante.

Listagem 8 – Implementação da ação de movimento físico

```
1 @agt_actuation
2 def executar_movimento(self, agent_name, acao):
3     self.print(f"[AMBIENTE] {agent_name} executando: {acao}")
4     return {"status": "executado", "acao": acao}
```

Fonte: Autoria própria (2026).

A segunda ação disponibilizada pelo ambiente, `registrar_evento`, armazena eventos para análise posterior, permitindo que o VA registre suas decisões e ações executadas. Esses registros podem ser posteriormente analisados para validação do comportamento ético do sistema.

4.4.3 Configuração de Cenários

A configuração de diferentes cenários de teste é realizada através da alteração direta dos valores das percepções em `environment.py`, antes de iniciar o ciclo de execução do sistema. Essa abordagem permite testar diversas situações de conflito ético sem modificar o código dos agentes, facilitando a validação experimental descrita no Capítulo 5.

Listagem 9 – Configuração do cenário base (pedestre no cruzamento)

```
1 # Cenario base: Pedestre atravessando com semaforo verde
2 ambiente._semaforo_aberto = 1
3 ambiente._pedestre_detectado = 1
4 ambiente._obstaculo = 0
5 ambiente._veiculo_detectado = 0
6 ambiente._veiculo_atras = 0
7 ambiente._veiculo_lateral = 0
8 ambiente._zona_escolar = 0
```

Fonte: Autoria própria (2026).

O código acima configura o cenário base descrito por Alves, Dennis e Fisher (2021), no qual há um pedestre atravessando (`ambiente._pedestre_detectado = 1`) a faixa enquanto o semáforo está verde (`ambiente._semaforo_aberto = 1`) para o VA.

4.5 Agente Ético

O `AgenteEtico` contém seis planos conforme o diagrama UML da seção 3.6. A implementação armazena o dicionário MODD como atributo de instância, permitindo consulta di-

reta durante o raciocínio ético. As crenças e objetivos iniciais são adicionados no construtor conforme apresentado no Listagem 10, sendo `status_etico` utilizada como contexto dos planos e `monitorar_ambiente` como gatilho inicial.

Listagem 10 – Inicialização do Agente Ético com MODD

```
class AgenteEtico(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
        self.add(Belief("status_etico", "monitorando"))
        self.add(Goal("monitorar_ambiente"))

        self.modd = {
            "pedestre": {
                "resolver_autonomo": True,
                "threshold_minimo": 1.0,
                "autoridade": "AgenteEtico"
            },
            "pedestre_zona_escolar": {
                "resolver_autonomo": True,
                "threshold_minimo": 4.0,
                "autoridade": "HITL"
            },
            "obstaculo": {
                "resolver_autonomo": True,
                "threshold_minimo": 0.5,
                "autoridade": "AgenteEtico"
            }
        }
```

Fonte: Autoria própria (2026).

Cada entrada do dicionário contém três campos: `resolver_autonomo` indica se aquele tipo de conflito admite resolução autônoma; `threshold_minimo` define o *threshold* — limiar mínimo de diferença de utilidade exigido antes de autorizar uma decisão autônoma — conforme modelado na seção 3.3; e `autoridade` indica qual agente deve resolver caso o limiar não seja atingido.

O primeiro plano (Listagem 11) lê as percepções do ambiente e classifica o tipo de conflito presente. A classificação é feita através de uma estrutura condicional que prioriza dilemas éticos (pedestre com veículo traseiro) e contextos de alta vigilância (zona escolar), tratando conflitos simples por último. Para cada tipo detectado, o código chama a função correspondente de `utilities.py`, que retorna uma instância de `ConflitoGaltung` com os três vértices preenchidos.

Listagem 11 – Detecção e classificação de conflitos

```
@pl(gain, Goal("monitorar_ambiente"), Belief("status_etico", Any))
def perceber_e_classificar_conflito(self, src, status):
    perc_pedestre = self.get(Belief("pedestre_detectado", Any, "Cruzamento"))
    perc_v_atras = self.get(Belief("veiculo_atras", Any, "Cruzamento"))
    perc_zona_escolar = self.get(Belief("zona_escolar", Any, "Cruzamento"))
    # ... demais percepcoes

    if pedestre and v_atras:
        conflito = criar_conflito_dilema(semaforo, pedestre, obstaculo,
                                         veiculo, v_atras, v_lateral)
    elif pedestre and zona_escolar:
        conflito = criar_conflito_pedestre_zona_escolar(...)
    elif pedestre:
        conflito = criar_conflito_pedestre(...)
    elif obstaculo:
        conflito = criar_conflito_obstaculo(...)

    if conflito_detectado:
        self.add(Belief("conflito_ativo", conflito))
        self.add(Goal("avaliar_conflito", conflito))
```

Fonte: Autoria própria (2026).

O segundo plano (Listagem 12) calcula as utilidades das três ações possíveis através da função `gerar_relatorio_utilidades`, que retorna um dicionário contendo as utilidades de frear, desviar e seguir, além das duas melhores ações identificadas. O código então verifica se há dilema comparando as duas maiores utilidades com margem de tolerância de 0,01. Caso não haja dilema, calcula a diferença entre as utilidades e invoca `modd_pode_resolver` para verificar se essa diferença atinge o *threshold* definido para aquele tipo de conflito. Se sim, prossegue para envio da recomendação ao VA e, caso contrário, escala ao HITL mesmo havendo ação superior.

Listagem 12 – Cálculo de utilidades com verificação MODD

```
@pl(gain, Goal("avaliar_conflito", Any), Belief("conflito_ativo", Any))
def calcular_utilidades_e_decidir(self, src, conflito, c):
    relatorio = gerar_relatorio_utilidades(conflito)
    eh_dilema = abs(relatorio["u_melhor"] - relatorio["u_segunda"]) < 0.01

    if not eh_dilema:
        utilidades_vals = sorted(relatorio["utilidades"].values(), reverse=True)
        diferenca = utilidades_vals[0] - utilidades_vals[1]
        tipo = conflito.get("tipo")

        if self.modd_pode_resolver(tipo, diferenca):
            recomendacao = HashableDict({
                "acao": relatorio["acao_melhor"],
                "utilidade": relatorio["u_melhor"],
                "conflito": conflito
            })
            self.add(Goal("enviar_recomendacao", recomendacao))
        else:
            self.add(Goal("escalar_para_hitl", ((conflito, relatorio),)))
    else:
        self.add(Goal("escalar_para_hitl", ((conflito, relatorio),)))
```

Fonte: Autoria própria (2026).

A integração entre cálculo de utilidades e MODD ocorre através da chamada a `modd_pode_resolver`, método detalhado na subseção 4.5.1. Esse método consulta o dicionário MODD usando o tipo do conflito como chave e verifica se a diferença de utilidade é maior ou igual ao *threshold* definido.

Os quatro planos restantes implementam as etapas finais do raciocínio ético:

- O terceiro plano monta a estrutura ECP e a envia ao VA usando a performativa `tell` do MASPY.
- O quarto plano envia o objetivo `investigar_dilema` ao HITL através de `achieve`, incluindo o conflito e o relatório de utilidades.
- O quinto plano responde a solicitações de alternativa vindas do VA quando há incoerência entre recomendação e percepções locais.
- O sexto plano recebe decisões do HITL e as repassa ao VA, incluindo o campo `decidido_por` no ECP para que o VA identifique a origem da instrução.

4.5.1 Implementação do MODD

Três métodos auxiliares encapsulam o acesso ao dicionário MODD, conforme apresentado no Listagem 13. O primeiro método, `consultar_modd`, implementa busca direta no dicionário usando o tipo do conflito como chave. O segundo método, `modd_pode_resolver`,

combina duas verificações: primeiro verifica se o campo `resolver_autonomo` está habilitado, depois verifica se a diferença de utilidade atinge o *threshold*. O terceiro método, `modd_explicar`, gera string formatada para registro em *log*.

Listagem 13 – Métodos de acesso ao MODD

```
def consultar_modd(self, tipo):
    return self.modd.get(tipo)

def modd_pode_resolver(self, tipo, diferenca):
    regra = self.consultar_modd(tipo)
    return regra["resolver_autonomo"] and diferenca >= regra["threshold_minimo"]

def modd_explicar(self, tipo, diferenca):
    regra = self.consultar_modd(tipo)
    pode = self.modd_pode_resolver(tipo, diferenca)
    modo = "(resolve)" if pode else "(transfer)"
    return (
        f"MODD {modo} | tipo={tipo} | diff={diferenca:.2f} "
        f"threshold={regra['threshold_minimo']} | "
        f"autoridade={regra['autoridade']}"
    )
```

Fonte: Autoria própria (2026).

A dupla verificação em `modd_pode_resolver` permite desabilitar resolução autônoma para certos tipos de conflito independentemente da diferença de utilidade, fornecendo camada adicional de controle sobre a autonomia do sistema.

4.6 Agente VA

O `AgenteVA` contém sete planos conforme o diagrama UML da seção 3.6. A implementação inicializa três crenças relacionadas ao estado de condução — `conduzindo`, `velocidade` e `localizacao` — sendo essas atualizadas conforme o veículo navega pelo ambiente, conforme apresentado no Listagem 14.

Listagem 14 – Inicialização do Agente VA

```
class AgenteVA(Agent):
    def __init__(self, agt_name):
        super().__init__(agt_name)
        self.add(Belief("conduzindo", 1))
        self.add(Belief("velocidade", 60))
        self.add(Belief("localizacao", "na_via"))
        self.add(Goal("conduzir"))
```

Fonte: Autoria própria (2026).

O segundo plano (Listagem 15) é ativado quando uma crença `sugestao_manobra_segura` é adicionada ao agente, sendo essa crença criada automaticamente pelo

MASPY quando uma mensagem com performativa `tell` chega do Agente Ético. O código extrai a ação recomendada e a origem da mensagem, lê as percepções locais do ambiente e verifica a coerência através de regras booleanas específicas para cada tipo de ação. Cada regra combina múltiplas percepções em uma expressão lógica que determina se aquela ação é adequada ao contexto atual.

Listagem 15 – Avaliação de coerência de recomendações

```
@pl(gain, Belief("sugestao_manobra_segura", Any), Belief("conduzindo", Any))
def avaliar_proposta_etica(self, src, recomendacao, status):
    acao = recomendacao.get("acao", "frear")
    origem = recomendacao.get("origem", "AgenteEtico")

    if origem == "HITL":
        self.add(Goal("aceitar_sugestao", ((acao, recomendacao),)))
        return

    p_semaforo = self.get(Belief("semaforo_aberto", Any, "Cruzamento"))
    p_pedestre = self.get(Belief("pedestre_detectado", Any, "Cruzamento"))
    p_v_lateral = self.get(Belief("veiculo_lateral", Any, "Cruzamento"))
    # ... demais percepcoes

    if acao == "frear":
        q3 = bool(pedestre or veiculo or obstaculo or not semaforo)
    elif acao == "desviar":
        q3 = bool(obstaculo and not pedestre and not veiculo and not v_lateral)
    elif acao == "seguir":
        q3 = bool(semaforo and not pedestre and not veiculo and not obstaculo)

    if q3:
        self.add(Goal("aceitar_sugestao", ((acao, recomendacao),)))
    else:
        situacao_perigosa = bool(pedestre or veiculo or v_lateral)
        if situacao_perigosa:
            self.add(Goal("pedir_detalhes", conflito))
        else:
            self.add(Goal("decisao_autonoma", conflito))
```

Fonte: Autoria própria (2026).

A regra de coerência para `desviar` é particularmente importante, pois inclui a verificação de `veiculo_lateral`. Essa verificação detecta o cenário onde há obstáculo à frente mas também veículo ao lado, tornando a manobra de desvio perigosa. Nesse caso, `q3` resulta falso e o código verifica se há situação perigosa — o que é verdade pela presença do veículo lateral — escalando para solicitação de alternativa ao Agente Ético.

Os cinco planos restantes coordenam a execução de manobras e comunicação com o Agente Ético:

- O terceiro plano invoca `self.call` para executar a ação `executar_movimento` no ambiente, passando o nome da manobra como parâmetro.

- O quarto plano monta um ECP vazio (ação desconhecida) e o envia ao Agente Ético através de `achieve`, solicitando alternativa.
- O quinto plano processa a resposta recebida, executando a ação alternativa sem nova verificação de coerência.
- O sexto plano decide autonomamente quando `q3` é falso mas não há perigo imediato.
- O sétimo plano ativa modo seguro quando um dilema é escalado ao HITL, reduzindo a velocidade a zero e aguardando decisão humana antes de prosseguir.

4.7 Agente HITL

O `AgenteHITL` contém três planos conforme o diagrama UML da seção 3.6. A implementação utiliza atributos do próprio agente — `_log_count` e `_logs` — para armazenar eventos internamente, sendo esses atributos acessíveis apenas dentro da classe e não expostos como crenças do agente, conforme apresentado no Listagem 16.

Listagem 16 – Inicialização do Agente HITL

```
1 class AgenteHITL(Agent):
2     def __init__(self, agt_name):
3         super().__init__(agt_name)
4         self.add(Belief("status", "observando"))
5         self.add(Belief("total_logs", 0))
6         self._log_count = 0
7         self._logs = []
```

Fonte: Autoria própria (2026).

O segundo plano (Listagem 17) extrai as informações do dilema recebido — conflito e relatório de utilidades — e invoca o método `_tomar_decisao_humana`. Esse método implementa estratégia conservadora priorizando proteção de vidas humanas, retornando sempre a ação de menor risco moral. Por exemplo, em dilemas envolvendo pedestre, a decisão é sempre frear. A instrução é então enviada ao Agente Ético através de `achieve`, incluindo o campo `decidido_por` marcado como "HITL"para que o VA identifique a origem da decisão.

Listagem 17 – Investigação de dilemas e decisão humana

```
@pl(gain, Goal("investigar_dilema", Any))
def tomar_decisao_humana(self, src, info):
    conflito = info.get("conflito", {})
    relatorio = info.get("relatorio", {})
    utilidades = relatorio.get("utilidades", {})

    self.print("DILEMA ETICO RECEBIDO - Investigacao iniciada")
    self.print(f"Utilidades: {utilidades}")

    decisao = self._tomar_decisao_humana(conflito, utilidades)

    instrucao = HashableDict({
        "acao": decisao,
        "conflito": conflito,
        "decidido_por": "HITL"
    })
    self.send("AgenteEtico", achieve,
              Goal("executar_instrucao_hitl", instrucao), "AgentComm")
```

Fonte: Autoria própria (2026).

Os três planos do HITL cobrem situações distintas de supervisão:

- O plano `auditar_evento` registra eventos recebidos do VA e do Agente Ético, emitindo alerta quando o VA decide autonomamente sem seguir recomendação ética.
- O plano `tomar_decisao_humana` analisa o dilema ético recebido e envia instrução ao Agente Ético, conforme apresentado no Listagem 17.
- O plano `registrar_decisao_autonoma` registra situações em que o VA descartou a recomendação do Agente Ético para análise posterior.

Em implementação real conectada a operador humano, o método `_tomar_decisao_humana` seria substituído por interface gráfica apresentando o conflito e solicitando decisão através de *input* interativo. A estrutura atual simula esse comportamento aplicando regras determinísticas, mas mantém a arquitetura preparada para substituição futura.

5 EXPERIMENTOS E VALIDAÇÃO

Este capítulo apresenta os experimentos realizados para validar o sistema multiagente ético proposto. São detalhados três cenários principais, cada um representando um caminho distinto de tomada de decisão ética: resolução autônoma pelo Agente Ético, transferência para HITL e identificação de dilema ético. Os demais cenários são sumarizados em uma tabela comparativa.

5.1 Metodologia de Avaliação

Os experimentos foram realizados através de simulações do sistema multiagente em cenários de cruzamento urbano. Cada cenário foi configurado através das sete percepções do ambiente (semáforo, pedestre, obstáculos e veículos) e avaliado segundo quatro critérios:

- **Decisão correta:** a ação escolhida respeita as diretrizes do *German Ethics Code* aplicáveis ao cenário (EG 1, EG 5, EG 7, EG 8);
- **Agente responsável:** quem tomou a decisão final — o próprio sistema (resolução autônoma), o Agente HITL (transferência por limiar MODD insuficiente) ou o HITL por dilema ético genuíno;
- **Respeito aos limiares:** os valores θ definidos no MODD foram aplicados corretamente conforme o tipo de conflito (seção 3.3);
- **Cálculos verificáveis:** os valores de utilidade calculados para cada ação podem ser conferidos nos logs gerados pelo sistema.

5.2 Cenário 1: Pedestre no Cruzamento

O primeiro cenário simula a situação onde o VA detecta um pedestre atravessando a faixa enquanto o semáforo está verde para o veículo. O conflito é classificado como `pedestre`, sendo que o MODD define $\theta = 1{,}0$ com resolução autônoma habilitada. A configuração das percepções para esse cenário é apresentada no Listagem 18.

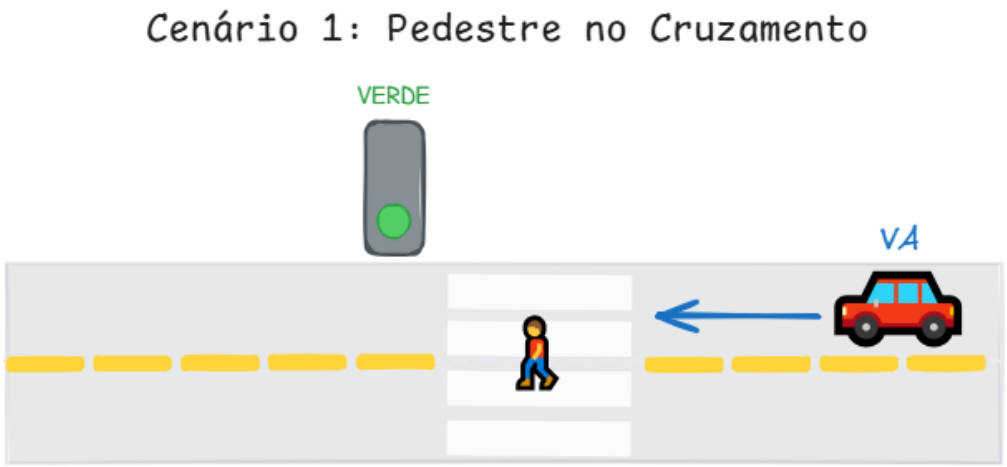
Listagem 18 – Configuração das percepções — Cenário 1

```
1 # Cenario 1: Pedestre no cruzamento (semaforo verde)
2 ambiente._semaforo_aberto = 1
3 ambiente._pedestre_detectado = 1
4 ambiente._obstaculo = 0
5 ambiente._veiculo_detectado = 0
6 ambiente._veiculo_atras = 0
7 ambiente._veiculo_lateral = 0
8 ambiente._zona_escolar = 0
```

Fonte: Autoria própria (2026).

A Figura 12 ilustra o cenário descrito, mostrando o veículo autônomo aproximando-se do cruzamento com o semáforo verde enquanto o pedestre atravessa a faixa.

Figura 12 – Ilustração — Cenário 1: Pedestre no Cruzamento



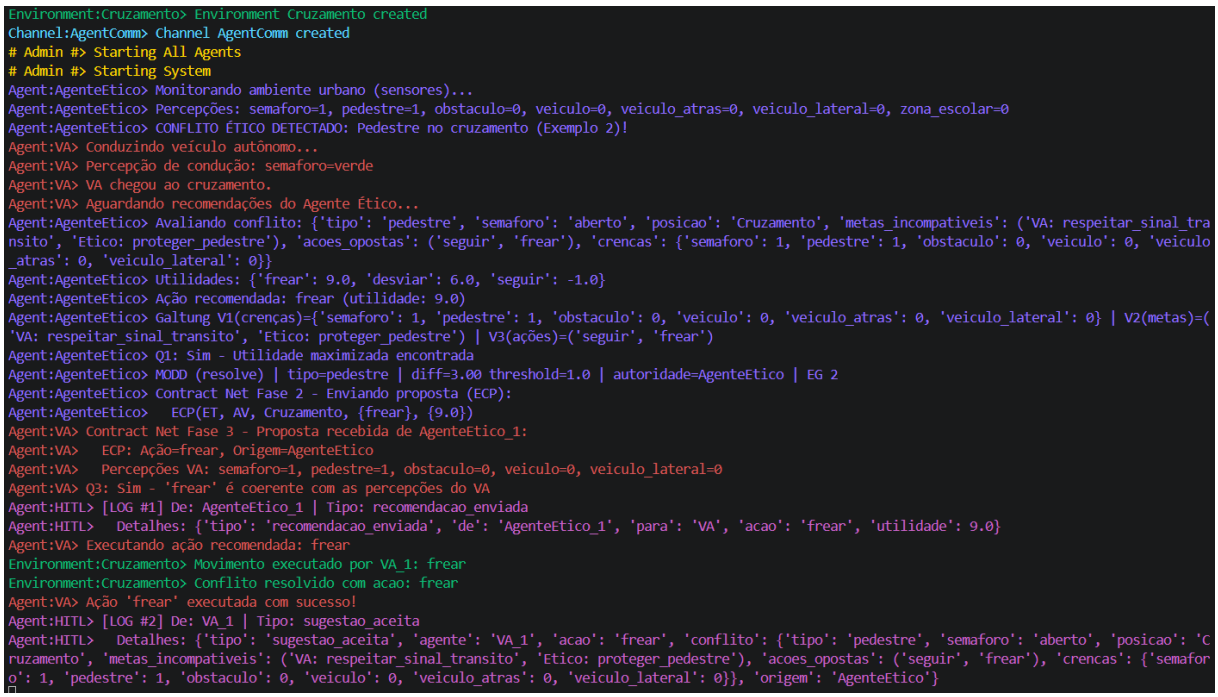
Fonte: Autoria própria.

O Agente Ético calcula as utilidades para as três ações disponíveis. Frear obtém $u = 9{,}0$, pois preserva a vida do pedestre (EG 7) e segue a regra de prioridade (EG 5). Desviar obtém $u = 6{,}0$, preservando a vida mas violando a faixa de pedestres. Seguir obtém $u = -1{,}0$, representando colisão com o pedestre — violação grave de EG 1 e EG 7.

A ação com maior utilidade é frear. O sistema verifica se há dilema calculando $|\Delta u| = |9{,}0 - 6{,}0| = 3{,}0$, valor maior que o limiar de dilema $0{,}01$. Como não há dilema, o sistema consulta o MODD para verificar se a decisão pode ser autônoma. A diferença de utilidade supera o limiar ($3{,}0 \geq 1{,}0$), e o MODD autoriza resolução autônoma pelo Agente Ético. O Ético então envia a proposta ao VA.

O VA recebe a recomendação de frear e verifica se a ação é coerente com suas percepções locais. Como há pedestre presente e frear é seguro, a verificação Q3 resulta verdadeira. O VA aceita a proposta e executa a manobra sem intervenção humana. O Figura 13 apresenta a saída do terminal durante a execução.

Figura 13 – Saída do terminal — Cenário 1: Pedestre no Cruzamento



Fonte: Autoria própria.

O resultado esperado é que o VA execute a ação de frear antes da faixa de pedestres. O relatório de utilidades registra os valores $u_{\text{frear}} = 9{,}0$, $u_{\text{desviar}} = 6{,}0$, $u_{\text{seguir}} = -1{,}0$ e indica resolução autônoma.

5.3 Cenário 2: Pedestre em Zona Escolar

O segundo cenário simula a situação onde o VA detecta um pedestre em zona escolar. O conflito é classificado como pedestre em zona escolar, e o MODD define limiar $\theta = 4{,}0$ com autoridade transferida ao HITL, tornando obrigatória a supervisão humana. A configuração difere do cenário anterior apenas pela ativação da zona escolar, conforme o Listagem 19.

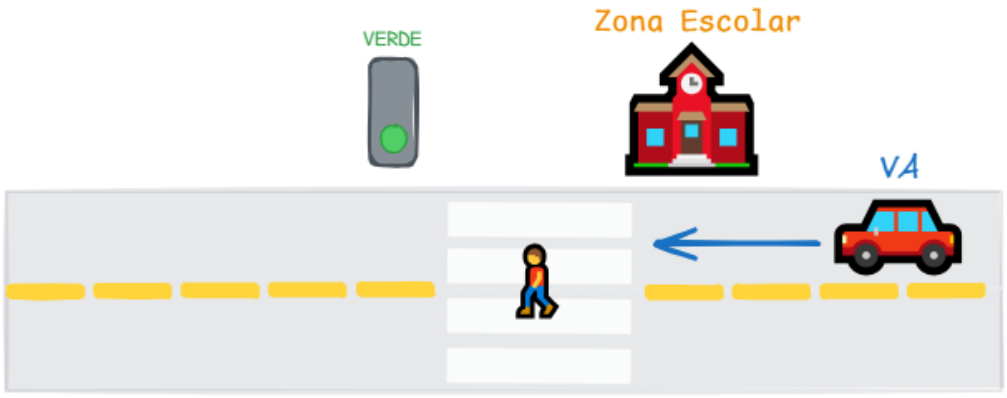
Listagem 19 – Configuração das percepções — Cenário 2

```
# Cenario 2: Pedestre em zona escolar (threshold elevado)
ambiente._semaforo_aberto = 1
ambiente._pedestre_detectado = 1
ambiente._obstaculo = 0
ambiente._veiculo_detectado = 0
ambiente._veiculo_atras = 0
ambiente._veiculo_lateral = 0
ambiente._zona_escolar = 1
```

Fonte: Autoria própria (2026).

A Figura 14 ilustra o cenário descrito, destacando a presença da sinalização de zona escolar no trecho onde o pedestre atravessa.

Figura 14 – Ilustração — Cenário 2: Pedestre em Zona Escolar

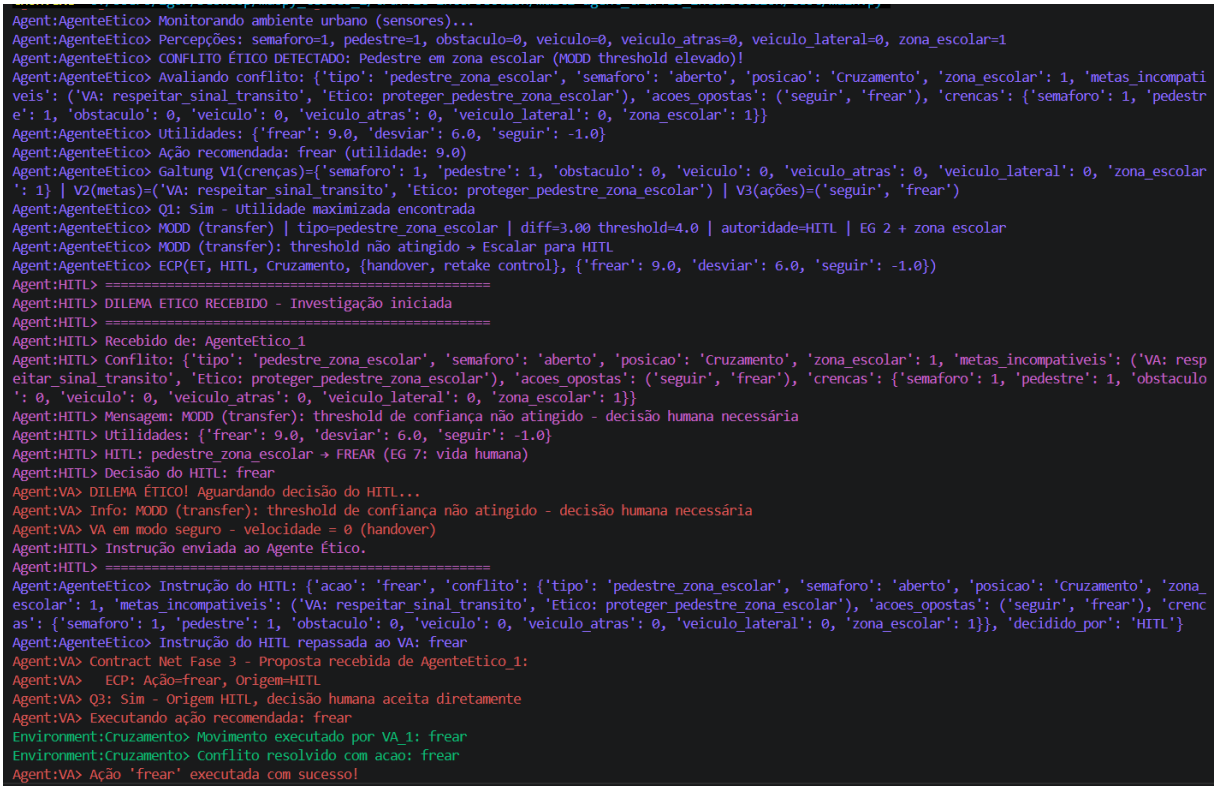


Fonte: Autoria própria.

As utilidades calculadas são as mesmas do Cenário 1: $u_{\text{frear}} = 9{,}0$, $u_{\text{desviar}} = 6{,}0$ e $u_{\text{seguir}} = -1{,}0$. O sistema verifica se há dilema calculando $|\Delta u| = 3{,}0$, valor maior que $0{,}01$, indicando que não há dilema. Porém, a diferença de utilidade é inferior ao limiar mais restritivo definido para zona escolar: $3{,}0 < \theta = 4{,}0$. Além disso, o MODD aponta a autoridade para o HITL.

Mesmo havendo uma ação claramente superior, o sistema não pode resolver autonomamente. O Agente Ético transfere o conflito completo ao Agente HITL, incluindo o relatório de utilidades. O operador humano recebe as informações, delibera e emite a decisão de frear, que é retransmitida ao VA. O VA aceita a decisão do HITL diretamente sem verificação de coerência, executando a manobra. O Figura 15 apresenta a saída do terminal.

Figura 15 – Saída do terminal — Cenário 2: Pedestre em Zona Escolar



Fonte: Autoria própria.

O resultado esperado é que o HITL seja acionado e a decisão final seja frear. O relatório registra a transferência ao HITL motivada tanto pelo limiar não atingido ($|\Delta u| < \theta$) quanto pela configuração de autoridade no MODD.

5.4 Cenário 3: Dilema Ético

O terceiro cenário simula uma situação onde o VA detecta simultaneamente um pedestre atravessando e um veículo traseiro muito próximo, configurando uma situação onde qualquer ação disponível implica risco a uma das partes. Esse cenário exercita o critério de detecção de dilema moral: $|u_i - u_j| < 0{,}01$ para o par de ações de maior utilidade. A configuração é apresentada no Listagem 20.

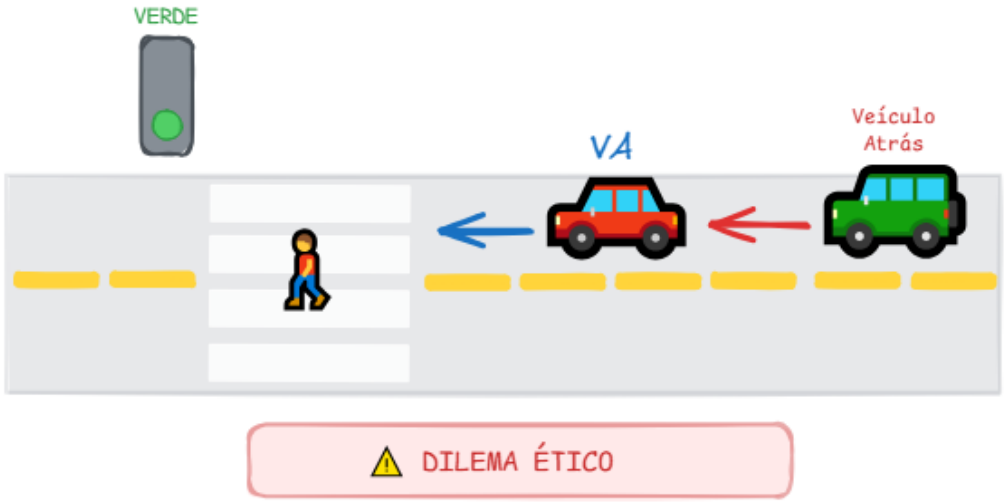
Listagem 20 – Configuração das percepções — Cenário 3

```
# Cenario 3: Dilema etico (pedestre + veiculo traseiro)
ambiente._semaforo_aberto = 1
ambiente._pedestre_detectado = 1
ambiente._obstaculo = 0
ambiente._veiculo_detectado = 0
ambiente._veiculo_atras = 1
ambiente._veiculo_lateral = 0
ambiente._zona_escolar = 0
```

Fonte: Autoria própria (2026).

A Figura 16 ilustra o cenário descrito, mostrando o pedestre atravessando a faixa e o veículo traseiro posicionado muito próximo ao VA.

Figura 16 – Ilustração — Cenário 3: Dilema Ético



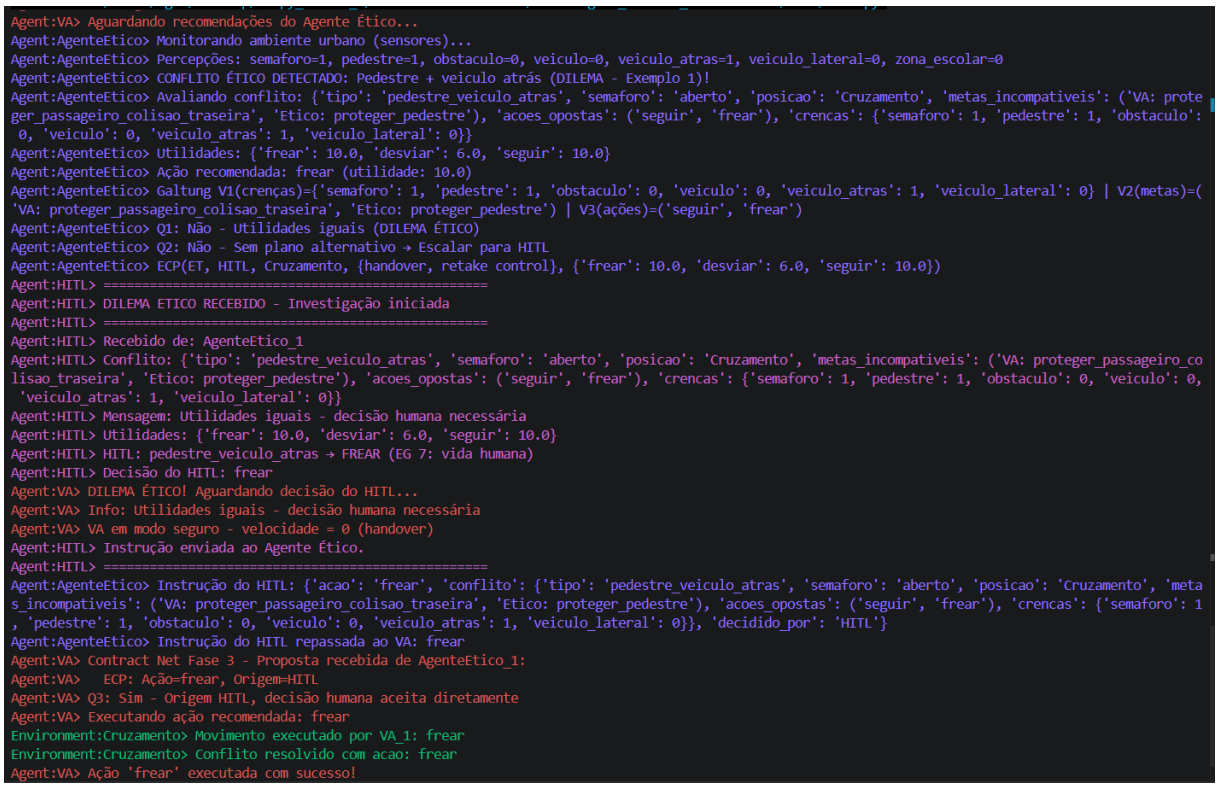
Fonte: Autoria própria.

O Agente Ético identifica o conflito como dilema e constrói o Triângulo de Galtung mapeando a contradição entre as metas "proteger pedestre"versus "proteger passageiros do veículo traseiro". As utilidades calculadas são $u_{\text{frear}} = 10{,}0$ — preservando o pedestre mas colocando em risco os passageiros traseiros — e $u_{\text{seguir}} = 10{,}0$ — preservando os passageiros mas colocando o pedestre em risco.

A diferença de utilidade é $|\Delta u| = |10{,}0 - 10{,}0| = 0{,}0 < 0{,}01$, satisfazendo o critério de dilema. O sistema reconhece que não há base racional para preferir autonomamente uma ação em detrimento da outra. Em conformidade com EG 8 do *German Ethics Code* — que estabelece que dilemas morais genuínos devem ser tratados por humanos —, o Agente Ético encaminha imediatamente o conflito ao HITL, independentemente das configurações do MODD.

O operador humano recebe as informações do dilema, incluindo ambas as utilidades equivalentes e as metas incompatíveis, delibera e decide frear, preservando o pedestre e assumindo o risco de colisão traseira. Essa decisão é retransmitida ao VA, que a executa sem verificação de coerência. O Figura 17 apresenta a saída do terminal.

Figura 17 – Saída do terminal — Cenário 3: Dilema Ético



Fonte: Autoria própria.

O resultado esperado é que o HITL seja acionado com a indicação explícita de dilema ético. O relatório registra a escalada obrigatória motivada pelo critério $|\Delta u| < 0{,}01$ e os valores $u_{\text{frear}} = 10{,}0$, $u_{\text{seguir}} = 10{,}0$.

5.5 Demais Cenários

Os três cenários restantes são apresentados de forma consolidada no Quadro 6. Todos foram executados com resolução autônoma pelo Agente Ético, exercitando os tipos de conflito veículo e obstáculo. A tabela apresenta, para cada cenário, as percepções que definem a situação, os valores de utilidade calculados para as três ações (frear, desviar e seguir), a comparação entre a diferença de utilidade e o limiar MODD, e o resultado final da negociação entre os agentes.

Quadro 6 – Resumo dos Cenários 4, 5 e 6

Cenário	**Percepções-chave**	**Utilidades (f/d/s)***	$\|\Delta u\|$ **vs** θ	**Desfecho**
C4: Veículo frontal	`veiculo=1`, `semaforo=1`	7,0 / 5,0 / 4,0	$2{,}0 \geq 1{,}0$	Resolve autônomo; VA aceita frear (Q3=Sim)
C5: Obstáculo simples	`obstaculo=1`, `semaforo=1`	6,0 / 6,5 / $-1{,}0$	$0{,}5 \geq 0{,}5$	Resolve autônomo; VA aceita desviar (Q3=Sim)
C6: Obstáculo + v. lateral	`obstaculo=1`, `v_lateral=1`, `semaforo=1`	6,0 / 6,5 / $-1{,}0$	$0{,}5 \geq 0{,}5$	Resolve autônomo; VA rejeita desviar (Q3=Não), solicita alternativa (Q4=Sim), executa frear

Fonte: Autoria própria.

*f/d/s = frear/desviar/seguir

O Cenário 6 merece destaque especial, pois demonstra a autonomia do VA dentro do protocolo Contract Net. Mesmo após o Agente Ético propor desviar como ação ótima, o VA identifica que desviar colidiria com o veículo lateral e rejeita a proposta através da verificação Q3. O VA solicita uma proposta alternativa através de Q4, e o Agente Ético recalcula indicando frear, ação que é então aceita e executada. Esse comportamento valida o mecanismo de verificação de coerência implementado no VA, garantindo que recomendações éticas sejam confrontadas com a realidade percebida localmente pelo veículo.

5.6 Discussão dos Resultados

Os seis cenários validam os três caminhos de resolução previstos na modelagem. No primeiro caminho — resolução autônoma —, presente nos Cenários 1, 4, 5 e 6, o Agente Ético calcula as utilidades, verifica que não há dilema ($|\Delta u| \geq 0{,}01$) e que o MODD autoriza a decisão autônoma ($|\Delta u| \geq \theta$). A proposta é enviada ao VA, que pode aceitá-la diretamente (Q3=Sim) ou rejeitá-la e solicitar alternativa (Q4=Sim), como ocorreu no Cenário 6.

No segundo caminho — transferência para HITL por configuração —, presente no Cenário 2, mesmo sem dilema, o MODD exige supervisão humana. O sistema aciona o Agente HITL em conformidade com EG 19 do *German Ethics Code* e aguarda a deliberação do operador. Esse mecanismo garante que contextos de alta vigilância — como zona escolar — recebam supervisão humana adicional independentemente da clareza da melhor ação.

No terceiro caminho — transferência para HITL por dilema —, presente no Cenário 3, quando as ações de maior utilidade são indistinguíveis ($|\Delta u| < 0{,}01$), o sistema reconhece o dilema ético e escala obrigatoriamente para o HITL, independentemente do MODD, em conformidade com EG 8. Esse mecanismo previne que o sistema tome decisões arbitrárias em situações moralmente complexas onde não há consenso ético claro.

Os experimentos foram realizados em ambiente simulado, com percepções fornecidas diretamente como valores numéricos, sem integração com sensores físicos ou outros VAs em rede. Os valores de utilidade e os limiares θ foram definidos manualmente com base no raciocínio ético teórico apresentado por Alves, Dennis e Fisher (2021). Em um sistema real, esses parâmetros demandariam calibração empírica através de experimentos com operadores humanos reais.

O sistema demonstra transparência completa das decisões através dos logs gerados, separação clara entre raciocínio ético (Agente Ético) e execução (VA), e capacidade de escalada supervisionada (HITL) para situações de alta complexidade moral. Esses resultados alinham-se com as diretrizes do Código de Ética Alemão apresentadas por Ethics Commission (2017) e com o conceito de *Governor Module* proposto por Alves, Dennis e Fisher (2021).

6 CONCLUSÃO

Este trabalho teve como objetivo geral modelar e implementar um sistema multiagente capaz de resolver conflitos éticos envolvendo veículos autônomos em cruzamentos de trânsito urbanos, seguindo como base a arquitetura de agente ético proposta por Alves, Dennis e Fisher (2021).

Todos os objetivos específicos estabelecidos foram atingidos. O primeiro, formalizar conflitos éticos através do Triângulo de Galtung, foi alcançado com a implementação de estruturas de dados que representam os três vértices do conflito e são transmitidas entre os agentes ao longo de todo o processo de decisão. O segundo, implementar MODDs com limites de autonomia por tipo de conflito, foi alcançado com a definição de limiares diferenciados que controlam quando o sistema resolve autonomamente e quando transfere ao HITL, conforme demonstrado nos Cenários 1 e 2. O terceiro, adaptar o protocolo Contract Net para negociação entre agentes, foi alcançado com a implementação das três fases de negociação e dos mecanismos de verificação de coerência no Agente VA. O quarto, desenvolver funções de utilidade ética baseadas no *German Ethics Code*, foi alcançado com a definição de constantes fundamentadas nas diretrizes EG 1, EG 2, EG 5 e EG 7 e com o critério formal de identificação de dilemas éticos. O quinto, validar o sistema em cenários representativos, foi alcançado com três cenários que cobrem os caminhos distintos de decisão da arquitetura, incluindo resolução autônoma, transferência por limite de autonomia contextual e reconhecimento de dilema ético genuíno.

As principais contribuições deste trabalho são: (1) a extensão da arquitetura de agente ético proposta por Alves, Dennis e Fisher (2021) com a implementação completa do MODD baseado em *thresholds* diferenciados por tipo de conflito, (2) a formalização de conflitos éticos através do Triângulo de Galtung implementado como estruturas de dados manipuláveis por agentes BDI, (3) a implementação de um protocolo de negociação ética baseado no Contract Net que permite ao VA questionar e solicitar alternativas às recomendações do Agente Ético, e (4) a demonstração experimental de um sistema multiagente ético funcional utilizando o *framework* MASPY, validando sua aplicabilidade em cenários de veículos autônomos.

Algumas limitações foram identificadas durante o desenvolvimento do trabalho. Os experimentos foram realizados em ambiente simulado, com percepções fornecidas diretamente como valores numéricos configurados em `main.py`, sem integração com sensores físicos reais ou simuladores de tráfego como SUMO ou CARLA. Os valores de utilidade e os *thresholds* θ foram definidos manualmente com base em raciocínio ético teórico, sendo que em um sistema real esses parâmetros demandariam calibração empírica através de experimentos com operadores humanos. Adicionalmente, o Agente HITL foi implementado com decisões simuladas através de regras determinísticas, não havendo interface gráfica para interação com operador humano real. Por fim, o sistema foi testado apenas com um único VA, não sendo avaliado o comportamento em cenários com múltiplos veículos autônomos interagindo simultaneamente.

Como trabalhos futuros, sugere-se: (1) integração do sistema com simuladores de tráfego realistas como SUMO ou CARLA, permitindo validação em cenários mais complexos

com múltiplos veículos e pedestres, (2) desenvolvimento de interface gráfica para o Agente HITL, possibilitando experimentos com operadores humanos reais e calibração empírica dos *thresholds* MODD, (3) extensão do sistema para cenários com múltiplos VAs, investigando como agentes éticos distribuídos podem coordenar decisões coletivas, (4) aplicação de técnicas de aprendizado de máquina para refinamento das funções de utilidade ética com base em decisões humanas observadas, (5) expansão do conjunto de conflitos éticos modelados, incluindo situações como mudança de faixa em rodovias, ultrapassagens e conversões em cruzamentos complexos, e (6) investigação de mecanismos de explicabilidade que permitam ao VA justificar suas decisões éticas para passageiros e autoridades regulatórias.

Este trabalho demonstra que é possível implementar sistemas multiagentes capazes de tomar decisões éticas em cenários de veículos autônomos, respeitando diretrizes morais estabelecidas e escalando situações complexas para supervisão humana quando necessário. A arquitetura proposta integra conceitos de máquinas éticas, formalização de conflitos e protocolos de negociação entre agentes, contribuindo para o avanço do estado da arte em veículos autônomos éticos e fornecendo uma base sólida para pesquisas futuras nessa área.

REFERÊNCIAS

ALGOtive. **Guía de inteligencia artificial autónoma**: el futuro de la IA, . 2022. Disponível em: https://www.algotive.ai/es-mx/blog/guia-de-inteligencia-artificial-autonoma-el-futuro-de-la-ia. Acesso em: 29 jan. 2025.

ALVES, G. V.; DENNIS, L.; FISHER, M. An agent-based architecture with support to ethical decisions on a road traffic scenario. *In*: IROS WORKSHOP ON BUILDING AND EVALUATING ETHICAL ROBOTIC SYSTEMS. 2021. **Anais [...]** [*s.n.*], 2021. Disponível em: https://personalpages.manchester.ac.uk/staff/louise.dennis/pubs/ADFers21.pdf.

BENCH-CAPON, T. Ethical approaches and autonomous systems. **Artificial Intelligence**, v. 281„ p. 103239, abr. 2020. Disponível em: http://dx.doi.org/10.1016/j.artint.2020.103239.

BREMNER, P. *et al.* On proactive, transparent, and verifiable ethical reasoning for robots. *In*: PROCEEDINGS OF THE IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS). 2019, Macau, China. **Anais [...]** Macau, China: [*s.n.*], 2019. p. 4567–4572.

DOWLING, K.; MAHALINGAM, S.; WHITTAKER, W. L. NavLab: an autonomous navigation testbed. *In*: **The Kluwer International Series in Engineering and Computer Science**. Kluwer Academic Publishers, 1990. p. 259–282. Disponível em: http://dx.doi.org/10.1007/978-1-4613-1533-9_12.

Ethics Commission. **Automated and Connected Driving**. Germany, 2017.

FERNANDES, L. E. R. *et al.* A rational agent controlling an autonomous vehicle: implementation and formal verification. **Electronic Proceedings in Theoretical Computer Science**, v. 257„ p. 35–42, set. 2017. Disponível em: http://dx.doi.org/10.4204/eptcs.257.5.

FIPA. **FIPA Contract Net Interaction Protocol Specification**, . 2002. Disponível em: http://www.fipa.org/specs/fipa00029/.

GALTUNG, J. Violence, peace, and peace research. **Journal of Peace Research**, v. 6, n. 3, p. 167–191, 1969.

MELLADO, A. L. L.; BORGES, A. P.; ALVES, G. V. MASPY: A Python-Based Framework for Developing BDI Multi-agent Systems. *In*: ADVANCES IN Practical Applications OF Agents, Multi-Agent Systems, AND Computational Social Science: The PAAMS Collection. 2025. **Anais [...]** Springer Nature Switzerland, 2025. p. 216–227. ISBN 978-3-032-07638-0. Disponível em: https://link.springer.com/chapter/10.1007/978-3-032-07638-0_18.

PICARD, R. W. **Affective Computing**. Cambridge, MA, USA: MIT Press, 1995.

RAKOW, A. *et al.* Why we need a notion of conflict for autonomous traffic agents. *In*: COLLIER, R. W. *et al.* (Ed.). MULTI-AGENT SYSTEMS: 21st european conference, eumas 2024, dublin, ireland, august 26–28, 2024, proceedings. 15685 de **Lecture Notes in Computer Science**., 2024, Cham. **Anais [...]** Cham: Springer, 2024. Disponível em: https://doi.org/10.1007/978-3-031-93930-3.

RAKOW, A.; SCHWAMMBERGER, M. Brake or drive: on the relation between morality and traffic rules when driving autonomously. *In*: PROCEEDINGS OF SOFTWARE ENGINEERING 2023 WORKSHOPS. 2023. **Anais [...]** Gesellschaft für Informatik e.V., 2023. p. 104–115. Disponível em: http://dx.doi.org/10.18420/SE2023-WS-12.

RAO, A. S.; GEORGEFF, M. P. Bdi agents: From theory to practice. **Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)**, „ p. 312–319, 1995.

REED, N. *et al.* Ethics of automated vehicles: breaking traffic rules for road safety. **Ethics and Information Technology**, v. 23, n. 4, p. 777–789, out. 2021. Disponível em: http://dx.doi.org/10.1007/s10676-021-09614-x.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence**: A modern approach. 3. ed. Upper Saddle River: Prentice Hall, 2010.

SMITH, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. **IEEE Transactions on Computers**, C-29, n. 12, p. 1104–1113, 1980.

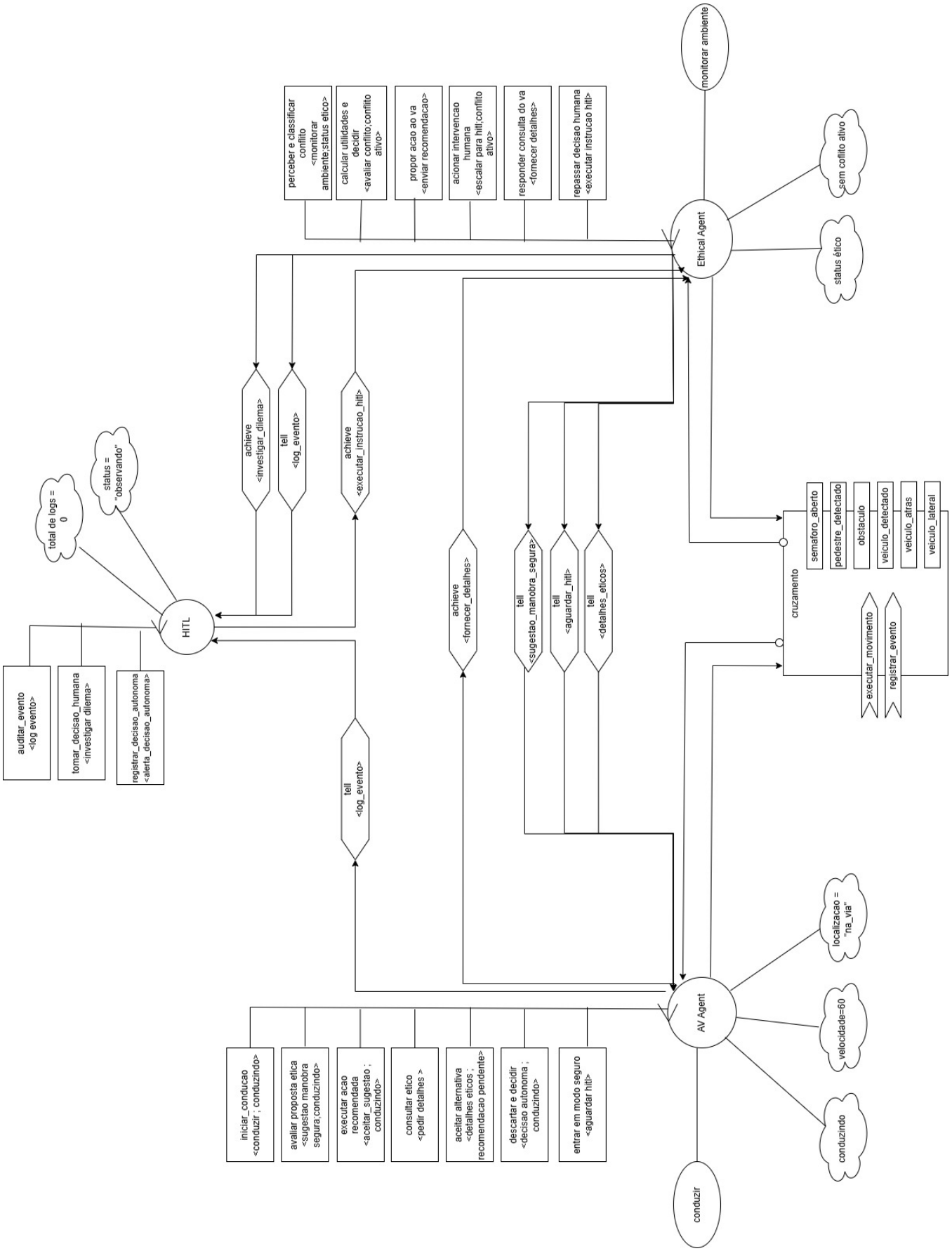
SPARROW, R. Killer robots. **Journal of Applied Philosophy**, v. 24, n. 1, p. 62–77, fev. 2007. Disponível em: http://dx.doi.org/10.1111/j.1468-5930.2007.00346.x.

TOLMEIJER, S. *et al.* Implementations in machine ethics. **ACM Computing Surveys**, v. 53, n. 6, p. 1–38, dez. 2020. Disponível em: http://dx.doi.org/10.1145/3419633.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. 2nd. ed. [*S.l.*]: John Wiley & Sons, 2009.

APÊNDICE A – Diagrama completo da arquitetura

Figura 18 – Diagrama completo da arquitetura do sistema



Fonte: Autoria própria.