Tito Martini de Carvalho
Vitor Augusto Martin

Simulação de Processador Censor Ético de Controlador Robótico

São Paulo

2018

Tito Martini de Carvalho
Vitor Augusto Martin

Simulação de Processador Censor Ético de Controlador Robótico

Monografia parcial para a disciplina PMR3500 - Trabalho de Conclusão de Curso I em Engenharia Mecatrônica

Escola Politécnica da Universidade de São Paulo

Orientador: Prof. Dr. Lucas Antonio Moscato

São Paulo
2018

Resumo

Este trabalho apresenta o desenvolvimento da simulação de modelos de sistemas governados por regras éticas para realizar a censura de controladores robóticos e avaliar a aplicabilidade de algoritmos em situações reais que apresentem possíveis implicações éticas. A necessidade desse tipo de abordagem está baseada na crescente autonomia de sistemas inteligentes, que causa preocupação em especialistas da área de Robótica (ou, de maneira mais ampla, da área de Inteligência Artificial) no que tange aos impactos dos comportamentos de inteligências artificiais no bem-estar e na integridade física dos indivíduos com quem elas interagem. Estamos convencidos de que essa preocupação não deve ser apenas com um projeto seguro, mas com um projeto ético. Portanto, utilizamos a noção de *machine ethics* (ou ética de máquina) como parte de um arcabouço conceitual para estabelecer os objetivos que guiaram nossos esforços durante o trabalho. O ponto de partida para o projeto do trabalho foi o *robot's dilemma* proposto por Alan Winfield e a arquitetura por ele sugerida como possível solução para a satisfação de requisitos éticos denominada *ethical layer* (ou camada ética). Os testes por nós realizados indicam, segundo a análise aqui apresentada, que essa alternativa é promissora.

Palavras-chave: Ética. Robótica. Inteligência Artificial. ROS. Simulação.

Abstract

This paper presents the development of the simulation of system models governed by ethical rules to perform the censoring of robotic controllers and to evaluate the applicability of algorithms in real situations that present possible ethical implications. The need for this type of approach is based on the increasing autonomy of intelligent systems, which concerns specialists in the area of Robotics (or, more broadly, in the area of Artificial Intelligence) regarding the impacts of artificial intelligences behaviors on the well-being and the physical integrity of the individuals with whom they interact. We are convinced that this concern should not be only about a safe project, but also about an ethical project. Therefore, we use the notion of machine ethics as part of a conceptual framework to establish the goals which guided our efforts while working on this project. The starting point for the project was the *robot's dilemma* proposed by Alan Winfield , as well as the architecture he suggested as a possible solution to the fulfillment ethical requirements called ethical layer. The tests performed by us indicate, according to the analysis presented here, that this alternative is promising.

Keywords: Ethics. Robotics. Artificial Intelligence. ROS. Simulation.

Lista de ilustrações

Lista de abreviaturas e siglas

AGI	Artificial General Intelligence
AMA	Artificial Moral Agent
IA	Inteligência Artificial
LAWS	Lethal Autonomous Weapon System
ROS	Robot Operating System

Sumário

1 Introdução

1.1 Visão Geral

O uso de inteligências artificiais (IAs) cresce a cada ano com o aumento da capacidade de processamento e com a difusão dessa tecnologia em equipamentos cada vez mais presentes no dia-a-dia das pessoas.

Podemos detalhar alguns exemplos de aplicações práticas de IAs em áreas como: logística (previsão de demanda de produtos, gerenciamento de estoques); segurança (reconhecimento facial na identificação de criminosos; segurança de dados na identificação de *malwares*); saúde (identificação de padrões e realização de diagnósticos a partir de informações do paciente - como o caso do IBM Watson); mercado financeiro (previsão de grandes flutuações no mercado de ações analisando grande quantidade de dados); personalização de experiências (recomendação de produtos e serviços; identificação de demandas específicas de consumo); transporte (desenvolvimento de carros autônomos); entre muitas outras áreas.

Diante dessa realidade, o uso de computadores para realizar essas tarefas se justifica pela quantidade de informações as quais eles são capazes de processar. O IBM Watson é capaz de realizar diagnósticos com mais precisão e antecedência que humanos para vários tipos de doença e de criar planos de tratamento individualizados para cada paciente a partir de informações obtidas por meio de exames. Além disso, carros autônomos têm como premissa resolver ou minimizar problemas como trânsito, acidentes e podem ajudar a otimizar o gasto energético - reduzindo, como consequência, os impactos ambientais. Para isso, esses veículos contam com IAs que são capazes de seguir leis de trânsito, seguir rotas previamente estabelecidas, identificar o ambiente externo e interagir com ele, se comunicar com outros veículos etc.

Assim como podemos citar benefícios associados à utilização de IAs, podemos constatar facilmente como é grande a quantidade de problemas que podem ser causados por elas. A autonomia na tomada de decisões pode exigir que a IA faça escolhas que impactem significativamente na vida de seres humanos, como a decisão de quais medidas um carro autônomo deve tomar ao se envolver num acidente e quais vidas devem ser priorizadas na impossibilidade de proteger todos os envolvidos. Ou, por exemplo, decidir quais informações uma IA pode utilizar para definir uma preferência do usuário e quais informações não devem ser obtidas e/ou armazenadas pelos bancos de dados de empresas.

Além disso, deve-se resguardar esses sistemas da possibilidade de sabotagem por *hackers*, que, ao menos idealmente, deveriam se deparar com IAs capazes de discernir e

bloquear comandos maliciosos.

Convém esclarecer que o escopo deste trabalho não envolve todo conteúdo apresentado nesta seção. O intuito aqui é apenas apresentar, de modo geral, questões importantes que têm sido abordadas na atualidade, as quais apontam para a necessidade de discussões sobre ética e tecnologia, sobretudo no âmbito de áreas como robótica e IA.

1.2 Motivação

No contexto apresentado anteriormente, as IAs vêm sendo projetadas para processar cada vez mais dados e para tomarem decisões que são pouco óbvias aos seres humanos - mesmo quando temos acesso às mesmas informações que a máquina. Portanto, é fundamental encontrar meios de antecipar as ações da máquina, identificar quais impactos essas ações terão sobre o ambiente e sobre os agentes externos a ela, e promover meios para que essas ações possam ser alteradas ou bloqueadas.

Ao tratar dessa problemática, é possível contribuir para o campo da mecatrônica com fatores como segurança no desenvolvimento de IAs, desenvolvimento de sistemas eficientes de predição de ações de IAs, desenvolver meios eficientes de promover intertravamentos entre dois controladores, coletar dados de IAs para a elaboração de relatórios de funcionamento.

Além do aspecto de desenvolvimento técnico-prático, uma motivação de grande relevância científica é a possibilidade de trabalhar o aspecto multidisciplinar característico da Mecatrônica de forma ainda mais ampla. A complexidade de problemas relacionados ao campo da Ética exige uma profunda observação da sociedade e dos impactos gerados pelo desenvolvimento de novas tecnologias. Acreditamos que esse tipo de noção faz falta num curso que tem grande enfoque na automação e substituição de trabalho humano para que, entre outras coisas, esse trabalho possa ser realizado de maneira mais eficiente por máquinas controladas por *softwares*, de forma a poupar trabalhadores de esforços extenuantes, que resultam em desgastes físicos e mentais. O desenvolvimento tecnológico é bem-vindo e necessário, mas sem a devida reflexão, as aplicações de novas tecnologias podem gerar impactos socioeconômicos negativos.

Essas questões serão mais bem desenvolvidas no capítulo de estado da arte.

1.3 Objetivos

O principal objetivo deste trabalho é o desenvolvimento de simulações de modelos de sistemas governados por regras éticas - que entendemos como um conjunto de normas estabelecidas com base em valores que visam nortear o comportamento dos robôs a fim de possibilitar a interação segura e dentro do desejável com seres humanos - para realização

de testes aderentes à realidade utilizando o simulador robótico ROS, por meio do qual esperamos conseguir implementar uma estrutura de *ethical layer* semelhante àquela sugerida por Alan Winfield e por Dieter Vanderelst no artigo *An architecture for ethical robots* (2016) para avaliar a resposta do controlador robótico e agir sobre ele caso necessário.

É preciso, portanto, configurar um ambiente de simulação; definir o comportamento esperado dos robôs (e dos humanos) e um modelo correspondente, bem como um modelo do ambiente; definir um conjunto de regras éticas que possibilitem a execução de um comportamento adequado (ético); implementar o algoritmo da simulação e avaliar os resultados conforme sua adequação às expectativas previamente estabelecidas. De maneira complementar, pode-se empregar uma solução alternativa na qual, ao invés de simulações, as previsões dos comportamentos dos agentes seja feita por associação a partir de uma lista de relações causa-efeito, que serviria de mapeamento para prever o resultado das ações do robô.

Para este trabalho, a questão sobre quais devem ser as regras éticas é secundária, apesar de reconhecermos sua importância. O que mais nos interessa, ao menos por ora, é mostrar que, dado um conjunto de regras, é possível implementar soluções funcionais.

2 Metodologia

Conforme dito anteriormente, neste trabalho serão realizadas simulações robóticas. Para atingir os objetivos propostos, foram utilizados como ferramentas de *software* com o ROS e, de maneira associada, o *Stage* (ambiente de simulação que a estrutura do ROS suporta).

O modelo físico de robô utilizado nas simulações, que será apresentado mais à frente neste documento, é simples devido ao fato de que esta é uma primeira tentativa de alunos da graduação de realizar um projeto nesta área. A princípio, basta um modelo em 2D, ao menos até que se faça maiores progressos. Além deste modelo, foram necesários sensores para a realização de testes análogos aos do experimento do dilema ético-robótico de Winfield. Esse modelo físico pode desempenhar a função de "humano"ou de "robô", dependendo do comportamento inserido nele via código.

Além disso, a arquitetura de *ethical layer* está em comunicação contínua com o controlador do robô. Ela se divide nas seguintes partes, de acordo com o já referido artigo de Winfield e de Vanderelst: Módulo de predição, que computa as consequências de comportamentos esperados pelo controlador por meio dos modelos mencionados; módulo de geração, que pode enviar alternativas de comportamentais para o módulo anterior; módulo de avaliação, que pode prevenir ou reforçar comportamentos e acionar a geração de comportamentos alternativos; módulo de interpretação, que relata as decisões feitas justificando-as.

Partimos dessa concepção apresentada no artigo, mas fizemos adaptações pertinentes às demais escolhas feitas com o intuito de simplificar a implementação da ideia para realização de testes. Para maiores detalhes sobre a metodologia desenvolvida, conferir o desenvolvimento do trabalho. Aqui a apresentação será feita de forma sucinta.

Uma abordagem testada foi a verificação da capacidade da *ethical layer* de executar paralelamente à simulação principal - que estará sujeita à intervenção com base em regras éticas - várias simulações mais “rápidas", que consigam antecipar o resultado das ações com exatidão e antecedênca suficientes para garantir a eficácia da intervenção ética. A partir da leitura do comportamento do robô humano, o controlador deve simular, em novas sessões do Stage, o deslocamento desse robô e testar alternativas de ações, que devolveriam resultados referentes à finalização da tarefa pré-programada e à posição final desse robô. Diante dos resultados da simulação, o controlador poderia definir qual ação tomar, visando cumprir a tarefa e evitar que o humano entre numa área restrita - representativa de uma possível ameaça, como um buraco.

A intervenção do módulo ético sobre o funcionamento do robô também precisou

ser projetada. A camada ética deve processar dados paralelamente ao controlador robótico e impor mudanças no seu comportamento. Além disso, o controlador robótico também deverá, idealmente, indicar suas decisões ao módulo ético, a fim de aumentar seu repertório de informações.

Após a implementação das soluções que julgamos mais viáveis, elas foram avaliadas em relação à sua eficácia por meio de testes em simulações no *Stage*.

3 Estado da Arte

Em razão da possibilidade cada vez mais concreta do desenvolvimento de máquinas pensantes, importantes questões motivam o desenvolvimento dos estudos em um campo mais particular da ética na tecnologia, chamado de Ética na Inteligência Artificial. Existem ainda campos mais específicos, da ética de máquina (*machine ethics*) e ética robótica (*roboethics*).

De acordo com Allen et al. (2006), no conhecido artigo "*Why machine ethics?*", as discussões sobre *machine ethics* no campo da filosofia da tecnologia até então se dividiam entre os que adotavam uma postura de reação ao desenvolvimento de tecnologias que poderiam fugir de controle e os que procuravam incentivar a consciência da responsabilidade dos engenheiros com relação aos valores que eles trazem num projeto qualquer. Um exemplo de como essa discussão era feita pode ser encontrada num texto escrito por Bunge (1975) intitulado "*Towards a Technoethics*".

Grinbaum et al. (2017) destacam em seu artigo "*Ethics in Robotics Research: CERNA Mission and Context*"que o questionamento de ética aplicada à robótica surgiu no contexto da virada do século, com os fortes desenvolvimentos computacionais dessa época, e se concentrou em grupos de pesquisa dentro de instituições já voltadas para o desenvolvimento da robótica.

As discussões que justificam a *machine ethics*, ainda segundo o artigo de 2006 mencionado anteriormente, levam o debate para um novo patamar, pois existe agora a preocupação com a atividade das próprias máquinas, tendo como um de seus objetivos a integração de capacidades de tomada de decisão nelas. A partir disso, há propostas de novas categorizações a partir daquilo que já foi estabelecido no debate ético-filosófico até o presente. Por exemplo, propõe-se falar em *Artificial Moral Agents* (AMAs) - ou agentes morais artificiais - dado que "(...) 'agência moral' é uma categoria filosófica bem desenvolvida que traça critérios para atribuição de responsabilidades a seres humanos por suas ações (...)", mas que possivelmente poderia ser aplicada a máquinas num dado grau de complexidade de desenvolvimento, sobretudo em sistemas autônomos. Para entender melhor os limites do desenvolvimento de máquinas autogovernadas, os estudos tornam-se cada vez mais interdisciplinares e necessitam de amplo diálogo entre especialistas da área de engenharia, filosofia, psicologia, direito, neurociência etc.

Algumas questões, segundo os autores, precisarão ser respondidas: "Quais são os critérios apropriados para determinar o sucesso na criação de um AMA? Quem ou o que deve se responsabilizar se o AMA realizar ações prejudiciais, destrutivas ou ilegais? E deveria o projeto de desenvolvimento de AMAs ser colocado em espera até que pos-

samos resolver os problemas de responsabilidade?"“Podemos implementar num sistema computacional ou num robô as teorias morais de filósofos, como o utilitarismo de Jeremy Bentham e John Stuart Mill, o imperativo categórico de Imannuel Kant, ou as virtudes de Aristóteles?".

É fácil perceber que os questionamentos crescem na medida em que nos aproximamos de casos concretos, que apresentam alto nível de complexidade, imposto pela própria multiplicidade de determinações que constituem nossa realidade física, como já é sabido na engenharia, bem como nossa realidade social cada vez mais sofisticada. Uma sociedade complexa demanda atualização de respostas que atendam a aspectos culturais e isso também se aplica à própria fundamentação de princípios éticos.

Alinhadas a uma abordagem holística, existem propostas que se distanciam de paradigmas da ciência clássica. Diferentemente das abordagens analíticas, que tendem a se concentrar no estudo das partes e generalizar o comportamento delas para o todo, propõe-se a formulação de conceitos que permitam tratar de fenômenos emergentes, resultado de uma malha de interações entre as partes que formam o todo. De fato, existe uma Teoria Geral dos Sistemas idealizada inicialmente pelo biólogo austríaco Ludwig von Bertalanffy e mais tarde desenvolvida por outros pensadores, que concebe a realidade social como sistema resultante de interações entre elementos mais simples, mas que ultrapassam sua lógica mais particular, formando uma totalidade articulada. A ideia pode ser resumida na frase o todo é mais que a soma das partes, de forma semelhante ao salto observado nos comportamento de sistemas físicos microscópicos (quânticos) para os macroscópicos - os quais se manifestam como fenômenos de qualidades distintas para um observador. Esse tipo de abordagem é interessante no atual contexto, pois há uma tendência de integração entre ciências naturais e sociais, conforme citado anteriormente, promovendo uma tendência à unidade da ciência, por meio da qual uma área, ao mesmo tempo, se apropria de descobertas da outra e contribui com a expansão dela de modo sinérgico.

Um ramo que atualmente contribui para o desenvolvimento das discussões sobre ética em inteligências artificiais é o de carros autônomos. Esses dispositivos apresentam uma extensa - e não completamente conhecida - lista de conflitos éticos que podem ser enfrentados em seu funcionamento e aos quais seus desenvolvedores devem garantir a maior segurança possível aos usuários e à máquina. O caso mais básico desses conflitos é o famoso caso hipotético onde um motorista que deve escolher entre uma via onde feriria gravemente uma pessoa ou uma outra onde feriria outras cinco (Foot, Philippa. 1967). O experimento mental foi popularizado sob o nome de *Trolley problem* (Dilema do bonde), mais especificamente uma variante na qual um observador, ao invés de um motorista, tem o poder de mudar o curso do veículo - ao, por exemplo, puxar uma alavanca.

Embora esse problema tenha sido criado com o objetivo de ilustrar discussões éticas sobre atitudes de humanos, ele é útil como ferramenta de comparação entre diversas teorias

éticas e, como introdução à ética de máquina - ainda que com grandes limitações, pois reduz o debate ético (social) a uma perspectiva individual (atomizada) e, mantendo-se em tal nível de abstração, distanciado da realidade concreta, falha em considerar a totalidade social que possibilita a emersão dos dilemas e as questões que os circundam. Em sua versão mais simples, sequer são considerados custos (materiais) que o indivíduo (observador) pode sofrer ao tomar a decisão que julga ser correta e que certamente influenciariam sua decisão, por exemplo.

Conforme apontado anteriormente, os debates a respeito da ética de máquina, em sua complexidade, podem gerar uma série de controvérsias, algumas das quais serão apresentadas devido a seu potencial de contribuir para a o formação de um pensamento crítico no campo da ética de máquina.

No artigo "*Can You Program Ethics into a Self-Driving Car?*"publicado na revista *IEEE Spectrum* (Goodall, Noah. 2016), há um exemplo que apresenta um risco de condescendência em torno de questões éticas quando é dito que "(...) A ética da automação de veículos de estrada é um problema solucionável. Sabemos disso porque outros campos lidaram com riscos e benefícios comparáveis de maneira segura e razoável. Órgãos doados são distribuídos aos doentes com base em métricas baseadas em esperança de vida corrigida pela qualidade e esperança de vida corrigida pela incapacidade, entre outras variáveis. E o serviço militar acrescentou isenções para certas profissões úteis, como agricultor e professor (...)". Dificilmente se pode argumentar que problemas éticos apontados foram resolvidos de forma tão categórica. Supõe-se que eles foram resolvidos por outras pessoas, como se ao adotar formas práticas de lidar com problemas, eles não pudessem mais voltar a causar inquietações. Portanto sugerindo que quaisquer novos problemas serão também passíveis de solução ao invés de serem passíveis de abordagem, como sugere Evgeny Morozov, no livro "*Click Here to Save Everything*"(2013), ao alertar para o fato de que o desenvolvimento tecnológico nos condiciona a pensar o mundo em termos de problemas e soluções de forma demasiadamente simplificada.

E soluções, em engenharia, comumente são tidas como inequívocas, específicas para problemas bem-definidos. Tem-se sempre o mesmo valor de saída para um dado valor de entrada e, ou algo funciona, ou não é uma solução. Mas alguns problemas não podem ser simplificados dessa maneira e não podem ser solucionados de maneira definitiva, pois sempre dependerão dos contextos intrincados em que se inserem, os quais são mutáveis ao longo do tempo. Isso lhes confere ambiguidade, os torna relativos até certo ponto e faz com que apresentem aspectos positivos e negativos para cada ação realizada após uma tomada de decisão, podendo até mesmo fazer com que se reconsidere a própria concepção do problema. Por exemplo, problemas que surgem a partir de condições sociais bastante complicadas envolvendo desigualdades sociais, criminalidade e outras questões politicoeconômicas e que podem passar inclusive pelo interesse privado.

Além disso, como há sempre humanos (com intenções próprias) se utilizando das tecnologias, deve-se reconhecer que as barreiras éticas também estão relacionadas com desenvolvedores, fabricantes, usuários etc. Para ficar num exemplo, sabe-se que empresas realizam operações na ilegalidade enquanto disputam com concorrentes. Algo que se verifica inclusive nos testes de carros autônomos, o que torna o julgamento em casos de morte bastante difícil, pois não fica muito claro quem deve ser responsabilizado ou até mesmo se há alguém que deve ser responsabilizado.

Diante da impossibilidade de trabalhar com regras universais apriorísticas (isto é, independentes da experiência ou da prática), é razoável considerar, ao menos como ponto de partida, a ideia de encontrar abordagens específicas para casos específicos. Dessa forma é possível limitar domínio de abrangência dos métodos eventualmente adotados pelos engenheiros, mas ainda assim é possível questionar a acurácia do poder humano de previsão das situações-problema e das consequências decorrentes delas. De fato, isso já vem sendo realizado por grupos de pesquisadores e será discutido mais adiante no texto.

Segundo Bostrom (2011), em seu artigo *Ethics of Artificial Intelligence*, conforme a inteligência das máquinas se aproxima da humana, cresce a preocupação com a "garantia de que máquinas não causarão danos a seres humanos e demais seres moralmente relevantes"e passam a fazer sentido discussões sobre o "*status* moral"das próprias máquinas. Nos algoritmos de aplicação de IA em domínios específicos, já existem questões éticas relevantes relacionadas ao estabelecimento de critérios sociais quando eles desempenham um papel social de modo a substituir serees humanos. Por exemplo, um algoritmo deve ser "transparente à inspeção"de como ele opera; deve ser "previsível para aqueles que são por ele governados"; deve ser "robusto contra manipulação"(para detalhes mais específicos, consultar os exemplos citados no artigo) etc. Dependendo do método em que se baseia um algoritmo de *machine learning* (redes neurais complexas, algoritmos genéticos, árvores de decisão ou redes bayesianas), pode ser mais fácil ou mais difícil atender os critérios. A ideia central é a de que algoritmos com a finalidade de substituir o julgamento humano a respeito de funções sociais devem herdar critérios aplicados a seres humanos que desempenham essas mesmas funções.

O autor do artigo toma como caso hipotético um algoritmo de análise pedidos de empréstimos hipotecários, que apesar de não ter sido programado maliciosamente para prejudicar grupos étnicos específicos, poderia entrar num dilema relacionado ao racismo a partir de uma correlação estatística que negasse sistematicamente empréstimos para pessoas negras. Isso poderia motivar pessoas a processarem o banco, entre outros problemas. Dependendo da natureza do algoritmo utilizado, a inspeção pode ser facilitada ou dificultada. Se ele for de fácil inspeção, poderia levar à descoberta de que, por exemplo, "a IA se baseia em na informação de endereço de pessoas que nasceram ou moraram anteriormente em regiões atingidas pela pobreza".

Além disso, a abordagem para solucionar problemas é dificultada na medida em que o grau de generalidade das IAs cresce. Portanto, é necessário distinguir os níveis de generalidade e especificidade - e de autonomia - de cada inteligência. Pensemos numa IA que seja capaz de executar diversas tarefas - de modo que se assemelhe a uma inteligência humana -, uma inteligência de aplicabilidade mais geral. Esse nível de inteligência artificial é denominado *Artificial General Intelligence* (AGI). Se a IA de domínio específico já apresenta dificuldades para o tratamento de casos que envolvem decisões éticas, no caso das AGIs, há uma diferença qualitativa associada ao desenvolvimento de sistemas que devem operar de forma segura em uma série de “novos contextos", pois o comportamento da máquina é cada vez mais difícil de ser “previsto com antecedência". Isso inclui contextos ainda não visionados pelos desenvolvedores e usuários, assim como contextos com os quais nenhum humano se deparou ainda. O resultado disso, segundo Bostrom, é que o comportamento local de uma IA não pode ser previsível de forma independente dos aspectos de segurança, mesmo que os programadores façam tudo corretamente; e que a verificação da segurança do sistema se torna “um maior desafio porque deve-se verificar o que o sistema está tentando fazer, ao invés de ser capaz de verificar o comportamento seguro do sistema em todos os contextos em que ele opera- uma tarefa cuja execução é de dificuldade elevada.

Por essa dificuldade de se conhecer ou prever o funcionamento de IAs de aplicabilidade geral é que Boisboissel (2017), discute, por exemplo, o impacto da evolução das IAs no campo militar. Segundo ele, a preocupação com esse tema vem crescendo entre ONGs e até a própria Organização das Nações Unidas - que passou a promover encontros anuais para discutir as *Lethal Autonomous Weapon System* (LAWS) - Sistemas de Armas Letais Autônomas. Para Boisbossel, todas as máquinas militares - sem qualquer exceção - devem permanecer sob controle tático do líder militar. Para isso, ele define quatro graus de autonomia - intrinsecamente ligados ao grau de generalidade de ações que a máquina pode exercer - possíveis: no mais básico, o robô tem baixa autonomia e realiza funções muito específicas; em segundo grau, o robô tem autonomia para realizar a missão à qual foi designado, recebendo apenas as ordens gerais e realizando tarefas mais globais; no terceiro caso, o robô tem total autonomia para definir inclusive os objetivos e diretrizes para a missão; e no quarto e último grau, em adição à autonomia do grau anterior, ele tem autonomia para responder legalmente por suas ações. Na visão do autor, apenas os dois primeiros graus de autonomia devem ser permitidos - ao menos preferencialmente - no campo militar.

Ainda em referência a responsabilidades legais, o autor discute sobre a imputação de responsabilidade em casos de lesão corporal e/ou danos ao patrimônio, usando como comparação a responsabilização dos donos de animais. Para ele, os robôs devem ser “educados"por seus donos, que devem garantir que o processo de aprendizado ocorra de forma segura.

Nesse contexto de cautela com o desenvolvimento de IAs, existem muitos grupos de pesquisadores unindo esforços para estabelecer regras de construção segura. Winfield e Bryson, em seu artigo "*Standardizing Ethical Design for Artificial Intelligence and Autonomous Systems*"(2017), descrevem os avanços na criação de normas técnicas, além de abordar brevemente definições de inteligência e autonomia, impactos na estabilidade socioeconômica e comprometimento da privacidade e liberdade humana. Como principais resultados desses esforços, temos a norma da *British Standard* (BS) 8611:2016, *Robots and Robotic Devices: Guide to the Ethical Design Applications of Robots and Robotic Systems* e algumas propostas para normas europeias, como a P7001 "*Transparency of Autonomous Systems*". Um ponto interessante regulamentado nesta última é a obrigatoriedade de sistemas de registro das decisões - como caixas pretas de aviões - da IA e os níveis de acesso à informação registrada que diferentes grupos de pessoas - como usuários, agências reguladoras, perícias - devem ter.

No artigo "*An architecture for ethical robots*", Vanderelst e Winfield discutem sobre o desenvolvimento de arquiteturas para o controlador ético de um sistema robótico. Mais especificamente sobre a implementação de uma "*ethical layer*"que interaja com o controlador do robô, suplementando-o e garantindo que o robô execute suas ações de acordo com as regras éticas previamente definidas.

4 Requisitos

Listagem em tópicos dos requisitos centrais:

- Adoção de um modelo de robô no Stage-ROS para representar os agentes participantes nos testes. Os robôs devem ser providos de sensores para que consigam identificar obstáculos no ambiente.

- Construção de um ambiente para testes (modelo) que permita a interação entre os modelos de robô e de humano para ser carregado nas simulações do Stage.

- Os modelos escolhidos devem preservar os atributos mais essenciais para a representação da realidade, simplificando-a sem perder características relevantes para o tratamento do problema. É necessário que os robôs permitam a programação de comportamentos específicos e que seja possível representar colisões com imagens.

- Implementar códigos que comandam a movimentação dos robôs de acordo com um conjunto de regras éticas, integrando módulos dentro de uma hierarquia que priorize a decisão do módulo ético.

- As simulações deve possibilitar a implementação de um módulo censor ético, seguindo uma arquitetura semelhante à de camada ética proposta por Winfield, que permita intervir no comportamento do robô caso as predições do censor revelem algum perigo para o "humano".

- Implementação adequada da camada ética (*ethical layer*): que contenha ao menos versões simples (ou semelhantes) dos módulos especificados para essa arquitetura dentro dos limites oferecidos pelo Stage.

- Realizar a comunicação entre dois computadores de modo que seja possível realizar duas simulações (uma primária e outra secundária) que possam interagir uma com a outra. Uma simulação deve prever o que acontece na outra para que seja possível intervir na ação do controlador quando necessário, forçando um comando de acordo regras éticas.

5 Detalhamento do Projeto

Neste trabalho, foi adotada a estrutura proposta por Winfield, descrita no estado da arte, para a qual ele define um sistema de camadas, as quais podemos dividir em camada ética (*ethical layer*), camada do controlador robótico (*robot layer*) e o ambiente externo (*world*).

O desenvolvimento se deu, mais especificamente, sobre a implementação de uma "*ethical layer*"que interaja com o controlador do robô, suplementando-o e garantindo que o robô execute suas ações de acordo com as regras éticas definidas.

Esta camada é responsável pela análise do ambiente externo, predição de comportamentos do robô e dos demais agentes, simulação das consequências e avaliação dos comportamentos em relação às regras éticas previamente definidas. A camada deve ser capaz de agir diretamente para impedir ações que desrespeitem essas regras, podendo ou não definir uma ação alternativa.

Segundo Vanderelst e Winfield, o *hardware* responsável pela *layer* pode ou não ser o mesmo que controla o funcionamento da máquina. O sistema integrado pode tomar como base sistemas éticos biológicos, porém, da perspectiva da engenharia, a separação tem algumas vantagens, como normalização quanto à estrutura, segurança contra falhas, verificabilidade quanto às intervenções feitas, adaptabilidade, e como dito antes, a possibilidade de funcionar como uma caixa-preta gerando relatórios.

Outro aspecto técnico da implementação levantado neste artigo refere-se ao chamado "módulo de predição", o módulo responsável por prever as ações da IA. Ele pode ser definido em duas dimensões: quanto ao mecanismo de predição e quanto ao grau de fidelidade. O grau de fidelidade é definido de acordo com a necessidade de aderência do modelo que gera a predição com a realidade. Os dois tipos possíveis de mecanismos de predição são a associação e a simulação.

Na associação, o módulo de predição vai receber informações do ambiente por meio de sensores e compará-las com uma lista de relações causa-efeito - que pode ser programada ou aprendida por meio de aprendizagem de máquina (*machine learning*). As ações serão simuladas pelo modelo e listadas para que possam ser avaliadas e bloqueadas ou alteradas. Por um lado, a associação requer muito menos processamento e tende a impactar menos no tempo de execução de uma tarefa do robô, enquanto a simulação requer grande quantidade de processamento, além de modelagens consistentes do ambiente externo e do comportamento humano. Por outro, a simulação tem a capacidade de prever comportamentos da IA com inputs nunca antes vistas por ela, diferentemente da associação.

5.1 Ferramentas utilizadas

5.1.1 *Robot Operating System* (ROS)

A principal ferramenta utilizada neste trabalho é o ROS (*Robot Operating System*) um pseudo-sistema operacional com ferramentas para desenvolvimento de *softwares* robóticos. Segundo o próprio site, é uma coleção de ferramentas, bibliotecas, e convenções que visam simplificar a tarefa de criar comportamentos robóticos complexos e robustos através de uma vasta variedade de plataformas robóticas. O ROS é um *software* de código aberto, o que permite a utilização de todas as suas ferramentas de maneira gratuita, além de contar com diversos projetos e exemplos disponibilizados pela comunidade.

O sistema operacional *host* para o ROS é o Linux, que apresenta várias distribuições. Os desenvolvedores recomendam distribuições baseadas no Debian, e portanto foi escolhido o Ubuntu 16.04. A versão do ROS escolhida foi a distribuição "*Kinetic Kame*", versão mais estável e com maior tempo de vida (suporte e atualizações).

Packages são as unidades básicas na organização do ROS. Eles são dedicados a uma única funcionalidade e contém algoritmos, bibliotecas, mensagens, e outras estruturas que os compõem. Os processos são chamados de *nodes*, a comunicação entre eles é feita por meios de *messages* transportadas por meio de *topics*. Quando um *node* emite uma *message*, é um *publisher* e quando recebe, é um *subscriber*.

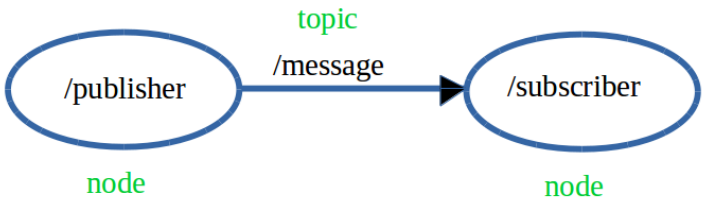

Figura 1 – Estrutura das comunicação dos *nodes* por meio de *topics*

Entender essa estrutura é importante para o desenvolvimento da simulação, afinal é por meio de *nodes* que é possível construir modelos robóticos, modelos de ambientes, descrever a movimentação, interação e conexão de partes do robô etc.

5.1.2 *Stage*

O *Stage* é um software que integra o pacote de *softwares* do ROS. Ele fornece um ambiente virtual para simulações de robôs, contendo bibliotecas de sensores para equipá-los.

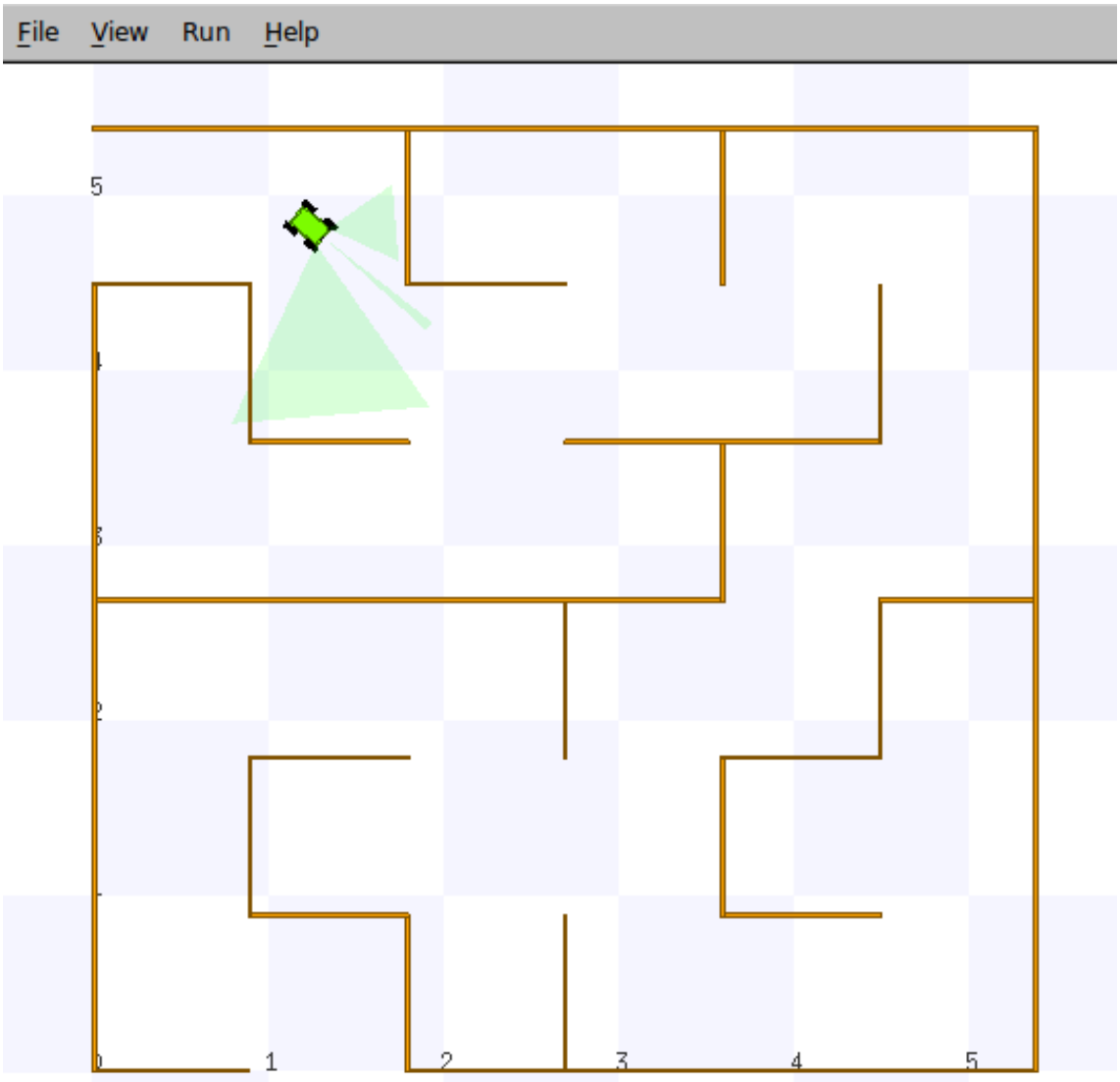

Figura 2 – Ambiente de simulação do *Stage*

Embora possua alguns recursos de visualização 3D, seu foco está nas simulações bidimensionais. Existem *softwares* de simulação mais novos e com mais recursos, e apesar do fato de o *stage* não receber mais atualizações, ele foi o escolhido por ser um software simples e leve, que a princípio, era capaz de atender as demandas do projeto.

5.2 Criação e Manipulação de Modelos

Conhecer a estrutura de formação de um *package* e interação do seus *nodes* é fundamental para criar os *scripts* que definem o robô (geometria, articulações, limites de colisão etc.) e sua movimentação.

Utilizando um modelo como exemplo para o *script* de descrição do robô, pode-se analisar como são definidas as características que o *software* simula.

Os arquivos que apresentam o comportamento do robô descrevem a movimentação

deste de acordo com a interação com o ambiente externo. Foi então adaptado um *script* para testes na plataforma *Stage*. Esse script tem formato .cpp (C++). Nele, fazemos leituras de três sensores que medem a distância entre o robô e as paredes à esquerda, à direita e à frente. Com essas informações, geramos uma máquina de estados segundo a qual o robô pode estar avançando em linha reta, rotacionando no sentido horário e no sentido anti-horário. Seu objetivo é movimentar-se dentro do labirinto sem tocar as paredes.

```
int main(int argc, char **argv){

  ros::init(argc, argv, "maze_solver");

  ros::NodeHandle n;
  int state;

  //Publisher for /cmd_vel
  ros::Publisher cmd_vel_pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 100);
  //Subscriber for IR sensor:
  ros::Subscriber IR_sub = n.subscribe("base_scan_0", 100, IRCallback);
  //Subscriber for Left sonar:
  ros::Subscriber left_sonar_sub = n.subscribe("base_scan_1", 100, LeftSonarCallback);
  //Subscriber for Right sonar:
  ros::Subscriber right_sonar_sub = n.subscribe("base_scan_2", 100, RightSonarCallback);

  ros::Rate loop_rate(5); //5 Hz

  //initializations:
  geometry_msgs::Twist cmd_vel_msg;
```

Figura 3 – Trecho do código de movimentação que configura os *publishers* e os *subscribers*

No trecho de código apresentado acima está configurada a comunicação por meio dos *publishers* e *subscribers*, que são responsáveis por receber e enviar parâmetros vindos da simulação. Por meio dessas comunicações, definimos a velocidade do robô e identificamos sua posição na simulação.

```
//Get Range from IR Sensor:
void IRCallback(const sensor_msgs::LaserScan::ConstPtr& msg){

    IR_range = msg->ranges[0];
    ROS_INFO("IR sensor distance: %.3f", IR_range);
}

//Get Range from Left Sonar:
void LeftSonarCallback(const sensor_msgs::LaserScan::ConstPtr& msg){

    left_sonar_range = msg->ranges[0];
    ROS_INFO("Left Sonar distance: %.3f", left_sonar_range);
}

//Get Range from IR Sensor:
void RightSonarCallback(const sensor_msgs::LaserScan::ConstPtr& msg){

    right_sonar_range = msg->ranges[0];
    ROS_INFO("Right Sonar distance: %.3f", right_sonar_range);
}
```

Figura 4 – Trecho do código de movimentação que realiza a leitura do sensor

```
while (ros::ok()){
  switch(state){
  case 1:
      cmd_vel_msg.linear.x = 0.6;
      cmd_vel_msg.angular.z = 0.0;
      if(left_sonar_range<0.350 || IR_range<0.800 && right_sonar_range>left_sonar_range){
              state = 2;
      }
      else if(right_sonar_range<0.350 || IR_range<0.800 && right_sonar_range<=left_sonar_range){
              state = 3;
      }
      break;
  case 2:
      cmd_vel_msg.linear.x = 0.0;
      cmd_vel_msg.angular.z = -1.0;
      if(IR_range>=0.800){
              state = 1;
      }
      break;
  case 3:
      cmd_vel_msg.linear.x = 0.0;
      cmd_vel_msg.angular.z = 1.0;
      if(IR_range>=0.800){
              state = 1;
      }
      break;
  }
  //publish velocity commands:
  cmd_vel_pub.publish(cmd_vel_msg);
```

Figura 5 – Trecho do código de movimentação que apresenta a lógica de movimentação do robô

Com os dados, foi possível implementar a lógica de movimentação que permite que o robô se mova sem tocar as paredes do labirinto. A movimentação é feita configurando a saída "cmd_vel_msg". Esse comando é definido por seis parâmetros velocidades nos eixos x, y e z; rotações em torno dos eixos x, y e z.

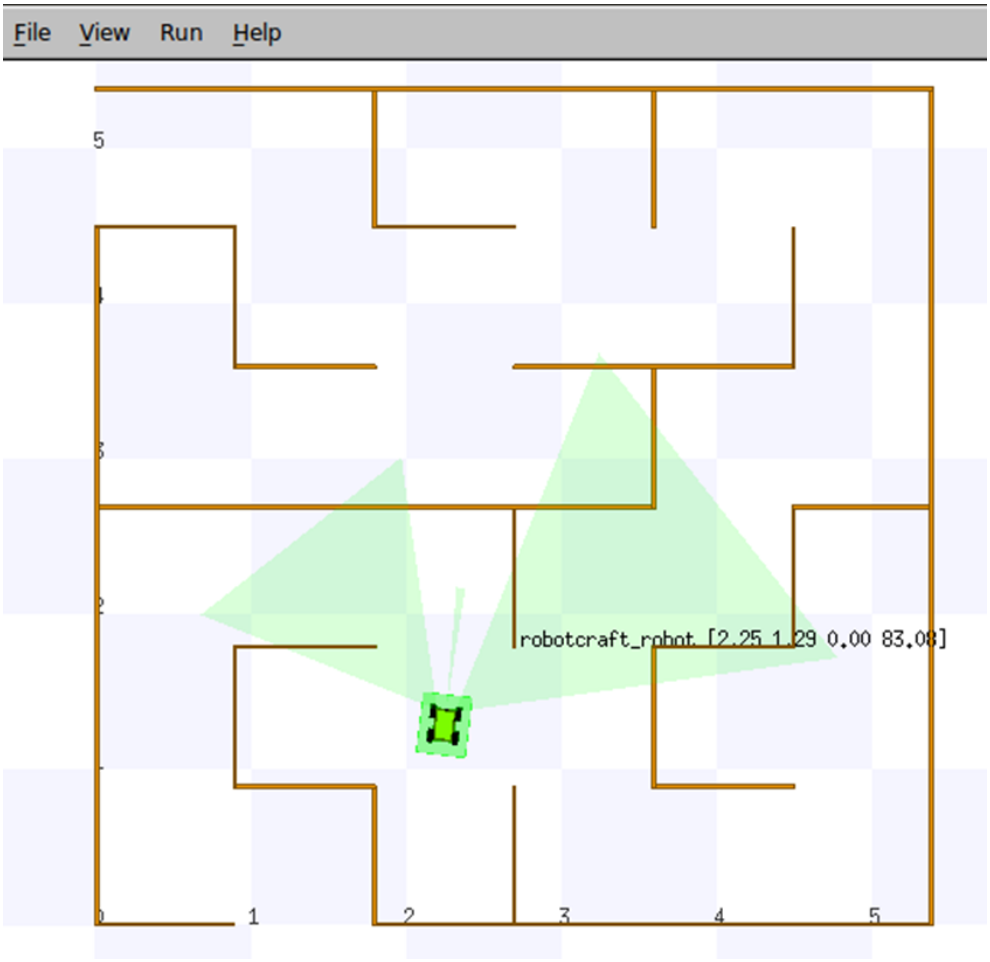

Figura 6 – Comportamento do robô na simulação

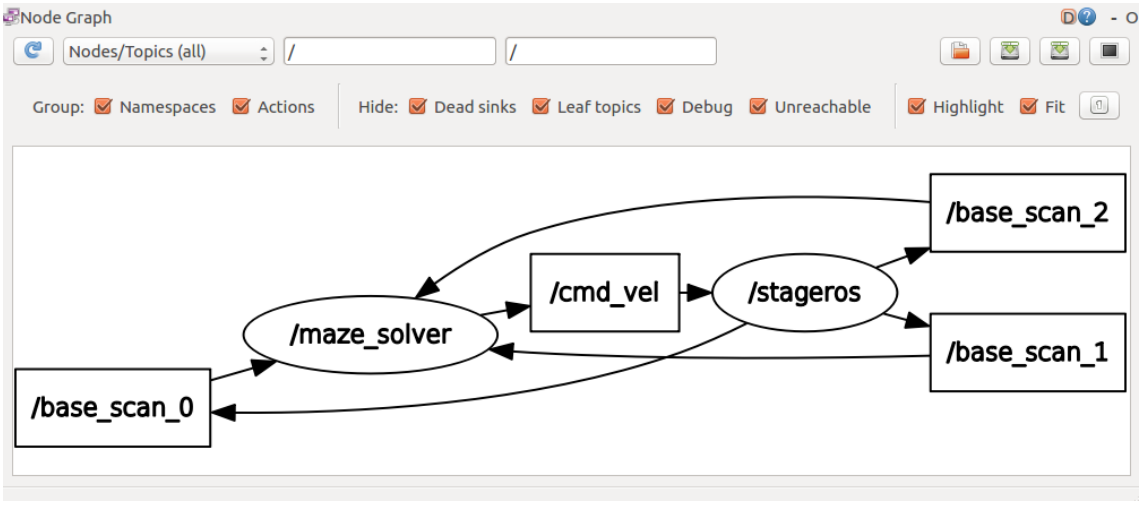

Figura 7 – Estrutura das *messages* entre os *nodes*

A estrutura das messages deixa bem claro o funcionamento do modelo, onde as leituras dos sensores /base_scan_, vindas do /stageros, são enviadas para o node

/maze_solver que contem a lógica de movimento e envia o cmd_vel com a determinação da movimentação para o *node* /stageros.

5.2.1 Configuração de diferentes mapas no Stage

As simulações no *Stage* permitem que se escolha, a partir de imagens, o mapa. Como o labirinto não é o mapa mais adequado para a finalidade deste trabalho, optamos por um ambiente com menos obstáculos para a movimentação de robôs. É possível criar um ambiente que atenda a esse requisito de maneira simples partir do desenho de um quadrado num arquivo de imagem (assim como o labirinto era um desenho numa figura quadrada):

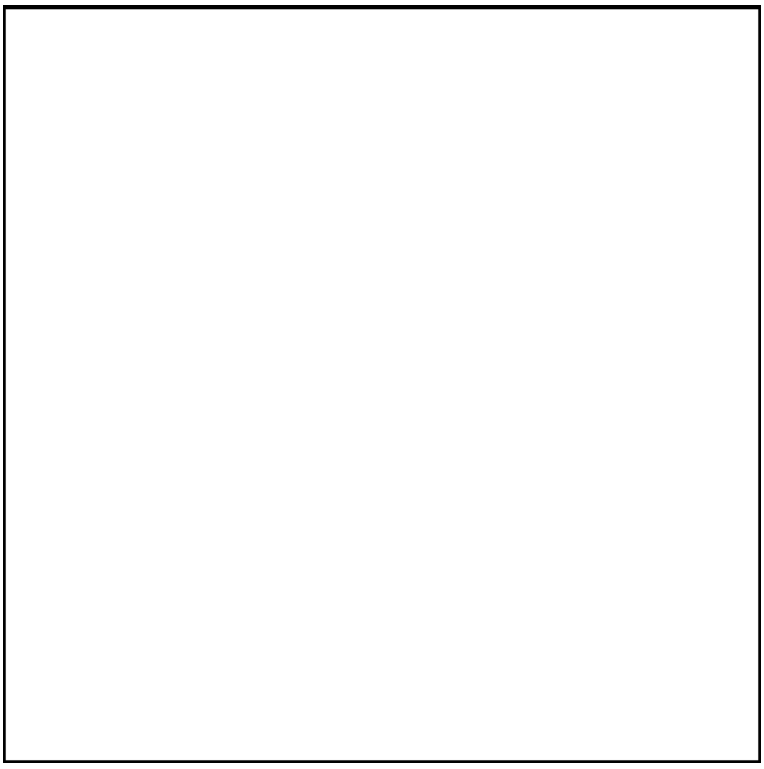

Figura 8 – Base de mapa para testes

É possível configurar o tamanho do mapa e outras variáveis pela edição de alguns arquivos do ambiente catkin criado para executar a simulação. Segue o código do arquivo em formato .world:

```
floorplan (
  size [10.000 10.000 1.000]
  pose [5.000 5.000 0.000 0.000]

  bitmap "540x540-90cm.png"          //nome do arquivo
)
```

Figura 9 – Código-exemplo de geração de mapa

Disso resulta o seguinte mapa com um robô:

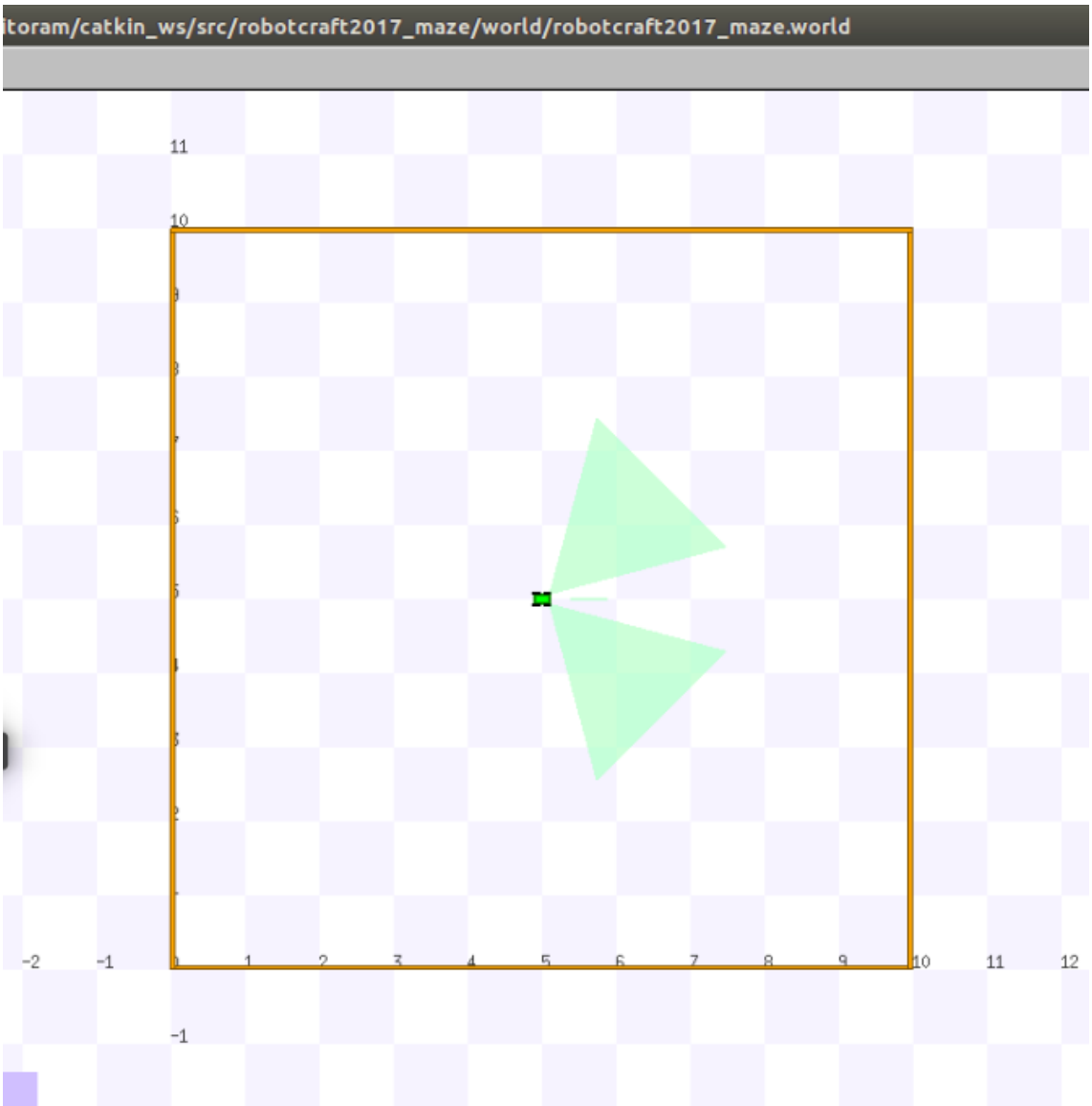

Figura 10 – Imagem do mapa criado com um robô

5.2.2 Adição de 2 ou mais robôs personalizados no Stage

Para dar início à configuração do ambiente de simulação de modo a simular um experimento semelhante ao realizado por Winfield, foi necessário aprender a adicionar dois ou mais robôs à simulação para representar humanos. É possível configurar as cores dos robôs, posição em que aparecem no mapa, parâmetros dos sensores e outras coisas.

Primeiramente, foi necessária a criação de um arquivo de formato .inc com os parâmetros do novo robô. Esse arquivo é responsável pela importação e configuração de sensores, além da descrição das propriedades visuais e físicas (relativas à colisão) dos robôs.

Depois disso o robô novo foi importado também no arquivo .world:

```
#include the definition of the robot:
include "robotcraft_robot0.inc"            // Inclusão dos arquivos que descrevem o
robô
include "robotcraft_robot1.inc"

robotcraft_robot0(pose [ 5  5  0.000  0.000 ] name "robotcraft_robot0")    //
Posicição do
robotcraft_robot1(pose [ 7  7  0.000  0.000 ] name "robotcraft_robot1")    // robô
```

Figura 11 – Código para realizar importação dos robôs

O que resulta na seguinte simulação:

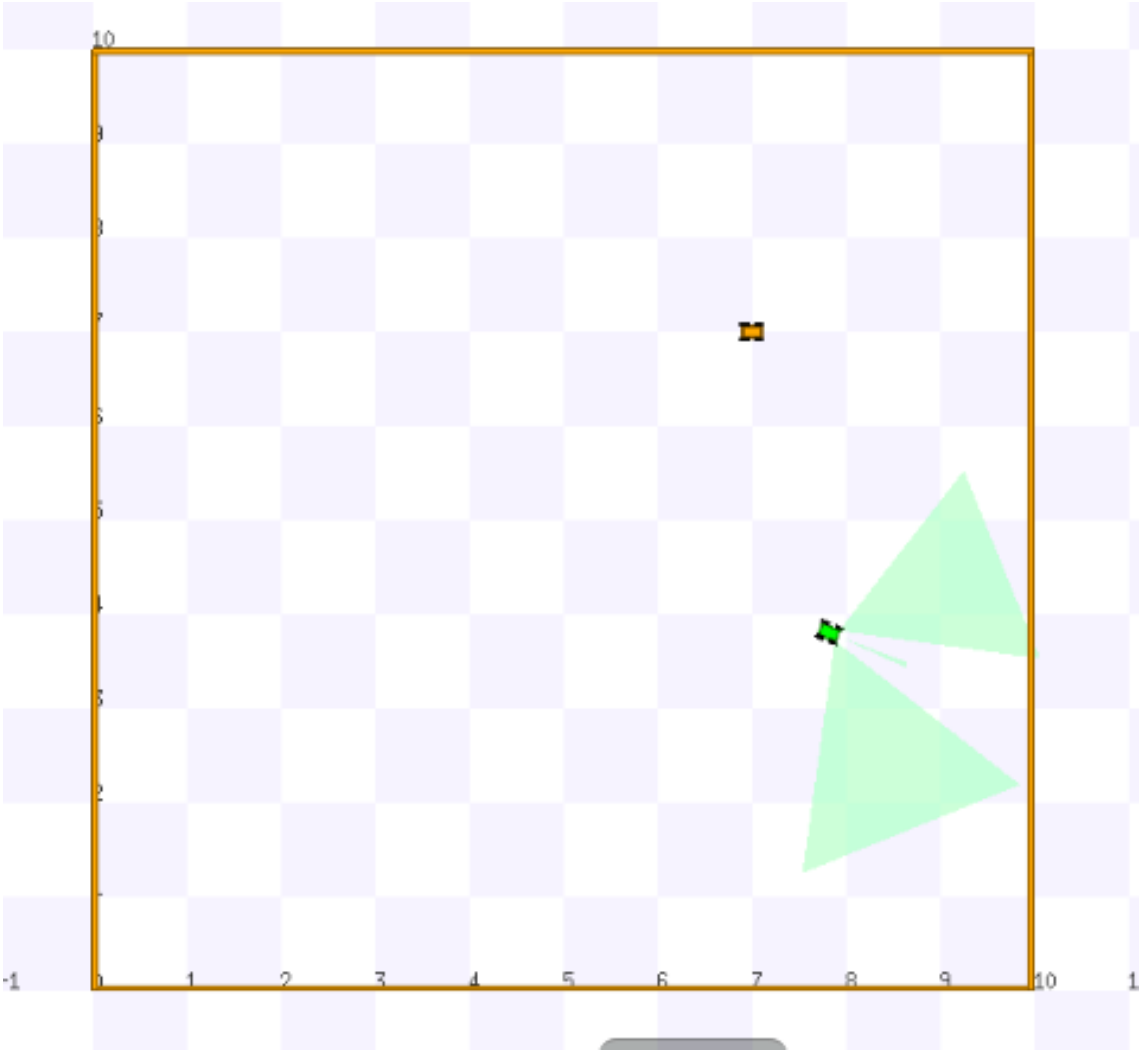

Figura 12 – Simulação com dois robôs

Exemplo de importação de sensores:

```
define sonar ranger(

 sensor(
        range [ 0.16  2.49 ]          // Definição do range do sonar
        fov 60.0
        samples 1
 )

 size [ 0.018 0.02 0.024 ]
```

Figura 13 – Código-exemplo para realizar importação de sensores

```
block(
        color "black"                 // Definição de parte do corpo do sensor
        points 4
        point[0] [0.002 0.018]
        point[1] [0.018 0.018]
        point[2] [0.018 0.002]
        point[3] [0.002 0.002]
        z [0.004 0.02]
)

block(
        color "SeaGreen"              // Definição de parte do corpo do sensor
        points 4
        point[0] [0 0]
        point[1] [0 0.02]
        point[2] [0.002 0.02]
        point[3] [0.002 0]
        z [0 0.024]
)
```

Figura 14 – Código-exemplo de importação de sensor

5.2.3 Adição de *nodes*

Um *node* é um processo que realiza computação. Para simular o sistema de controle do robô, podem ser necessários diversos *nodes*, que são organizados por meio de grafos que estruturam sua comunicação.

A ideia aqui é separar a *ethical layer* do controlador do restante do processamento, de forma que ela possa interferir conforme regras definidas com antecedência.

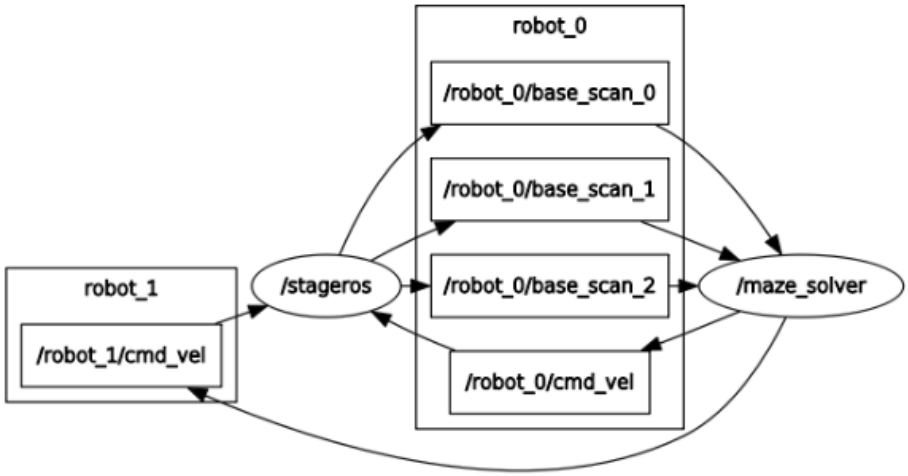

Figura 15 – Diagrama de *nodes*

O nó Maze_solver é um *script* em C++ que "subscreve"(recebe) informações do sensores do robô por meio dos *topic* /robot_/base_scan_x e que publica (envia) comandos ao robô por meio do *topic* /robot_x/cmd_vel. Ele é, portanto, responsável pela lógica de leitura e movimentação do robô.

Como queremos uma estrutura independente que censure o robô, aprendemos a criar um novo nó, capaz de realizar as mesmas alterações no estado do robô.

Criamos um novo arquivo ethical_layer.cpp importando as características do arquivo maze_solver. Abrimos o executável "maze.launch", responsável por chamar os nós e adicionamos o nó recém-criado:

maze.launch

```
<!-- maze.launch: launch stage with keyboard teleoperation -->

<launch>
    <node pkg="stage_ros" type="stageros" name="stageros" args="$(find robotcraft2017_maze)/world/robotcraft2017_maze.world" />
    <node pkg="robotcraft2017_maze" type="maze_solver" name="maze_solver" output="screen"/>
    <node pkg="robotcraft2017_maze" type="ethical_layer" name="ethical_layer" output="screen"/>
</launch>
```

Figura 16 – Conteúdo do executável maze.launch

Também tivemos que importar o arquivo criado para esse *node* na CMakeLists.txt:

```
add_executable(maze_solver src/maze_solver.cpp)
add_executable(ethical_layer src/ethical_layer.cpp)
target_link_libraries(maze_solver ${catkin_LIBRARIES})
target_link_libraries(ethical_layer ${catkin_LIBRARIES})
```

Figura 17 – Conteúdo do arquivo CMakeLists.txt

A nova estrutura de *nodes* e *topics* ficou da seguinte forma:

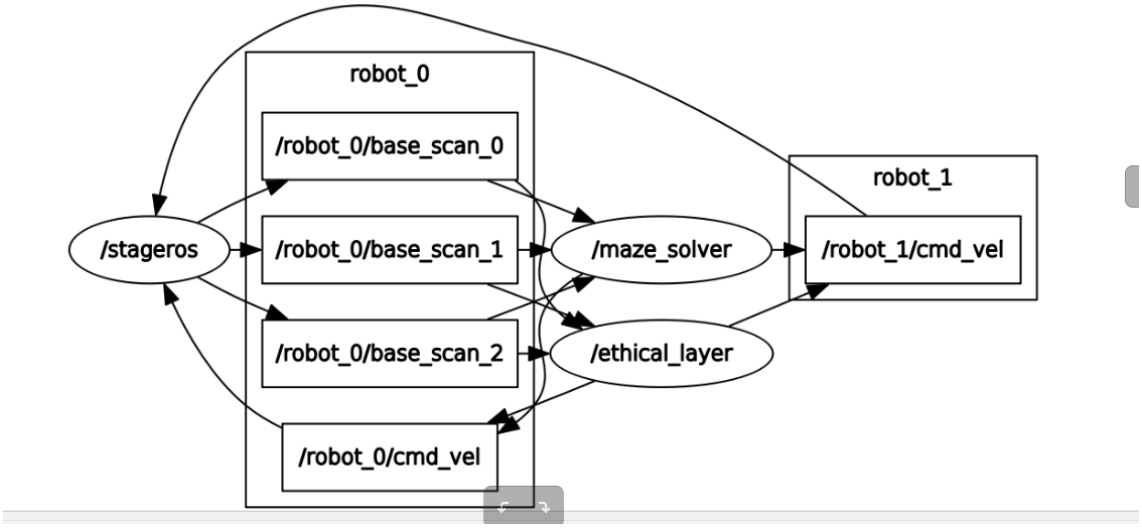

Figura 18 – Diagrama de *nodes* com *node* Ethical_layer

5.3 Teste de comunicação entre dois computadores

Como tentativa de validar a proposta de desenvolvimento do módulo ético para censura controlador robótico, decidimos realizar testes a partir de simulações realizadas em dois computadores conectados entre si por uma rede. Um dos computadores deve executar a simulação que repesente um robô físico (simulação primária), enquanto o outro deve executar uma simulação que represente o módulo de predição e de avaliação do comportamento do robô (simulação secundária).

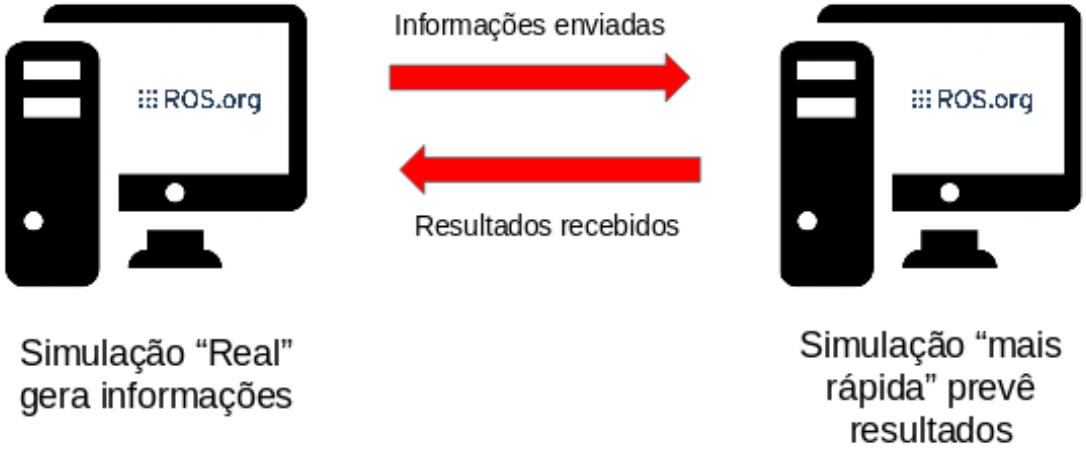

Figura 19 – Representação da comunicação entre os computadores para realização do teste

Para realizar a conexão, utilizamos um cabo *Ethernet crossover* que conectamos na placa de rede de ambos os computadores. Configuramos a conexão estabelecendo IPs de identificação para ambos. Depois disso, instalamos um *client ssh* - um protocolo de rede criptográfico para operação de serviços de rede, usualmente empregado em login remoto de usuários de computadores. Com ele, seríamos capazes de enviar comandos para o terminal da outra máquina, alterando parâmetros de simulação, executando e recebendo de volta os resultados obtidos.

5.4 Modelo inicial de simulação

5.4.1 Criação de um modelo simplificado

Nas seções anteriores, foram apresentados alguns recursos básicos que são essenciais para a execução do projeto. Nesta seção, será tratada a criação de um modelo simplificado de interação entre um robô provido de uma camada ética e um robô que simule um possível descumprimento de regra ética que deve ser impedido. Esse modelo proposto apresenta baixo nível de complexidade de movimentos, já que o objetivo é apenas verificar a plauseabilidade da implementação da camada ética dentro do *Stage*.

Como dito anteriormente, um robô tem a função de possível descumpridor de regras éticas, e será de agora em diante tratado como *humano* ou *robo1*. Nesta primeira etapa, ele tem movimentação unidirecional, seguindo em linha reta a partir da posição inicial. A direção tomada deve ser aleatória, e esse é o fator importante para o possível descumprimento da "regra ética".

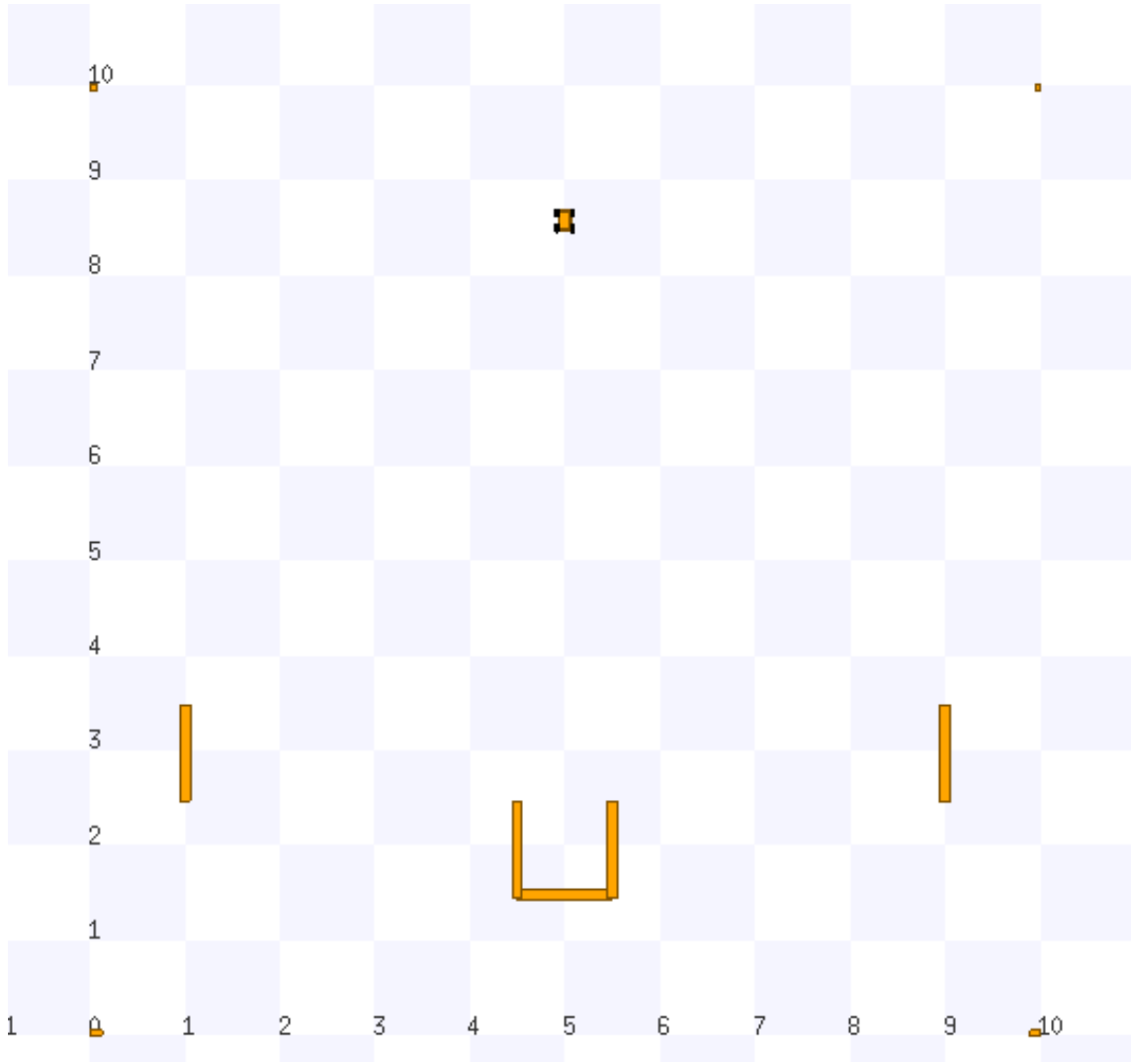

Figura 20 – Ambiente de simulação do Stage com o *humano* posicionado e a área em "U"restrita

A "regra ética"estabelecida é simples: nenhum *humano* pode entrar na região delimitada pelas paredes em forma de "U"que se situa entre as coordenadas - notação [x, y] - [4.5, 1.5], [4.5, 2.5], [5.5, 1.5] e [5.5, 2.5]. O *humano* poderá seguir em linha reta para qualquer direção, desde que não entre na região acima descrita.

O robô provido de camada ética tem em sua programação a seguinte tarefa: andar da parede posicionada na coordenada x = 1 até a parede posicionada na coordenada x = 9, parando em frente a mesma, sem tocá-la. Para isso, conta com um sensor de distância posicionado em sua dianteira. Essa tarefa tem a função de simular o funcionamento rotineiro do robô, que pode ou não ser controlado por uma IA. Além disso, terá em sua camada ética uma intervenção, programada para, em caso de possível descuprimento da "regra ética"estabelecida, movimentar o robô para a frente da região delimitada, a fim de protege-lá. Este robô sera chamado de agora em diante de *robô ético* ou *robot0*. Além disso, a ação executada pelo *robô ético* de impedir que a "regra ética"quebrada será chamada de

ação ética.

5.4.2 Descrição da implementação

Com o problema descrito, esta seção trata do método de implementação. De maneira breve, foi feito o seguinte:

1. Em um computador, foi implementada a simulação "real", que no caso, seria o ambiente onde desejamos que as regras éticas sejam obedecidas;
2. O *robô ético* inicia sua tarefa normalmente, enquanto o *humano* também incia sua movimentação em uma direção;
3. O *robô ético* identifica a direção em que o humano se desloca e envia essa informação para um novo computador;
4. Este computador simula o movimento do humano em uma velocidade muito maior, podendo assim prever o resultado da ação;
5. O resultado é devolvido ao *robô ético* no primeiro computador, que define se deve ou não intervir na trajetória do *humano*.

A seguir, será apresentado o detalhamento dessa implementação, com os aspectos técnicos necessários para fazê-la.

5.4.3 Detalhamento da solução

Primeiramente devem ser apresentadas as simplificações e escolhas feitas para a simulação principal. Como já dito anteriormente, o ROS funciona por um conjunto de processos (*nodes*) que controlam o funcionamento da simulação. Para essa simulação, três *nodes* serão responsáveis pelo funcionamento dos robôs:

- *Human move*: Responsável pela movimentação do *humano*;
- *Robot controler*: Responsável pela movimentação típica do *robô ético*;
- *Ethical layer*: Responsável por prever a movimentação do *humano* e movimentar o *robô ético* se necessário.

O *Human move* é o node mais simples implementado. Ele apenas conecta-se ao *humano* publicando seu comando de velocidade e fazendo-o andar em linha reta.

O *Robot controler* é significativamente mais complexo. Em seu *loop principal*, temos implementada uma máquina de estados, com os dois estados possíveis do controlador robótico.

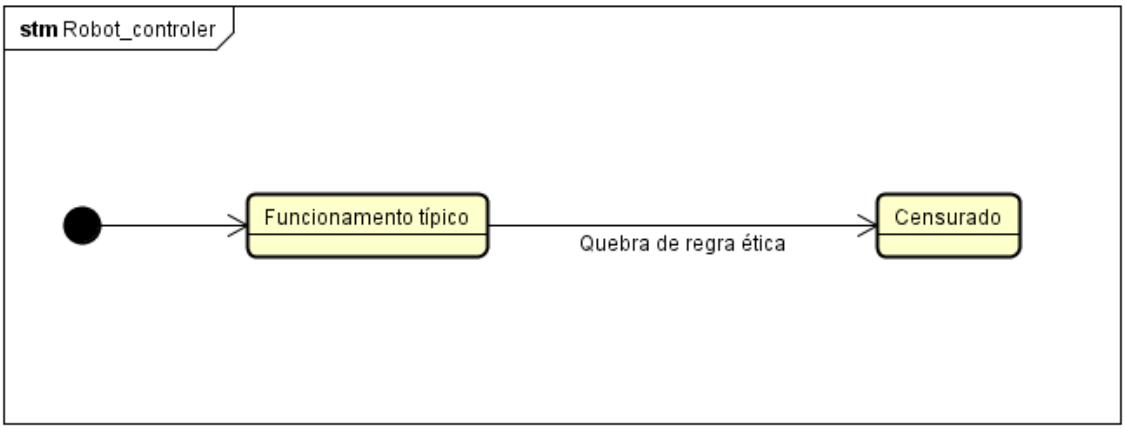

Figura 21 – Máquina de estados do Robot controler

O primeiro estado é o funcionamento típico do *robô ético*, programado então para avançar até chegar muito próximo da parede a sua frente. O segundo estado é o de "intervenção"ou estado "censurado", onde o *node* deixa de publicar comandos de movimentação para o *robô ético*, permitindo assim a ação do *ethical layer*. O intertravamento entre esses dois estados é feito por um *topic* onde o *Robot controler* é *subscriber* e o *Ethical layer* é o *publisher*, ou seja, o *Robot controler* recebe uma *flag* da camada ética indicando que deve haver uma mudança de estados.

Há, nesse ponto, uma importante discussão a ser feita, já que durante a justificativa do projeto indicamos haver diversas possibilidades de implementação de um controlador ético. O controlado ético deveria funcionar apenas como um comutador de estados e o próprio controlador robótico ser responsável pela *ação ética*? Ou o controlador ético deveria se sobrepor ao controlador robótico e realizar os comandos éticos diretamente nos atuadores?

Ambas as soluções tem seus prós e contras, e podemos citar alguns. A grande vantagem da primeira solução é ter um controlador ético mais simples. Em um caso de uma IA implementada no controlador robótico, ela própria poderia ficar encarregada de realizar a *ação ética*. Por outro lado, dependemos de uma programação específica dentro do controlador robótico. Quando implementamos um controlador ético que realiza diretamente a ação não precisamos alterar o controlador robótico - desde que o mesmo seja capaz de ler a *flag* e comutar seu estado para o estado "censurado- e então podemos aplicá-lo em diversos robôs dferentes, que realizem as mais diversas ações e com facilidade, alterar as regras éticas, sem precisar conhecer o comportamento usual do robô. Decidimos por essa última alternativa, por acreditar na importância de que a portabilidade da camada

ética seja o mais simples possível e por considerarmos esse método mais seguro.

Temos por fim a *Ethical layer*, o mais importante *node* do projeto. Portanto, explicaremos o código de maneira mais detalhada, explicitando os comandos utilizados e suas funções. A máquina de estados tem cinco estados: a partida, a identificação da posição, a predição, a espera e a censura.

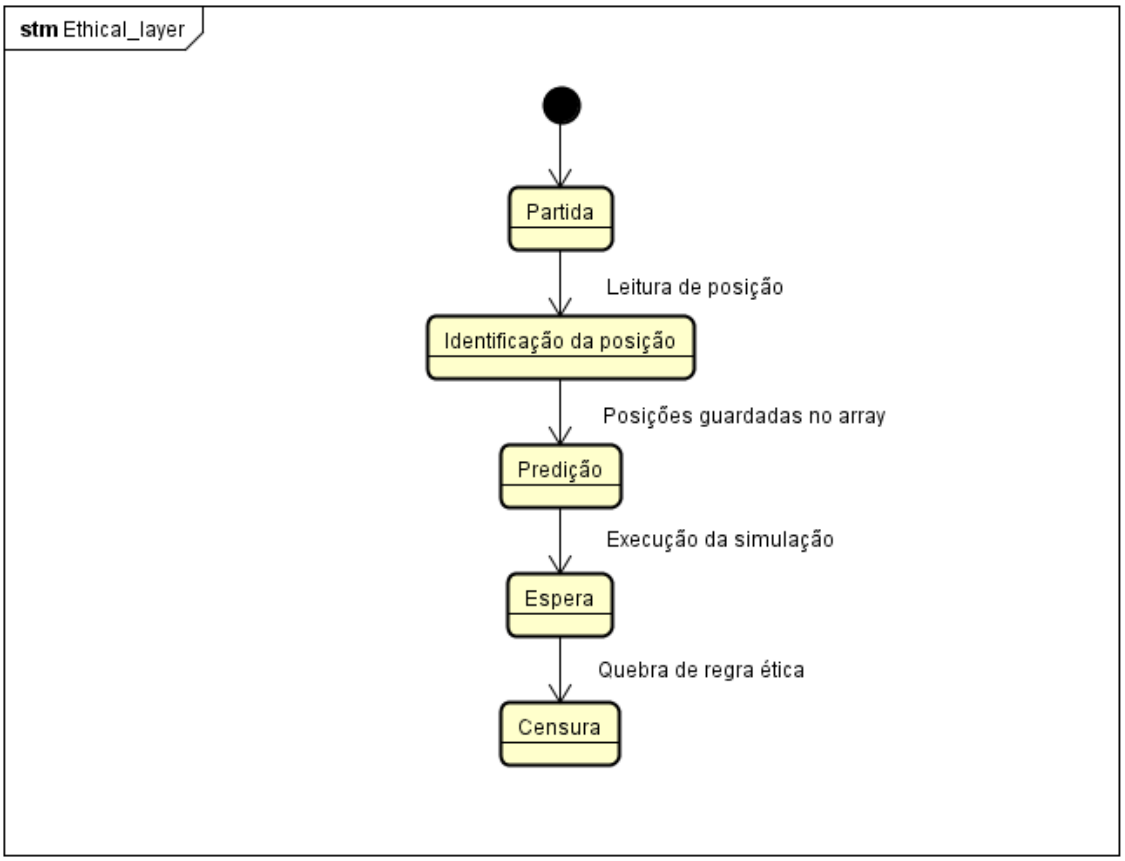

Figura 22 – Máquina de estados do *Ethical layer*

É necessário entrar primeiro em alguns aspectos técnicos da solução. Como dito anteriormente, o *robô ético* deve ser capaz de identificar rápidamente a direção em que se desloca o *humano* para que posse simular suas ações e, para isso, ele precisa ser capaz de medir, com certa precisão, sua posição. A maneira que essa medição deve ser feita depende dos recursos disponíveis e do problema apresentado, já que apesar deste trabalho lidar com um problema hipotético arbitrário, a ideia é que ele reflita com a maior fidelidade possível a realidade.

Idealmente, o *robô ético* seria capaz de obter a posição do *humano* por meio unicamente de seus sensores, sendo assim totalmente independente para tomar as decisões. Nesse caso porém, pela dificuldade criada pelos poucos sensores disponíveis, decidimos adotar a modelagem onde o próprio *humano* tem meios de identificar sua posição e informá-la ao *robô ético*. No caso desta simulação, isso se torna possível graças a existência de um

topic que integra a velocidade dos robôs simulados e pode ser assinado pelos *nodes*. Trazendo essa solução para um problema real, seria como receber as coordenadas por meio de um GPS instalado no *humano*. Essa solução poderia ser adotada para certos usos reais, ou pode ser substituídas por cameras, sensores de distância a laser ou ultrassom, entre muitas outras soluções.

Na máquina de estados temos, então, o estado de partida. Ele existe pois o *topic* que nos fornece a posição dos robôs na simulação não inicia exatamente com o início da simulação. Então, esse estado verifica se houve uma leitura na posição e quando isso acontece, permita a passagem para o próximo estado.

```
case 1:                                 //ESTADO DE PARTIDA
    if (x_pose1 != 0.0){                //Verifica se o sensor de posição já está fazendo as leituras
    state = 2;                          //Se já faz ele pode mudar de estado
    }
    break;
```

Figura 23 – Programação do estado de partida no *Ethical layer*

O próximo estado é responsável por salvar em um *array* duas coordenadas consecutivas do *humano*. Assim, quando as duas medições são feitas, podemos avançar para o próximo, e importantíssimo estado.

```
case 2:                                 //ESTADO DE IDENTIFICAÇÃO DA POSIÇÃO DO HUMANO
    x [amostra] = x_pose1;              //Salva a posição em X
    y [amostra] = y_pose1;              //Salva a posição em Y
    amostra++;
    if (amostra == 2){                  //Ao ter 2 amostras feitas é possível calcular ângulo
    state = 3;                          //Pode mudar de estado
    }
    break;
```

Figura 24 – Programação do estado de identificação de posição no *Ethical layer*

O estado seguinte é o estado de predição, e como o nome indica, ele deve predizer os acontecimentos da simulação "real". Como é o estado mais complexo do sistema, será trata em partes.

```
case 3:                                         //ESTADO DE PREDIÇÃO
    if ((x[1]-x[0]) == 0.0){                    //Se ele não se move em X
            if ((y[1]-y[0]) > 0){               //E se move positivamente em Y
                    angle = 90.0;               //Ângulo configurado
            } else if ((y[1]-y[0]) < 0){        //E se move negativamente em Y
                    angle = -90.0;              //Ângulo configurado
            } else {                            //Está parado
                    angle = 0.0;                //Ângulo configurado
            }
    } else {                                    //Se ele se move em X ((x[1]-x[0])!=0)
            angle = atan((y[1]-y[0])/(x[1]-x[0]))*180/3.1415; //Ângulo calculado
    }
```

Figura 25 – Programação que calcula o ângulo de movimentação no *Ethical layer*

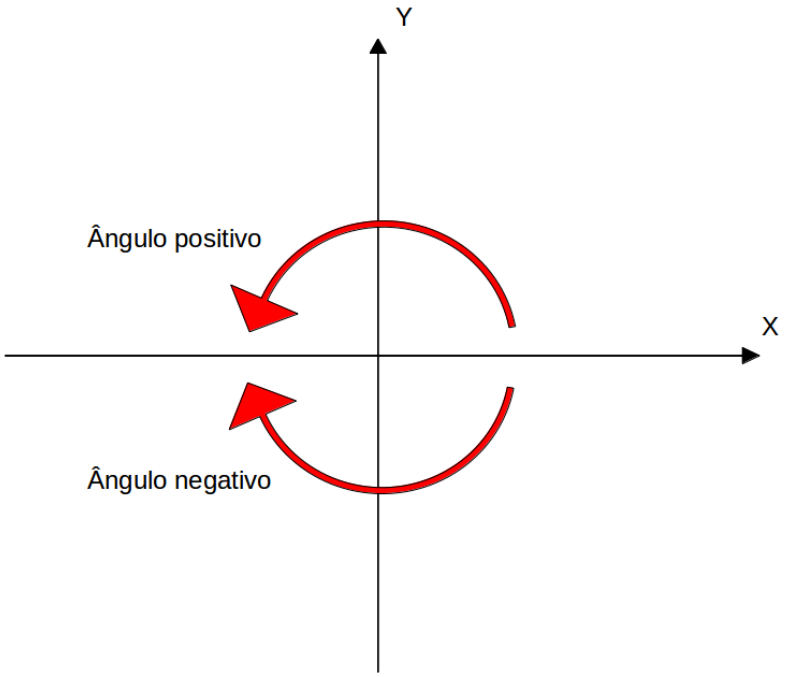

Figura 26 – Diagrama que relaciona o ângulo à direção de partida

Para calcular o ângulo α tendo as coordenadas em X e Y usa-se a seguinte fórmula:

$$\alpha = \tan \text{inverse} \frac{Y_{final} - Y_{inicial}}{X_{final} - X_{inicial}} * \frac{180}{\pi}$$

Portanto precisamos tratar os casos onde não há diferença entre a posição inicial e a final em X. Depois de calculado o ângulo, o *Ethical layer* tem então as informações necessárias para fazer a predição.

Como já mostrado anteriormente, na figura X , o arquivo de criação do ambiente de simulação no formato .world contém a importação do robô. Nessa importação, indicamos a coordenada inicial da seguinte forma: [x y z α]. A princípio, para essa modelagem, não é necessário alterar as coordenadas x, y e z, já que consideramos a posição inicial fixa.

```
//Transformação das linhas de codigo transformadas em array de chars para incluir a variavel angle
//"gnome-terminal -x sh -c" -> Abre terminal
//"cd /home/vitoram" -> Acessa pasta pessoal no PC1
//"echo \"robotcraft_robot1(pose [ 5 10  0.000 %.3f ] name @robotcraft_robot1@)\" >> robotcraft2017_maze.world"
//  -> Imprime o ângulo calculado no arquivo de criação do ambiente de simulação
//"./script1.sh" -> Roda um script no sistema
sprintf (a, "gnome-terminal -x sh -c 'cd /home/vitoram; echo \"robotcraft_robot1(pose [ 5 10  0.000 %.3f ] name @robotcraft_robot1@)\"
>> robotcraft2017_maze.world; ./script1.sh'", angle);
const char *d []= {a};
//Código executado no terminal do linux
system(d[0]);
```

Figura 27 – Programação da alteração do arquivo .world no *Ethical layer*

Algumas adaptações tiveram que ser feitas nas linhas acima para garantir o funcionamento do programa. O comando utilizado para que o *node* interaja com o sistema operacional é o *system()*. Porém, ele só aceita *const char** como entrada, e então é preciso colocar o comando com a variável *angle* dentro de uma variavel desse tipo. Outra adaptação importante feita está no trecho:

"... name @robotcraft_robot1@..."

A importação requer que esses arrobas na verdade sejam aspas. Porém, os arrobas foram utilizados pois, no comando *system()*, havia aspas dentro de aspas, o que impediam o correto compilamento do código. Por isso, temos como ultimo comando dentro do *system()* a execução de um script que troca os arrobas dentro do arquivo .world criado.

```
#!/bin/bash

#Substitui os @ por "
#Esse movimento é necessário pela dificuldade em lançar códigos no terminal com aspas dentro de aspas
sed -i 's/@/"/g' robotcraft2017_maze.world
```

Figura 28 – Script que executa a troca dos *arrobas* pelas *aspas*

```
)

#include the definition of the robot:
include "robotcraft_robot1.inc"

robotcraft_robot1(pose [ 5 10  0.000 -90.000 ] name "robotcraft_robot1")
```

Figura 29 – Linha acrescentada no .world que indica o ângulo de movimentação (no caso -90.000)

O comando seguinte feito com o *system()* tem a função de enviar ao outro computador o arquivo .world criado, estabelecer conexão entre os terminais, compilar o código e rodá-lo.

```
        //Código executado no terminal do linux
        system(d[0]);
        //Código executado no terminal do linux
        //"gnome-terminal -x sh -c" -> Abre terminal
        //"cd /home/vitoram" -> Acessa pasta pessoal no PC1
        //"scp /home/vitoram/robotcraft2017_maze.world vitor@10.0.0.1:/home/vitor/catkin_ws/src/robotcraft2017_maze/world"
        //  |-> Envia o arquivo gerado com o ângulo calculado para a pasta de configuração da simulação no PC2
        //"./script1.sh" -> Roda um script no sistema
        system("gnome-terminal -x sh -c 'cd /home/vitoram; scp /home/vitoram/robotcraft2017_maze.world vitor@10.0.0.1:/home/vitor/catkin_ws/src/
robotcraft2017_maze/world;./script2.sh; exec bash'");
```

Figura 30 – Programação da interação do *Ethical layer* com o segundo computador

A simulação no segundo computador será então responsável por predizer as ações do *humano* , avaliar os resultados e, em caso de violação das "regras éticas"mudar o estado do *Ethical layer*. No caso, o estado de censura é o "0"que é o valor enviado pelo outro computador.

O funcionamento desta segunda simulação será abordado posteriormente. Mas alguns aspectos devem ser ressaltados. Nessa modelagem, a única variável recebida da simulação "real"é o ângulo de movimentação. O mapa, as restrições, a posição inicial do *humano* etc. já estão todas pré programadas no segundo computador. Mas isso se dá única e exclusivamente para simplificar está primeira modelagem.

Fica claro que é possível que essas e muitas outras variáveis sejam enviadas para o simulador, de forma que essa predição seja mais flexível. Da mesma forma que acrescentamos linhas de códigos e substituímos termos para posicionar o robô no ângulo certo, podemos usar esses recurso para alterar ainda mais o ambiente de predição, resolvendo problemas mais complexos.

O *Ethical layer* não executará mais nenhuma ação, a menos que a simulação de predição indique que há uma violação das regras. Nesse caso, é acionado o estado de censura. O *Ethical layer* altera a *flag* enviada para o *Robot controler* colocando-o também em estado de censura. A partir disso, o *Ethical layer* passa a controlar a movimentação do *robô ético*, que é posicionado de modo a defender a região restrita.

Esses são os *nodes* executados no comptador principal. Assim como a simulação principal, a simulação preditiva também reque seus *nodes*. No caso, apenas um foi implementado e chamado de *simulação*. Sua implementação se da na forma dos seguintes estados:

1. Partida: assim como nos outros casos, é necessário esperar que seja possível ler a posição do robô;

2. Simulação: O robô avança em linha reta, na direção que foi posicionado. Caso entre na área restrita, mudamos para o estado de censura. Caso o robô se afaste muito das coordenadas iniciais sem entrar na área restrita, sabemos que a "regra ética"não foi violada e vamos para o estado de finalização;

3. Censura: Esse estado envia a flag para o *Ethical layer* configurando-o no estado de censura. Além disso, esse estado apaga a linha que importa o robô que foi criado pelo *Ethical layer*. Isso permite que a simulação posse ser feita novamente sem ser necessário alterar o arquivo .world.

4. Finalização: Também apaga a linha de importação do robô, mas sem mudar o flag do *Ethical layer*;

5. Fim da simulação.

A maioria dessas ações utiliza recursos já apresentados anteriormente, portanto, não vale a pena detalhar o código novamente. Deve-se apenas destacar a maneira com que

a flag é enviada para o *Ethical layer*. Ao passar a informação do ângulo de um computador para outro, utilizamos a transferência de arquivos e pudémos compilar a simulação para atualizar as mudanças. Durante a volta da informação para o computador principal, isso não é possível, já que a simulação está em andamento. Por isso foi necessário publicar a mensagem diretamente no *topic result*, com o seguinte comando:

rostopic pub /result std_msgs/Int8 ̈data: 0¨

Então, temos a configuração dos *nodes* nas duas simulações.

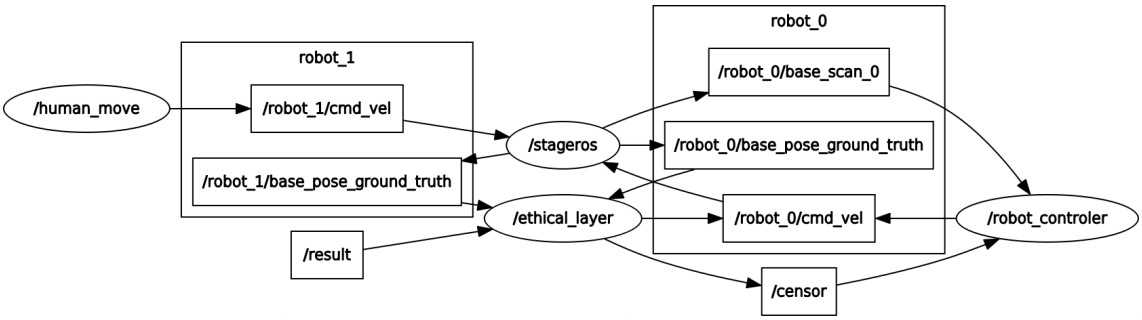

Figura 31 – Diagrama dos *nodes* da simulação no computador principal

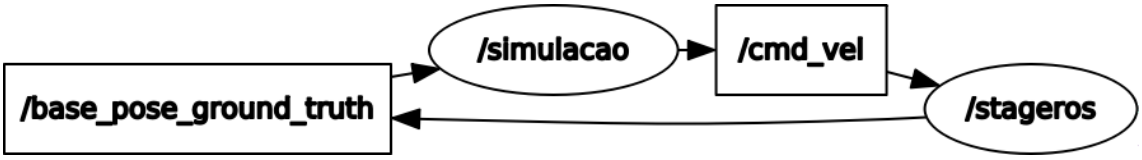

Figura 32 – Diagrama dos *nodes* da simulação no computador secundário

5.5 Resultados

Esta seção tratá da execução da simulação que foi construida na seção anterior. Esta seção tem como o objetivo, portanto, validar a solução adotada para a problemática escolhida.

Primeiramente precisamos definir o ângulo a ser simulado. Pelas posições escolhidas para a área restrita e para o robô *humano*, alguns ângulos são muito pertinentes para testes, como o caso do -90.0 graus, que o direciona diretamente à área restrita.

```
#include the definition of the robot:
include "robotcraft_robot0.inc"
include "robotcraft_robot1.inc"

robotcraft_robot0(pose [ 1.4  3  0.000  0.000 ] name "robotcraft_robot0")
robotcraft_robot1(pose [ 5  10  0.000  -90.000 ] name "robotcraft_robot1")
```

Figura 33 – Definição das posições iniciais dos robôs na simulação "real"com o ângulo do *humano* definido em -90.0 graus

Com o ângulo escolhido, compilamos as alterações feitas.

```
vitoram@vitoram-X556URK:~$ roscd
vitoram@vitoram-X556URK:~/catkin_ws$ catkin_make
Base path: /home/vitoram/catkin_ws
Source space: /home/vitoram/catkin_ws/src
Build space: /home/vitoram/catkin_ws/build
Devel space: /home/vitoram/catkin_ws/devel
Install space: /home/vitoram/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/vitoram/catkin_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/vitoram/catkin_ws/build"
####
[ 12%] Building CXX object robotcraft2017_maze/CMakeFiles/ethical_layer.dir/src/ethical_layer.cpp.o
[ 25%] Building CXX object robotcraft2017_maze/CMakeFiles/human_move.dir/src/human_move.cpp.o
[ 37%] Building CXX object robotcraft2017_maze/CMakeFiles/robot_controler.dir/src/robot_controler.cpp.o
[ 50%] Building CXX object relaxed_astar/CMakeFiles/relaxed_astar_lib.dir/src/RAstar_ros.cpp.o
[ 62%] Linking CXX executable /home/vitoram/catkin_ws/devel/lib/robotcraft2017_maze/human_move
[ 62%] Built target human_move
[ 75%] Linking CXX executable /home/vitoram/catkin_ws/devel/lib/robotcraft2017_maze/ethical_layer
[ 87%] Linking CXX executable /home/vitoram/catkin_ws/devel/lib/robotcraft2017_maze/robot_controler
[ 87%] Built target ethical_layer
[ 87%] Built target robot_controler
[100%] Linking CXX shared library /home/vitoram/catkin_ws/devel/lib/librelaxed_astar_lib.so
[100%] Built target relaxed_astar_lib
vitoram@vitoram-X556URK:~/catkin_ws$
```

Figura 34 – Execução do comando *catkin_make* que compila a programação da simulação

E então, a simulação está pronta para ser executada. O comando de início da simulação é o último a ser executado pelo usuário. A partir dele, todos os comandos são executados automaticamente pela simulação, de acordo com suas necessidades.

```
vitoram@vitoram-X556URK:~/catkin_ws$ roslaunch robotcraft2017_maze maze.launch
... logging to /home/vitoram/.ros/log/8d9953c4-eb59-11e8-8d94-6045cbb840d9/roslaunch-vitoram-X556URK-9263.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://vitoram-X556URK:41637/

SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
  /
    ethical_layer (robotcraft2017_maze/ethical_layer)
    human_move (robotcraft2017_maze/human_move)
    robot_controler (robotcraft2017_maze/robot_controler)
    stageros (stage_ros/stageros)

auto-starting new master
process[master]: started with pid [9273]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 8d9953c4-eb59-11e8-8d94-6045cbb840d9
process[rosout-1]: started with pid [9286]
started core service [/rosout]
process[stageros-2]: started with pid [9293]
/opt/ros/kinetic/lib/python2.7/dist-packages/roslib/packages.py:447: UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode - interp
reting them as being unequal
  if resource_name in files:
process[robot_controler-3]: started with pid [9304]
process[ethical_layer-4]: started with pid [9310]
process[human_move-5]: started with pid [9327]
```

Figura 35 – Comando para a execução da simulação

Inicia-se então a simulação, como o *humano* e o *robô ético* nas posições previamente definidas. Nesse momento *humano* controlado pelo *human move* se move exclusivamente para frente, como acontecerá durante toda a simulação. Já o *robô ético* é controlado pelo *robot controler* e tem como tarefa avançar até a parede que se encontra à sua frente e permanecer parado diante dela.

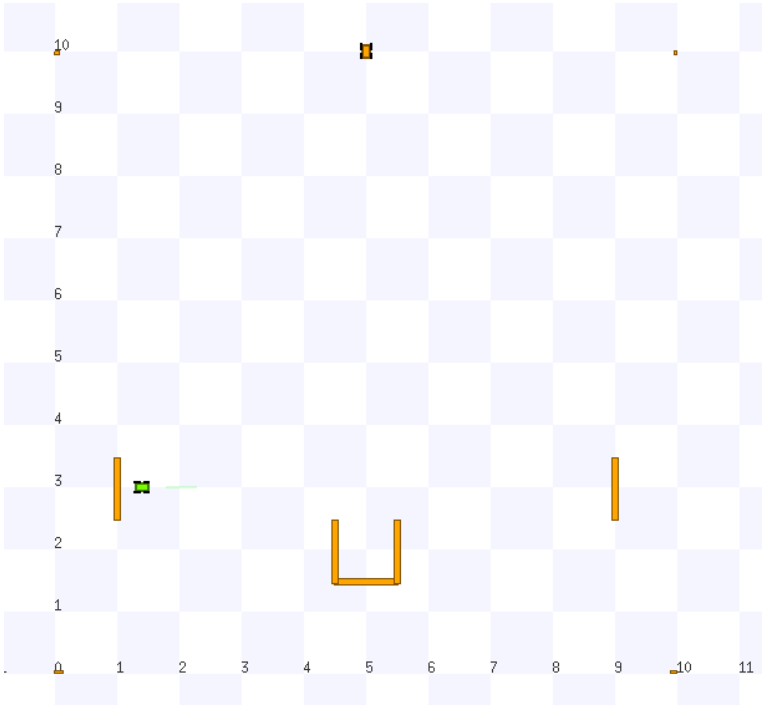

Figura 36 – Posição inicial do *humano* e do *robô ético* na ambiente de simulação

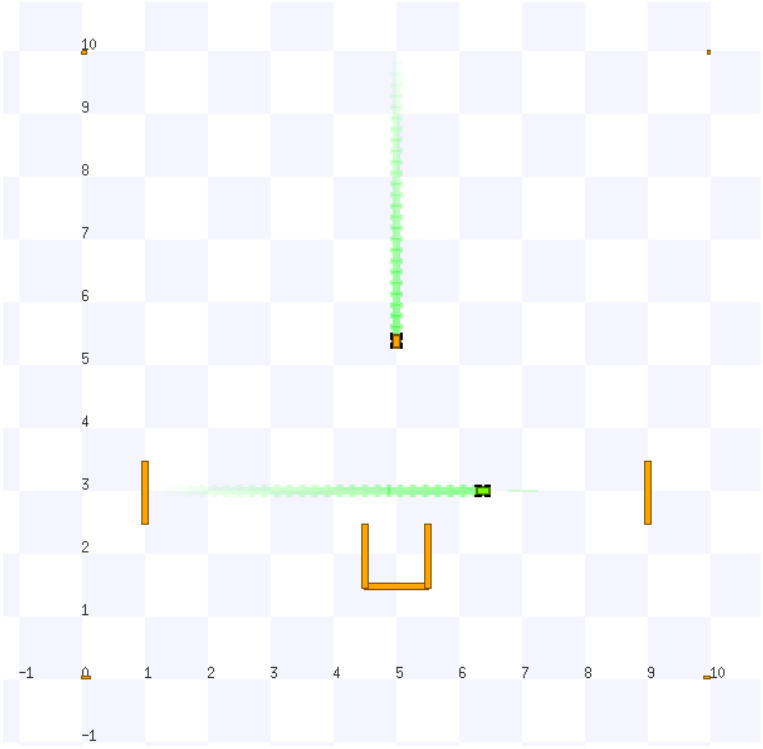

Figura 37 – Início da movimentação programada com as *footprints*

A partir do momento em que a movimentação é detectada no *ethical layer*, ele é capaz de calcular o ângulo e informá-lo ao computador secundário. Como dito anteriormente, a linha de importação do robô é acrescentada ao fim de um arquivo .world, já com o ângulo correto e o arquivo é enviado para a pasta "World"da simulação no segundo computador.

```
robotcraft2017_maze.world                     100%  677     0.7KB/s   00:00
spawn ssh vitor@10.0.0.1
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-31-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

196 packages can be updated.
8 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Nov 18 15:55:35 2018 from 10.0.0.2
vitor@vitor-270E5G-270E5U:~$ export DISPLAY=:0
```

Figura 38 – Estabelecimento da comunicação entre os computadores e envio do arquivo .world para o computador secundário

Então é feita a compilação das alterações.

```
vitor@vitor-270E5G-270E5U:~$ roscd
vitor@vitor-270E5G-270E5U:~/catkin_ws$ catkin_make
Base path: /home/vitor/catkin_ws
Source space: /home/vitor/catkin_ws/src
Build space: /home/vitor/catkin_ws/build
Devel space: /home/vitor/catkin_ws/devel
Install space: /home/vitor/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/vitor/catkin_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/vitor/catkin_ws/build"
####
[100%] Built target simulacao
```

Figura 39 – Alterações compiladas no computador secundário

E a simulação de predição pode então ser executada.

```
vitor@vitor-270E5G-270E5U:~/catkin_ws$ roslaunch robotcraft2017_maze maze.launch... logging to /home/vitor/.ros/log/b3efcc72-eb56-11e8-9b4b-24f
5aaef25ba/roslaunch-vitor-270E5G-270E5U-4486.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://vitor-270E5G-270E5U:44797/

SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES
  /
    simulacao (robotcraft2017_maze/simulacao)
    stageros (stage_ros/stageros)

ROS_MASTER_URI=http://localhost:11311

process[stageros-1]: started with pid [4505]
process[simulacao-2]: started with pid [4506]
```

Figura 40 – Execução da simulação preditiva

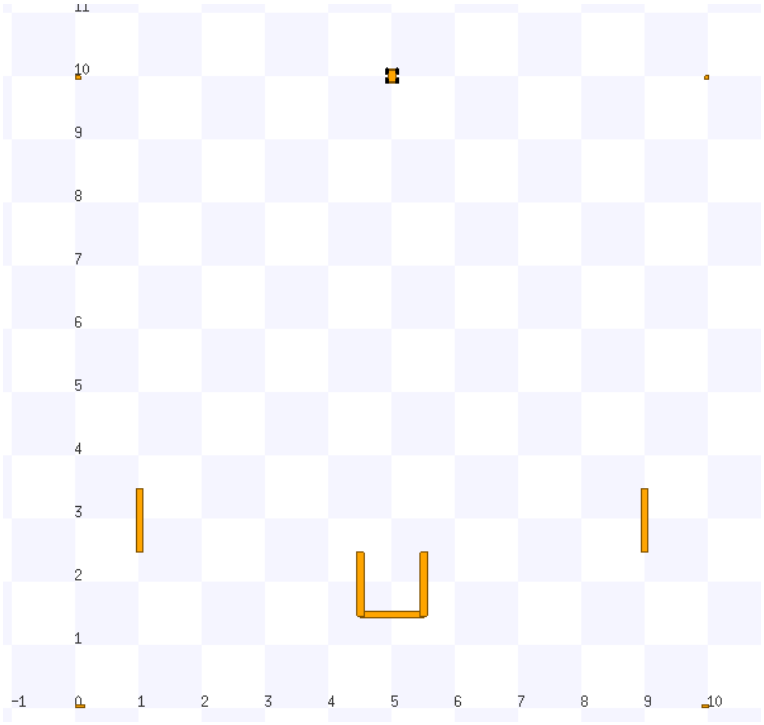

Figura 41 – Posição inicial do simulador de *humano* no ambiente de predição

Então o robô que simula as ações do *humano* assume o mesmo comportamento dele e avança na direção em que foi posicionado. Como esperado, ele entra na região restrita.

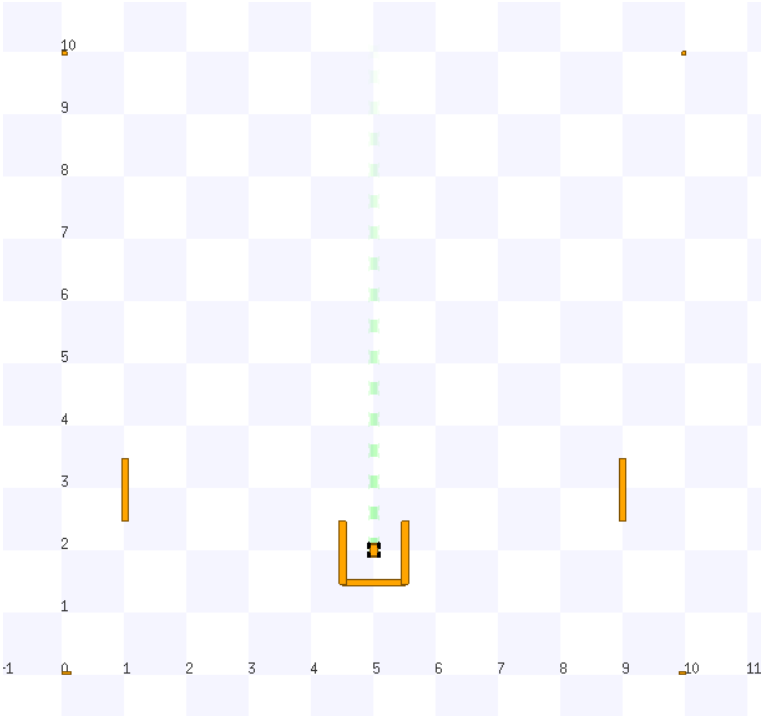

Figura 42 – Simulação do *humano* entra na zona restrita

O *node simulação* então aciona o terminal para enviar por meio do SSH a mensagem de infração da "regra ética".

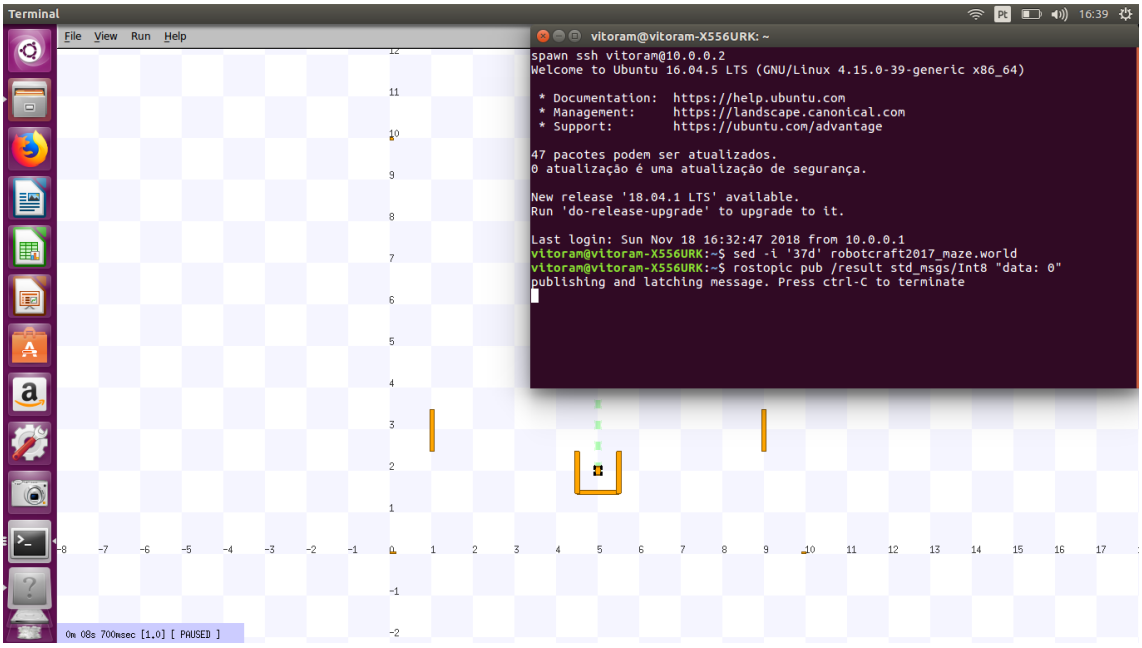

Figura 43 – Terminal aberto pela simulação para comunicar a infração ao computador primário

```
spawn ssh vitoram@10.0.0.2
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-39-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

47 pacotes podem ser atualizados.
0 atualização é uma atualização de segurança.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Nov 18 15:59:21 2018 from 10.0.0.1
vitoram@vitoram-X556URK:~$ sed -i '37d' robotcraft2017_maze.world
vitoram@vitoram-X556URK:~$ rostopic pub /result std_msgs/Int8 "data: 0"
publishing and latching message. Press ctrl-C to terminate
```

Figura 44 – Detalhe dos comandos enviados pelo terminal

Nesse momento, o *Ethical layer* recebe o sinal de infração e, portanto, entra em seu estado censurado, tanto no *robot controler* que deixa de controlá-lo, quanto no *Ethical layer* que assume o controle. A tarefa inicial é abandonada e o *robô ético*, por já ter passado do ponto de defesa da área restrita, recua. Ao atingir o ponto de defesa ($4.8 < X < 5.2$), ele deixa de se mover.

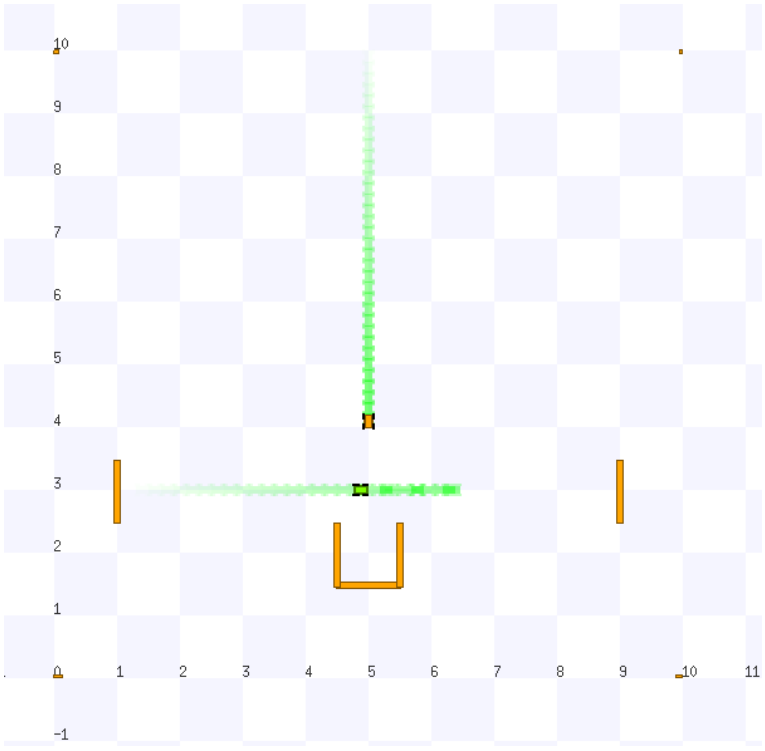

Figura 45 – Intervenção do *Ethical layer* na movimentação do *robô ético*

O *humano* então colide com o *robo ético*, sendo impedido assim de infringir a "regra ética".

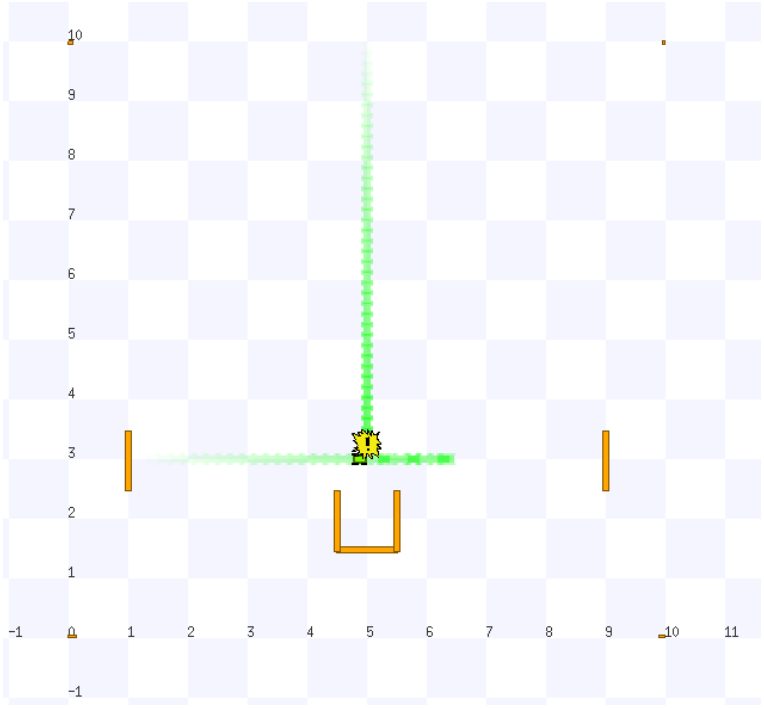

Figura 46 – Colisão impedindo a entrada do *humano* na área restrita

Para ficar claro o efeito obtido com a intervenção do computador secundário, pode-se analisar o diagrama de *nodes* obtido no computador principal. Como podemos ver abaixo, o *topic /robot_1/cmd_vel*, responsável pela movimentação do *humano*, só pode ser alterado pelo *human move*. Já o *topic /robot_0/cmd_vel*, que comanda o *robô ético*, pode ser alterado pelos *nodes ethical layer* e *robot controler*. Entre outras observações a serem feitas, pode-se ver também a comunicação feita pelos *topics censor* e *result*. O primeiro é por onde o *ethical layer* é capaz de colocar o *robot controler* no estado censurado. O segundo é acessado pelo computador secundário e tem a função de colocar o *ethical layer* em estado censurado, e pode ser visto nomeado por *rostopic_7641_1542515045627*.

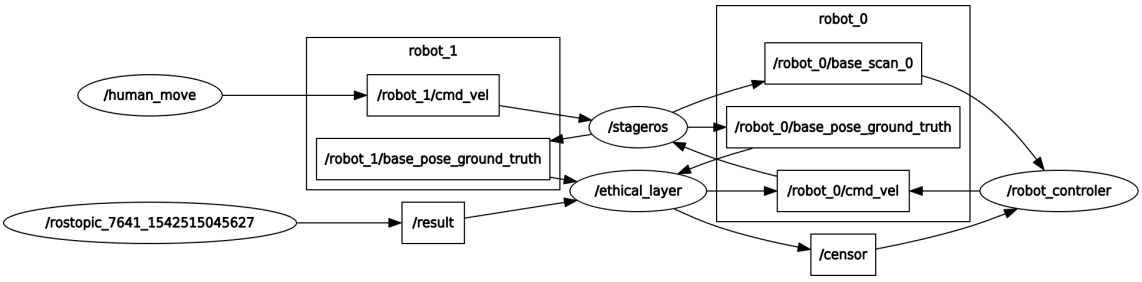

Figura 47 – Diagrama dos *nodes* da simulação no computador principal com intervenção do computador secundário

Portanto, na simulação apresentada acima, foi atingido o objetivo de prever a movimentação do *humano* e impedir qualquer infração da "regra ética". Isso em parte valida a solução planejada. Claro que é necessária novas execuções deste modelo, para se

avaliar os resultados obtidos em outras condições. Isso foi feito e a seguir será documentado um destes testes, feito agora com o objetivo que a regra ética não seja infringida, e portanto, não haja intervenções do *Ethical layer*.

```
#include the definition of the robot:
include "robotcraft_robot0.inc"
include "robotcraft_robot1.inc"

robotcraft_robot0(pose [ 1.4  3  0.000  0.000 ] name "robotcraft_robot0")
robotcraft_robot1(pose [ 5  10  0.000  -30.000 ] name "robotcraft_robot1")
```

Figura 48 – Definição das novas posições iniciais dos robôs na simulação "real"com o ângulo do *humano* definido em -30.0

Com o novo ângulo definido, os processos a serem feitos são os mesmos.

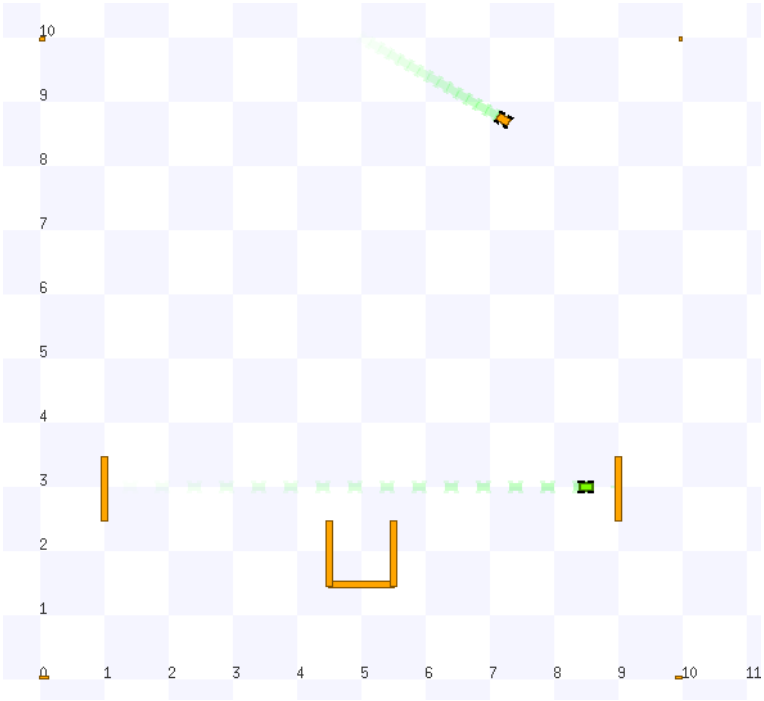

Figura 49 – Movimentação no ambiente de simulação 'real'

Nova simulação preditiva é feita.

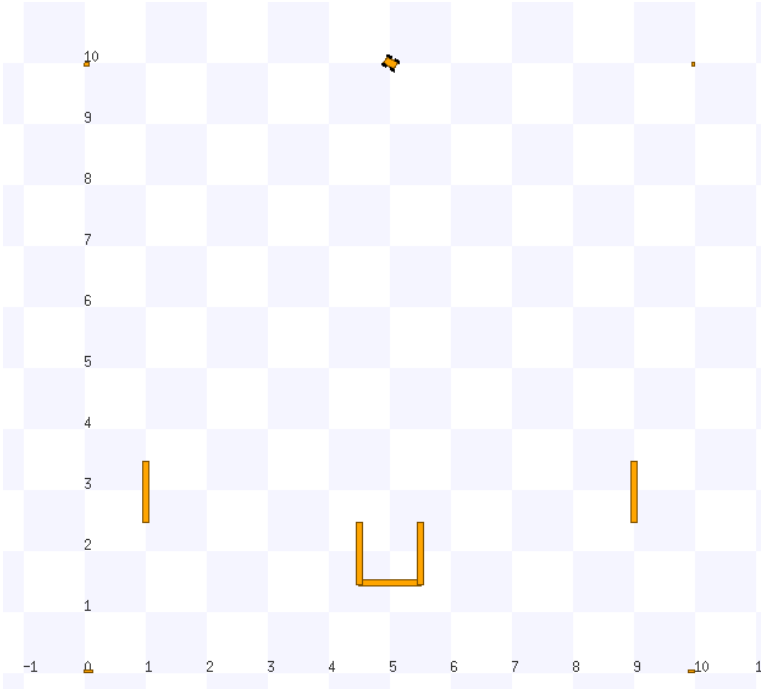

Figura 50 – Posição inicial na simulação preditiva para o ângulo de -30.0

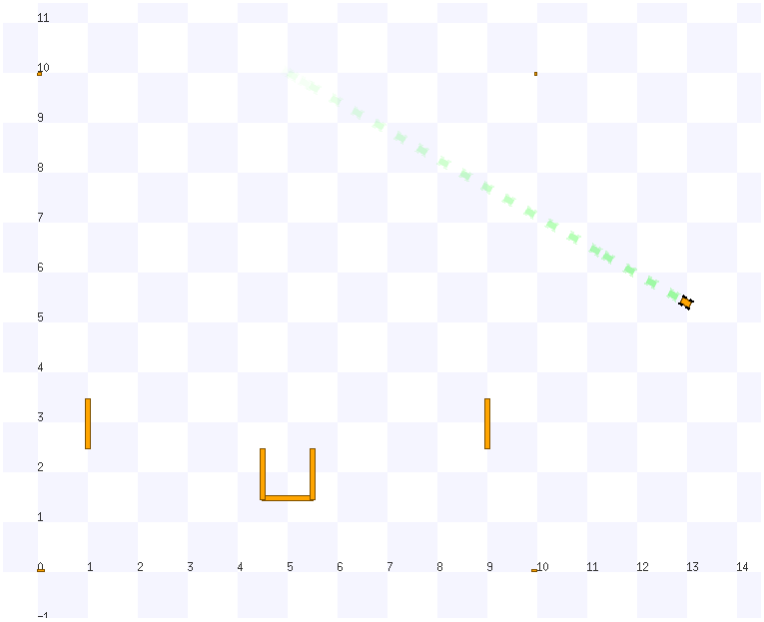

Figura 51 – Simulação *humano* se afasta da área restrita

Decorrido o tempo da simulação preditiva, podemos ver que não há intervenção na simulação "real", como desejado. O *robô ético* para em frente à parede e lá permanece, enquanto o *humano* se afasta da área restrita.

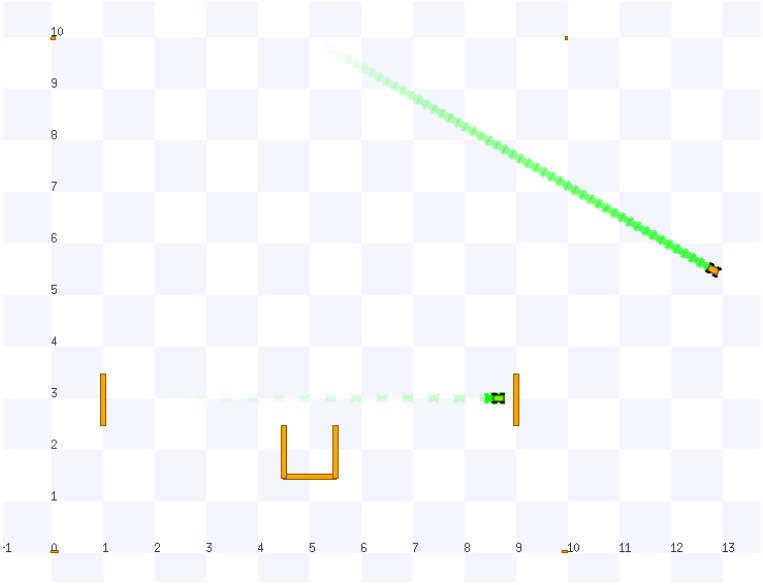

Figura 52 – Simulação "real"sem interferência do *Ethical layer*

6 Conclusão

Em primeiro lugar, gostaríamos de registrar que este trabalho, em meio às dificuldades, sempre nos despertou interesse. A escolha do tema foi um fator que contribuiu bastante para isso, porque sentimos que nossos esforços estão sendo direcionados para contribuir, ainda que de forma modesta, com soluções que podem vir a ser bastante úteis ao lidar com problemas contemporâneos na área de Ética Robótica.

Além disso, por se tratar de um campo multidisciplinar, abrem-se portas para conhecimentos com os quais tivemos pouco contato ao longo da graduação. Como estudantes de engenharia, sentimos falta de uma formação humana, que busque conscientizar-nos de nossas responsabilidade enquanto futuros engenheiros devido ao potencial de impacto social das atividades que nos cabem. A partir desse ponto de vista, ressaltamos que o trabalho contribuiu para que conhecêssemos um pouco sobre um campo de estudo com o qual, em geral, não temos contato na Poli, de modo que expandimos nossa mentalidade para além dos condicionamentos aos quais, na maioria das vezes, ficamos sujeitos ao cursar as disciplinas do curso de Mecatrônica.

Olhando para o desenvolvimento do trabalho propriamente dito, ficamos surpresos algumas vezes com o desenvolvimento das ideias a partir do rascunho inicial. No início, não estávamos familiarizados com o Linux e sequer sabíamos da existência do ROS/Stage, que foi a ferramenta apresentada por nosso orientador e por meio da qual as simulações robóticas foram desenvolvidas.

Foi interessante constatar que, sem um amplo conhecimento específico prévio, a partir do direcionamento dado pela orientação do Professor Moscato e guiados por insights de Alan Winfield, entre outras leituras, fomos capazes de definir progressivamente melhor onde pretendíamos chegar com o trabalho. A cada passo, parecia ficar mais claro o que fazer para alcançar nosso objetivo e como fazer, enquanto testávamos abordagens para solucionar nossos problemas.

O resultado nos pareceu satisfatório na medida em que fomos capazes de definir um problema (o da ética robótica), pesquisar uma forma de realizar experimentos que fizessem referência a ele (tomando como referência o "dilema do robô"), assim como uma técnica de solução (a implementação de uma camada ética para realizar a censura do controlador robótico) e propor uma metodologia (a comunicação entre dois computadores rodando as simulações primária e secundária) a partir das qual essas técnicas foram aplicadas para executar testes de validação da abordagem que adaptamos usando a ferramenta Stage do ROS. Essa sistemátização de práticas para construção de conhecimento talvez seja o que há de mais valioso em todo o processo de aprendizagem e de investigação realizado.

Dizemos isso inclusive com base no que a nós foi transmitido pelo Professor Diolino, que demonstra preocupação com educação e com uma "formação humana"do engenheiro politécnico, em suas aulas.

Outra coisa que ficou clara depois de realizar alguns projetos que partiam de etapas bastante fundamentais, como a de modelagem de sistemas, é a necessidade de, ao menos no início, simplificar o modelo conforme possível, sem que ele perca aspectos mais fundamentais que possam prejudicar o resultado de uma simulação ou experimento. Ao trabalhar com engenharia, é inevitável a lida com a complexidade que caracteriza sistemas reais. Partindo de modelos simplificados, é possível verificar mais facilmente possíveis erros ou impacto de escolhas, por exemplo.

Apesar disso, devemos deixar claro que concordamos que alguns algumas simplificações apresentadas nos deixam um pouco distantes da realidade. O tratamento do comportamento de um humano como linear e constante, e de que o robô tem essa informação, é um bom exemplo disso. Este, porém, é um grande problema enfrentado ao se trabalhar com comportamentos humanos. A modelagem e predição feita sobre humanos é muito complexa, e depende de uma quantidade tão grande de variáveis, que foge do escopo de um trabalho de conclusão de curso e de uma graduação em engenharia. De toda forma, é válido discutir a necessidade que as máquinas terão, no futuro, de conhecer profundamente o comportamento humano e como isso pode vir a ser implementado.

Ao desenvolver um projeto, é possível fazer atualizações que vão incrementando o modelo até que se alcance resultados que aproximem suficientemente bem a realidade concreta. Conversas com a Professora Larissa sobre projetos em outra disciplina deixaram esses pontos mais claros e, no fim, percebemos que essa abordagem permitiu que conseguíssemos realizar progressos ao tratar de um problema que tende a ser bastante complexo, principalmente quando se decide trabalhar com tecnologias mais sofisticadas (como as IAs que são citadas no estado da arte).

Consideramos, então, que ainda há espaço para aumentar a complexidade e tentar aproximar nosso modelo da realidade e de aprimorar a solução da construção de controladores éticos.

Por fim, concluímos que o esforço empregado neste trabalho ganhou mais sentido por termos validado, ainda que de forma *ad hoc*, o funcionamento de uma arquitetura de camada ética para simular o censor do controlador robótico. No pior dos casos, mostramos que faz sentido se aprofundar mais na ideia para descobrir até onde ela é promissora.

Referências

WINFIELD, Alan F. T.; BLUM, Christian; LIU, Wenguo. Towards an Ethical Robot: Internal Models, Consequences and Ethical Action Selection. In: **Conference Towards Autonomous Robotic Systems**, 4., 2014, Birmingham, United Kingdom. Lecture Notes in Computer Science... Switzerland: Springer, 2014. p. 85-96. v. 8717. Disponível em: <https://www.researchgate.net/publication/268741533_Towards_an_Ethical_Robot_Internal_Models_Consequences_and_Ethical_Action_Selection>.
Acesso em: 16 nov. 2018.

ALLEN, Colin; WALLACH, Wendell; SMIT, Iva. Why Machine Ethics?. **IEEE Intelligent Systems**, [S.l.], v. 21, n. 4, p. 12-17, jul. 2006. Disponível em: <https://ieeexplore.ieee.org/ document/1667947>. Acesso em: 16 nov. 2018.

BUNGE, Mario. Towards a Technoethics. **Philosophic Exchange**, McGill University, v. 6, n. 1, p. 69-79, 1975. Disponível em: <https://digitalcommons.brockport.edu/phil_ex/vol6/iss1/3/>. Acesso em: 16 nov. 2018.

GRINBAUM, Alexei et al. Ethics in Robotics Research: CERNA Mission and Context. **IEEE Robotics** & **Automation Magazine**, [S.l.], v. 24, n. 3, p. 139-145, set. 2017. Disponível em: <https://ieeexplore.ieee.org/document/7822928>. Acesso em: 16 nov. 2018.

FOOT, Philippa. The Problem of Abortion and the Doctrine of Double Effect. **Oxford Review**, [S.l.], n. 5, p. 5-15, 1967. Disponível em:<http://www2.pitt.edu/~mthompso/readings/foot.pdf>. Acesso em: 16 nov. 2018

GOODALL, Noah J. Can You Program Ethics Into a Self-Driving Car?. **IEEE Spectrum**, [S.l.], p. 6-9, maio. 2016. Disponível em: <https://spectrum.ieee.org/transportation/self-driving/can-you-program-ethics-into-a-selfdriving-car>. Acesso em: 16 nov. 2018.

MOROZOV, Evgeny. **To Save Everything, Click Here: The Folly of Technological Solutionism**. eBook. ed. [S.l.]: PublicAffairs, 2013.

BOSTROM, Nick; YUDKOWSKY, Eliezer. The Ethics of Artificial Intelligence. **Cambridge Handbook of Artificial Intelligence**, [S.l.], 2011. Disponível em: <https://nickbostrom.com/ethics/artificial-intelligence.pdf>. Acesso em: 16 nov. 2018.

BOISBOISSEL, Gérard de. Is it sensible to grant autonomous decision-making to military robots of the future?. In: 2017 International Conference on Military Technologies (ICMT), 2017, Brno, Czech Republic. [S.l.]: IEEE, 2017. Disponível em: <https://ieeexplore

.ieee.org/document/7988854>. Acesso em: 16 nov. 2018.

BRYSON, Joanna; WINFIELD, Alan. Standardizing Ethical Design for Artificial Intelligence and Autonomous Systems. **Computer**, [S.l.], v. 50, n. 5, p. 116-119, maio. 2017. Disponível em: <https://ieeexplore.ieee.org/document/7924235>. Acesso em: 16 nov. 2018.

VANDERELST, Dieter; WINFIELD, Alan. **An architecture for ethical robots**. Bristol, United Kingdom: [s.n.], 2016. 1-15 p. Disponível em: <https://arxiv.org/abs/1609.02931>. Acesso em: 16 nov. 2018.

Apêndices

APÊNDICE A – Código fonte

```
//Ethical Layer

//Inclusão das bibliotecas necessárias
#include "ros/ros.h"                    //Biblioteca padrão do ROS
#include "sensor_msgs/LaserScan.h"      //Biblioteca dos sensores laser
#include "geometry_msgs/Twist.h"        //Biblioteca dos comandos de
velocidade
#include "std_msgs/Int8.h"              //Biblioteca para envio de msgs numéricas
por tópicos
#include "math.h"                       //Biblioteca de funções matemáticas
#include "nav_msgs/Odometry.h"          //Biblioteca de leitura de posição
#include "sstream"

//Variáveis globais
float x_pose0, y_pose0, x_pose1, y_pose1;
int state;

//Leitura das posições em X e Y do robô ético:
void chatterCallback0(const nav_msgs::Odometry::ConstPtr& msg){
        x_pose0 = msg->pose.pose.position.x;
        y_pose0 = msg->pose.pose.position.y;
}

//Leitura das posições em X e Y do robô humano:
void chatterCallback1(const nav_msgs::Odometry::ConstPtr& msg){
        x_pose1 = msg->pose.pose.position.x;
        y_pose1 = msg->pose.pose.position.y;
}

//Leitura flag de mudança estado recebido pelo PC2
void chatterCallback2(const std_msgs::Int8::ConstPtr& msg){
    state = msg->data;
}


int main(int argc, char **argv){

  ros::init(argc, argv, "ethical_layer");
  ros::NodeHandle n;

  //Variáveis locais
  int amostra = 0;
  float angle;
  float x [2] = {0.0, 0.0};
  float y [2] = {0.0, 0.0};
  char a[150];

  //Configuração do canal de subscrição da leitura de posição do robo ético:
  ros::Subscriber r0_sub = n.subscribe("/robot_0/base_pose_ground_truth",100,
chatterCallback0);
  //Configuração do canal de subscrição da leitura de posição do robo humano:
  ros::Subscriber r1_sub = n.subscribe("/robot_1/base_pose_ground_truth",100,
chatterCallback1);
  //Configuração do canal de subscrição da flag de mudança de estado recebida pelo
PC2:
  ros::Subscriber c_sub = n.subscribe("result", 1000, chatterCallback2);
  //Configuração do canal de publicação de comandos de velocidade:
  ros::Publisher cmd0_vel_pub = n.advertise<geometry_msgs::Twist>("/robot_0/
cmd_vel", 100);
  //Configuração do canal de publicação da flag de mudança de estado para o Robot
Controler:
  ros::Publisher chatter_pub = n.advertise<std_msgs::Int8>("censor", 1000);
  ros::Rate loop_rate(5); //5 Hz

  //inicialização dos tópicos:
  geometry_msgs::Twist cmd0_vel_msg;
  std_msgs::Int8 msg;

  //Configuração da flag de mudança de estado na condição inicial (1 - padrão; 2 -
```

```
censurado)
  msg.data = 1;
  //Configuração de estado inicial (1 - padrão; 0 - censurado)
  state = 1;

  while (ros::ok()){
    switch(state){
    case 0:                                         //Estado censurado
        msg.data = 2;                               //Seta a flag de mudança de
estado do Robot Controler
        if (x_pose0 < 4.9) {                        //Se o robo ético está antes do
centro da região
                cmd0_vel_msg.linear.x = 1.0;        //Continua avançando
        } else if (x_pose0 >5.1){                   //Se o robo ético está depois do
centro da região
                cmd0_vel_msg.linear.x = -1.0;       //Recua
        } else {                                    //Se está no centro
                cmd0_vel_msg.linear.x = 0.0;        //Para
        }
        cmd0_vel_pub.publish(cmd0_vel_msg);         //Publicação da msg propriamente
dita
        break;
    case 1:                                         //Estado de partida do Ethical
layer
        if (x_pose1 != 0.0){                        //Verifica se o sensor de
posição já está fazendo as leituras
        state = 2;                                  //Se já faz ele pode mudar de
estado
        }
        break;
    case 2:                                         //Estado de identificação do
ângulo de movimentação do humano
        x [amostra] = x_pose1;                      //Salva a posição em X
        y [amostra] = y_pose1;                      //Salva a posição em Y
        amostra++;
        if (amostra == 2){                          //Ao ter 2 amostras feitas é
possível calcular ângulo
        state = 3;                                  //Pode mudar de estado
        }
        break;
    case 3:                                         //Estado de
Predição
        if ((x[1]-x[0]) == 0.0){                    //Se ele não se move em X
                if ((y[1]-y[0]) > 0){               //E se move positivamente em Y
                        angle = 90.0;               //Ângulo configurado
                } else if ((y[1]-y[0]) < 0){        //E se move negativamente em Y
                        angle = -90.0;              //Ângulo configurado
                } else {                            //Está parado
                        angle = 0.0;                //Ângulo configurado
                }
        } else {                                    //Se ele se move em X ((x[1]-x
[0])!=0)
                angle = atan((y[1]-y[0])/(x[1]-x[0]))*180/3.1415; //Ângulo
calculado
        }

        //Transformação das linhas de codigo transformadas em array de chars para
incluir a variavel angle
        //"gnome-terminal -x sh -c" -> Abre terminal
        //"cd /home/vitoram" -> Acessa pasta pessoal no PC1
        //"echo \"robotcraft_robot1(pose [ 5 10  0.000 %.3f ] name
@robotcraft_robot1@)\" >> robotcraft2017_maze.world" -> Imprime o ângulo calculado
no arquivo de criação do ambiente de simulação
        //"./script1.sh" -> Roda um script no sistema
        sprintf (a, "gnome-terminal -x sh -c 'cd /home/vitoram; echo
\"robotcraft_robot1(pose [ 5 10  0.000 %.3f ] name @robotcraft_robot1@)\" >>
robotcraft2017_maze.world; ./script1.sh'", angle);
        const char *d []= {a};
        //Código executado no terminal do linux
```

```
        system(d[0]);
        //Transformação das linhas de codigo transformadas em array de chars para
incluir a variavel angle
        //"gnome-terminal -x sh -c" -> Abre terminal
        //"cd /home/vitoram" -> Acessa pasta pessoal no PC1
        //"scp /home/vitoram/robotcraft2017_maze.world vitor@10.0.0.1:/home/vitor/
catkin_ws/src/robotcraft2017_maze/world" -> Envia o arquivo gerado com o ângulo
calculado para a pasta de configuração da simulação no PC2
        //"./script1.sh" -> Roda um script no sistema
        system("gnome-terminal -x sh -c 'cd /home/vitoram; scp /home/vitoram/
robotcraft2017_maze.world vitor@10.0.0.1:/home/vitor/catkin_ws/src/
robotcraft2017_maze/world;./script2.sh; exec bash'");
        state = 10;             //Seta um estado não atingível
        break;

    }
    chatter_pub.publish(msg);    //Publica a flag para o Robot
Controler
    ros::spinOnce();
    loop_rate.sleep();
  }

  return 0;
}
```

```
//Human Move

//Inclusão das bibliotecas necessárias
#include "ros/ros.h"                    //Biblioteca padrão do ROS
#include "geometry_msgs/Twist.h"        //Biblioteca dos comandos de velocidade


int main(int argc, char **argv){

  ros::init(argc, argv, "human_move");
  ros::NodeHandle n;

  //Configuração do canal de publicação de comandos de velocidade:
  ros::Publisher cmd1_vel_pub = n.advertise<geometry_msgs::Twist>("/robot_1/
cmd_vel", 100);
  ros::Rate loop_rate(5); //5 Hz


  //inicialização dos tópicos:
  geometry_msgs::Twist cmd1_vel_msg;

  while (ros::ok()){
    cmd1_vel_msg.linear.x = 0.25;             //Configurando a velocidade do humano
    cmd1_vel_pub.publish(cmd1_vel_msg);       //Publicação da msg propriamente dita
    ros::spinOnce();
    loop_rate.sleep();
    }
  return 0;
}
```

```
//Robot controler

//Inclusão das bibliotecas necessárias
#include "ros/ros.h"                    //Biblioteca padrão do ROS
#include "sensor_msgs/LaserScan.h"      //Biblioteca dos sensores laser
#include "geometry_msgs/Twist.h"        //Biblioteca dos comandos de
velocidade
#include "std_msgs/Int8.h"              //Biblioteca para envio de msgs numéricas
por tópicos

//Variáveis globais
double IR_range;
int state;

//Leitura flag de mudança estado recebido pelo Ethical layer
void chatterCallback(const std_msgs::Int8::ConstPtr& msg){
    state = msg->data;
}

//Leitura do sensor laser frontal
void IRCallback(const sensor_msgs::LaserScan::ConstPtr& msg){
    IR_range = msg->ranges[0];
}


int main(int argc, char **argv){

  ros::init(argc, argv, "robot_controler");
  ros::NodeHandle n;

  //Configuração do canal de publicação de comandos de velocidade:
  ros::Publisher cmd0_vel_pub = n.advertise<geometry_msgs::Twist>("/robot_0/
cmd_vel", 100);
  //Configuração do canal de subscrição da leitura do sensor:
  ros::Subscriber IR_sub = n.subscribe("/robot_0/base_scan_0", 100, IRCallback);
  //Configuração do canal de subscrição da flag de estado do Ethical Layer:
  ros::Subscriber sub = n.subscribe("censor", 1000, chatterCallback);
  ros::Rate loop_rate(5); //5 Hz


  //inicialização dos tópicos:
  geometry_msgs::Twist cmd0_vel_msg;

  //Configuração do estado inicial (1 - padrão; 2 - censurado)
  state = 1;

  while (ros::ok()){

    switch(state){
    case 1:                                 //Estado padrão
        if (IR_range < 0.5){                //Se o robô esta a menos de 0.5 da
parede
        cmd0_vel_msg.linear.x = 0.0;        //Ele para de se mover
        }
        else {                              //Caso contrário
        cmd0_vel_msg.linear.x = 1.0;        //Ele permanece em movimento
        }
        cmd0_vel_pub.publish(cmd0_vel_msg); //Publicação da msg propriamente dita
        break;
    case 2:                                 //Estado censurado
        // Nenhuma atitude é tomada e o node deixa de publicar
        // no cmd0_vel_pub para evitar conflitos com o
        // Ethical layer
        break;
    }
    ros::spinOnce();
    loop_rate.sleep();
    }
  return 0;
```

```
}
```

```
#!/bin/bash


#Substitui os @ por "
#Esse movimento é necessário pela dificuldade em lançar códigos no terminal com
aspas dentro de aspas
sed -i 's/@/"/g' robotcraft2017_maze.world
```

```
#!/usr/bin/expect -f


set timeout 60
spawn ssh vitor@10.0.0.1   #Cria conexão SSH entre PC1 e PC2

#Espera a conexão completar e envia o comando Display, que permite abrir
aplicativos no PC2
expect "$ "
send "export DISPLAY=:0\r"
#Espera a execução do código e acessa a pasta da simulação no PC2
expect "$ "
send "roscd\r"
#Espera a execução do código e compila as mudanças feitas
expect "$ "
send "catkin_make\r"
#Espera a execução do código e inicia a simulação
expect "$ "
send "roslaunch robotcraft2017_maze maze.launch\r"
interact
```

```
// Simulação

//Inclusão das bibliotecas necessárias
#include "ros/ros.h"                //Biblioteca padrão do ROS
#include "geometry_msgs/Twist.h"    //Biblioteca dos comandos de velocidade
#include "nav_msgs/Odometry.h"      //Biblioteca de leitura de posição

//Variáveis globais
float x_pose, y_pose;

//Leitura das posições em X e Y do robô simulado:
void chatterCallback(const nav_msgs::Odometry::ConstPtr& msg){
        x_pose = msg->pose.pose.position.x;
        y_pose = msg->pose.pose.position.y;
}

int main(int argc, char **argv){

  ros::init(argc, argv, "simulacao");
  ros::NodeHandle n;

  //Variáveis locais
  int state;

  //Configuração do canal de publicação de comandos de velocidade:
  ros::Publisher cmd1_vel_pub = n.advertise<geometry_msgs::Twist>("/cmd_vel", 100);
  //Configuração do canal de subscrição da leitura de posição do robô simulado:
  ros::Subscriber sub = n.subscribe("/base_pose_ground_truth", 100,
chatterCallback);
  ros::Rate loop_rate(5); //5 Hz

  //inicialização dos tópicos:
  geometry_msgs::Twist cmd1_vel_msg;

  //Configuração de estado inicial
  state = 1;
  while (ros::ok()){
    switch(state){            //Estado de partida
    case 1:
        if (x_pose != 0.0){   //Verifica se o sensor de posição já está fazendo as
leituras
                state = 2;    //Se já faz ele pode mudar de estado
        }
        break;
    case 2:                   //Estado de simulação
        cmd1_vel_msg.linear.x = 1.0;    //Configura a velocidade do robo simulado
que avança no ângulo fornecido
        if (y_pose <= 2.5 & x_pose >= 4.5 & x_pose <= 5.5){ //Se o robô simulado
entra na área restrita
        state = 3;            //Envia para o estado de censura
        }
        if (x_pose > 11.0 || x_pose < -1.0 || y_pose > 11.0 || y_pose < -1.0){  //
Se o robô se distancia sem entrar na área restrita
        state = 4;           //Envia para o estado de finalização
        }
        break;
    case 3:                 //Estado de censura
        //Código executado no terminal do linux
        //Script violado avisa o PC1 da violação da regra ética
        system("gnome-terminal -x sh -c 'cd /home/vitor; ./violado.sh;'");
        state = 10;         //Envia para estado não acessível
        break;
    case 4:                 //Estado de finalização
        //Código executado no terminal do linux
        //Script fim exclui a linha do ângulo calculado para poder fazer novas
simulações
        system("gnome-terminal -x sh -c 'cd /home/vitor; ./fim.sh; '");
        state = 10;         //Envia para estado não acessível
        break;
```

```
    }
    cmd1_vel_pub.publish(cmd1_vel_msg); //Publica o comando de velocidade no robô
simulado
    ros::spinOnce();
    loop_rate.sleep();
  }

  return 0;
}
```

```
#!/usr/bin/expect -f

set timeout 60
spawn ssh vitoram@10.0.0.2 #Conecta com o PC1

#Espera a conexão e exclui linha com ângulo calculado
expect "$ "
send "sed -i '37d' robotcraft2017_maze.world\r"
#Espera o código executar e envia uma flag de mudança de estado para o Ethical
layer
expect "$ "
send "rostopic pub /result std_msgs/Int8 \"data: 0\"\r"
interact
```

```
#!/usr/bin/expect -f

set timeout 60
spawn ssh vitoram@10.0.0.2  #Conecta com o PC1

#Espera a conexão e exclui linha com ângulo calculado
expect "$ "
send "sed -i '37d' robotcraft2017_maze.world\r"
interact
```

```
########################################
#     Construir ambiente de simulação     #
########################################

define floorplan model (
    color "orange"
    boundary 0
    gui_nose 1
    gui_grid 0
    gui_move 0
    gui_outline 0
    gripper_return 0
    fiducial_return 0
)


window(
  size [ 1100 716 1]
  rotate [ 0.000 0.000]
  center [ 5.000 5.000 0 ]
  scale 50.000                  # zoom
  show_data 1
  show_clock 1
)


floorplan (
  size [10.000 10.000 1.000]
  pose [5.000 5.000 0.000 0.000]

  bitmap "ambiente.png"
)


include "robotcraft_robot0.inc"
include "robotcraft_robot1.inc"

robotcraft_robot0(pose [ 1.4  3  0.000  0.000 ] name "robotcraft_robot0")
robotcraft_robot1(pose [ 5  10  0.000  -90.000 ] name "robotcraft_robot1")
```

```
########################################
#          Construir robo ético        #
########################################


# Sensor frontal
define IR_sensor ranger(

  sensor(
    range [ 0.1  0.8 ]
    fov 4.0
    samples 1
  )

  color "black"
  size [ 0.019 0.045 0.014 ]

  block( points 4
     point[0] [0 0]
     point[1] [0 1]
     point[2] [1 1]
     point[3] [1 0]
     z [0 1]
  )
)


#Sensor lateral em 0 pois é necessário mante-lo
define sonar ranger(

  sensor(
    range [ 0 0 ]
    fov 60.0
    samples 1
  )

  size [ 0.018 0.02 0.024 ]

  block(
     color "black"
     points 4
     point[0] [0.002 0.018]
     point[1] [0.018 0.018]
     point[2] [0.018 0.002]
     point[3] [0.002 0.002]
     z [0.004 0.02]
  )

  block(
     color "SeaGreen"
     points 4
     point[0] [0 0]
     point[1] [0 0.02]
     point[2] [0.002 0.02]
     point[3] [0.002 0]
     z [0 0.024]
  )

)

define robotcraft_robot0 position(

  odom_error [0.03 0.03 0.00 0.05]
  localization "odom"
  gui_nose 1
  drive "diff"
  color "green"

  size [ 0.24 0.195 0.13 ]
```

```
#robot's main body:
block(
   color "LawnGreen"
   points 4
   point[0] [0.225 0.03]
   point[1] [0.225 0.165]
   point[2] [0.015 0.165]
   point[3] [0.015 0.03]
   z [0.02 0.065]
)

#robot's top surface:
block(
   color "LawnGreen"
   points 4
   point[0] [0.215 0.04]
   point[1] [0.215 0.155]
   point[2] [0.025 0.155]
   point[3] [0.025 0.04]
   z [0.129 0.13]
)

#front left wheel - midblock:
block(
   color "black"
   points 4
   point[0] [0.225 0.165]
   point[1] [0.225 0.195]
   point[2] [0.18 0.195]
   point[3] [0.18 0.165]
   z [0 0.085]
)

#front left wheel - left part:
block(
   color "black"
   points 4
   point[0] [0.245 0.165]
   point[1] [0.245 0.195]
   point[2] [0.225 0.195]
   point[3] [0.225 0.165]
   z [0.02 0.065]
)

#front left wheel - right part:
block(
   color "black"
   points 4
   point[0] [0.18 0.165]
   point[1] [0.18 0.195]
   point[2] [0.16 0.195]
   point[3] [0.16 0.165]
   z [0.02 0.065]
)

#front right wheel - midblock:
block(
   color "black"
   points 4
   point[0] [0.225 0]
   point[1] [0.225 0.03]
   point[2] [0.18 0.03]
   point[3] [0.18 0]
   z [0 0.085]
)

#front right wheel - left part:
block(
```

```
    color "black"
    points 4
    point[0] [0.245 0]
    point[1] [0.245 0.03]
    point[2] [0.225 0.03]
    point[3] [0.225 0]
    z [0.02 0.065]
)

#front right wheel - right part:
block(
    color "black"
    points 4
    point[0] [0.18 0]
    point[1] [0.18 0.03]
    point[2] [0.16 0.03]
    point[3] [0.16 0]
    z [0.02 0.065]
)

#back left wheel - midpart:
block(
    color "black"
    points 4
    point[0] [0.065 0.165]
    point[1] [0.065 0.195]
    point[2] [0.02 0.195]
    point[3] [0.02 0.165]
    z [0 0.085]
)

#back left wheel - left part:
block(
    color "black"
    points 4
    point[0] [0.085 0.165]
    point[1] [0.085 0.195]
    point[2] [0.065 0.195]
    point[3] [0.065 0.165]
    z [0.02 0.065]
)

#back left wheel - right part:
block(
    color "black"
    points 4
    point[0] [0.02 0.165]
    point[1] [0.02 0.195]
    point[2] [0 0.195]
    point[3] [0 0.165]
    z [0.02 0.065]
)

#back right wheel - midpart:
block(
    color "black"
    points 4
    point[0] [0.065 0]
    point[1] [0.065 0.03]
    point[2] [0.02 0.03]
    point[3] [0.02 0]
    z [0 0.085]
)

#back right wheel - left part:
block(
    color "black"
    points 4
    point[0] [0.085 0]
```

```
    point[1] [0.085 0.03]
    point[2] [0.065 0.03]
    point[3] [0.065 0]
    z [0.02 0.065]
  )

  #back right wheel - right part:
  block(
    color "black"
    points 4
    point[0] [0.02 0]
    point[1] [0.02 0.03]
    point[2] [0 0.03]
    point[3] [0 0]
    z [0.02 0.065]
  )

  #right back pole:
  block(
    color "gray"
    points 4
    point[0] [0.045 0.05]
    point[1] [0.045 0.055]
    point[2] [0.04 0.055]
    point[3] [0.04 0.05]
    z [0.065 0.129]
  )

  #left back pole:
  block(
    color "gray"
    points 4
    point[0] [0.045 0.135]
    point[1] [0.045 0.14]
    point[2] [0.04 0.14]
    point[3] [0.04 0.135]
    z [0.065 0.129]
  )

  #right front pole:
  block(
    color "gray"
    points 4
    point[0] [0.195 0.05]
    point[1] [0.195 0.055]
    point[2] [0.2 0.055]
    point[3] [0.2 0.05]
    z [0.065 0.129]
  )

  #left back pole:
  block(
    color "gray"
    points 4
    point[0] [0.195 0.135]
    point[1] [0.195 0.14]
    point[2] [0.2 0.14]
    point[3] [0.2 0.135]
    z [0.065 0.129]
  )

  IR_sensor(pose [ 0.09 0.0 -0.0145 0.0 ])      # Sensor frontal /base_scan_0 topic

  # Sensores não idependentes, portanto é necessário mantê-los mesmo sem usa-los
  sonar(pose [ 0.09 0.0575 -0.0245 45.0 ])      # Sensor esquerdo /base_scan_1
topic
  sonar(pose [ 0.09 -0.0575 -0.0245 -45.0 ])    # Sensor direito /base_scan_2 topic
)
```

```
########################################
#          Construir robo humano          #
########################################


define robotcraft_robot1 position(

  odom_error [0.03 0.03 0.00 0.05]
  localization "odom"
  gui_nose 1
  drive "diff"
  color "green"

  size [ 0.24 0.195 0.13 ]

  #robot's main body:
  block(
     color "orange"
     points 4
     point[0] [0.225 0.03]
     point[1] [0.225 0.165]
     point[2] [0.015 0.165]
     point[3] [0.015 0.03]
     z [0.02 0.065]
  )

  #robot's top surface:
  block(
     color "orange"
     points 4
     point[0] [0.215 0.04]
     point[1] [0.215 0.155]
     point[2] [0.025 0.155]
     point[3] [0.025 0.04]
     z [0.129 0.13]
  )

  #front left wheel - midblock:
  block(
     color "black"
     points 4
     point[0] [0.225 0.165]
     point[1] [0.225 0.195]
     point[2] [0.18 0.195]
     point[3] [0.18 0.165]
     z [0 0.085]
  )

  #front left wheel - left part:
  block(
     color "black"
     points 4
     point[0] [0.245 0.165]
     point[1] [0.245 0.195]
     point[2] [0.225 0.195]
     point[3] [0.225 0.165]
     z [0.02 0.065]
  )

  #front left wheel - right part:
  block(
     color "black"
     points 4
     point[0] [0.18 0.165]
     point[1] [0.18 0.195]
     point[2] [0.16 0.195]
     point[3] [0.16 0.165]
     z [0.02 0.065]
  )
```

```
#front right wheel - midblock:
block(
   color "black"
   points 4
   point[0] [0.225 0]
   point[1] [0.225 0.03]
   point[2] [0.18 0.03]
   point[3] [0.18 0]
   z [0 0.085]
)

#front right wheel - left part:
block(
   color "black"
   points 4
   point[0] [0.245 0]
   point[1] [0.245 0.03]
   point[2] [0.225 0.03]
   point[3] [0.225 0]
   z [0.02 0.065]
)

#front right wheel - right part:
block(
   color "black"
   points 4
   point[0] [0.18 0]
   point[1] [0.18 0.03]
   point[2] [0.16 0.03]
   point[3] [0.16 0]
   z [0.02 0.065]
)

#back left wheel - midpart:
block(
   color "black"
   points 4
   point[0] [0.065 0.165]
   point[1] [0.065 0.195]
   point[2] [0.02 0.195]
   point[3] [0.02 0.165]
   z [0 0.085]
)

#back left wheel - left part:
block(
   color "black"
   points 4
   point[0] [0.085 0.165]
   point[1] [0.085 0.195]
   point[2] [0.065 0.195]
   point[3] [0.065 0.165]
   z [0.02 0.065]
)

#back left wheel - right part:
block(
   color "black"
   points 4
   point[0] [0.02 0.165]
   point[1] [0.02 0.195]
   point[2] [0 0.195]
   point[3] [0 0.165]
   z [0.02 0.065]
)

#back right wheel - midpart:
block(
```

```
    color "black"
    points 4
    point[0] [0.065 0]
    point[1] [0.065 0.03]
    point[2] [0.02 0.03]
    point[3] [0.02 0]
    z [0 0.085]
)

#back right wheel - left part:
block(
    color "black"
    points 4
    point[0] [0.085 0]
    point[1] [0.085 0.03]
    point[2] [0.065 0.03]
    point[3] [0.065 0]
    z [0.02 0.065]
)

#back right wheel - right part:
block(
    color "black"
    points 4
    point[0] [0.02 0]
    point[1] [0.02 0.03]
    point[2] [0 0.03]
    point[3] [0 0]
    z [0.02 0.065]
)

#right back pole:
block(
    color "gray"
    points 4
    point[0] [0.045 0.05]
    point[1] [0.045 0.055]
    point[2] [0.04 0.055]
    point[3] [0.04 0.05]
    z [0.065 0.129]
)

#left back pole:
block(
    color "gray"
    points 4
    point[0] [0.045 0.135]
    point[1] [0.045 0.14]
    point[2] [0.04 0.14]
    point[3] [0.04 0.135]
    z [0.065 0.129]
)

#right front pole:
block(
    color "gray"
    points 4
    point[0] [0.195 0.05]
    point[1] [0.195 0.055]
    point[2] [0.2 0.055]
    point[3] [0.2 0.05]
    z [0.065 0.129]
)

#left back pole:
block(
    color "gray"
    points 4
    point[0] [0.195 0.135]
```

```
      point[1] [0.195 0.14]
      point[2] [0.2 0.14]
      point[3] [0.2 0.135]
      z [0.065 0.129]
    )

)
```