UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Planejamento e navegação multirrobô na plataforma ROS

Dannilo Samuel Silva Miranda

Itajubá, 15 de fevereiro de 2019

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Dannilo Samuel Silva Miranda

Planejamento e navegação multirrobô na plataforma ROS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Automação e Sistemas Elétricos Industriais

Orientador: Prof. Dr. Guilherme Sousa Bastos
Coorientador: Prof. Dr. Luiz Edival de Souza

15 de fevereiro de 2019
Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Planejamento e navegação multirrobô na plataforma ROS

Dannilo Samuel Silva Miranda

Dissertação aprovada por banca examinadora em 15 de Fevereiro de 2019, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica.**

Banca Examinadora:
Prof. Dr. Guilherme Sousa Bastos (Orientador)
Prof. Dr. Luiz Edival de Souza (Coorientador)
Prof. Dr. Alexandre Baratella Lugli
Prof. Dr. João Paulo Reus Rodrigues Leite

Itajubá
2019

Dannilo Samuel Silva Miranda

Planejamento e navegação multirrobô na plataforma ROS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Trabalho aprovado. Itajubá, 15 de Fevereiro de 2019:

Prof. Dr. Guilherme Sousa Bastos
Orientador

Prof. Dr. Luiz Edival de Souza
Coorientador

Prof. Dr. Alexandre Baratella Lugli

Prof. Dr. João Paulo Reus Rodrigues Leite

Itajubá
15 de fevereiro de 2019

Agradecimentos

Agradeço primeiramente à Deus pela vida e pelos dons que ele me concedeu.

Aos meus pais e minhas irmãs pelo apoio, confiança e conselhos, sem eles eu não teria forças para finalizar este trabalho.

Aos colegas do Laboratório de Robótica pela companhia, conselhos e ajuda no desenvolvimento dos trabalhos, em especial ao Vinícius Nogueira que me acompanhou durante toda a caminhada.

Agradeço de forma especial a minha namorada Cecília que esteve sempre me ajudando, revisando os meus textos e dando todo apoio. Sei que ela nunca duvidou de mim.

Agradeço ao meu orientador Guilherme e ao coorientador Edival pelos conselhos, suporte e ensinamentos, os quais foram essenciais para a execução e conclusão do trabalho. Aos membros da banca examinadora que dispuseram de seu tempo e conhecimento para analisar e lapidar este trabalho.

Um muito obrigado a todos, sem vocês nunca seria possível a realização deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

"Por vezes sentimos que aquilo que fazemos não é senão uma gota de água no mar. Mas o mar seria menor se lhe faltasse uma gota"

(Madre Teresa de Calcutar)

Resumo

Planejamento automático é uma ferramenta essencial para a execução de tarefas complexas. Por meio dele, é possível gerar de forma automática uma sequência de ações que leve um robô de um estado inicial a um estado desejado. O *framework* ROSPlan faz a integração de um planejador automático de tarefas com o sistema ROS (*Robot Operation System*), que é uma das plataformas mais utilizadas para a programação de robôs. Apesar do ROSPlan ter sido desenvolvido em 2015, o mesmo não detém uma performance eficiente ao executar planos com mais de um robô, ainda que seja uma área em constante crescimento nos últimos anos. Desta forma, o presente trabalho tem como foco realizar as alterações necessárias no pacote ROSPlan para que seja possível a sua utilização em sistemas multirrobôs. Desenvolveu-se um novo método para despachar as mensagens e um sistema que garanta que apenas os robôs especificados nas mensagens executem as tarefas designadas, além de um pacote que realiza o planejamento de rota para multirrobôs. Para este último, fez-se necessário desenvolver um programa para a execução das rotas geradas. Com o intuito de verificar o funcionamento das alterações realizadas no ROSPlan e no pacote desenvolvido para navegação, são feitas simulações utilizando o *software* Gazebo. Os resultados apurados se mostram satisfatórios, considerando que demonstra uma maior robustez na execução de planos com múltiplos robôs e um bom planejamento das rotas, evitando colisões com outros robôs ainda na fase de planejamento.

Palavras-chaves: Multirrobô; Planejamento Automático;Planejamento de Rota; ROS; ROSPlan.

Abstract

Automatic planning is an essential tool for performing complex tasks. It is able to automatically generate a sequence of actions that takes a robot from an initial state to a desired state. The ROSPlan framework integrates an automatic task scheduler with the ROS (Robot Operation System), which is one of the most commonly used platforms for robot programming. Although ROSPlan was developed in 2015, it does not perform efficiently when executing plans with more than one robot and this is a growing area in the last years. In this way, the present work focuses on making the necessary changes in the ROSPlan package so that it is possible to use it in multi-robot systems. A new method for dispatch messages is developed and a system that ensures that only the robots specified in the messages perform the assigned tasks, as well as a packag for multi-robot path planning. For the latter it became necessary to develop a program for the execution of the path generated. In order to verify the operation of the changes made in ROSPlan and in the package developed for navigation, simulations are made using the software Gazebo. The results obtained are satisfactory, considering that it shows a greater robustness in the execution of plans with multiple robots and a good path planning, avoiding collisions with other robots still in the planning phase.

Key-words: Automatic Planning; Multirobot; Path Planning ROS; ROSPlan.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ADL	*Action Description Language*
AIPS	*Artificial Intelligence Planning System*
AuRA	*Autonomus Robot Architecture*
BSD	*Berkeley Software Distribution*
ICAPS	*International Conference on Automated Planning and Scheduling*
IA	Inteligencia Artificial
IDL	*Interface Definition Language*
LARS	*Latin American Robotics Symposium*
MRTA	*Multi-Robot Task Allocation*
MT	*Multi-task Robot*
PDDL	*Planning Domain Definition Language*
POPF	*Pop Flags*
ROS	*Robot Operating System*
RPN	*Probabilistic Roadmap planner*
SMR	Sistema Multirrobô
ST	*Single Task Robot*
STRAIR	*STanford Artificial Intelligence Robot*
STRIPS	*Stanford Research Institute Problem Solver*
TA	*Time-extended assignment*
UNIFEI	Universidade Federal de Itajubá
VANT	Veículo Aério Não Tripulado
WP	*Waypoint*

Sumário

1 Introdução

No capítulo 1 é contextualizada a proposta de estudo desenvolvida no decorrer deste trabalho, apresentando a importância do planejamento de alto nível para sistemas complexos, a crescente tendência dos pesquisadores por sistemas multirrobôs e a necessidade de um sistema de navegação adequado para cada aplicação. São também apresentados os objetivos do trabalho e as principais contribuições geradas, e, por fim, é explicitada a estruturação completa do trabalho.

1.1 Motivação

Há mais de 50 anos, os pesquisadores trabalham no desenvolvimento de robôs móveis autônomos para que sejam capazes de executar diversas tarefas como a limpeza de um ambiente, fazer a busca e a entrega de bens, examinar áreas hostis ou remotas, alertar e resgatar vítimas em cenário de desastre, entre outros. De uma forma geral, espera-se que os robôs consigam de forma autônoma realizar esforços extenuantes, repetitivos ou até tarefas perigosas que, anteriormente, eram realizadas por trabalhadores humanos (NOCKS, 2007).

Para que haja a realização da maioria das tarefas é necessário que o robô seja capaz de se deslocar livremente em um ambiente, e, para isto, é imprescindível que exista um sistema de navegação instalado, que seja responsável por gerar as trajetórias do robô e garantir a sua perfeita navegação.

A principal função do sistema de navegação é gerar sinais que chegam aos atuadores de direção, velocidade e aceleração, fazendo com que o robô assuma comportamentos diversos de acordo com a necessidade que tem para cumprir um objetivo especificado. O projeto do sistema de navegação depende do tipo de tarefa a ser executada, das características do robô e do ambiente, sendo que aspectos como obstáculos não esperados, imprecisão de sensores e perda de tração devem ser levados em consideração para o desenvolvimento de um sistema, o que torna o projeto muito complexo.

Desta forma, os pesquisadores, com o objetivo de gerar trabalhos mais significativos nas suas áreas de domínio, subdividem a navegação em quatro principais pilares: percepção - a forma que o robô interpreta os dados dos sensores, cognição - as tomadas de decisão para alcançar um determinado objetivo, controle - refere-se as técnicas de controle dos motores para executar uma trajetória desejada, e por fim, a localização - reconhecimento da posição do robô dentro de um mapa (SIEGWART et al., 2011).

Outro ponto significativo é referente a quantidade de robôs que será utilizado,

visto que a maioria das tarefas que são executas por um único robô, também podem ser executadas por um time de robôs móveis que trabalham em paralelo. Considerando que uma tarefa pode ser decomposta em várias sub-tarefas, cada robô pode executar uma pequena parte para chegar ao objetivo final. Desta maneira, o plano pode ser completado com maior rapidez e robustez. Além disso, um robô complexo que consegue executar diversas tarefas pode ser substituído por diversos robôs mais simples e mais baratos, facilitando a programação e diminuindo o custo total do trabalho.

As linguagens e algoritmos de planejamento provenientes do campo da Inteligência Artificial (IA), especificamente da área denominada de Planejamento Automático, é uma boa opção para determinar quais robôs serão utilizados e qual a sequência de tarefas que executarão para o cumprimento da missão. Basicamente, o objetivo do planejamento é produzir uma sequência de ações que deverão ser executadas por um ou mais agentes, com a finalidade de direcionar o sistema do estado inicial para um estado em que todos os objetivos foram alcançados.

O ROSPlan (CASHMORE et al., 2015) é um *framework* que realiza a integração do Planejamento Automático com o ROS (*Robotic Operation System*) que, por sua vez, é um *framework* largamente utilizado pelos pesquisadores da área de robótica para o desenvolvimento de software.

O ROSPlan é capaz de gerar o estado inicial em PDDL (*Planning Domain Definition Language*), fazer a chamada do planejador de forma automática, validar o plano gerado, converter as mensagens em PDDL para mensagens que podem ser executadas pelos controladores de baixo nível, acompanhar a execução do plano e replanejar em caso de falha. Entretanto, este *framework* foi desenvolvido para aplicações com apenas um robô, não apresentando um método eficiente de despacho das ações para multirrobôs, nem uma verificação por parte dos robôs das mensagens recebidas, identificando se aquela mensagem realmente deve ser executada por eles.

Além disso, apesar de já existirem sistemas de navegação no ROS, e estes poderem ser utilizados em mais de um robô, o desvio de objetos dinâmicos será feito apenas de forma reativa. Portanto, assim que fosse encontrado um obstáculo, o robô iria gerar um novo plano evitando aquele objeto que não estava em sua base de conhecimento previamente.

Diante do exposto acima, a principal motivação da dissertação está em conseguir executar um sistema multirrobô com o ROSPlan de forma mais eficiente que as soluções disponíveis na versão original, e que o sistema de navegação consiga prever colisões com outros robôs e gerar um novo caminho ainda na fase de planejamento.

1.2 Objetivos

O principal objetivo é modificar o ROSPlan, com o intuito de possibilitar a aplicação em multirrobôs de forma mais eficiente. Assim, é indispensável desenvolver um novo método que irá despachar as mensagens com as ações que os robôs irão executar, sendo que as mensagens devem ser enviadas o mais breve possível, mas não inviabilizando ou atrapalhando a execução do plano principal.

Ademais, possui, também, como objetivo trabalhar na cognição de um sistema de navegação: gerar o caminho que cada robô deve seguir em um sistema multirrobô e considerar ainda medidas preventivas para evitar colisões com os demais que compartilhem o mesmo ambiente.

Por fim, desenvolveu-se um pacote de controle, que mesmo simples, é utilizado para executar a navegação e verificar o funcionamento do plano gerado pelo sistema de cognição. A localização é feita apenas utilizando o sensor odométrico, e não é utilizado nenhum outro sensor para auxiliar na navegação.

1.3 Contribuições

A dissertação tem como principal contribuição a adaptação do ROSPlan para que seja possível o planejamento e a execução de tarefas que exija a cooperação e a coordenação de múltiplos robôs, bem como a criação de uma interface para facilitar a simulação de ambientes. Esta interface irá criar os arquivos necessários para a execução de cada robô com uma identificação única, que terá suma importância na execução dos planos.

Outra contribuição é a criação de um pacote de navegação no ROS em que os caminhos são planejados levando em consideração os outros robôs que estão no ambiente e os caminhos que irão percorrer.

1.4 Estrutura do Trabalho

A dissertação está estruturada em sete capítulos na seguinte disposição. No capítulo 1 tem-se uma introdução sobre o trabalho. No capítulo 2 faz-se uma introdução teórica sobre Agentes Inteligentes, no qual são apresentadas as vantagens de sistemas que utilizam multirrobôs e a principal taxonomia para alocação de tarefas em sistemas multirrobôs. Posteriormente, no capítulo 3, é realizada uma revisão bibliográfica sobre planejamento automático. Já o capítulo 4 mostra a importância de um sistema de navegação e como é feito o planejamento de rota para robôs. O capítulo 5 é destinado a apresentação do ROS e suas principais ferramentas, assim como do pacote ROSPlan e alguns trabalhos que também o teve como base para o seu desenvolvimento. No capí-

tulo 6 é mostrado como foi realizado o desenvolvimento do trabalho e os resultados dos experimentos. A conclusão e trabalhos futuros são discutidos no último capítulo.

2 Agentes e Alocação de Tarefas

Neste capítulo é feita uma introdução sobre agentes e a diferença entre um agente racional e um agente não racional. Assim como, há uma abordagem sobre agentes robóticos, que é um dos principais temas do trabalho. Por fim, uma taxonomia para alocação de tarefas em sistemas multirrobôs é apresentada.

2.1 Agente Racional

Um agente pode ser definido como tudo aquilo que é capaz de perceber o ambiente por meio de sensores e de agir sobre ele de forma autônoma por meio de atuadores (NORVIG; RUSSELL, 2014). A título de exemplo, um agente robótico pode utilizar sensores *laser* e câmeras para perceber o ambiente, bem como um conjunto de locomoção e braço robótico para atuar no ambiente.

O estudo de agentes está presente em muitos subcampos da ciência da computação e inteligência artificial, sendo aplicado na área da manufatura, construção, exploração do espaço, assistência em desastre e auxílio em cirurgia médica (LATOMBE, 2012).

Caso um agente faça tudo certo, ele pode ser considerado racional, ou seja, um agente racional é aquele que seleciona a ação, ou o conjunto de ações, que venha maximizar a sua medida de desempenho a partir do seu conhecimento interno e pelo conjunto de percepções do meio em que se encontra. Sendo assim, a medida de desempenho pode ser diferente a depender de como é avaliada, pois em uma mesma aplicação um agente pode ser definido como racional ou não.

Norvig e Russell (2014) apresenta um exemplo em que um agente aspirador de pó limpa um quadrado se ele estiver sujo e passa para o outro quadrado se o primeiro não estiver sujo. Considerando, um ambiente em que exista apenas dois quadrados para serem limpos, em que a posição do robô e das sujeiras não sejam previamente conhecidas, e que a medida de desempenho oferece o prêmio de um ponto para cada quadrado limpo, pode-se afirmar que nesta situação e com esta medida de desempenho, este agente terá o desempenho tão alto quanto qualquer um outro, portanto pode ser considerado racional. No entanto, este mesmo agente pode ser considerado irracional caso o agente fique oscilando de um lado para o outro e a medida de desempenho aplique uma penalidade para cada movimento do robô.

Dentre os desafios do campo da IA (Inteligência Artificial), um deles é projetar o programa do agente para que aja de forma racional. Assim, o trabalho trata tanto do planejamento das ações quanto de sua execução. Deste modo, os agentes robóticos deverão

receber por meio da entrada dos sensores ou de conhecimento informado a percepção atual do sistema e devolverão uma ação ou um conjunto de ações para os atuadores a fim de completar o objetivo de forma eficiente.

2.2 Agentes robóticos

O termo "robô" foi criado em 1921 por Karel Ĉapek para se referir a criação de vários trabalhadores humanoides. Ele se referia a corpos orgânicos, e não à máquinas, entretanto, o mesmo termo foi utilizado algumas décadas depois para se referir as máquinas inseridas nas indústrias (NOCKS, 2007). Atualmente, os robôs são considerados agentes físicos que executam tarefas manipulando o ambiente a partir dos seus sensores e atuadores.

Os sensores permitem que o robô tenha uma percepção do ambiente, visto que podem ser câmeras, sensores ultrassônicos, giroscópios, acelerômetros, entre outros. Os atuadores, por sua vez, são elementos que produz movimento, atendendo a comandos que podem ser manuais, eléctricos ou mecânicos. Alguns exemplos são as pernas, rodas, juntas e garras.

Ao entender o seu significado, percebe-se que, por meio dos atuadores e sensores, é possível ter um sistema de navegação autônomo responsável por gerar as trajetórias que serão seguidas pelo robô. A partir dos sinais providos pelos sensores, informando o estado do ambiente ao longo do percurso, é possível enviar sinais aos atuadores de direção, velocidade e aceleração, fazendo com que o robô assuma comportamentos diversos para alcançar os seus objetivos.

Inúmeras aplicações atribuídas a um único robô móvel podem ser executadas com vantagens por um sistema de múltiplos robôs. Por essa razão, no fim da década de 80 e por toda a década de 90, houve uma significativa mudança no foco de pesquisa na robótica móvel, no lugar de utilizar apenas um robô, os pesquisadores passaram a investigar problemas envolvendo múltiplos robôs (GERKEY; MATARIĆ, 2004; PARKER et al., 2000).

Os sistemas multirrobôs (SMR) diferem dos sistemas de um único robô, na medida em que existem vários robôs que modelam os objetivos e ações a serem realizadas (STONE; VELOSO, 2000). Portanto, por este sistema apresentar diversas vantagens sobre aqueles, a maioria dos sistemas reais inspirados em aplicações domésticas, de transporte, logística e manufatura envolvem sistemas multi agente (CROSBY; JONSSON; ROVATSOS, 2014).

Guzzoni et al. (1997) apresenta como uma das vantagens de um SMR a possibilidade de realizar as tarefas com mais agilidade, entretanto, ele chama a atenção para que o número de robôs inseridos no ambiente não seja excessivo, visto que poderá ocasionar

a competição pelos recursos, por exemplo, as rotas.

Burgard et al. (2005) utiliza uma abordagem com multirrobô para diminuir o tempo de exploração de ambientes. O autor sugere uma coordenação dos robôs com a finalidade de lhes oferecerem uma escolha apropriada dos pontos alvos, assim o conjunto explora simultaneamente diferentes regiões do ambiente, completando a missão de exploração com muito mais agilidade.

No mesmo contexto, Wurm (2012) dispõe que um time de robôs também pode oferecer maior robustez a falha por meio da redundância, uma vez que quando um robô designado a uma tarefa falha, pode ser substituído por outro que também seja capaz de executar aquela tarefa de modo a evitar o exato ponto em que houve o erro.

Chouhan, Singh e Niyogi (2015) apresentam alguns exemplos de tarefas que não podem ser executadas por apenas um robô. Eles acreditam que seja necessário uma coordenação entre os robôs para que possam em uma ação conjunta realizar tarefas mais complexas, que apenas um robô não tenha a capacidade de executar sozinho. Assim, uma arquitetura para multirrobô deve oferecer um mecanismo para assegurar que as atividades dos vários agentes do sistema sejam coordenadas, além de fornecer uma estrutura de comunicação que permita a troca de informações.

Segundo Rich e Knight (1993), o maior problema a ser enfrentado no projeto de qualquer sistema de raciocínio distribuído é como as ações de cada agente podem ser coordenadas para que funcionem efetivamente em conjunto. As seguintes abordagens são apresentadas por ele:

- Um agente denominado de mestre é responsável por fazer um plano e distribuir as partes para os outros agentes "escravos"que cumprem suas tarefas e comunicam os resultados.
- Um agente é responsável por decompor o problema em subproblemas, mas depois existe uma negociação entre os agentes para decidir qual tarefa cada um deles irá executar.
- Não há nenhum agente responsável, todos os agentes devem cooperar tanto na formação do plano quanto na sua execução afim de atingir um objetivo comum para todos os agentes.
- Não há nenhum agente responsável e não há garantia de que haverá um objetivo comum para todos os agentes, sendo que eles podem competir entre si para alcançarem os seus objetivos pessoais.

A primeira alternativa apresentada acima será a utilizada na dissertação, que consiste na utilização de um único agente para fazer a divisão do objetivo em subobjetivos e

distribui-los aos outros agentes de forma coordenada. A etapa de decomposição do problema é essencialmente a mesma dos sistemas de planejamento com um único agente, esta será explanada no Capítulo 3.

Assim que o plano for gerado, as tarefas devem ser alocadas aos agentes disponíveis de forma a completar o objetivo global. Portanto, quando o agente "mestre" designa uma tarefa para algum "escravo", este deve comunicar a sua conclusão ao mestre, que por sua vez utilizará a informação para coordenar a execução e envio das outras sub-tarefas.

2.3 Alocação de tarefas

No intuito dos múltiplos robôs conseguirem atuar em um mesmo ambiente, é imprescindível fazer uma coordenação entre eles de forma que executem as tarefas sem interferir ou obstruir os caminhos dos outros robôs do ambiente. Além disso, é desejado que, em um sistema com múltiplos robôs, exista a cooperação entre eles para executar tarefas que não poderiam ser feitas por apenas um robô.

O problema da alocação de tarefas refere-se a determinar qual robô deve executar qual tarefa para alcançar um objetivo global. Os problemas dessa natureza buscam como solução atribuir otimamente um conjunto de tarefas para um conjunto de robôs, de maneira que o desempenho geral seja otimizado.

Diante disso, Gerkey e Matarić (2004) propõem uma taxonomia que classifica um problema de alocação de tarefas em três eixos e, desta forma, facilita as pesquisas e os trabalhos na área de sistemas de múltiplos robôs.

O primeiro eixo refere-se à quantidade de tarefas que um robô pode executar por vez, sendo classificado como ST (*single task robot*), em que o robô pode executar apenas uma tarefa por vez, ou MT (*multi-task robot*), que significa que o robô pode executar múltiplas tarefas simultaneamente.

Já o segundo eixo refere-se a quantidade de robôs necessários para executar uma única tarefa. Aqui também existem duas possibilidades, SR (*single robot task*), para cada tarefa requer exatamente um robô, enquanto que o MR (*multi-robot tasks*) significa que algumas tarefas podem requerer multiplos robôs.

Por fim, o terceiro eixo refere-se ao tipo de alocação do problema, sendo a primeira denominada de IA (*instantaneous assignment*), que significa que as informações disponíveis sobre os robôs, as tarefas e o ambiente permitem apenas uma alocação instantânea de tarefas para os robôs, sem planejamento para futuras alocações. Ou seja, a alocação das tarefas é feita de forma instantânea, sem levar em consideração o estado futuro do sistema, caso contrário, o sistema é considerado TA (*time-extended assignment*) em que mais informações estão disponíveis, como o conjunto de todas as tarefas que precisam ser

atribuídas, ou um modelo de como as tarefas devem chegar ao longo do tempo. Neste último caso, diversas tarefas são alocadas para um robô, o qual deve executar cada alocação conforme seu agendamento.

A representação visual da taxonomia é ilustrada na figura 1 em que cada eixo representa uma variável. Um problema de alocação de tarefa é definida por três argumentos, sendo o primeiro referente a quantidade de tarefas que um robô pode executar por vez, retirado do primeiro eixo. O segundo é referente a quantidade de robôs necessários para executar uma única tarefa, retirado do segundo eixo, e o terceiro argumento é referente ao tipo de alocação de tarefa, retirado do terceiro eixo.

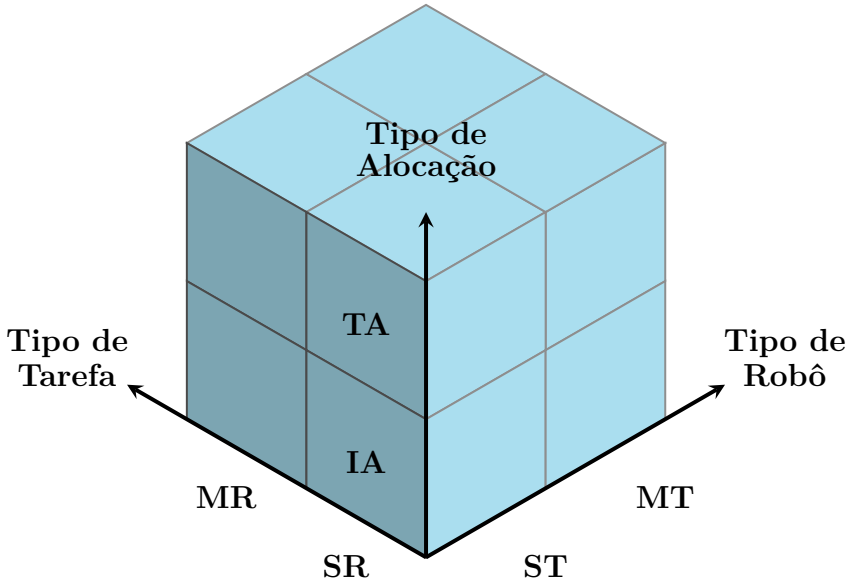

Figura 1 – Representação visual da taxonomia de três eixos sugerida por Gerkey e Matarić (2004).

A combinação dos eixos gera oito classes de problemas MRTA bem definidos. O presente trabalho aborda sistemas MT-MR-TA, portanto a alocação de tarefas permite que um robô realize mais de uma tarefa ao mesmo tempo; que uma tarefa exija mais de um robô para ser completada; e que diversas tarefas possam ser alocadas a um robô e ele faça a sua execução de acordo com um planejamento.

Para fazer a alocação das tarefas foi utilizado um algoritmo de planejamento automático, que será explicado com mais detalhes no próximo capítulo. O algoritmo permite definir quais tarefas precisarão ser executadas para completar o objetivo final, quais robôs serão utilizados e a quantidade de robôs necessários para a realização da tarefa.

Apesar do algoritmo de planejamento fornecer uma agenda com os tempos que as ações devem ser executadas, em aplicações reais, dificilmente o robô executa a tarefa rigorosamente no tempo planejado, o que leva os robôs a ficarem algum tempo ociosos ou necessitarem de um replanejamento por não terminar a tarefa no tempo previsto. No

intuito de resolver o problema, foi desenvolvido um algoritmo para coordenar a execução das tarefas planejadas. O algoritmo de coordenação é explicado na seção 6.1.

3 Planejamento Automático

No campo da Inteligência Artificial existe uma área chamada de Planejamento Automático. Esta área lida com a geração automática da sequência de ações necessárias para resolver um problema. A partir de um estado inicial, de um conjunto de ações e de um estado objetivo; o planejador deve determinar uma sequência de ações a serem tomadas para alcançar o estado objetivo a partir da antecipação dos resultados da execução de cada ação.

Ghallab, Nau e Traverso (2004) afirmam que o planejamento se concentra em escolher e organizar ações para alterar o estado do sistema. Assim, o autor menciona que, para gerar um modelo conceitual de planejamento, é necessário um modelo geral para um sistema dinâmico. Diante disso, ele apresenta o modelo de um sistema de transição de estados como uma 4-tupla $\sum(S, A, E, \gamma)$ em que:

- S=s1,s2,... é um conjunto de estados finitos ou recursivamente enumerável.
- A=a1,a2,... é um conjunto de ações finitas ou recursivamente enumerável.
- E=e1,e2,... é um conjunto de eventos finitos ou recursivamente enumerável e
- γ:S x A x E $\rightarrow 2^S$ é a função de transição dos estados.

Este modelo também pode ser representado por um grafo direcionado assim como ilustrado na figura 2. Na figura é possível observar um contêiner, um guincho que pode pegar e soltar o contêiner, bem como um robô que pode carregar o contêiner, e mover entre dois locais. Neste modelo, o conjunto de estados "S" pode ser definido como $S = s_0, s_1, s_2, s_3, s_4, s_5$, o conjunto de ações é dada por $A = pegar$, $largar$, $carregar$, $descarregar$, $mover1$, $mover2$; e quanto aos eventos, não existem no sistema. O estado de transição pode ser representado por um arco de "s" para "s"' em que "s"' $\epsilon\gamma(s, u)$, "u" é um par (a,e), tal que $a \epsilon$ A e $e \epsilon$ E. Sendo assim, o arco (s_0, s_1) é dado pela ação pegar, o arco (s_4, s_5) pela ação mover2 e assim por diante.

De acordo com Rich e Knight (1993) a palavra planejamento refere-se ao processo de pré-calcular várias etapas de um procedimento de solução de problema antes de executar qualquer uma delas. Para que isto seja possível, é necessário o conhecimento prévio dos requisitos para a realização de cada ação e dos efeitos causados por ela. Os requerimentos ou precondições indicam o que é necessário ser verdadeiro antes que a ação seja aplicada, enquanto que o efeito indica o que será alterado no ambiente quando a ação for aplicada. As possíveis representações para as ações carregar e descarregar são:

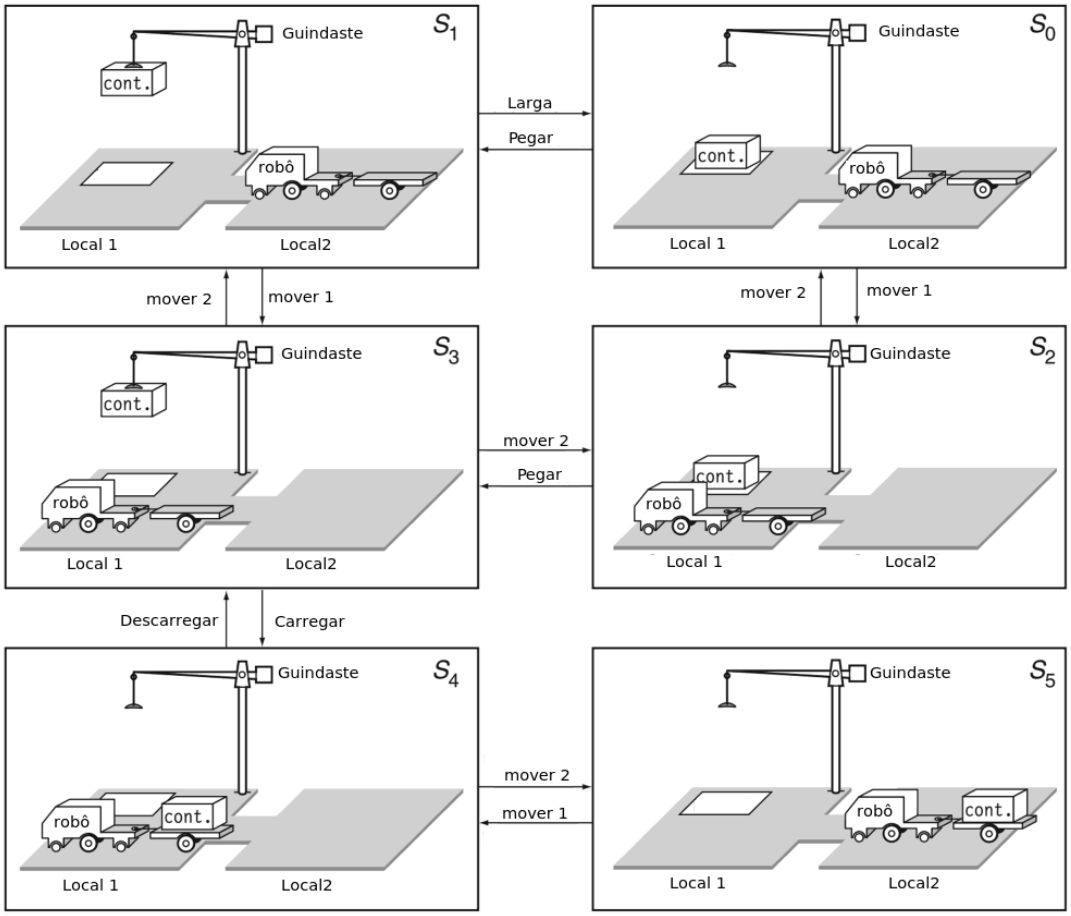

Figura 2 – Gráfico direcionado de um problema de Logística (GHALLAB; NAU; TRAVERSO, 2004)

Ação Carregar

precondições: Robô no local 1, guincho segurando contêiner e robô livre.

efeitos: Robô carregando contêiner e guincho livre.

Ação Descarregar

precondições: Guincho livre, robô no local 1 e robô carregando contêiner.

efeitos: Robô livre e guincho segurando contêiner.

O conjunto de ações devidamente sequenciadas para resolver o problema de planejamento é denominado **plano**, já o algoritmo que realiza esse processo é o **planejador**. A partir da descrição formal do problema, o planejador deve derivar as heurísticas e realizar as inferências e buscas necessárias para solucionar o problema.

Num problema de planejamento pode se destacar 3 elementos principais: o estado inicial, o estado final e o conjunto de ações que podem ser tomadas. O "estado inicial" descreve o ambiente antes de qualquer ação ou evento acontecer. "O estado final", ou objetivo, é aquele em que se espera alcançar após a execução do plano. O "conjunto de ações"

são todas as ações que são possíveis de serem executadas neste ambiente. Considerando como exemplo o sistema de transição de estados representado na figura 2, sendo o estado inicial s_0, o estado final s_5, o conjunto de ações do problema (pegar, largar, carregar, descarregar, mover1, mover2), ter-se-á um possível plano que pode ser retornado como mover1, carregar, move2.

3.1 Algoritmos de Busca

Existem diversas maneiras de encontrar a solução de um problema de planejamento. As mais simples são a busca no espaço de estados para frente e para trás. Neste algoritmo, o espaço de busca é um subconjunto do espaço de estados. Cada nó corresponde a um estado no ambiente, cada arco corresponde a um estado de transição e o plano corresponde ao caminho encontrado no espaço de busca.

3.1.1 Busca para frente

A busca para frente é um algoritmo não determinístico que começa a executar do estado inicial e aplica todas as ações possíveis do problema, ou seja, todas as ações em que as precondições são satisfeitas. Do conjunto de estados resultante, verifica se algum deles é o estado objetivo, caso não o seja, aplica-se novamente o conjunto de todas as ações aplicáveis a cada um dos estados gerados até que seja encontrado o estado objetivo.

Esta busca, frequentemente, é ineficiente para ser aplicada na prática, pois é propensa a explorar ações irrelevantes. Considere o exemplo em que exista 10 aeroportos, e que cada aeroporto possua 5 aviões e 20 peças de carga. O objetivo é mover toda a carga do aeroporto A para o B. A descoberta da solução pode ser difícil porque o fator médio de ramificação é enorme: cada um dos 50 aviões pode voar para nove outros aeroportos, e cada um dos 200 pacotes pode ser descarregado ou carregado em qualquer avião no aeroporto. Desta forma, existe no mínimo 450 ações possíveis de serem aplicadas quando todos os pacotes estão nos aeroportos sem aviões; e no máximo 10.450 ações possíveis de serem aplicadas quando todos os pacotes e aviões estão no mesmo aeroporto. Assumindo assim, que a média seja de 2000 ações por estado, o grafo de busca até a profundidade da solução óbvia tem cerca de 2000^{41} nós (NORVIG; RUSSELL, 2014).

3.1.2 Busca para trás

A ideia na busca para trás é começar do objetivo e retroceder com as ações obtendo sub-objetivos até que seja alcançado o estado inicial. Ela também é conhecida como busca de estados relevantes porque considera apenas as ações que são relevantes ao objetivo (NORVIG; RUSSELL, 2014).

A busca inicia-se a partir de um conjunto de literais que forma o estado objetivo. Assim, são buscadas ações que terão como efeito os literais desejados e qualquer outro adicional, sendo que o algoritmo procede até que seja encontrado um estado que satisfaça as condições do estado inicial.

Na figura 3 está representado o princípio da busca para frente e para trás.

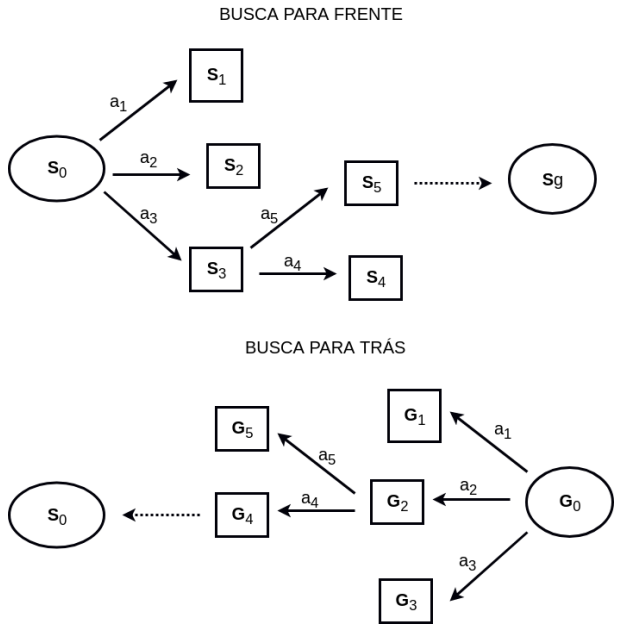

Figura 3 – Representação da busca para frente e para trás

3.2 Grafo de planejamento

O grafo de planejamento é utilizado para elaborar planos para problemas que possam ser representados usando lógica proposicional (COPPIN, 2013), sendo composto por diversos níveis, como demonstrado na figura 4. O primeiro nível corresponde ao estado inicial (S_0), nele estão presentes todas as proposições verdadeiras daquele estado. O próximo nível está representado por A_0, ele corresponde a todas as ações instanciadas que podem ser aplicadas em S_0. O nível seguinte contém todos os possíveis literais que podem ser verdadeiros a partir do estado S_0 e da aplicação das ações A_0. Os próximos níveis seguem a mesma lógica, revesando entre níveis S_i e A_i até que o critério de parada seja atendido.

Para construir um grafo de planejamento deve-se levar em consideração as ações persistentes e as exclusões mútuas. As ações persistentes são aquelas proposições que não apareceram por meio da aplicação de uma ação, sendo que elas persistiram no nível anterior. As exclusões mútuas registram conflitos entre ações ou efeitos que são mutuamente exclusivas, como na figura 4, em que as ações $MoverSobre(A, B)$ é mutuamente exclusiva com $MoverSobre(B, A)$ ou em relação as proposições $livre(b)$ e $sobre(A, B)$.

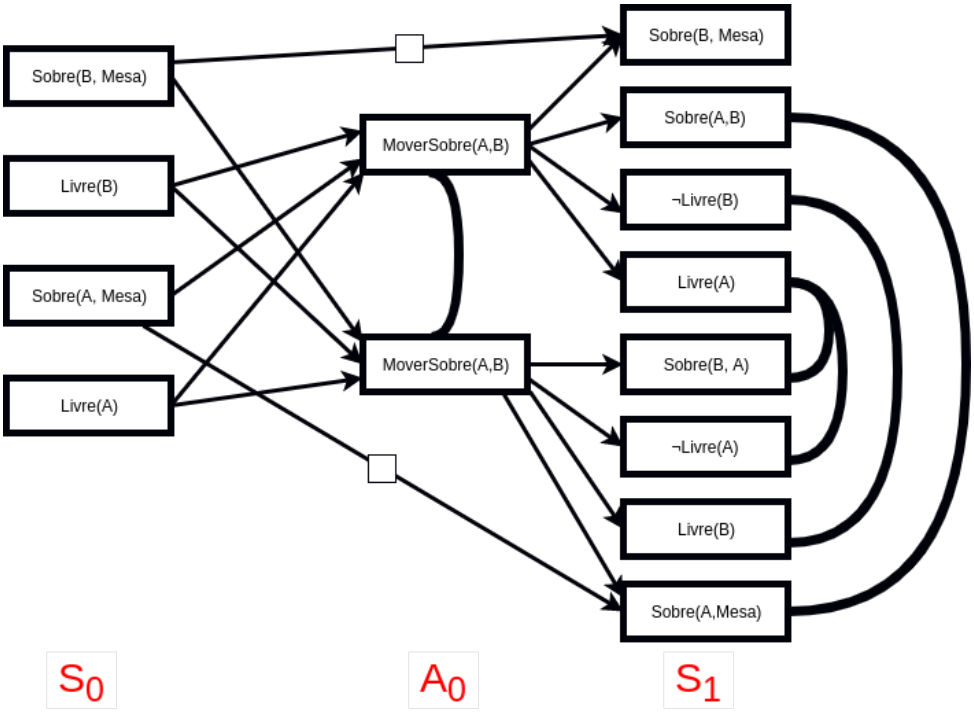

Figura 4 – Grafo de Planejamento

Um algoritmo que pode ser utilizado para conseguir um plano a partir do grafo de planejamento é o *GraphPlan* (BLUM; FURST, 1997). Ele adiciona repetidamente níveis ao grafo de planejamento até que todas as proposições do estado objetivo apareça, de forma que não sejam mutuamente excludentes. Uma vez que todos os objetivos tenham aparecido, empenha-se em encontrar um plano que resolva o problema e, em caso de falha, tenta-se expandir mais um nível até que não exista mais razão para continuar.

Caso exista um plano, o algoritmo do GraphPlan garante encontrá-lo e que ele seja o plano mais curto possível (COPPIN, 2013).

3.3 Linguagens de Planejamento

A linguagem de planejamento é um método padronizado para comunicar instruções para o algoritmo de planejamento. É um conjunto de regras sintáticas e semânticas usadas para definir os objetos, o estado inicial do ambiente, os objetivos e as regras. Atualmente, os planejadores mais utilizados são baseados na linguagem PDDL, que se deriva da STRIPS. Nos próximos tópicos será explanado um pouco sobre as principais linguagens de planejamento.

3.3.0.1 STRIPS

STRIPS (FIKES; NILSSON, 1971) (*Stanford Research Institute Problem Solver*), que significa Solucionador de Problemas do Instituto de Pesquisa *Stanford*, foi um dos

primeiros sistemas de planejamento utilizado no planejamento automático, embora tenha sido substituído por uma série de métodos mais sofisticados. A linguagem empregada para representar problemas de planejamento foi utilizada como base para desenvolver outros planejadores, como o ADL (*Action Description Language*) e o PDDL (*Planning Domain Description Language*).

O STRIPS foi projetado para ser o componente de planejamento de *software* no projeto do robô Shakey, na SRI(*scientific research institute*). Ele apresenta um sistema de busca no espaço de estados que utiliza a análise de meios-fins, que consiste em considerar as diferenças entre o estado objetivo e o estado atual, e selecionar ações que visem diminuir esta diferença. O sistema de planejamento foi desenvolvido para que agentes robóticos fossem capazes de navegar por um mundo de blocos[1], como também ser utilizados em outros problemas de planejamento.

Para ilustrar o uso do STRIPS considere o domínio do mundo dos blocos representado na figura 5. O problema consiste em uma mesa, três blocos (a,b,c) e um braço de robô que pode deslocar os blocos. O estado inicial é exibido na figura 6 em que o bloco "a" e o bloco "c" estão sobre a mesa, e o bloco "b" está sobre o bloco "c". O objetivo será colocar o bloco "c" sobre o bloco "a".

Figura 5 – Fragmento de um domínio STRIPS
(COPPIN, 2013)

Os estados do mundo são representados por predicados de primeira ordem[2]. No exemplo é utilizado dois predicados: $sobre(x, y)$, que representa que o bloco x está sobre o bloco y; e $livre(x)$, que representa que não há um bloco sobre o bloco x. A letra M é utilizada para representar a mesa, que será grande o suficiente para comportar todos os blocos de uma vez.

[1] Problema que consiste em rearranjar a disposição de blocos empilhados de acordo com uma disposição estabelecida

[2] Um estudo aprofundado é abordado no capitulo 9 de (NORVIG; RUSSELL, 2014)

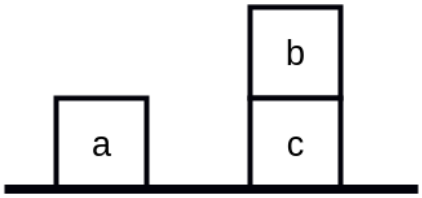

Figura 6 – Estado inicial do mundo dos blocos
(COPPIN, 2013)

O estado inicial, assim como representado na figura 6, é definido como *"Sobre(a,M) Sobre(b,c) Sobre(c,M) Livre(b) Livre(a) Livre(M)"*. Todo predicado não mencionado é considerado falso, pois o STRIPS utiliza a hipótese do mundo fechado.

As ações são representadas por um nome seguido de uma lista de parâmetros, das precondições e dos efeitos, sendo que qualquer variável que esteja representada nas precondições ou nos efeitos deve também estar na lista de parâmetros. A ação $MoverSobre(x, y)$ significa que o objeto "x" é movido de onde ele estiver e é colocado sobre o objeto "y".

Na representação do estado inicial, a precondição aceita apenas predicados positivos diferente dos efeitos que também podem receber predicados negativos. Nos efeitos, os predicados negativos representam as precondições que serão deletadas e os predicados positivos representam os que serão adicionados.

3.3.1 ADL E PDDL

ADL (PEDNAULT, 1987) é uma linguagem mais expressiva que a STRIPS e consegue representar alguns problemas que não são possíveis de ser representados em STRIPS. Algumas das diferenças entre essas duas linguagens são(COPPIN, 2013):

- Inclusão de efeitos condicionais.
- Utilização de disjunção nas precondições.
- Utilização de predicados negativos nas precondições.
- Suporte à variáveis tipadas, isto faz com que menos regras precisam ser expressas.
- Uso de quantificadores existenciais e universais.
- Faz uso de hipótese de mundo aberto, ou seja, literais não mencionados em um estado são desconhecidos.

A linguagem PDDL (MCDERMOTT et al., 1998) foi desenvolvida para a competição de planejamento AIPS (*Artificial Intelligence Planning System*) (MCDERMOTT, 2000) em 1998, que teve como objetivo criar padrões de domínios de planejamento, para

que os pesquisadores pudessem tornar as comparações entre os planejadores mais significativas e, consequentemente, conseguissem medir a evolução geral no campo da inteligência artificial.

Uma das regras decididas para a competição, foi que seria utilizada a linguagem de planejamento PDDL. Desde então, esta linguagem se tornou o padrão para competições de planejamento e é utilizada na principal conferência de planejamento, a ICAPS (*International Conference on Automated Planning and Scheduling*). O PDDL pode ser utilizado para representar STRIPS e ADL, e foi introduzido para fornecer uma notação comum que possa ser usado por qualquer sistema de planejamento.

Existem 5 versões oficiais da linguagem PDDL (PDDL1.2, PDDL2.1 PDDL2.2 PDDL3.0 e PDDL3.1) em que novos recursos foram sendo adicionados, por exemplo, o tempo, as variáveis de custo de uma ação, variáveis com novos estados, entre outros.

A versão 2.1 foi a escolhida para ser utilizada nesta dissertação, pois apesar de não ser a última, já está bem consolidada e apresenta bons planejadores. Na versão 2.1, já é possível utilizar ações com duração, operações numéricas, efeitos contínuos, efeitos condicionais e métrica na resolução do problema.

A representação da linguagem PDDL é feita em 2 arquivos, o arquivo domínio e o arquivo problema. Para entender a estrutura da linguagem, considere o algoritmo abaixo que representa um domínio simples na versão 2.1.

Algoritmo 3.1 – Domínio do mundo dos blocos em PDDL 2.1

```
1  (define (domain travel)
2      (:requirements :strips :typing :fluents :disjunctive-
          preconditions :durative-actions)
3      (:types
4          waypoint
5          robot)
6      (:predicates
7          (robot_at ?v - robot ?wp - waypoint)
8          (visited ?wp - waypoint)
9          (connected ?wp1 ?wp2 - waypoint)    )
10     (:functions (fuel-level ?v - robot)
11         (fuel-required ?wp1 ?wp2 - waypoint))
12     (:durative-action goto_waypoint
13         :parameters (?v - robot ?from ?to - waypoint)
14         :duration ( = ?duration 60)
15         :condition (and
16             (at start (>= (fuel-level ?v) (fuel-required ?from ?
                to)))
17                 (at start (robot_at ?v ?from))
```

```
18                  (over all (connected ?from ?to)))
19          :effect (and
20                  (at end (visited ?to))
21                  (at end (robot_at ?v ?to))
22                  (at end (decrease (fuel-level ?v) (fuel-required
                       ?from ?to)))
23                  (at start (not (robot_at ?v ?from))))
24      )
25  )
```

Na primeira linha é definido o nome do domínio ”viagem”, em (2) os requisitos da linguagem utilizada, de (3) a (5) são definidos os tipos dos objetos que podem ser criados e de (6) a (9) são definidos os predicados que serão utilizados nas ações. Já as funções estão representadas nas linhas (10) e (11) e permitem medir o consumo de recurso, em que a função *fuel-level* representa o nível de combustível no robô, enquanto que *fuel-required* aponta o quanto de combustível precisa para ir de um ponto ao outro.

A descrição das ações é dada a partir de (12) definindo o nome da ação. Na linha (13) são apresentados os parâmetros que são utilizados e o seu tipo, e na linha (14) está a duração da ação. Da linha (15) à linha (18) estão as precondições que devem ser atendidas antes de começar a executar a ação (*at start*) e as que devem ser atendidas durante toda a execução da ação (*over all*). Da linha (19) à linha (23) estão os efeitos que acontecem quando começar a executar a ação (*at start*) e as que acontecem quando termina de executar a ação(*at end*).

Nas linhas (16) e (22) são utilizadas funções para trabalhar com os recursos. Na linha (16) verifica se o nível do combustível que há no robô é suficiente para percorrer o percurso desejado, enquanto que na linha (22) é aplicado o efeito de subtrair o valor de combustível utilizado para o robô realizar a ação.

Obtendo o domínio modelado, deve-se criar um problema de planejamento para ser resolvido. Um exemplo de um arquivo problema que pode ser utilizado para este domínio está ilustrado no algoritmo abaixo.

Algoritmo 3.2 – Arquivo problema do mundo dos blocos em PDDL 2.1

```
1 (define (problem traveltoparisandrome)
2     (:domain travel)
3     (:objects
4         robot1 robot2 - robot
5         wp1 wp2 wp3 wp4 - waypoint)
6     (:init
7         (robot_at robot1 wp3)
8         (robot_at robot2 wp2)
```

```
9           (= (fuel-level robot1) 100)
10          (= (fuel-level robot2) 100)
11          (connected wp1 wp2)
12          (connected wp2 wp3)
13          (connected wp3 wp4)
14          (connected wp3 wp1)
15          (connected wp3 wp2)
16          (connected wp2 wp1)
17          (= (fuel-required wp1 wp2) 40)
18          (= (fuel-required wp2 wp3) 30)
19          (= (fuel-required wp3 wp4) 50)
20          (= (fuel-required wp3 wp1) 35)
21          (= (fuel-required wp3 wp2) 40)
22          (= (fuel-required wp2 wp1) 40)
23      )
24      (:goal (and (robot_at robot1 wp1)
25                  (robot_at robot2 wp3))
26      )
27      (:metric maximize (+(fuel-level robot1)(fuel-level robot2)))
28  )
```

Na linha (1) é definido o nome do problema, na linha (2) o domínio a qual pertence o problema. Nas linhas (3),(4) e (5) são definidos os objetos do problema, neste caso são dois robôs (robot1,robot2) e quatro pontos de referência (wp1,wp2,wp3,wp4). Logo depois, da linha (6) à linha (22) é descrito o estado inicial. Para tanto, é informado onde cada robô está, qual o seu nível de combustível, os pontos que estão conectados e o consumo de combustível para ir de um ponto a outro.

Na linha (23) e (24) são definidos os objetivos e na linha (26) a métrica que é utilizada para resolver o problema. No problema apresentado foi utilizada uma métrica que maximiza a soma do nível de combustível do robô1 e do robô2.

Por meio dos dois arquivos supracitados e do planejador POPF (*pop flags*) é gerado o plano apresentado na tabela 1. Mudando a métrica para minimizar o nível de combustível final, "(+(fuel-level robot1)(fuel-level robot2))", o plano da tabela 2 é gerado. A primeira coluna da tabela mostra o tempo em que a ação planejada deve começar a ser executada, a segunda coluna é a ação que será aplicada e os objetos aplicados, e a última refere-se ao tempo de duração da ação.

Neste trabalho, é apresentado apenas alguns dos recursos oferecidos pela versão 2.1 da linguagem PDDL, sendo que, em Fox e Long (2003), é possível encontrar uma descrição completa e detalhada de tudo que é possível de ser representado utilizando a versão 2.1.

Tabela 1 – Plano com métrica para minimizar o nível de combustível

Início da Ação (s)	**Ação a ser Executada**	**Duração (s)**
0.000	goto_waypoint robot2 wp2 wp3	[60.000]
0.000	goto_waypoint robot1 wp3 wp2	[60.000]
60.001	goto_waypoint robot1 wp2 wp1	[60.000]
60.001	goto_waypoint robot2 wp3 wp2	[60.000]
120.002	goto_waypoint robot2 wp2 wp3	[60.000]

Tabela 2 – Plano com métrica para maximizar o nível de combustível

Início da Ação (s)	**Ação a ser Executada**	**Duração (s)**
0.000	goto_waypoint robot2 wp2 wp3	[60.000]
0.000	goto_waypoint robot1 wp3 wp2	[60.000]

3.4 Trabalhos Relacionados

A utilização de planejamento para a tomada de decisão está presente em diversos trabalhos relacionados a robótica, alguns deles serão apresentados abaixo.

Em Cantoni (2010) faz-se uma avaliação do uso do PDDL dentro do planejamento de missões para robôs aéreos. A linguagem em questão foi utilizada para gerar automaticamente as sequências de ações necessárias para o cumprimento de missões aéreas. O trabalho destacou que a linguagem PDDL possui mecanismos interessantes para a geração de planos temporais quando múltiplos veículos são considerados, efetua cálculos de tempo complexos e explora o paralelismo e a concorrência das ações de maneira automática. Apesar disso, há a impossibilidade de utilizar funções como seno, cosseno e raiz quadrada, o que dificulta a representação de problemas com cálculos complexos.

Santos (2009) implementou um sistema simulado de navegação autônoma para robôs móveis baseado na arquitetura híbrida AuRA(*Autonomus Robot Architecture*). No intuito de realizar o planejamento das atividades a serem desenvolvidas, utilizou-se a linguagem PDDL. Desta forma, o sistema selecionaria o conjunto de ações que o levaria a atingir o objetivo pré-estabelecido. O autor destaca que os planos foram gerados de acordo com o esperado, com a eficiência e a velocidade esperada, e que a arquitetura proposta mostrou-se bastante funcional, provando ser flexível e modular.

Cashmore et al. (2014) utiliza a linguagem PDDL para planejamento de tarefa temporal de veículos subaquáticos autônomos. A ideia é que o veículo consiga fazer missões de inspeção sem a intervenção humana otimizando o tempo gasto para completar a missão. Foi feita a combinação de planejamento de rota e planejamento de tarefa. O planejamento de rota é usado para gerar um conjunto de pontos que será utilizado como base para o planejamento de tarefa. Ao utilizar o planejamento temporal e uma métrica para diminuir a duração da execução do plano, o planejador conseguiu escolher pontos em que havia uma melhor visibilidade dos pontos em que se precisava fazer inspeção. Desta forma, diminuía a

quantidade de inspeções necessárias e, consequentemente, o tempo de execução do plano.

Crosby e Petrick (2014) investiga como os problemas de planejamento centralizado, cooperativo e multi-agente com restrições de ações e agentes heterogêneos podem ser codificados com algumas pequenas adições no PDDL e como esses domínios codificados podem ser resolvidos através de uma tradução para o planejamento temporal.

Sanelli et al. (2017) implementa um sistema robótico no qual utiliza planejamento condicional para gerar e executar interações de curto prazo em um ambiente público. O autor faz uso de dois componentes que já são utilizados com sucesso para o planejamento: o ROSPlan que será comentando posteriormente e o *Petri Net Plans* que é um *framework* para colaboração e coordenação no sistema multirrobô.

Liang et al. (2017) criou um *framework*, que combina o planejamento automático e programação por demonstração, com o intuito de que os usuários que não possuem conhecimento em programação sejam capazes de programar um robô. Nos experimentos, avaliou-se a utilização do *framework* por pessoas sem conhecimento em programação e concluiu que o nível de representação de uma ação em termos de precondição e efeitos é adequada para este público e que eles podem ensinar a um robô um novo modelo de ação que possa ser usado por um planejador automático.

Chouhan, Singh e Niyogi (2015) trabalhou com problemas que não podem ser resolvidos apenas por um único agente, logo ele fez uma abordagem centralizada de planejamento multiagente com ação conjunta e desenvolveu um novo planejador para multiagente. Os autores apresentam em seu trabalho um dos principais problemas de sistemas multiagente e do planejamento centralizado que é o grande número de objetos a se trabalhar, uma vez que o agente é considerado como um objeto do plano. Sendo assim, o espaço de estados fica ainda maior e mais difícil de encontrar uma solução, entretanto é confirmado que em algumas situações tratar um problema de multiagente como de agente simples retorna planos menores e mais fáceis de serem resolvidos.

4 Planejamento de rota

Quando um robô autônomo opera em um determinado ambiente é necessário que ele consiga se locomover de um ponto ao outro sem que colida com obstáculos, sendo necessário um sistema de navegação que gere a trajetória e acompanhe a execução do plano.

O planejamento da trajetória pode abranger diversos aspectos, como a movimentação entre obstáculos fixos e móveis, coordenação da movimentação entre múltiplos robôs, raciocínio sobre incerteza para movimentos baseados em sensores, entre outros. Algumas das abordagens mais utilizadas para resolver problemas de planejamento de movimento são: decomposição em células, decomposição em campo potencial e *roadmaps*. Cada método tem as suas particularidades e nem sempre conseguem resolver todos os problemas.

4.1 Decomposição em células

O método de decomposição em células consiste em dividir o espaço livre em um número finito de regiões simples que podem ser chamadas de células. Quando a intersecção de duas células não é nula, significa que estas duas células são adjacentes. A partir de células adjacentes é possível construir um grafo não-dirigido chamado de grafo de conectividade.

Com o grafo de conectividade o problema de planejamento de caminho pode ser resolvido por meio de um algoritmo de busca em grafo discreto e o resultado da busca gera uma possível sequência de células que leva o robô do ponto inicial ao ponto final.

A decomposição das células podem ser feitas por um método exato ou aproximado. O método exato, como o próprio nome já diz, decompõe o espaço livre em um conjunto de células que cobre exatamente o espaço livre. Para isto, ele deve permitir que as células tenham formas irregulares nos lugares em que são adjacentes aos limites do espaço livre. É também chamado de método completo por permitir que um caminho entre duas quaisquer configurações seja obtido sempre que for possível.

No método aproximado, as células possuem uma forma definida previamente e devem ser simples. Apesar deste método não ser completo, pode-se ajustar a sua precisão ao diminuir o tamanho das células utilizadas até que consiga encontrar um caminho, caso este exista. É relevante salientar que, ao diminuir o tamanho das células, é requerido mais memória de armazenamento e maior tempo de processamento. Contudo, este método apresenta a vantagem de ser mais simples e é o mais utilizado na prática (OTTONI; LAGES, 2003).

Uma forma de representar o ambiente é utilizar o mapa de *bit*, este tem a mesma quantidade de dimensões que o problema, bem como adota que cada célula representa uma região quadrada de lado arbitrário. Cada célula recebe o valor de "1"se existir interseção com algum obstáculo e "0"caso a célula esteja completamente vazia. Para fazer o planejamento da trajetória deve-se utilizar apenas as células marcadas com "0". A figura 7 ilustra uma decomposição tirada de Ottoni e Lages (2003).

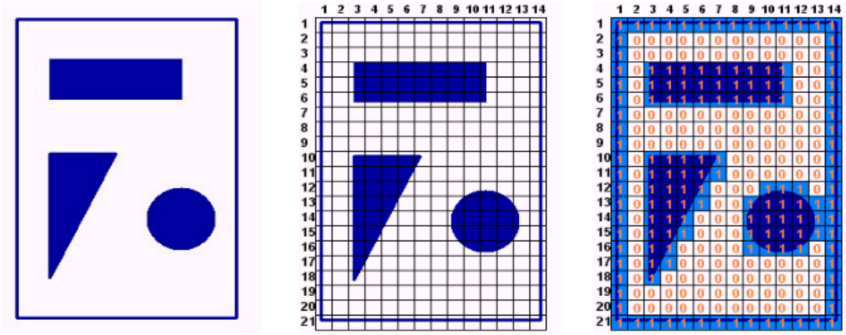

Figura 7 – Decomposição em células pelo método aproximado utilizando mapa de bit (OTTONI; LAGES, 2003)

4.2 Decomposição em campo Potencial

O método de decomposição em campo potencial considera que forças atuam sobre o robô, em que um potencial atrativo é gerado para a posição que é objetivo do robô, enquanto que um potencial repulsivo é gerado pelos obstáculos, assim como representada na figura 8. O potencial atrativo é gerado independente dos obstáculos e o potencial repulsivo é gerado independente do objetivo.

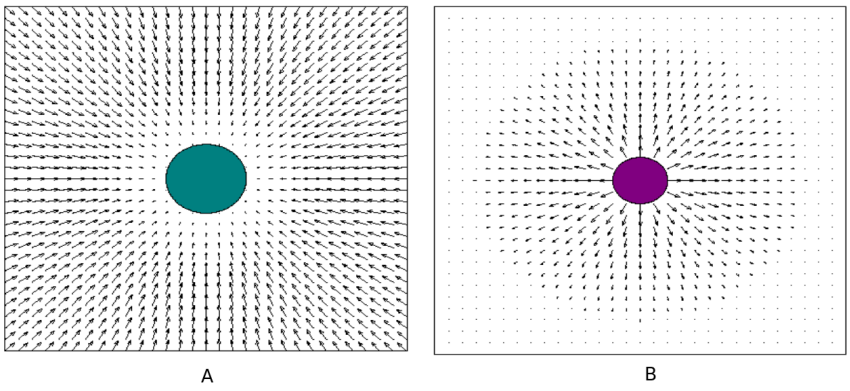

Figura 8 – Representação da decomposição em campo potencial (MEZENCIO, 2002)

O campo potencial resultante é a soma vetorial da força atrativa direcionada para a meta e das forças repulsivas proveniente dos obstáculos(FARIA, 2006), uma vez que estas forças agem no robô. Caso ele siga o gradiente, espera-se que atinja seu objetivo evitando os obstáculos.

É uma técnica utilizada para planejamento de trajetória de robôs que necessitam navegar por ambientes desconhecidos, pois apresenta um baixo custo computacional (FARIA, 2006). A sua maior fraqueza é que não há garantia que o robô vá alcançar o seu objetivo, podendo cair em um problema de mínimo local. Estes são pontos em que o robô guiado pelo algoritmo fica preso, visto que todas as saídas ao seu redor possuem o mesmo peso atrativo e repulsivo.

4.3 *Roadmaps*

Os *roadmaps* como o nome já diz, são caminhos pré-definidos a partir de regras (métodos). Sua finalidade consiste em encontrar rotas candidatas para fazer parte do *roadmap* por meio de uma distribuição uniforme sobre o espaço de configuração. Os caminhos livres de colisão são mantidos (AMATO; WU, 1996). Assim que o *roadmap* for obtido, o planejamento deve, então, conectar a posição inicial e final do robô ao *roadmap* e buscar um trajeto entre estes dois pontos. Caso exista um caminho, o mesmo deve ser encontrado e dividido em três partes: o ponto inicial ao *roadmap*, um caminho no *roadmap* e, por fim, do *roadmap* para o ponto objetivo.

Existe vários métodos baseados nesta ideia para gerar os *roadmaps*, pode-se destacar o grafo de visibilidade, diagrama de Voronoi, rede de caminho livre e silhueta.

4.4 Planejamento de caminho para multirrobôs

O planejamento de caminho para multirrobôs pode ser feito utilizando qualquer uma das três técnicas mostradas anteriormente, desde que haja um algoritmo de coordenação para os robôs.

Svestka e Overmars (1995) introduz o seu trabalho demonstrando sobre as abordagens centralizada e distribuída para o planejamento de rota para multirrobôs, sendo que a abordagem centralizada trata o problema de multirrobôs como se fosse um problema de um único robô composto que apresenta inúmeros graus de liberdade. Desta forma, é mais simples de resolver o problema, porém, acaba tendo um espaço de configuração maior e, como consequência, um maior custo de processamento.

Já a abordagem descentralizada pode utilizar como estratégia a geração de planos individuais para cada robô de forma independente e, em um segundo estágio, coordenar a utilização destes caminhos para que os robôs não colidam uns com os outros.

Berg e Overmars (2005b) apresenta um método para movimentação em ambiente dinâmico que utiliza *roadmap*. O sistema é dividido em duas etapas, na primeira etapa o *roadmap* é construído para a parte estática do cenário, sem considerar os obstáculos dinâmicos e a dimensão do tempo, isto pode ser feito utilizando um método padrão

de RPN(*Probabilistic Roadmap planner*). Na segunda fase, é feita uma busca por uma trajetória quase ideal de tempo entre o ponto de início e o objetivo no *roadmap* sem que colida com os obstáculos dinâmicos.

Ge e Cui (2002) propõe uma abordagem utilizando campo potencial para obstáculos e objetivos móveis. O potencial atrativo é definido por uma função da posição relativa e da velocidade do alvo em relação ao robô. O potencial repulsivo também é definido como a velocidade e a posição relativa do robô em relação aos obstáculos. Consequentemente, a força resultante é definida como o gradiente negativo do potencial em termos de posição e velocidade em vez de apenas posição. As novas definições das funções potenciais e da força resultante permitem que o robô rastreie o alvo da maneira desejada.

Fahimi, Nataraj e Ashrafiuon (2009) utiliza campo potencial harmônico para desviar dos obstáculos dinâmicos, sendo que todos os robôs utilizados são considerados como obstáculos dinâmicos para os outros, assim como no trabalho de Ge e Cui (2002). O caminho de cada robô é planejado com base na mudança de posição dos outros robôs e na posição dos obstáculos estacionários e em movimento.

O método de decomposição em células para multirrobôs é semelhante ao aplicado a problemas com um único robô. O primeiro passo é definir o espaço livre do mapa pelo método aproximado ou exato, depois, basta aplicar um algoritmo que consiga encontrar o caminho de forma que desvie de todos os outros robôs e obstáculos móveis. Um exemplo de trabalho que utiliza a decomposição em células em problemas com multirrobôs está descrito em Siméon, Leroy e Lauumond (2002).

Além das abordagens aqui descritas, uma das técnicas mais simples e utilizadas para a coordenação de robôs em ambientes dinâmicos é o *priorized planning* (ERDMANN; LOZANO-PEREZ, 1987), que consiste em determinar uma prioridade para cada robô. Os caminhos são definidos a partir do robô que tem maior prioridade para o de menor prioridade por um algoritmo de busca, por exemplo, o A* ou dijkstra. Cada caminho gerado deve evitar a colisão com os obstáculos fixos e com os robôs de maior prioridade que podem ser considerados obstáculos dinâmicos.

Existe diversas adaptações do *classical prioritized planning* ou plano prioritário clássico, que visa melhorar e resolver alguns problemas, que não foram possíveis de serem resolvidos no algoritmo original. Cáp et al. (2015) propõe uma versão revisada do *prioritized planning* que garante fornecer uma solução caso exista um caminho para todos os robôs que atinjam o objetivo evitando a posição de início dos robôs com menor prioridade e as posições objetivo dos robôs com maior prioridade. Também, foi proposta uma versão assíncrona e descentralizada do *prioritized planning*, tanto para a versão clássica como para a versão revisada.

Berg e Overmars (2005a) realizou um estudo sobre a escolha da prioridade dos

robôs, e, concluíram que uma boa escolha da prioridade é crucial para o sucesso do planejador. Além disso, afirmam que o algoritmo pode ser aplicado em situações em que os robôs tem tempo de partida diferentes.

A estratégia do *prioritized planning* já foi utilizada em conjunto com a decomposição de células (SIMÉON; LEROY; LAUUMOND, 2002), *roadmap*(BERG; OVERMARS, 2005a; CÁP et al., 2015) e em conjunto com campo potencial para resolver alguns possíveis conflitos (WARREN, 1990).

5 Planejamento Utilizando ROS

No capítulo 5 é apresentado o funcionamento e a estrutura do *framework* ROSPlan. Esta é uma ferramenta utilizada para estabelecer o planejamento e o acompanhamento da execução de planos envolvendo robôs. Um melhor entendimento da estrutura do ROSPlan é feita através de uma introdução sobre o *middleware* ROS, no qual é utilizado como ambiente de desenvolvimento.

5.1 ROS - Robot Operating System

A grande diversidade de *hardware* na robótica dificulta o compartilhamento e a reutilização de códigos ao escrever *softwares* para robôs. Um sistema robótico heterogêneo abrange diversas áreas do desenvolvimento de *software*, desde o desenvolvimento de drive de baixo nível para a percepção e comunicação até o desenvolvimento de aplicações de alto nível de abstração. Assim, é difícil que um único pesquisador consiga resolver todos os problemas e alcançar resultados satisfatórios (JULIO, 2015).

No intuito de gerenciar a complexidade e facilitar a criação de um sistema robótico, muitos pesquisadores começaram a construir ambientes que auxiliam desde a construção da arquitetura do sistema até a integração com o *hardware* dos robôs. Isto resultou em muitos *softwares* de sistemas robóticos usados tanto no meio acadêmico como na indústria, no qual cada *framework* foi desenvolvido para um propósito particular, em decorrência das fraquezas percebidas de outras estruturas disponíveis, ou para dar ênfase sobre aspectos considerados mais importantes no design do processo(QUIGLEY et al., 2009).

O ROS (*Robot Operating System*) (QUIGLEY et al., 2009) é um desses ambientes que foi projetado para atender um conjunto específico de desafios encontrados ao desenvolver robôs de serviços como parte do projeto STAIR (*STanford Artificial Intelligence Robot*) na Universidade de Stanford. No entanto, a arquitetura resultante é muito mais geral que os domínios de robô de serviço e manipulação móvel. A sua ênfase na pesquisa robótica integrativa de larga escala se mostra útil em uma ampla variedade de situações, à medida que os sistemas robóticos se tornam cada vez mais complexos.

O ROS é um *framework* flexível com o objetivo de promover o desenvolvimento colaborativo de software para a área da robótica (JULIO, 2015). Este *framework* fornece os serviços que se esperaria de um sistema operacional, incluindo abstração de hardware, controle de dispositivos de baixo nível, implementação de funcionalidades comumente utilizadas, troca de mensagens entre processos e gerenciamento de pacotes (ROS, 2007).

As principais características do ROS podem ser resumidas em: uma arquitetura

Peer-to-peer; baseado em ferramentas; multilingual; pequeno; e que seja gratuito e *open-source* (QUIGLEY et al., 2009).

O sistema ROS consiste em um conjunto de processos que podem ser executados em diferentes *hosts*, conectados em uma topologia *peer-to-peer*. Cada um dos nós da rede funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central. Este modo de funcionamento se torna vantajoso se os computadores forem conectados em uma rede heterogênea. A topologia *peer-to-peer* requer algum tipo de mecanismo de pesquisa para permitir que os processos se encontrem no tempo de execução, que no ROS é denominado de *Master*.

Cada pesquisador possui uma preferência para linguagem de programação. As preferências são resultado de decisões que o programador faz por tempo de programação, facilidade de debugar, sintaxe, eficiência no tempo de execução, dentre outras opções. Por essas razões, o ROS foi desenvolvido para suportar quatro linguagens diferentes: C++, PHYTON, Octave e LISP. Para se tornar mais viável, foi utilizado uma linguagem neutra IDL (*interface definition language*) para descrever as mensagens entre os módulos. Esta característica do ROS se faz importante pela proposta de utilizar *softwares* de desenvolvedores diferentes.

Como última característica, pode-se destacar que o ROS é modular, visto que pacotes configuráveis existentes podem ser combinados para realizar uma aplicação específica, e, é baseado em um grande número de pequenas ferramentas que executam várias tarefas como: navegar pela árvore do código fonte, fixar e pegar parâmetros de configuração, visualizar a topologia de configuração, medição de utilização de banda, entre outros.

Um dos fatores que mais colaboram na difusão e o crescimento do ROS é o fato de ser uma plataforma de código aberto e gratuito, sendo distribuído sobre os termos de licença BSD (*Berkeley Software Distribution*), permitindo o desenvolvimento tanto de projetos comerciais quanto de projetos não-comerciais. O ROS envia dados entre módulos usando comunicações entre processos, e não requer que os módulos se vinculem no mesmo executável, permitindo que diversos pesquisadores consigam fazer contribuições de forma paralela.

Os conceitos fundamentais de uma implementação ROS são: nós, mensagens, tópicos e serviços. Os nós são processos que executam computação. Um sistema de controle de robô geralmente apresenta vários nós, cada nó possui um objetivo, podendo ser o controle do *laser*, o controle dos motores da roda, o planejamento do caminho, entre outros. Os nós são combinados em um gráfico e se comunicam entre si usando tópicos de transmissão, serviços e parâmetros.

Uma mensagem é simplesmente uma estrutura de dados que compreende campos digitados que podem incluir estruturas arbitrariamente aninhadas e matrizes, assim

como nas estruturas da linguagem C. As mensagens são utilizadas para fornecer uma comunicação entre os nós, que pode ser feita por meio de tópicos ou serviços.

Os tópicos seguem o padrão de publicador - subscritor. Portanto, as mensagens são enviadas de um nó para um tópico por meio de uma publicação, sendo que o tópico irá receber um nome que é usado para identificar o conteúdo da mensagem. Um outro nó que está interessado em um determinado tipo de dado irá subscrever o tópico apropriado para obter o seu conteúdo.

Embora o modelo de publicação-assinatura baseado em tópico seja um paradigma de comunicação flexível, seu esquema de roteamento "*broadcast*" não é apropriado para transações síncronas, por esta razão são utilizados os "serviços". O serviço é definido por um nome de *string* e um par de mensagens estritamente tipadas, uma para o pedido e uma para a resposta. O nó oferece um serviço e um cliente o utiliza. Para isso, é enviada uma mensagem de solicitação por parte do cliente que fica aguardando a resposta. Referente ao fluxo de informação, tem-se que quando a mensagem é enviada por publicador/subscritor é de forma unidirecional, da origem para os destinos, enquanto que ao utilizar um serviço, as informações são bidirecionais.

Com a finalidade de que consiga fazer essa comunicação entre os nós é preciso que seja feito um gerenciamento utilizado o "ROS Master", também chamado de "nó mestre". O "ROS Master" fornece serviços de nomeação e registro para que os outros nós individuais do sistema ROS consigam localizar uns aos outros, uma vez que esses nós se localizam e se comunicam entre si.

Por exemplo, considere dois nós, um nó do sensor, e um nó que informa a distância que as extremidades do robô estão de algum obstáculo. Uma típica sequência de eventos é dada pelo nó do sensor notificando o mestre que deseja publicar as distâncias lidas no tópico "distância". Após este evento, o sensor publica os dados para o tópico "distância". No entanto, ninguém está assinando esse tópico ainda, ou seja, nenhum dado é realmente enviado. Quando o nó "distância do obstáculo" desejar se inscrever no tópico "distância" para verificar se existe alguma informação disponível, ele faz uma solicitação ao nó mestre. Agora que o tópico "distância" tem um editor e um assinante, o nó mestre notifica o nó "sensor" e o nó "distância do obstáculo" sobre a existência um do outro para que eles possam iniciar a transferência de dados entre si. A figura 9 mostra cada etapa deste processo.

Atualmente, o ROS possui 12 distribuições, sendo elas: "ROS Box Turtle", de março de 2010; "ROS C Turtle", de agosto de 2010; "Ros Diamondback", de março de 2011"; "ROS Electric Emys", de agosto de 2011; "ROS Fuerte Turtle", de abril de 2012; "ROS Groovy Galapagos", de dezembro de 2012; "ROS Hydro Medusa" de setembro de 2013; "ROS Indigo Igloo" de julho de 2014; "ROS Jade Turtle" de março de 2015; "ROS Kinetic Kame" de março de 2016; "ROS Lunar Loggerhead" de março de 2017 e o "ROS

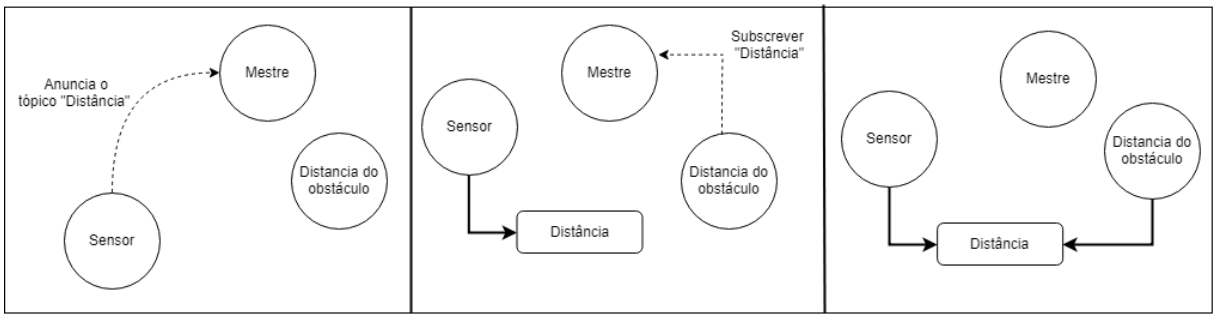

Figura 9 – Etapas da comunicação

Melodic Morenia" março de 2018, sendo que quatro ainda estão em suas vidas úteis, são elas: Indigo, Kinectic, Lunar e Melodic (ROS, 2007).

Como já dito anteriormente, uma das grandes vantagens do ROS é a quantidade de trabalhos e pacotes desenvolvidos por diversos pesquisadores. Entretanto, muitos destes pacotes apresentam falta de compatibilidade, gerada principalmente pela falta de suporte dada pela comunidade, especialmente nos que possuem desenvolvimento independente. Deste modo, o pesquisador acaba tendo que tentar fazer adaptações no código de terceiros com intuito de compatibilizar softwares, mas que nem sempre é possível por falta de compatibilidade de bibliotecas. Contudo, o pesquisador acaba buscando um conjunto de pacotes compatíveis, mesmo que não sejam os mais adequados (NOGUEIRA, 2018).

5.2 ROSPlan

Um pacote relevante para o projeto desenvolvido é o ROSPlan (CASHMORE et al., 2015). Este *framework* faz a integração de um planejador de tarefas com um sistema ROS. A arquitetura é altamente modular e fornece ferramentas para:

- Gerar o estado inicial em PDDL a partir dos dados do sensor e dados armazenados na base de conhecimento;
- Automatizar a chamada do planejador;
- Realizar o processamento e validação do plano;
- Lidar com o despacho de ações levando em consideração a mudança do ambiente e a falha na ação;
- Fazer a tradução de uma ação planejada em uma ação ROS para os controladores de nível inferior.

O ROSPlan inclui dois nós, o nó da base de conhecimento (KB) e o nó do planejamento do sistema (PS), sendo demonstrado uma visão geral na figura 10. O retângulo

vermelho representa o *framework* ROSPlan com os seus respectivos nós. As informações que os sensores adquirem do ambiente são tratadas e passadas constantemente para o ROSPlan, que utiliza estes dados para construir as instanciações dos problemas de planejamento e informar o envio do plano (retângulos azuis). As ações são despachadas como ações do ROS e executadas por controladores de nível inferior (retângulo verde) que respondem de forma reativa a eventos imediatos e fornecem *feedback*.

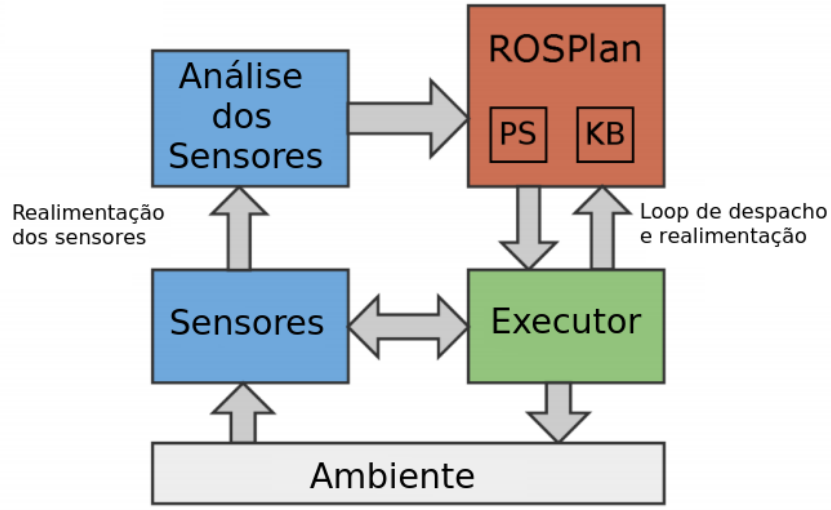

Figura 10 – Visão geral do *framework* ROSPlan. Adaptado de Cashmore et al. (2015)

A base de conhecimento(KB) é simplesmente um conjunto de interfaces para coletar um modelo atualizado do ambiente. Já o nó do planejamento do sistema (PS) faz toda parte de planejamento, incluindo: a geração do estado inicial a partir dos dados armazenados na base de conhecimento, a transmissão da instância de problema do PDDL para o planejador, a execução do pós-processamento dos dados validando o plano e o envio de cada ação decidindo quando é imprescindível replanejar.

A aquisição de conhecimento está relacionada a formação da base de conhecimento por meio das informações dos sensores, das informações fornecidas pelo usuário, das informações adquiridas do domínio e das informações obtidas durante a execução do plano. Estas informações são utilizadas para gerar o arquivo problema em PDDL; fazer a transformação da ação PDDL em uma ação ROS que pode ser executada por um robô; e notificar o planejador caso exista uma modificação no ambiente que invalide o plano.

Por meio dos dados fornecidos pela base de conhecimento, o sistema de planejamento constrói uma instanciação do problema PDDL e envia para um planejador externo. Com o plano gerado é construído um filtro que é utilizado para verificar a validade do plano durante a execução. Além disso, o sistema de planejamento é responsável por despachar as mensagens e fazer o replanejamento caso o filtro do plano seja violado.

As ações são enviadas a partir de uma associação entre uma mensagem PDDL e uma ação do ROS. A tradução da mensagem é feita por um componente separado

a partir dos dados da base de conhecimento. Sendo assim, uma mensagem em PDDL do tipo (goto_waypoint robot1 wp1 wp3) é traduzida em uma mensagem ROS com as coordenadas reais dos pontos wp1 e wp2.

O ROSPlan oferece quatro formas diferentes de enviar as mensagens de acordo com dois parâmetros que são definidos no momento de executar o pacote. Os parâmetros são:"*dispatch-concurrent*" e "*dispatch-on-completion*", e as suas combinações descreve os seguintes comportamentos:

- "dispatch_concurrent && dispatch_on_completion": As ações são enviadas todas de uma vez.
- "!dispatch_concurrent && dispatch_on_completion": As ações são enviadas assim que a anterior é finalizada.
- "!dispatch_concurrent && !dispatch_on_completion": As ações são enviadas desde que já tenha passado o tempo em que foram planejadas para acontecer e assim que a ação anterior tenha sido completada.
- "dispatch_concurrent && !dispatch_on_completion": As ações são enviadas no exato momento em que elas foram planejadas.

Durante a execução do plano pode ser necessário o replanejamento que é baseado na reformulação do problema e pode se dar por três razões, que são: a ação que está sendo executada retorne falha; a base de conhecimento informa ao planejador uma alteração que invalida o plano ou aparece uma nova informação importante para o plano; ou quando a ação atual supera significativamente o tempo planejado.

No primeiro caso dos citados acima, como o *loop* de despacho da ação já está quebrado, é necessário, apenas, que seja gerado uma nova instanciação do problema a partir do modelo atual do ambiente e depois construir um novo plano. Já nos casos 2 e 3 apontados no tópicos acima é necessário primeiro parar a execução do plano para depois começar o replanejamento.

5.3 Trabalhos com o ROSPlan

A importância e relevância do ROSPlan pode ser comprovada com a diversidade de trabalhos que estão sendo publicados utilizando o pacote por completo, parte do pacote, ou que faz alguma modificação e complementação nele. Nesta seção, alguns destes trabalhos serão citados.

5.3.1 Short-Term Human-Robot Interaction through Conditional Planning and Execution

O trabalho de Sanelli et al. (2017) trata de interações homem-robô de curta duração em cenários em que um robô é colocado em um espaço interagindo com usuários que não tiveram nenhuma experiência com robô. Esta forma de interação é caracterizada por: utilizar usuários que não teve nenhum treinamento e não estão cientes das capacidades do robô; cada interação é assumida como sendo executada por um usuário diferente; e as interações são de curto tempo.

Foi desenvolvido então, um método e um sistema robótico usando o planejamento condicional para gerar e executar interações de curto prazo por um robô implantado em um ambiente público. O método proposto integra dois componentes já utilizados com sucesso para planejamento em robótica: ROSPlan e Petri Net Plans.

A utilização de um planejamento condicional foi necessário, pois as interações com usuários sem treinamento não podem ser previstas a priori, e avaliar o estado atual completo não é fácil para o robô. Logo, o planejamento clássico não é adequado para o processo já que, sem o raciocínio *offline* sobre contingências diferentes, é possível que o novo plano resulte em um estado sem saída ou interação repetida.

Sanelli et al. (2017) teve como principal objetivo fornecer uma estrutura para a geração de planos condicionais robustos que possam lidar com a incerteza no estado inicial. Para tanto, foi proposta uma arquitetura multi-camadas que inclui um Sistema de Planejamento, um Tradutor e um Executor de Planos. A sua estrutura é mostrada na figura 11. O Sistema de Planejamento contém a formulação do curto prazo, como um problema de planejamento condicional, que gera um plano usando um planejador condicional. O plano é, então, convertido em um Plano Petri-Net (PNP) e, finalmente, é executado pelo módulo executor PNP.

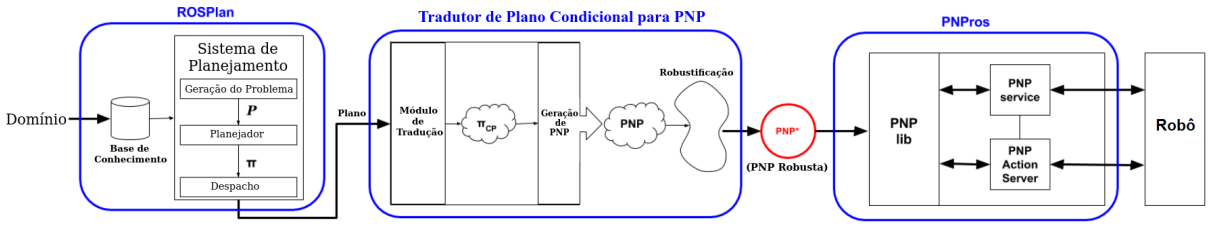

Figura 11 – Visão Geral do Sistema Proposto por Sanelli et al. (2017) (Adaptada)

5.3.2 Autonomous Search-Detect-Track for Small UAVs

No artigo de Morris et al. (2017) um sistema é descrito para procurar, detectar e rastrear autonomamente um objeto de interesse com um pequeno veículo aéreo não tripulado (VANTs). O veículo recebe uma ou mais áreas para pesquisar. Caso o objeto

seja detectado, o VANT segue o alvo, mantendo uma distância fixa e centralizada no plano da imagem. Já se o objeto for perdido, o VANT irá procurar o alvo.

A principal contribuição da pesquisa de Morris at al. é uma estrutura que fornece uma interface entre planejamento de alto nível, sensoriamento e controle de baixo nível para resolver o problema de procurar, detectar e rastrear objetos de interesse com VANTS de forma autônoma. Como uma motivação extra, foi utilizada a situação problema de procura e rastreamento de caçadores de rinocerontes. Para tal, os VANTS são equipados com sensores noturnos e diurnos para pesquisar em uma grande área e localizar caçadores furtivos em tempo hábil para que os guardas florestais possam ser enviados para deter os caçadores ilegais antes que eles ataquem.

Alguns dos desafios técnicos na detecção, navegação e tomada de decisões para esta aplicação incluem escolher onde procurar, distinguir - animais de caçadores com base no movimento - e rastrear um alvo hostil tentando evitar a captura. Com a incerteza do local de destino, torna-se essencial planejar um caminho que garanta a cobertura suficiente de uma área enquanto usa-se dados, como o local de destino conhecido anteriormente para focar a pesquisa e não violar as restrições de recursos. No momento da execução, o veículo deve basear-se em uma classificação robusta da imagem em tempo real para detectar um alvo em potencial, além de contar com um controlador visual baseado em imagem para manter o alvo visível.

A arquitetura utilizada para resolver o problema está representada na figura 12. Um planejador de tarefas de alto nível alterna entre dois objetivos da missão: pesquisar e rastrear. O “sistema de busca” aciona um sistema de controle baseado na trajetória combinado com um sistema de detecção para identificar um objeto de interesse. O “sistema de rastreio” faz a combinação entre um sistema IBVS (*Image-Based Visual Servoing*) e um rastreador para seguir o alvo. O envio das ações do planejamento de alto nível assegura que o sistema alternará continuamente entre pesquisar, desde que o objeto não esteja em exibição, e rastrear quando o objeto for encontrado.

5.3.3 Generating Symbolic Representation from Sensor Data: Inferring knowledge in Robotics Competitions

Rodríguez-Lera, Martín-Rico e Matelián-Olivera (2018) desenvolveu um sistema que faz a transformação de dados coletados por sensores em conhecimento simbólico. Estes propõem que um componente de inferência dedicado a transformar a informação do sensor da camada sub-simbólica em uma camada simbólica.

A arquitetura do trabalho divide o controle do robô em três subsistemas: deliberativo, reativo e motivacional. O subsistema motivacional favorece comportamentos não monótonos no robô. O subsistema reativo visa fornecer comportamentos instantâneos

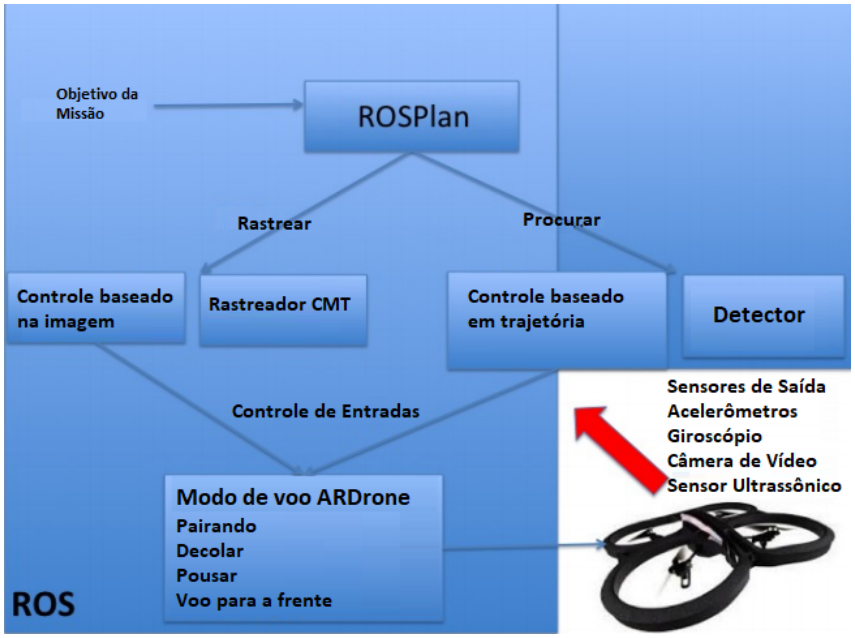

Figura 12 – Arquitetura utilizada na procura de caçador. Adaptada de Morris et al. (2017)

para resolver situações reais e não esperadas. Já o subsistema deliberativo é aquele que gerencia informações simbólicas para gerar comportamentos de longo prazo.

Ambos os subsistemas são executados no ROS. Particularmente, cada subsistema tem componentes específicos: o Subsistema Reativo é executado em uma Máquina de Estado Finito; o Subsistema Deliberativo executa o ROSPlan; e o Subsistema Motivacional, que é um componente desenvolvido em ROS, adiciona um certo grau de entropia ao sistema para evitar comportamentos repetitivos durante as competições de robótica.

O componente de interface de conhecimento converte informações obtidas no subsistema reativo para o modelo simbólico no subsistema deliberativo. Esse intercâmbio de informações é formalizado usando o padrão PDDL 2.1., visto que quando a informação é gerada, o ROSPlan é alimentado com essa informação para gerar novos planos e tomar decisões. A representação destes subsistemas e a forma como eles se conectam está representado na figura 13.

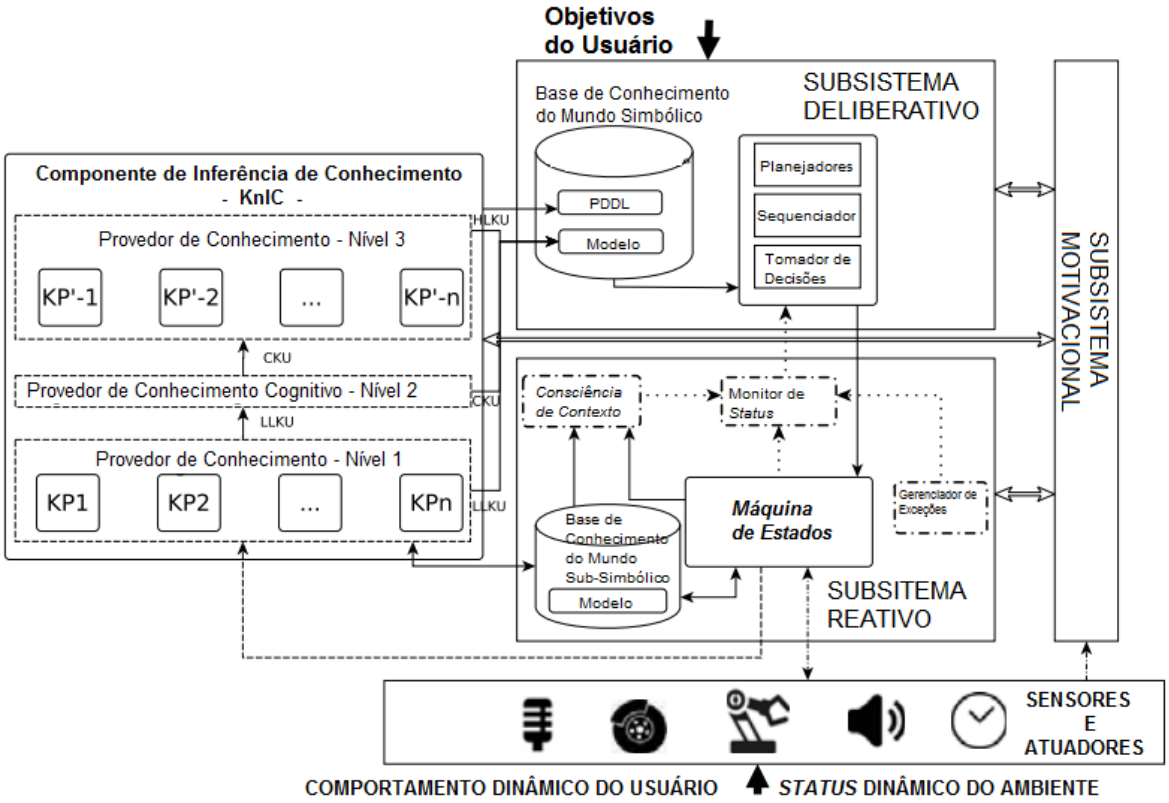

Figura 13 – Representação (adaptada) da topologia hibrida sugerida por Rodríguez-Lera, Martín-Rico e Matelián-Olivera (2018)

6 Desenvolvimento da Proposta

A fim de que múltiplos robôs consigam atuar em um mesmo ambiente é necessário fazer uma coordenação entre eles, de forma que executem as tarefas sem interferir ou obstruir os caminhos dos outros robôs no ambiente. Além disso, é desejado que em um sistema com múltiplos robôs exista a cooperação entre eles para executar tarefas que não poderiam ser realizadas por apenas um robô. Neste capítulo, serão mostradas as alterações feitas no pacote ROSPlan, assim como os pacotes desenvolvidos para uma melhor execução de planos com mais de um robô atuando no ambiente.

6.1 Planejamento e Método de Despacho

Como comentado no Capítulo 1, um dos objetivos deste trabalho é fazer uma adaptação no pacote ROSPlan para que seja possível executar planos com mais de um robô atuando em um mesmo ambiente, de forma que os robôs cooperem na realização das tarefas para alcançar um objetivo global.

O planejamento das tarefas é feito por um planejador PDDL2.1 e os robôs são tratados como objetos do plano. Caso os robôs sejam diferentes, isto deve ser definido durante a elaboração do domínio, separando os robôs em tipos de objetos. Na elaboração das ações do plano, também deve-se especificar o tipo de robô que é capaz de executar aquela tarefa, sendo possível colocar mais de um tipo de robô dependendo da ação.

Durante a fase de planejamento, o planejador aloca qualquer robô que seja capaz de realizar aquele tipo de tarefa e que esteja disponível durante a formação do plano. No intuito de maximizar algum ganho durante o planejamento, pode-se criar um determinado tipo de variável que indique a recompensa; ou a eficiência de utilizar cada tipo de robô e, por fim, utilizar a variável na métrica de planejamento, da mesma maneira que fora mostrado na subseção 3.3.1.

Um ponto crucial para o bom funcionamento do sistema é o despacho das ações a serem executadas. Um plano temporal indica o tempo exato em que uma ação deve começar a ser executada, a sua duração e o momento em que ela deve ser finalizada. Entretanto, ao se tratar da execução de plano em um ambiente real, comumente as tarefas podem ser concluídas antes ou depois do tempo planejado (MUSCETTOLA; MORRIS; TSAMARDINOS, 1998).

Dentre as opções de despacho das ações no ROSPlan, nenhuma delas consegue atender um sistema com mais de um robô, e que apresente ações que dependam da conclusão de uma outra tarefa, de forma paralela e eficiente.

O primeiro método que o ROSPlan oferece é “dispatch-concurrent && dispatch-on-completion” em que todas as tarefas são despachadas de uma única vez, sem nenhum controle. Este tipo de comportamento quase nunca é esperado, uma vez que não é possível executar planos em que uma tarefa dependa da conclusão de uma outra. O segundo e o terceiro método são: “!dispatch-concurrent && dispatch-on-completion” e “!dispatch-concurrent && !dispatch-on-completion”. Ambos possuem o despacho de forma sequencial, sendo que o segundo método apenas se diferencia do terceiro ao despachar as mensagens apenas depois que o tempo em que ela for planejada já tenha passado. Embora o segundo e o terceiro método já permitirem que mais de um robô atue no plano, apenas uma tarefa será executada por vez. Sendo assim, enquanto um robô executa a tarefa recebida, o outro irá permanecer parado.

No quarto e último método, “dispatch-concurrent && !dispatch-on-completion”, as mensagens são enviadas exatamente no momento em que foram planejadas, sendo o método capaz de enviar mensagens para mais de um robô de forma paralela e de conseguir trabalhar com ações dependentes. Entretanto, caso aconteça algum atraso na execução de qualquer tarefa ou o tempo planejado não for suficiente para a execução, o plano retornará falha e é feito um replanejamento. Assim como, caso uma ação finalize antes do tempo programado ou mesmo que o robô já possa executar a próxima ação, ele ficará em espera até o tempo programado para o início da ação seguinte.

Apesar de ser um método positivo para situações em que aconteçam eventos importantes em determinados momentos durante a execução do plano, na maioria das aplicações ele não se mostra muito eficaz por ter que respeitar rigorosamente o tempo planejado, e como dito anteriormente, em aplicações reais dificilmente o tempo de execução será igual ao tempo planejado.

O método sugerido nesta dissertação consiste em criar uma lista a partir do plano temporal em ordem cronológica e despachar as mensagens em sequência, desde que as suas pré-condições estejam todas atendidas. Além disso, é necessário verificar se uma ação que está sendo despachada não atrapalha a execução das que estão em andamento.

Para exemplificar o funcionamento do método sugerido, considere o "mundo dos blocos"sendo executado por dois manipuladores robóticos em um ambiente em que não exista a possibilidade de colisão entre os manipuladores. As possíveis ações são:

- *empilhar(R,B1,B2)*: Significa que um manipulador R irá empilhar o bloco B1 sobre bloco B2.
- *desempilhar (R,B1,B2)*: Significa que o manipulador R irá desempilhar o bloco B1 do bloco B2
- *colocarsobremesa(R,B)*: O manipulador R colocará o bloco B sobre a mesa.

- *retirarmesa(R,B)*: O manipulador R irá tirar o bloco B da mesa.

Os predicados utilizados são:

- livres(B): O bloco B está no topo da pilha e pode ser desempilhado ou o manipulador não está segurando nenhum bloco.
- segurando(R,B): O manipulador R está segurando o bloco B.
- sobre(B1, B2): O bloco B1 está sobre o bloco B2.
- namesa(B): O bloco B está sobre a mesa.

As pré-condições e os efeitos de cada ação estão representados no fragmento de domínio do Algoritmo 6.1. O estado inicial está ilustrado no lado esquerdo e o estado objetivo no direito da Figura 14. Considere o tempo de execução de qualquer uma das ações como 4 segundos. Um possível plano gerado pelo planejador temporal POPF (COLES et al., 2010) está representado na Tabela 3.

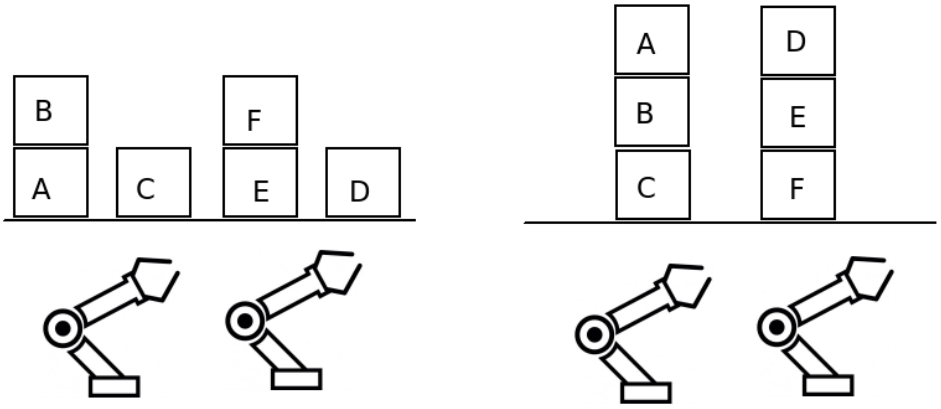

Figura 14 – Estado inicial e final do problema do mundo dos blocos com dois manipuladores

Tabela 3 – Plano gerado para o mundo dos blocos com dois manipuladores

Início da Ação (s)	**Ação a ser Executada**	**Duração (s)**
0.000	desempilhar r1 a b	4.000
4.001	retirarmesa r2 b	4.000
8.002	empilhar r2 b c	4.000
12.003	empilhar r1 a b	4.000
12.003	desempilhar r2 f e	4.000
16.004	retirarmesa r1 e	4.000
20.005	empilhar r1 e d	4.000
24.006	empilhar r2 f e	4.000

Algoritmo 6.1 Representação das ações do domínio do mundo dos blocos

```
(:durative-action empilhar
    :parameters (?r - robot ?b1 ?b2 - box)
    :duration ( = ?duration 4)
    :condition (and
            (at start (free ?b2))
            (at start (hold ?r ?b1)))
    :effect (and
            (at start (not (free ?b2)))
            (at end (not(hold ?r ?b1)))
            (at end (free ?r))
            (at end (free ?b1))
            (at end (above ?b1 ?b2))))

(:durative-action desempilhar
    :parameters (?r - robot ?b1 ?b2 - box)
    :duration ( = ?duration 4)
    :condition (and
            (at start (free ?b1))
            (at start (free ?r))
            (at start (above ?b1 ?b2)))
    :effect (and
            (at start (not (free ?r)))
            (at start (not (free ?b1)))
            (at end (hold ?r ?b1))
            (at end (free ?b2))))

(:durative-action colocarsobremesa
    :parameters (?r - robot ?b - box)
    :duration ( = ?duration 4)
    :condition (and
            (at start (hold ?r ?b)))
    :effect (and
            (at end (namesa ?b))
            (at end (free ?b))
            (at end (free ?r))
            (at end (not(hold ?r ?b)))))

(:durative-action retirarmesa
    :parameters (?r - robot ?b - box)
    :duration ( = ?duration 4)
    :condition (and
            (at start (free ?r))
            (at start (free ?b))
            (at start (namesa ?b)))
    :effect (and
            (at start (not(free ?b)))
            (at start (not(free ?r)))
            (at start (not (namesa ?b)))
            (at end (hold ?r ?b))))
```

O método despacha as mensagens assim que todos os requisitos estejam satisfeitos, mas respeitando a ordem planejada, o que é feito para garantir que a sequência das ações que geram o resultado desejado não seja violada. Por exemplo, ao iniciar a execução do plano, todas as pré condições das ações 1 e 5 são atendidas, sendo a ação 1 (desempilhar r1 a b) e as pré-condições (estar livre o bloco “a”, estar livre o robô “r1” e o bloco “a” estar sobre o bloco “b”); e a ação 5 (desempilhar r2 f e) e suas pré-condições (estar livre o bloco “f”, estar livre o robô “r2” e o bloco “f” estar sobre o bloco “e”). Na hipótese destas duas ações serem executadas no início do plano, o manipulador 1 iria desempilhar o bloco “A” e ao mesmo tempo o manipulador 2 desempilhar o bloco “F”. Depois de finalizada as ações, não existiria nenhuma outra ação do plano que poderia ser executada, e o plano falharia. Percebe-se, então, que a ação 5 só poderá ser executada caso todas as ações anteriores a ela já tenham sido começadas a serem executadas e as suas pré-condições ainda estejam satisfeitas.

O algoritmo de despacho funciona da seguinte forma: inicialmente gera-se uma lista com as ações de forma sequencial, então é verificado se todas as pré-condições da primeira ação estão satisfeitas. Em caso positivo, ela é despachada para ser executada e adicionada a uma outra lista que representa as ações que estão sendo executadas, chamada de lista de execução. Logo que a primeira ação é despachada, começa a verificação da segunda ação. Primeiro é verificado se todas as suas pré-condições foram atendidas e, caso positivo, é verificado se algum efeito da ação atrapalha ou invalida a execução das ações que estão na lista de ações em execução. Caso nenhum efeito interfira nas que estão sendo executadas, então a mesma também é despachada. Esta verificação é feita até que todas as mensagens sejam despachadas. O diagrama de blocos da Figura 15 ilustra o funcionamento.

6.2 Execução das tarefas

Para fazer a execução das tarefas, cada robô recebe uma nomeação diferente, denominada de *namespace*. O *namespace* é utilizado para promover um encapsulamento e evitar conflito de recursos. Isto é extremamente necessário em um ambiente com mais de um robô. Considerando que dois robôs possuem um tópico para a publicação da odometria com o nome */odom* e, na hipótese de estarem integrados em um mesmo sistema, o conflito de nomes dos tópicos é evitado pelo *namespace*. Desta forma, caso seja definido que os *namespace* serão “robo1” e “robo2”, e os *tópicos* terão os nomes /robo1/odom e /robo2/odom, respectivamente.

Cada ação que o robô pode exercer foi implementada em um nó separado e agregada ao robô, sendo que todos estes nós devem sobrescrever o tópico do despacho e verificar se as ações publicadas são destinadas a eles. Quando qualquer ação é publicada no tópico de despacho, todos os nós sobrescritos verificam o que está escrito nesta ação.

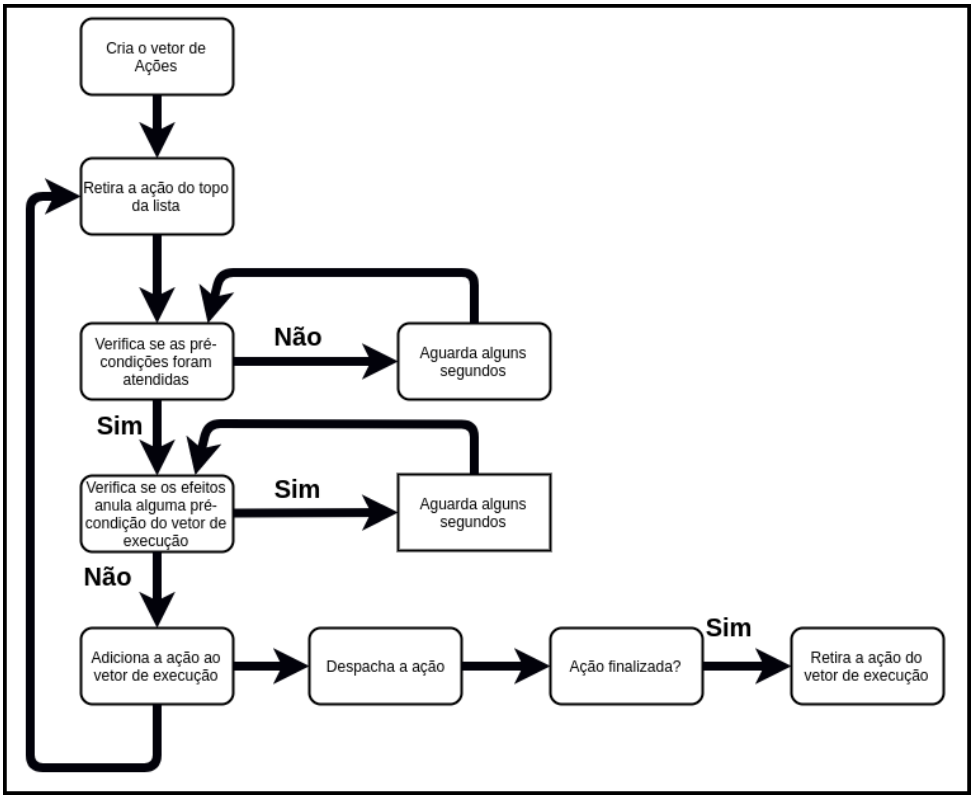

Figura 15 – Representação por diagrama de blocos do funcionamento do despacho das ações

Primeiro, é verificado se a ação corresponde ao robô em questão, depois verifica qual o tipo de ação que se trata, e, por fim, se esta ação está devidamente formada, apresentando todos os argumentos necessários. A tarefa só é executada caso todos estes requisitos forem atendidos. A Figura 16 mostra todos os nós recebendo a mensagem, no entanto, após a verificação, apenas o nó destacado continua ativo e executa a tarefa.

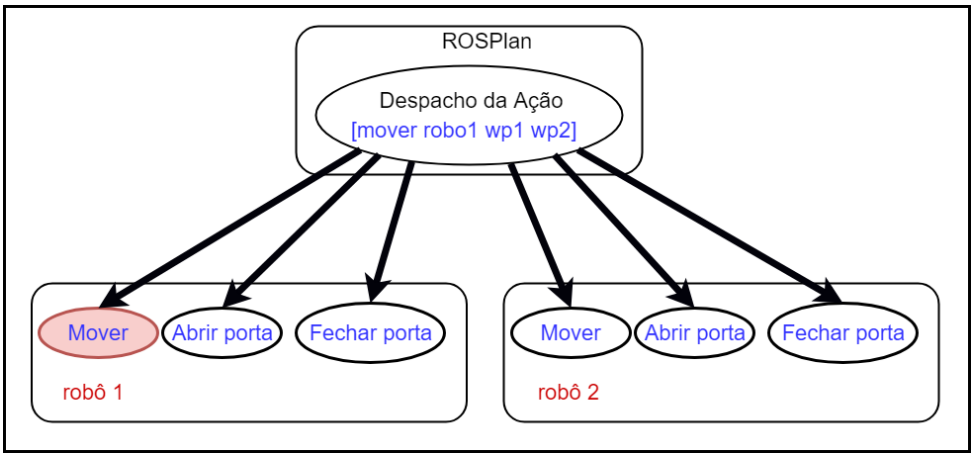

Figura 16 – Recebimento das ações pelos nós dos robôs

A partir do instante que o robô finaliza a execução da tarefa, uma mensagem de conclusão é enviada ao tópico de tarefas concluídas. Esta mensagem atualiza o estado de

execução de tarefas para "finalizada", adiciona os efeitos da ação na base de conhecimento e exclui a ação do vetor de "Ações em execução", que foi comentado na seção 6.1.

6.3 Planejamento do caminho e navegação

No ROS já existem alguns pacotes de navegação, tais como o *move_base*(ROS, a) e o *mrpt_navigation*(ROS, b) que são capazes de mover um robô por um mapa desviando de obstáculos. Todavia, nenhum deles executa o planejamento da rota considerando outros robôs no mapa.

A grande vantagem de considerar o caminho a ser tomado por todos os robôs é conseguir prevenir uma colisão na fase de planejamento, trocando a rota antes de começar a executar o plano. Considere o cenário da Figura 17, levando em consideração que os pontos S1 e S2 representam os robôs e que os pontos G1 e G2 são os seus respectivos pontos de chegada. Se cada robô desconsiderar a existência do outro durante o planejamento, o provável caminho que eles tendem a seguir são os representados pelas setas. Desta forma, no decorrer da execução do plano uma colisão ou um replanejamento seria inevitável.

Considere agora que cada robô consiga perceber um obstáculo que não foi mapeado quando ele está a um quadrado de distância, ao chegar na posição ilustrada na Figura 18.a é observado pelos dois robôs o impedimento de passar por aquele caminho. Então, os dois fariam um replanejamento da rota que novamente resultaria em colisão ou replanejamento no ponto marcado em azul, assim como representado na Figura 18.b.

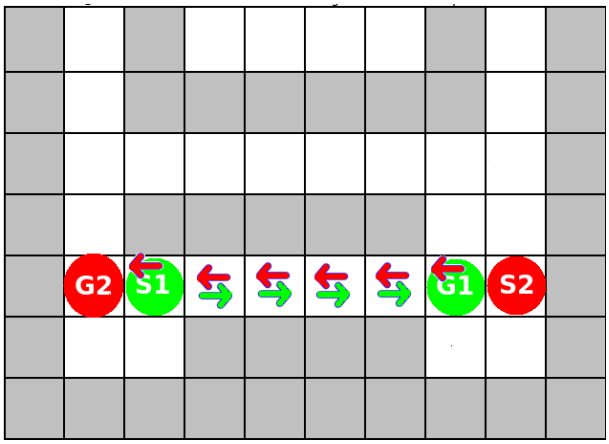

Figura 17 – Caminho gerado sem considerar a existência de outros robôs

Como já citado na seção 4.4, existem diversos trabalhos que realizam o planejamento utilizando multirrobôs. Entretanto, a maioria deles apenas faz o planejamento de alto nível, sem a execução do plano por robôs simulados ou reais. Além do que, até o início do desenvolvimento desta dissertação, também não foi encontrado nenhum trabalho que tenha feito esse tipo de planejamento e navegação utilizando o *framework* ROS.

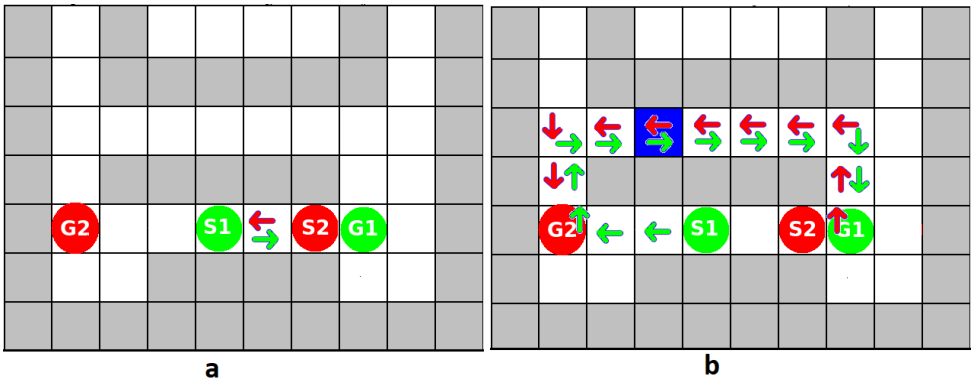

Figura 18 – Ponto de colisão e novo plano gerado

Para gerar os planos dos robôs é utilizado o *prioritized planning*. Sua escolha é devido a simplicidade do algoritmo que consiste em estabelecer uma prioridade diferente para cada robô e gerar o plano de forma sequencial. O robô com maior prioridade tem o plano gerado primeiro, já o que tem menor prioridade é gerado de forma a evitar colisão com o de maior prioridade.

Assim, para conseguir evitar as colisões, o conceito de espaço-tempo é usado, no qual cada robô ocupa uma certa região durante um período de tempo τ (x, y, t). A partir de um mapa W, o ponto de início do robô "i" é "Si" e o seu ponto de chegada "Gi", o algoritmo deve ser capaz de retornar um plano π evitando as trajetórias θ dos robôs com maior prioridade. O pseudocódigo 1 mostra o funcionamento do *Classical Prioritized Planning*.

Pseudocódigo 1 Classical Prioritized Planning

procedure ALGORITHM PP
  $\Delta \leftarrow \emptyset$
  for $i \leftarrow 1...n$ **do**
    $\pi_i \leftarrow$ `BEST_TRAJ`$(\omega, \Delta, S_i, G_i)$
    if $\pi_i = \emptyset$ **then**
      report failure and terminate
    $\Delta \leftarrow \Delta \cup R_i^{\Delta}(\pi_i)$
function BEST_TRAJ$(\omega, \Delta, S_i, G_i)$
  return optimal satisfying trajectory for robot i in ω that avoids regions Δ if it exists, otherwise return $\emptyset$

Com o intuito de adaptar este algoritmo para uma aplicação em ROS, foram feitas algumas adaptações. O planejamento da trajetória dos robôs é realizado em um nó centralizado, sendo que quando algum robô recebe uma ação que demanda um caminho no mapa faz-se uma requisição ao planejador por um *Serviço ROS*. Este é definido por um par de mensagens, uma que faz a requisição da tarefa e outra que responde. Desta forma, o robô faz a requisição do serviço por meio de uma mensagem com sua posição inicial e sua posição objetivo, e espera a resposta com o caminho que terá que seguir.

O planejador calcula o caminho evitando as trajetórias dos robôs com maior prioridade, uma vez que a prioridade é dada para o robô que fez o requerimento da trajetória primeiro. A qualquer momento pode ser realizada uma nova requisição de caminho para um robô que já completou uma primeira navegação ou para um novo robô que está sendo integrado no plano.

O cálculo do caminho é feito por um algoritmo A* com o acréscimo da variável "tempo". O robô pode atuar das seguintes formas: virar para a direita, virar para a esquerda, ir para frente ou ficar parado, sendo que cada movimento demora uma unidade de tempo. O plano é gerado evitando que dois robôs estejam no mesmo ponto e ao mesmo tempo. Ao gerar a primeira trajetória, todo o caminho τ (x, y, t) é adicionado a um vetor Δ, este vetor representa todos os caminhos ocupados. Para gerar o caminho do próximo robô, primeiro é calculado o caminho mais curto, como se não existisse nenhum outro robô, em seguida verifica-se se o caminho gerado τ (x, y, t) possui algum elemento igual aos elementos do vetor Δ. Caso exista algum elemento igual, um novo caminho é gerado evitando este ponto.

O plano obtido pelo robô nunca apresenta movimentos desnecessários, e sempre tenta deixar o robô em movimento. O robô somente permanece parado quando não existe nenhum caminho em que ele consiga começar a executar.

Do mesmo modo, deve-se atentar a um outro tipo de problema. Quando o plano mostra uma inversão de posição dos robôs, os vetores não identificarão posições iguais no mesmo instante de tempo, mas na execução eles passariam um sobre o outro. Por exemplo, considere que o robô1 esteja na posição (3,4), no instante de tempo t=5, e que no seu plano o próximo ponto seja a posição (4,4), no instante de tempo t=6. Já o robô2 está na posição (4,4), no instante t=5, e sua próxima posição seja (3,4), no instante de tempo t=6. O vetor de caminhos do robô1 seria τ1= [(3, 4, 5) (4,4,6)] e o do robô2 τ2= [(4, 4, 5) (3,4,6). Pode-se verificar que eles não possuem elementos iguais. Entretanto, no ambiente contínuo eles estariam passando pelo mesmo ponto no instante de tempo t=5,5.

Na resolução do problema, é feita a seguinte verificação: para cada elemento do plano gerado para o robô1, faz-se uma comparação com as coordenadas planejadas para o robô2, no mesmo instante de tempo. Logo após, é verificado se as coordenadas do robô1 no tempo "t" são iguais as coordenadas do robô2 no instante de tempo "t+1" e, em caso positivo, é feita a verificação inversa em que o instante "t" do robô2 é comparado com o instante "t+1" do robô1. Se novamente forem iguais é detectada uma possível colisão e este caminho é evitado pelo robô de menor prioridade.

Assim que o robô recebe a resposta do serviço com o seu caminho, a trajetória começa a ser executada. O controle da navegação é feito utilizando como entrada apenas os valores fornecidos pela odometria do robô. O robô compara a sua posição e orientação atual com o ponto e a orientação planejada e envia os comandos necessários para os

motores para que esta diferença se torne zero. Logo que o primeiro ponto do vetor de posições é alcançado, o robô envia uma mensagem para o planejador informando que já foi alcançada aquela posição, podendo assim ser retirada do vetor Δ. Consequentemente, quando um novo robô entra no sistema, ele recebe como primeiro valor de tempo o que está na base do vetor Δ.

6.4 Interface Gráfica

Para facilitar os testes e visualização do pacote por outras pessoas, foi desenvolvida uma interface gráfica em que é possível definir a quantidade de robôs desejados no ambiente e os pontos de interesse no mapa. Na interface, já estão pré-definidos os dois mapas utilizados nos experimentos das subseções 6.6.1 e 6.6.2 . A Figura 19 ilustra essa interface: a imagem da esquerda corresponde ao primeiro exemplo e a imagem da direita corresponde ao segundo exemplo.

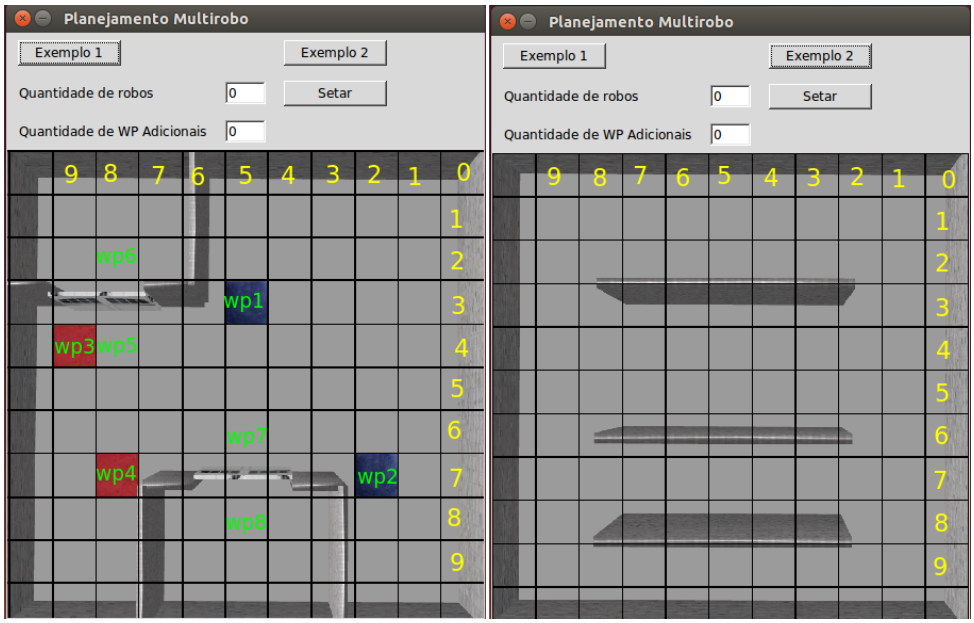

Figura 19 – Interface gráfica desenvolvida

No exemplo 1, já existem 8 *waypoints* pré-definidos que correspondem a pontos importantes para o domínio do exemplo. Os *waypoints* wp1, wp2, wp3 e wp4 são para abrir e fechar as portas e os *waypoints* wp5 wp6 wp7 e wp8 são os pontos de conexão entre os ambientes fechados e a área aberta. Pode ser adicionado mais *waypoints* em qualquer lugar do mapa, sendo necessário indicar a quantidade a mais de pontos desejados, preencher a quantidade de robôs que será utilizado no exemplo e apertar no botão "Setar". Como resultado, a tela da Figura 20 aparece e deve ser preenchida a caixa com os *waypoints* desejados. O primeiro valor corresponde a coordenada em "x" e o segundo valor a coordenada em "y".

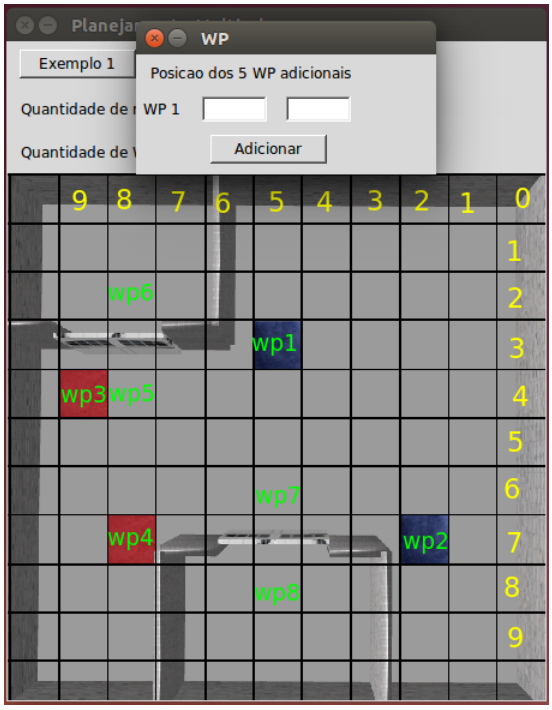

Figura 20 – Tela de preenchimento dos waypoints adicionais

Após adicionar todos os valores, esta caixa some e aparece a da Figura 21. Neste momento, é preenchido o estado inicial e o objetivo. A maioria das instanciações básicas são feitas automaticamente, como o estado das portas e as conexões entre os *waypoints*. Para o estado inicial só é necessário definir onde serão os pontos iniciais dos robôs e para o objetivo pode ser definido os pontos finais, a situação das portas (aberta, fechada) e os pontos que os robôs devem visitar.

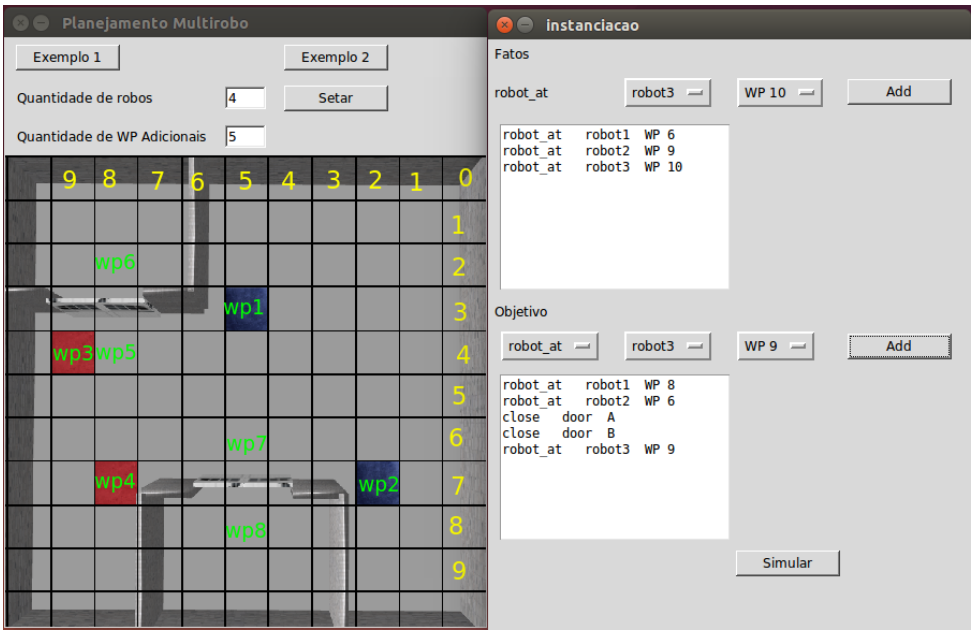

Figura 21 – Tela de instanciação do estado inicial e objetivo

6.5 Detalhamento do funcionamento do Sistema

A interface gráfica executa quatro tarefas. A primeira é definir o mapa que é utilizado para a simulação e carregar os robôs que estão encapsulados em um *namespace* juntamente com suas habilidades. Portanto, a partir das opções fornecidas pela interface gráfica é possível escolher entre o exemplo 1 e 2. A interface carrega os devidos arquivos para gerar o mapa correspondente para a simulação e, referente a quantidade de robôs e a sua capacidade, é gerado o encapsulamento de acordo com o fragmento de arquivo mostrado na Figura 22.

```
  <!-- BEGIN ROBOT 1-->
  <group ns="robot1">
    <param name="tf_prefix" value="robot1_tf" />
    <include file="$(find multi_turtlebot)/launch/one_robot.launch" >
      <arg name="init_pose" value="-x 1 -y 1 -z 0" />
      <arg name="robot_name"  value="Robot1" />
    </include>

        <node name="nav_multirobot" pkg="nav_multirobot" type="navigation"
respawn="false" output="screen"/>

<node name="rosplan_interface_movebase1" pkg="rosplan_interface_movebase"
type="rpmovebase" respawn="false" output="screen">
    <param name="name_robot" value="robot1"/>
    <param name="action_server" value="/robot1/nav_multirobot/plan" />
    <param name="pddl_action_name" value="goto_waypoint" />
    <param name="reset_odometry" value="/robot1/mobile_base/commands/reset_odometry" />
</node>

<node name="rosplan_interface_opendoor1" pkg="rosplan_interface_opendoor"
type="rpopendoor" respawn="false" output="screen">
    <param name="name_robot" value="robot1"/>
    <param name="pddl_action_name" value="open_door" />
</node>

<node name="rosplan_interface_closedoor1" pkg="rosplan_interface_closedoor"
type="rpclosedoor" respawn="false" output="screen">
    <param name="name_robot" value="robot1"/>
    <param name="pddl_action_name" value="close_door" />
</node>

  </group>
```

Figura 22 – Encapsulamento das ações do robô

Os nomes dos robôs são dados de forma incremental, começando por "*robot*1". A posição inicial do robô é ajustada de acordo com a entrada fornecida pelo usuário, sendo a posição z sempre igual a 0. Os únicos pacotes de ações implementados foram os de mover, abrir e fechar a porta. Dentre os parâmetros enviados para estes pacotes, vale ressaltar o "*pddl_action_name*" que indica a ação PDDL a que ele se refere.

A segunda função da interface gráfica é gerar um arquivo com todos os *waypoints* que podem ser visitados. Estes *waypoints* representam apenas pontos de partida e de chegada, não precisando especificar todos os pontos que o robô pode estar presente. No mínimo, deve ser definido como *waypoints* o local de início dos robôs, sendo no exemplo

1 necessário, ainda, os pontos de abrir e fechar a porta e as conexões antes e depois da porta.

A terceira função da interface é criar um arquivo de código que sintetiza o problema a ser solucionado. O arquivo gerado é utilizado juntamente com o arquivo domínio para a geração do plano.

Por fim, a quarta função é realizar a chamada de todos os pacotes necessários para executar a simulação. Além do ROSPlan e dos pacotes que o integra, também é feita a chamada da interface de acompanhamento da execução do plano e o pacote de navegação desenvolvido.

Uma visão geral e simplificada do funcionamento pode ser observada na Figura 23. No bloco com a marcação do número 1 está representado o planejamento feito pelo ROSPlan. A partir do plano gerado é feito o despacho como explicado na seção 6.1. A mensagem é publicada no tópico "/kcl_rosplan/action_dispatch", em que todos os pacotes de execução de ações de todos os robôs sobrescrevem este tópico (representados nos blocos com número 2).

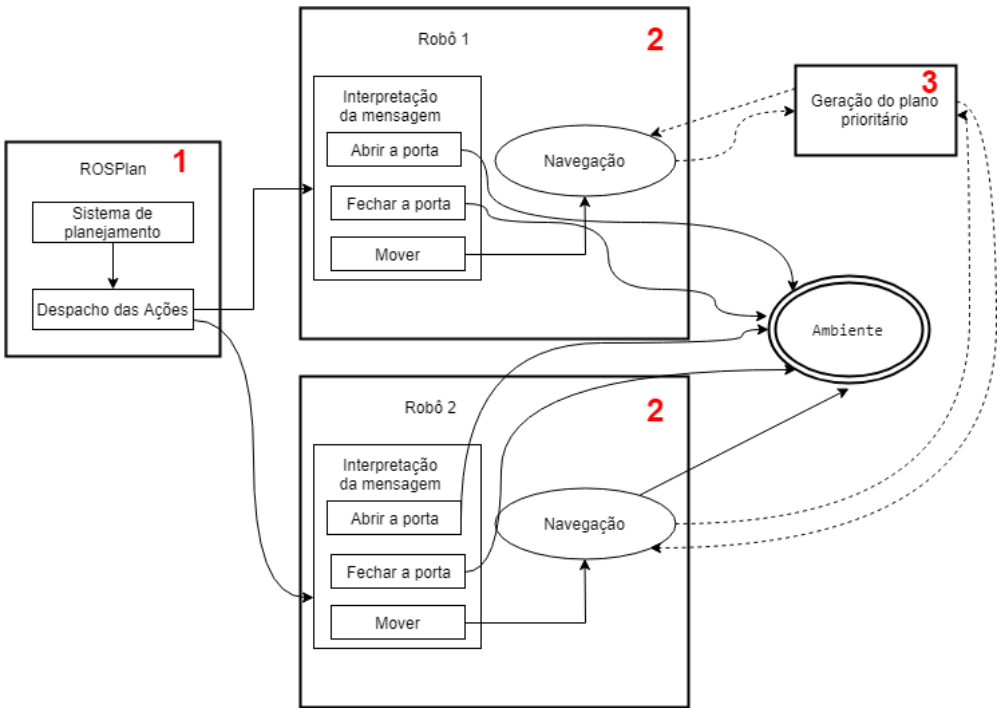

Figura 23 – Diagrama simplificado do funcionamento do sistema

A verificação que é feita para definir qual ação vai ser executada está representada na figura 24.

```
            nh.getParam("name_robot", nomedorobo);
            for(size_t i=0; i<msg->parameters.size(); i++) {
                    if(0==msg->parameters[i].key.compare("v")) {
                            name = msg->parameters[i].value;
                    }
            }
            if(name!= nomedorobo) return;
                // check action name
                if(0!=msg->name.compare(params.name)) return;
                ROS_INFO("KCL: (%s) action recieved", params.name.c_str());

                // check PDDL parameters
                std::vector<bool> found(params.typed_parameters.size(), false);
                std::map<std::string, std::string> boundParameters;
                for(size_t j=0; j<params.typed_parameters.size(); j++) {
                        for(size_t i=0; i<msg->parameters.size(); i++) {
                                if(params.typed_parameters[j].key == msg->parameters[i].key) {
                                        boundParameters[msg->parameters[i].key] = msg->parameters
[i].value;
                                        found[j] = true;
                                        break;
                                }
                        }
                        if(!found[j]) {
                                ROS_INFO("KCL: (%s) aborting action dispatch; malformed
parameters, missing %s", params.name.c_str(), params.typed_parameters[j].key.c_str());
                                return;
                        }
                }
```

Figura 24 – Fragmento do código que verifica se a ação possui os parâmetros necessários e se é destinada ao robô em questão

O primeiro *loop* verifica a mensagem recebida para descobrir a qual robô ela pertence, sendo que a mensagem publicada no tópico "/kcl_rosplan/action_dispatch" é do tipo:

#actionDispatch message

int32 action_id

string name

diagnostic_msgs/KeyValue[] parameters

float32 duration

float32 dispatch_time

A mensagem "diagnostic_msgs/KeyValue" é composta por:

string key

string value

Ao fazer a sintetização do problema pela interface define-se que a "key" referente ao nome do robô seja representada por "v". Assim, a *tag* é procurada na ação recebida e, caso seja igual ao do robô que está fazendo a análise, o algoritmo prossegue com a execução da ação.

As ações de abrir e fechar as portas apenas atuam no ambiente de simulação por

meio de publicações em tópicos do simulador. Já a ação de mover pelo mapa atua por meio de dois serviços, o primeiro é responsável por buscar um caminho para o robô percorrer, sendo necessário fornecer o ponto em que o robô está e o seu objetivo. Como resposta deste serviço, é fornecido um vetor em que cada elemento tem uma mensagem com o tipo de movimento que será executado (seguir, virar, permanecer), o momento que isso ocorrer, a posição e a orientação do robô.

Uma vez que o primeiro serviço encontra e retorna um plano, ativa-se o segundo serviço que faz a execução do plano encontrado. Durante a sua execução são publicadas, em um tópico, as partes do plano que já foram completadas para que aquele ponto possa ser utilizado para um outro plano.

Com a finalidade de facilitar o planejamento, não é considerado o tempo real que o robô leva para executar cada ação, mas é considerado que ele leva o mesmo tempo para cada ação, ou seja, a tarefa de virar, permanecer parado ou seguir é contabilizada como uma unidade de tempo.

Por fim, é criado um vetor que contém todos os planos que estão sendo executados. Desta forma, quando um novo robô solicita um caminho, é utilizado o instante de tempo menor deste vetor para ser o ponto de partida do seu plano.

Ao finalizar a execução de cada ação por completo, o robô publica no tópico "/kcl_rosplan/action_feedback", e com isso é atualizado o status desta ação para "*action achieved*".

6.6 Simulações

Nesta seção serão descritas duas simulações com execução dos robôs utilizando a interface gráfica e outras duas simulações apenas com dados, sem utilizar a interface gráfica desenvolvida. As simulações foram feitas com o intuito de demonstrar o funcionamento das modificações implementadas no pacote ROSPlan e no pacote de navegação desenvolvido.

Nas simulações com execução, utilizou-se o simulador *Gazebo*(KOENIG; HOWARD, 2004) que é uma ferramenta de simulação de robôs 3D de código aberto amplamente apoiada e usada por muitas organizações de pesquisa (CHEN et al., 2009). A comunicação do simulador é feita por trocas de mensagem do tipo publicador/subscritor, o que facilita a sua integração com o ROS. Em todas as simulações utilizou-se robôs *TurtleBot*, como o da figura 25.

Figura 25 – Representação do Turtlebot

6.6.1 Simulação 1

A primeira simulação[1] tem o intuito de verificar a capacidade do novo método de despachar as mensagens, para isto foi construído o ambiente da figura 26. As marcas em azul são pontos em que os robôs conseguem abrir as portas e os pontos em vermelho para fechá-las, sendo “WP1” e o “WP2” para abrir e fechar a porta “A” e “WP3” e “WP4” para abrir e fechar a porta “B”.

A geração dos caminhos para a navegação é realizada a partir da decomposição em células pelo método aproximado, dividindo o ambiente em regiões igualmente espaçadas. Um mapa de bits é utilizado para fazer a representação, sendo os espaços ocupados representados pelo digito ”1” e os espaços livres pelo digito ”0”. A Figura 26 mostra o mapa do lado esquerdo e como está a decomposição do lado direito.

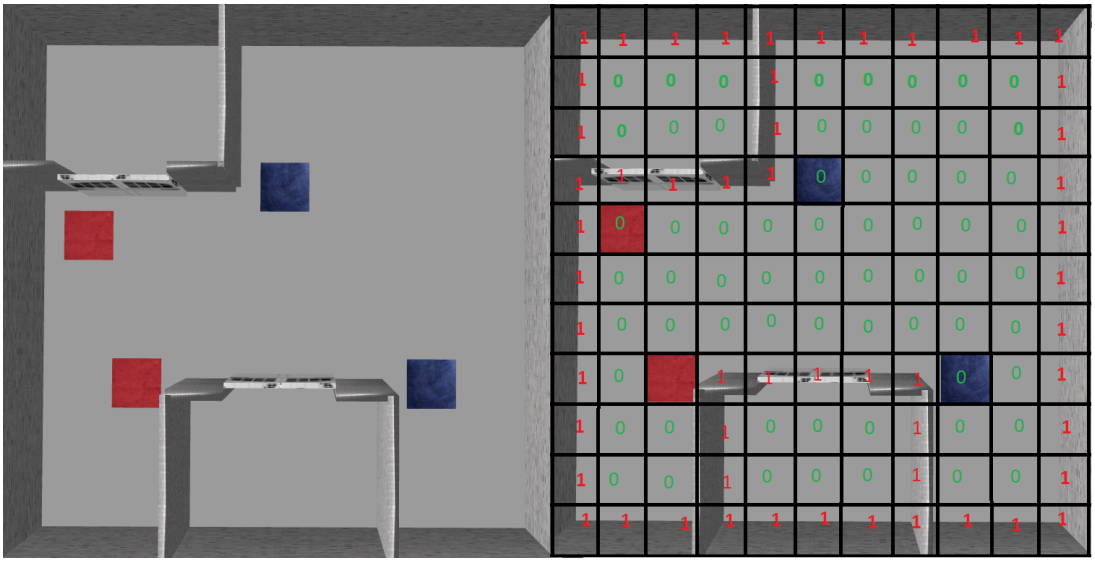

Figura 26 – Representação em mapa de bits

[1] https://www.youtube.com/watch?v=cVRndd8PeQI

Foram utilizados dois robôs, que precisam cooperar para concluir o plano. Os robôs podem se locomover entre os pontos marcados (wp), abrir as portas "A" e "B", quando estão nos pontos "WP3" e "WP2", e fechar as portas quando estão nos pontos "WP6" e "WP7". As pré-condições e os efeitos da execução de cada uma das ações estão representadas no algoritmo 6.2.

Algoritmo 6.2 – Domínio utilizado nas simulações

```
(define (domain turtlebot_demo)
(:requirements :strips :typing :fluents
               :disjunctive-preconditions :durative-actions)
(:types
        waypoint
        robot
        door)
(:predicates
        (robot_at ?v - robot ?wp - waypoint)
        (connected ?from ?to - waypoint)
        (unlock ?d - door ?w ?from ?to - waypoint)
        (lock ?d - door ?w ?from ?to - waypoint)
        (open ?d - door)
        (close ?d - door)
        (visited ?wp - waypoint))

(:functions
        (distance ?from ?to - waypoint))

(:durative-action goto_waypoint
        :parameters (?v - robot ?from ?to - waypoint)
        :duration ( = ?duration 10)
        :condition (and
                (at start (robot_at ?v ?from))
                (over all (connected ?from ?to)))
        :effect (and
                (at start (not (robot_at ?v ?from)))
                (at end (robot_at ?v ?to))
                (at end (visited ?to))))
(:durative-action close_door
        :parameters(?v - robot ?d - door ?w ?from ?to - waypoint)
        :duration (= ?duration 5)
        :condition (and
                (at start(lock ?d ?w ?from ?to))
                (at start(robot_at ?v ?w))
```

```
36                  (at start(open ?d)))
37          :effect (and
38                  (at start (not(open ?d)))
39                  (at start(not(connected ?from ?to)))
40                  (at start(not(connected ?to ?from)))
41                  (at end(close ?d))))
42  (:durative-action open_door
43          :parameters(?v - robot ?d - door ?w ?from ?to - waypoint)
44          :duration (= ?duration 5)
45          :condition (and
46                  (at start(unlock ?d ?w ?from ?to))
47                  (at start(robot_at ?v ?w))
48                  (at start(close ?d)))
49          :effect (and
50                  (at start (not(close ?d)))
51                  (at end(connected ?from ?to))
52                  (at end(connected ?to ?from))
53                  (at end(open ?d)))))
```

O plano gerado pelo planejador POPF, considera o estado inicial evidenciado na Figura 28, com o objetivo de ter o robô2 no ponto WP10 e a porta A fechada. A Tabela 4 ilustra o plano gerado para a simulação 1.

Tabela 4 – Plano gerado para a simulação 1

Início da Ação (s)	**Ação a ser Executada**	**Duração (s)**
0.000	goto_waypoint robot1 wp1 wp3	[10.000]
0.000	goto_waypoint robot2 wp0 wp5	[10.000]
10.001	open_door robot1 b wp3 wp5 wp8	[5.000]
10.002	goto_waypoint robot1 wp3 wp6	[10.000]
15.002	goto_waypoint robot2 wp5 wp8	[10.000]
25.002	close_door robot1 b wp6 wp5 wp8	[5.000]
25.003	goto_waypoint robot2 wp8 wp10	[10.000]

A forma que o algoritmo despacha as mensagens está representada na Figura 27. Inicialmente, é enviada a mensagem 1, em seguida é feita a verificação da mensagem 2. Como todos os requisitos para despachar a mensagem foram atendidos, ela também é enviada. Neste momento, o robo1 e o robo2 realizam suas tarefas 1 e 2, respectivamente, de forma simultânea. Todas as outras mensagens são verificadas assim como explicado na Figura 15, seção 6.1. As setas em preto da Figura 27 indicam a dependência entre as mensagens, ou seja, as mensagens em que as setas estão saindo devem ser concluídas antes que seja enviada a mensagem em que está sendo apontada. Por exemplo, a mensagem 4 e

3 só podem ser enviadas depois que a mensagem 1 for concluída e a mensagem 5 só pode ser enviada depois que as mensagens 2 e 3 forem concluídas.

A conexão em vermelho mostra um efeito da tarefa 6 que cancela uma pré-condição da tarefa 5. Deste modo, a mensagem 6 só pode ser enviada depois que a tarefa 5 for concluída. Isto é esperado, pois mesmo que todas as pré-condições necessárias para fechar a porta seja atendida, faz-se necessário que o "robô1" espere o "robô2" atravessar a porta primeiro, já que fechar a porta cancela a pré-condição de estar conectado os dois pontos em que o robô2 está atravessando.

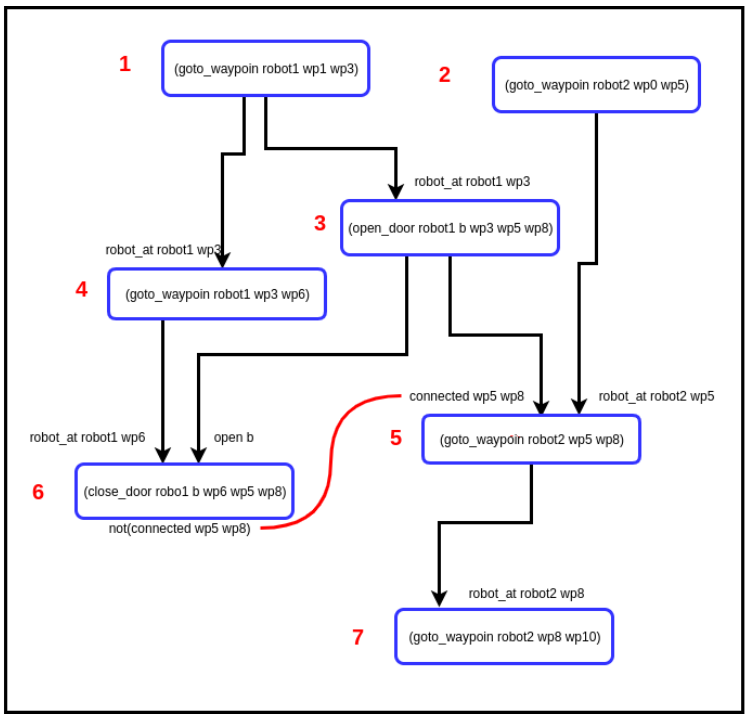

Figura 27 – Sequencia de despacho das mensagens

Com relação ao planejamento de rota, ocorre uma prevenção de colisão. Quando as mensagens 1 e 2 são enviadas, o sistema requisita uma rota para o robô 1 do ponto "WP1" para o "WP3", logo em seguida uma rota para o robo2 do "WP0" para o "WP5". O caminho mais curto para o robô1 está representado em vermelho na Figura 28 e o caminho mais curto para o robô2 está representado em azul. Seguindo estes caminhos, existe uma previsão de colisão no ponto em amarelo. Sendo assim, o sistema mantêm o caminho do robô1 por ele ter solicitado primeiro e cria o caminho em verde para o robô2. De acordo com a execução do plano, os caminhos para executar as ações 4, 5 e 7 são solicitados e devidamente gerados sem que ocorra nenhuma colisão.

Na execução do exemplo observa-se um momento em que o robô1 executa duas ações ao mesmo tempo, abrir a porta e se locomover para um outro ponto. Tal comportamento é esperado, uma vez que a ação de abrir a porta requer que o robô esteja no ponto específico apenas no início da execução da tarefa, diferente do que acontece durante a

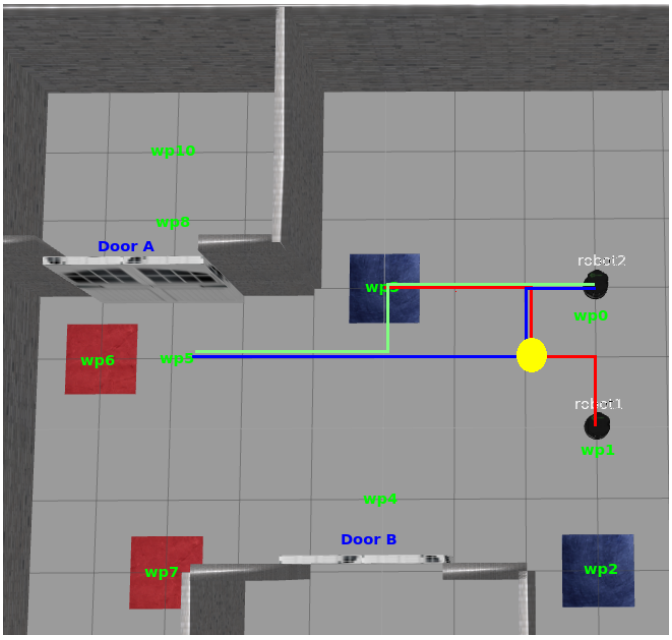

Figura 28 – Simulação do planejamento feito pelos robôs

execução das ações 6 e 5. Esta característica de poder executar mais de uma ação ao mesmo tempo está presente na taxonomia de Gerkey e Matarić (2004).

6.6.2 Simulação 2

A segunda simulação[1] tem a intenção de verificar a capacidade do pacote de navegação de evitar colisões em um ambiente com vários robôs cruzando o mapa. Para isto, foi criado um mapa com oito robôs e quatro corredores. O mapa e a posição inicial dos robôs estão representados na Figura 29. A numeração em cada robô representa o nome de cada robô e os objetivos são:

- Mover Robô_1 do wp0 (1,1) para o wp8 (9,8)
- Mover Robô_2 do wp1 (1,3) para o wp9 (9,6)
- Mover Robô_3 do wp2 (1,4) para o wp10 (9,4)
- Mover Robô_4 do wp3 (1,7) para o wp11 (9,2)
- Mover Robô_5 do wp4 (9,1) para o wp12 (1,6)
- Mover Robô_6 do wp5 (9,3) para o wp13 (1,8)
- Mover Robô_7 do wp6 (9,5) para o wp14 (1,4)
- Mover Robô_8 do wp7 (9,7) para o wp15 (1,2)

[1] https://youtu.be/taGBOF6tEIA

O plano PDDL gerado está representado na tabela 5. O robô que recebe a maior prioridade é aquele que faz a requisição do plano primeiro. Como a requisição é feita de acordo com o plano, o robô de maior prioridade será o Robô3 e o de menor prioridade será o robô8.

Tabela 5 – Plano gerado para os 8 robôs

Início da Ação (s)	**Ação a ser Executada**	**Duração (s)**
0.000	goto_waypoint robot3 wp2 wp10	[10.000]
0.000	goto_waypoint robot4 wp3 wp11	[10.000]
0.000	goto_waypoint robot1 wp0 wp8	[10.000]
0.000	goto_waypoint robot5 wp4 wp12	[10.000]
0.000	goto_waypoint robot7 wp6 wp14	[10.000]
0.000	goto_waypoint robot6 wp5 wp13	[10.000]
0.000	goto_waypoint robot2 wp2 wp9	[10.000]
0.000	goto_waypoint robot8 wp7 wp15	[10.000]

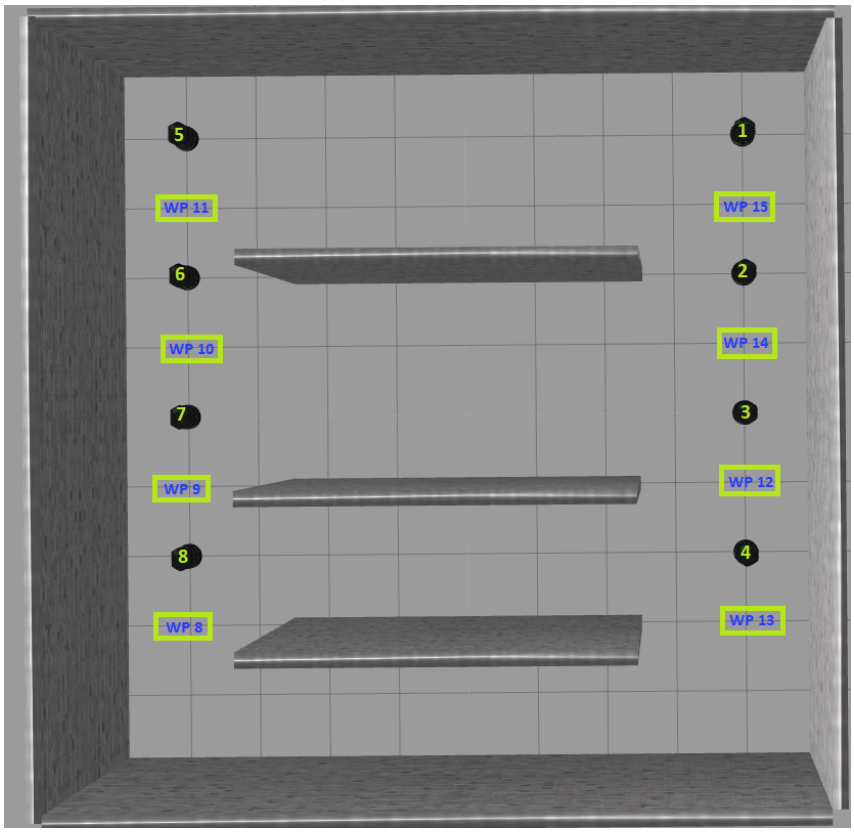

Figura 29 – Estado inicial da Simulação 2

Assim, como esperado, os caminhos são gerados para todos os robôs, evitando as colisões já na fase de planejamento. Mesmo sem nenhum sensor que detecte a presença dos outros robôs, foi possível que todos os robôs completassem seus objetivos e chegassem nos destinos finais. O plano gerado para cada robô está representado a seguir, sendo que o tempo (T) é dado em segundos.

Robô 3		
T	**Ação**	**Posição**
0	Manter	(1,5)
1	Virar	(1,5)
2	Seguir	(1,4)
3	Virar	(1,4)
4	Seguir	(2,4)
5	Seguir	(3,4)
6	Seguir	(4,4)
7	Seguir	(5,4)
8	Seguir	(6,4)
9	Seguir	(7,4)
10	Seguir	(8,4)
11	Seguir	(9,4)

Robô 5		
T	**Ação**	**Posição**
0	Manter	(9,1)
1	Virar	(9,1)
2	Virar	(9,1)
3	Seguir	(7,1)
4	Seguir	(5,1)
5	Seguir	(4,1)
6	Virar	(4,1)
7	Seguir	(4,2)
8	Virar	(4,2)
9	Seguir	(3,2)
10	Seguir	(2,2)
11	Virar	(2,2)
13	Seguir	(2,3)
14	Seguir	(2,4)
15	Seguir	(2,5)
16	Seguir	(2,6)
17	Virar	(2,6)
18	Seguir	(1,6)

Robô 4		
T	**Ação**	**Posição**
0	Manter	(1,7)
1	Virar	(1,7)
2	Seguir	(1,6)
3	Seguir	(1,5)
4	Virar	(1,5)
5	Seguir	(2,5)
6	Seguir	(3,5)
7	Virar	(3,5)
8	Seguir	(3,4)
9	Virar	(3,4)
10	Seguir	(4,4)
11	Seguir	(5,4)
13	Seguir	(6,4)
14	Seguir	(7,4)
15	Seguir	(8,4)
16	Virar	(8,4)
17	Seguir	(8,3)
18	Seguir	(8,2)
19	Virar	(8,2)
20	Seguir	(9,2)

Robô 7		
T	**Ação**	**Posição**
0	Manter	(9,5)
1	Virar	(9,5)
2	Virar	(9,5)
3	Virar	(8,5)
4	Seguir	(8,3)
5	Seguir	(8,2)
6	Virar	(8,2)
7	Seguir	(7,2)
8	Seguir	(6,2)
9	Seguir	(5,2)
10	Seguir	(4,2)
11	Seguir	(3,2)
13	Seguir	(2,2)
14	Seguir	(1,2)
15	Virar	(1,2)
16	Seguir	(1,3)
17	Seguir	(1,4)

Robô 1		
T	**Ação**	**Posição**
0	Manter	(1,1)
1	Seguir	(2,1)
2	Virar	(2,1)
3	Seguir	(2,3)
4	Seguir	(2,5)
5	Virar	(2,5)
6	Seguir	(3,5)
7	Seguir	(4,5)
8	Seguir	(5,5)
9	Seguir	(6,5)
10	Seguir	(7,5)
11	Seguir	(8,5)
13	Virar	(8,5)
14	Seguir	(8,6)
15	Seguir	(8,7)
16	Seguir	(8,8)
17	Virar	(8,8)
18	Seguir	(9,8)

Robô 6		
T	**Ação**	**Posição**
0	Manter	(9,3)
1	Virar	(9,3)
2	Virar	(9,3)
3	Seguir	(8,3)
4	Virar	(8,3)
5	Seguir	(8,2)
6	Virar	(8,2)
7	Seguir	(7,2)
8	Seguir	(6,2)
9	Seguir	(5,2)
10	Virar	(5,2)
11	Seguir	(5,1)
12	Virar	(5,1)
13	Seguir	(4,1)
14	Seguir	(3,1)
15	Seguir	(2,1)
16	Virar	(2,1)
17	Seguir	(2,2)
18	Seguir	(2,3)
19	Seguir	(2,4)
20	Seguir	(2,5)
21	Seguir	(2,6)
22	Seguir	(2,7)
23	Virar	(2,7)
24	Seguir	(1,7)
25	Virar	(1,7)
26	Seguir	(1,8)

Robô 2		
T	**Ação**	**Posição**
0	Manter	(1,3)
1	Manter	(1,3)
2	Manter	(1,3)
3	Manter	(1,3)
4	Manter	(1,3)
5	Manter	(1,3)
6	Manter	(1,3)
7	Virar	(1,3)
8	Seguir	(1,2)
9	Seguir	(1,1)
10	Virar	(1,1)
11	Seguir	(2,1)
12	Seguir	(3,1)
13	Virar	(3,1)
14	Seguir	(3,2)
15	Virar	(3,2)
16	Seguir	(4,2)
17	Seguir	(5,2)
18	Seguir	(6,2)
19	Seguir	(7,2)
20	Seguir	(8,2)
21	Virar	(8,2)
22	Seguir	(8,3)
23	Seguir	(8,4)
24	Seguir	(8,5)
25	Virar	(8,5)
26	Seguir	(9,5)
27	Virar	(9,5)
28	Seguir	(9,6)

Robô 8		
T	**Ação**	**Posição**
0	Manter	(9,7)
1	Virar	(9,7)
2	Virar	(9,7)
3	Seguir	(8,7)
4	Virar	(8,7)
5	Seguir	(8,6)
6	Seguir	(8,5)
7	Seguir	(8,4)
8	Seguir	(8,3)
9	Seguir	(8,2)
10	Virar	(8,2)
11	Seguir	(7,2)
12	Seguir	(6,2)
13	Seguir	(5,2)
14	Seguir	(4,2)
15	Virar	(4,2)
16	Seguir	(4,1)
17	Virar	(4,1)
18	Seguir	(3,1)
19	Seguir	(2,1)
20	Virar	(2,1)
21	Seguir	(2,2)
22	Virar	(2,2)
23	Seguir	(1,2)

6.6.3 Simulação 3

O objetivo da simulação 3 é mostrar o funcionamento da requisição de planos para um robô quando outro já está realizando algum movimento no mapa. Nesta simulação, não é utilizado o planejamento de tarefa, pois o objetivo é verificar como o planejador de caminho gera os pontos que devem ser percorridos pelos robôs. Portanto, a requisição de plano foi feita por um algoritmo de finalidade especifica.

Para esta simulação, são utilizados 5 robôs numerados de 1 a 5 que percorreram o mesmo mapa da simulação 2. A simulação é feita da seguinte forma: primeiro é realizada a requisição do caminhos para o robô 1 sair do ponto (1,1) e mover para o ponto (8,3) e para o robô 2 sair do ponto (8,1) e ir para o ponto (3,5). Depois de 5 segundos, o robô 3 irá requerer um caminho do ponto (1,5) para (8,4), após mais 2 segundos, os robôs 4 e 5 irão requerer os caminhos dos pontos (5,7) e (9,2) para os pontos (2,1) e (2,2), respectivamente.

O plano gerado está representado na tabela abaixo, sendo x e y as coordenadas do robô e A a sua rotação dada em graus.

Tabela 6 – Caminhos gerados para a simulação 3

Tempo	**Robô 1**			**Robô 2**			**Robô 3**			**Robô 4**			**Robô 5**		
	x	**y**	**A**	**x**	**y**	**A**	**x**	**y**	**A**	**x**	**y**	**A**	**x**	**y**	**A**
0	1	1	180	8	1	180									
1	2	1	180	8	1	90									
2	3	1	180	8	1	0									
3	4	1	180	7	1	0									
4	4	1	180	6	1	0									
5	4	2	180	5	1	0	1	5	180						
6	4	2	180	4	1	0	1	5	270						
7	5	2	180	3	1	0	1	4	270	5	7	180	9	2	180
8	6	2	180	3	1	270	1	4	180	5	7	90	9	2	90
9	7	2	180	3	2	270	2	4	180	5	7	0	9	1	90
10	8	2	180	3	2	0	3	4	180	4	7	0	9	1	0
11	8	2	270	2	2	0	4	4	180	3	7	0	8	1	0
12	8	3	270	2	2	270	5	4	180	2	7	0	7	1	0
13				2	3	270	6	4	180	2	7	90	6	1	0
14				2	4	270	7	4	180	2	6	90	5	1	0
15				2	4	270	8	4	180	2	5	90	4	1	0
16				3	4	270				2	4	90	3	1	0
17				3	4	270				2	3	90	3	1	270
18				3	5	270				2	2	90	3	2	270
19										2	1	90	3	2	270
20													2	2	270

A geração dos pontos ocorrem de acordo com o previsto, seguindo a prioridade dos robôs que fazem a requisição primeiro. Desta forma, quando o robô 3 faz a requisição de um caminho, é dado a ele o tempo que está na base do vetor que guarda todos os caminhos planejados, assim como explicado na seção 6.3. Caso todos os robôs já tenham terminado de executar seus planos, não haverá nenhum elemento no vetor de caminhos e, portanto, o próximo robô que fizer a requisição receberá o tempo 0.

6.6.4 Simulação 4

A quarta simulação é feita para comparar o método de despacho do ROSPlan e o proposto na seção 6.1. Para realizar esta simulação, foi utilizado o domínio "ROVER". Este domínio possui 5 ações: mover, pegar uma amostra, deixar uma amostra, tirar foto e recarregar. As pré-condições e os efeitos de cada ação podem ser observados no domínio do Apêndice A.1. No arquivo problema são definidos 3 robôs, 12 pontos que podem ser visitados (wp1...wp12), 9 amostras e 4 possíveis objetivos. Os robôs podem recarregar

a bateria no wp1 e deixar as amostras no wp7. O arquivo problema completo pode ser observado no ApêndiceA.2.

O plano gerado para esse problema está descrito na Tabela a seguir:

Tabela 7 – Plano gerado a partir do domínio “ROVER”

Tempo (s)	**Ação**	**Duração (s)**
0.000:	move rover1 waypoint6 waypoint7	[5.000]
0.000:	move rover2 waypoint6 waypoint3	[5.000]
0.000:	move rover3 waypoint6 waypoint8	[5.000]
5.001:	take_image rover1 objective2 waypoint7	[7.000]
5.001:	take_image rover3 objective3 waypoint8	[7.000]
5.001:	take_image rover2 objective1 waypoint3	[7.000]
5.002:	take-sample rover2 sample2 waypoint3	[8.000]
12.001:	move rover3 waypoint8 waypoint4	[5.000]
12.001:	move rover1 waypoint7 waypoint6	[5.000]
13.002:	move rover2 waypoint3 waypoint6	[5.000]
17.002:	move rover3 waypoint4 waypoint9	[5.000]
17.002:	move rover1 waypoint6 waypoint3	[5.000]
18.003:	move rover2 waypoint6 waypoint7	[5.000]
22.003:	move rover3 waypoint9 waypoint1	[5.000]
22.003:	move rover1 waypoint3 waypoint4	[5.000]
23.004:	drop-sample rover2 sample2 waypoint7	[1.000]
24.004:	move rover2 waypoint7 waypoint6	[5.000]
27.004:	take_image rover3 objective4 waypoint1	[7.000]
27.004:	move rover1 waypoint4 waypoint9	[5.000]
29.005:	move rover2 waypoint6 waypoint3	[5.000]
32.005:	move rover1 waypoint9 waypoint1	[5.000]
34.004:	move rover3 waypoint1 waypoint5	[5.000]
37.006:	recharge rover1 waypoint1	[7.900]
44.907:	move rover1 waypoint1 waypoint5	[5.000]
49.908:	move rover1 waypoint5 waypoint2	[5.000]
54.909:	take-sample rover1 sample1 waypoint2	[8.000]
62.909:	move rover1 waypoint2 waypoint5	[5.000]
67.910:	move rover1 waypoint5 waypoint1	[5.000]
72.911:	move rover1 waypoint1 waypoint9	[5.000]
77.912:	move rover1 waypoint9 waypoint4	[5.000]
82.913:	move rover1 waypoint4 waypoint3	[5.000]
87.914:	move rover1 waypoint3 waypoint6	[5.000]
92.915:	move rover1 waypoint6 waypoint7	[5.000]
97.916:	drop-sample rover1 sample1 waypoint7	[1.000]

Quando utiliza-se na simulação exatamente o tempo planejado para a execução da ação, tanto o método de despacho “dispatch-concurrent && !dispatch-on-completion”, como o desenvolvido, possuem o mesmo desempenho. Entretanto, quando utilizado um tempo variável, o método desenvolvido apresenta um desempenho melhor. No caso do

tempo de execução ser menor que o planejado, muitas vezes o algoritmo consegue adiantar a execução do plano, e quando o tempo de execução é maior, o algoritmo continua a executar o plano até o tempo máximo que for definido. Só depois disso ocorre o replanejamento. Já o método fornecido pelo ROSPlan, quando há qualquer diferença no tempo de execução, há um retorno de falha e tem que replanejar, apresentando o mesmo plano, mas resetando o tempo de execução daquela ação. Mesmo que agora a ação seja concluída com apenas poucos segundos, o algoritmo espera terminar todo o tempo planejado antes de enviar a próxima ação.

Com esse experimento, conclui-se que o algoritmo desenvolvido obtém desempenho melhor que o ROSPlan em quase todas as situações, sendo que, apenas quando as tarefas são executadas exatamente no tempo planejado é que o desempenho dos dois métodos são iguais.

7 Conclusão e Trabalhos Futuros

7.1 Conclusão

O pacote ROSPlan é uma ótima ferramenta para o planejamento e execução de tarefas na robótica, falhando apenas em aplicações com múltiplos robôs. A partir do novo método de despacho de mensagem desenvolvido e das adaptações feitas no código para o reconhecimento e correta execução das tarefas alocadas, o pacote se torna capaz de executar planos que apresentem mais de um robô, de forma mais robusta ao replanejamento, e consegue executar o plano no pior caso, no tempo em que as ações foram planejadas.

No entanto, dependendo da aplicação, ainda pode ser interessante utilizar os outros métodos de despacho de mensagem, como em um ambiente que aconteça algum evento programado para um determinado tempo e que este seja importante para a conclusão do plano. Neste contexto, o método "dispatch-concurrent && !dispatch-on-completion", em que os tempos de planejamento é rigorosamente seguido, é mais apropriado. Em qualquer outra aplicação, o método desenvolvido tem o desempenho igual ou superior.

Na simulação 1, verifica-se que o método de despacho desenvolvido atende as expectativas, uma vez que ele despachava as mensagens sempre que possível, desde que não invalide o plano. Já na simulação 2, é possível verificar que o sistema de navegação consegue levar um robô com poucos sensores de um ponto a outro, evitando colisões, desde que os planos sejam pequenos, visto que o planejamento do caminho é feito considerando que o robô demora um tempo específico para percorrer uma unidade de medida para frente, permanecer parado ou virar. E, assim, como dito anteriormente, em aplicações reais, o tempo de execução dificilmente será igual ao planejado. Portanto, ocorre um acúmulo de erro que pode acabar levando o sistema para uma colisão.

Na terceira simulação, é verificado a geração dos caminhos quando já existem robôs executando as tarefas. Por fim, na simulação 4, é feito um comparativo com o método mais indicado para multirrobôs que o ROSPlan apresenta, sendo que o método desenvolvido apresenta um melhor desempenho em todas as simulações.

Salienta-se que parte dos resultados deste trabalho foi publicado no LARS (*Latin American Robotics Symposium*) em um artigo com o título: "A ROSPlan-based multi-robot navigation system"(MIRANDA; SOUZA; BASTOS, 2018) que segue em anexo.

7.2 Trabalhos Futuros

A dissertação teve como foco apenas o planejamento. Sendo assim, o sistema de navegação ainda precisa de muitas melhorias na percepção, controle e localização. Com relação a trabalhos futuros, é interessante aperfeiçoar os outros pontos em que o planejamento da trajetória inicial seria utilizado apenas para guiar inicialmente o robô e que durante a execução ele utilize outros sensores para fazer pequenas adaptações no plano e corrigir os erros acumulados.

Além disso, a forma em que o planejamento do caminho é feito também pode ser melhorada, uma vez que ainda não é obtido o caminho com menor custo, apenas procura-se um caminho livre de colisão para chegar ao objetivo final.

Por fim, conclui-se que seria interessante utilizar domínios mais complexos que apresente uma maior variedade de ações; utilizar robôs com diferentes características e capacidades; bem como, a partir do desenvolvimento do sistema de navegação mais robusto, realizar a verificação deste sistema em aplicações reais.

Referências

AMATO, N. M.; WU, Y. A randomized roadmap method for path and manipulation planning. In: IEEE. *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. [S.l.], 1996. v. 1, p. 113–120. 38

BERG, J. P. V. D.; OVERMARS, M. H. Prioritized motion planning for multiple robots. In: IEEE. *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. [S.l.], 2005. p. 430–435. 39, 40

BERG, J. P. V. D.; OVERMARS, M. H. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, IEEE, v. 21, n. 5, p. 885–897, 2005. 38

BLUM, A. L.; FURST, M. L. Fast planning through planning graph analysis. *Artificial intelligence*, Elsevier, v. 90, n. 1, p. 281–300, 1997. 28

BURGARD, W. et al. Coordinated multi-robot exploration. *IEEE Transactions on robotics*, IEEE, v. 21, n. 3, p. 376–386, 2005. 20

CANTONI, L. F. A. *Avaliação do uso da linguagem PDDL no planejamento de missões para robôs aéreos*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DE MINAS GERAIS, 2010. 34

CÁP, M. et al. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering*, IEEE, v. 12, n. 3, p. 835–849, 2015. 39, 40

CASHMORE, M. et al. Auv mission control via temporal planning. In: *IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2014. p. 6535–6541. 34

CASHMORE, M. et al. Rosplan: Planning in the robot operating system. In: *ICAPS*. [S.l.: s.n.], 2015. p. 333–341. 9, 15, 44, 45

CHEN, I. et al. A simulation environment for openrtm-aist. In: IEEE. *System Integration, 2009. SII 2009. IEEE/SICE International Symposium on*. [S.l.], 2009. p. 113–117. 65

CHOUHAN, S. S.; SINGH, A.; NIYOGI, R. Multi-agent planning with joint actions. In: IEEE. *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. [S.l.], 2015. p. 1284–1290. 20, 35

COLES, A. J. et al. Forward-chaining partial-order planning. In: *Twentieth International Conference on Automated Planning and Scheduling (ICAPS 10)*. [S.l.]: AAAI Press, 2010. 53

COPPIN, B. Inteligência artificial. *Rio de Janeiro: LTC*, 2013. 27, 28, 29, 30

CROSBY, M.; JONSSON, A.; ROVATSOS, M. A single-agent approach to multiagent planning. In: IOS PRESS. *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. [S.l.], 2014. p. 237–242. 19

CROSBY, M.; PETRICK, R. P. Temporal multiagent planning with concurrent action constraints. In: *ICAPS workshop on distributed and multi-agent planning (DMAP)*. [S.l.: s.n.], 2014. 35

ERDMANN, M.; LOZANO-PEREZ, T. On multiple moving objects. *Algorithmica*, Springer, v. 2, n. 1-4, p. 477, 1987. 39

FAHIMI, F.; NATARAJ, C.; ASHRAFIUON, H. Real-time obstacle avoidance for multiple mobile robots. *Robotica*, Cambridge University Press, v. 27, n. 2, p. 189–198, 2009. 39

FARIA, G. *Uma Arquitetura de controle inteligente para múltiplos robôs*. Tese (Doutorado) — Universidade de São Paulo, 2006. 37, 38

FIKES, R. E.; NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, Elsevier, v. 2, n. 3-4, p. 189–208, 1971. 28

FOX, M.; LONG, D. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 2003. 33

GE, S. S.; CUI, Y. J. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, Springer, v. 13, n. 3, p. 207–222, 2002. 39

GERKEY, B. P.; MATARIĆ, M. J. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, SAGE Publications, v. 23, n. 9, p. 939–954, 2004. 19, 21, 22, 70

GHALLAB, M.; NAU, D.; TRAVERSO, P. *Automated Planning: theory and practice*. [S.l.]: Elsevier, 2004. 9, 24, 25

GUZZONI, D. et al. Many robots make short work: Report of the sri international mobile robot team. *AI magazine*, v. 18, n. 1, p. 55, 1997. 19

JULIO, R. E. Dbml: Uma biblioteca de gerenciamento dinâmico de banda para sistemas multirrobôs baseado em ROS. 2015. 41

KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE. *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. [S.l.], 2004. v. 3, p. 2149–2154. 65

LATOMBE, J.-C. *Robot motion planning*. [S.l.]: Springer Science & Business Media, 2012. v. 124. 18

LIANG, Y. S. et al. A framework for robot programming in cobotic environments: First user experiments. In: ACM. *Proceedings of the 3rd International Conference on Mechatronics and Robotics Engineering*. [S.l.], 2017. p. 30–35. 35

MCDERMOTT, D. et al. PDDL-the planning domain definition language. 1998. 30

MCDERMOTT, D. M. The 1998 ai planning systems competition. *AI magazine*, v. 21, n. 2, p. 35, 2000. 30

MEZENCIO, R. *Implementação do método de campos potenciais para navegação de robôs móveis baseada em computação reconfigurável*. Tese (Doutorado) — Universidade de São Paulo, 2002. 9, 37

MIRANDA, D. S. S.; SOUZA, L. E. de; BASTOS, G. S. A ROSPlan-based multi-robot navigation system. In: IEEE. *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. [S.l.], 2018. p. 248–253. 77

MORRIS, R. et al. Autonomous search-detect-track for small uavs. *IntEx 2017*, p. 19, 2017. 9, 47, 49

MUSCETTOLA, N.; MORRIS, P.; TSAMARDINOS, I. Reformulating temporal plans for efficient execution. In: *KR*. [S.l.: s.n.], 1998. p. 444–452. 51

NOCKS, L. *The robot: the life story of a technology*. [S.l.]: Greenwood Publishing Group, 2007. 14, 19

NOGUEIRA, V. V. E. *Planejamento de Missão para VANTs em Ambientes Estocásticos*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2018. 44

NORVIG, P.; RUSSELL, S. *Inteligência Artificial, 3a Edição*. [S.l.]: Elsevier Brasil, 2014. 18, 26, 29

OTTONI, G. d. L.; LAGES, W. F. Navegação de robôs móveis em ambientes desconhecidos utilizando sonares de ultra-som. *Sba: Controle e Automação Sociedade Brasileira de Automatica*, SciELO Brasil, v. 14, n. 4, p. 402–411, 2003. 9, 36, 37

PARKER, L. E. et al. Current state of the art in distributed autonomous mobile robotics. In: *DARS*. [S.l.: s.n.], 2000. p. 3–14. 19

PEDNAULT, E. P. D. Toward a mathematical theory of plan synthesis. Stanford University, 1987. 30

QUIGLEY, M. et al. ROS: an open-source robot operating system. In: KOBE. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, p. 5. 41, 42

RICH, E.; KNIGHT, K. *Inteligência Artificial, 2a Edição*. [S.l.]: McGraw-Hill, 1993. 20, 24

RODRÍGUEZ-LERA, F. J.; MARTÍN-RICO, F.; MATELIÁN-OLIVERA, V. Generating symbolic representation from sensor data: Inferring knowledge in robotics competitions. In: IEEE. *Autonomous Robot Systems and Competitions (ICARSC), 2018 IEEE International Conference on*. [S.l.], 2018. p. 261–266. 9, 48, 50

ROS. *move_base*. Disponível em: <http://wiki.ros.org/move_base>. Acesso em: 22 março 2018. 57

ROS. *mrpt_navigation*. Disponível em: <http://wiki.ros.org/mrpt_navigation>. Acesso em: 22 março 2018. 57

ROS. *Wiki ROS*. 2007. Disponível em: <http://wiki.ros.org/>. Acesso em: 1 agosto 2014. 41, 44

SANELLI, V. et al. Short-term human robot interaction through conditional planning and execution. In: *Proc. of International Conference on Automated Planning and Scheduling (ICAPS)*. [S.l.: s.n.], 2017. 9, 35, 47

SANTOS, K. R. d. S. *SISTEMA DE NAVEGAÇÃO AUTÔNOMA PARA ROBÔS MÓVEIS BASEADO EM ARQUITETURA HÍBRIDA: TEORIA E APLICAÇÃO*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DE ITAJUBÁ, 2009. 34

SIEGWART, R. et al. *Introduction to autonomous mobile robots*. [S.l.]: MIT press, 2011. 14

SIMÉON, T.; LEROY, S.; LAUUMOND, J.-P. Path coordination for multiple mobile robots: A resolution-complete algorithm. *IEEE Transactions on Robotics and Automation*, IEEE, v. 18, n. 1, p. 42–49, 2002. 39, 40

STONE, P.; VELOSO, M. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, Springer, v. 8, n. 3, p. 345–383, 2000. 19

SVESTKA, P.; OVERMARS, M. H. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In: IEEE. *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. [S.l.], 1995. v. 2, p. 1631–1636. 38

WARREN, C. W. Multiple robot path coordination using artificial potential fields. In: IEEE. *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*. [S.l.], 1990. p. 500–505. 40

WURM, K. M. *Techniques for Multi-Robot Coordination and Navigation*. Tese (Doutorado) — Dissertation, Universität Freiburg, 2012, 2012. 20

Apêndices

APÊNDICE A – Representação do Problema ROVER em PDDL

A.1 Arquivo Domínio

```
54 (define (domain rover-domain)
55
56     (:requirements :durative-actions :fluents
57                    :duration-inequalities)
58
59     (:functions
60         (battery-amount ?rover)
61         (sample-amount ?rover)
62         (recharge-rate ?rover)
63         (battery-capacity)
64         (sample-capacity)
65     )
66
67     (:predicates
68             (can-move ?from-waypoint ?to-waypoint)
69         (is-visible ?objective ?waypoint)
70         (is-in ?sample ?waypoint)
71         (been-at ?rover ?waypoint)
72         (carry ?rover ?sample)
73         (at ?rover ?waypoint)
74         (is-recharging-dock ?waypoint)
75         (is-dropping-dock ?waypoint)
76         (taken-image ?objective)
77         (stored-sample ?sample)
78         (objective ?objective)
79         (waypoint ?waypoint)
80         (sample ?sample)
81         (rover ?rover)
82         )
83
84     (:durative-action move
85         :parameters
86             (?rover
```

```
87              ?from-waypoint
88              ?to-waypoint)
89
90          :duration
91              (= ?duration 5)
92
93          :condition
94                  (and
95                      (at start (rover ?rover))
96                      (at start (waypoint ?from-waypoint))
97                      (at start (waypoint ?to-waypoint))
98                      (over all (can-move ?from-waypoint ?to-
                            waypoint))
99                      (at start (at ?rover ?from-waypoint))
100                     (at start (> (battery-amount ?rover) 8)))
101
102         :effect
103                 (and
104                     (at end (at ?rover ?to-waypoint))
105                     (at end (been-at ?rover ?to-waypoint))
106                     (at start (not (at ?rover ?from-waypoint)))
107                     (at start (decrease (battery-amount ?rover)
                            8)))
108         )
109
110     (:durative-action take-sample
111         :parameters
112             (?rover
113              ?sample
114              ?waypoint)
115
116         :duration
117             (= ?duration 8)
118
119         :condition
120                 (and
121                     (at start (rover ?rover))
122                     (at start (waypoint ?waypoint))
123                     (over all (at ?rover ?waypoint))
124                     (at start (at ?rover ?waypoint))
125                     (at start (>= (battery-amount ?rover) 5))
126                     (at start (is-in ?sample ?waypoint))
```

```
127                     (at start (< (sample-amount ?rover) (sample-
                           capacity))))
128
129         :effect
130                 (and
131                     (at end (not (is-in ?sample ?waypoint)))
132                     (at start (decrease (battery-amount ?rover)
                           5))
133                     (at end (carry ?rover ?sample))
134                     (at end (increase (sample-amount ?rover) 1)))
135         )
136
137     (:durative-action drop-sample
138         :parameters
139             (?rover
140              ?sample
141              ?waypoint)
142
143         :duration
144             (= ?duration 1)
145
146         :condition
147             (and
148                 (at start (rover ?rover))
149                 (at start (sample ?sample))
150                 (at start (waypoint ?waypoint))
151                 (at start (is-dropping-dock ?waypoint))
152                 (over all (at ?rover ?waypoint))
153                 (at start (at ?rover ?waypoint))
154                 (at start (carry ?rover ?sample))
155                 (at start (> (battery-amount ?rover) 2)))
156
157         :effect
158                 (and
159                 (at end (is-in ?sample ?waypoint))
160                 (at end (not (carry ?rover ?sample)))
161                 (at end (stored-sample ?sample))
162                 (at start (decrease (battery-amount ?rover) 2))
163                 (at end (decrease (sample-amount ?rover) 1)))
164     )
165
166     (:durative-action take_image
```

```
167         :parameters
168             (?rover
169              ?objective
170              ?waypoint)
171
172         :duration
173             (= ?duration 7)
174
175         :condition
176             (and
177                 (at start (rover ?rover))
178                 (at start (objective ?objective))
179                 (at start (waypoint ?waypoint))
180                 (at start (at ?rover ?waypoint))
181                 (over all (at ?rover ?waypoint))
182                 (over all (is-visible ?objective ?waypoint))
183                 (at start (> (battery-amount ?rover) 1)))
184
185         :effect
186             (and
187                 (at end (taken-image ?objective))
188                 (at start (decrease (battery-amount ?rover) 1)))
189     )
190
191     (:durative-action recharge
192         :parameters
193             (?rover
194              ?waypoint)
195
196         :duration
197             (= ?duration
198                 (/ (- 80 (battery-amount ?rover)) (recharge-rate
                       ?rover)))
199
200         :condition
201                 (and
202                     (at start (rover ?rover))
203                     (at start (waypoint ?waypoint))
204                     (at start (at ?rover ?waypoint))
205                     (over all (at ?rover ?waypoint))
206                     (at start (is-recharging-dock ?waypoint))
207                     (at start (< (battery-amount ?rover) 80)))
```

```
208
209         :effect
210                 (at end
211                     (increase (battery-amount ?rover)
212                         (* ?duration (recharge-rate ?rover))))
213         )
214 )
```

A.2 Arquivo Problema

```
215 (define (problem rover-1)
216
217     (:domain
218         rover-domain
219     )
220
221     (:objects
222         waypoint1 waypoint2 waypoint3 waypoint4 waypoint5
               waypoint6
223         waypoint7 waypoint8 waypoint9 waypoint10 waypoint11
               waypoint12
224
225         sample1 sample2 sample3 sample4 sample5 sample6 sample7
               sample8
226         sample9
227
228         objective1 objective2 objective3 objective4
229
230         rover1 rover2 rover3
231     )
232
233     (:init
234         (= (battery-capacity) 100)
235         (= (sample-capacity) 2)
236
237         (waypoint waypoint1) (waypoint waypoint2) (waypoint
               waypoint3)
238         (waypoint waypoint4) (waypoint waypoint5) (waypoint
               waypoint6)
239         (waypoint waypoint7) (waypoint waypoint8) (waypoint
               waypoint9)
240
```

```
241         (sample sample1) (sample sample2) (sample sample3) (
               sample sample4)
242         (sample sample5) (sample sample6)
243
244         (objective objective1) (objective objective2) (objective
               objective3)
245         (objective objective4)
246
247         (can-move waypoint1 waypoint5) (can-move waypoint1
               waypoint9)
248         (can-move waypoint2 waypoint5) (can-move waypoint3
               waypoint4)
249         (can-move waypoint3 waypoint6) (can-move waypoint4
               waypoint3)
250         (can-move waypoint4 waypoint8) (can-move waypoint4
               waypoint9)
251         (can-move waypoint5 waypoint1) (can-move waypoint5
               waypoint2)
252         (can-move waypoint6 waypoint3) (can-move waypoint6
               waypoint7)
253         (can-move waypoint6 waypoint8) (can-move waypoint7
               waypoint6)
254         (can-move waypoint8 waypoint4) (can-move waypoint8
               waypoint6)
255         (can-move waypoint9 waypoint1) (can-move waypoint9
               waypoint4)
256
257         (is-visible objective1 waypoint2) (is-visible objective1
               waypoint3)
258         (is-visible objective1 waypoint4) (is-visible objective2
               waypoint5)
259         (is-visible objective2 waypoint7) (is-visible objective3
               waypoint8)
260         (is-visible objective4 waypoint5) (is-visible objective4
               waypoint1)
261
262
263         (is-in sample1 waypoint2) (is-in sample2 waypoint3)
264         (is-in sample3 waypoint9) (is-in sample4 waypoint8)
265         (is-in sample5 waypoint3) (is-in sample6 waypoint3)
266
267         (is-recharging-dock waypoint1)
```

```
268         (is-dropping-dock waypoint7)
269
270         (rover rover1)
271         (at rover1 waypoint6)
272         (= (battery-amount rover1) 50)
273         (= (recharge-rate rover1) 10)
274         (= (sample-amount rover1) 0)
275
276         (rover rover2)
277         (at rover2 waypoint6)
278         (= (battery-amount rover2) 50)
279         (= (recharge-rate rover2) 10)
280         (= (sample-amount rover2) 0)
281
282         (rover rover3)
283         (at rover3 waypoint6)
284         (= (battery-amount rover3) 50)
285         (= (recharge-rate rover3) 10)
286         (= (sample-amount rover3) 0)
287     )
288
289     (:goal
290         (and
291             (stored-sample sample1)
292             (stored-sample sample2)
293
294             (taken-image objective1)
295             (taken-image objective2)
296             (taken-image objective3)
297             (taken-image objective4))
298     )
299
300     (:metric
301         minimize (total-time)
302     )
303 )
```

Anexos

2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)

A ROSPlan-based multi-robot navigation system

Dannilo Samuel Silva Miranda, Luiz Edival de Souza, Guilherme Sousa Bastos
Department of Electrical Engineering, Federal University of Itajubá
Av. BPS, 1303, Bairro Pinheirinho - Itajubá MG, Brazil

***Abstract*—Multi-robot systems is a growing research field in robotics and artificial intelligence. Applying multiple robots on single robot systems, generally increase the execution robustness while minimizing its time. Besides this, if the execution requires joint or cooperative actions there isn't better option. This paper proposes a multi-robot version of RosPlan, which that is a framework for embedding a generic task planner in a ROS system. The main contributions include a new method to dispatch the actions; a system to ensure that each robot execute only its actions and a new ROS Package for multi-robot navigation.**

***Keywords*-RosPlan; PDDL; Planning; Multi-robot;**

I. Introduction

In recent years the number of multi-robot systems deployed in field applications has risen dramatically, replacing dangerous tasks that were once done only by humans, like fire-fighting [1], landmine detection [2] and underwater missions [3].

Control a robot is a complex task and requires a lot of skills, like determine which robots are better for the task and the sequence of high-level tasks that should be performed to accomplish the mission. In the case of multi-robot, this assignment becomes even more difficult perform by a human. So, providing the autonomy to the robots or designating a computer to do the planning and controlling the execution of the tasks is a better alternative.

Automatic planning is an area within the field of Artificial Intelligence that can be defined like the process of considering and organizing actions to achieve goals, before starting to execute them [3]. This tool is vital for a robotic agent operating in an environment where a chain of events could lead to a dead-end state. The planning community has standardized STRIPS-like planning with the introduction of the Planning Domain Definition Language (PDDL) [4]. In this language, a planning problem is formally described by providing two files: the *domain file* containing the actions with their preconditions and effects and the *problem file* containing the initial state and the goals.

Combining task planning and robotics present several challenges, most of these are addressed by the ROSPlan [5]. ROSPlan is a framework introduced to provide a generic method for planning and executing tasks in a ROS (Robot Operating System) system. It is a link between two standard approaches, PDDL and ROS. Basically, ROSPlan provides tools to generate the problem file from the knowledge parsed, automate calls to the planner, them post-process and validate the plan, handle the main dispatch loop and monitor the execution of the plan. ROSPlan also supports replanning in any case of failure.

The introduction of a multi-robot system can facilitate and increase the possibilities of executing a task, allowing plans that can not be executed by just one robot be executed by a set of robots with a common goal. Even sharing a common goal it is necessary to coordinate the dispatch of actions for each robot, and so that the actions are not carried out in a disorderly manner and invalidate the plan. The first contribution of this paper is a new method for dispatching actions in a coordinated way. Our approach allows to dispatch more than one action at time to several different robots, being robots able to perform their actions in a shared environment without interfere negatively in the actions of the others.

In addiction, for this work was develop a navigation package for multi-robot, because although already existent navigation packages in ROS and it works with multi-robot, none of them do multi-robot path planning.

II. Related Works

The task planning is essential in solving complex problems, a good planner have to choose and organize tasks to bring the system from the initial state to a goal. Many authors have already verified the effectiveness of task planning in applications with robots like [6] that use the planning to solve problems with aerial missions. In this work was evaluated the expressiveness of the PDDL language to describe a scenario with multi-robot in a temporal plan and it was pointed that this type of planning is able to execute complex time calculations and exploit the parallelism and competition of actions automatically.

In [7] is discussed the planning problems that can not be solved by single agents and needs the cooperation of several robots to achieve the goal. The experimental results of this work show that, multi-agent planning problem having a greater number of objects, agents, and location cannot be solved by existing planners. Since, these planners assumed agents as an object the state space drastically increases for them. Therefore, they fail after a certain level. Trying to work around this problem we propose a planning in two

978-1-5386-7761-2/18/$31.00 ©2018 IEEE
DOI 10.1109/LARS/SBR/WRE.2018.00053

stages, first a high level planning for the actions and then a path planning during the execution of the plan.

In this article, we chose work with the framework ROSPlan because this package proved effective in its proposal, and many researchers are adding their contributions in it, like the [8], that build a conditional planning for generating and executing short-term interactions by a robot deployed in a public environment; the [9], that describe a system for autonomously searching, detecting, and tracking an object of interest with a small unmanned aerial vehicle (sUAV); and the [3], that work with opportunistic planning problem, in theirs work are explores the execution of planned autonomous underwater vehicle (AUV) missions where opportunities to achieve additional utility can arise during execution.

A great advantage of ROSPlan is to have been built in the platform ROS [10] that is an architecture that provides libraries and tools to help software developers build robotic systems. ROS has become a popular platform for robotics research and one of its advantages it is licensed under an open source, and each year the number of contributions increases significantly.

III. Planning System

The task planning will work in two stages. In the first state a classical algorithm of planning will generate the height level action, such as indicate where the robot should go (end point) or what it should do. This planning happens just before it start executing the planning or when the plan was invalidated and a replannig was requested.

When it comes to an action of locomotion, the second state of planning happen, from the data provided by the classical planning, a new algorithm developed to a path plan is used. This algorithm is responsible for defining each point that the robot has to pass until it achieves its end point. For example, the Figure 1 shows that the robot_1 is in the point S1 and the robot_2 is in the point S2. The goal is the robot_1 is to arrive at G1 and robot_2 at G2. The classical planning will generate the task to the robot_1 leave from S1 and arrive in the spot G1 and the robot_2 leave from S2 and arrive in the spot G2. The path planning package shows all points that the robot must go through to reach the goal avoiding collisions.

IV. Dispatch Method

After the planner generates the list of actions that have to be executed is necessary to coordinate the dispatch of actions for each robot, so that all the actions are executed without the plan being invalidated and in this way, the final goal is achieved. ROSPlan offers some strategies to dispatch the plan, this strategies are based on the ROS launch file parameters, “dispatch-concurrent” and “dispatch-on-completion”. The behavior of dispatch depends on the combination of these values.

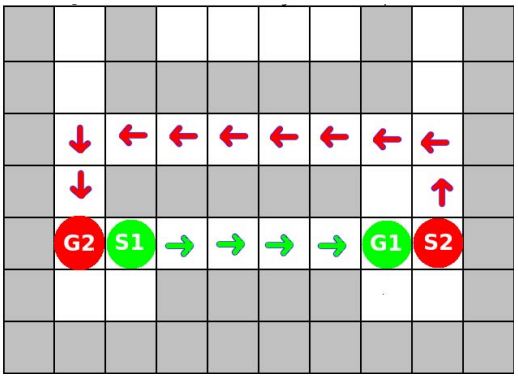

Figure 1. Example of path generated by the planner. The green arrow indicates the path of the robot_1 and the red arrow the path of the robot_2

- “dispatch_concurrent && dispatch_on_completion”: Actions are dispatched all at once;
- “!dispatch_concurrent && dispatch_on_completion”: Actions are dispatched as soon as the previous action is completed;
- “!dispatch_concurrent && !dispatch_on_completion”: Actions are dispatched at the time they are scheduled in the plan, or as soon as the previous actions completes;
- “dispatch_concurrent && !dispatch_on_completion”: Actions are dispatched at the time they are scheduled in the plan.

None of these options are desirable for a multi-robot system. The first option sends all the action at the beginning, without any checking, this situation is almost never wanted. The second option dispatch only one action at a time; in other words, the second action will be send only when the first one is finished. As result of that a robot cannot execute multi-action neither joint actions. The third option sends the action only when the time in which it was scheduled has already gone, however it dispatch the next action only if the previous has already finished. The forth and last one, the action can be sent, at the same time, to two robots. In this case the actions will be dispatched at the exact time they were scheduled. The problem with this option is that it is not very robust. If the execution time of any action be longer than the scheduled one, the plan will fail requiring then a replanning. As well, if the action ends before the planned time, the robot will not be able to advance to the next action, staying paused until the scheduled time for this action runs out.

The developed method consists in a supervisor make a list with actions, in a sequential form, then check if all the pre-conditions of the first action are satisfied. In the positive case, this action will be executed and added in another list that represents the execution action list. As soon as the first action is dispatched, the second action check begins. Primarily, is verified if comply with all the pre-condition. In

a positive case, the supervisor will ascertain that the effects of this action will not get away or negate the execution of the others actions that are in the execution list. If any of the effect actions interfere in which they are being executed, then this action is also dispatch. This procedure is made until the all messages are dispatch. The block diagram in the Figure 2 illustrates the operation.

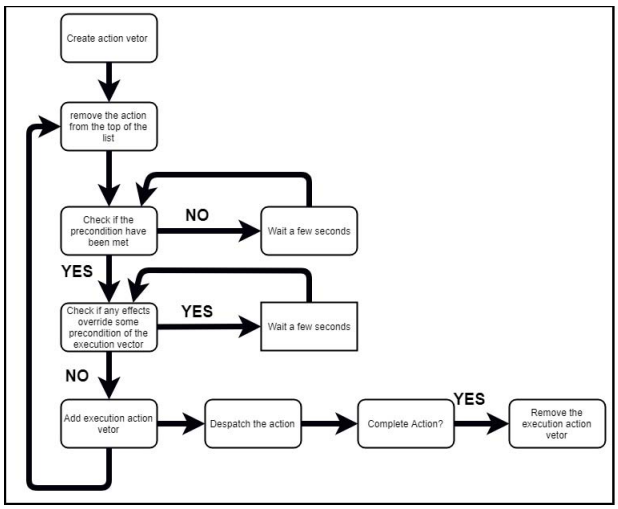

Figure 2. Block diagram of the dispatch method steps

For a better comprehension of the work and to exemplify, consider the actions with their preconditions and effects of the following domain:

```
(:durative-action goto_waypoint
:parameters (?v - robot ?from
             ?to - waypoint)
:duration ( = ?duration 10)
:condition (and
   (at start (robot_at ?v ?from))
   (over all (connected ?from ?to)))
:effect (and
   (at start (not (robot_at ?v ?from)))
   (at end (robot_at ?v ?to))
   (at end (visited ?to))))
(:durative-action close_door
:parameters(?v - robot ?d - door
            ?w ?from ?to - waypoint)
:duration (= ?duration 5)
:condition (and
   (at start(lock ?d ?w ?from ?to))
   (at start(robot_at ?v ?w))
   (at start(open ?d)))
:effect (and
   (at start (not(open ?d)))
   (at start(not(connected ?from ?to)))
   (at start(not(connected ?to ?from)))
   (at end(close ?d))))
(:durative-action open_door
:parameters(?v - robot ?d - door ?w ?from
            ?to - waypoint)
:duration (= ?duration 5)
:condition (and
   (at start(unlock ?d ?w ?from ?to))
   (at start(robot_at ?v ?w))
   (at start(close ?d)))
:effect (and
   (at start (not(close ?d)))
   (at end(connected ?from ?to))
   (at end(connected ?to ?from))
   (at end(open ?d))))
)
```

A possible PDDL plan for two robots that can be generated is:

- (goto_waypoint robot1 wp1 wp3)
- (goto_waypoint robot2 wp0 wp5)
- (open_door robot1 b wp3 wp5 wp8)
- (goto_waypoint robot1 wp3 wp6)
- (goto_waypoint robot2 wp5 wp8)
- (close_door robot1 b wp6 wp5 wp8)
- (goto_waypoint robot2 wp8 wp10)

In order to dispatch these actions, the supervisor checks the first message of the list, how all preconditions are attended and if does not have any action in the executed list. Then, the supervisor dispatches the action_1 and adds it in the execution list. The same is made with the next action of the planner list; in other words, the preconditions of the action_2 are attended, it starts to verify if any effects of the action_2 cancels some precondition of those action in the list that are executed. If there is not any limitation, action_2 is dispatched and added in the list of actions that are been executed. The dispatch of all the others actions is made in the same way.

When some action is concluded with success, the algorithm receives a message and take off this action from the execution list. Figure 3 represents all the actions previously listed with the precondition that need to be attended to dispatch the next action.

Notice that to complete the action_6 is only necessary that actions 4 and 3 have already been executed, however it cannot be dispatched while the action_5 has not concluded, since the effect (not(connected wp5 wp8)) of the action_6 cancels the precondition(connected wp5 wp8) of the action_5.

V. Task execution

To perform the tasks, each robot is entered in a different namespace. The namespace is used to avoid resource conflict and to promote a encapsulation, and this is extremely necessary in an environment with more than one robot. Whereas

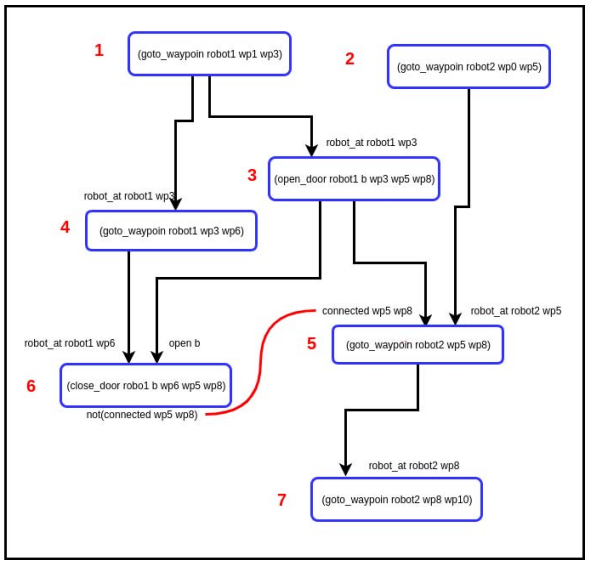

Figure 3. Flow diagram of the dispatch method

two robots have a topic for publishing its odometry with the name "\odom", if these two robots are integrate in the same system, the names conflicts of the topics is avoided by the namespace given to each robot.

Every action that the robot can execute was implemented in a separate node and added to the robot package. All these nodes should be subscribe the dispatch topic and have a mechanism to verify if the published action is intended for them. In other words, when any action is published in the dispatch topics, all subscribed nodes have to verify what is written in this action. First it is verified if it corresponds at robot, then verify which action is involved, and finally if this action is well formed, presenting all the necessary arguments. Having all these requirements met, the node will be activated and the task executed.

As soon as the robot completes the task execution, a conclusion massage should be send to the complete topic tasks. This massage will update the execution status to finished task, added the action effects in the knowledge base and will delete the vector actions in executions, as explained in the previous section.

VI. Navigation Package

The ability of a robot to move freely in an environment greatly increases the possibilities of tasks that it can perform. For this, an autonomous navigation system installed is needed, which is responsible to generate the robots trajectories.

At the ROS environment there are already some navigation packages (like move_base and mrpt_navigation) that are capable to move the robot by the map and deviates it from obstacles. However, any of them execute planning for multi-robots, in which one have a prediction of collision and that there is previously generated alternative plans to avoid them.

There are already several works in the literature that approaches the trajectory coordination of multiple mobile robots [11]–[14]. The differential of this work is the application of a technique of a multiple robot path planning in the navigation. In addition, new robots can be inserted in the environment at any moment being a plan created to avoid collisions with fixed obstacles and robots that are already executing some task.

The "classical prioritized plannin" [11] is a method widely used for this type of application. We choose this technique because of the simplicity of the algorithm that consists in assigning a different priority for each robot and generate the plan sequentially. The robot that has the highest priority has its plan generate first, and the one with lowest priority are generated in order to avoid collisions with the highest priority.

In order to avoid collisions with another robots, the concept of time-space is used, in which a robot occupies a certain region during a period of time $\tau(x, y, t)$. From a map ω provided by the planner, the starting points S_i and the arrival G_i of a robot. The algorithm must be able to return a plan π avoiding the trajectory Δ that the robots with higher priority will pass. The pseudo code of the Classical Prioritized Planning is represented in the Algorithm 1.

Algorithm 1 Classical Prioritized Planning

```
1: procedure ALGORITHM PP
2:     Δ ← ∅
3:     for i ← 1...n do
4:         π_i ← BEST_TRAJ(ω, Δ, S_i, G_i)
5:         if π_i = ∅ then
6:             report failure and terminate
7:         Δ ← Δ ∪ π_i
8: function BEST_TRAJ(ω, Δ, S_i, G_i)
9:     return optimal satisfying trajectory for robot i in ω that
10:    avoids regions Δ if it exists, otherwise return ∅
```

Some adaptations in the algorithm were made for this work, the planning of the trajectory of each robot is done in a central separated node, when some robots receive an action that demands a path in the map, it must take a request to the planner through a ROS SERVICE, which is defined by a pair of messages: one for the request and other one for the reply. The robot calls the service by sending the request message with its start and goal position waiting the reply. The planner will calculate the path avoiding the trajectories that have already been send, add in the vector of trajectory execution Δ and reply the path to the robot. With the planned path, the robot begins the navigation using the odometry to locate its pose. At each subpoint that the robot achieve, it publishes in a topic of sub_goal achieve that the path planner is subscriber, and the vector of trajectory Δ in execution is updated.

A simplified block diagram of all step of the planner is represented in Figure 4. The block 1 is the ROSPlan, where the task planning is done and the actions list to be executed is generated. The dispatch of actions is done as explained above. When a message is sent, all robots receive it, however only the one for which the message is destined will execute. If the message is to move in the map, the robot requests the path to the pathplan, represented by the block 3. As soon as it receives the response it starts to execute the task.

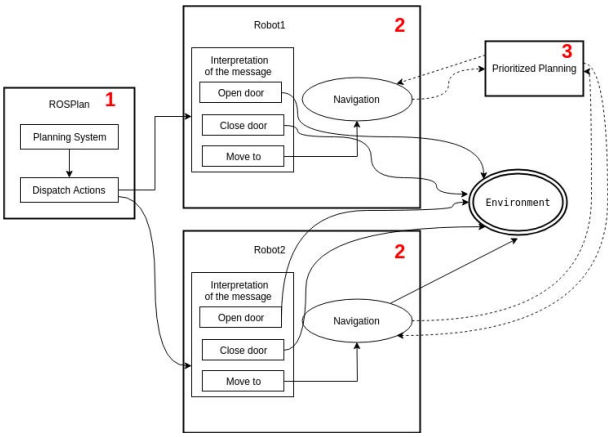

Figure 4. Block diagram of system steps

VII. Simulation

The environment presented in Figures 5 and Figures 7 were developed in GAZEBO [15]. Each robot was managed with the same namespace used in the task planner. This way facilitates the verification that the robots should do when a message arrives to them. A node with the interpretation of each action also have to be executed for each robot.

Here we can highlight two main examples, both using the previously domain shown in "Dispatch Method" section. The initial state of the first example is in the Figure 5 and the goal to be achieved is to have the "robot_2" in the wp10 and the "door b" closed.

As expected, the dispatch of the action happened like the figure 3, and in this example was possible observed the capacity of this method generate a plan that can have more them one robot execute actions in the same time and the capacity of one robot execute more them one action simultaneously, therefore what will limit how the action will be dispatch and if the robot will be able to execute multi action is the file domain.

Still in this example was possible observe a collision that was avoided. In the path of robot_1 from wp1 to wp3 it goes in the same path of robot_2 from wp0 to wp5. The first path generated to the robot_1 is represented by the red line in the Figure 6, the blue line represents the path of the

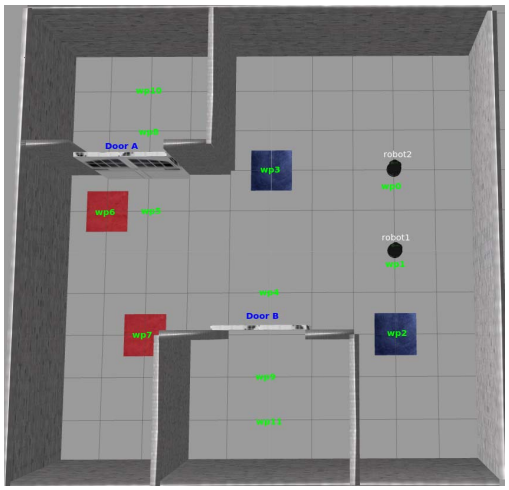

Figure 5. Initial states of the environment

robot_2. In the yellow point a collision would happen, but it was identified and the path represent by the green line was sent to robot_2 because it had lower priority. The simulation can be watched at the video[1].

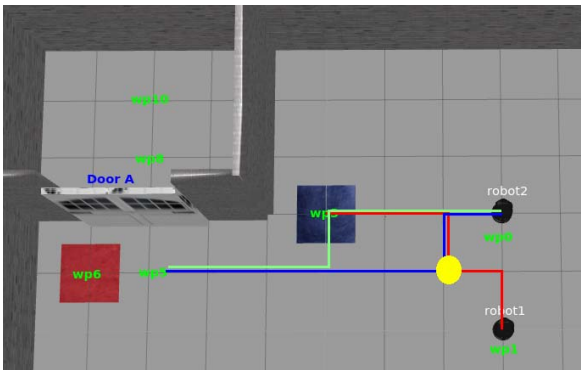

Figure 6. Path taken by robots and collision avoided

The second simulation has the intention to check the ability of the navigation package to avoid collisions in an environment with multiple robots crossing the map. For this, a map with seven robots and four corridors was created, the map and the initial position of the robots is represented in the figure 7. The numbering on each robot represents which robot it is and the order of priority that was used to find the ways, the circle with the numbers represents the destinations of them.

As expected, the paths was generated for all robots avoid the collisions already in the planning phase and even without any sensors that detect the presence of each robot. This way, was possible that all the robots arrived at their final destination.

[1]https://www.youtube.com/watch?v=cVRndd8PeQI

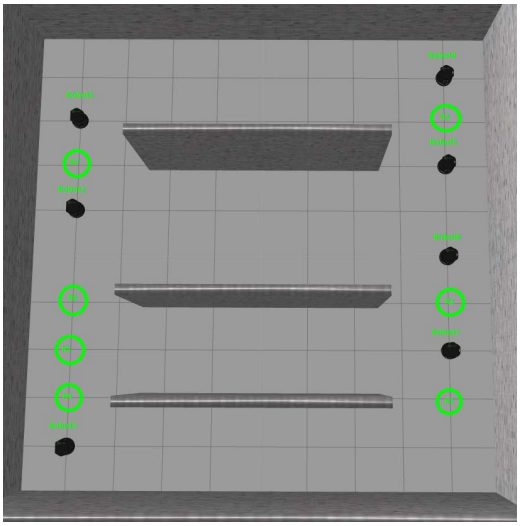

Figure 7. Initial and final state of the second example

VIII. Conclusion

This paper provided a tool to work with a multi-robot system using the ROSPlan package. The dispatch method described here worked well, not needing replanning every time an action delays and advancing the next actions of the plan when possible. The message verification system ensured that only the robots that the messages were destined performed the action. The navigation system was able to prevent collisions, as result, the robot did not need to arrive in the collision point to generate a new path.

There is still a lot of work to be done, particularly in navigation, since the Priority planning is not always able to find a valid plan, returning failure and needing a replanning. For future works we will be study extension of priority planning to decentralized system, a method to automatically create the navigation map, and use more sensors in localization and deviation of obstacles.

Acknowledgment

The authors of this paper would like to thank Capes for the financial support.

References

[1] J. Saez-Pons, L. Alboul, J. Penders, and L. Nomdedeu, "Multi-robot team formation control in the guardians project," *Industrial Robot: An International Journal*, vol. 37, no. 4, pp. 372–383, 2010.

[2] R. K. Megalingam, V. Gontu, R. Chanda, P. K. Yadav, and A. P. Kumar, "Landmine detection and reporting using light weight zumo bot," in *Inventive Computing and Informatics (ICICI), International Conference on*. IEEE, 2017, pp. 618–622.

[3] M. Cashmore, M. Fox, D. Long, D. Magazzeni, and B. Ridder, "Opportunistic planning in autonomous underwater missions," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 519–530, 2018.

[4] V. Estivill-Castro and J. Ferrer-Mestres, "Path-finding in dynamic environments with pddl-planners," in *Advanced Robotics (ICAR), 2013 16th International Conference on*. IEEE, 2013, pp. 1–7.

[5] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtós, and M. Carreras, "Rosplan: Planning in the robot operating system." in *ICAPS*, 2015, pp. 333–341.

[6] L. F. Cantoni, M. F. Campos, and L. Chaimowicz, "Investigação da linguagem pddl no planejamento de missões para robôs aéreos," *X SBAI–Simpósio Brasileiro de Automação Inteligente, São João del Rei, Minas Gerais. Brasil*, 2011.

[7] S. S. Chouhan, A. Singh, and R. Niyogi, "Multi-agent planning with joint actions," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2015, pp. 1284–1290.

[8] V. Sanelli, M. Cashmore, D. Magazzeni, and L. Iocchi, "Short-term human-robot interaction through conditional planning and execution," *International Conference on Automated Planning and Scheduling (ICAPS)*.

[9] R. Morris, A. Chakrabarty, J. Baculi, X. Bouyssounouse, and R. Hunt, "Autonomous search-detect-track for small uavs," *IntEx 2017*, p. 19, 2017.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[11] M. Čáp, P. Novák, A. Kleiner, and M. Selecký, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 835–849, 2015.

[12] D. Silver, "Cooperative pathfinding." *AIIDE*, vol. 1, pp. 117–122, 2005.

[13] P. Svestka and M. H. Overmars, "Coordinated motion planning for multiple car-like robots using probabilistic roadmaps," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 2. IEEE, 1995, pp. 1631–1636.

[14] C. W. Warren, "Multiple robot path coordination using artificial potential fields," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*. IEEE, 1990, pp. 500–505.

[15] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.