UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ENGENHARIA ELÉTRICA

CECILIA DE GOIS PARENTE

DESENVOLVIMENTO DO CONTROLE E COMUNICAÇÃO DE DOIS ROBÔS DENTRO DE UM DEPÓSITO LOGÍSTICO

FORTALEZA

2022

CECILIA DE GOIS PARENTE

DESENVOLVIMENTO DO CONTROLE E COMUNICAÇÃO DE DOIS ROBÔS DENTRO DE UM DEPÓSITO LOGÍSTICO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Engenharia Elétrica.

Orientador: Prof. Dr. Luiz Henrique Silva Colado Barreto

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P252d Parente, Cecilia de Gois.
Desenvolvimento do controle e comunicação de dois robôs dentro de um depósito logístico / Cecilia de Gois Parente. – 2022.
97 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2022.
Orientação: Prof. Dr. Luiz Henrique Silva Colado Barreto.

1. Armazém logístico. 2. Automação. 3. Robôs. 4. Cadeia de Suprimentos. I. Título.
CDD 621.3

CECILIA DE GOIS PARENTE

DESENVOLVIMENTO DO CONTROLE E COMUNICAÇÃO DE DOIS ROBÔS DENTRO DE UM DEPÓSITO LOGÍSTICO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Luiz Henrique Silva Colado Barreto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Bismark Claure Torrico
Universidade Federal do Ceará (UFC)

Prof. Dr. Dalton de Araújo Honório
Universidade Federal do Ceará (UFC)

A todos que estudaram, riram e trabalharam nessa jornada comigo.

AGRADECIMENTOS

Este trabalho não seria possível sem o apoio incansável da minha família e dos meus amigos. Não só para a execução deste, como também ao longo dos não triviais anos de graduação. Usufruo, portanto, deste espaço para expor minha gratidão e enaltecê-los como posso.

À minha mãe, que entre os percalços da vida e a distância, se fez tão compreensiva e presente, mesmo se tratando de um oficio tão longe do seu e mesmo com minhas ambições tão diferentes das suas. À minha irmã, que nunca hesitou em ter confiança em mim, te agradeço e jamais deixarei de abusar do seu cuidado e carinho.

Aos meus amigos da UFC, que nunca deixaram nenhuma derrota ser maior que as anedotas que poderíamos tirar de cada situação. Obrigada por todas as conversas, os cafés, as reuniões e as noites viradas. O andamento não linear do curso nos obriga a construir nossa comunidade, onde a gente se ajuda e é gentil uns com os outros. Por isso, meu agradecimento vai em especial para Alice, Nathan, Jadir, Nic, Caio, Davi, Sorys, Fausto e Barata.

Agradeço também aos professores e companheiros de disciplina da Ecole Centrale, em especial ao Rémy Dammaretz e aos professores Alexandre Kruszewski e Armand Toguyèni, pela disponibilidade e dedicação, sem eles este trabalho não existiria.

Por fim, agradeço ao meu orientador Prof. Dr. Luiz Henrique Silva Colado Barreto pela disponibilidade e por ter acompanhado e contribuído com a concretização desse processo. Aos membros da banca pela disponibilidade e contribuições.

“Dog days are over, dog days are done.”

(Florence + The Machine)

RESUMO

Diante do crescente fluxo de mercadorias, a demanda por agilidade e eficiência na cadeia de suprimentos, mais conhecida em inglês por *Supply Chain*, se tornou uma necessidade atual. Sabendo que depósitos logísticos apresentam grande potencial de automatização, este trabalho propõe a implementação de robôs autônomos no processo de estocagem e entrega de artigos dentro de armazéns. Inicialmente, foi-se definido um cenário de aplicação para mostrar a viabilidade e o potencial da implementação de robôs. Em seguida, estudou-se técnicas de localização interna, por exemplo por radio frequência e ultrassônicas, de comunicação, como TCP/IP, e de controle de movimentos em embarcados. Utilizando dois robôs e uma máquina central, simulou-se um armazém logístico automatizado, onde os robôs conseguiram com êxito se deslocar até o estoque e do estoque até a área de entrega, realizando o caminho mais curto, sem colisões. Os testes e a implementação foram realizados com os materiais disponibilizados nos laboratórios da Ecole Centrale Lille, na França.

Palavras-chave: Armazém Logístico. Automação. Robôs. Cadeia de Suprimentos.

ABSTRACT

Face of the increasing flow of goods, the demand for agility and efficiency in the Supply Chain has become a current necessity. Knowing that logistic warehouses have great potential for automation, this work proposes the implementation of autonomous robots in the process of storage and delivery articles in warehouses. Initially, a scenario was defined to show the feasibility and potential of robot implementation. Then, indoor positioning systems were studied, for example by radio frequency and ultrasound, and communication techniques, such as TCP/IP, and embedded motion control. Using two robots and a central machine, an automated logistics warehouse was simulated, where the robots were able to successfully move to the stock and from the stock to the delivery area, taking the shortest path without collisions. The tests and implementation were carried out with the materials available at the laboratories of the Ecole Centrale Lille in France.

Keywords: Warehouse. Automation. Robots. Supply Chain.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o ambiente empresarial cada vez mais competitivo, a automação de processos, torna-se um forte diferencial para as empresas, na tentativa de alcançar a excelência do atendimento ao cliente. Dessa forma, cada vez mais as empresas buscam alternativas para facilitar o gerenciamento de suas atividades, visando aumentar o controle, a eficiência, a redução de custos e, consequentemente, melhorar o nível do serviço prestado.

Muitas tendências estão impulsionando a automação para o topo da agenda de vários proprietários de depósitos logísticos, nomeadamente estas três: uma crescente escassez de mão-de-obra, uma explosão da procura por parte dos retalhistas em linha, e alguns avanços técnicos intrigantes. Juntando tudo isto, o (MCKINSEY, 2019) estima que a indústria dos transportes e da armazenagem tem o terceiro maior potencial de automatização de qualquer setor. A logística contratual e as empresas de encomendas, ou ainda empresas de logísticas, são particularmente beneficiadas. A automatização está também em cima da mesa em outras empresas de transporte, tais como empresas de camionagem e operadores portuários. Entretanto, esse trabalho é voltado para o cenário de um armazém logístico.

Há muitos processos logísticos que podem ser automatizados, incluindo separação, recuperação de artigos do estoque, processamento de faturas e pedidos, e cumprimento de encomendas. A automatização de tarefas pequenas e repetitivas permite aos trabalhadores concentrarem-se em atividades de nível superior. Os robôs de automação podem trabalhar 24 horas por dia, 365 dias por ano, não fazem pausas para almoço, dias de doença, ou férias, e são 100% livres de dramaturgia. Além do aumento da produtividade e eficiência, o que leva ao aumento da satisfação do cliente, à longo termo, o custo de produção também é reduzido. De acordo com (CLINE *et al.*, 2015), a implementação de robôs no chão de fábrica pode resultar em rápidas poupanças de custos de até 50%.

Diante disso, a motivação deste trabalho é mostrar que a automatização de um deposito logístico pode ser realizada por fases e de maneira comedida, exemplificando a implementação de robôs dentro de um cenário de um armazém logístico em pequena escala.

O intuito é de otimizar o processo de recuperação de artigos do estoque, utilizando-se robôs para buscar artigos do estoque e levar para dentro do caminhão. Para isso, fez-se uso da programação de sistemas em tempo real, do controle discreto e de algoritmos de gerenciamento de rotas, no intuito de atender uma demanda do mercado. Os objetivos deste trabalho serão melhor descritos na sessão seguinte

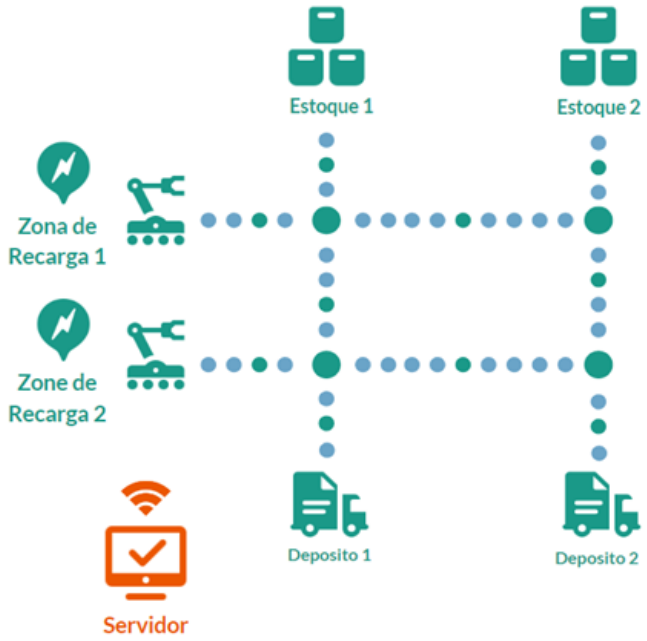
1.1 Objetivos

O objetivo geral desse trabalho é mostrar a facilidade e o potencial da automatização de um depósito logístico, simulando a recuperação e o despejo de artigos por dois robôs dentro de um armazém logístico.

Os mesmos terão suas trajetórias definidas por um supervisor à distância, que calculara as melhores rotas para cada robô, tendo em vista sua localização. Os robôs serão colocados em uma zona mapeada por balizas de localização (ROBOTICS, 2021) que enviarão ao supervisor as coordenadas de cada robô.

De posse das coordenadas em tempo real dos robôs e em posse das comandas de cada item à ser recuperado, o supervisor definirá a trajetória dos robôs para buscar o item no estoque e despejar no caminhão. A Figura 1 ilustra o esquema do que deseja ser desenvolvido.

Figura 1 – Esquema prático a ser desenvolvido



Fonte: propria autora

O objetivo especifico é de desenvolver o controle e a comunicação de dois robôs do tipo *Ultimate 2.0* (MAKEBLOCK, 2015) para simular o recuperação de artigos dentro de um depósito logístico.

Foram utilizadas técnicas de controle para controlar a velocidade dos robôs, como controladores Proporcional Integral (PI) (OGATA, 1982), sistemas de comunicação e programação em tempo real para possibilitar a comunicação entre servidor e robôs, além do gerenciamento de comandas de artigos à recuperar no estoque. O uso de técnicas de pesquisas operacionais

também serão utilizadas para o cálculo de trajetórias otimizadas e reduzir o tempo de trajeto.

1.2 Metodologia adotada

O presente trabalho envolve a junção entre uma pesquisa bibliográfica acerca do tema estudado e a execução de um projeto prático envolvendo o controle de dois robôs na recuperação de artigos dentro de um deposito logístico. Sua classificação, quanto à tipologia, é de pesquisa qualitativa. Ela representa "um conjunto de práticas que transformam o mundo visível em dados representativos, incluindo notas, entrevistas, fotografias, registros e lembretes" (CRESWELL, 2010). As estratégias mais comuns para coleta de dados nessa metodologia são perguntas abertas, dados de entrevistas, dados de observação, dados de documentos e dados audiovisuais, análise de texto e imagem e interpretação de temas e de padrões.

Quanto aos objetivos da pesquisa, é classificada como aplicada, que, conforme (SCHINDLER, 2011) visa à produção de conhecimentos e tecnologias para aplicação prática dos resultados, diferindo da pesquisa intervencionista pela ausência de uma aplicação de cunho imediato em uma circunstância real de necessidade. Trabalha-se com a apresentação de conhecimentos que possibilitem a tomada de decisões e o beneficiamento de grupos, ou ainda que propiciem avanços posteriores em termos de novas tecnologias. Essa característica reflete diretamente o tema deste trabalho, que toma como objetivo a aplicação de robôs dentro de um deposito logístico para otimizar a recuperação de artigos do estoque.

A Figura 2 ilustra a metodologia utilizada. Iniciando com desenvolvimento de uma competência técnica, ou seja, um estudo aprofundado de programação de embarcados, construção de sistemas em tempo real e compreensão de técnicas de controle. Em seguida, estudou-se diferentes soluções possíveis para o cenário em questão, considerando as condições de teste e as implementações práticas.

Após esse embasamento teórico, deu-se início ao desenvolvimento da solução propriamente dita. Começou-se pelo desenvolvimento da comunicação entre robôs e servidor. Em seguida, iniciou-se o controle da trajetória dos robôs, controlando-se a velocidade dos motores. Além disso, foram-se desenvolvidas melhorias na solução proposta, utilizando inteligência artificial e algoritmos de otimização de rotas. Finalmente, os testes foram realizados nos laboratórios da escola de engenheiros francesa Ecole Centrale Lille.

Figura 2 – Metodologia utilizada

Fonte: própria autora

1.3 Estrutura do Trabalho

O presente trabalho é dividido em 5 capítulos, os quais foram organizados da seguinte forma:

Capítulo 01 – Introdução

- Motivação, objetivos, metodologia, estrutura do trabalho.

Capítulo 02 – Fundamentação Teórica

- Revisão da programação de embarcados, de controladores proporcionais e de métodos numéricos.

Capítulo 03 – Projeto e Desenvolvimento

- Estudo de como inserir robôs dentro de um deposito logístico para otimizar a cadeia de suprimentos. Tomou-se o estado da arte deste tema.

Capítulo 04 – Fase de Testes e Resultados

- Aplicação do que foi desenvolvido, corrigindo parâmetros e melhorando a performance.

Capítulo 05 – Conclusão e Trabalhos Futuros

- Resultados comentados e considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Em depósitos logísticos tradicionais, as operações de recuperação de artigo no estoque, de entrega e de gerenciamento do sistema de mercadorias são realizadas por meio de recursos humanos e de algumas ferramentas computacionais. Entretanto, são inúmeros as desvantagens operacionais e competitivas que os depósitos logísticos convencionais apresentam (JABBAR *et al.*, 2016). Primeiro, o tempo de estocagem e destocagem dos inventários logísticos. Segundo, o uso de pessoas para realizar serviços braçais é desperdiçar as habilidades humanas. Isso evidencia a necessidade e o potencial de transformar os depósitos logísticos tradicionais em depósitos logísticos inteligentes ou, em inglês, smart warehouses.

A chave para essa transformação é automatizar o processo de cadeia de suprimentos, fazendo-se uso de sistemas físico-cibernéticos (SFC) (LEE *et al.*, 2015). Como o próprio nome já diz, esses sistemas englobam qualquer tipo de ferramenta computacional, física, ou os dois juntos, visando melhorar um processo. Eles podem monitorar e criar uma copia virtual dos processos industriais reais, tornando possível controlar e comunicar com objetos, além de prever e tomar melhores decisões em tempo real.

Atualmente, melhorar a eficiência no tempo de execução, na energia consumida e na precisão de um processo é uma necessidade, não só de um ponto de vista econômico, como também ambiental. Uma vez diante de um processo inteligente e automático, os erros, o custo e o desperdício diminuem consideravelmente.

Portanto, a proposta deste trabalho é automatizar o processo de *picking* de um deposito logístico, programando e controlando robôs para realizar essa tarefa. Para isso, é necessário explorar o estado da arte atual no que concerne: a geolocalização em ambientes internos de robôs e o gerenciamento da cooperação de multi-robôs, no que tange o controle de trajetórias e a comunicação entre eles.

2.1 Geolocalização de ambientes internos

A localização precisa de objetos beneficia várias aplicações contextuais na era da indústria 4.0, por exemplo, rastreamento e classificação automática de produtos. Atualmente, existem várias técnicas de localização tais como GPS, *Wireless Fidelity* (Wi-Fi), Identificação por Radiofrequência (RFID) e ondas ultrassônicas.

O Sistema de Posicionamento Global (GPS) recupera a mobilidade de unidades

receptoras para calcular sua posição por meio da medição tempo de chegada do sinal de radio emitido pelos satélites (GETTING, 1993). Em áreas externas, o GPS e Sistemas de Navegação Global por Satélite (GNSS) em geral foram adotadas como tecnologia de posicionamento de fato devido às informações de localização altamente precisas que eles fornecem globalmente com um erro tipicamente inferior a 10m.

Entretanto, tal tecnologia baseada em satélite falha em particular ambientes como interiores ou desfiladeiros urbanos. Estas falhas são principalmente devido ao sinal de baixa potência de recepção e baixa visibilidade dos satélites em áreas internas. Portanto, as tecnologias de navegação não-GNSS são essenciais para tais regiões, tais como as citadas anteriormente: Wi-Fi, RFID e ondas ultrasônicas.

O sistema de posicionamento Wi-Fi tem se tornado cada vez mais popular no ambiente interno devido à ampla implantação do Wi-Fi nos últimos anos (MAUTZ, 2012). O uso desses sinais é uma abordagem tentadora, uma vez que os pontos de acesso estão prontamente disponíveis em muitos ambientes internos e é possível utilizar dispositivos de hardware móvel padrão. Além disso, eles apresentam um alcance de 50 m até 100 m que normalmente é coberto pelo alcance da *Wireless Local Area Networks* (WLAN) de Bluetooth ou RFID.

As método mais popular de posicionamento Wi-Fi é fazer uso de RSSI (Indicadores de Força do Sinal Recebido) que são fáceis de extrair em redes 802.11 e podem funcionar em hardware WLAN de prateleira. O propósito deste modelo é calcular a distância entre uma fonte radiante e um receptor através da exploração da atenuação da RSSI com a distância. Em geral, o uso de RSSI em combinação com WLAN pode ser feito utilizando a estratégia de Impressão digital (FP) (KHAN *et al.*, 2017).

Entretanto, uma grande dificuldade técnica no uso de RSSI surge do fato de que, em ambientes internos, alguns sinais não atenuam a distância de forma previsível devido à sombra, reflexão, refração e absorção pelas estruturas e objetos do ambiente. Os valores RSSI dependem em grande parte do ambiente de propagação, tornando muito difícil a criação de modelos adequados que descrevam a relação entre os valores RSSI e a posição do receptor em ambientes internos reais.

Dessa forma, os métodos de impressão digital que dependem da simples comparação de medidas empíricas sem aplicação de um modelo teórico se tornaram o método mais favorável em comparação com a modelagem analítica. Todavia, se faz necessário muitas vezes de uma longa etapa online, onde os RSSI são recuperados de diferentes localizações dentro de um

ambiente e são salvos numa especie de base de dados que ira calibrar a modelagem.

Portanto, é possível perceber que o Wi-Fi apresenta algumas desvantagens, principalmente quando se analisa sua aplicação dentro de um deposito logístico. Por exemplo, em virtude de seus sinais de baixa potência que apresentam curto alcance, os sinais são altamente interferidos pela presença de paredes e prateleiras. Além disso, por estar na mesma banda passante de outros sinais, como *Bluetooth*, micro-ondas e frequências de telefones sem fio, ele se torna facilmente suscetível a interferências.

Outra possibilidade de geolocalização em ambientes internos é a identificação de radio frequência (KHAN *et al.*, 2017). Esse sistema consiste em um leitor com uma antena que interage com transceptores ativos próximos ou tags passivos. Usando tecnologia RFID, os dados podem ser transmitidos das etiquetas RFID para o leitor através de ondas de rádio. Normalmente, os dados consistem na identificação única da etiqueta que pode ser relacionada a informações disponíveis sobre a posição da etiqueta RFID.

Geralmente, os sistemas RFID podem ser discretos para o usuário, integrando as etiquetas no pavimento, debaixo do tapete ou nas paredes sem linha de visão direta, uma vez que as ondas de rádio têm a capacidade de penetração de materiais sólidos até certo ponto. Quanto maior a freqüência, mais o sinal sofre com a atenuação.

O autor de *Indoor Positioning Technologies* (KHAN *et al.*, 2017) descreve no livro varias técnicas de para empregar o método RFID e, dentre elas, destacam-se a de Proximidade, a RSSI e a do Tempo de Chegada. Como o próprio nome já diz, a técnica por proximidade indica a presença de uma etiqueta e depende bastante da frequência de leitura para identificação. Alternativamente, os indicadores de força de sinal recebido podem ser usados para estimativa de faixa grosseira, a fim de aplicar técnicas de multilateração. Por fim, a técnica do Tempo de Chegada que tem como base o calculo da distância entre um leitor e uma etiqueta. Porém para apresentar uma resolução melhor que um metro, deve ser usada uma largura de banda de pelo menos 10 kHz e múltiplas observações precisam ser calculadas como média.

Mesmo com suas desvantagens a tecnologia de RFID tem vantagens atraentes sobre as outras técnicas, especialmente em aplicação de grandes áreas, como são usualmente os depósitos logísticos. Por exemplo, as etiquetas passivas RFID são de baixo custo, de pequeno tamanho e sem bateria, o que torna o RFID pertinente para sistemas de localização na indústria 4.0.

Entretanto, o foco principal deste trabalho esta na simulação de robôs autônomos

dentro de um deposito logístico e, apesar das atraentes características das técnicas RFID, para simplificar a simulação, utilizou-se a geolocalização por ondas ultrassônicas.

Tanto a geolocalização por WLAN, como por RFID, apresentam precisões tipicas de 3 metros ou mais. Segundo Mike Hazas, no artigo *Broadband Ultrasonic Location Systems for Improved Indoor Positioning* (HAZAS; HOPPER, 2006), sistemas de localização com granulometria fina são capazes de fornecer informações de posicionamento com precisão em centímetro, que pode ser usada seja para melhorar as aplicações existentes, seja para possibilitar novas.

Os sistemas de localização por ultrassom têm provado ser uma solução relativamente simples e eficaz para o posicionamento em interiores de granulometria fina, que é o caso da simulação deste trabalho. Transdutores ultrassônicos são relativamente baratos, e seus sinais normalmente têm requisitos de processamento mais baixos do que as soluções por sinais de infravermelho ou de banda larga, resultando em sistemas simples e de menor custo que provaram ser eficazes para áreas amplas e para interiores de granulação fina.

2.2 Comunicação em sistemas de multi-robôs

Sabendo que os robôs se movimentam, e no cenário de deposito logístico, em distâncias consideravelmente grandes, eles necessitam ser suficientemente independentes de fios ou cabos. Dessa forma, a comunicação entre eles deve ser sem fio, mesmo que a comunicação com fio apresente uma grande performance.

Segundo (LIU *et al.*, 2012) uma estrategia comum da comunicação de sistemas de multi-robôs é criar uma central de comando, de forma que todos os robôs em campo serão controlados por ela. Além disso, cada robô deve apresentar um IP exclusivo, de forma que seja possível diferenciar as maquinas. Essas medidas são interessantes para reduzir o numero de dados transmitidos e otimizar o controle da comunicação.

Dentre os métodos de comunicação, um que se destaca em aplicações de robôs mobile é o Cliente/Servidor (HOANG *et al.*, 2015). O modo Cliente-Servidor (CS) é um modo de compartilhamento de informações amplamente utilizado no sistema de informação. O núcleo da estrutura do sistema CS é a distribuição da aplicação de nível de tarefa entre cliente e servidor. O cliente geralmente se refere ao PC ou Raspberry. Ele fornece ao cliente terminal uma interface muito amigável, por exemplo, o *Microsoft Windows* e assim por diante. O servidor fornece ao cliente um grupo de usuários compartilhando o programa de serviço. O servidor de banco de

dados é o mais comum. Ele faz com que muitos clientes compartilhem o mesmo acesso às fontes de informação. Os clientes e servidores no sistema C/S podem ser conectados através de LAN, WAN ou internet.

No módulo Cliente-Servidor, ele permite compartilhar informações e recursos entre sistemas, por exemplo, arquivos, espaço em disco, processadores e periféricos podem colaborar e transmitir mensagem inevitavelmente entre muitos processadores. Alguns dos processadores podem funcionar como clientes enquanto outros estão trabalhando como servidores.

No cenário de um deposito logístico carregado de robôs para exercer o *picking*, estes serão considerado como os clientes no modo C/S, enquanto o supervisor, a máquina central, será o servidor.

2.3 Controle de trajetória em sistemas de multi-robôs

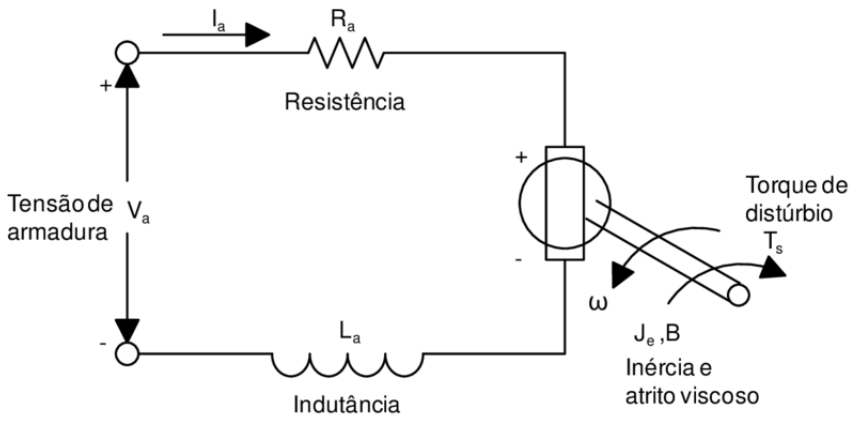
Para a maioria das aplicações envolvendo robôs embarcados, o controle da trajetória é feito manipulando a velocidade angular das rodas que possuem os motores de propulsão, mais especificamente, controlando a tensão que é aplicada neles (NITULESCU, 2008). A maioria desses sistemas, que utilizam motores Corrente Continua (CC), apresentam um comportamento não linear, o que define uma dinâmica mais complexa, o que torna necessário encontrar uma equação matemática que descreve o sistema antes de desenvolver o controlador.

Varias estratégias de controle são propostas por diversos autores com o intuito de controlar a trajetória de um robô. Dentre elas, são enaltecidas as técnicas que utilizam controladores convencionais como a alimentação em malha fechada (NAWAWI; OSMAN, 2017), noções de espaço de estado com alocação de polos, a utilização de observadores de estado e os controladores proporcionais.

Analisar um sistema do ponto de vista de espaços de estados nada mais é que usar uma notação vetorial-matricial que coloca em evidencia as entradas e saídas do sistema. Ela faz parte de uma das técnicas de controle moderno (OGATA, 1982) e é essencial para estudar sistemas que apresentam muitas entradas e muitas saídas, uma vez que a complexidade do sistema não é interferida. Para aplicações envolvendo embarcados, por exemplo, que já apresentam diversas entradas, como posição, velocidade e ângulo, é usual analisar o sistema utilizando espaço de estados.

Para desenhar a equação que representa os motores CC, fez-se uso do trabalho de (KHATOON *et al.*, 2018), utilizando espaço de estados. A partir do modelo de motor mostrado

Figura 3 – Diagrama de um motor CC



Fonte: (KHATOON *et al.*, 2018)

na figura 3, é possível determinar as equações na forma de variáveis de estado que caracterizam o robô. Sabe-se que o torque T_s é proporcional a corrente que circula no circuito da figura I_a como é mostrado na equação 2.1. Já a amplitude da tensão induzida é uma função linear da velocidade angular e pode ser escrita como é feito na equação 2.2. K_s representa as constantes de inércia do motor e K_a a contante que considera o campo magnético e o número de espiras.

$$T_s = I_a K_s. \tag{2.1}$$

Sabendo que os estados de um motor CC são definidos como velocidade e aceleração angular e de posse das leis de Kichhorff (MORAIS, 2016), é trivial manipular as equações e encontrar a equação 2.3 que caracteriza o sistema na forma de variáveis de estado.

$$V_a = K_a \omega. \tag{2.2}$$

$$\begin{pmatrix} \dot{\theta} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -\frac{K_s K_a}{I_a R} \end{pmatrix} \begin{pmatrix} \theta \\ \omega \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{K_s}{I_a R} & -\frac{1}{I_a} \end{pmatrix} \begin{pmatrix} V_a \\ T_s. \end{pmatrix} \tag{2.3}$$

A partir dessa representação é possível estudar a estabilidade e controlabilidade do sistema e, uma vez sendo instável e controlável, estudar a alocação de polos de controle. Ao invés de abordar apenas os polos dominantes, que é o caso de sistemas de controle convencionais que apresentam uma entrada e uma saída, a alocação de polos permite alocar todos os polos de malhas fechada. Dessa forma, as variáveis do sistema são consideradas e amortizadas (OGATA, 1982).

Uma vez que a alocação de polos em malha fechada só pode ser realizada se todas as variáveis estiverem disponíveis para realimentação, na pratica que normalmente não é o caso, faz-se uso de observadores de estado. Um observador de estado estima as variáveis de estado baseado nas medidas das variáveis de saída e das variáveis de controle (ISHIKAWA; SAMPEI, 1995).

Em diversas aplicações envolvendo embarcados, o emprego de controladores do tipo proporcional, proporcional integral (PI) e proporcional integral derivativo (PID) é bem comum, por exemplo, no caso da estabilização de um pêndulo invertido ou de braços robóticos e aplicações hidráulicas. Nesta aplicação, o controle empregado faz uso de um controlador PI, com observador de estado.

2.4 Algoritmos de otimização de trajetória

Sabendo que os robôs seguirão trajetórias para ir recuperar um artigo no estoque e em seguida entrega-lo na zona de deposito, existe, portanto, a necessidade de definir esses caminhos. Para isso, utilizou-se algoritmos do problema do caminho mais curto a fim de otimizar o tempo de busca.

Antes de compreender o que é o problema do caminho mais curto e como esse algoritmo foi se desenvolvendo para algoritmos mais refinados, como Djikstra, é preciso compreender primeiramente em que se consiste a teoria de gráficos.

Um gráfico é um par (E,A) (CHEN *et al.*, 2022) composto por um conjunto E de pontos (ou nós) e um conjunto A de arcos que são pares de pontos: a origem e o fim do arco. O conjunto A define uma relação binária entre os pontos, no caso em que E e A são conjuntos finitos. Um caminho é uma sequencia de arcos "em linha": para dois arcos sucessivos, a origem do primeiro é o fim do segundo. Ela une sua origem (a do primeiro arco) à sua extremidade (a do último arco), passando por uma sequencia de pontos. Para o caminho vazio denotado θ, concordamos que ele une qualquer ponto a si mesmo. Um caminho não vazio é um circuito se sua origem e fim forem idênticos; é elementar se não contiver nenhum circuito, exceto possivelmente ele mesmo.

Dito isso, o problema do caminho mais curto é o problema algorítmico de encontrar um caminho de um vértice a outro de modo que a soma dos pesos dos arcos nesse caminho seja mínima. Há muitas variantes deste problema, dependendo se o gráfico é finito, direcionado ou não, se cada arco ou borda tem um valor que pode ser um peso ou um comprimento. Um caminho

mais curto entre dois nós é um caminho que minimiza a soma dos valores dos arcos percorridos. Para calcular um caminho mais curto, há muitos algoritmos, dependendo da natureza dos valores e das restrições adicionais que podem ser impostas.

Neste trabalho, considerou-se que todos os circuitos do gráfico possuem um peso positivo e que os circuitos mais curtos são aqueles que representam o menor peso. Dentre os algoritmos do caminho mais curto, o que teoricamente é mais eficiente para essas referência é o algoritmo de Djikstra (DIJKSTRA, 2021).

O algoritmo de Dijkstra consiste em encontrar os caminhos mais curtos de um único vértice s ∈ S para cada outro vértice de um gráfico ponderado G = (S, A). Antes de determinar os caminhos mais curtos entre s e qualquer outro vértice t, primeiro determina-se o comprimento de cada um desses caminhos mais curtos, ou seja, a função.

Para isso, usa-se um atributo d[v] do vértice v ∈ S que contém o peso do suposto caminho mais curto de s para v. Este atributo é corrigido à medida que o algoritmo prossegue para que contenha (v) no final da execução. Obviamente, este atributo é, em cada etapa do o algoritmo, um dos maiores do peso de um caminho mais curto de s para v.

Inicialmente d[s] = 0 e d[v] = ∞ para cada v ≠ s. O algoritmo mantém uma partição dos vértices em três estados: vértices não rotulados, aqueles com custos tentativos infinitos; vértices rotulados, aqueles com custo tentativo finito cujo custo mínimo ainda não é conhecido; e vértices escaneados, aqueles cujo custo mínimo é conhecido. Inicialmente, os s são etiquetados e todos os outros vértices não são etiquetados.

Em seguida, começa o processo de escanear os outros vértices. Se almeja-se escanear o vértice (u,v) ∈ S, isso consiste em testar se é possível, ao passar por u, melhorar o caminho mais curto para v. Se o teste for negativo, nenhuma atualização é realizada. Se o teste for positivo, significa que um caminho menor é exibido para passar de vértice de vértice s para vértice v e atualização ilustrada na equação 2.4 deve então ser realizada.

$$d[v] = d[u] + g_{u,v}. \tag{2.4}$$

Em cada etapa do algoritmo, é selecionado o vértice u que satisfaz as seguintes propriedades:

1. u ainda não foi selecionado;
2. o atributo d[u] é o mais baixo (entre todos os vértices ainda não selecionados).

Isto é chamado de algoritmo glutão: a cada passo é selecionada a melhor sub-solução, ou seja, o vértice com a melhor distância. Uma vez visitados todos os vértices, encontra-se todos os caminhos mais curtos (de s a qualquer v) de tamanho inferior ou igual ao número de vértices em todos. Encontra-se, portanto, todos os caminhos mais curtos (de s a qualquer v).

3 PROJETO E DESENVOLVIMENTO

Para o desenvolvimento de um projeto ou de uma solução de engenharia, é preciso, inicialmente, definir um cenário de aplicação e, em seguida, traçar as funcionalidades do sistema desejado. Neste trabalho, como já citado na seção 1.2, o cenário de aplicação definido é a simulação de um deposito logístico automatizado com robôs que coletam artigos do estoque e os entregam na zona desejada.

Os *hardwares* desta aplicação são compostos de dois Raspberry pi, um para cada robô, um computador, o que vem a ser a central de comando – supervisor -, quatro balizas de geolocalização fixas e duas moveis. A partir do cenário traçado, definiu-se a necessidade de se construir um programa para o supervisor e um programa independente para os robôs. Estes possuiriam códigos semelhantes, uma vez que possuem as mesmas funções. Suas diferenças serão definidas no programa do supervisor, definindo-se robô 1 e robô 2.

As balizas de geolocalização têm o papel de recuperar as coordenadas dos robôs no ambiente simulado. Assim, o programa que recupera essas informações para enviar ao supervisor está inserido no código deste. Portanto, trabalha-se com a criação de dois programas principais, robô e supervisor que serão melhor detalhados nas sessões seguintes.

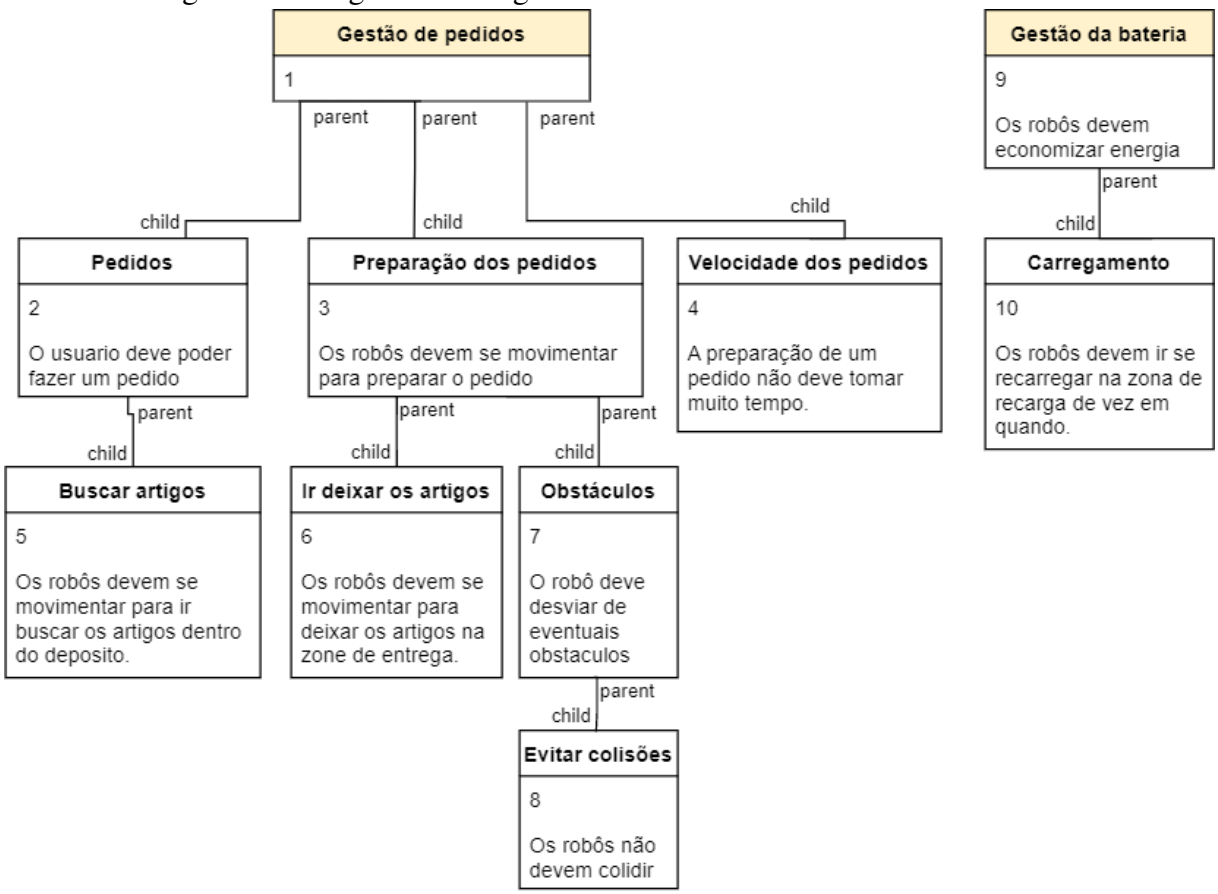
A fim de definir e entender as funcionalidades que cada agente terá, foram desenhados diagramas de exigência, de sequência e de caso de utilização. Vale ressaltar que as balizas de geolocalização não são mostradas dos diagramas apresentados, porém elas são consideradas.

Um diagrama de exigências modela os requisitos a serem verificados pelo sistema, ligando as soluções implementadas com as necessidades definidas nas especificações (T., 2016). Ele traduz funcionalidades ou restrições, dentre elas o desempenho, a confiabilidade e as condições de segurança, com o que deve ser satisfeito pelo sistema. O diagrama de exigências deste trabalho é ilustrado na Figura 4.

Já um diagrama de sequência é um tipo de diagrama de iteração, pois descreve como e em que ordem vários objetos operam juntos (LUCIDCHART, 2022). Estes diagramas são usados tanto por desenvolvedores de software quanto por gerentes de negócios para analisar as exigências de um novo sistema ou para documentar um processo existente. Os diagramas de sequência são às vezes chamados de diagramas de eventos ou cenários de eventos.

A Figura 5 ilustra o diagrama de sequência deste trabalho. Nele é possível ver e entender o funcionamento lógico e a sequência de eventos do cenário proposto. As caixas retangulares representam o tempo necessário de cada evento e as malhas em amarelo designam

Figura 4 – Diagrama de exigências



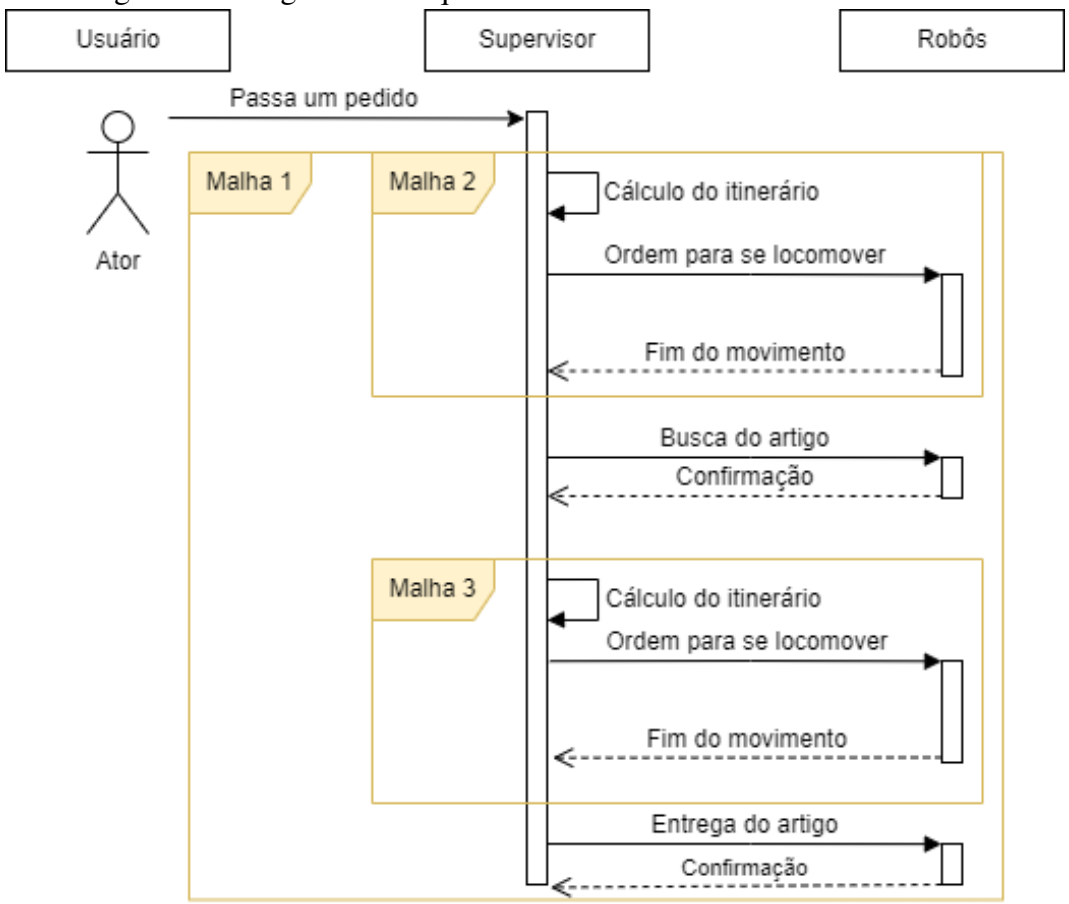
Fonte: propria autora.

que aquele evento acontece em *loop*.

Diagramas de caso de utilização ilustram e definem o contexto e as exigências de todo um sistema, ou partes essenciais deste (IBM, 2021). Estes são desenvolvidos principalmente nos estágios iniciais de um projeto e os consultará ao longo de todo o processo de desenvolvimento. Pode-se modelar um sistema complexo com um único diagrama de caso de uso, ou criar muitos diagramas de caso, para modelar os componentes do sistema. A Figura 6 ilustra o diagrama de caso de uso deste trabalho.

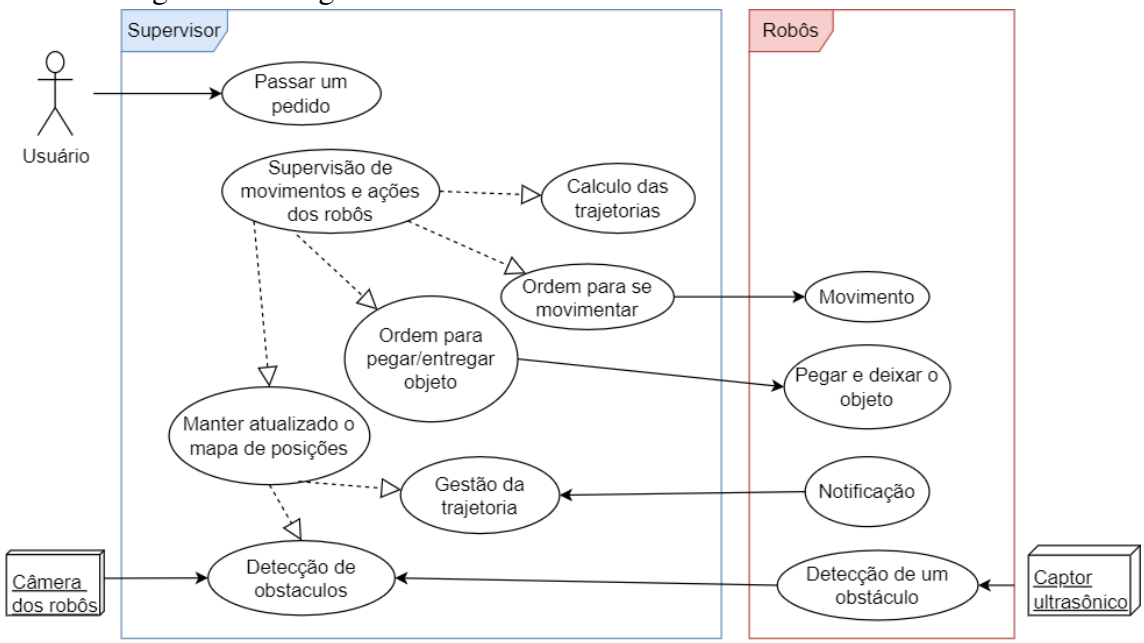
A partir desses diagramas é possível compreender as funcionalidades de cada agente e começar a coloca-las em pratica. O trabalho foi divido em dois desenvolvimento principais, o programa do supervisor e o programa do robô. Mesmo que as funcionalidades sejam intrínsecas, essa divisão permite, do ponto de vista do programador, uma melhor visão do que se deseja obter.

Figura 5 – Diagrama de sequência



Fonte: própria autora.

Figura 6 – Diagrama de caso de uso



Fonte: própria autora.

3.1 Cenário e visão geral

De posse dos diagramas desenhados, a abordagem para simular a automatização de um deposito logístico com robôs, pode ser então definida.

As quatro balizas de geolocalização fixas delimitam o espaço onde os robôs vão se movimentar. Essas recuperam as coordenadas de todos os pontos desse espaço e, uma vez que serão acopladas balizas moveis nos robôs, será possível conhecer as posições destes para então traçar suas trajetórias. Como visto em 3.2, essas balizas são controladas à partir de um programa – Marvelmind – e, em linguagem C, é possível recuperar as coordenadas das balizas moveis vistas no Marvelmind. Esse processo faz parte das funcionalidades do supervisor e será melhor detalhado na sessão seguinte.

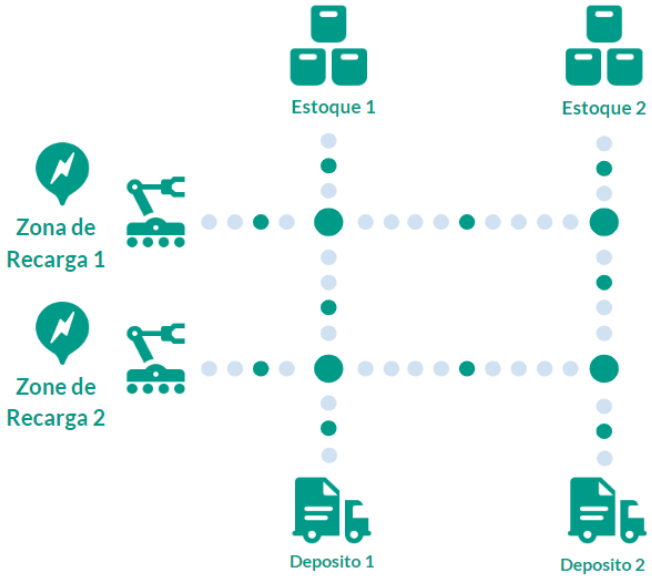
Dentro deste espaço de simulação, em 3m² foram definidos seis pontos fixos: dois representando as duas estações de recarga; dois representando os dois estoques e dois representando as duas zonas de deposito. Em seguida, traçou-se caminhos fixos, nos quais os robôs devem passar ou cruzar para se locomover a fim de definir uma espécie de pista para os robôs. Foi uma maneira de tornar o cenário mais próximo da realidade, uma vez fixado os caminhos dos robôs, os demais do deposito logístico podem ser dedicados a outros usos.

Esses caminhos foram fragmentos em pontos de checagem, para prevenir colisões entre os robôs. A Figura 7 ilustra os pontos de checagem em verde escuro. Para atravessar os cruzamentos os robôs deverão parar antes do cruzamento e verificar se o cruzamento esta livre, se sim eles poderão passar.

Será responsabilidade do supervisor de recuperar as coordenadas dos pontos de cruzamento, estoque, deposito e recarga. Além de receber os pedidos e envia-los aos robôs, calcular suas trajetórias e enviar autorizações de movimentos a fim de prevenir colisões. O supervisor será a maquina pensante de todo o sistema.

Os robôs, por sua vez, deverão receber e executar todos esses comandos através de um raspberry que neles estão acoplados. Além de controlar o motor e seu respectivo giroscópio para suavizar os movimentos dos robôs. A sessão seguinte explicara em detalhe o programa desenvolvido desses dois agentes.

Figura 7 – Caminhos dos robôs dentro da zona das balizas fixas



Fonte: própria autora.

3.2 Características do robô

Dois robôs (MAKEBLOCK, 2015) foram utilizados neste trabalho para simular o cenário de picking de um deposito logístico. A Figura 8 ilustra as características físicas do robô. Foram feitas algumas modificações no seus hardwares, então suas especificações são um pouco diferentes do modelo de fábrica.

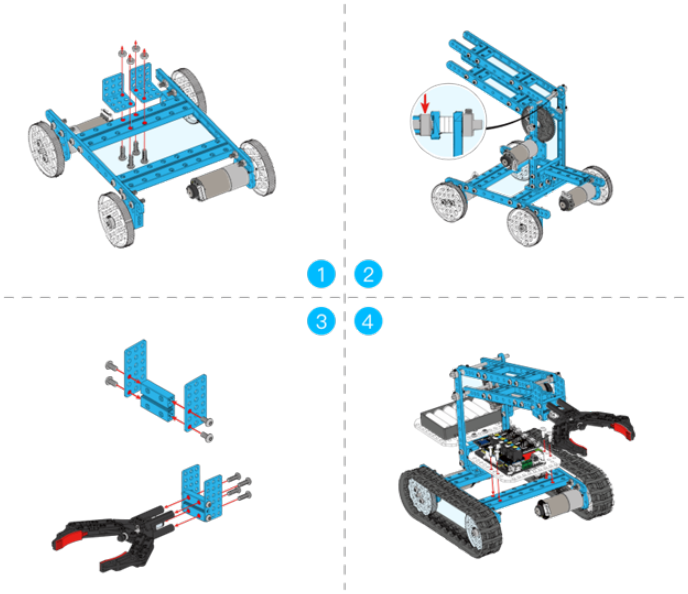
Dentre as modificações feitas, pode-se elencar:

- A percepção do robô é atualizada com um sensor de linha de seguimento, um sensor ultrassônico, uma *Inertial Mesure Unit* de 6 dof e um sistema de localização interna conectado a um Rapsberry Pi 3. O Raspberry é adicionado para poder ser conectado ao computador e para uma posterior capacidade de visão por computador.
- Um espaço para localizar uma bateria de lítio foi adicionado abaixo do Rasperry Pi.

Todas as especificações detalhadas do robô estão descritas na Figura A. Dentre os componentes, vale ressaltar que algumas funcionalidades do robô não foram exploradas, por exemplo seu braço robótico, o sensor de linha de seguimento e o sensor de distância para detectar obstáculos. Outra estratégia foi desenvolvida para prevenir colisões e ela será abordada nas seções seguintes.

Cada robô conta com um Arduino acoplado para possibilitar o controle da velocidade

Figura 8 – Modelo dos robôs utilizados



Fonte: (MAKEBLOCK, 2015)

Figura 9 – Balizas de geolocalização

Fonte: (MAKEBLOCK, 2015)

das rodas. Neste trabalho, o Arduino serviu principalmente para receber os comandos do Raspberry e traduzi-los em comandos para os motores DC a fim de controlar a trajetória dos robôs.

Além do Raspberry Pi 3 e do Arduino, foram adicionadas balizas de geolocalização moveis sobre os robôs. Essas balizas fazem parte da coleção de monitoração de ambientes internos da Marvelmind (ROBOTICS, 2021). Cada baliza móvel se comunica com balizas fixas que definem no plano todas as coordenadas do ambiente de simulação. Dessa forma, sabendo que cada baliza móvel representa um robô, torna-se possível leva-los para o caminho desejado. A Figura 9 mostra o formato das balizas utilizadas.

Na figura, o menor dispositivo é conectado em um computador, para essa aplicação é o computador do supervisor, e ele é responsável por captar as informação de todas as balizas e enviar ao supervisor.

3.3 Programa do supervisor

A fim de simplificar a comunicação entre os robôs, definiu-se uma máquina supervisória central para controlar todo o sistema. Dessa forma, a definição da trajetória, as informações de geolocalização e a comunicação entre robôs é monopolizada em uma só maquina a fim de facilitar o gerenciamento do sistema.

O supervisor é a máquina responsável por receber os pedidos de artigos no estoque, identificar os robôs disponíveis para ir recuperar o artigo, definir os caminhos dos robôs para prevenir colisões e garantir a comunicação entre eles. A partir desta descrição é possível, portanto, definir quais processos são necessários para desenvolver essas funcionalidades.

Inicialmente, criou-se um processo para a gestão de pedidos, ou seja, o processo que será encarregado de receber as comandas de artigos a serem pegos no estoque e encaminhados para zona de deposito. Dentre os algoritmos de gerenciamento de pedidos existem dois tipos: os algoritmos não preventivos que impedem que um processo assuma o controle do processador antes que o processo atual tenha terminado; e os algoritmos preventivos que podem se apropriar do processador por um processo antes do final do processo atual.

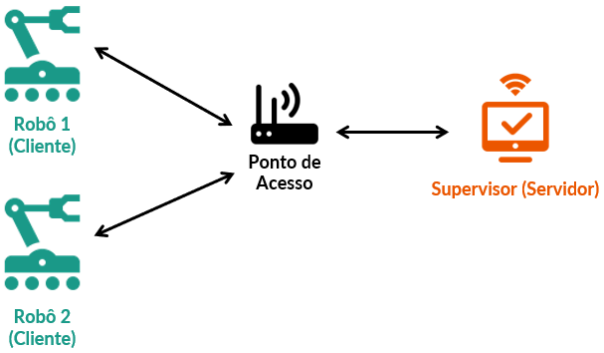
Neste trabalho, utilizou-se um algoritmo não preventivo do tipo *First In First Out* (FIFO), ou seja, que trata os processos de acordo com sua ordem de chegada, sem considerar o tempo de execução e nenhuma outra característica. Esse processo é repetido até que todos os processos da fila se esgotem. Foram criados sinais para informar a chegada de cada pedido e a sua adição à lista de espera. Assim, é possível ter uma ideia da quantidade de pedidos e poder gerencia-los.

Em seguida, inicia-se um processo para reconhecer os robôs que estão em campo e poder associa-los ao pedido que está sendo tratado. Para isso, fez-se uso do processo de ping que faz parte dos comandos POSIX e está presente no ambiente Linux. O comando *ping* envia uma mensagem de gestão dentro de uma rede para receber uma mensagem do mesmo tipo de um anfitrião da rede. Se o anfitrião é operacional e se encontra dentro da rede, ele responderá à mensagem.

Dessa forma, uma vez lançado esse processo, o supervisor consegue reconhecer todos os robôs em campo, saber seus endereços e começar a comunicação com eles. A figura 10 ilustra os fluxos de comunicação sem fio do sistema.

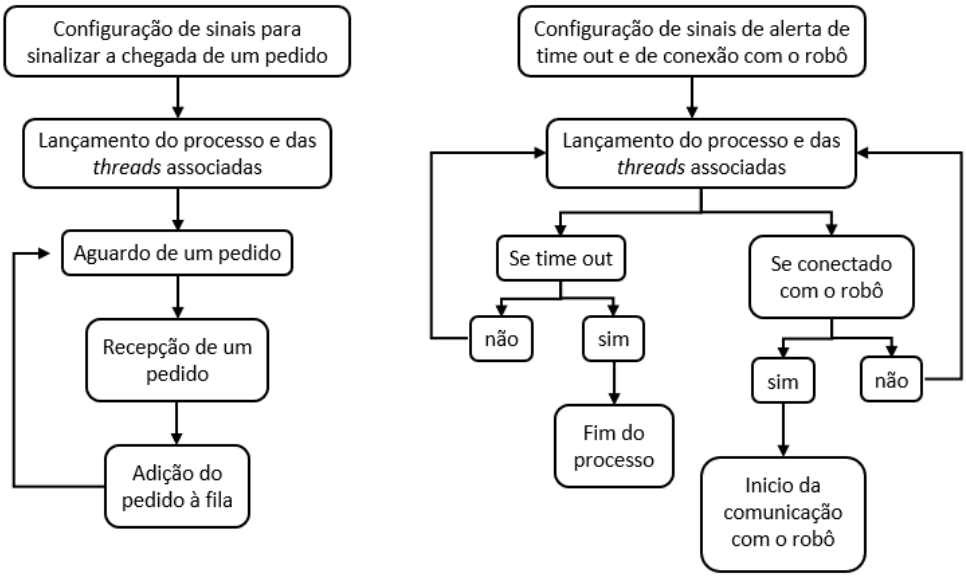
Ademais, foram configurados sinais de alarme para identificar quando o tempo de procura de novas maquinas na rede acaba, um sinal para informar uma vez que o robô esta

Figura 10 – Comunicação TCP/IP do sistema



Fonte: própria autora.

Figura 11 – Processos de comunicação e gestão de pedidos



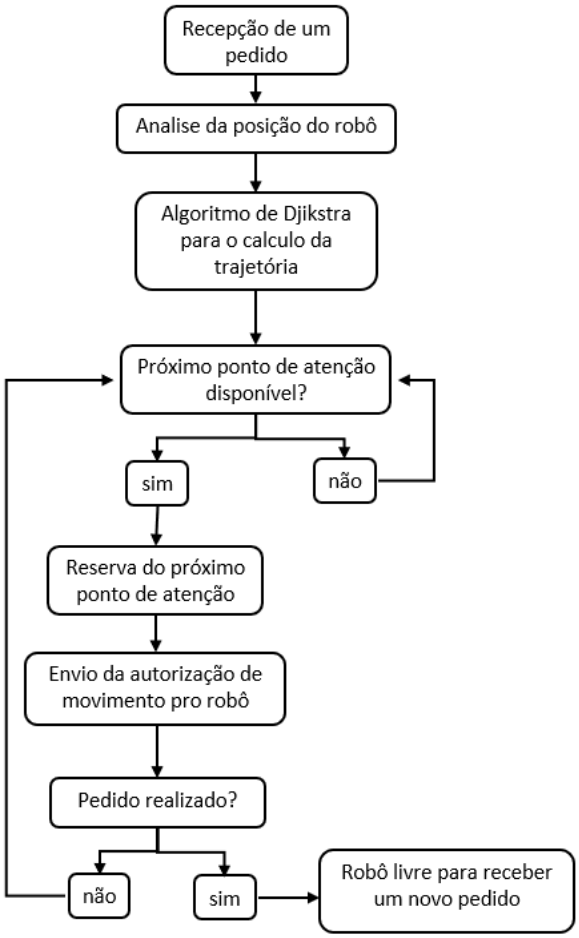
Fonte: própria autora.

conectado com o supervisor e um para informar a perda de conexão. A figura 11 ilustra o fluxo computacional desses dois processos, bem como a configuração dos principais sinais de alerta.

Vale ressaltar que para criação dos processos do tipo FIFO, utilizou-se a função POSIX *fifo_t* e sua chamada é feita no programa de chamadas de funções e bibliotecas do supervisor no apêndice C. Toda a configuração desses processos foi feita no código principal do supervisor contido no apêndice A.

Uma vez que foi definido no escopo da simulação do projeto que os robôs teriam espaços dedicados à sua movimentação, é preciso portanto recuperar os pontos fixos que caracterizam esses espaços. Então, antes do calculo do melhor caminhos, existe uma fase de inicialização do campo, ou seja, a recuperação dos pontos de atenção. Tais pontos representam a localização das zonas de deposito, os estoques, as zonas de recarga de onde partem os robôs e os pontos de atenção dos cruzamentos. Essa fase nada mais é que o arquivamento das coordenadas desses

Figura 12 – Supervisão do caminho dos robôs



Fonte: própria autora.

pontos dadas pelas balizas de geolocalização. No apêndice B é possível ver o detalhamento do código de recuperação das posições iniciais.

Em posse dos pontos de atenção, construiu-se um terceiro processo: a supervisão da trajetória dos robôs responsável por definir o caminho que os robôs irão seguir. Levando em consideração o pedido feito e a posição de cada robô, esse processo lança as tarefas para autorizar cada movimento e receber as posições dos robôs a cada instante. A figura 12 ilustra como a trajetória é definida.

Essas tarefas estão presente no programa de supervisão, a quarta parte do programa do supervisor, e é neste programa que o supervisor define a melhor trajetória possível para o robô realizar, ou seja, aquela que levará menos tempo. Para isso, ele faz uso do algoritmo do melhor caminho de Djikstra, descrito na seção 2.

É importante salientar que todos os programas construídos acabam fazendo uso recorrente de uma mesma função. Por exemplo, a função *check* criada para verificação de erros, ela é usada em todas as criações de processos, sinais e tarefas, tanto no código do supervisor

quanto no código do robô. Portanto, criou-se um diretório *include* para armazenar todas as bibliotecas dessas funções criadas para servir este trabalho e elas estão listadas na tabela 1.

Finalmente, a execução das funcionalidades do supervisor se dá da seguinte maneira: inicialização do mapa de posições fixas, detecção de robôs disponíveis busca de artigos, verificação da lista de pedidos e envio de pedidos e posições para os robôs.

Em seguida, os processos descritos anteriormente são lançados e o supervisor começa a interagir com os robôs, recebendo suas posições e enviando novas posições. A cada cruzamento, o supervisor identifica que o robô esta no ponto de atenção, então ele analisa o movimento dos demais robôs para ver se o cruzamento esta livre e, assim, poder autorizar o movimento do robô em questão.

Tabela 1 – Bibliotecas criadas

Biblioteca	Função
check.h	Verificação de erros
communication.h	Envio e espera de uma mensagem
dijkstra.h	Envio da matriz dijkstra
fifo.h	Criação de um processo FIFO
graph.h	Envio da matriz que representa os pontos no ambiente de simulação
marvelmind.h	Recuperação das posições do marvelmind
matrix.h	Envio de uma matriz de tamanho NxN
signals.h	Configuração de um sinal
types.h	biblioteca que armazena todas as variaveis utilizadas no codigo

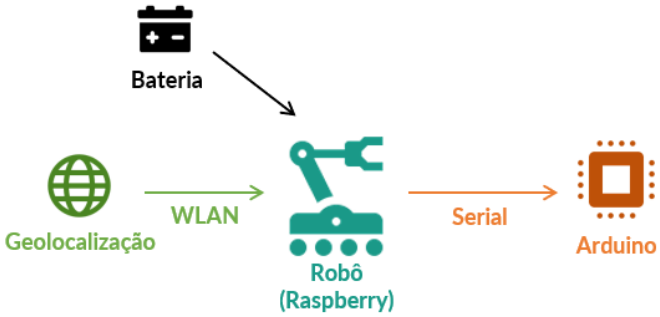
Fonte: propria autora.

3.4 Programa do robô

Os dois robôs móveis utilizados nessa simulação são idênticos e do tipo (MAKEBLOCK, 2015). Para esta aplicação, à sua estrutura inicial, que já dispõe de um Arduíno, foram adicionados um Raspberry Pi, uma bateria portátil e uma antena de sinais ultrassônicos. O Raspberry Pi é um microprocessador encarregado de abrigar os códigos em linguagem C e de estabelecer a conexão WLAN a fim de permitir a comunicação TCP/IP com o supervisor. Ele é conectado em série com o microcontrolador Arduíno a fim de receber e enviar os comandos de movimentos do robô.

Já a bateria portátil é responsável por alimentar todo o sistema e é conectada em série com a baliza de localização móvel e com o Arduíno. Por fim, a antena ou baliza de sinais ultrassônicos (ROBOTICS, 2021) envia sua localização para o servidor Marvelmind, para posteriormente o supervisor recebê-las. A figura 13 ilustra como as conexões foram feitas.

Figura 13 – Conexões de cada robô



Fonte: própria autora.

Sabendo que os robôs são encarregados de: estabelecer uma comunicação com o supervisor, estar relacionado com o mesmo plano geográfico que o supervisor, enviar comandos de movimento para o Arduíno e captar o sinal da baliza de localização móvel; é possível definir as tarefas e processos que o programa dos robôs deve conter.

Inicialmente, semelhante ao programa do supervisor, é lançado o processo de *ping* para reconhecer a máquina supervisora e possibilitar a comunicação. Sinais para alertar que a conexão foi estabelecida ou interrompida foram igualmente configurados. A *thread* associada a esse processo é a de receber ou enviar uma mensagem ao supervisor.

O segundo processo do programa é dedicado à trajetória do robô. Ele possui uma *thread* responsável pela pela conexão com o ambiente *Marvelmind* e receber a posição real do robô. É importante lembrar que, neste trabalho para simplificar a simulação, escolheu-se uma metodologia assíncrona e centralizada, de forma que somente o supervisor sabe a posição dos demais robôs e, portanto, aquele é responsável por prevenir colisões, quando define o caminho de cada robô.

O terceiro processo esta relacionado com o movimento do robô. A ele estão associadas *threads* que enviam para o Arduíno os comandos para os dois motores, para avançar, recuar, girar e ficar parado, à depender da posição atual do robô em comparação com aquela na qual ele deseja chegar.

Funciona da seguinte maneira: o robô recebe uma mensagem do supervisor, avisando que ele deve ir para uma posição x,y. O robô, portanto, de posse da sua posição inicial - x0, y0 -, calcula o ângulo entre a posição inicial e a final. Esse ângulo será o parâmetro de controle do movimento que deverá tender a zero.

Sabendo que o robô possui um ângulo definido de partida, medido pelo modulo giroscópio contido no Arduíno, foi-se necessário realizar uma calibragem inicial para corrigir

esse ângulo com aquele do parâmetro do controlador. Portanto, inicialmente, o robô realiza um movimento para frente e para trás para recuperar esse ângulo e corrigi-lo na configuração da referência do controlador.

Antes de criar a malha do controlador, é preciso analisar as equações de cinemática do robô para extrair as duas variáveis de estado - ângulo do movimento e a velocidade linear - que serão empregadas pelo controlador em seguida. Para isso, este trabalho fez uso da analise cinemática do robô contido em (ÖZBARAN *et al.*, 2020) e aplicou-se o mesmo método.

Além disso, vale ressaltar que o controle de posição não foi desenvolvido uma vez que foi utilizado os dispositivos Marvelmind para a geolocalização. Portanto, o controlador desenvolvido serviu apenas para controlar a velocidade dos robôs de forma a mantê-los dentro da trajetória.

Como já visto na seção 3.2, robôs são movidos por dois servos motores de corrente continua, portanto, duas velocidades. Essas duas velocidades serão permanentemente dois vetores paralelos que compartilham o mesmo eixo e, dessa forma, a velocidade linear será a media das duas velocidades dos motores. A equação 3.1 ilustra isso.

$$v = \frac{v_L + v_R}{2}. \tag{3.1}$$

Com isso, é possível calcular as velocidades nos eixos x e y como é mostrado nas equações 3.2 e 3.3.

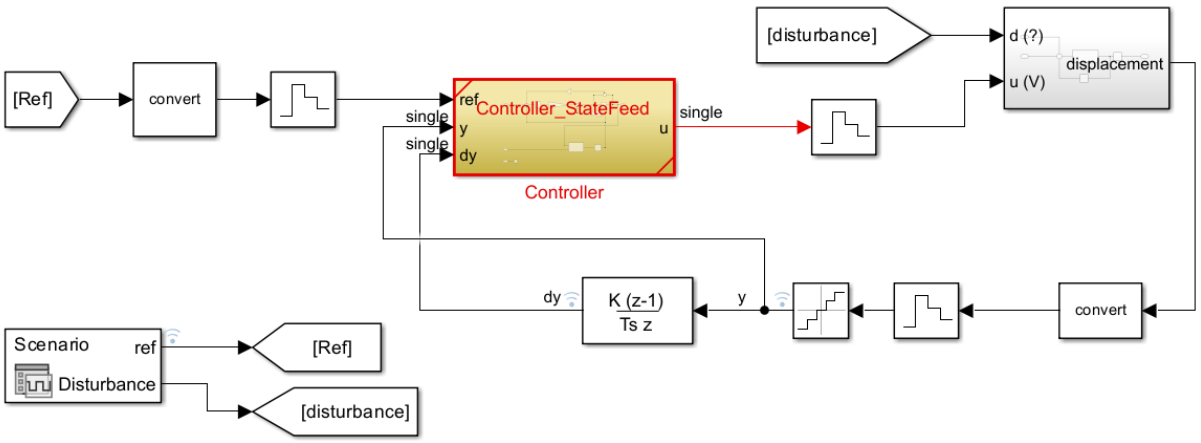
$$\dot{x} = \frac{v_L + v_R}{2} sin\theta. \tag{3.2}$$

$$\dot{y} = \frac{v_L + v_R}{2} cos\theta. \tag{3.3}$$

A partir disso, é suficiente analisar as posições cartesianas x e y como referencias da roda esquerda e direita para chegar à conclusão da equação 3.4.

$$\dot{\theta} = \frac{v_L + v_R}{l_A}. \tag{3.4}$$

Figura 14 – Controlador no Simulink



Fonte: própria autora.

Sabendo que as equações 3.2 e 3.3 não são lineares, é necessário tomar como base algumas suposições que não prejudicarão a execução do projeto. Definiu-se que as duas acelerações instantâneas aR e aL são sempre iguais em modulo. Se uma delas é positiva, o robô executa um movimento linear, se uma delas é negativa o robô executará uma curva especial no seu próprio eixo.

$$H(s) = \frac{\omega(s)}{U(s)} = \frac{K}{1+Ts}. \tag{3.5}$$

O servo motor de corrente continua é uma das formas mais comuns para energizar um sistema de robôs embarcados (NITULESCU, 2008). Ele é um sistema de primeira ordem e sua função de transferência é dada pela equação 3.5. Empregando um controlador PI no domínio do tempo, a estrutura de controle da velocidade para um servo motor é ilustrada na figura 15. A referência do ângulo de entrada (theta) é calculada tomando como base as posições iniciais e finais do robô, recuperadas com as antenas do *Marvelmind*.

O controle foi construído utilizando a alocação de polos e fazendo uso de observadores de estado. Para isso, o sistema precisa ser definido na forma de espaço de estados. Para simplificar a aplicação, reduziu-se o numero de constantes, assim, as matrizes A, B, C são encontradas a partir da função de transferência encontrada anteriormente e são da forma 3.6. A tabela 2 mostra os valores das constantes utilizados.

$$A = [\frac{-1}{\tau}]; \qquad B = [\frac{K}{\tau}]; \qquad C = [1]. \tag{3.6}$$

Tabela 2 – Constantes utilizadas no controle

Constante	Valor
K	50 mm/s/V
τ	0,02 s
Distância entre os dentes da correia (L)	2,032 mm
Numero de dentes da correia (N)	90
Raio da roda (R)	46 mm
Pulso por giro do motor (P)	8
Tempo de resposta do controlador	5ms

Fonte: própria autora.

A partir dessa representação e utilizando a ferramenta *MATLAB R2021a*, é possível, portanto, definir a matrizes do sistema de variáveis de estado, no espaço continuo e discreto. Em seguida, para estudar a estabilidade desse sistema e sua controlabilidade, é necessário analisar seus valores próprios (*vp*) e a matriz de controlabilidade (*ctrb*) que podem ser geradas através dos comandos *eig* e *ctrb* do Matlab. A equação 3.7 mostra o resultado.

A partir desses resultados, nota-se que o sistema é controlável e observável. Portanto, sabendo que o sistema é de primeira ordem, a malha de controle desenvolvida foi um controlador proporcional em malha fechada com um observador.

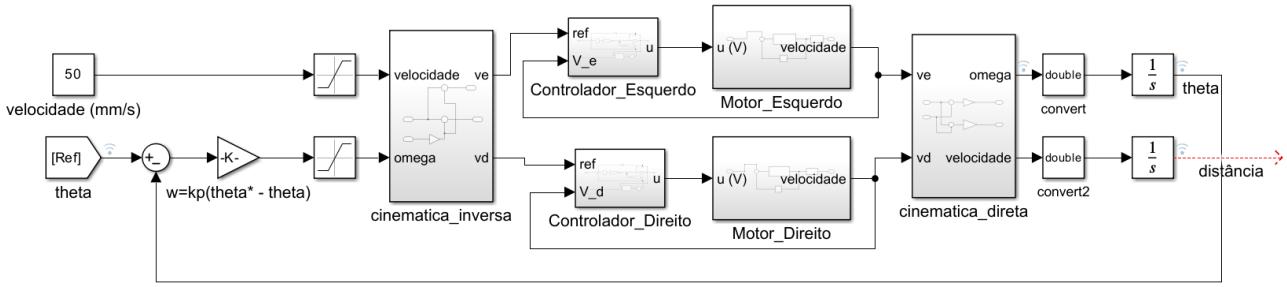
Uma vez que o programa de cada robô estará contido no seu respectivo Raspberry Pi e os comandos de controle de velocidade serão enviados em linhas de código para o Arduíno, a equação de controle deve ser feita de maneira discreta. O esquema do Simulink é mostrado na figura 14.

$$vp = -50; \qquad ctrb = 2500; \qquad rank = 1. \tag{3.7}$$

O esquemático final do controle é ilustrado na figura 15 e leva em consideração os dois motores. Para esse esquema, definiu-se a velocidade V como constante, para Theta tendendo ao Theta de referência. Dessa forma, no caso em que o ângulo de referência for maior que 180°, com a velocidade constante, o robô fará uma parábola para se alinhar com a direção do alvo. Essa escolha toma mais tempo, porém simplifica o processo.

Uma vez que o controlador foi definido e escrito em linhas de código, com a conexão com o supervisor estabelecida e os motores suficientemente controlados para executar os movimentos necessários, o programa do robô é finalizado e ele segue detalhado no apêndice E.

Figura 15 – Malha final de controle



Fonte: própria autora.

4 FASE DE TESTES E RESULTADOS

Com os programas, estudos e análises finalizados, inicia-se, então, a fase de testes, onde, finalmente, a comunicação, o controle e a localização serão postos para atuar juntos.

O cenário de teste é composto por uma estrutura de madeira, uma área de 3m² para locomoção dos robôs, dois robôs do tipo Ultimate Makeblock (MAKEBLOCK, 2015), um computador externo, para representar o supervisor, e as antenas ultrassônicas Marvelmind (ROBOTICS, 2021).

O suporte de madeira foi construído para acoplar as antenas fixas do dispositivo de geolocalização Marvelmind e na figura 16 é possível ver que o software recebe o sinais das quatro antenas. A área interna formada pelas arestas 1, 2, 3, 4 corresponde à cobertura de sinais da antena e é nessa área que os robôs vão se locomover.

É importante salientar que antenas moveis também foram fixadas nos robôs para sinalizar quando eles se deslocam no plano. Esse cenário foi realizado, fazendo-se uso dos equipamentos e dispositivos contidos no laboratório de automação da Ecole Centrale Lile, na França. A montagem do robô final é ilustrada na figura 17.

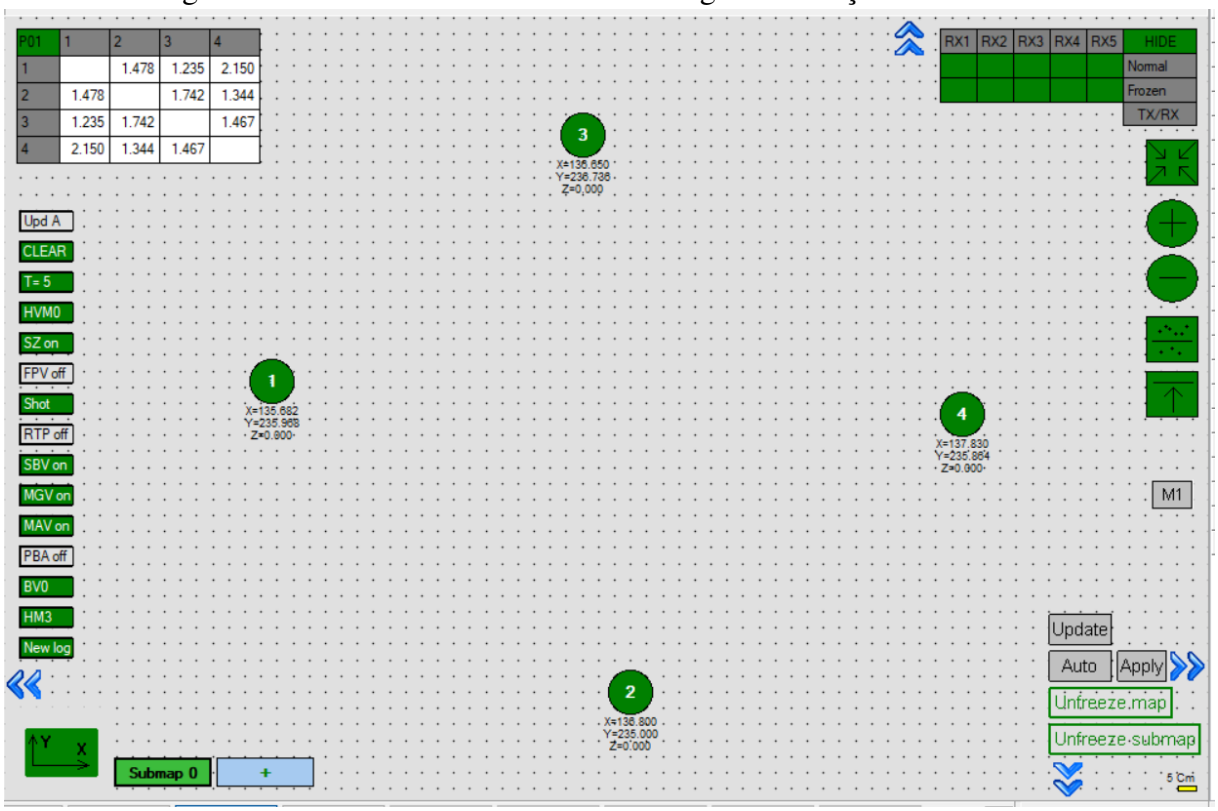
As antenas de geolocalização por sinais ultrassônicos apresentaram uma alta precisão no que tange a recepção de sinais e o envio de coordenadas para a máquina supervisora. Não houve falhas na comunicação, nem superposição de sinais na utilização de dois robôs em campo.

Sabendo que todos os movimentos realizados pelos robôs só serão possíveis se o supervisor enviar a trajetória de cada robô, foi-se necessário, portanto, iniciar os testes de comunicação entre as duas máquinas. Decidido que essa comunicação seria do tipo TCP/IP, um sinal estável de WLAN era preciso. Uma vez que esse sinal foi estabelecido, o processo de reconhecimento dos robôs pelo supervisor foi concluído com sucesso e essa conexão não apresentou problemas nos testes realizados.

Após receber um pedido, o robô terá que se deslocar da zona de recarga até um estoque, em seguida, ir até a zona de depósito e, por fim, voltar para a zona de recarga. Como já explicado nas seções precedentes, pontos de controle da trajetória foram definidos para obrigar os robôs a caminharem em uma espécie de pista dedicada a eles. Dessa forma, a malha de controle vai corrigindo o ângulo de deslocamento do robô conforme ele avança nos pontos de controle.

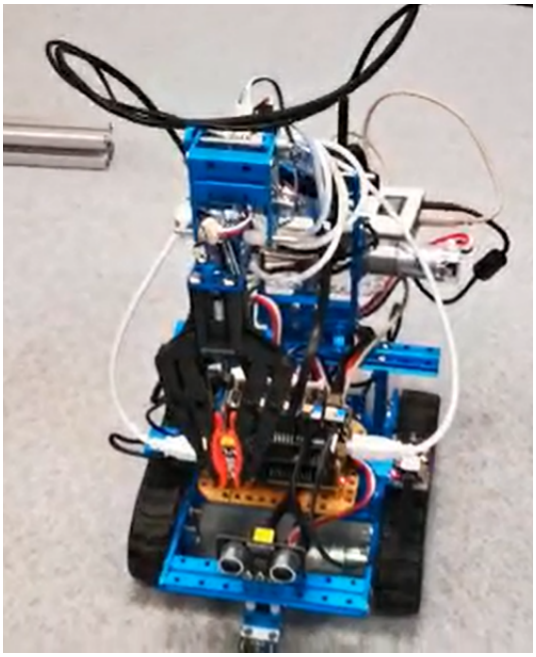
Os primeiros resultados obtidos foram os relacionados à malha de controle da velocidade e deslocamento do robô. Sabendo que a referência de entrada é o ângulo no qual o robô deve se posicionar para, em seguida, executar o movimento em linha reta e chegar à

Figura 16 – Conexão das antenas fixas de geolocalização



Fonte: própria autora.

Figura 17 – Montagem Robô Final

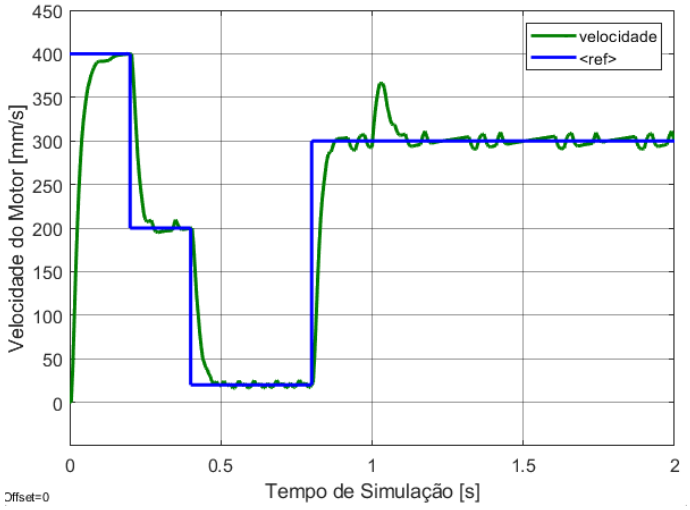


Fonte: própria autora.

posição desejada, este ângulo é analisado em forma de velocidade angular, esta é convertida em velocidade linear para, então, comparar com a velocidade de saída e tender o erro a zero.

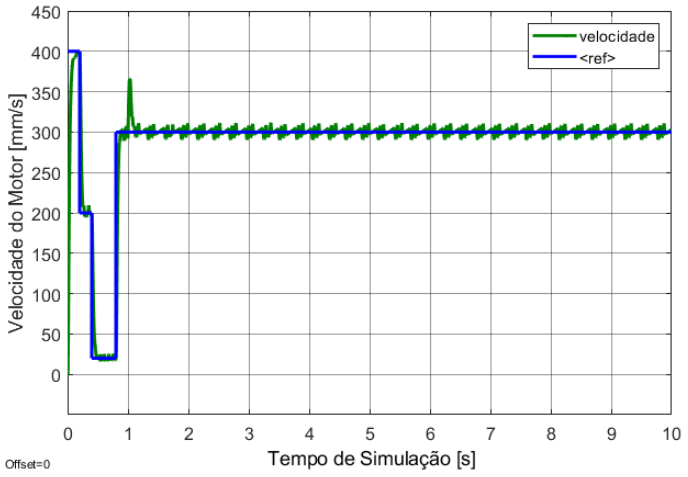
O esquemático do controle já foi ilustrado na figura 15, e na figura 18 é possível

Figura 18 – Velocidade do motor esquerdo (mm/s)



Fonte: própria autora.

Figura 19 – Velocidade do motor esquerdo (mm/s) em um horizonte 10 s



Fonte: própria autora.

ver os resultados de simulação das curvas da velocidade para um determinada referência, feitos no *Simulink*. Na figura é possível observar que a saída acompanha os diferentes valores de referência. As principais causas dos picos de velocidade em regime permanente são oriundas da não linearidade do motor, uma vez que esses tipos de motores embarcados não apresentam uma alta precisão. Entretanto, para essa aplicação, esses picos foram ignorados, uma vez que na realização de testes o movimento do robô não foi comprometido.

Se o horizonte de observação for aumentado, percebe-se que ao longo do movimento a velocidade acompanha a referência, a figura 19 ilustra isso. Vale ressaltar que as duas figuras que ilustram a velocidade correspondem às velocidades do motor esquerdo. O motor direito terá as mesmas formas de curva, de forma que a velocidade linear do robô será dada pela média aritmética das duas velocidades, como já visto anteriormente.

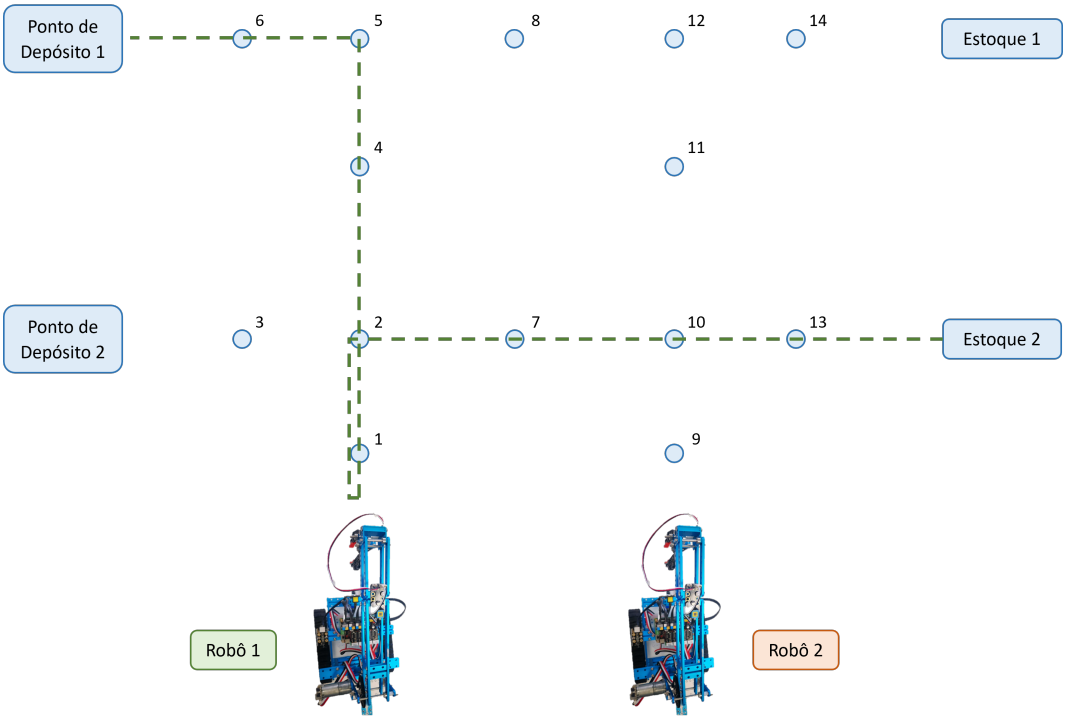
Figura 20 – Pedido do Robô 1 e sua trajetória.

```
Pedido: 1 artigo, deposito 1, estoque 2

Trajetoria: 1-2-4-5-6-D1-6-5-4-2-7-10-13-E2-13-10-7-2-1-R1
```

Fonte: própria autora.

Figura 21 – Trajetória Robô 1



Fonte: própria autora.

Os testes práticos, utilizando os robôs dentro da área reservada, foram realizados inicialmente com um só robô. Uma vez que o pedido foi feito, o máquina supervisora define o caminho mais curto para ser percorrido e verifica se este caminho está livre. Se estiver, ela envia o comando para um dos robôs e este realiza sua trajetória. A figura 20 mostra o caminho enviado para o robô 1 e a figura 21 ilustra o caminho realizado. Os resultados práticos do que ilustra essas figuras pode ser encontrado na referência (PARENTE, 2022).

O teste realizado com um só robô serviu para configurar bem alguns parâmetros de controle que em simulação não tinham sido considerados, bem como a definição de uma velocidade constante para o caso em que o ângulo de referência é maior que 180° de forma que o movimento possa ser corrigido ao longo da trajetória. Percebeu-se também que o controle desenvolvido permitiu um movimento linear e continuo do robô, deixando-o sempre nos limites da pista criada.

A comunicação e os testes mudam quando os dois robôs compartilham do mesmo terreno. O supervisor deve prevenir colisões e, para isso, o que se é feito é enviar a trajetória por partes. Se o robô 1, deve pegar um artigo no depósito 1 e entregá-lo no estoque 2, e o se

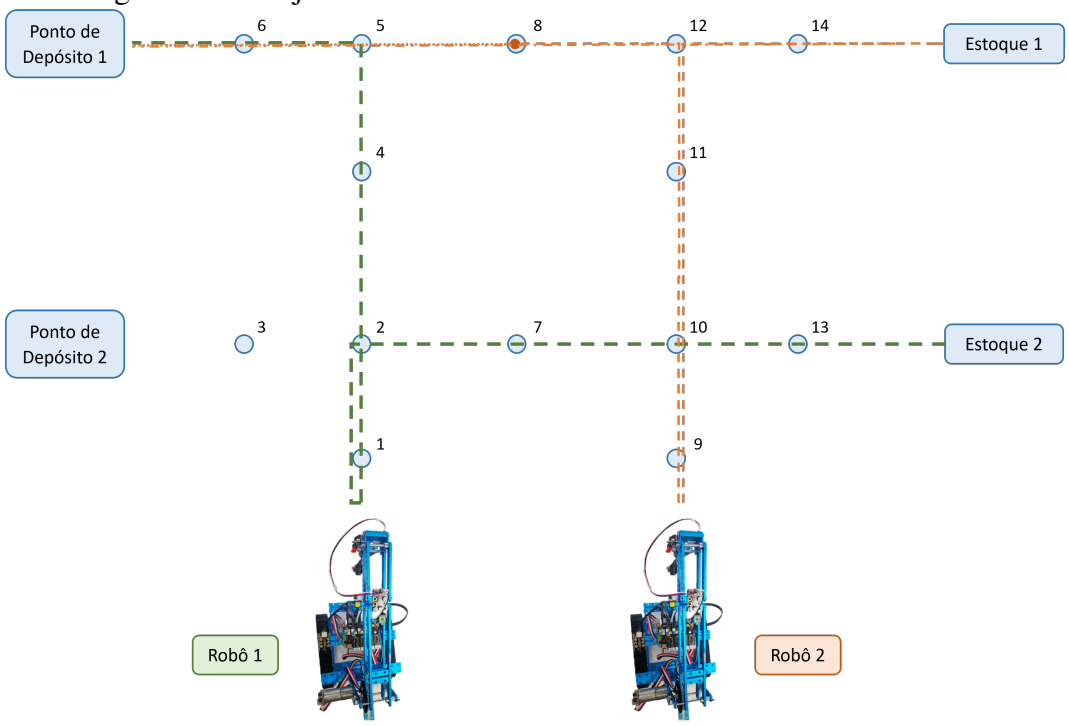
Figura 22 – Pedido do Robô 2 e sua trajetória.

```
Pedido: 1 artigo, deposito 2, estoque 1

Trajetoria: 9-10-11-12-8
Espera...
Espera...
Espera...
Trajetoria: 5-6-D1-6-5-8-12-14-E1-14-12-11-10-9-R2
```

Fonte: própria autora.

Figura 23 – Trajetória Robô 2.



Fonte: própria autora.

o robô 2 deve pegar um artigo no depósito 1 e entregá-lo no estoque 1, uma vez que o robô 1 é prioridade, porque recebeu o pedido primeiro, o robô 2 terá momentos de espera em alguns pontos de controle. A figura 22 ilustra como acontece a comunicação do supervisor para o robô 2.

A figura 23 ilustra a trajetória dos dois robôs, sabendo que o robô 2 faz uma pausa no seu movimento para esperar a disponibilidade dos pontos de controle 5 e 6, pelo robô 1. Uma vez que isso acontece, o robô 1 toma uma vantagem de tempo em relação ao robô 2, prevenindo uma possível colisão no ponto 10. É importante salientar que o robô 1 foi prioridade porque recebeu o pedido primeiro. Quando ele voltar a estar disponível para receber pedidos, o robô 2 será prioridade uma vez que esta realizando o seu movimento.

Os testes realizados com dois robôs apresentaram êxito no que tange a construção do movimento. Os motores apresentaram mais interferências que as previstas em simulação, fazendo com que o robô não executasse o caminho em linha reta com fluidez, ao longo de toda

a sua trajetória o seu movimento era corrigido. No entanto, os resultados corresponderam às expectativas do projeto. Os robôs executaram as tarefas programadas, sem colidir.

Já o supervisor, apresentou resultados positivos muito mais rapidamente que os robôs. O programa preveniu colisões, uma vez que foram definidas prioridades, e garantiu a execução do menor caminho com o algoritmo de Djikstra.

No que tange a comunicação dos robôs com a máquina supervisora, percebeu-se que esse sistema tem uma forte dependência à qualidade do sinal WLAN da sala. Nos momentos dos testes, houve instabilidade do sinal e, portanto, os robôs pararam de operar. Aplicando no cenário de um depósito logístico, em momentos de falha ou de manutenções os robôs não conseguiriam ser operados de forma ótima e seria necessário estabelecer medidas para prevenir isso.

5 CONCLUSÕES E TRABALHOS FUTUROS

Esse projeto foi desenvolvido com o intuito de ratificar os benefícios da automação no mercado logístico, no tange a eficiência e a continuidade dos seus processos, por meio da simulação de um cenário de *picking* dentro de um depósito automatizado com robôs. Uma vez que nem todos os materiais utilizados são de fácil acesso, o projeto contou com o apoio de materiais do laboratório de automação e controle da Ecole Centrale Lille.

Os resultados práticos corresponderam às expectativas iniciais do projeto e àquilo que foi estudado na fundamentação teórica. A escolha da utilização de sinais ultrassônicos para executar a geolocalização dos robôs foi pertinente, pois, diante do cenário de teste, era necessário uma considerada precisão em uma pequena área. Naturalmente que para a aplicação dentro deu um depósito logístico, a escolha mais adequada seria sinais de rádio ou o próprio WLAN.

Percebeu-se também que a comunicação entre máquinas apresentou estabilidade e continuidade na conexão. No que tange a construção e controle do caminho à ser percorrido pelos robôs, os resultados foram ótimos, os robôs executaram os movimentos esperados, porém com uma velocidade mais baixa do que era previsto, em virtude de interferências não previstas. Vale ressaltar, que essa perda em eficiência, não comprometeu a performance do movimento, nem do projeto.

A execução deste projeto permitiu ampliar e solidificar conhecimentos a respeito de embarcados, controle e automação, mais especificamente, conceitos de programação em tempo real, sistemas de comunicação distribuídas e técnicas de controle discreto.

Além disso, é importante salientar que esse projeto corrobora para o fomento de novas iniciativas de pesquisa, especialmente no que tange a automação de embarcados. Sugere-se, portanto, a título de trabalhos futuros, a integração de um sensor de obstáculos para afinar a prevenção de colisões e tornar o sistema menos dependente de uma máquina central, no caso o supervisor. Analisar o impacto dessa implementação no que tange a performance dos robôs, sua comunicação e se compromete ou não o supervisor.

Ademais, adicionar um braço robótico que realmente recolhe objetos do estoque, implementando uma leitura de *QR Code* do produto de forma que o robô possa atualizar o sistema de planejamento de estoque no momento da recuperação do artigo. Outra iniciativa de pesquisa seria a definição de um cenário de testes mais semelhantes à realidade, onde o embarcado esteja mais próximo de uma produto viável mínimo, onde ele teria as mesmas funcionalidades que são descritas neste trabalho, porém com um grau de execução mais realista.

Diante dos resultados obtidos, conhecimentos adquiridos e as iniciativas de pesquisas que podem ser aproveitadas com o projeto, este trabalho consegue mostrar o potencial da tecnologia de embarcados para estabelecer continuidade e garantir a precisão processos logísticos.

REFERÊNCIAS

CHEN, S.; WANG, F.-B.; DING, N. Application of improved dijkstra path planning algorithm for industrial stacking robots. In: **2022 International Seminar on Computer Science and Engineering Technology (SCSET)**. [S.l.: s.n.], 2022. p. 19–22.

CLINE, B.; HENRY, M.; JUSTICE, C. Rise of the robots. **KPMG**, p. 5–8, 2015.

CRESWELL, J. **Projeto de pesquisa: métodos qualitativo, quantitativo e misto;**: Tradução magda lopes. [S.l.]: Porto Alegre: ARTMED, 2010.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. HAL Open Science, 2021.

GETTING, I. The global positioning system. **IEEE Spectrum**, v. 30, n. 12, p. 36–47, 1993.

HAZAS, M.; HOPPER, A. Broadband ultrasonic location systems for improved indoor positioning. p. 536–548, 2006.

HOANG, N. M.; HA, H.; KIM, D. E.; HAN, S. I.; LEE, J.-M. An approach for controlling the communication signal flow in three-mobile robot system based on wireless communication. In: **2015 15th International Conference on Control, Automation and Systems (ICCAS)**. [S.l.: s.n.], 2015. p. 671–674.

IBM. Diagrammes de cas d'utilisation. IBM, 2021.

ISHIKAWA, M.; SAMPEI, M. State estimation of non-holonomic mobile robots using nonlinear observers. In: **Proceedings of 1995 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 1995. v. 2, p. 1379–1384 vol.2.

JABBAR, S.; KHAN, M.; SILVA B.N.AND HAN, K. A rest-based industrial web of things' framework for smart warehousing. **Supercomput**, v. 1, n. 1, p. 1–16, 2016.

KHAN, M.; KAI, Y. D.; GUL, H. U. Indoor wi-fi positioning algorithm based on combination of location fingerprint and unscented kalman filter. p. 693–698, 2017.

KHATOON, S.; CHATURVEDI, D. K.; HASAN, N.; ISTIYAQUE, M. Optimal controller design for two wheel mobile robot. In: **2018 3rd International Innovative Applications of Computational Intelligence on Power, Energy and Controls with their Impact on Humanity (CIPECH)**. [S.l.: s.n.], 2018. p. 1–5.

LEE, J.; BAGHERI, B.; KAO, H. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. **Manuf.**, v. 3, n. 1, p. 18–23, 2015.

LIU, H.; STOLL, N.; JUNGINGER, S.; THUROW, K. A common wireless remote control system for mobile robots in laboratory. In: **2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings**. [S.l.: s.n.], 2012. p. 688–693.

LUCIDCHART. **Qu'est-ce qu'un diagramme de séquence dans le langage UML ?** 2022. Disponível em: <https://www.lucidchart.com/pages/fr/diagramme-de-sequence-uml>. Acesso em: 11 jul. 2022.

MAKEBLOCK. **Datasheet Ultimate 2.0**. 2015. Disponível em: <https://www.makeblock.com/steam-kits/mbot-ultimate-2#Specifications>. Acesso em: 11 jul. 2022.

MAUTZ, D. R. **Indoor Positionning Technologies**: Habilitation thesis. [S.l.]: ETH: Zurich, 2012.

MCKINSEY, G. I. **Automation in logistics: Big opportunity, bigger uncertainty**. 2019. Disponível em: <https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/automation-in-logistics-big-opportunity-bigger-uncertainty>. Acesso em: 11 jul. 2022.

MORAIS, P. V. Leis de kirchhoff. **Unesp**, 2016.

NAWAWI, M. N. A. S. W.; OSMAN, J. H. S. Real-time control of a two-wheeled inverted pendulum mobile robot. In: **World Academy of Science, Engineering and Technology**. [S.l.: s.n.], 2017. p. 876–880.

NITULESCU, M. Theoretical aspects in wheeled mobile robot control. In: **2008 IEEE International Conference on Automation, Quality and Testing, Robotics**. [S.l.: s.n.], 2008. v. 2, p. 331–336.

OGATA, K. **Engenharia de Controle Moderno**. [S.l.]: Rio de Janeiro, 1982.

PARENTE, C. de G. **Video da Trajetória do Robô**. 2022. Disponível em: <https://drive.google.com/file/d/1IDqG-N_McvDH9LnHwkbdpOt-8nToTel6/view?usp=sharing>. Acesso em: Ecole Centrale Lille, setembro 2022.

ROBOTICS, M. **Marvelmind Indoor Navigation System**. 2021. Disponível em: <https://marvelmind.com/product/starter-set-hw-v4-9-nia/>. Acesso em: 11 jul. 2022.

SCHINDLER, D. C. **Business Research Methods**. [S.l.: s.n.], 2011.

T., V. **SysML - Diagramme des exigences**. [S.l.: s.n.], 2016.

ÖZBARAN, C.; DILIBAL, S.; SUNGUR, G. Mechatronic system design of a smart mobile warehouse robot for automated storage/retrieval systems. In: **2020 Innovations in Intelligent Systems and Applications Conference (ASYU)**. [S.l.: s.n.], 2020. p. 1–6.

APÊNDICE A – CODIGO DO SUPERVISOR 1

Código-fonte 1 – Codigo principal do supervisor em C

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <libgen.h>
6
7  #include "supervision.h"
8
9  #define SUPERVISOR_PORT 1234
10 #define SUPERVISOR_PORT_PING 1235
11
12 #define PING_TIMEOUT 6 // s
13
14 int main() {
15   pthread_t receiveMessageTid[2];
16   int se;
17   struct sockaddr_in addr;
18   pid_t pingPid;
19   pid_t supervisionPid;
20   union sigval sigValue;
21       char arg[2];
22   int pathLen = 500;
23   char supervisionPath[pathLen];
24   if (readlink ("/proc/self/exe", supervisionPath, pathLen)
       != -1) {
25     dirname(supervisionPath);
26     strcat(supervisionPath, "/supervision");
27   }
28
```

```
29  // Init orders gestion
30  orderPid = fork();
31  CHECK(orderPid, "fork()");
32  if (orderPid == 0) {
33    // Order Process
34    fifoInit(&orderFifo);
35    installSignalHandler(SIGNAL_ADD_ORDER,
        orderSignalHandler); // Add order signal
36    installSignalHandler(SIGNAL_GET_ORDER,
        orderSignalHandler); // Get order signal
37    while (pause());
38    exit(0);
39  }
40  printf("Processus de gestion des commandes lanc  avec le
      pid = %d\n", orderPid);
41
42  // Init ping Process
43  pingPid = fork();
44  CHECK(pingPid, "fork()");
45  if (pingPid == 0) {
46    // Order Process
47    pingProcess();
48    // Should not be here
49    exit(0);
50  }
51  //printf("Processus de ping lanc  avec le pid = %d\n",
      pingPid);
52
53  // Define Supervisor addr
54  addr.sin_family = AF_INET;
55  addr.sin_port = htons(SUPERVISOR_PORT);
56  addr.sin_addr.s_addr = INADDR_ANY;
```

```
57
58  // Setup listening socket
59  se = socket(AF_INET, SOCK_STREAM, 0);
60  CHECK(se, "--socket()");
61  CHECK(bind(se, (const struct sockaddr *) &addr,sizeof(
        addr)), "bind(1)");
62  listen(se, 1);
63
64  // Init communication with robots
65  #ifndef MODE_NO_ROBOT
66  for (long i = 0; i < 2; i++) {
67    sd[i] = accept(se, NULL, NULL);
68    printf("Connexion avec robot %ld\n", i);
69
70    sigValue.sival_int = i;
71    sigqueue(pingPid, SIGUSR1, sigValue);
72
73    // Creating thread
74    CHECK_T(
75        pthread_create(&receiveMessageTid[i], NULL, (void *
              (*) (void *)) receiveMessageThread, (void *) i)
            ,
76        "--pthread_create() receive message"
77    );
78
79  }
80  #endif
81
82  // Supervsision Process
83  /**supervisionPid = fork();
84  CHECK(supervisionPid, "fork()");
85  if (supervisionPid == 0) {
```

```
86      // Suppervision process
87      sprintf(arg, "%d", orderPid);
88      //CHECK(execl(supervisionPath, "supervision", arg, (
            char *) 0), "execl()");
89      supervision();
90      // Should not be here
91      exit(0);
92    } **/
93    //printf("Processus de supervision lanc  avec le pid = %
        d\n", supervisionPid);
94    supervision();
95
96    while(pause());
97  }
98
99  // Thread de reception de messages r seau
100 void * receiveMessageThread(long i) {
101   message_t receivedMessage;
102
103   printf("D but thread robot %ld\n", i);
104   while (1) {
105     if (waitForMessage(sd[i], &receivedMessage)) {
106       // Message received
107       //printf("Message recu !\n");
108       switch (receivedMessage.code) {
109         case CODE_POSITION:
110           printf("Get position of robot %ld : x = %d ; y =
                %d\n", i, receivedMessage.x, receivedMessage.y
                );
111           CHECK_T(pthread_mutex_lock(&robotsMutex[i]), "
                mutex_lock()");
112           robots[i].x = receivedMessage.x;
```

```
113             robots[i].y = receivedMessage.y;
114             CHECK_T(pthread_mutex_unlock(&robotsMutex[i]), "
                    mutex_unlock()");
115           break;
116           case CODE_ARRET_URGENCE:
117             printf("Arret urgence !\n");
118             // TODO
119           break;
120           case CODE_OK:
121             robots[i].ok = 1;
122           break;
123         }
124       }
125     }
126 }
127
128 // Ping Process
129 void pingProcess() {
130   int sdPing;
131   struct sockaddr_in addr;
132   struct sockaddr_in srcPingAddr;
133   socklen_t srcPingAddrSize;
134
135   message_ping_t pingMessage;
136
137   addr.sin_family = AF_INET;
138   addr.sin_port = htons(SUPERVISOR_PORT_PING);
139   addr.sin_addr.s_addr = INADDR_ANY;
140
141   clock_gettime(CLOCK_REALTIME, &lastPingTime[0]);
142   clock_gettime(CLOCK_REALTIME, &lastPingTime[1]);
143
```

```
144   // Setup listening socket
145   sdPing = socket(AF_INET, SOCK_DGRAM, 0);
146   CHECK(sdPing, "--socket()");
147   CHECK(bind(sdPing, (const struct sockaddr *) &addr,sizeof
          (addr)), "bind()");
148
149   installSignalHandler(SIGALRM, pingSignalHandler);
150   installSignalHandler(SIGUSR1, pingSignalHandler);
151   alarm(PING_TIMEOUT);
152
153   while (1) {
154     srcPingAddrSize = sizeof(srcPingAddr);
155     if (recvfrom(sdPing, &pingMessage, sizeof(
            message_ping_t), 0, (struct sockaddr *) &srcPingAddr
            , &srcPingAddrSize) == sizeof(message_ping_t)) {
156       //printf("Ping re u du Robot %d\n", pingMessage.
              robotId);
157
158       if (pingMessage.robotId < 2) {
159         clock_gettime(CLOCK_REALTIME, &lastPingTime[
                pingMessage.robotId]);
160       }
161       sendto(sdPing, &pingMessage, sizeof(pingMessage), 0,
              (const struct sockaddr *) &srcPingAddr,
              srcPingAddrSize);
162     }
163   }
164
165   pthread_exit(NULL);
166 }
167
168
```

```
169 // Signal Handler
170 void orderSignalHandler(int sigNum, siginfo_t * sigInfo,
       void * null) {
171   commande_t * commande = NULL;
172   union sigval signalValue;
173   switch(sigNum) {
174     case SIGNAL_ADD_ORDER:
175       // Add a new order
176       // Format : AABB
177       commande = malloc(sizeof(commande_t));
178       commande->a = sigInfo->si_int / 100;
179       commande->b = sigInfo->si_int - 100*commande->a;
180       //printf("Ajout d'une commande dans la liste : A = %d
              B = %d\n", commande->a, commande->b);
181       if (fifoAdd(&orderFifo,(void *) commande) == -1) {
182         printf("Impossible d'enregistr  la commande : file
              d'attente remplie\n");
183       }
184     break;
185     case SIGNAL_GET_ORDER:
186       commande = (commande_t *) fifoGet(&orderFifo);
187       if (commande == NULL) {
188         //printf("Pas de commande en attente\n");
189       } else {
190         //printf("R cup ration de commande : A = %d B= %d
              \n", commande->a, commande->b);
191         signalValue.sival_int = commande->a *100 + commande
              ->b;
192         sigqueue(sigInfo->si_pid, SIGNAL_GET_ORDER,
              signalValue);
193
194       }
```

```
195
196         free(commande);
197       break;
198     }
199 }
200
201 // Ping signal handler
202 void pingSignalHandler(int sigNum, siginfo_t * sigInfo,
      void * null) {
203   struct timespec time;
204   switch (sigNum) {
205     case SIGALRM:
206       // Check all last pings
207       clock_gettime(CLOCK_REALTIME, &time);
208       if (connectedRobot[0] && time.tv_sec - lastPingTime
          [0].tv_sec > PING_TIMEOUT) {
209         // Lost connexion with Robot 0
210         printf("Perte de connexion avec le Robot 0\n");
211         // TODO - traiter la perte de connexion
212       }
213       if (connectedRobot[1] && time.tv_sec - lastPingTime
          [1].tv_sec > PING_TIMEOUT) {
214         // Lost connexion with Robot 0
215         printf("Perte de connexion avec le Robot 1\n");
216         // TODO - traiter la perte de connexion
217       }
218       alarm(PING_TIMEOUT);
219     break;
220     case SIGUSR1:
221       // Robot connected
222       // Initialise le traitement du ping uniquement
          lorsque le robot est connect
```

```
223         connectedRobot[sigInfo->si_int] = 1;
224       break;
225     }
226 }
```

APÊNDICE B – PROGRAMA DO SUPERVISOR 2

Código-fonte 2 – Função de inicialização do ambiente de simulação

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  #include "../include/communication.h"
6  #include "../include/check.h"
7
8  #define SUPERVISOR_PORT 1234
9  #define NB_SOMMETS 24
10
11 int main() {
12   FILE * fd;
13
14   int sd, se;
15   struct sockaddr_in addr;
16
17   int i = 0;
18   int n;
19   char * res = malloc(sizeof(char)*2000);
20
21   printf("Connexion ...\n");
22
23   addr.sin_family = AF_INET;
24   addr.sin_port = htons(SUPERVISOR_PORT);
25   addr.sin_addr.s_addr = INADDR_ANY;
26
27   // Setup listening socket
28   se = socket(AF_INET, SOCK_STREAM, 0);
29   CHECK(se, "--socket()");
```

```
30    CHECK(bind(se, (const struct sockaddr *) &addr,sizeof(
          addr)), "bind(1)");
31    listen(se, 1);
32
33    sd = accept(se, NULL, NULL);
34    printf("Connect  avec le robot\n");
35
36    printf("Attente de message\n");
37    n = recv(sd, res, sizeof(char)*2000, 0);
38
39    //fprintf(stderr, "%s", res);
40    fd = fopen("superviseur/graphData.txt", "w");
41    fputs(res, fd);
42
43    printf("Data printed in superviseur/graphData.txt\n");
44
45    fclose(fd);
46
47    free(res);
48 }
```

APÊNDICE C – PROGRAMA DO SUPERVISOR 3

Código-fonte 3 – Chamadas de funções e bibliotecas do supervisor

```
1  #include <unistd.h>
2  #include <pthread.h>
3
4  #include "../include/types.h"
5  #include "../include/check.h"
6  #include "../include/fifo.h"
7  #include "../include/signals.h"
8  #include "../include/matrix.h"
9  #include "../include/dijkstra.h"
10 #include "../include/graph.h"
11 #include "../include/communication.h"
12
13 void * receiveMessageThread(long i);
14 void orderSignalHandler(int sigNum, siginfo_t * sigInfo,
      void * null);
15 void pingProcess();
16 void pingSignalHandler(int sigNum, siginfo_t * sigInfo,
      void * null);
17
18 matrix_t initGraph();
19 void initSommets(graph_sommet_t * sommets);
20 void takeOrderSignalHandler(int sigNum, siginfo_t * sigInfo
      , void * null);
21 void * robotThread(long i);
22
23 void lockSegmentGraph(int idRobot, int segmentId);
24 void unlockSegmentGraph(int idRobot, int segmentId);
25 int reserveEtape(int idRobot, int croisementId, int
      segmentId);
```

```
26 void sectionTrajet(int idRobot, int idDest);
27 void trajet(int idRobot,int stock, int depot);
28 void mooveRobot(int idRobot, int dest);
29 void supervision();
30
31 int sd[2];
32 int connectedRobot[2];
33
34 fifo_t orderFifo;
35 pid_t orderPid;
36
37 struct timespec lastPingTime[2];
38 commande_t currentOrder[2];
39 pid_t orderPid;
40 union sigval signalValue;
41 robot_t robots[2];
42 matrix_t graph[2];
43 pthread_mutex_t robotsMutex[2];
44 graph_sommet_t sommets[NB_SOMMETS];
45 pthread_mutex_t graphMutex[2];
46 pthread_mutex_t reservationMutex;
47 pthread_mutex_t segmentsMutex[NB_SEGMENTS];
48 pthread_mutex_t croisementsMutex[NB_CROISEMENTS];
49 int loadingStation[2];
```

APÊNDICE D – PROGRAMA DO SUPERVISOR 4

Código-fonte 4 – Supervisão das trajetorias dos robôs

```
1  #include "supervision.h"
2
3  //int main(int argc, char * argv[]) {
4  void supervision() {
5    pthread_t robotTid[2];
6
7    /**
8    if (argc != 2) {
9      printf("Usage : %s <pid_order_process>\n", argv[0]);
10   }
11   orderPid = atoi(argv[1]);
12   **/
13   currentOrder[0].a = 0;
14   currentOrder[0].b = 0;
15   currentOrder[1].a = 0;
16   currentOrder[1].b = 0;
17
18   // Init complete graph
19   graph[0] = initGraph();
20   graph[1] = initGraph();
21   //printMat(&graph[0]);
22   initSommets(&sommets[0]);
23   printf("Supervision\n");
24   // Init all mutex
25   CHECK_T(pthread_mutex_init(&reservationMutex , NULL), "
       pthread_mutex_init()");
26   CHECK_T(pthread_mutex_init(&graphMutex[0] , NULL), "
       pthread_mutex_init()");
```

```
27  CHECK_T(pthread_mutex_init(&graphMutex[1] , NULL), "
       pthread_mutex_init()");
28  for (int i = 0; i < NB_SEGMENTS; i++)
29    CHECK_T(pthread_mutex_init(&segmentsMutex[i] , NULL), "
         pthread_mutex_init()");
30  for (int i = 0; i < NB_CROISEMENTS; i++)
31    CHECK_T(pthread_mutex_init(&croisementsMutex[i] , NULL)
       , " pthread_mutex_init()");
32
33  // Init robots
34  robots[0].id = 0;
35  robots[1].id = 1;
36  CHECK_T(pthread_mutex_init(&robotsMutex[0], NULL), "
       mutex_init()");
37  CHECK_T(pthread_mutex_init(&robotsMutex[1], NULL), "
       mutex_init()");
38  loadingStation[0] = SOMMET_RECHARGE_0;
39  loadingStation[1] = SOMMET_RECHARGE_1;
40  robots[0].sommetId = loadingStation[0]; // TODO position
       initiale des robots
41  robots[1].sommetId = loadingStation[1];
42  //printf("sommet robot 0 : %d\n", robots[0].sommetId);
43  //printf("Segment robot 0 : %d\n", getSegmentId(robots
       [0].sommetId));
44  //printf("sommet robot 1 : %d\n", robots[1].sommetId);
45  //printf("Segment robot 1 : %d\n", getSegmentId(robots
       [1].sommetId));
46  //CHECK_T(pthread_mutex_lock(&segmentsMutex[getSegmentId(
       robots[0].sommetId)]), "mutex_lock()");
47  //CHECK_T(pthread_mutex_lock(&segmentsMutex[getSegmentId(
       robots[1].sommetId)]), "mutex_lock()");
48
```

```
49    // Get order signals
50    installSignalHandler(SIGALRM, takeOrderSignalHandler);
51    installSignalHandler(SIGNAL_GET_ORDER,
          takeOrderSignalHandler);
52    alarm(2);
53
54    // Thread de supervision de trajectoires
55    CHECK_T(
56        pthread_create(&robotTid[0], NULL, (void * (*) (void
              *)) robotThread, (void *) 0),
57        "--pthread_create()"
58    );
59    CHECK_T(
60        pthread_create(&robotTid[1], NULL, (void * (*) (void
              *)) robotThread, (void *) 1),
61        "--pthread_create()"
62    );
63
64    while (pause());
65
66    //return 0;
67  }
68
69  // Enl ve un segment du graph si il est au centre
70  void lockSegmentGraph(int idRobot, int segmentId) {
71    int i, j;
72    CHECK_T(pthread_mutex_lock(&graphMutex[idRobot]), "
          mutex_lock()");
73    getSommetsSegmentCentral(segmentId, &i, &j);
74    if (i != -1) {
75      setMatrix(&graph[idRobot], i, j, INFINITY);
76      setMatrix(&graph[idRobot], j, i, INFINITY);
```

```
77    }
78    CHECK_T(pthread_mutex_unlock(&graphMutex[idRobot]), "
          mutex_lock()");
79  }
80
81  // Rajoute le segment au graph si il est au centre
82  void unlockSegmentGraph(int idRobot, int segmentId) {
83    int i, j;
84    CHECK_T(pthread_mutex_lock(&graphMutex[idRobot]), "
          mutex_lock()");
85    getSommetsSegmentCentral(segmentId, &i, &j);
86    if (i != -1) {
87      setMatrix(&graph[idRobot], i, j, 1);
88      setMatrix(&graph[idRobot], j, i, 1);
89    }
90    CHECK_T(pthread_mutex_unlock(&graphMutex[idRobot]), "
          mutex_lock()");
91  }
92
93  // Essaie de reserver un croisement et le segement suivant
94  // Renvoi 1 si OK
95  // Renvoi 0 sinon
96  int reserveEtape(int idRobot, int croisementId, int
       segmentId) {
97    CHECK_T(pthread_mutex_lock(&reservationMutex), "
          mutex_lock()");
98    // Essaie de reserver le croisement
99    if (pthread_mutex_trylock(&croisementsMutex[croisementId
          ]) == 0) {
100     // Essaie de r server le prochian segement
101     if (pthread_mutex_trylock(&segmentsMutex[segmentId]) ==
            0) {
```

```
102         // Croisement et segement reserv s
103         CHECK_T(pthread_mutex_unlock(&reservationMutex), "
                mutex_unlock()");
104         //printf("\033[0;31m\tC%d\n\tS%d\n\033[0m",
                croisementId, segmentId);
105         lockSegmentGraph(idRobot == 0 ? 1 : 0, segmentId);
106         return 1;
107       } else {
108         // Le segement ne peux pas  tre  r serv
109         // On lib re le croisement
110         CHECK_T(pthread_mutex_unlock(&croisementsMutex[
                croisementId]), "mutex_unlock()");
111       }
112     }
113
114     // Arrive ici que si on n'as pas pu tout r server
115     CHECK_T(pthread_mutex_unlock(&reservationMutex), "
          mutex_unlock()");
116     return 0;
117 }
118
119 // Supervise le trajet d'un robot de sa position au d pot ,
        ou au stock ou    une station de recharge
120 void sectionTrajet(int idRobot, int idDest) {
121     int * chemin;
122     int sizeChemin;
123     int croisementId;
124     int segmentId;
125     int ancienSegmentId;
126
127     do {
```

```
128     CHECK_T(pthread_mutex_lock(&graphMutex[idRobot]), "
           mutex_lock()");
129     chemin = dijkstra(graph[idRobot], robots[idRobot].
           sommetId, idDest, &sizeChemin);
130     CHECK_T(pthread_mutex_unlock(&graphMutex[idRobot]), "
           mutex_lock()");
131     // Si on arrive    la fin de l' tape
132     if (sizeChemin == 0) {
133       // Chemin bloqu  pour le moment
134       usleep(500000);
135     }
136     else if (sizeChemin == 1) {
137       // GO chemin[0]
138       mooveRobot(idRobot, chemin[0]);
139     }
140     // Si la prochaine  tape  est un croisement :
141     else if (sommets[chemin[0]].type == CROSS) {
142       // On r cup re l'identifiant du croisement et du
             prochain segment
143       croisementId = getCroisementId(chemin[0]);
144       segmentId = getSegmentId(chemin[1]);
145       CHECK(croisementId, "getCroisementId()");
146       CHECK(segmentId, "getSegmentId()");
147
148       // On essaie de reserver le croisement et le prochain
             segement
149       if (reserveEtape(idRobot, croisementId, segmentId)) {
150         //On a r ussi    reserver
151         ancienSegmentId = getSegmentId(robots[idRobot].
             sommetId);
152         CHECK(ancienSegmentId, "getSegmentId()");
153
```

```
154         // GO chemin[0]
155         mooveRobot(idRobot, chemin[0]);
156         // On peut lib rer l'ancien segment
157         CHECK_T(pthread_mutex_unlock(&segmentsMutex[
               ancienSegmentId]), "mutex_unlock()");
158         //printf("\033[0;32m\tS%d\n\033[0m",
               ancienSegmentId);
159         unlockSegmentGraph(idRobot == 0 ? 1 : 0,
               ancienSegmentId);
160         // GO chemin[1]
161         mooveRobot(idRobot, chemin[1]);
162         // On peut lib rer le croisement
163         CHECK_T(pthread_mutex_unlock(&croisementsMutex[
               croisementId]), "mutex_unlock()");
164         //printf("\033[0;32m\tC%d\n\033[0m", croisementId);
165       } else {
166         // On a pas r ussi    reserver
167         // On attend un peu avant de recalucler une
               nouvelle trajectoire si n cessaire et de
               r essayer
168         usleep(500000);
169       }
170     }
171     // Si la prochaine  tape  n'est ni la fin ni un
          croisement
172     else {
173       // GO chemin[0]
174       mooveRobot(idRobot, chemin[0]);
175     }
176     free(chemin);
177
178   } while(sizeChemin != 1);
```

```
179
180   CHECK_T(pthread_mutex_lock(&graphMutex[idRobot == 0 ? 1 :
          0]), "mutex_lock()");
181   freeMatrix(&graph[idRobot == 0 ? 1 : 0]);
182   graph[idRobot == 0 ? 1 : 0] = initGraph();
183   CHECK_T(pthread_mutex_unlock(&graphMutex[idRobot == 0 ? 1
          : 0]), "mutex_lock()");
184 }
185
186 // Ordonne au robot de se d placer
187 void mooveRobot(int idRobot, int dest) {
188   #ifndef MODE_NO_ROBOT
189     message_t message;
190     int x, y;
191     int dx, dy;
192
193     message.code = CODE_MOOVE;
194     message.x = sommets[dest].x;
195     message.y = sommets[dest].y;
196
197     printf("Le robot %d doit aller au point %d\n", idRobot,
            dest);
198     sendMessage(sd[idRobot], &message);
199
200     do {
201       usleep(50000);
202       CHECK_T(pthread_mutex_lock(&robotsMutex[idRobot]), "
            mutex_lock()");
203       x = robots[idRobot].x;
204       y = robots[idRobot].y;
205       CHECK_T(pthread_mutex_unlock(&robotsMutex[idRobot]),
            "mutex_unlock()");
```

```
206       dx = x - sommets[dest].x;
207       dy = y - sommets[dest].y;
208
209     } while(dx > POSITION_PRECISION || dx < -
            POSITION_PRECISION || dy > POSITION_PRECISION || dy
            < -POSITION_PRECISION);
210     printf("Le robot %d est arriv  au point %d\n", idRobot
            , dest);
211
212     robots[idRobot].sommetId = dest;
213
214     /**if (sommets[idRobot].type == STOCK) {
215       robots[idRobot].ok = 0;
216       message.code = CODE_TAKE_STOCK;
217       sendMessage(sd[idRobot], &message);
218       while (!robots[idRobot].ok) sleep(1);
219     }
220     else if (sommets[idRobot].type == OUT) {
221       robots[idRobot].ok = 0;
222       message.code = CODE_LET_OUT;
223       sendMessage(sd[idRobot], &message);
224       while (!robots[idRobot].ok) sleep(1);
225     }**/
226   #else
227     printf("Le robot %d va au point %d\n", idRobot, dest);
228     sleep(1);
229     robots[idRobot].sommetId = dest;
230     if (sommets[dest].type == STOCK || sommets[dest].type
            == OUT || sommets[dest].type == LOADING_STATION) {
231       printf("Le robot %d %s\n", idRobot,
232         sommets[dest].type == STOCK ? "prend dans le stock"
              : (sommets[dest].type == OUT ? "d pose le
```

```
            stock" : " se recharge")
233       );
234       sleep(1);
235     }
236   #endif
237 }
238
239
240 void trajet(int idRobot,int stock, int depot) {
241   // Go stock
242   sectionTrajet(idRobot, stock);
243   // Go d pot
244   sectionTrajet(idRobot, depot);
245 }
246
247 void * robotThread(long i) {
248   //printf("Robot %ld\n",i);
249   while(1){
250     if (i == 0 && currentOrder[i].a !=0) {
251       // Dijkstra de la position du robot 0 au stock a
252       // Supervision du d placement
253       // Dijkstra du stock a au d pot 0
254       // Supervision du d placement
255       trajet(i, SOMMET_STOCK_A,SOMMET_DEPOT_0);
256       currentOrder[i].a--;
257     }
258     else if (i == 1 && currentOrder[i].b !=0) {
259       // Dijkstra de la position du robot 1 au stock B
260       // Supervision du d placement
261       // Dijkstra du stock a au d pot 1
262       // Supervision du d placement
263       trajet(i, SOMMET_STOCK_B,SOMMET_DEPOT_1);
```

```
264         currentOrder[i].b--;
265       }
266       else if (i == 0 && currentOrder[i].b !=0) {
267         // Dijkstra de la position du robot 0 au stock B
268         // Supervision du d placement
269         // Dijkstra du stock a au d pot 0
270         // Supervision du d placement
271         trajet(i, SOMMET_STOCK_B,SOMMET_DEPOT_0);
272         currentOrder[i].b--;
273       }
274       else if (i == 1 && currentOrder[i].a !=0) {
275         // Dijkstra de la position du robot 1 au stock A
276         // Supervision du d placement
277         // Dijkstra du stock a au d pot 1
278         // Supervision du d placement
279         trajet(i, SOMMET_STOCK_A,SOMMET_DEPOT_1);
280         currentOrder[i].a--;
281       }
282       else if (robots[i].sommetId != loadingStation[i] &&
             currentOrder[i].a == 0 && currentOrder[i].b == 0) {
283         sectionTrajet(i, loadingStation[i]);
284       }
285     }
286     pthread_exit(NULL);
287 }
288
289 // Signal Handler
290 void takeOrderSignalHandler(int sigNum, siginfo_t * sigInfo
      , void * null) {
291     commande_t * commande = NULL;
292     switch(sigNum) {
293       case SIGALRM:
```

```
294       // Si on a la capacit  de traiter une commande en
              plus, on interroge le processus de gestion des
              commandes afin de r cup rer une commande dans la
               liste en attente
295       if ((currentOrder[0].a == 0 && currentOrder[0].b ==
              0) || (currentOrder[1].a == 0 && currentOrder[1].b
               == 0)) {
296         // Get new order
297         signalValue.sival_int = 0;
298         sigqueue((pid_t) orderPid, SIGNAL_GET_ORDER,
                signalValue);
299       }
300       alarm(2);
301     break;
302     case SIGNAL_GET_ORDER:
303       // Le processus de gestion des commandes r pond avec
              une commande
304       if (currentOrder[0].a == 0 && currentOrder[0].b == 0)
              {
305         currentOrder[0].a = sigInfo->si_int / 100;
306         currentOrder[0].b = sigInfo->si_int - 100*
                currentOrder[0].a;
307         printf("Nouvelle commande 0 : A = %d B = %d\n",
                currentOrder[0].a, currentOrder[0].b);
308       } else if (currentOrder[1].a == 0 && currentOrder[1].
              b == 0) {
309         currentOrder[1].a = sigInfo->si_int / 100;
310         currentOrder[1].b = sigInfo->si_int - 100*
                currentOrder[1].a;
311         printf("Nouvelle commande 1 : A = %d B = %d\n",
                currentOrder[1].a, currentOrder[1].b);
312       }
```

```
313         break;
314       }
315   }
```

APÊNDICE E – PROGRAMA DO ROBÔ

Código-fonte 5 – Programa completo do robô

```
#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/types.h>
#include <math.h>

#include "../include/communication.h"
#include "../include/types.h"
#include "../include/check.h"
#include "../include/marvelmind.h"

#define SUPERVISOR_IP "127.0.0.1"
#define SUPERVISOR_PORT 1234
#define SUPERVISOR_PORT_PING 1235

#define PING_TIMEOUT 6 // s
#define PING_PERIOD 4 // s
```

```
27
28 #define SPEED_LEFT  "$S40,-40\n"
29 #define SPEED_RIGHT "$S-40,40\n"
30 #define SPEED_FORWARD "$S50,50\n"
31 #define SPEED_BACKWARD  "$S-50,-50\n"
32
33 #define PRECISION_ANGLE 0.5
34
35 void pingProcess();
36 void * serialReadThread();
37 void serialReadData();
38 void * recevMessageThread();
39 void mooveRobot(int x, int y);
40 void * marvelMindThread();
41 int open_port(char* port_name);
42 void SetupSerialOptions(int fd);
43 void Stop();
44 void Forward();
45 void Backward();
46 void Left();
47 void Right();
48 void init();
49
50 int sd;
51 int fd;
52 arduino_data_t arduinoData;
53 pthread_mutex_t arduinoDataMutex;
54 struct MarvelmindHedge * marvelMindHedge;
55 struct PositionValue marvelMindPosition;
56 pthread_mutex_t marvelMindMutex;
57 float angle0;
58 float gyro0;
```

```
59 char * supervisorIp;
60
61 int main(int argc, char * argv[]) {
62   struct sockaddr_in serverAddr;
63   message_t message;
64   pid_t pingPid;
65   pthread_t serialReadTid;
66   pthread_t recevMessageTid;
67   pthread_t marvelMindTid;
68
69   if (argc == 1) {
70     supervisorIp = SUPERVISOR_IP;
71     #ifndef MODE_NO_ROBOT
72     fd = open_port("/dev/ttyUSB0");
73     #endif
74   } else if (argc == 2) {
75     supervisorIp = argv[1];
76     #ifndef MODE_NO_ROBOT
77     fd = open_port("/dev/ttyUSB0");
78     #endif
79   } else {
80     supervisorIp = argv[1];
81     #ifndef MODE_NO_ROBOT
82     if (argc == 3) fd = open_port(argv[2]);
83     #endif
84   }
85
86   #ifndef MODE_NO_ROBOT
87   CHECK(fd, "open_port()");
88   SetupSerialOptions(fd);
89   #endif
90
```

```
91  // Marvel Mind
92  marvelMindHedge = createMarvelmindHedge();
93  startMarvelmindHedge(marvelMindHedge);
94  CHECK_T(pthread_mutex_init(&marvelMindMutex , NULL), "
       pthread_mutex_init()");
95
96  // Arduino Data
97  CHECK_T(pthread_mutex_init(&arduinoDataMutex , NULL), "
       pthread_mutex_init()");
98
99  // Connexion with supervisor
100 printf("Connexion ...\n");
101 clientConnect(&sd, &serverAddr, supervisorIp,
       SUPERVISOR_PORT);
102 printf("Connect  avec le superviseur\n");
103
104 // Init ping process
105 pingPid = fork();
106 CHECK(pingPid, "fork()");
107 if (pingPid == 0) {
108   // Ping Process
109   pingProcess();
110   // Should not be here
111   exit(0);
112 }
113
114 // Start all threads
115 CHECK_T(
116     pthread_create(&serialReadTid, NULL, (void * (*) (
          void *)) serialReadThread, NULL),
117     "--pthread_create()"
118 );
```

```
119
120   CHECK_T(
121       pthread_create(&marvelMindTid, NULL, (void * (*) (
              void *)) marvelMindThread, NULL),
122       "--pthread_create()"
123   );
124
125   // Initialize robot position
126   #ifndef MODE_NO_ROBOT
127   init();
128   #endif
129
130   // Receive message thread
131   CHECK_T(
132       pthread_create(&recevMessageTid, NULL, (void * (*) (
              void *)) recevMessageThread, NULL),
133       "--pthread_create()"
134   );
135
136   //mooveRobot(-154,290);
137
138   while(pause());
139 }
140
141 // Serial reading thread
142 void * serialReadThread() {
143   while (1) {
144     serialReadData();
145   }
146   pthread_exit(NULL);
147 }
148
```

```
149 // Function to read and parse data in serial
150 void serialReadData() {
151   char data[101];
152   char * p;
153 
154   int n = read(fd, &data, 100);
155   data[n] = '\0';
156   if (n != 0) {
157     if (data[0] == '$') {
158       CHECK_T(pthread_mutex_lock(&arduinoDataMutex), "
            mutex_lock()");
159       //printf("%s", data);
160       p = strtok(data, ",");
161       arduinoData.speedR = atoi(p + 1);
162       //printf("Speed R : %d\n", arduinoData.speedR);
163 
164       p = strtok(NULL, ",");
165       arduinoData.speedL = atoi(p);
166       //printf("Speed L : %d\n", arduinoData.speedL);
167 
168       p = strtok(NULL, ",");
169       arduinoData.gyroZ = 360-(float) atoi(p)/100;
170       //printf("Gyro Z : %d\n", arduinoData.gyroZ);
171 
172       p = strtok(NULL, ",");
173       arduinoData.ultraSound = atoi(p);
174       //printf("UltraSound: %d\n", arduinoData.ultraSound);
175       CHECK_T(pthread_mutex_unlock(&arduinoDataMutex), "
            mutex_unlock()");
176 
177     }
178   }
```

```
179 }
180
181 // Receive message thread
182 void * recevMessageThread() {
183   message_t receivedMessage;
184   message_t messageToSend;
185   while (1) {
186     if (waitForMessage(sd, &receivedMessage)) {
187       // Message received
188       switch (receivedMessage.code) {
189         case CODE_MOOVE:
190           printf("Message recu : Move x = %d; y = %d !\n",
                receivedMessage.x, receivedMessage.y);
191           #ifndef MODE_NO_ROBOT
192           mooveRobot(receivedMessage.x, receivedMessage.y);
193
194           messageToSend.code = CODE_POSITION;
195
196           CHECK_T(pthread_mutex_lock(&marvelMindMutex), "
                mutex_lock()");
197           messageToSend.x = marvelMindPosition.x;
198           messageToSend.y = marvelMindPosition.y;
199           CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
                mutex_unlock()");
200           #else
201             messageToSend.code = CODE_POSITION;
202             messageToSend.x = receivedMessage.x;
203             messageToSend.y = receivedMessage.y;
204             sleep(1);
205           #endif
206           sendMessage(sd, &messageToSend);
207
```

```
208             break;
209           case CODE_GET_POSITION:
210             printf("Message recu : Get Position \n");
211             // TODO
212             break;
213           case CODE_TAKE_STOCK:
214             printf("Message recu : Prend dans le stock\n");
215             sleep(2);
216             messageToSend.code = CODE_OK;
217             sendMessage(sd, &messageToSend);
218             break;
219           case CODE_LET_OUT:
220             printf("Message recu : Prend d pose le stock\n")
                  ;
221             sleep(2);
222             messageToSend.code = CODE_OK;
223             sendMessage(sd, &messageToSend);
224             break;
225         }
226
227       }
228     }
229
230     pthread_exit(NULL);
231 }
232
233 void mooveRobot(int x1, int y1) {
234     int x0, y0, x, y;
235     float theta_xy;
236     float theta;
237     float gyro;
238     float gyro_xy;
```

```
239
240   CHECK_T(pthread_mutex_lock(&marvelMindMutex), "mutex_lock
          ()");
241   x0 = marvelMindPosition.x;
242   y0 = marvelMindPosition.y;
243   CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
          mutex_unlock()");
244
245   theta_xy = atanf((float) (y1-y0)/(x1-x0))*180/3.1415;
246   if (x1-x0 < 0) theta_xy = 180 + theta_xy;
247
248   //theta_xy = gyro_xy - gyro0 + angle0;
249   gyro_xy = theta_xy + gyro0 - angle0;
250
251   do {
252     CHECK_T(pthread_mutex_lock(&arduinoDataMutex), "
            mutex_lock()");
253     gyro = arduinoData.gyroZ;
254     CHECK_T(pthread_mutex_unlock(&arduinoDataMutex), "
            mutex_unlock()");
255     if (gyro - gyro_xy > 0) Left();
256     else Right();
257   } while(gyro - gyro_xy > PRECISION_ANGLE || gyro -gyro_xy
        < - PRECISION_ANGLE);
258
259   //theta_gyro = 180-(theta_xy + gyro0 + angle0);
260   //printf("Theta_xy = %f theta_gyro = %f\n", theta_xy,
          theta_gyro);
261
262   // Marche
263   /**
264   do {
```

```
265     CHECK_T(pthread_mutex_lock(&arduinoDataMutex), "
           mutex_lock()");
266     gyro = arduinoData.gyroZ;
267     CHECK_T(pthread_mutex_unlock(&arduinoDataMutex), "
           mutex_unlock()");
268     //printf("%f\n", gyro);
269     theta = gyro - gyro0 + angle0;
270     Right();
271     //printf("Gyro : %f Gyro voulu : %f\n", gyro,
           theta_gyro);
272   } while(theta - theta_xy > PRECISION_ANGLE || theta -
       theta_xy < - PRECISION_ANGLE);
273   **/
274   Stop();
275
276   //printf("Theta_xy = %f, Theta = %f\n", theta_xy, theta);
277
278   do {
279     Forward();
280     CHECK_T(pthread_mutex_lock(&marvelMindMutex), "
           mutex_lock()");
281     x = marvelMindPosition.x;
282     y = marvelMindPosition.y;
283     CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
           mutex_unlock()");
284     usleep(500000);
285   } while(pow((x - x1),2) + pow((y - y1),2) > pow(
       POSITION_PRECISION,2));
286   Stop();
287
288   // Pr venir superviseur
289 }
```

```
290
291 void * marvelMindThread() {
292   while (1) {
293     CHECK_T(pthread_mutex_lock(&marvelMindMutex), "
          mutex_lock()");
294     getPositionFromMarvelmindHedge(marvelMindHedge, &
          marvelMindPosition);
295     CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
          mutex_unlock()");
296     //printf("X = %d ; Y = %d\n", marvelMindPosition.x,
          marvelMindPosition.y);
297     usleep(500000);
298   }
299   pthread_exit(NULL);
300 }
301
302 void pingProcess() {
303   int sdPing;
304   struct sockaddr_in addr;
305   struct sockaddr_in serverPingAddr;
306   fd_set readFd;
307   int err;
308   message_ping_t pingMessage;
309   struct timeval timeout = { PING_TIMEOUT, 0 };
310
311   addr.sin_family = AF_INET;
312   addr.sin_port = 0;
313   addr.sin_addr.s_addr = INADDR_ANY;
314
315   serverPingAddr.sin_family = AF_INET;
316   serverPingAddr.sin_port = htons(SUPERVISOR_PORT_PING);
317   serverPingAddr.sin_addr.s_addr = inet_addr(supervisorIp);
```

```
318
319   pingMessage.robotId = ROBOT_ID;
320   pingMessage.batteryLevel = 100;
321
322   // Setup socket
323   sdPing = socket(AF_INET, SOCK_DGRAM, 0);
324   CHECK(sdPing, "--socket()");
325   CHECK(bind(sdPing, (const struct sockaddr *) &addr,sizeof
         (addr)), "bind()");
326
327   FD_ZERO(&readFd);
328       FD_SET(sdPing, &readFd);
329
330   while (1) {
331     sendto(sdPing, &pingMessage, sizeof(message_ping_t), 0,
          (const struct sockaddr *) &serverPingAddr, sizeof(
         serverPingAddr));
332
333     timeout.tv_sec = PING_TIMEOUT;
334     timeout.tv_usec = 0;
335     err = select(sdPing + 1, &readFd, NULL, NULL, &timeout)
          ;
336     if (err <= 0) {
337           // Timeout
338       printf("Perte de connexion avec le superviseur\n");
339     } else if (err > 0) {
340       if (recv(sdPing, NULL, sizeof(message_ping_t), 0) ==
           sizeof(message_ping_t)) {
341         // Received ping
342       }
343     }
344     sleep(PING_PERIOD);
```

```
345   }
346 }
347
348 /*
349  * 'open_port()' - Open serial port 1.
350  * Returns the file descriptor on success or -1 on error.
351  */
352
353 int open_port(char* port_name)
354 {
355     int fd; /* File descriptor for the port */
356
357     fd = open(port_name, O_RDWR | O_NOCTTY | O_NDELAY);
358     if (fd == -1)
359     {
360         /*
361     * Could not open the port.
362     */
363         char message[100];
364         sprintf(message,"open_port: Unable to open %s",
                port_name);
365         perror(message);
366     }
367     else
368         fcntl(fd, F_SETFL, 0); // non blocking mode
369
370     return (fd);
371 }
372
373 /*
374 *    Setup the serial port with standard to communicate with
      arduino
```

```
375 *   Setup read function to be blocking until at least 1
    byte is read.
376 *
377 */
378 void SetupSerialOptions(int fd)
379 {
380     struct termios options;
381     /*
382      *Get the current options for the port...
383     */
384
385     tcgetattr(fd, &options);
386
387     /*
388     * Set the baud rates to 19200...
389     */
390
391     cfsetispeed(&options, B115200);
392     cfsetospeed(&options, B115200);
393
394     /* 8N1 Mode */
395     options.c_cflag |= (CREAD | CLOCAL); /* Enable receiver
        ,Ignore Modem Control lines       */
396
397     options.c_cflag &= ~PARENB; /* Disables the Parity
        Enable bit(PARENB),So No Parity   */
398     options.c_cflag &= ~CSTOPB; /* CSTOPB = 2 Stop bits,
        here it is cleared so 1 Stop bit */
399     options.c_cflag &= ~CSIZE;  /* Clears the mask for
        setting the data size             */
400     options.c_cflag |= CS8;     /* Set the data bits = 8
                                       */
```

```
401
402     options.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG); /*
          Non Cannonical mode                                */
403
404     options.c_oflag &= ~OPOST; /*No Output Processing*/
405
406     /* Setting Time outs */ http://unixwiz.net/techtips/
          termios-vmin-vtime.html
407     options.c_cc[VMIN] = 1;     /* Read at least 1
          characters */
408     options.c_cc[VTIME] = 0; /* tenth of seconds*/
409
410     int status;
411     // ioctl(fd, TIOCMGET, &status);
412
413     // status |= TIOCM_DTR;
414     // status |= TIOCM_RTS;
415
416     // ioctl(fd, TIOCMSET, &status);
417     /*
418     * Set the new options for the port...
419     */
420
421     if (tcsetattr(fd, TCSANOW, &options) != 0)
422         printf("\n  ERROR ! in Setting attributes\n");
423
424     tcflush(fd, TCIFLUSH); /* Discards old data in the rx
          buffer            */
425 }
426
427 void init() {
428   int x0, y0, x, y, x1, y1;
```

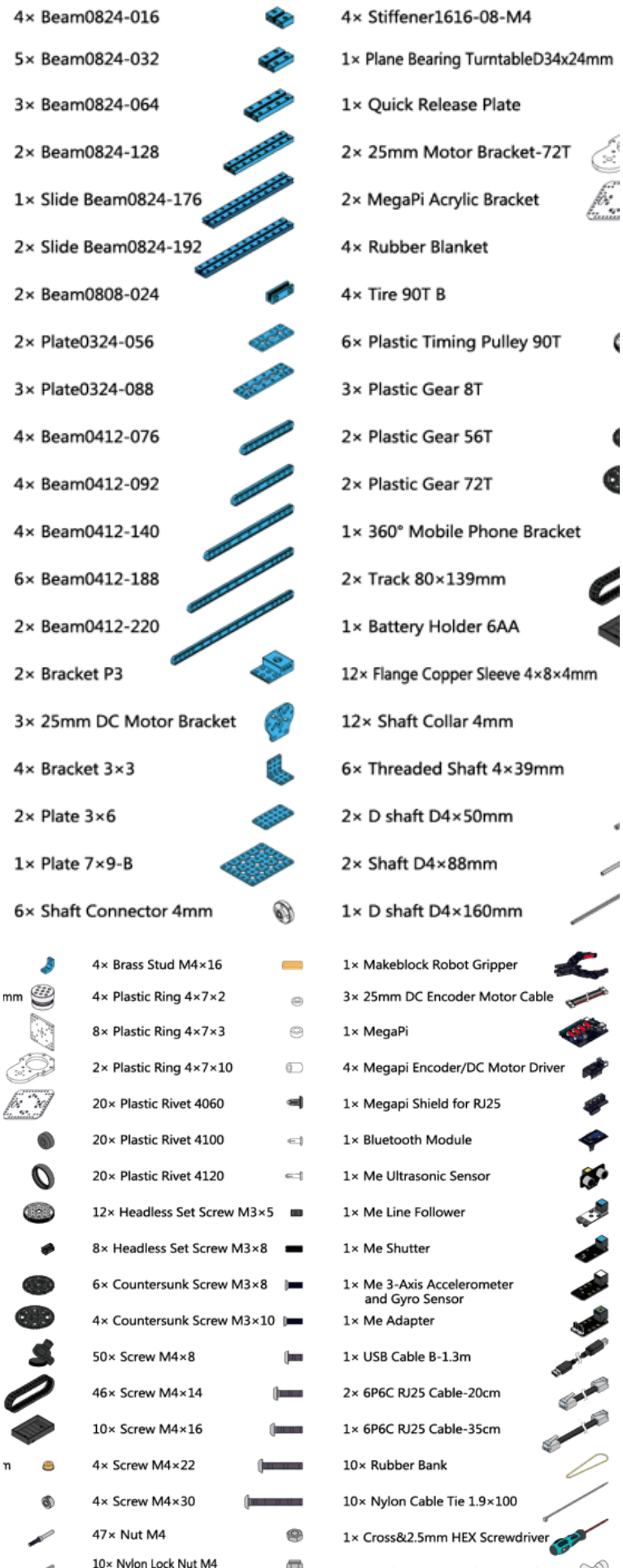
```
429
430   do {
431     CHECK_T(pthread_mutex_lock(&marvelMindMutex), "
          mutex_lock()");
432     x0 = marvelMindPosition.x;
433     y0 = marvelMindPosition.y;
434     CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
          mutex_unlock()");
435     usleep(500000);
436   } while( x0 == 0 && y0 == 0);
437
438   do {
439     Forward();
440     CHECK_T(pthread_mutex_lock(&marvelMindMutex), "
          mutex_lock()");
441     x = marvelMindPosition.x;
442     y = marvelMindPosition.y;
443     CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
          mutex_unlock()");
444     usleep(500000);
445   } while(pow((x - x0),2) + pow((y - y0),2) < pow(300,2));
446
447   Stop(fd);
448   x1 = x;
449   y1 = y;
450   usleep(500000);
451
452   do {
453     Backward();
454     CHECK_T(pthread_mutex_lock(&marvelMindMutex), "
          mutex_lock()");
455     x = marvelMindPosition.x;
```

```
456     y = marvelMindPosition.y;
457     CHECK_T(pthread_mutex_unlock(&marvelMindMutex), "
           mutex_unlock()");
458     usleep(500000);
459   } while(pow((x - x0),2) + pow((y - y0),2) > pow(
       POSITION_PRECISION,2));
460   Stop(fd);
461
462   angle0 = atanf((float) (y1-y0)/(x1-x0))*180/3.1415; //
       angle en radian
463   if (x1-x0 < 0) angle0 = 180 + angle0;
464   //if (angle0 < 0) angle0 = 360 - angle0;
465   CHECK_T(pthread_mutex_lock(&arduinoDataMutex), "
       mutex_lock()");
466   gyro0 = arduinoData.gyroZ;
467   CHECK_T(pthread_mutex_unlock(&arduinoDataMutex), "
       mutex_unlock()");
468   //printf("Angle  : %f, Gyro 0 : %f\n", angle0, gyro0);
469   //printf("X0 = %d Y0 = %d X1 = %d Y1 = %d\n", x0, y0, x1,
        y1);
470 }
471
472 void Forward()
473 {
474     int n;
475     n = write(fd, SPEED_FORWARD, strlen(SPEED_FORWARD)+1);
         //$speedR,speedL,gyroZ,ultrasound\n en mm/s
476     if (n < 0)
477       printf("write() of 10 bytes failed!\n");
478     usleep(2000); //5s plutot utiliser des alarms
479 }
480
```

```
481 void Backward()
482 {
483     int n;
484     n = write(fd, SPEED_BACKWARD, strlen(SPEED_BACKWARD)+1)
        ; //$speedR,speedL,gyroZ,ultrasound\n en mm/s
485     if (n < 0)
486   printf("write() of 10 bytes failed!\n");
487     usleep(2000); //5ms plutot utiliser des alarms
488 }
489
490 void Left()
491 {
492     int n;
493     n = write(fd, SPEED_LEFT, strlen(SPEED_LEFT)+1); //
        $speedR,speedL,gyroZ,ultrasound\n en mm/s
494     if (n < 0)
495   printf("write() of 10 bytes failed!\n");
496     usleep(5000); //5ms plutot utiliser des alarms
497 }
498
499 void Right()
500 {
501     int n;
502     n = write(fd, SPEED_RIGHT, strlen(SPEED_RIGHT)+1); //
        $speedR,speedL,gyroZ,ultrasound\n en mm/s
503     if (n < 0)
504   printf("write() of 10 bytes failed!\n");
505     usleep(5000); //5ms plutot utiliser des alarms
506 }
507
508 void Stop()
509 {
```

```
510     int n;
511     n = write(fd, "$S0,0\n", 18); //$speedR,speedL,gyroZ,
            ultrasound\n en mm/s
512     if (n < 0)
513   printf("write() of 10 bytes failed!\n");
514     usleep(5000);
515 }
```

ANEXO A – ESPECIFICAÇÕES DOS ROBÔS



Fonte: (MAKEBLOCK, 2015)