UNIVERSIDADE
BEIRA INTERIOR

Simulação Computacional de um Rover Robótico Agrícola para Pulverização Controlada de Infestantes e Recolha de Frutos Caídos

João Pedro Lopes Ribeiro

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletromecânica
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Miguel de Figueiredo Dinis Oliveira Gaspar

janeiro de 2022

Dedicatória

Ao meu Pai, que sempre olhará por mim...

Agradecimentos

Chegado aqui e colocando em perspetiva todos os meus anos de ensino, de formação profissional e de vivências faz sentido agradecer a muitas pessoas. Começando na minha professora da escola primária (não pelos puxões de orelhas) até aos professores da universidade, passando pelos funcionários, pelos colegas de escola e os colegas de trabalho, pois todos eles tiveram influência no meu crescimento e são de alguma forma responsáveis por estar aqui hoje.

Mas como este texto não deve ser mais extenso que o próprio trabalho desenvolvido e também porque não haveria folhas suficientes para escrever o nome de todas estas pessoas, prefiro focar-me nas pessoas que me ajudaram nesta fase da minha vida e que tornaram possível a realização deste documento.

Entre estas pessoas está a minha família, todos eles sabem quem são e a todos eles agradeço, em especial há minha mãe, por todo o carinho dado e por me ter pago todos os meus anos de ensino.

Agradecer ainda aos meus amigos mais próximos por todos os conselhos e apoio dado e claro agradecer ao meu pilar, à Andreia Alves que sem ela nada disto tinha sido possível.

Resumo

O contínuo aumento da população mundial tem avultado a necessidade de bens alimentares, levando a um aumento das explorações agrícolas para garantir o fornecimento destes bens, de uma forma direta, às populações e de uma forma indireta a todas as indústrias transformadoras do ramo alimentar. Esta condição tem levado a agricultura a reinventar-se e a introduzir novas técnicas e ferramentas para garantir um maior controlo das colheitas e um aumento dos rendimentos na produção alimentar. No entanto, a falta de mão-de-obra aliada à evolução de infestantes resistentes a herbicidas, tem levado a crescentes constrangimentos numa produção agrícola que a cada dia que passa se pretende mais elevada e de melhor qualidade. Porém, com a crescente evolução das áreas da eletrónica, da automação e da robótica, surgem novos caminhos para responder a estes problemas. Nesse sentido, o Grupo Operacional PrunusBot desenvolveu um rover robótico para otimizar as tarefas de controlo de infestantes e de recolha de frutos caídos no chão de pomares. Em que, para o caso do controlo de infestantes é proposto um sistema de pulverização de precisão, reduzindo a quantidade de herbicida a aplicar. A recolha de frutos caídos ambiciona promover a sustentabilidade do processo, dirigindo os frutos caídos para alimentação animal e com a perspetiva de analisar o efeito desta atividade na redução da atividade microbiana nas culturas da campanha seguinte, evitando consequentes danos.

No sentido de dar continuidade a esta investigação, a presente dissertação propõe a recriação deste rover robótico num software de simulação robótico e propõe ainda a criação de um modelo de simulação semelhante ao de uma exploração agrícola. Pretende-se deste modo desenvolver um algoritmo de controlo do rover robótico nas tarefas de pulverização controlada e de recolha de frutos caídos. Este trabalho sugere o desenvolvimento e teste através de um simulador robótico por este apresentar benefícios em relação aos métodos tradicionais, nomeadamente a nível de rapidez, facilidade de implementação e teste de diferentes cenários e hipóteses de operação podendo até efetuar simulações de forma simultânea. O uso de simuladores robóticos apresenta ainda um maior grau de liberdade e criatividade, uma vez que não existem preocupações relativas à danificação do *hardware*. De referir também que os custos de desenvolvimento são muito reduzidos.

Palavras-chave

Agricultura, Robótica, Simuladores Robóticos, rover robótico, pulverização controlada, controlo de infestantes, recolha de frutos.

Abstract

The continuous rise in world population has increased the need for food, leading to a rise of agricultural holdings to ensure de supply of these goods, directly to the populations and indirectly to all the processing industries in the food business. This situation has led agriculture to reinvent itself and introduce new technics and tools to ensure a tighter control of the crops and increase yields in food production. However, the lack of labour coupled with the evolution of weeds resistant to herbicides, created a crisis in agricultural food production. However, with the growing evolution in electronics, automation, and robotics, new paths are emerging to solve these problems. With this in mind, the Operational Group PrunusBot developed a robotic rover, to optimize the tasks of weed control and collection of fallen fruits of an orchard. In weed control, it is proposed a localized spraying system, therefore, reducing the amount of applied herbicides. With fruit collection, it's possible to direct the fallen fruits for animal feeding, and aims to reduce the microbial activity on the next campaign crops, therefore avoiding damage.

To continue this research, this dissertation proposes the recreation of this robotic rover on a robotic simulation software. It also proposes the recreation of a similar environment to that of a agricultural exploration, in order to later create and algorithm that controls the rover on the tasks of localized spraying and fallen fruit collection. This dissertation suggests the creation and testing of these algorithms through a robotic simulador, because of the benefits in relation to a more traditional method, namely the speed and ease of testing different scenarios and hypothesis, with the added benefit of being able to test the two tasks simultaneously. This method also allows for greater freedom and creativity because there are no concerns about hardware damage. It should also be noted that the development costs are very low.

Keywords

Agriculture, Robotics, Robotic simulators, Robotic Rover, Controlled Spraying, weed control, Fruit collection.

Índice

Lista de Figuras

Lista de Tabelas

Nomenclatura

Geral:

D	Diâmetro [mm];
f	Frequência [Hz];
V	Velocidade Linear [m/s];
v	Velocidade Angular [rad/s];
RPM	Rotações Por Minuto [rad/s];

Acrónimos:

API	Application Programming Interface;
ARGoS	Autonomous Robots go Swarming;
CAD	Computer-Aided Design;
CAE	Computer Aided Engineering;
CCD	Charge Coupled Device;
CMOS	Complementary Metal–Oxide–Semiconductor;
CNN	Redes Neurais Convulsionais;
CPU	Central Processing Unit;
DGPS	GPS Diferencial;
EA	Evolutionary Algorithm;
EPFL	Swiss Federal Institute of Technology;
FEADER	Fundo Europeu Agrícola de Desenvolvimento;
FOG	Giroscópio de Fibra Ótica;
GPS	Global Positioning System;
HO	Homem-Máquina;
IR	Infravermelho;
LIDAR	Light Detection And Ranging;
LQR	Linear Quadratic Regulator;
MPC	Model Predictive Control;

NIR	Infravermelho Próximo;
OH	Operador Homem;
OMPL	Open Motion Planning Library;
PID	Proportional-Integral-Derivative;
PrunusBot	Sistema Robótico Aéreo Autónomo de Pulverização de Precisão e Previsão De Produção Frutícola;
RGB	Red Green Blue;
ROS	Robot Operating System;
RTK	Cinemática em Tempo Real;
RTK-GPS	Real-time kinematic Positioning;
SI	Swarm Intelligence;
SMC	Sliding Mode Control;
SRA	Sistemas Robóticos Autónomos;
UBI	Universidade da Beira Interior;
URDF	Unied Robot Description Format;
VA	Veículos Autónomos;
V-REP	Virtual Robot Experimentation Platform.

1. Introdução

Neste capítulo introdutório é apresentada uma perspetiva geral do que é a agricultura de precisão, do que a automação pode fazer pela agricultura e como o uso de veículos autónomos não tripulados pode ajudar em diversas atividades agrícolas, como por exemplo o controlo de infestantes e recolha de frutos caídos.

1.1. Enquadramento

Com o aumento proeminente da população mundial, simultaneamente têm duplicado a necessidade de alimentos e produtos agrícolas. Esta condição ocorre quando a mão-de-obra qualificada no campo tem vindo a diminuir, por existirem outras atividades económicas mais atrativas e que levam as gerações mais novas para os grandes centros urbanos, criando ainda um problema de desertificação populacional nas zonas rurais. Este é, por exemplo, o caso do Japão onde a idade média dos agricultores aumentou para mais de 60 anos, tendo 28% dos agricultores mais de 65 anos [1].

Aliado a esta questão está o aumento da população mundial que se prevê passar das 6,8 biliões de pessoas, para mais de 10 biliões de pessoas que se estima existirem em 2050 [2].

Fruto deste crescimento populacional e também dos fatores ambientais, a agricultura tem procurado modernizar-se e acompanhar os desenvolvimentos tecnológicos de forma a dar uma resposta às necessidades mundiais. Desde a introdução de sistemas pouco complexos para o controlo de sistemas de rega, uso de sistemas de doseamento de fertilizantes e pesticidas, entre outras técnicas [3].

Após a década de 90, a agricultura sofreu uma mecanização em massa, o que começou a provocar pequenos problemas ambientais, que se foram agravando no passar dos anos, como a produção de gases de efeito estufa, tornando a agricultura num dos setores mais poluentes do mundo nos dias de hoje [3].

1.2. O problema em estudo e a sua relevância

Uma das principais causas que tem contribuído para a poluição neste setor é o uso de herbicidas para a remoção de infestantes. O problema da presença de infestantes tem sido gerido com a rotatividade das culturas, pela utilização de sistemas de controlo mecânico para a remoção das infestantes e através da aplicação de uma ampla variedade de herbicidas [4]. No entanto, a evolução de infestantes resistentes a herbicidas, aliado ao facto da descoberta de novos tipos de herbicidas ter cessado nos últimos 30 anos, está a gerar uma crise na gestão dos infestantes na agricultura [5]. Estima-se que atualmente, a nível mundial, as perdas provocadas pelos infestantes resistentes a herbicidas nas culturas seja de aproximadamente 500 milhões de dólares por ano, no entanto este valor pode subir para os 100 mil milhões de dólares por ano quando o controlo químico é perdido [6]. Por exemplo, as cinco formas de resistência do cânhamo d'água (*Amaranthus tuberculatus*) em Ilinois estão apenas a um gene da perda total do controlo químico. Isto porque sementes com resistência natural a herbicidas estão a aumentar, existindo problemas semelhantes na produção de soja [7].

Uma alternativa à remoção das infestantes através de processos químicos é a remoção mecânica, com a utilização de equipamentos agrícolas ou veículos autónomos. A utilização de sistemas mecânicos na remoção das infestantes tem como alvo, geralmente, as infestantes jovens, incluindo as sementes em germinação [5]. Estes sistemas mecânicos são utilizados, por norma, antes de qualquer plantação em que é feita uma mobilização do solo, com recurso a charruas, rolos destroçadores, fresas, etc. Esta mobilização permite a remoção das infestantes germinadas e permite ainda a obtenção de uma adequada estruturação do solo que seja suficiente ao bom desenvolvimento radicular de cada espécie e/ou variedade de plantas [8]. No entanto, após a plantação o controlo mecânico das infestantes limita-se geralmente às áreas entre as linhas das culturas. Isto porque a remoção manual das infestantes jovens entre as árvores é difícil e impraticável em grande escala [5]. Além disso, o uso de maquinaria entre as fileiras tem desvantagens, como a compactação do solo devido ao elevado peso, existindo ainda a incapacidade destas máquinas trabalharem nas linhas de cultivo após o crescimento das plantações [9]. Mas com a evolução que tem existido no campo da eletrónica, da automação e da robótica, surgem novos caminhos para a remoção mecânica de infestantes e que conseguem um aumento da produtividade com um menor custo de produção [10].

Tudo isto acontece num tempo em que a consciência sobre a conservação do meio ambiente e as alterações climáticas estão a mudar a base da produção agrícola, limitando o uso de produtos químicos, como fertilizantes e pesticidas. E foi por estas premissas que nasceu a agricultura de precisão, com o intuído de reduzir os custos, aumentar a produtividade, proteger o meio ambiente e o bem-estar dos trabalhadores e levando assim a automação para o mundo da agricultura [10].

1.3. Objetivos e contribuição da dissertação

Foi dentro deste conceito que nasceu o Grupo Operacional PrunusBot - Sistema robótico aéreo autónomo de pulverização de precisão e previsão de produção frutícola [11], liderado pela Universidade Beira Interior e que tem como objetivo o desenvolvimento de sistemas robóticos destinados à inovação tecnológica na fruticultura, nomeadamente em pomares de prunóideas na região da Beira Interior.

No âmbito deste projeto foi construído um rover robótico terrestre, autónomo e multifacetado, projetado para a utilização em pomares de pessegueiros [12]. Os principais objetivos deste rover robótico visam a pulverização de precisão para controlo de infestantes e ainda a recolha dos frutos caídos no chão de pomares. Assim, a presente dissertação enquadra-se neste projeto, propondo a simulação computacional do funcionamento do rover robótico, sendo também proposto um algoritmo de controlo para cumprir as funções que o rover desempenha.

A simulação computacional foi realizada no simulador CoppeliaSim e consistiu em carregar para o simulador uma imagem 3D do protótipo desenvolvido, bem como todas as suas características, físicas, mecânicas e elétricas e posteriormente testar os vários algoritmos desenvolvidos

Deste modo, a principal contribuição deste trabalho consistiu no desenvolvimento de algoritmos de controlo, regulação e comando do rover robótico, sendo assim dada a possibilidade do trabalho desenvolvido vir a ser posteriormente aplicado e testado no protótipo construído. Além disso, parte do trabalho desenvolvido poderá ser aplicado em outros protótipos que venham a ser construídos.

1.4. Visão geral e organização da dissertação

A presente dissertação está organizada num conjunto de seis capítulos, sendo que neste primeiro Capítulo foi feita uma introdução, geral, ao problema em análise e foi dado a conhecer o contributo que este trabalho pretende dar na área da robótica para a agricultura.

No Capítulo 2 é analisado de uma forma mais aprofundada o que é a automação e a robótica e como estas áreas podem ser aplicadas na agricultura, sendo apresentados os desafios que existem nesta implementação. São depois apresentados os principais componentes que qualquer sistema robótico deve ter para um bom funcionamento nesta atividade, e são ainda descritas as características e medidas de desempenho para esse bom funcionamento. Por fim, são apresentados os sistemas robóticos, de maior destaque, que já foram construídos nas diversas áreas da agricultura bem como os principais métodos para o controlo destes sistemas robóticos, nomeadamente, de sistemas robóticos autónomos.

No Capítulo 3 é dado a conhecer o rover robótico onde se baseia o presente trabalho, sendo feita uma breve descrição de todos os seus componentes e analisado o funcionamento dos eixos do envelope de trabalho. Seguidamente, é feita uma breve apresentação aos simuladores robóticos, sendo apresentados alguns dos simuladores que existem e que se adaptam ao trabalho proposto. Por fim, é apresentado o simulador escolhido para este trabalho e os motivos que levaram a essa escolha.

No capítulo 4 é descrito o processo de recriação deste rover robótico no simulador escolhido, bem como os princípios que foram seguidos para esta recriação, é igualmente descrito o processo de criação do ambiente 3D. Além disso, são também apresentados os princípios que foram seguidos no desenvolvimento de um algoritmo de controlo, considerando todas as condições necessárias cumprir.

O Capítulo 5 apresenta os principais resultados obtidos nas diversas simulações realizadas, sendo analisados parâmetros como velocidade de operação, processamento da imagem, tempo de operação, entre outros. São também apresentados neste capítulo dois casos de estudo. O primeiro caso de estudo é referente à tarefa de recolha de objetos, enquanto o segundo caso de estudo se refere à pulverização controlada. Seguidamente, é apresentada uma breve análise ao comportamento do rover robótico e do algoritmo proposto para cada uma das tarefas.

Por fim, no Capítulo 6, são enunciadas as principais conclusões do trabalho realizado e sugeridas direções de investigação para desenvolvimentos futuros.

2. Estado da Arte

Neste capítulo são descritos os maiores desafios que têm sido relatados na literatura científica, ao longo dos últimos anos, na implementação da automação/robótica na agricultura. É também realizada uma análise às várias tipologias de robôs que têm sido desenvolvidas bem como os resultados que se têm atingido.

2.1. Introdução

As operações agrícolas são complexas, diversificadas, de trabalho intensivo e direcionadas à cultura, no entanto a produtividade agrícola aumentou significativamente e continuamente ao longo dos séculos por consequência da mecanização, intensificação e, mais recentemente, com a introdução da automação [13].

Os robôs agrícolas tem o potencial de elevar a qualidade dos produtos frescos, reduzir os custos de produção, reduzir o trabalho manual mais penoso e compensar a falta de trabalhadores que existe no setor, pelo menos em algumas partes do mundo [14]. Sendo que a falta de trabalhadores é ainda mais agravada pela tendência de aumento dos campos agrícolas [15].

Embora a robótica e a automação exijam equipamentos e uma mão-de-obra especializada (e consequentemente mais cara), esta gera um aumento na produtividade agrícola, pois por norma a mão-de-obra necessária diminui o suficiente para compensar o custo inicial mais alto [16], [17]. Além disso, a redução das tarefas realizadas em condições adversas e o aumento da qualidade de vida do agricultor pode criar novos argumentos para aumentar a atratividade da profissão agrícola [13], [18]. Isto quando a idade média da mão-de-obra do setor agrícola tem tido um enorme aumento nos últimos anos, demonstrando que esta profissão não é suficientemente atraente para as gerações mais jovens [16], [17].

2.2. Desafios na implementação da automação na agricultura

A automação e a robótica para aplicações agrícolas requerem tecnologias avançadas para lidar com os ambientes complexos e altamente variáveis que são característicos dos campos agrícolas, ao contrário das aplicações industriais, que lidam com tarefas relativamente simples, repetitivas,

bem definidas e pré-determinadas em ambientes estáveis e replicáveis [13], [18]. Além disso, a produção agrícola lida com produtos vivos (frutas, vegetais e flores) que são altamente sensíveis às condições ambientais e físicas como temperatura, humidade, gás, pressão, abrasão e aceleração [19]. De referir ainda que os produtos vivos requerem operações de manuseamento suaves, precisas e muitas vezes complicadas para manter a qualidade suficiente para percorrer a distância e o tempo que separam o local de produção do consumidor final. A junção destas características torna a substituição da habilidade humana por máquinas ou automação extremamente difícil [20].

Por outro lado, os produtos agrícolas têm normalmente um valor comercial relativamente baixo, portanto, o custo de utilização de um sistema de automação por unidade de produto deve ser baixo para ser economicamente viável [13].

Segundo Mousazadeh [10] as principais limitações que existem na aplicação da robótica no domínio agrícola são:

- Áreas operacionais muito grandes;
- Solo irregular e que pode ter alterações até diárias;
- Deslizamento das rodas pode ter muita importância (dependendo do tipo de operação);
- Condições ambientais (chuva, nevoeiro e poeira) podem afetar o funcionamento dos sensores e de outros equipamentos;
- Necessários sistemas de baixo custo;
- Sazonalidade de cada operação agrícola.

A maioria das operações agrícolas ocorre em ambientes não estruturados, caracterizados por mudanças rápidas no tempo e no espaço. Isto porque o terreno, a vegetação, a visibilidade, a iluminação e outras condições atmosféricas são difíceis de definir, variam continuamente, têm incertezas inerentes e geram situações imprevisíveis e dinâmicas [14].

Segundo Bechar & Vigneault [21], o mundo robótico pode ser dividido em quatro grupos tendo em conta as características estruturais dos objetos e dos ambientes:

1. Ambientes e objetos estruturados;
2. Ambientes não estruturados e objetos estruturados;
3. Ambientes estruturados e objetos não estruturados;
4. Ambientes e objetos não estruturados.

Estando cada um destes grupos associado a uma determinada área de aplicações robótica, conforme indicado na Tabela 1.

Tabela 1 –Os quatro grupos existentes e os dominios robóticos associados, adaptado de [21].

		Ambientes	
		Estruturados	*Não Estruturados*
Objetos	*Estruturados*	Aplicações Industriais	Aplicações Militares, aeroespaciais, subaquáticas
	Não Estruturados	Aplicações Medicinais	Aplicações Agrícolas

A aplicação da robótica no domínio agrícola está associado ao quarto grupo em que nada está estruturado e este requisito de operar em ambientes não estruturados dificulta a aplicação da robótica e resulta em sistemas de difícil conceção e caros, tornando-os num desafio em termos de conceção e comercialização [21].

Segundo Steinfeld [22], em muitas situações os robôs autónomos para a agricultura falham devido à quantidade de eventos inesperados que surgem.

Ainda assim e segundo Avital Bechar & Vigneault [21], a implementação da tecnologia robótica na agricultura é realizável se pelo menos uma destas condições for observada:

- O custo da utilização dos sistemas robóticos for inferior ao custo de qualquer método simultâneo;
- O uso de robôs aumenta a capacidade de produção agrícola, e consequentemente o lucro;
- O uso de robôs melhora a qualidade e uniformidade da produção;
- O uso de robôs minimiza a incerteza e a variação nos processos de crescimento e produção;
- O uso de robôs permite ao agricultor tomar decisões e atuar em maior resolução e/ou aumentar a qualidade do produto em comparação aos sistemas concorrentes para obter otimização nas etapas de cultivo e produção;
- O robô é capaz de realizar tarefas específicas que são definidas como perigosas ou que não podem ser executadas manualmente.

2.3. Interação Homem-Máquina

Os robôs agrícolas requerem o desenvolvimento de tecnologias avançadas para lidar com os ambientes, produtos complexos e altamente variáveis [13]. Além disso, a sazonalidade da agricultura torna difícil atingir um alto nível de utilização como é o caso da indústria.

No entanto, mesmo que a viabilidade técnica e económica da maioria das aplicações de robótica de campo agrícola não seja alcançada num futuro próximo, usando o conhecimento e tecnologias existentes, a autonomia parcial pode agregar valor à máquina muito antes que os robôs de produção estejam totalmente disponíveis numa versão completamente autónoma.

Além disso, segundo o princípio de Pareto, cerca de 80% de uma tarefa é de fácil adaptação à robótica/automação, mas os restantes 20% são de difícil adaptação. Ainda assim, se forem automatizadas as partes fáceis de uma tarefa, pode-se reduzir o trabalho manual em cerca de 80% [23]. Além disso, o desenvolvimento de robôs parcialmente autónomos é um excelente caminho de transição para o desenvolvimento e teste de *software* e *hardware* que eventualmente poderá vir a ser integrado em sistemas totalmente autónomos [21].

2.3.1. Conceito robô – homem

As capacidades humanas de perceção, pensamento e ação são ainda incomparáveis em ambientes com anomalias e eventos imprevistos [24]. Como resultado, as habilidades humanas e do robô são ainda complementares [25].

Sheridan & Daniel [26] descreveram 10 níveis de colaboração entre o robô–homem, variando de totalmente autónomo, sem intervenção humana, a totalmente manual. O nível de colaboração necessário é alterado de um para outro durante o desempenho da própria tarefa, sempre que houver uma alteração quer seja a nível do ambiente, zona de cultivo, robô ou dos parâmetros do Operador-Homem.

Introduzir um Operador-Homem (OH) numa operação com ciclos para interagir com, em vez de apenas supervisionar o sistema, é uma tendência na investigação em robótica agrícola, que pode ajudar a melhorar o desempenho e reduzir a complexidade do sistema. Na verdade, tem sido realizados vários progressos nas últimas duas décadas neste sentido [21]. Por exemplo, Ceres et al. [27] desenvolveram a cooperação de um robô agrícola com um OH, que ajudou a resolver três problemas difíceis:

- Conduzir o robô pelo campo, de árvore em árvore ou de fileira em fileira;
- Detetar e localizar os produtos;
- Agarrar e destacar produtos selecionados.

Yoshisada Nagasaka et al. [15] desenvolveram um sistema Robô-Homem em que o operador controla vários sistemas operacionais semiautónomos em arrozais. Já Bechar & Edan [28] desenvolveram num sistema robótico para deteção de melões, relatando que o sistema aumentou, em média, a deteção em 4% quando comparado com a deteção manual e 14% em comparação com um sistema totalmente autónomo.

2.3.2. Sistemas robóticos autónomos

Os veículos de condução autónoma são estudados há muitos anos, existindo uma série de inovações exploradas já desde a década de 1920, no entanto, o conceito de veículos agrícolas totalmente autónomos está ainda longe dos nossos dias [10].

Os sistemas robóticos autónomos (SRA) são desenvolvidos para realizar tarefas, tomar decisões e agir em tempo real sem intervenção humana. Estes sistemas são necessários em domínios que exigem redução da mão-de-obra e a carga de trabalho e são também mais adequados para aplicações que exigem precisão repetida e alta produção em condições estáveis [21].

Os requisitos básicos para alcançar um grau razoável de autonomia é ter um bom nível de sensorização e de processamento. Como tal, os SRA devem possuir um elevado grau de flexibilidade para se envolverem em condições ambientais em constante mudança, bem como para processar as informações que recebem dos seus próprios sensores [21].

Segundo Ng & Trivedi [29], são frequentemente encontrados dois desafios importantes ao projetar SRA:

1. Requisitos de resposta não linear em tempo real subjacentes à formulação do controlo entre motor- sensor;
2. Modelação e qual a abordagem que um ser humano faria para resolver os problemas encontrados.

Tal como acontece com os sistemas robô-homem, a natureza sazonal da agricultura torna difícil que um SRA atinja um alto nível de utilização. Assim, o complexo ambiente agrícola, combinado com uma produção intensiva e a curta temporada de produção, exigem que um SRA seja robusto com curto tempo de desenvolvimento e baixo custo [13].

Mesmo sendo difícil de alcançar a autonomia total num robô agrícola, os sistemas robóticos parcialmente autónomos podem acrescentar já algum valor na produção agrícola, muito antes que a autonomia total seja alcançada, isto se o custo do equipamento for suficientemente baixo e o seu desempenho alto o suficiente para o tornar economicamente viável [21]. Além disso, se a intervenção humana necessária for baixa o suficiente, pode ser possível um único operador para supervisionar ou colaborar com várias máquinas [23].

2.4. Principais Componentes

2.4.1. Princípios e habilidades exigidas

Os robôs agrícolas são geralmente projetados para executar uma 'tarefa principal', que geralmente é uma tarefa agrícola específica, como plantar, lavrar, podar, colher, empacotar, etc. Para executar a 'tarefa principal', o SRA requer a capacidade de realizar várias 'tarefas de apoio', como por exemplo, localização, navegação, deteção do objeto a tratar, etc. As informações e comandos são transferidos entre as 'tarefas de apoio' e as 'tarefas principais '. Cada 'tarefa de apoio' controla um ou vários subsistemas e dispositivos, e um subsistema ou dispositivo pode servir a várias 'tarefas de apoio' [21], como exemplificado na Figura 1.

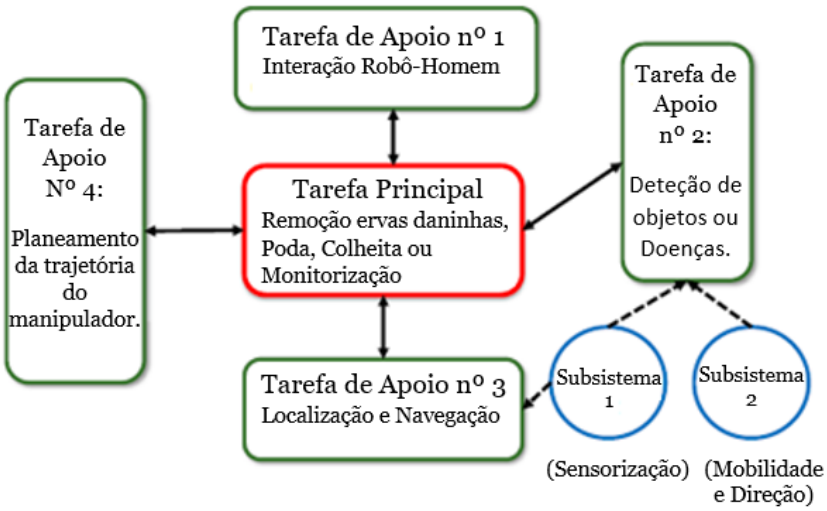


Figura 1 - Estrutura dos subsistemas das tarefas de um robô agrícola. Setas sólidas representam comandos, transferência de dados e informações; setas tracejadas representam conexões conceituais. Adaptado de [21].

2.4.2. Mobilidade e direção

De acordo com Grimstad et al. [30], precisam de ser feitas várias reflexões no desenvolvimento de um robô agrícola, como a necessidade de operar durante os períodos de chuva sem ficar preso ou danificar a estrutura do solo, mantendo os custos gerais do robô num nível que o torne economicamente viável. A estrutura/plataforma do robô deve ser flexível, o que reduz a complexidade, mas permite que todas as rodas estejam em contato com o solo. As plataformas, usadas na agricultura de forma comum, compreendem plataformas de 4 rodas com tração nas 4 ou 2 rodas e direção nas 2 ou 4 rodas. Existem também algumas plataformas com tração em 6 rodas e até plataformas movidas com lagartas.

Haibo et al. [31] desenvolveram um rover robótico com tração nas 4 rodas e direção nas 4 rodas que se adapta ao ambiente de trabalho e aos requisitos agronómicos das técnicas de sementeira de precisão do trigo (Figura 2a). A rotação e direção das rodas são controladas de forma

independente com quatro servomotores para propulsão e quatro motores de passo para direção. Um controlador central coordena os oito motores. Usam ainda quatro módulos de roda idênticos, cada um capaz de proporcionar direção e propulsão. A plataforma permite orientar o veículo em qualquer direção e modificar ou manter a orientação do veículo independentemente da direção de deslocamento do veículo, mesmo durante o processo de viragem. Os testes de campo mostraram que as taxas de sementeira ultrapassam 93% em diferentes velocidades de sementeira. Um projeto semelhante foi desenvolvido para deteção de infestantes [32], conforme exposto na Figura 2b.

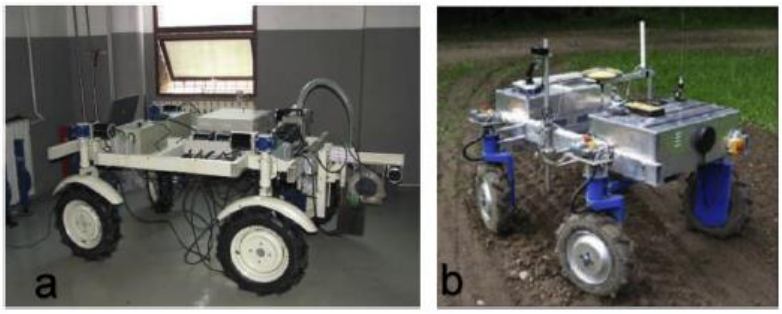


Figura 2 - Plataformas robóticas de tração e direção nas 4 rodas: a) Plataforma para plantação de precisão de trigo, adaptado de [31]; b) Detecção de infestantes, adaptado de [32].

2.4.3. Deteção e auto localização

Atualmente os sensores são utilizados para uma ampla variedade de funções, desde mapeamento, localização, navegação, orientação, deteção de plantas, reconhecimento e medição de parâmetros ambientais (incluindo solo, água, ar e plantas). Estes dados recolhidos são depois usados para apoiar na tomada de decisões, na execução de tarefas e na avaliação do desempenho do robô [21].

Segundo Avital Bechar & Vigneaul [21], os sensores podem ser divididos em categorias de acordo com os dados que geram:

- Medição de movimento, ou seja, odometria ou pontos de referência inerciais e artificiais, posicionamento/radar de laser ou radar de ondas;
- Deteção local de características usando sonar ou visão máquina [33];
- Posicionamento absoluto usando um sistema de posicionamento global (GPS);
- Parâmetros ambientais usando uma ampla variedade de sensores, como infravermelho próximo (NIR) ou infravermelho (IR), raio X, fluorescência, acústica, ótica, visão 2-D e 3-D, medidor de força, etc. [34], [35].

Os sensores podem ainda ser divididos com base na posição ou informação que estão a recolher, por exemplo, sensores internos e externos. Sensores internos medem o estado das diferentes partes de um sistema, por exemplo, codificadores para medição dos ângulos numa articulação ou

roda, acelerómetros para medir as acelerações lineares ou a inércia e giroscópios para medir as acelerações rotacionais. Estes sensores são geralmente usados para cálculo de ângulos mortos [33]. No entanto, os sensores internos tendem a desviar e a acumular erros devido a mudanças de temperatura, gravidade, deslizamento das rodas e exposição a campos magnéticos locais ou proximidade de materiais magnéticos. Quando os dados do sensor inercial são integrados para fornecer posição e orientação, essas fontes de erro podem levar rapidamente a importantes desvios posicionais [21].

Já os sensores externos usados na robótica agrícola ou sistemas automatizados destinam-se à recolha de informações ambientais sobre o estado do sistema em relação à posição do robô e ao posicionamento local dos vários componentes. Os sistemas mais comuns são: GPS, infravermelho, visão máquina, radar a laser *Light Detection And Ranging* (LIDAR), ondas ultrassónicas e hiper-espectrais [33]. Entre estes sensores externos, a visão máquina e os sensores GPS têm alcançado o maior sucesso comercial [36].

Os sensores GPS, que fornecem posicionamento absoluto do sistema, são usados para navegação e orientação. Os sistemas GPS avançados e de alta precisão, ou seja, GPS cinemático em tempo real (RTK) e GPS diferencial (DGPS), permitem medições precisas em tempo real. No entanto, o elevado custo é um dos principais fatores para o aumento do custo do sistema robótico [37].

Os sistemas de visão são sensores altamente versáteis e frequentemente usados para navegação, orientação e deteção de plantas ou objetos, mas também para medições de características de plantas associadas a tarefas agrícolas específicas. O sensor consiste numa câmara e um dispositivo de carga acoplada (CCD) ou um semicondutor de óxido de metal complementar (CMOS), geralmente operando num *NIR* ou espectros visíveis para extrair informações de cor e profundidade [38], sendo uma ferramenta de deteção relativamente barata e poderosa. No entanto, deve ser combinado com outros sensores dentro da estrutura em uso para uma orientação adequada [33].

Os sensores de LIDAR são usados para medições de distância, mapeamento, deteção e prevenção de obstáculos. Devido à sua capacidade de medir com precisão uma posição vetorial relativa (distância e direção), é uma ferramenta promissora para aplicações de orientação de robôs agrícolas [39]. No entanto a existência de poeiras, neblina ou outras condições semelhantes, podem bloquear a luz e reduzir a precisão e o desempenho [39].

Assim e de acordo com o que foi dito acima, precisam ainda de ser realizadas mais investigações em ambientes agrícolas diferentes e conduzidas em sistemas de fusão multisensor para gerar, montar e coordenar todas as informações necessárias para obter um sistema de localização totalmente autónomo ou com as habilidades necessárias de deteção para executar tarefas específicas, uma vez que muitos dos estudos publicados na literatura científicos se centram muito

num único sensor (por exemplo, GPS), que pode não reconstruir todas as informações necessárias [21].

2.4.4. Planeamento e orientação de caminhos

Existem relatos de desenvolvimento de sistemas de orientação automáticos para veículos agrícolas há mais de 50 anos, no entanto com pouco sucesso [40]. Porém, os sistemas de orientação em linhas de cultivo têm alcançado recentemente um alto nível de automação e algum sucesso comercial [36].

O planeamento de caminhos é considerado uma sub tarefa da navegação, que por sua vez é uma das "tarefas de apoio" mais comuns e necessárias nos robôs agrícolas [41]. O problema comum no planeamento de caminhos passa por encontrar um caminho de boa qualidade de um ponto de origem a um ponto de destino e que evite a colisão com obstáculos [42].

Por exemplo, Canning et al. [43] desenvolveram um robô de navegação na floresta com o uso de codificadores, sensores ultrassónicos e um controlador de lógica fuzzy para melhorar a segurança das operações florestais, removendo o operador do veículo e reduzindo os custos ao automatizar essas operações, como observado na Figura 3.

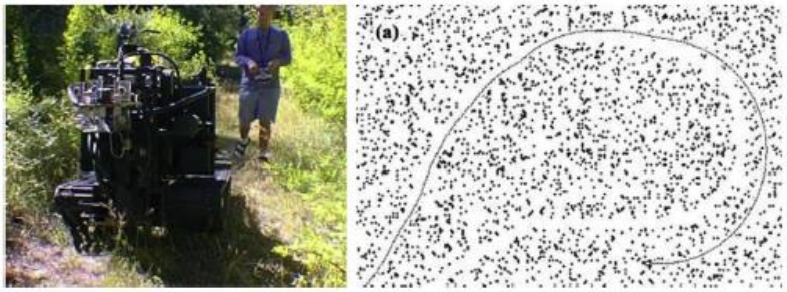


Figura 3 - Robô de navegação em floresta com encoders, sensores ultrassónicos e um controlador fuzzy, adaptado de [43].

Têm sido investigados métodos adicionais na navegação e planeamento de caminhos que incluem o mapeamento probabilístico de estradas, métodos baseados no comportamento, algoritmos genéticos, redes neuronais artificiais, lógica Fuzzy, aprendizagem por reforço e em alguns casos combinações destes métodos [44].

2.4.5. Manipuladores robóticos e garras mecânicas

Na robótica, um manipulador é geralmente um dispositivo eletromecânico do tipo braço que é capaz de se mover num espaço confinado e que termina numa ferramenta ou garra mecânica. Os manipuladores são classificados de acordo com os graus de liberdade, tipos de juntas, comprimento dos elos e comprimento do deslocamento [45].

A principal tarefa de um manipulador é mover a ferramenta ou a garra mecânica até uma determinada posição e orientando essa mesma ferramenta para que possa interagir com um objeto ou executar uma tarefa. Geralmente, o manipulador e a respetiva ferramenta são projetados para executar uma tarefa num ambiente específico. No entanto, um manipulador que tenha um determinado propósito pode ser adaptado para a realização de outras tarefas, nem que para isso seja utilizada outra ferramenta/garra mecânica [21].

Atualmente, os robôs têm uma ampla utilização no domínio industrial. No entanto, apesar da enorme variedade de robôs industriais e das suas capacidades para se adaptarem a muitas tarefas agrícolas, o elevado peso, o alto consumo de energia e os altos custos económicos têm no tornado inadequado para fins agrícolas [46].

Para a implementação da robótica nas tarefas agrícolas, nomeadamente para o controlo de infestantes, cultivo, colheita e transporte, seria usado um manipulador simples e básico, isto se necessário, e uma ferramenta/garra mecânica, que em alguns casos pode nem ter necessidade de contato físico com o objeto. Na maioria dos casos, a ferramenta seria um sensor ou um aplicador, como por exemplo uma câmara ou um bico de pulverização. Devido a todos estes fatores, a maioria dos manipuladores e ferramentas mais simples para tarefas agrícolas devem ser especificamente projetados em vez de adaptados dos sistemas utilizados na indústria [47].

A escolha do mecanismo a utilizar numa determinada ferramenta depende sempre da tarefa que se pretende executar (medição, destruição, colheita), do ambiente (subsolo, solo, água, ar) e do objeto que será manipulado (solo, planta, fruta) [47].

De referir ainda que os objetos a manipular têm uma rigidez menor do que os materiais que cobrem as ferramentas/garras mecânicas o que os pode danificar facilmente. Portanto, quando o contato físico é necessário, a garra mecânica deve ser acompanhada por sensores que meçam a direção e a força a aplicar no objeto de forma a evitar danificá-lo ou a deixar marcas. Devendo para isso serem projetados mapas para as garras compreenderem o planeamento necessário na interação com o objeto [19].

Essas medidas são ainda mais importantes quando é necessário executar movimentos específicos para a realização de uma determinada tarefa. Um exemplo típico é a colheita da maçã, que não consiste simplesmente em puxar o fruto da árvore em direções aleatórias até que o fruto se solte da árvore. Os movimentos de colheita são específicos para cada fruto e dependem da posição e orientação dos frutos em relação à estrutura e posição da árvore e ainda à orientação dos ramos e demais frutos próximos. Nos últimos anos, vários sistemas de braços múltiplos foram desenvolvidos para aplicações agrícolas [20].

No entanto, ainda não existem manipuladores para uso agrícola a serem comercializados [14], nem existe uma metodologia na descrição das tarefas para projetar um braço robótico [21]. Para a estrutura em causa, foi desenvolvida uma solução de braço robótico cartesiano ao qual se

encontrava incorporada uma garra mecânica com 4 dedos destinada a apanhar os frutos caídos no chão [48].

2.5. Características e Medidas de Desempenho nos Robôs Agrícolas

De acordo com Vigneault & Bechar [49] o processo clássico da operação de um robô agrícola consiste em quatro etapas:

1. O robô deteta e adquire dados em bruto sobre o ambiente, tarefa e o seu estado, usando para isso vários sensores;
2. O robô processa e analisa os dados recebidos dos seus sensores para gerar raciocínio e uma perceção do ambiente, da tarefa ou de seu estado para algum nível de consciência da situação;
3. O robô gera um plano operacional baseado na perceção do ambiente e do estado, ou dos objetivos da tarefa;
4. O robô executa as ações necessárias incluídas no plano operacional.

Ao realizar uma tarefa agrícola num ambiente não estruturado, o robô deve repetir estas quatro etapas continuamente, uma vez que o estado da tarefa e o ambiente estão em constante alteração. Na verdade, a capacidade limitada dos sistemas robóticos de raciocinar e planear em tais ambientes resulta num desempenho global baixo, o que faz com que este processo pareça ineficaz [50]. Para melhorar o desempenho do robô, têm sido desenvolvidas, recentemente, várias abordagens. Isto inclui robótica baseada no comportamento [51], [52], que vincula a deteção (primeira fase) à ação (quarto fase), resultando num tipo reativo de comportamento, ou integrando um OH nas fases de raciocínio (segunda fase) e planeamento (terceira fase) [28], [53], [54].

Em ambas as abordagens, as diferentes fases do processo de operação requerem a execução de subtarefas. Com a abordagem baseada no comportamento, o robô controla apenas estas subtarefas. Com a integração de um OH, o controlo dessas subtarefas é realizado pelo operador ou pelo robô. Um exemplo desse processo é apresentado na Tabela 2 relativo a num estudo a de poda seletiva em árvores [55].

Tabela 2 - Fases do processo de operação de uma tarefa de poda de árvore seletiva, adaptado de [55].

Fase	Subtarefa	Controlo
Sensorização	O robô faz a aquisição da imagem	Robô
Raciocínio	Ponto de corte e deteção da orientação	HO ou Robô
Planeamento	Planeamento da trajetória	Robô
Ação	O manipulador alcança e corta os ramos em causa	Robô

Os robôs para aplicações agrícolas que operam em ambientes não estruturados requerem duas categorias de habilidades. A primeira trata das funcionalidades do robô, como por exemplo:

- Evitar obstáculos;
- Auto localização e construção de mapas;
- Planeamento de caminho e navegação.

A segunda trata de aplicações específicas, como:

- Veículos para transporte de mercadorias;
- Segurança, reconhecimento e exploração;
- Plantio, colheita, classificação e manuseio.

2.5.1. Medidas de desempenho para a comparação robótica

De acordo com Vigneault & Bechar [49], várias medidas de desempenho e requisitos tecnológicos têm de ser considerados antes de se ponderar uma aplicação comercial. No geral, as medidas de desempenho são as mesmas, mas a importância de cada medida e os seus efeitos na modelação de sistemas de sensorização, necessários, são específicos para cada tarefa agrícola, tipo de produto, ambiente, etc.

A avaliação do desempenho e a adequação da tecnologia são também necessários para comparar robôs e avaliar o progresso técnico. Infelizmente, não existem métodos ou parâmetros padrão para avaliar o desempenho dos diferentes sistemas testados. Esta falta de parâmetros padronizados torna difícil comparar os diferentes dispositivos ou tecnologias, ou mesmo medir o progresso de sistemas individuais [49].

Num estudo realizado por Vigneault & Bechar [49], que tinha por intuito comparar os diferentes sistemas, dispositivos e tecnologias, identificaram-se alguns parâmetros como sendo os potencialmente mais úteis na comparação de desempenho entre sistemas robóticos e o progresso técnico existente. Esses parâmetros encontram-se descritos na Tabela 3.

Tabela 3 - Medidas de desempenho, requisitos tecnológicos, descrições e unidades de medida padrão usadas para comparar robôs e determinar o progresso técnico, adaptado de [49].

Medida ou Requisito	Descrição
Tempo de Ciclo(s)	Tempo médio necessário para completar um ciclo de ação específico numa tarefa (por exemplo, manusear um objeto individual, colher uma fruta, podar um galho).
Tempo de operação em condições de tempo reais	Tempo médio necessário para cumprir uma missão ou completar uma tarefa específica em tempo real, condições agrícolas dinâmicas.
Velocidade de operação em condições de tempo real (m s^{-1})	Velocidade média medida durante a execução de uma tarefa em tempo real, condições agrícolas dinâmicas.
Taxa de Produção (kg h^{-1}, ha h^{-1}, número de ações h^{-1}, etc)	Quantidade de produto, área ou ações bem-sucedidas (por exemplo, número de infestantes pulverizadas ou árvores podadas) tratadas por unidade de tempo.
Capacidade de operar nas condições de tempo real ($CORT^+$ ou $CORT^-$)	Capacidade de operar em condições de tempo real, apresentando como resultado binário: • ($CORT^+$) - capaz de operar em condições de tempo real; • ($CORT^+$) - Incapaz de operar em condições em tempo real.
Capacidade de Deteção (CD^+ ou CD^-)	Capacidade de medir, avaliar ou detetar objetos, bordas, cores ou características físicas, químicas ou óticas que um sistema robótico precisa para realizar uma tarefa ou missão específica; apresentando como resultado binário: • CD+ - Capaz de realizar uma deteção específica; • CD- - Não é capaz de realizar esta deteção.
Desempenho Deteção (%)	Desempenho dos sistemas robóticos com capacidade de deteção. Os resultados da deteção podem ser: TT, TF, FT, FF. DP é a razão do n.º de deteções apropriadas (TT+ FF) sobre a soma de todas as tentativas de deteção feitas pelo sistema robótico durante o desempenho de uma tarefa ou missão específica em tempo real em condições agrícolas dinâmicas.
Tomada de decisão apropriada (%)	Proporção do número de decisões apropriadas sobre todas as decisões do mesmo tipo feitas durante a execução de uma tarefa ou missão específica em condições agrícolas dinâmicas e em tempo real; o tipo de decisão pode variar consideravelmente entre as tarefas, como virar para a direção certa, colher o produto apropriado, aplicar um tratamento específico em uma superfície específica, iniciar ou terminar uma tarefa na posição apropriada e etc.
Taxa de sucesso das ações (%)	Relação de ações (apanha de frutos, poda de ramos, etc.) que resultaram em sucesso, considerando a descrição da tarefa e sem comprometer a colheita, pelo n.º total de tentativas.
Erro de posição médio e desvio padrão no erro de posição (mm,º, etc.)	A média e o desvio padrão do erro de posicionamento, é a diferença em termos de distância e ângulo entre a localização e orientação estimadas e a posição real do objeto (árvore, fruta, linha, limite de campo, etc.)

Tabela 4 - Medidas de desempenho, requisitos tecnológicos, descrições e unidades de medida padrão usadas para comparar robôs e determinar o progresso técnico, adaptado de [49] (cont.).

Medida ou Requisito	Descrição
Segurança	A segurança do sistema para humanos, infraestruturas e plantas é definido conforme a natureza da tarefa e o ambiente agrícola também deve ser avaliado se é fechado ou aberto, apesar deste aspeto ser pouco utilizado e difícil comparação.
Totalidade	Capacidade do sistema robótico fornecer uma solução completa para executar e concluir uma tarefa agrícola, integrar a solução e coordenar a comunicação de cada informação necessária de e para todos os subsistemas.

2.6. Operações Agrícolas com Aplicações Robóticas

2.6.1. Operações de replantação em plantas recém-nascidas

O replante é um estágio preliminar das operações de cultivo e produção e pode ser usado em muitos tipos de cultivo. A implementação da automação em operações de replante é viável em diversas situações, principalmente quando as operações são repetitivas e com poucas semanas de diferença na mesma parcela, como é o caso de vegetais folhosos ou ervas. Quando o transplante envolve grandes áreas ou um trabalho humano intensivo e/ou com alta precisão, os dispositivos de transplante automatizados oferecem muitas vantagens, mas geralmente são complexos e caros [56].

Mao et al. [57] desenvolveram um dispositivo de colheita do tipo pinça, para transplante automático de mudas em estufa. O mecanismo robótico consiste num manipulador, com uma ferramenta e dois transportadores, com uma taxa de transplante de 22 mudas por minuto.

Nagasaka et al. [58], [59] desenvolveram um sistema para transplante automatizado de seis linhas em culturas de arroz (ver Fig. 4). Este sistema é composto por um sistema de localização *Real-time kinematic positioning* (RTK-GPS) e um giroscópio de fibra ótica (FOG) e sensores giroscópicos para medir a direção e inclinação do veículo. O objetivo do sistema FOG é corrigir eventuais erros de leitura do RTK-GPS causados pela influência do movimento e inclinação do veículo na antena do GPS quando o transportador está em movimento.

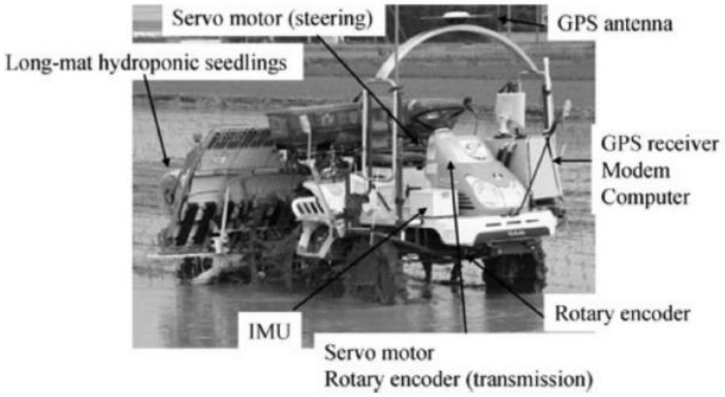


Figura 4 - Sistema de transplante de arroz, automatizado, adaptado de [59].

Com base no trabalho anterior, Nagasaka et al. [60] modificaram o SRA para culturas hidropónicas num sistema do tipo tapete longo, para mudas de arroz, tendo adicionado uma máquina de aspersão de herbicida, resultando num sistema de transplante de arroz totalmente automatizado. Este sistema foi capaz de dirigir em linha reta ao longo do terreno e fazer as curvas em cada extremidade.

Ryu et al. [61] desenvolveram um sistema robótico para o transplante de plantas que estão em tabuleiros alveolares, este sistema é equipado com um manipulador cartesiano de dois motores elétricos lineares, uma ferramenta, um tapete rolante com tabuleiros de encaixe e um sistema de visão. A ferramenta é composta por dois cilindros pneumáticos que ligam a duas pinças que recolhem as plantas recém-nascidas dos tabuleiros para a zona de plantação. Os tapetes rolantes são acionados por servomotores e movem os tabuleiros para a posição desejada. O sistema de visão identificava as células vazias.

2.6.2. Poda e desbaste

A poda de árvores é uma tarefa que exige muito trabalho e a mão-de-obra representa mais de 25% dos custos totais. Os principais objetivos desta tarefa são aumentar a exposição à luz solar, controlar a forma das árvores e remover os galhos protuberantes. Na maioria dos pomares, essa tarefa é realizada uma vez por ano e até 20% dos galhos são removidos seletivamente [49].

Embora não existam muitos relatos na literatura de SRA ou sistema automatizados de desbaste de frutos ou poda de árvores, existem alguns projetos em andamento que envolvem o desenvolvimento de sistemas de mecanização e automação para estes processos [49], [62].

Por exemplo, Morris [63] desenvolveu um sistema para vinhas que permite a mecanização da poda feita no verão, fazendo a remoção das folhas, raleio de rebentos e frutos, garantindo a manutenção das partes visíveis na superfície da videira, mantendo e melhorando assim a

qualidade dos frutos. O uso do sistema completo de mecanização da vinha reduz o trabalho manual em cerca 45 a 62%.

Já A. Bechar et al. [55] desenvolveram um sistema para poda seletiva em árvores. O sistema consiste num:

- Manipulador;
- Câmara colorida;
- Sensor de distância;
- Laser de feixe único;
- Interface Homem – Máquina;
- Ferramenta de corte (serra circular).

A ferramenta de corte, a câmara e o sensor de laser são montados na ferramenta do manipulador e alinhados paralelamente um ao outro, como observado na Figura 5.

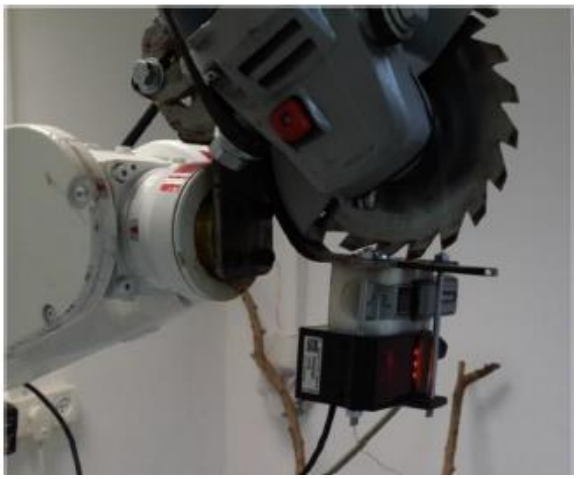


Figura 5 - Ferramenta de Corte, adaptado de [55].

O sistema funciona em duas fases, na primeira fase, a câmara transfere uma imagem 2D da árvore para um Operador Humano, que marca os galhos a serem removidos num display. Na segunda fase, o sistema funciona em modo autónomo. O manipulador manobra o sensor a laser para medir a distância do galho e calcula uma trajetória até o ponto de corte na tela. De seguida, segue a trajetória para realizar o corte. Sendo conduzidos dois tipos de planeamento de movimento entre a localização inicial da ferramenta e o ponto de corte: movimento linear em coordenadas cartesianas globais e no espaço da junta do robô. Este sistema encontra-se em fase de testes.

2.6.3. Colheita

A colheita de frutos é uma das tarefas mais comuns na agricultura e é também uma das áreas mais exigentes e desafiadoras para a robótica agrícola. As investigações sobre colheitas e apanha de

frutas com SRA começou há mais de três décadas e tem-se focado principalmente em campos abertos e plantações de pomares, como citrinos, maçãs, algodão, tomate, melão e melancia [27], [64]–[67].

Têm existido muitos desenvolvimentos na produção de manipuladores e ferramentas específicas para este tipo de operações, como relatam inúmeros estudos [27], [65], [68]–[70].

A taxa atual dos SRA na deteção dos frutos que precisam ser colhidos é de cerca de 75 a 85%, mas estes SRA apresentam ainda dificuldades em evitar obstáculos e em alcançar, agarrar e manusear frutas em tempo real no ambiente agrícola [49].

De salientar ainda que os componentes eletrónicos têm tido grandes melhorias na última década quer ao nível da capacidade de computação, quer os vários tipos de sensores, como os RTK-GPS, câmaras 3D, LIDAR, etc. Estes avanços, têm obviamente contribuído para o desenvolvimento dos SRA na colheita de frutas [39], [71], [80], [72]–[79].

Ceres et al. [27] apresentaram numa abordagem um Agribot, num sistema Robô-Homem que combina funções humanas e de máquina para recolher maçã. O HO deteta e identifica cada fruta a ser colhida usando um joystick e um feixe de laser infravermelho. Este sinal informa ao sistema de medição 3D para calcular a posição do alvo. As coordenadas esféricas, indicam a localização da fruta e incluem dois ângulos ortogonais e um raio, e são geradas dentro de um ciclo de tempo para a medição de posição de 29 ms.

Existem ainda vários estudos em algoritmos de visão máquina para deteção de frutas. Por exemplo [81] desenvolveram um algoritmo para deteção de pepino com base em imagens espectrais NIR, capturando uma imagem mono-espectral de 850 nm para resolver os problemas de segmentação de cores semelhantes num ambiente complexo. Este SRA obteve pontuação para reconhecimento de 83,3% e 100%.

Já Hayashi et al. [82] desenvolveram um robô para colher de beringela baseado num algoritmo de visão máquina, combinando um operador de segmento de cor e um operador de divisão vertical para detetar frutas em diferentes condições de iluminação. Um controlador Fuzzy fornece feedback de tamanho visual para atuar num manipulador e na ferramenta incluída no apanhador de frutas. O ensaio, realizado em condições de laboratório, resultou numa colheita de 62,5% e um tempo de ciclo 64,1 seg/fruto.

Cui et al. [83] desenvolveram um robô do tipo cartesiano para cultura de morangos em linha. O sistema consiste em duas câmaras a cores, um sistema de colheita, um sensor de fibra ótica e uma unidade de controlo móvel. O sistema deteta os morango, avalia sua maturação, colhe os frutos, um de cada vez, cortando os seus pedúnculos. O sistema de deteção na colheita foi de 93,6%, em relação ao nível de maturação desejado foi superior a 50%, enquanto o tempo de ciclo teve o valor máximo de 16,6/s fruto.

Assunção et al. [84] desenvolveram um modelo de deteção de objetos Faster R-CNN para deteção de imagens de pêssegos. Os resultados do modelo mostraram um desempenho relativamente bom, inclusive para frutos agrupados e oclusos. Estes resultados mostraram ainda um grande potencial da aplicação do modelo Faster R-CNN para implementar um sistema de estimação de produção em pomares. Este potencial foi estudado por Cunha et al. [85] que desenvolveu um trabalho de previsão da saúde dos pomares de pessegueiros, em que obteve resultados bastante promissores.

A colheita de frutas é uma das tarefas agrícolas mais complicadas para os SRA. Esta tarefa consiste em várias etapas diferentes e envolve o contato físico com a fruta em alta precisão de localização e orientação, tomada de decisão em tempo real, destacamento da fruta da planta sem danificar nenhuma delas e o armazenamento temporário da fruta em condições seguras [49].

O desempenho dos SRA na apanha de frutas não melhorou substancialmente nas últimas três décadas, de uma forma geral o desempenho na deteção é, bastante aceitável, de 85%, no destacamento uma eficácia 75% e na colheita uma eficácia 66% e um tempo de ciclo de 33 seg/fruta [86].

2.6.4. Controlo de infestantes e monitorização de doenças

Existem inúmeros agentes bióticos e abióticos que afetam o potencial rendimento das culturas. De acordo com Oerke & Dehne [87], cerca de 40% da produção mundial de alimentos é perdida por estes fatores.

As principais ameaças dos agentes abióticos numa cultura são problemas térmicos e de falta de água (alturas de seca). Já no caso dos agentes bióticos, as principais ameaças devem-se a doenças, insetos e pragas que podem afetar as colheitas. Para alcançar bons rendimentos nos sistemas de cultivo agrícola, é essencial a monitorização/controlo destes fatores durante as fases de cultivo e colheita, de modo a detetar e prevenir a propagação de doenças e evitar perdas significativas de produtividade [88].

O controlo de pragas, doenças e infestantes é uma tarefa que tem de ser feita com alguma periodicidade e que consome muito tempo, além disso pode ainda expor o operador ao perigo de contaminação com produtos químicos perigosos. A automação na pulverização ou na remoção de infestantes pode eliminar um trabalho penoso, substituindo por uma máquina flexível, que realiza a operação com qualidade e precisão [88].

O desenvolvimento de sistemas de controlo de infestantes, incluindo a deteção e remoção de infestantes, tem sido um dos principais campos de pesquisa na robótica agrícola nas últimas décadas [88]–[95].

Até o momento, alguns SRA para o controlo completo de infestantes foram testados nas condições de campo real [36].

Os SRA para o controlo de infestantes são compostos, principalmente, por quatro tecnologias:

- Orientação;
- Deteção e identificação de infestantes;
- Remoção precisa de infestantes em linha;
- Mapeamento.

Na remoção das infestantes e respetivo controlo seletivo em linha, existem quatro tipos de mecanismos [36]:

- Meios mecânicos;
- Meios térmicos;
- Meios químicos;
- Meios elétricos.

Destes métodos, o controlo químico, por exemplo, pulverização, é o amplamente mais utilizado.

Slaughter et al. [96] desenvolveram um sistema de pulverização de infestantes para aplicar na beira das estradas. Este protótipo foi utilizado num sistema de visão ligado a um computador que controla o sistema de pulverização e a seleção da formulação do líquido de pulverização. Esse sistema reduziu o uso de pesticidas em até 97%.

Já Nishiwaki et al. [97] desenvolveram um pulverizador com um mecanismo de posicionamento de bico para pulverização de infestantes em campos de arroz. Em que os resultados obtidos eram bons quando os níveis da iluminação natural eram altos.

Oberti et al. [91] desenvolveram um SRA de pulverização seletiva para tratar o oídio (doença fúngica) em videiras. Neste sistema, utilizaram um manipulador robótico 6-DOF equipado com uma ferramenta de pulverização de precisão com um sistema integrado de deteção de doenças baseado em imagens multiespectrais R-G-NIR. Os resultados indicam que o robô detetou e pulverizou 85% a 100% da área afetada dentro da plantação e reduziu o uso de pesticidas entre 65% a 85% quando comparado com o método de pulverização homogénea convencional.

Existem também vários estudos que abordaram o uso de meios mecânicos para o controlo de infestantes. Por exemplo Åstrand & Baerveldt [98], [99] desenvolveram um SRA para controlar as infestantes num campo de produção de beterraba sacarina composto essencialmente por três módulos:

- Módulo de visão para o sistema de orientação;

- Módulo de visão para a deteção de infestantes dentro da linha;
- Enxada rotativa seletiva para a remoção mecânica de infestantes.

O módulo de visão incorporou uma abordagem da fusão dos sensores que combinava o contexto espacial, forma da planta e recursos de cor para melhorar o reconhecimento. Este SRA foi capaz de seguir a linha por si só e remover seletivamente as infestantes entre as plantas. Os resultados indicaram que a abordagem de fusão de sensores é um método robusto para lidar com variações na aparência das plantas e espécies de infestantes, mas apenas quando as plantas da cultura são grandes e a densidade de infestantes é bastante baixa. Os resultados do teste de campo em plantações de beterraba sacarina mostraram que 99% das plantas cultivadas não foram removidas e que cerca de 41-53% das infestantes foram removidas.

Schor et al. [100] desenvolveram um robô para a monitorização de doenças em plantações de pimenta em estufas, tendo utilizado câmaras RGB e multiespectrais e um sensor de feixe de laser montado no manipulador robótico. O sistema detetava oídio (doença fúngica) e o vírus da murcha-manchada do tomate com um desempenho na deteção de 95% e 90% respetivamente, com um tempo de ciclo médio de 26,7 seg/planta.

Assunção et al. [101] desenvolveu um sistema de apoio à tomada de decisão na classificação de doenças em pêssegos. Pois e de acordo com os autores hoje em dia a deteção de características de doenças em frutas é feita, de forma geral, artesanalmente. Assim os autores propõe, através da utilização de redes neurais convulsionais (CNNs), a criação de uma pequena e eficiente rede convolucional para funcionar em dispositivos móveis e que classifique o estado dos pêssegos em quatro estados (Figura 6).

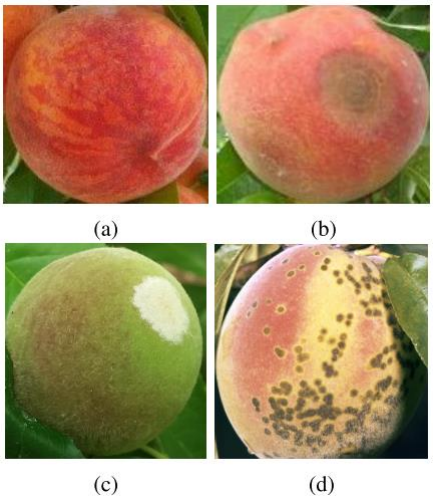


Figura 6 - Imagens de pêssegos e as classes existentes: (a) Saudável (b) Podre (c) Mofo (d) Sarna, adaptado de [101].

Esta técnica funcionou com a transferência de aprendizagem e estratégias de aumento de dados, tendo o modelo proposto atingido uma pontuação de 0,96 no F1 da Macroaverage. Referir ainda que o modelo não classificou erroneamente nenhuma classe de doença. Esta conquista mostra o potencial de usar pequenos modelos CNN na classificação de doenças de frutas.

Por todos os exemplos dados acima verifica-se que têm existido um progresso importante no controlo de infestantes e na monitorização de doenças nas últimas décadas. Estes avanços devem-se principalmente às melhorias do *hardware* dos sensores, que resultou numa diminuição dos erros de posicionamento e num aumento do desempenho da deteção e numa melhor tomada de decisão, o que melhorou também o tempo de operação [91].

No entanto e segundo Vigneault & Bechar [49] para alcançar um desempenho satisfatório, existem ainda algumas lacunas que precisam ser abordadas:

1. Aumentar o desempenho na deteção e a respetiva confiabilidade em outras variedades de infestantes e também ser adaptado a ambientes não estruturados e dinâmicos;
2. Replicar a aplicação de alta precisão a ambientes não estruturados e dinâmicos para minimizar custos, estendendo o uso do robô a todo o tipo de superfícies agrícolas;
3. Melhorar a totalidade, ou integração de todos os subsistemas, para permitir um desempenho sustentável.

2.7. Métodos para o Controlo de Direção de Veículos Autónomos

A criação de um sistema de controlo de direção de veículos autónomos (VA) é um dos maiores desafios para a automação, devido às várias restrições inerentes da mobilidade [102] e por isso a literatura tem proposto inúmeros métodos para automatizar e melhorar a eficiência dos sistemas de controlo de direção dos VA.

Numa análise realizada por Rasib et al. [103] aos diferentes métodos que têm sido criados, adaptados e melhorados ao longo dos últimos anos para o controlo de direção dos VA, o autor propõe a categorização destes métodos, criando dois grandes grupos os Determinísticos e os heurísticos/híbridos, em que cada um destes terá um conjunto de subgrupos conforme demostrado na Figura 7.

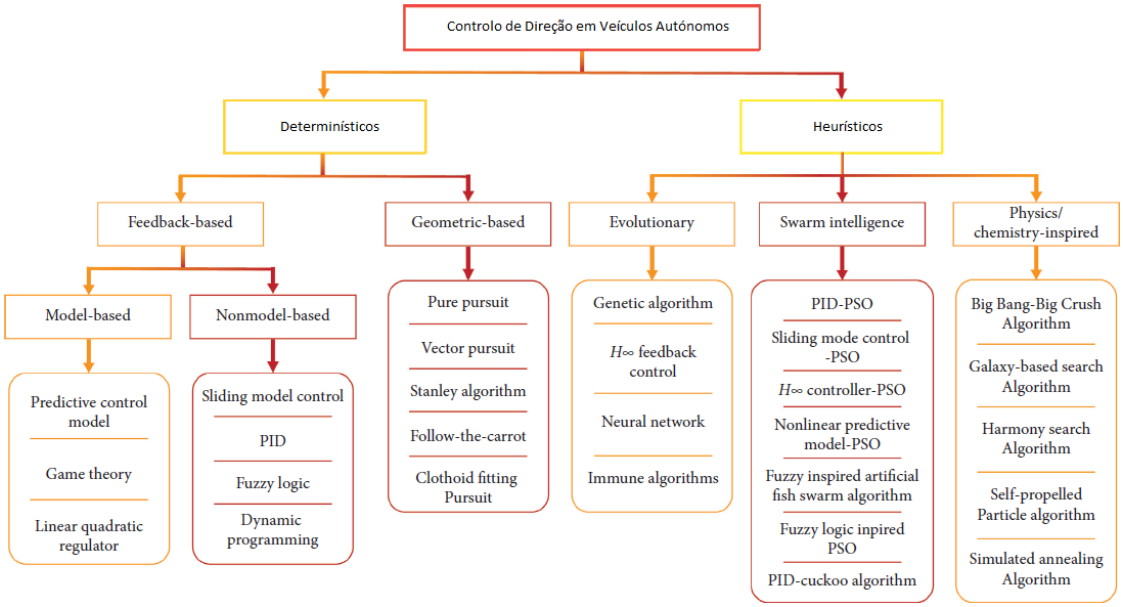


Figura 7 – Categorização dos vários métodos de controlo de direção em VA, adaptado de [103].

2.7.1. Métodos determinísticos

De acordo com Siddique & Adeli [104], os métodos determinísticos produzem sempre o mesmo *output* em relação à condição do *input* dado. Estes métodos são clássicos e sem aleatoriedade e também com menor capacidade de exploração simultaneamente [105]. Os modelos matemáticos determinísticos fornecem uma solução única que representa os resultados de certas entradas da experiência.

Além destes métodos produzirem o mesmo *output* para o *input* inicial dado, também consomem a mesma quantidade de tempo, memória e recursos todas as vezes que repetem o input correspondente. Os métodos determinísticos podem dividir-se em duas categorias, que são os métodos baseados em geometria e os métodos baseados em feedback [102].

2.7.1.1. Métodos de base geométrica

Na literatura são apresentados vários métodos de base geométrica para automatizar o sistema de condução dos veículos autônomos. Os controladores apresentados estabelecem uma relação geométrica entre a posição atual do veículo e o caminho de referência correspondente para estimar o ângulo do volante. Os métodos geométricos mais utilizados são "*pure pursuit*", "*Stanley*" e "*Vector Pursuit*" [106].

2.7.1.2. Métodos baseados em feedback

São métodos de controlo de direção baseados no erro de feedback que geralmente se concentra na entrada e na saída do sistema e ignora a lei do movimento. Estes métodos compreendem um

feedback de estado linear em desvios direcionais, desvios de caminho lateral e outros derivados. Uma das principais vantagens deste controlador é que um controlador de feedback bem executado não é sensível quando comparado com métodos de controlo de direção de base geométrica [102]. Este tipo de controlador é categorizado, na literatura, em dois tipos: "*Model-based"* e "*Nonmodel-base*"

Os "*model-based"* são essencialmente modelos matemáticos, amplamente utilizados para determinar estratégias no controlo dos veículos autónomos. Em comparação com os métodos geométricos acima mencionados, esses métodos consistem em estruturas matemáticas mais complexas. As abordagens que estes utilizam fazem todas as previsões possíveis sobre o domínio do problema na forma de um modelo, e esse paradigma ajuda a compreender e resolver o problema real. Os controladores de "*model-based"* mais utilizados são "*Model Predictive Control"* (MPC), "*Linear Quadratic Regulator"* (LQR) e *Game Theory* [103].

Os "*Non model-base*" são controladores baseados no conceito de otimizar a recompensa antecipada da experiência real, sem considerar o modelo ou a experiência anterior. O objetivo principal destes métodos é obter as recompensas associadas às ações atuais e correspondentes. Estes controladores não requerem nenhum detalhe essencial em relação aos parâmetros de atuadores ou manipuladores; portanto, nenhum modelo matemático para o manipulador é necessário na execução destes sistemas. O tipo de controladores mais utilizado neste tipo são os "P*roportional-Integral-Derivative" (PID),* "*Fuzzy Logic*", "*Dynamic Programming"* e "*Sliding Mode Control"* [103].

2.7.2. Métodos heurísticos

De acordo com Siddique & Adeli [104], os métodos heurísticos caracterizam-se por produzir sempre outputs diferentes para os *inputs* dados nas diferentes simulações que se realizem.

Os métodos heurísticos são inerentes a alguma aleatoriedade, no entanto as suas capacidades de exploração são aprimoradas, mas as capacidades de operação reduzidas [105]. Os valores dos parâmetros e as condições iniciais de um sistema heurístico contribuem para uma série de resultados diferentes. Além disso, em contraste com o método determinístico, estes métodos podem mostrar comportamentos diferentes em execuções diferentes para a mesma entrada.

Estes métodos não são preditivos porque o resultado de um processo não é determinado objetivamente, devido à falta de compreensão de uma relação de causa/efeito ou à incapacidade de reconhecer as condições iniciais. No entanto, esses métodos são mais proficientes em lidar com problemas difíceis (ou seja, problemas que não têm soluções conhecidas em tempo polinomial) [104].

Há uma variedade de métodos heurísticos inspirados na natureza, que são relatados na literatura e que são categorizados por [103] em métodos Evolucionários Bioinspirados "*Bioinspired*

Evolutionary", Inteligência de Enxame "*Swarm Intelligence"* e de Inspiração Físico-Química "*Physics-Chemistry Inspired"*.

2.7.2.1. Métodos evolucionários bioinspirados

Os métodos Evolucionários Bioinspirados "*Evolutionary Algorithms"* (EAs) são métodos de otimização de busca heurística, paralelos, globais ou baseados em custos, sendo baseados nos princípios da natureza introduzidos por Darwin em 1859. Os EAs são algoritmos bem conhecidos e são inspirados na evolução biológica da natureza que é responsável pela criação de todos os seres vivos na Terra e as técnicas que estes usam para comunicarem entre si. Este conceito teórico é usado para encontrar soluções para problemas desafiantes.

Os principais *EAs* utilizados no controlo de direção dos veículos autónomos são os algoritmos genéticos "*Genetic Algorithms*", controlador *H*∞ "*H*∞*Controller*", Neuro-evolutivo "*Neuroevolution*" e algoritmo imune "*Immune Algorithm*" [107].

2.7.2.2. Métodos de inteligência de enxame

Os *métodos de Inteligência de Enxame "Swarm Intelligence"* (SI) são utilizados como técnicas de otimização dos controladores de direção dos carros autónomos para remover o procedimento de tentativa e erro, sendo que muitas áreas nos carros autônomos requerem a resolução de vários problemas usando técnicas de otimização [108].

Enquanto os métodos analíticos sofrem com os problemas da dimensionalidade e da convergência lenta, as técnicas baseadas em SI podem ser uma alternativa muito coerente [103]. Diferentes técnicas pertencentes à família SI e que resolvem problemas complexos, desde o planeamento até o controlo do veículo, são o controlador Proporcional-Integral-Derivativo com otimização por enxame de partículas "*PID-PSO", controlo de modo de deslizamento com otimização por enxame de partículas "Sliding mode control-PSO" e* modelo preditivo não linear com otimização por enxame de partículas "*Nonlinear predictive model-PSO"* [109].

2.7.2.3. Métodos de inspiração físico/química

Os algoritmos heurísticos não são bioinspirados, à exceção dos sistemas de controlo de direção de Inspiração Físico-Química **"***Physics/chemistry-inspired".* Embora a física e a química sejam áreas separadas as leis fundamentais de ambos são idênticas, podendo assim ser simplesmente agrupados como algoritmos baseados na física-química [103].

Os algoritmos deste grupo mais descritos na literatura são o "*Big Bang-Big Crunch" (BB-BC)* e o "*Harmony search" (HS)* e desempenham um papel vital na otimização de controladores de direção. O BB-BC é baseado no teorema hipotético da criação e destruição do universo, este método fornece melhores resultados de otimização enquanto converge para a solução ótima

definida com uma alta velocidade de convergência, menor tempo computacional e baixo custo de computação [110].

O método *HS* foi introduzido por Geem et al. [111] em 2001. A função da improvisação e a aplicação de engenharia baseia-se em fenômenos musicais e existem muitos exemplos na literatura da utilização deste sistema para problemas de otimização [112].

As principais vantagens deste método é ter menos parâmetros para ajustar, fácil implementação e uma capacidade de convergência rápida [113]. No entanto, este método apresenta alguns defeitos, como velocidade de convergência lenta e convergência prematura. [103].

2.8. Nota Conclusiva

O interesse por robôs móveis tem tido um crescimento notório, levando a uma enorme quantidade de trabalhos desenvolvidos nos últimos 30 anos em aspetos críticos como a locomoção, perceção, localização, mapeamento, rastreamento de movimentos de sistemas 3D e navegação dinâmica [114].

No entanto e como se pode perceber ao longo deste capítulo, a utilização de robôs na área agrícola ainda tem um longo caminho a percorrer até à conceção de sistemas robóticos que consigam realizar os diversos trabalhos de forma completamente autónoma.

Observou-se ainda que já existem alguns trabalhos em operações de controlo de infestantes, pelo que o pretendido com o trabalho que agora se apresenta é que este possa dar mais um contributo nesta área. Quer para a pulverização de precisão, quer para a apanha de frutas caídas no chão das áreas agrícolas que prejudicam a qualidade dos solos.

Foram ainda apresentado os vários tipos de controladores/algoritmos que existem para o controlo de direção dos VA. Importando referir que para o tipo de plataforma a utilizar e o tipo de trabalho a executar os controladores mais relatos na literatura são os controladores do tipo "*PID*" [115]–[117], Lógica Fuzzy "*Fuzzy Logic"* [118]–[120], Algoritmos Genéticos "*Genetic Algorithms"* [121], [122], "*LQR"* [123], [124], "*Sliding mode control-PSO*" [125], [126] e Controlador *H*∞ "*H∞Controller*" [127], [128]. Existindo ainda trabalhos que relatam o uso de controladores do tipo Redes Neurais "*Neural Networks"*, Lógica Fuzzy-Neurais "*Neuro-Fuzzy logic"* e "*PID-PSO"* [129]–[131].

3. Materiais e Métodos

Neste capítulo é apresentado o rover robótico em que se baseia todo o trabalho de simulação. Sendo também discutido o funcionamento dos simuladores robóticos e como estes podem ser úteis no desenvolvimento de algoritmos para o controlo de sistemas robóticos, neste caso para o controlo do rover robótico em estudo.

Adicionalmente é apresentada uma descrição detalhada sobre o simulador robótico escolhido para o trabalho que se propõe realizar, bem como as diferentes funcionalidades de que dispõe.

3.1. Rover Robótico Existente

Veiros [12] na fase de conceção do rover robótico começou por definir um conjunto de premissas que este teria de cumprir, nomeadamente:

- Recolher os frutos caídos no chão do pomar;
- Altura máxima: 50 cm, para que lhe seja possível passar por baixo das copas das árvores sem danificar ramos e frutos;
- Carga útil: ±2 kg caixa da fruta + 50 pêssegos ≈ 170g/cada → 8,5 kg;
- Comprimento e largura não definidos, mas a premissa inicial era a criação de um robô de pequenas dimensões, ágil e bastante manobrável, facilitando a deslocação entre árvores.

Definidas estas premissas, importava agora criar uma plataforma robótica que tivesse também uma construção fiável e que fosse economicamente viável. Assim, Veiros [12] realizou todo o processo de conceção inicial no *software* SolidWorks. Este é um *software* de desenho assistido por computador (CAD) e engenharia assistida por computador (CAE), que entre muitas funcionalidades, destaca-se a possibilidade de criar e desenhar componentes, conjuntos e desenhos técnicos [132].

Veiros [12] indica que este processo foi importante, pois até atingir um modelo final foi necessário realizar várias alterações na idealização do rover robótico até porque, nos primeiros desenhos, alguns componentes são apenas representações feitas com o intuito de se ter uma noção das dimensões e distribuição dos componentes. O autor refere ainda que a criação do modelo 3D foi bastante útil pois possibilitou o melhoramento de componentes e a deteção de erros e falhas na fase de projeto, podendo assim ser corrigidos antes de iniciar a construção do protótipo. De referir

que permitiu ainda a elaboração de uma lista do material necessário, fazendo com que a construção não fosse interrompida por falta de peças e componentes.

Na Figura 8 é possível ver o desenho do *layout* final que foi adotado para a construção do rover robótico.

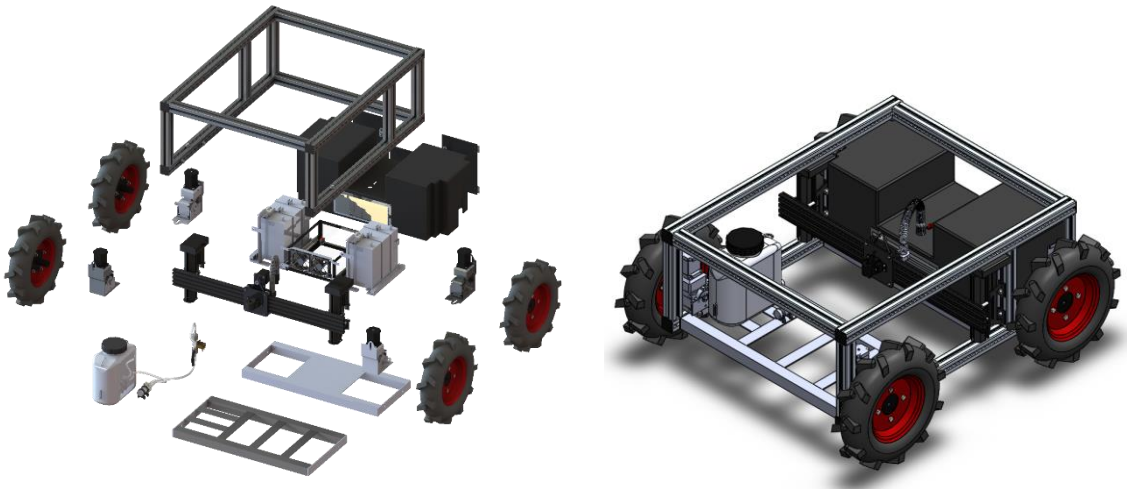


Figura 8 - *Layout* da versão final do rover robótico, adaptado de [12].

Definida a soluço final a construir, pode-se constatar, que o rover robótico terá 2 rodas fixas paralelas e longitudinais de cada lado da estrutura (plataforma do tipo skid-steer) e que de acordo com Secchi [133], a estrutura morfologia é a descrita na Tabela 4.

Tabela 5 - Estrutura morfológica do rover robótico.

Tipo de Morfologia	Opção adotada
Tipo de Ambiente para operação:	Não-estruturado
Tipo de sistema de locomoção:	Robô Terrestre com rodas
Tipo de rodas:	Roda Fixa
Disposição das rodas:	Quadriciclo
Tração e direção:	Tração e direção no mesmo eixo (Tração diferencial)

Feita esta conceção e para poder selecionar os vários componentes a utilizar neste rover robótico, o autor realizou um dimensionamento de todos estes componentes. Sendo feita de seguida uma breve descrição dos mesmos.

Para os motores e caixas redutoras foram selecionados 4 motores de passo, NEMA 23 com controlador integrado, conseguindo produzir um binário de 1 Nm, a 600 rpm, com alimentação a 24 V. Já para as caixas redutoras, optou-se por uma caixa redutora sem fim com um rácio de redução de 25, ficando assim a saída com 25 Nm. Na Figura 9 pode-se observar o conjunto motor/caixa redutora selecionados.

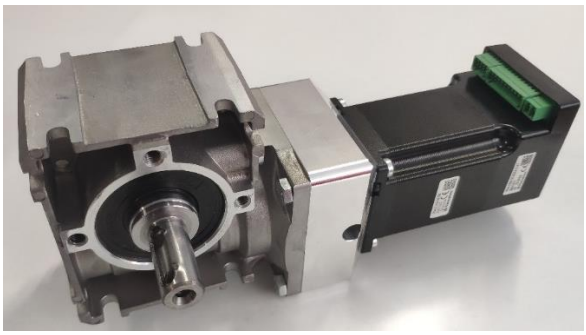


Figura 9 - Conjunto motor/caixa redutora selecionado, adaptado de [12].

Para a estrutura do rover optou-se por utilizar perfis T-slot de alumínio, pois estes apresentam uma boa relação peso/resistência. A medida selecionada foi de 45x45 por ter uma boa base para uma fácil instalação dos suportes dos motores, visto estes apertarem diretamente na estrutura. Outra vantagem da aplicação deste tipo de perfil é a facilidade que existe na alteração de fixações, bem como na introdução de novos componentes quando necessário.

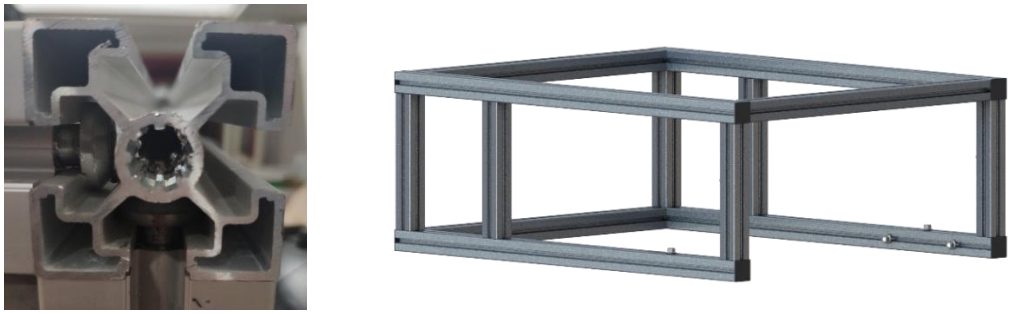


Figura 10 – Detalhe do perfil de alumínio, adaptado de [12].

Para garantir a autonomia na realização das várias tarefas no campo agrícola o mesmo foi equipado com 2 baterias gel de 12v 55Ah, a escolha pelas baterias gel deveu-se principalmente à boa relação custo/desempenho. Este sistema permite uma autonomia de 2,5 horas ao veículo.

Foi ainda desenvolvida uma caixa para proteção das baterias, tendo esta caixa sido feita em poliestireno extrudido envolvido por fibra de vidro, criando assim um material tipo “sandwich”, extremamente resistente.

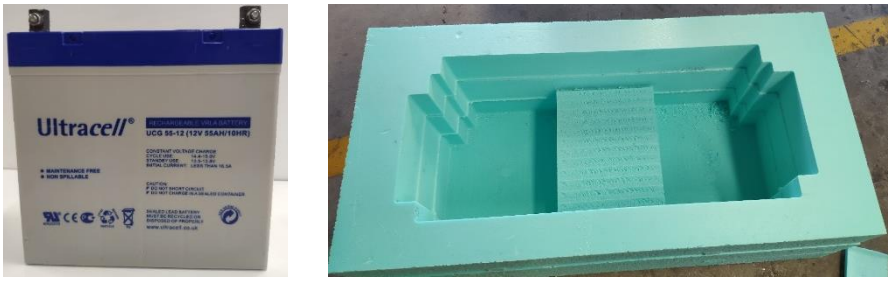


Figura 11 – Bateria de gel de 12v 55 Ah e Caixa para Proteção das baterias selecionadas, adaptado de [12].

Neste momento, e após a realização de ensaios experimentais, verificou-se que o peso elevado das duas baterias de gel (2x17 kg), condicionavam a locomoção do rover robótico em terreno agrícola. Assim sendo, optou-se por usar 2 baterias de LIPO de 10000 mAh 18,5V, com um peso total de 2x1,1 kg.

Para o controlo do robô, optou-se por utilizar um Arduino Mega e um Rasperry Pi 3b+, sendo que o Arduíno controla toda a componente de locomoção e o Rasperry Pi a componente de deteção de frutos e infestantes, bem como o posicionamento dos eixos lineares e da garra e gere ainda o sistema de pulverização. A supervisão, comando das operações e comunicação entre os dois microcontroladores e realizada por ESP32.

Já o sistema de pulverização é composto por uma pequena bomba de água, um depósito com capacidade para 5 litros, um solenóide e um bico de pulverização (ver Figura 12). A bomba tem uma tensão de alimentação compreendida entre 6 a 12 V e produz um caudal máximo de 3 l/min, sendo o suficiente para alimentar o sistema. Dadas as semelhanças entre este sistema e o do pulverizador elétrico, foram adquiridos componentes com caraterísticas similares para o sistema desenvolvido.

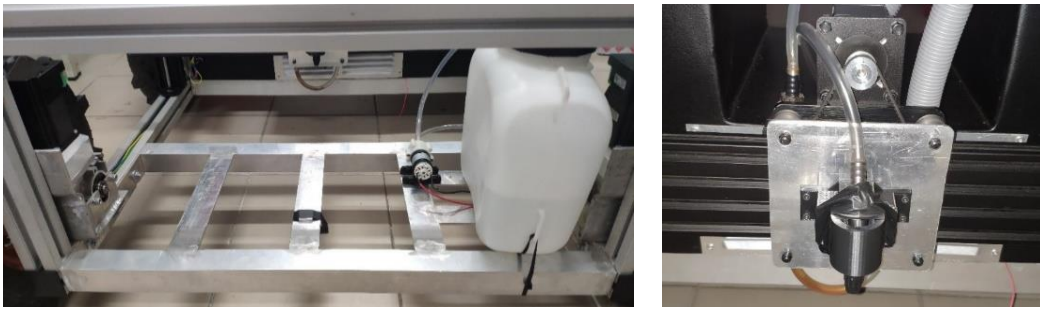


Figura 12 - Suporte traseiro para e Bico de Pulverização.

Feito o dimensionamento e seleção de todos os componentes robóticos, foi feita a montagem de todos estes componentes, podendo o resultado final da construção deste rover robótico ser observado na Figura 13.

Figura 13 - Resultado final do Prótótipo construido, adaptado de [12].

3.1.1. Eixos de funcionamento do rover robótico

Apresentada toda a plataforma do rover robótico, importa agora especificar os eixos lineares que foram criados nesta plataforma e que servem ou para a pulverização de precisão ou para a recolha de frutos caídos no campo agrícola.

A plataforma possui três eixos lineares, podendo o envelope de trabalho ser verificado na Figura 14.

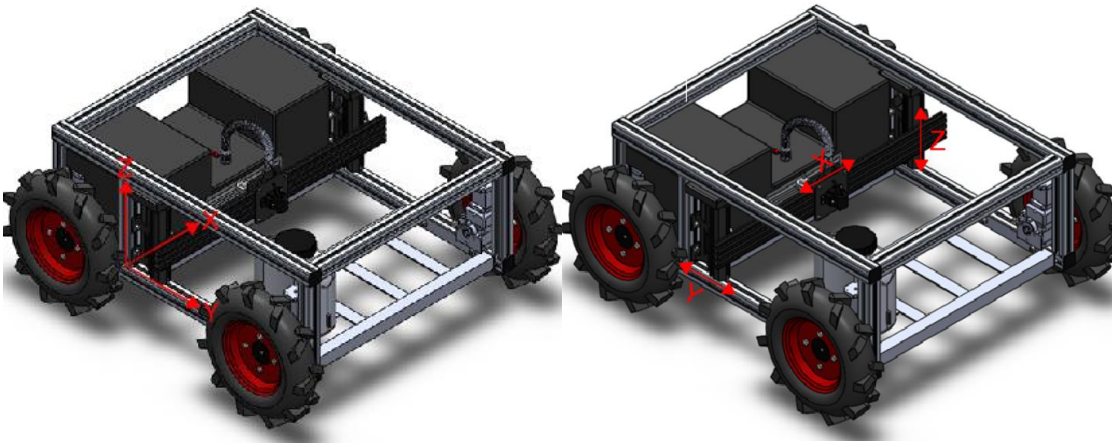


Figura 14 – Envelope de trabalho do rover robótico, adaptado de [12].

Na Figura 14 verifica-se que no suporte do eixo X está instalado o bico pulverizador. No entanto e consoante o trabalho a executar este poderá ser substituído por uma garra mecânica para a recolha de objetos. De referir ainda que no caso da pulverização de precisão serão utilizados mais os eixos X e Z para colocar o bico pulverizador no exato local que se pretende pulverizar. Já para a recolha de objetos será necessário utilizar também o eixo Y para levar o objeto capturado até ao cesto que estará instalado no suporte traseiro onde antes estaria o depósito da solução utilizada na pulverização.

3.2. Simuladores Robóticos

A validação de projetos na área de robótica é uma tarefa complicada. Para apoiar e facilitar esta tarefa existem atualmente inúmeras ferramentas, como simuladores 3D [134].

Um simulador robótico é um programa capaz de criar um ambiente virtual onde se podem desenvolver, visualizar e alterar programas robóticos. Neste ambiente virtual é possível simular o comportamento de um robô para que este possa ser visualizado pelo programador que pretende testar os seus programas. [135].

Esta tecnologia de simulação torna-se cada vez mais corrente entre os fabricantes de robôs e produtores de *software* responsáveis pela criação destes simuladores, pois trata-se de uma tecnologia que permite de forma rápida e acessível a realização de testes nos sistemas robóticos. Sendo que atualmente já todos os grandes fabricantes disponibilizam simuladores para o efeito [135].

As vantagens que estes simuladores trazem são inúmeras, sendo destacadas abaixo algumas das referidas por Faria e F.M. Teixeira [134], [136].

- Custo de desenvolvimento muito reduzido, o custo inicial é apenas ao nível intelectual, pelo menos até haver investir em materiais/*hardware*;
- Possibilidade de testar diferentes cenários e hipóteses na produção e validação de *software* ou algoritmos sem a existência de um *hardware*;
- Aumento da liberdade e criatividade, uma vez que não existem preocupações relativas à danificação do *hardware*;
- Capacidade de realizar várias iterações rapidamente (enquanto num cenário real seria necessário preparar o sistema e o ambiente no qual este se situa);
- Pode ser flexível e dinâmico, adaptando sensores específicos para melhorar os resultados;
- Oportunidade de testar várias hipóteses simultaneamente (várias simulações a correr em paralelo).

Analisado os inúmeros *softwares* de simulação que são descritos na literatura científica, foram pré-selecionados um conjunto de *softwares* que mostraram ser adequados ao tipo de trabalho que se pretende desenvolver.

Na escolha destes *softwares*, foi tido em consideração as principais características que F.M. Teixeira e G.E. Teixeira [135], [136] referem ser necessárias:

- **Linguagens Suportadas -** linguagens de programação para o desenvolvimento de programas robóticos;
- **Motor de Física -** Tecnologia que permite a simulação de sistemas físicos mais próximos da realidade;

- **Manipulação de malhas -** Conjunto de vértices, arestas e superfícies que definem a forma de um objeto. É desta forma que se representam os vários objetos, incluindo robôs. Em alguns simuladores, a malha pode ser alterada e manipulada de diferentes formas;
- **Bibliotecas -** Existência de modelos de robóticos no simulador que podem ser explorados na sua totalidade, possibilitando uma aprendizagem rápida. E que tenha ainda outros tipos de componentes como tapetes rolantes, sensores, ferramentas e controladores.

De seguida são apresentados os *softwares* mais relevantes e mais indicados para o tipo de trabalho que se pretende desenvolver.

3.2.1. Gazebo

O simulador Gazebo foi desenvolvido em 2002, pela Open Source Robotics Foundation. É um simulador multi-robôs pois tem a capacidade de simular vários robôs, objetos e sensores. Em 2009 foi integrado o ROS (*Robot Operating System*) e desde aí que é um dos simuladores mais utilizado por essa comunidade. Este permite a comunicação através de fichas comunicação, facilitando a integração com scripts em vários idiomas. Uma das vantagens na utilização desta ferramenta é que dispõe de um bom suporte ao nível da documentação, uma grande biblioteca de elementos de cena, e um site com tutoriais completos e exemplos [135], [137].

3.2.2. Webots

O simulador Webots foi criado em 1996 pela Swiss Federal Institute of Technology (EPFL), sendo atualmente gerido pela Cyberobotics. É um simulador muito utilizado na área da indústria, educação e investigação. Oferecendo um ambiente completo para o desenvolvimento modular, de programação e de simulação de robôs. É possível construir novos modelos ou importá-los de CAD ou do URDF (*Unied Robot Description Format*). A Webots possui uma vasta biblioteca de componentes, desde: robôs de mesa de duas rodas, armas industriais e veículos subaquáticos autónomos. Possui ainda vários componentes ao nível de sensores, atuadores, objetos e materiais. [135], [137].

3.2.3. ARGoS

O simulador ARGoS (*Autonomous Robots go Swarming*) foi desenvolvido inicialmente pelo projeto Swarmanoid, e mais tarde financiamento pela ERC Advanced Grant E-SWARM e pelo projeto ASCENS [138]. Este *software* foi pensado para simulações em que é necessário colocar um grande número de robôs em simulação de forma simultânea. As principais características são:

- Possuir uma arquitetura modular que possibilita ao utilizador configurar todos os aspetos da simulação;

- Permitir o uso de vários motores de física na mesma simulação em simultâneo;
- Distribuir a rotina principal da simulação por múltiplas *threads*, o que leva a um melhor aproveitamento das capacidades dos processadores multi-core.

Os robôs simulados no ARGoS podem ser programados, entre outras linguagens, em C++ e Lua (scripts). O simulador conta com uma pequena biblioteca de modelos simples de robôs e uma das grandes desvantagens é não deixar importar outros modelos. Caso seja necessário, é preciso modelá-los com o OpenGL. Sendo a complexidade deste aspeto compensada pelo desempenho nas simulações que envolvem uma grande quantidade de robôs [136], [138].

3.2.4. CoppeliaSim

A história deste simulador começou com o nome V-REP (*Virtual Robot Experimentation Platform*) que foi criado por Marc Freese e lançado em 2010. Desde aí, Mark Freese criou a "*Coppelia Robotics AG"*, empresa responsável pela gestão e melhoria do *software*, tendo-o tornado num dos simuladores mais modernos disponíveis [137].

Desde o lançamento deste *software* já saíram várias versões, sendo que em novembro de 2019, no lançamento da quarta versão, foi substituído o nome para CoppeliaSim. Este *software* pode ser usado para o desenvolvimento de protótipos rápidos, simulação de sistemas de automação e ensino.

O programa possui uma grande coleção de robôs e sensores existentes no mercado, possuindo ainda a capacidade de importar novos modelos ou criá-los usando as capacidades de modelação integrada. Sendo que se for utilizado o ROS é até possível conectar em robôs existentes [137].

Este simulador permite ainda o controlo de robôs através de seis tipos de programação, sendo todos mutuamente compatíveis na simulação. Esta possibilidade deve-se à interface de programação de aplicações que o CoppeliaSim dispõe e que permite uma integração de tópicos ROS e *BlueZero,* criação de interfaces de programação da aplicação (*Application Programming Interface*s – APIs) para clientes remotos, *Plug-ins*, entre outras formas de programação. Uma vez que o CoppeliaSim permite testar programas criados em plataformas diferentes, torna-se assim numa solução bastante versátil. Para além disso dispõe ainda de quatro motores de física e possibilita ainda deteção de colisões, cálculo de distâncias, simulação com sensores de proximidade e sistemas e visão. Por todas estas funcionalidades que o *software* dispõe é muitas vezes apelidado de "canivete suíço" na comunidade de simulação robótica. [135], [139].

3.2.5. Comparativo dos simuladores existentes

Todos estes simuladores demonstraram integrar as funcionalidades e especificações necessárias para o trabalho que se propõe realizar, existindo em alguns casos características muito idênticas

entre simuladores. Para ajudar na seleção, foi realizada uma tabela síntese com as principais características de cada um destes simuladores.

Tabela 6 - Comparativo dos simuladores robóticos, adaptado de [134], [139], [140].

Características	**Simuladores**			
	Gazebo	**Webots**	**ARGoS**	**CoppeliaSim**
Motor de Física	ODE, Bullet, Simbody, DART	Proprietary based on ODE	Multiple-physics engines	ODE, Bullet, Vortex, Newton
Sistemas operacionais suportados/ *Software* suportado / compatível	Linux, MacOS, Windows	Linux, MacOS, Windows	Linux, MacOS	Linux, Mac OS, Windows
Linguagem de Programação	C++	C++	C++	LUA
Suporte a arquivos CAD / Suporte a ficheiros CAD	SDF/URDF, OBJ, STL, Collada	WBT, VRML, X3D	Não suporta	OBJ, STL, DXF, 3DS, Collada,URDF
Suporte API	C++	C, C++, Python, Java, Matlab, ROS	C++	C/C++, Python, Java, Urbi, Matlab/Octave
Suporte ROS	SIM	SIM	SIM	SIM
Conexão com outras aplicações	ROS, TCP, UDP, entre outos	ROS, TCP	ROS	ROS, TCP, UDP, entre outos
Licença	Grátis	Grátis com acesso limitado	Grátis	Grátis para usos não comerciais ou educacionais

Apresentadas todas estas características, conclui-se que no caso do ARGos é um simulador mais virado para aplicações robóticas que simulam uma grande quantidade de robôs em simultâneo. Têm motores de física próprios podendo não estar otimizados. No entanto, em termos de desempenho, para uma cena pequena com 5 robôs este simulador apresenta melhores resultados do que o Gazebo e o CoppeliaSim [140], [141].

Porém o ARGos tem como desvantagem o facto de ter pouca documentação e uma comunidade de utilizadores pequena o que ia tornar a aprendizagem do *software* mais difícil, além disso a simulação que se pretende realizar terá apenas um robô pelo que não se iria tirar partido das principais vantagens deste simulador.

Em relação ao Webots, este demonstrou cumprir todos os requisitos necessários, tem uma enorme lista de componentes para utilizar e tem também a possibilidade de adicionar novos

componentes. No entanto tem a desvantagem de a licença gratuita ter muitas limitações, para a simulação em causa seria necessário a versão *premium*.

Posto isto restam os simuladores CoppeliaSim e o Gazebo, as principais diferenças que foram observadas entre estes, foram:

- Existem restrições no Gazebo em relação aos pontos de edição em malhas 3D [142];
- O CoppeliaSim é um simulador que consome mais recursos, no entanto a capacidade de gerar automaticamente novos *threads* nos vários núcleos do CPU e tirar assim proveito da quantidade total de energia do CPU, é um aspeto que o Gazebo falha em exibir [142];
- Gazebo é um simulador projetado para trabalhar nativamente com o ROS. No entanto as funções do CoppeliaSim de comunicação com o ROS são suficientes para garantir uma boa integração [143];
- Ivaldi et al. [144] realizou um estudo comparativo entre vários simuladores de robóticos existentes e o CoppeliaSim foi considerado o que teve maior taxa de satisfação;
- No caso do Gazebo [141], [142] relatam alguns crashes inesperados durante a utilização desta ferramenta.

Analisados os pontos anteriores optou-se por utilizar o CoppeliaSim visto que para além destes pontos, esta ferramenta dispõe de vários recursos como motores de física, uma biblioteca de modelos abrangente, a capacidade de interagir com o ambiente durante a simulação e o mais importante a manipulação e otimização das malhas.

Referir ainda que Nogueira [145] comparou estes dois simuladores em:

- Integração com o ROS;
- Modelação de cenário;
- Modificação dos modelos de robôs existentes;
- Controlo programático e utilização da CPU.

E conclui que o CoppeliaSim apresenta uma maior eficiência nos diversos tópicos em relação ao Gazebo. A única desvantagem foi na integração do ROS, pois o Gazebo é um simulador projetado para trabalhar nativamente em ROS. Todavia, as funções do CoppeliaSim de comunicação com o ROS são suficientes para garantir uma boa integração.

A escolha é também justificada pela facilidade de adaptação ao mesmo, algo possível devido à interface gráfica muito simplificada e "*user-friendly*" e por toda a documentação, exemplos e tutoriais disponibilizados pela Coppelia Robotics e ainda pela grande comunidade e muito ativa, que partilha experiências e que proporciona muitos tutoriais, quer no fórum oficial da Coppelia Robotics quer em algumas redes sociais como o Youtube.

3.3. Funcionalidades do CoppeliaSim

Antes de iniciar o trabalho de recriação do rover robótico no simulador escolhido e a criação de um algoritmo para o controlo da mesma, importa primeiro conhecer as várias funcionalidades deste simulador. Toda a informação apresentada de seguida sobre as várias funcionalidades do simulador foi retirada do manual de utilizador, site e fórum da Coppelia Robotics [146]–[148], salvo indicação contrária.

3.3.1. Interface gráfica

A Figura 15 apresenta a janela da aplicação onde é possível ver, editar, interagir e simular a "*cena*" e os seus respetivos modelos.

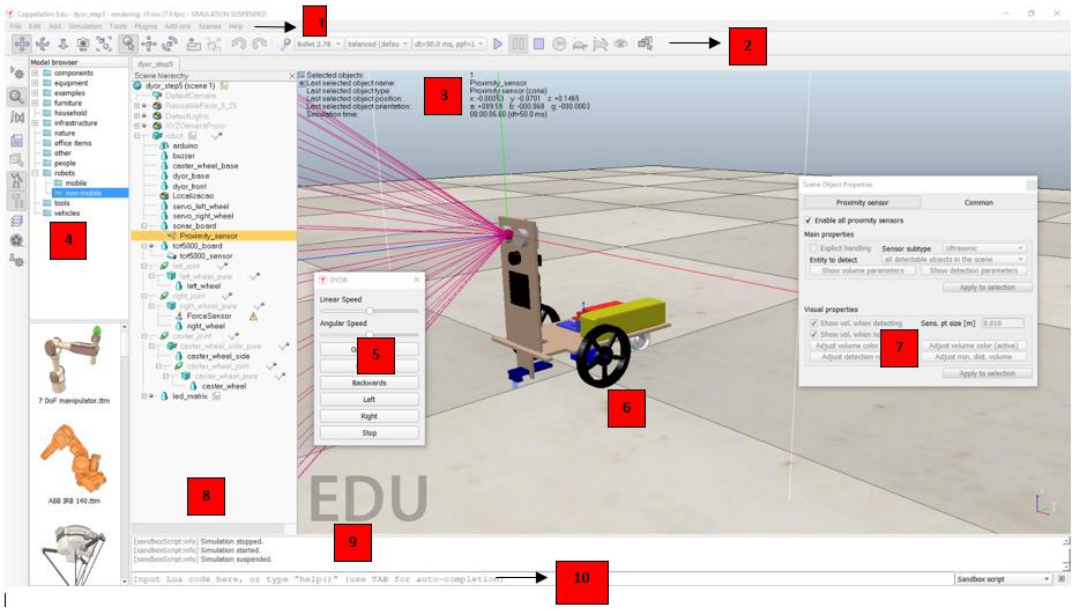


Figura 15 – Interface Gráfica do simulador.

Além desta, existe ainda a janela da consola, que é um elemento não interativo e que permite apenas mostrar informação relativa à simulação (ex.: *plug-ins* que foram carregados e o estado da sua inicialização), o que para o trabalho em causa não é relevante dado que não houve interação entre *software* externo ou máquinas como robôs e o CoppeliaSim.

Face à Figura 15 que identifica a interface gráfica do CoppeliaSim, é especificado abaixo as caraterísticas de cada um dos pontos:

1. **Barra de menu -** Permite aceder às funcionalidades do simulador que, ao contrário das mais utilizadas, não podem ser acedidas através da interação com os modelos, menus pop-up e barras de ferramentas;

2. **Barras de ferramentas -** Estes elementos estão presentes, para comodidade do utilizador, representam as funções mais utilizadas e essenciais à interação com o simulador, especificações (Figura 16 e 17);

3. **Texto informativo -** Contém informação alusiva ao objeto que se encontra selecionado num determinado momento e de parâmetros e estados da simulação;

4. **Browser dos modelos -** Numa parte superior mostra as pastas de modelos do CoppeliaSim; por outro lado, na parte inferior, encontram-se as *thumbnails* dos modelos que podem ser incluídos na cena através da ação de *drag-and-drop* suportada pelo simulador;

5. **Caixas de diálogo -** Recurso que aparece durante a interação com a janela principal e, através do qual, torna-se possível editar vários parâmetros relativos aos modelos ou à cena;

6. **Cena -** Demonstra a parte gráfica da simulação, ou seja, o resultado final do que foi criado e programado;

7. **Interface personalizada pelo utilizador -** É possível fazer uma configuração rápida de todos os componentes inseridos na "*cena*" através desta janela que surge para cada um dos objetos sempre que solicitado;

8. **Hierarquia da cena -** Aqui pode ser analisado todo o conteúdo de uma cena, ou seja, todos os objetos que a compõe. Uma vez que cada objeto é construído de forma hierárquica, esta constituição é representada pela árvore da sua hierarquia, como ilustrado na Figura 18. Com um duplo clique no nome de cada objeto, o utilizador consegue aceder à "*Interface Personalizada"* que a permite alterar. É também com esta funcionalidade do simulador, através do *drag-and-drop*, que se criam as relações parentais entre os objetos (objetos filho são arrastados para dentro da estrutura do objeto pai);

9. **Barra de estados:** Elemento responsável por exibir informação sobre operações, comandos e mensagens de erros. Além disto, o utilizador também pode usá-la para imprimir *strings* a partir de um *script*;

10. **Linha de comandos:** é usada para introduzir e executar código em Lua.

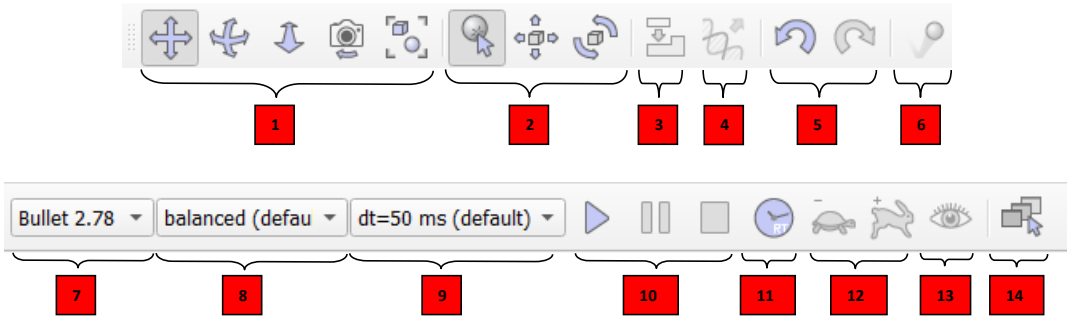


Figura 16 - Especificidades da barra de ferramentas.

Legenda:

1. Navegação de câmara
2. Manipulação de objetos
3. Montar e desmontar
4. Transferência de DND para componentes semelhantes
5. Undo e Redo
6. Visualização e verificação do conteúdo dinâmico
7. Precisão do motor de física
8. Step da simulação
9. Iniciar, Pausa e Terminar a simulação
10. Simulação em tempo real
11. Controlo de velocidade da simulação
12. Visualização on/off
13. Selecionador de páginas

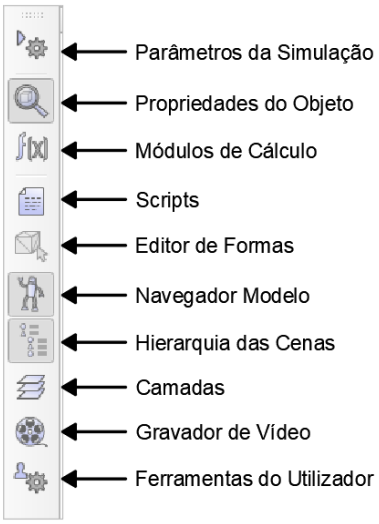


Figura 17 - Especificidades da barra de ferramentas.

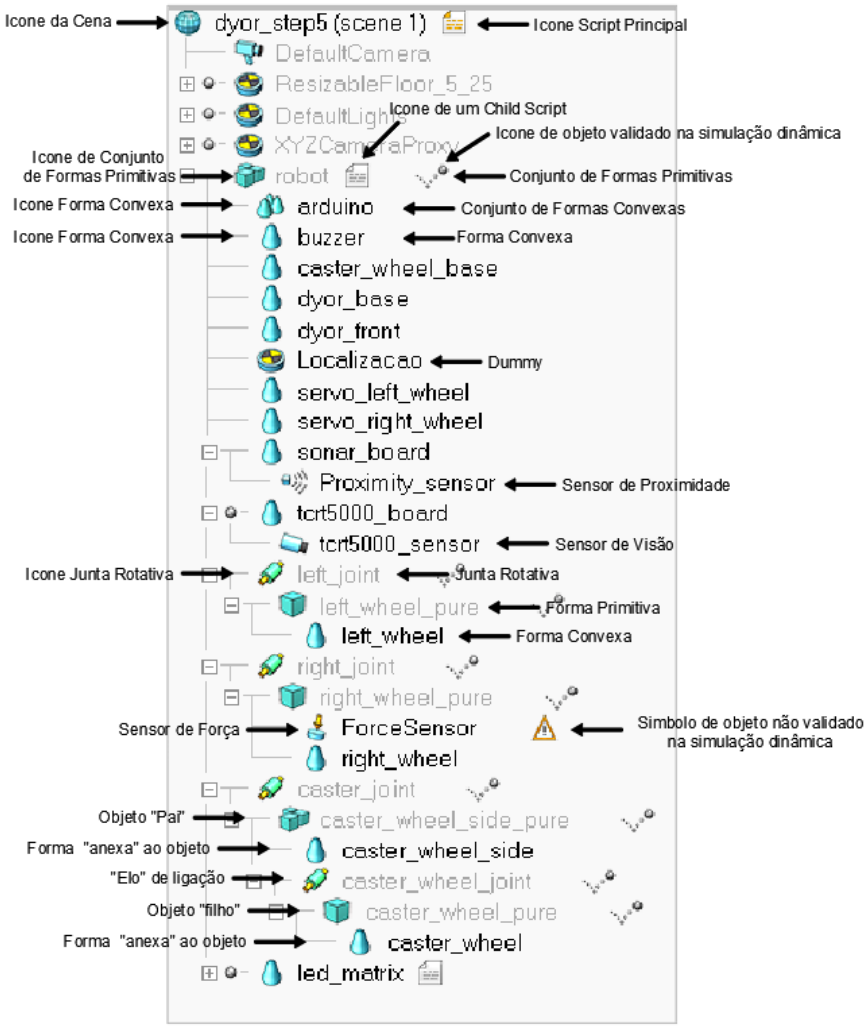


Figura 18 - Exemplo de uma hierarquia criada para o controlo de um robô, composta por formas, objetos, juntas, sensores de força e sensores de visão.

3.3.2. Objetos em cena

Os objetos em cena são os blocos de construção básicos do CoppeliaSim e a partir dos quais são desenvolvidas as outras capacidades. Tudo o que pode ser visualizado numa cena são objetos e estes podem ser estáticos ou dinâmicos, (com a ajuda de *scripts*) e podem ser usados para ler dados, interagir entre si, entre outras funcionalidades. Numa cena, os objetos podem estar em hierarquia (como exemplificado na Figura 18), o que pode influenciar o comportamento independente de outros objetos.

O principal objeto a utilizar no simulador é a "forma". Estas podem ser de dois tipos:

Forma convexa - É uma forma feita de malhas em formato triangular. Esta malha é dada por um conjunto de vértices, arestas e superfícies que definem a forma do objeto. Estes podem

assumir qualquer forma e são utilizados para representar vários tipos de objetos, incluindo robôs [135].

No entanto para efeitos de simulação estas têm de ser desconsiderados, pois iriam tornar a simulação extremamente pesada (dependendo também da forma em questão) e de difícil execução [147].

Forma primitiva - Estas são as formas utilizadas para efeitos de simulação. Têm corpo rígido e podem ser usadas de forma dinâmica. No entanto, as formas geométricas que estas podem ter são limitadas, pelo simulador, a formas do tipo: planares, em disco, paralelepípedos, esféricas e cilíndricas. Sempre que estas formas são utilizadas, importa especificar se serão do tipo:

- **Dynamic ou static -** As formas dinâmicas (*dynamic*) irão cair ou ser influenciadas por forças externas. Relativamente às estáticas (*static*), estas ficarão fixas numa determinada posição ou seguirão o movimento do objeto pai na hierarquia da cena;

- **Respondable ou non-respondable -** Uma forma responsiva (*respondable*) irá causar uma reação aquando da colisão com outras formas deste tipo; em contrapartida, tal não irá acontecer se a forma for não responsiva (*non-respondable*).

Para além das formas, existe todo um conjunto de objetos que podem ser utilizados na "*cena*", que são:

- **Juntas -** Podem ser de quatro tipos, de rotação, prismáticas, esféricas ou de parafuso. As juntas ligam dois ou mais objetos, com vários graus de liberdade e podem operar em diferentes modos, como força ou cinemática inversa;

- **Sensores de proximidade -** Medem a distância mínima do sensor, que geralmente está ligada a um objeto diferente, a qualquer outro objeto visível contido no volume de deteção configurável. Isto leva a uma deteção mais realista e contínua, ao contrário dos sensores do tipo raio;

- **Sensores de visão -** Permitem a extração de dados complexos sobre uma imagem, como tamanho, cores ou profundidade. Em conjunto com um *plugin* e um processamento de imagem integrado, é possível filtrar e analisar a imagem;

- **Sensores de força -** Medem a força e o binário e são representados por ligações rígidas. Estas ligações dependendo das condições, pelo que podem quebrar;

- **Gráficos -** Podem gravar e registar uma grande variedade de dados, como tempo, curvas X/Y ou 3D;

- **Luz e Câmara -** Utilizado apenas para a visualização de cenários, embora as luzes possam influenciar diretamente o funcionamento de um dos tipos de sensores;

- **Espelhos -** Funcionam como espelhos reais e têm a função de recorte;

- ***Dummies* -** São usados como um quadro de referência e normalmente anexados a outros objetos da cena. Normalmente, são utilizados nos scripts como referências (ex. localização). Não têm funções por si próprios;

- **Caminhos -** Permitem que outros objetos (principalmente formas) executem um movimento definido no espaço (orientação e posição). Pode ser usado, por exemplo, para correias transportadoras ou veículos;

- **Oc trees -** Representam uma partição espacial baseada no voxel de uma forma. Este pode ser usado para cálculos mais rápidos ou para uma representação simplificada de uma malha complexa;

- **Nuvens de pontos -** São pontos de armazenamento semelhantes aos OCtrees. Também usado para cálculos mais rápidos.

3.3.3. Módulos de cálculo

Um objeto de cena por si só não tem muito uso e, portanto, precisa de um algoritmo ou comando para interagir ou se mover. Os módulos de cálculo não são mais do que algoritmos que vêm pré-construídos no CoppeliaSim e que podem ser modificados e implementados através da janela de interface do utilizador, apresentada na Figura 19.

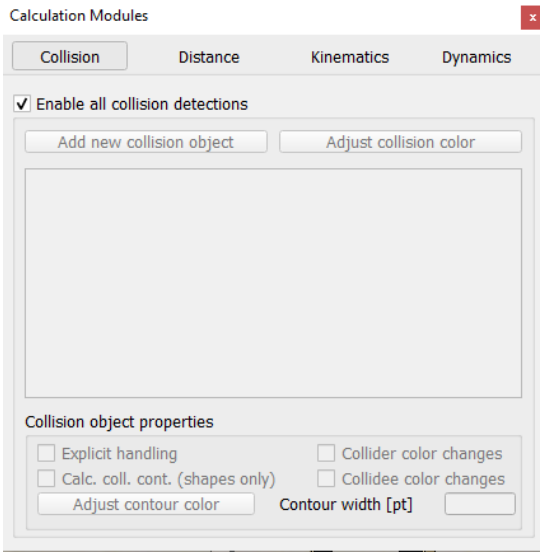


Figura 19 - Janela do utilizador para os módulos de cálculo.

A vantagem da utilização destes módulos existentes é que as funções básicas, que são muitas vezes usadas em qualquer simulação, não requerem bibliotecas externas ou *plug-ins,* tornando assim qualquer cena portátil e fácil de partilhar. Os módulos de cálculos são também acessíveis pelos scripts, e embora possam ser utilizados exclusivamente através da janela do utilizador, para uma utilização mais completa e dinâmica, é necessário os comandos disponíveis nos scripts.

Os principais módulos de cálculo disponíveis são:

Módulo de deteção de colisão - Verifica se há interferências entre formas ou coleção de formas, ou seja, existe uma colisão quando há uma sobreposição de formas, *octrees* ou nuvens de pontos. Este módulo de colisão é independente das respostas de colisão que os motores de física darão;

Módulo de cálculo de distância - Permite medir rapidamente a distância mínima entre duas malhas ou coleção de formas;

Módulo de cinemática - Funciona tanto para cinemática direta como inversa de qualquer mecanismo, como redundante, ramificado ou fechado. Suporta resoluções condicionais, amortecidas/não amortecidas e ponderadas, bem como prevenção de colisões;

Motor de física ou Módulo de dinâmica - Permite um tratamento realista das interações dos corpos rígidos, como colisões ou agarramentos. Permitindo a utilização de 4 motores de física diferentes. O motor *Bullet Physics*, o *Open Dynamics Engine* (ODE), o *Vortex Dynamics* e o *Newton Dynamics*;

Módulo de planeamento de trajetórias - Permite o planeamento de trajetórias de tarefas holonómicas e não holonómicas através de um algoritmo baseado na aproximação de derivadas chamado *Rapidly-exploring Random Tree* (RRT). Embora este esteja ainda disponível, os criadores do CoppeliaSim já não o recomendam, pois foi substituído pelo *plugin* OMPL;

OMPL - É uma biblioteca que inclui uma variedade de algoritmos de planeamento de movimento baseados em amostragem. A própria biblioteca consiste apenas em algoritmos sem verificação de colisão ou interface de utilizador, para que possa ser mais facilmente implementada com outros programas. Embora este não seja um módulo de cálculo, pode substituir o módulo de planeamento de trajetórias.

3.3.4. Mecanismos de controlo

Para construir qualquer cena complexa, que envolva movimento, cálculos ou interação, é necessário mais do que objetos de cena e os módulos de cálculo básicos. Para controlar todos os aspetos de uma simulação, podem ser escolhidas diferentes abordagens, tendo cada uma as suas vantagens. Ao executar uma simulação, o código pode ser executado de três diferentes maneiras:

- Diferentes máquinas (computador ou robô), ligadas ao computador onde a simulação está a ser realizada;
- Mesma máquina, mas num processo separado da rotina da simulação;
- Código e simulação executados no mesmo computador.

Existem quatro formas de implementar o código e programar um modelo no CoppeliaSim (ver Figura 20):

- **Scripts embutidos -** São o principal mecanismo de controlo e um dos mais poderosos, pois permitem que ao utilizador programar diretamente no modelo, sem nenhum *software* externo. A linguagem de codificação usada aqui é o Lua;
- ***Plug-ins* -** São usados para personalizar ainda mais uma cena, ampliar funcionalidades ou registar novos comandos de script. Algumas das funcionalidades, como a interface OMPL ou ROS, são implementadas por meio de *plug-ins*, usando uma interface C/C++;
- **Add-ons -** Podem ser funções independentes ou ser usadas junto do código executado regularmente. Esta forma usa também uma interface Lua e pode personalizar o simulador;
- **API remoto -** Facilita a interação entre *softwares* externos ou máquinas, como robôs reais e o CoppeliaSim. Isto pode ser feito usando cinco linguagens de codificação diferentes.

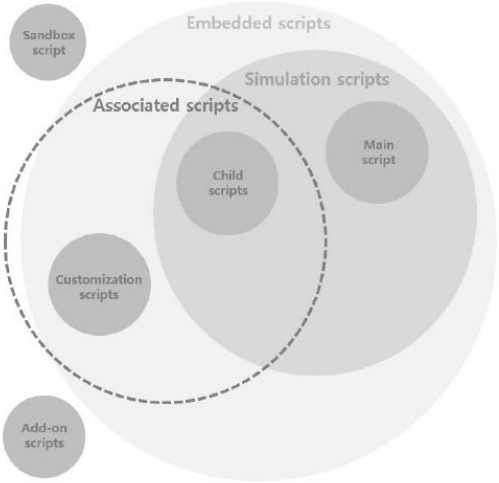


Figura 20 – Tipos de scripts suportados pelo simulador.

Os scripts embutidos são o principal mecanismo de controlo utilizado no CoppeliaSim. Este método tem uma integração fácil, escalabilidade inerente e sem conflitos com as diferentes versões, menos esforço para criar, modificar e manter e sem problemas de sincronização. Os scripts embutidos permitem ainda a utilização das mais de quinhentas funções que a biblioteca de API dispõe e que cobrem quase todas as possibilidades de programação. Os *scripts* embutidos

é onde estão ligados todos os outros mecanismos de controlo e são fundamentais em qualquer cena [137].

O CoppeliaSim dispõe de cinco tipos de scripts, podendo a relação entre estes ser analisada na imagem 19 para utilizar:

- **Main script -** É o script principal e está sempre presente em qualquer cena, uma vez que contém o código que faz com que seja possível realizar a simulação, razão pela qual não deve ser modicado. Além disso, é responsável por chamar todos os outros scripts secundários (está também identificado na Figura 18);

- **Child scripts -** Estes scripts estão associados a um objeto ou modelo presente na "*cena"* e são usados para implementar os respetivos algoritmos de controlo. Porém, também podem ser aplicados noutros contextos. Existem dois tipos de *Child scripts*, os *Non-threaded* e os *Threaded*. Os primeiros são constituídos por funções que são chamadas a cada passo da simulação e realizam uma tarefa (*callbacks*), devolvendo depois o controlo ao script que as chama. É essencial que isto aconteça, caso contrário, a simulação irá parar e não prosseguirá. O script *Non-threaded* é constituído, fundamentalmente por quatro funções:

  - *sysCall init*: É a função de inicialização e também a única que não é opcional. Apenas é executada uma vez durante a simulação e contém o código de inicialização;

  - *sysCall actuation*: É executada em cada passo da simulação durante a fase de atuação. Normalmente, contém o código relacionado com a atuação dos componentes do modelo;

  - *sysCall sensing*: Função que também é executada em todos os passos da simulação durante a fase de sensorizacão. Contém, por norma, o código relativo à aquisição de dados através dos sensores do modelo;

  - *sysCall cleanup*: É a função de restauro e é executada uma vez no final da simulação ou quando o script é eliminado.

  No que toca aos scripts do tipo *Threaded*, estes são executados numa nova *thread*, paralela à simulação. Por esta razão, não há desde logo sincronia com o passo da simulação, embora isso possa ser conseguido recorrendo a um determinado conjunto de funções. Em contrapartida, a simulação não será interrompida durante a sua execução, sendo que a execução do script é que é interrompida no final de um passo e retomada no seguinte.

- **Scripts de customização -** Como acontece no exemplo apresentado anteriormente, estes scripts também estão associados a um objeto ou modelo, no entanto, são dedicados à customização de parâmetros do mesmo (p. ex.: altura, comprimento, etc.);

- **Scripts Add-on e Sandbox -** Podem ser empregues pelo utilizador para implementar certas funcionalidades mais genéricas (que não estão ligadas a um modelo ou cena em específico), permitindo, até um certo nível, personalizar o simulador.

3.4. Nota Conclusiva

Este capítulo tem como propósito dar a conhecer todos os componentes e funcionalidades do rover robótico considerado neste estudo. Pois, o seu conhecimento prévio é indispensável para perceber quais os principais aspetos a ter em conta no processo de recriação do rover robótico no simulador agora escolhido.

Além disso, é também útil o conhecimento dos *softwares* de simulação computacional para sistemas robóticos, as potencialidades que estes têm e o contributo que podem dar no desenvolvimento de sistemas robóticos. Assim, foram apresentados alguns dos principais simuladores robóticos existentes e com base na análise realizada concluiu-se que o mais adequado para o trabalho que se pretende realizar é o CopelliaSim, sendo apresentadas as várias funcionalidades que este dispõe.

4. Modelação do Rover Robótico Agrícola

Este capítulo descreve detalhadamente a implementação do modelo 3D, desenvolvido por Veiros [12], no *software* CoppeliaSim, bem como os aspetos tidos em conta no sentido de tornar a simulação experimental o mais realista possível.

É também apresentado o algoritmo proposto para deteção e recolha de frutos caídos. Seguidamente, o modelo 3D foi implementado e simulado num ambiente concebido de forma a replicar o meio agrícola com diversos objetos espalhados, de forma aleatória, pelo chão e com objetos físicos nas laterais de forma similar a um pomar.

4.1. Criação do Rover Robótico no CoppeliaSim

Apresentado o *software* utilizado nesta experiência, bem como as várias funcionalidades que esta ferramenta dispõe, é agora descrito todo o processo utilizado na transição do modelo 3D para o CoppeliaSim.

Como já referido, o rover robótico foi idealizado no SolidWorks e só depois foi realizada a sua construção física. Assim, para implementar e simular experimentalmente o rover robótico no simulador CoppeliaSim, a representação final que deu origem ao rover robótico, apresentada na Figura 8, foi importada para o respetivo simulador.

4.1.1. Criação das formas

Para realizar a importação do desenho 3D foi necessário converter o mesmo num formato do tipo: ".obj", ".dxf", ".ply" ou ".stl". No caso, o formato utilizado foi o ".stl", que era o formato de maior facilidade e precisão para carregar no simulador. De referir que o desenho importado é carregado como formas do tipo convexas, que como referido no capítulo 3.3.2, são formas feitas de malhas triangulares e que podem assumir qualquer forma geométrica.

Os componentes que foram importados para a simulação encontram-se descritos na Tabela 6. Depois de feita esta importação, foram criadas as formas primitivas com as dimensões e propriedades físicas (ex: massa específica, peso, atrito, amortecimento, etc.) idênticas às dos componentes originais, com o intuito de tornar os comportamentos do rover robótico o mais semelhante possível com a realidade.

De referir ainda que na importação dos vários componentes do rover robótico para o simulador foram ignorados alguns objetos que eram desprezáveis para efeitos de simulação e que a tornavam apenas computacionalmente mais pesada, como parafusos e acessórios de fixação.

Tabela 7 – Elementos constituintes do rover robótico, importados para o simulador.

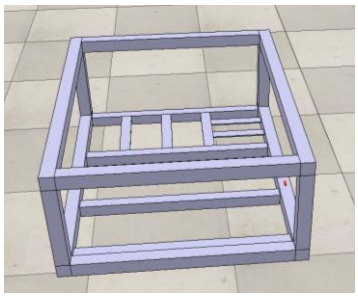
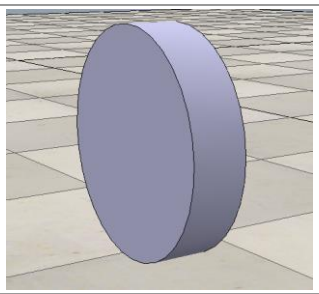
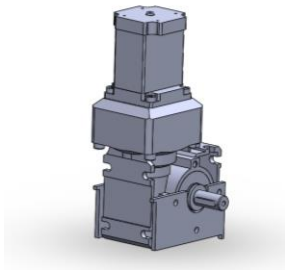
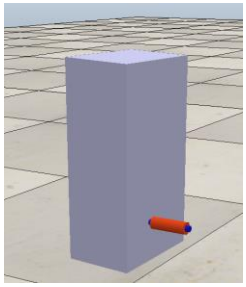
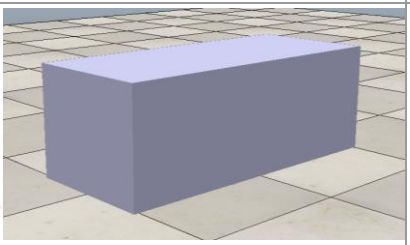
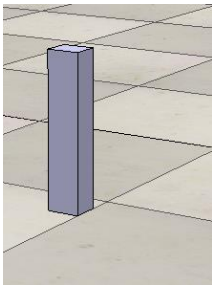
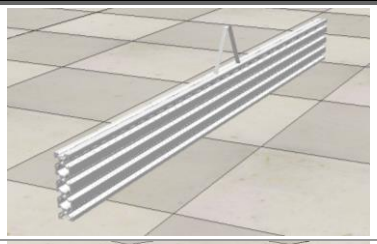
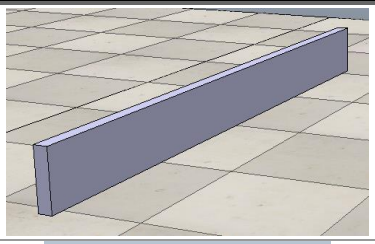
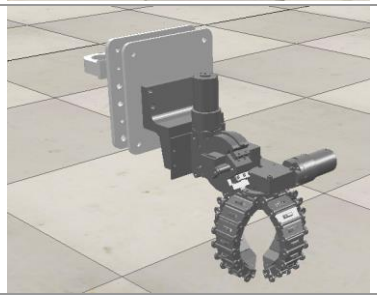
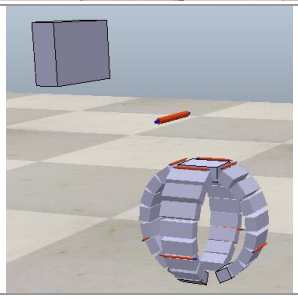
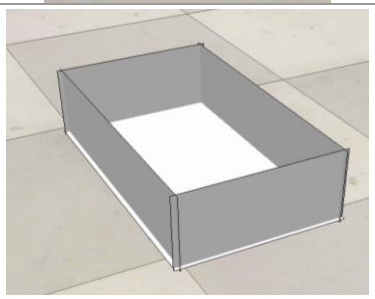
Objeto	Modelo 3D	Forma Criada	Peso
Chassis			15,29 kg
Roda			0,57 kg
Motor de Passo			0,63 kg
Caixa para Baterias e restantes acessórios			11,59 kg
Conjunto para eixo Z e Y			0,41 kg

Tabela 6 – Elementos constituintes do rover robótico, importados para o simulador (cont.).

Objeto	Modelo 3D	Forma Criada	Peso
Barra metálica eixo X			0,65 kg
Pinça para apanhar frutas			0,52 kg
Embalagem de armazenamento de Frutas			3,13 kg

No caso da caixa que alberga as baterias e outros componentes de controlo, foi tido em conta apenas o peso total da caixa com as baterias e todos os seus acessórios inerentes, para verificar assim eventuais problemas de tração que possam ou não existir. Concluída esta etapa, fica-se com o desenho original (formas feitas de malhas triangulares) e com as formas primitivas que serão as usadas para efeitos de simulação, podendo ambas ser observadas na Figura 21.

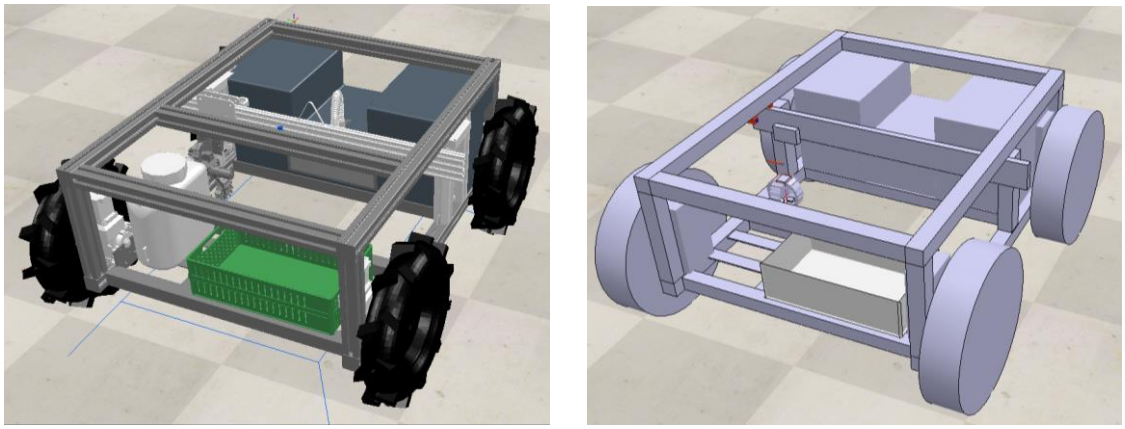


Figura 21 - Layout final do rover robótico carregado para o simulador.

O passo seguinte foi criar todas as juntas que o rover robótico possui. Na criação destas juntas foi tido em conta todas as características mecânicas e elétricas das originais, como velocidade, binário e controlo de posição no caso das juntas prismáticas e da garra mecânica. As juntas criadas foram:

- 4 Juntas de rotação para as rodas de locomoção do veículo (motores de passo DC);
- Junta Prismática do eixo n.º3 (Eixo Z);
- Junta Prismática do eixo n.º2 (Eixo Y);
- Junta Prismática do eixo n.º1 (Eixo X);
- Junta Rotacional da Garra mecânica;
- Junta Rotacional das várias pinças mecânicas.

Como referido no capítulo 3.1.1, o suporte do eixo X poderá ter instalado um bico pulverizador ou uma garra mecânica, consoante a atividade que se pretenda realizar. Como neste caso o primeiro e principal trabalho que se pretende realizar é o da recolha de frutas, foi já instalada a garra mecânica no suporte do eixo X.

Esta garra foi um trabalho também desenvolvido pelo Grupo Operacional PrunusBot, Operação nº PDR2020-101-03158 (líder), Consórcio nº 340, Iniciativa nº 140, promovido pelo PDR2020 e cofinanciado pelo FEADER e União Europeia no âmbito do Programa Portugal 2020 e será utilizada nos trabalhos práticos que se pretendem realizar com este rover robótico na recolha de frutas.

De uma forma geral esta garra mecânica é constituída por um conjunto de 4 pinças que abrem ou fecham para recolher os objetos e a montante destas pinças existe uma junta principal que permite rodar as quatro pinças da garra mecânica, para que esta fique numa posição mais elevada e possa colocar os objetos capturados dentro da embalagem. Para além desta junta principal, cada pinça da garra mecânica irá ter um total de 8 juntas rotativas para facilitar a recolha dos objetos e para não os danificar, dado a sensibilidade dos objetos a recolher.

No entanto, para efeitos de simulação verificou-se que a criação de todas estas juntas não acrescentava valor à simulação. Pelo contrário, tornava o algoritmo mais pesado e por sua vez a simulação mais exigente computacionalmente e com maior duração. Assim, optou-se por utilizar apenas duas das juntas de cada pinça

Esta garra mecânica está representada na Figura 22, na qual se encontra apresentado o desenho importado do ficheiro original, em formas convexas, e também as formas primitivas, que foram criadas desse desenho original e que servirão de base à simulação. Podendo também observar-se as respetivas juntas que serão utilizadas.

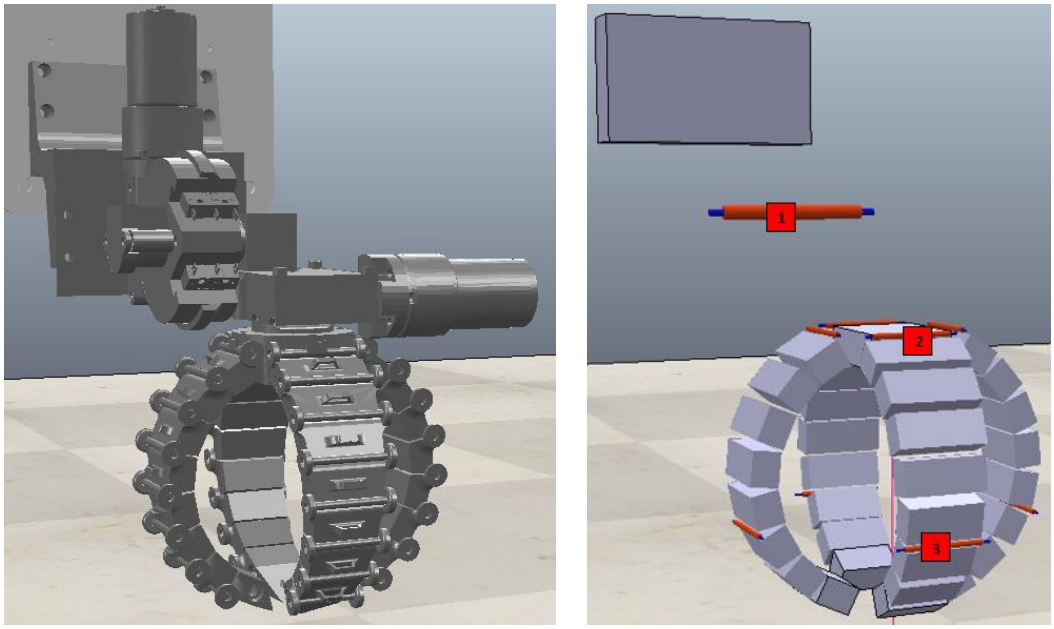


Figura 22 - Recriação da garra mecânica no simulador.

Legenda:

1. Junta Principal da Garra mecânica;
2. Junta nº1 da Pinça n.ºx;
3. Junta nº2 da Pinça n.ºx.

Criadas todas as formas primitivas e juntas, quer da garra mecânica quer do rover robótico, o passo seguinte passou por colocar todos estes componentes nas mesmas posições em que se encontram os componentes originais, de forma precisa, ficando assim todos estes elementos numa sobreposição perfeita com o desenho original e de forma completamente alinhados para o movimento das juntas.

No entanto, para efeitos de simulação, o rover robótico não está ainda concluído, pois e de acordo com o mencionado no capítulo 3.3.1 importa agora criar uma ligação entre estes objetos numa estrutura de forma hierarquia. A hierarquia que foi criada é a observada na Figura 23.

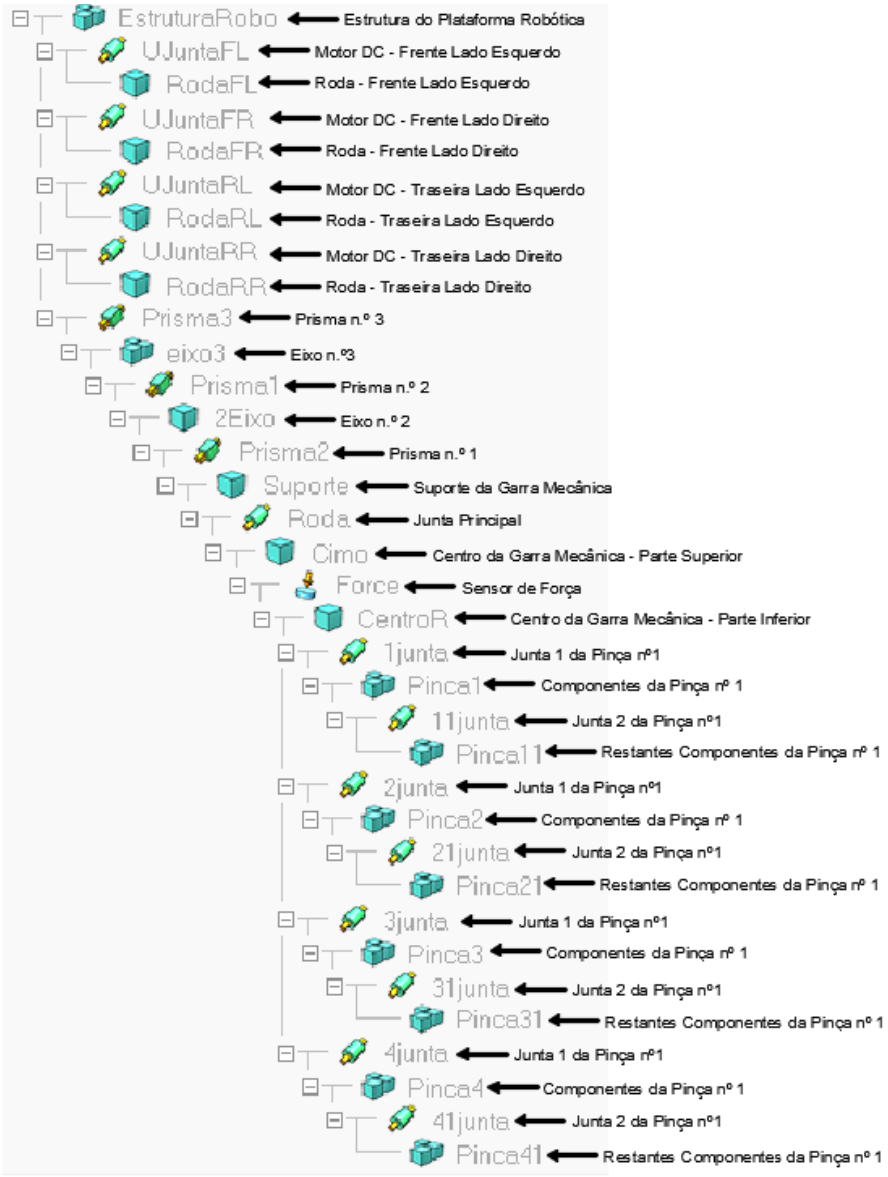


Figura 23 - Hierarquia criada para o rover robótico.

Definida a hierarquia dos vários componentes que fazem parte deste rover robótico, já é possível iniciar a simulação. Porém, as formas importadas do desenho inicial não serão utilizadas. Assim e apenas com o intuito de melhorar a parte gráfica da simulação, estes desenhos 3D que foram importados serão "anexados" às formas primitivas dos vários componentes. Com esta ação o que irá acontecer durante as simulações é que estes desenhos 3D importados irão seguir o movimento das formas primitivas durante a simulação. Esta estrutura final pode ser observada na Figura 24.

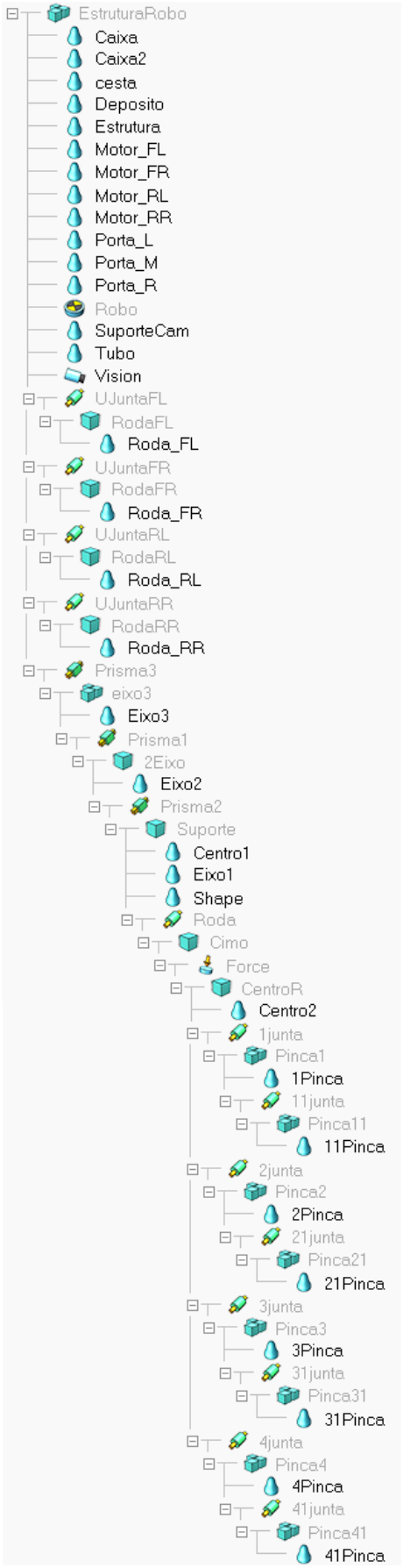


Figura 24 - Hierarquia Final.

De referir que a esta nova estrutura foram ainda adicionados componentes que não faziam parte do rover robótico original, mas que serão necessários para a criação do algoritmo proposto, como é o caso de um "*dummy*" para saber a localização do rover robótico. Estes componentes foram anexados à forma primitiva "EstruturaRobo" do rover robótico, pois é ai que estes seriam instalados caso fossem colocados na realidade.

4.1.2. Inserção de câmara de vídeo

Criado o rover robótico, falta apenas adicionar uma câmara de vídeo para a captação de imagens e poder assim localizar os objetos a recolher. A câmara que está a ser utilizada neste rover robótico é o modelo *"Raspberry Pi Camera v2"*. Esta câmara permite a captação de imagens em três resoluções diferentes: 1080p30, 720p60 e 640x480p90. Podendo as restantes características ser consultadas na ficha técnica dos anexos do presente documento.

Como referido no capítulo 3.3.2, o CoppeliaSim permite a inserção de vários tipos de sensores, incluindo sensores de visão. Depois de inserida a câmara e posicionada de forma idêntica à instalada no rover robótico, foi necessário apenas a configuração dos vários parâmetros, como descrito na Figura 25.

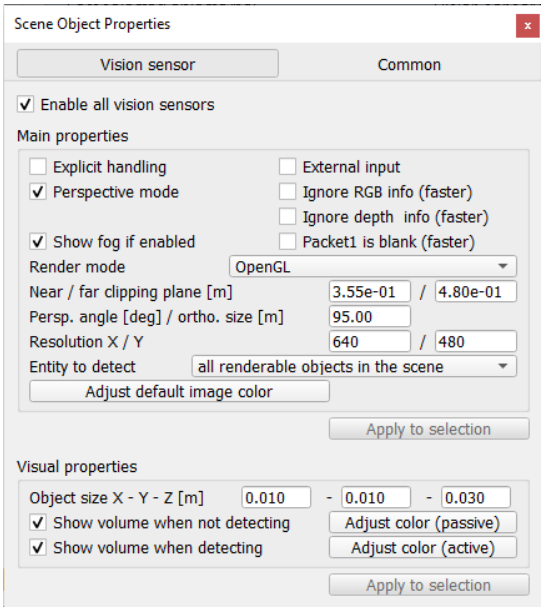


Figura 25 - Definição dos vários parâmetros do sensor de visão.

Uma vez que a resolução da câmara instalada é relativamente elevada, observou-se que as simulações tornavam-se ligeiramente pesadas, pelo que se optou por utilizar a resolução mais baixa, que para efeitos de deteção de objetos e precisão na recolha dos mesmos mostrou ser indiferente, pelo menos para o algoritmo implementado.

A câmara foi instalada num local idêntico à do protótipo original, isto para garantir que o ângulo de visão será igual á da realidade. A câmara está instalada na parte superior do rover robótico, tendo sido adicionado um perfil de alumínio a meio da estrutura do rover robótico para permitir uma visão da parte central e inferior do rover robótico, como observado na Figura 26, em que a câmara está representada na cor azul.

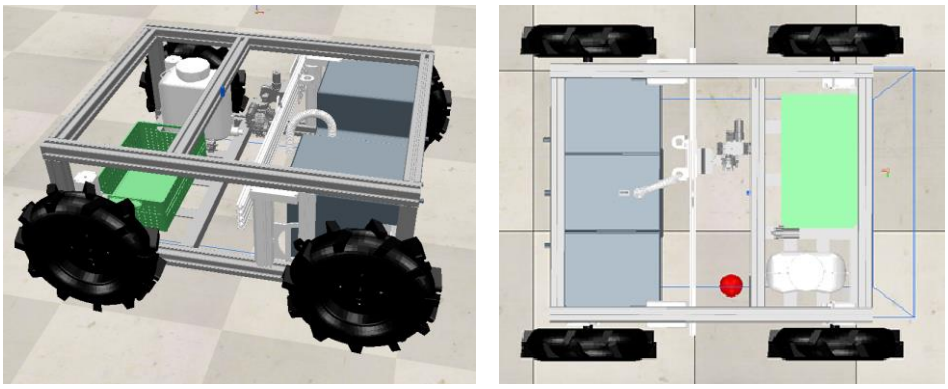


Figura 26 – Instalação da câmara no Rover robótico.

Estando a câmara instalada e configurada corretamente, o passo seguinte foi definir as zonas da imagem que devem ser analisadas quando esta estiver a operar. A definição dessas zonas pode ser observada na Figura 27.

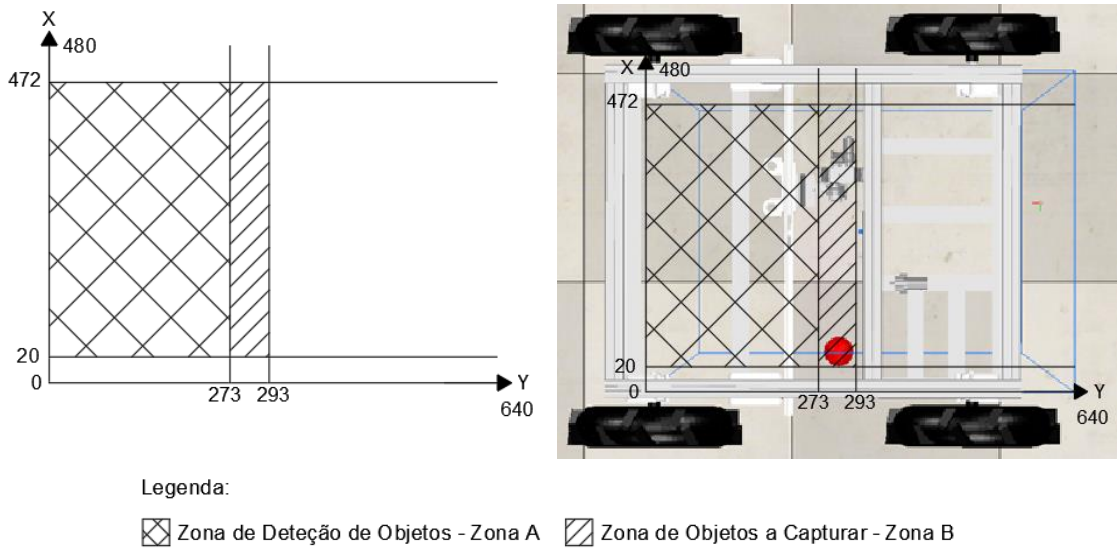


Figura 27 - Definição de zonas no sensor de visão.

Como já referido, a resolução escolhida para esta simulação foi de 640x480 p90, em que será uma imagem vídeo com uma largura de 640 *pixels* por uma altura de 480, possuindo um total de 307.200 *pixels* com uma taxa de atualização de 90 imagens por segundo.

4.2. Algoritmo Proposto

Como já referido, pretende-se que este rover robótico realize duas tarefas no entanto para o processo de idealização de um algoritmo foi tido como referência a tarefa de recolha de objetos por esta ser a mais complexa. Podendo posteriormente ser adaptado, facilmente, à tarefa de pulverização. Assim e para a tarefa de recolha de objetos foi idealizado um fluxograma, representado na Figura 28, que cumpre as várias etapas e condições lógicas necessárias ao correto funcionamento do algoritmo.

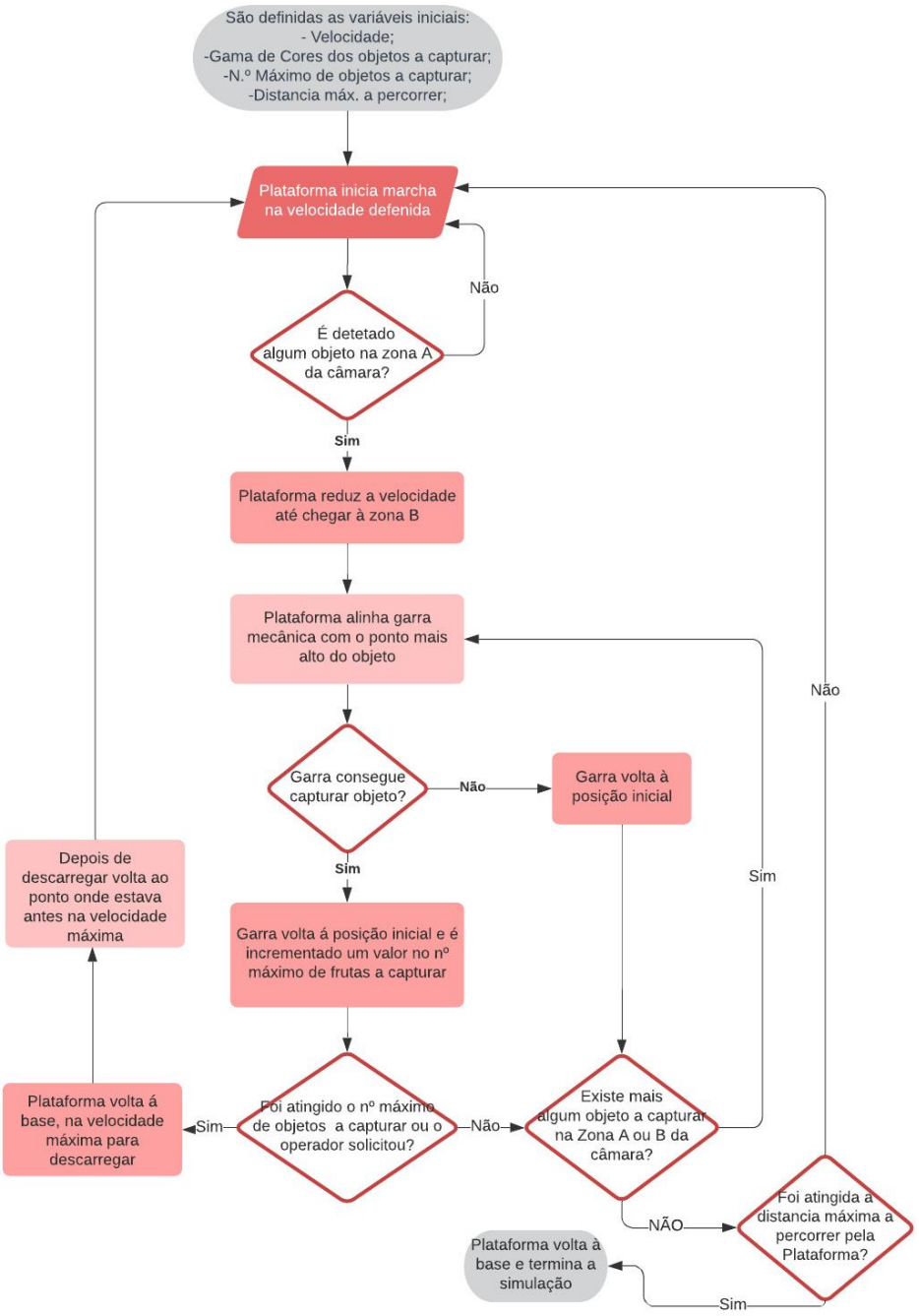


Figura 28 - Fluxograma para controlo do rover robótico.

4.2.1. Criação do algoritmo

Estando idealizado o algoritmo que se pretende construir, foi iniciada a conceção do mesmo no simulador escolhido. Como referido no capítulo 3.3.4, este simulador permite inserir vários tipos de *scripts*, no entanto e uma vez que o algoritmo que se pretende criar é para o controlo de um rover robótico, o script a utilizar será do tipo *Child scripts*, pois este é o tipo de *script* que permite a associação a um objeto em específico.

Dentro dos *Child scripts* foi necessário escolher entre os do tipo *Non-threaded ou Threaded*. No entanto, e das várias simulações realizadas, *observou*-se que o script que me melhor se adaptava era o *Child script do tipo Non-threaded,* quer pelo facto de ser mais fácil de implementar, quer a nível de fluidez da simulação. Além disso, segundo os próprios desenvolvedores do CoppeliaSim, os scripts *Non-threaded* devem ser sempre a primeira escolha visto que existem certos aspetos inerentes aos *Threaded* que poderão afetar o desempenho durante a simulação [147]. A inserção e ligação deste script ao rover robótico pode ser observada na Figura 29.

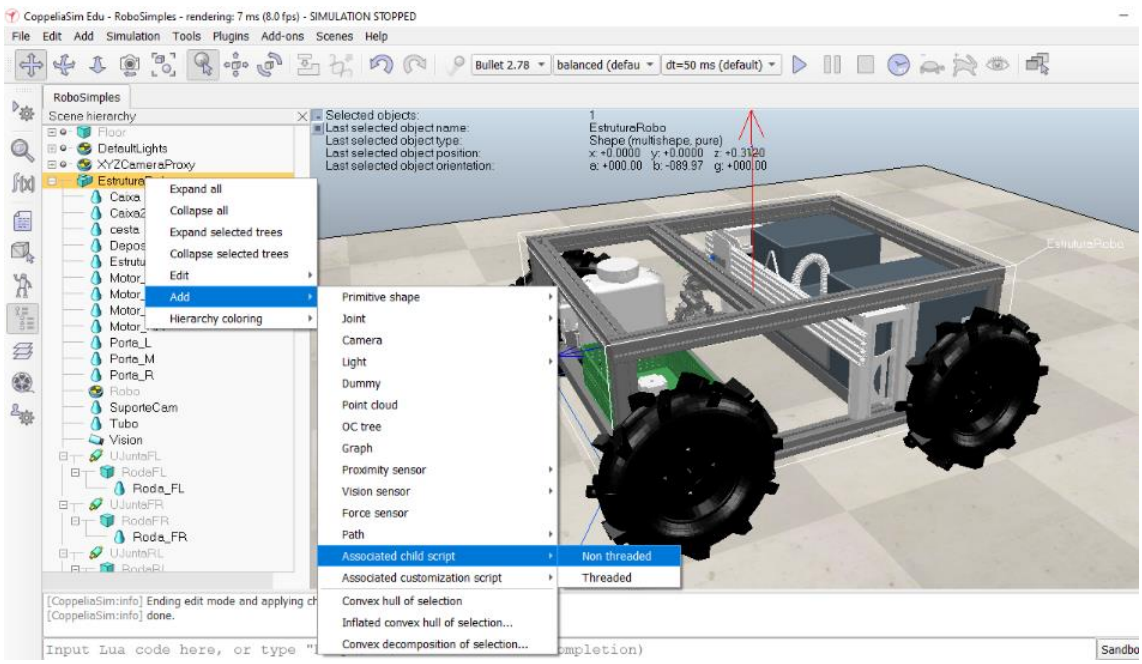


Figura 29 - Inserção de script.

Associado o *script* ao objeto que se pretende controlar, neste caso ao rover robótico, foi iniciada a construção do algoritmo que controlará o rover robótico. Por forma a não tornar este subcapítulo demasiado extenso, serão apresentados apenas os passos mais importantes que foram tomados na criação deste algoritmo podendo, no entanto, ser consultado nos anexos do presente documento o algoritmo de forma integral.

Na fase inicial foram declarados os vários objetos que importa controlar nesta experiência. A declaração destes objetos foi feita com recurso à função "*sim.getObjectHandle*", obtida na biblioteca de funções do API em que é inserido o nome do objeto a controlar e atribuído uma

variável para o controlo desse mesmo objeto. A declaração destes objetos pode ser observada na Figura 30.

```
WheelFL=sim.getObjectHandle('UJuntaFL')    --Junta da roda da frente lado esquerdo
WheelFR=sim.getObjectHandle('UJuntaFR')    --Junta da roda da frente lado direito
WheelRL=sim.getObjectHandle('UJuntaRL')    --Junta da roda da traseira lado esquerdo
WheelRR=sim.getObjectHandle('UJuntaRR')    --Junta da roda da traseira lado direito
tv=sim.getObjectHandle('Vision')           --Sensor de visao (Camara Video)

eixo1=sim.getObjectHandle('Prisma1')    --Junta prismatica do eixo n.?1
eixo2=sim.getObjectHandle('Prisma2')    --Junta prismatica do eixo n.?2
eixo3=sim.getObjectHandle('Prisma3')    --Junta prismatica do eixo n.?3


junta1=sim.getObjectHandle('1junta')     --Junta n. 1 da pinca 1
junta2=sim.getObjectHandle('2junta')     --Junta n. 1 da pinca 2
junta3=sim.getObjectHandle('3junta')     --Junta n. 1 da pinca 3
junta4=sim.getObjectHandle('4junta')     --Junta n. 1 da pinca 4

junta11=sim.getObjectHandle('11junta')   --Junta n. 2 da pinca 1
junta21=sim.getObjectHandle('21junta')   --Junta n. 2 da pinca 2
junta31=sim.getObjectHandle('31junta')   --Junta n. 2 da pinca 3
junta41=sim.getObjectHandle('41junta')   --Junta n. 2 da pinca 4

rota=sim.getObjectHandle('Roda')     --Junta principal da garra mecanica

robo=sim.getObjectHandle('Robo')     --Dummy sobre localizacao do Robo
stop=sim.getObjectHandle('Stop')     --Dummy sobre distannia maxima a percorrer
base=sim.getObjectHandle('Base')     --Dummy sobre localizacao da Base

estrutura=sim.getObjectHandle('EstruturaRobo')  --Estrutura total do robo
sensforc=sim.getObjectHandle('Force')    --Sensor de força implementado na garra
```

Figura 30 - Código criado para declaração dos vários objetos.

Fica ainda em falta a declaração dos objetos que deverão ser recolhidos. Dado o facto de serem muitos objetos, foi utilizado um ciclo *for* para facilitar a declaração de todos estes objetos, tendo sido aproveitado a criação deste ciclo para definir as características físicas dos mesmos, nomeadamente o seu peso. Tal pode observado na Figura 31.

```
esfera = {}
for i=0,14,1 do
esfera[i]=sim.getObjectHandle('Sphere'..i)
sim.setShapeMass(esfera[i],0.1)
end
```

Figura 31 - Código criado para declaração dos objetos a recolher.

Declarados todos os objetos a utilizar, foram ainda definidos os valores de algumas das variáveis a utilizar, servindo em muitos casos apenas para definir um valor inicial da variável, que poderá ser alterado no decorrer da simulação. Estes valores encontram-se na Figura 32.

Para além disso foi ainda adicionada a função "*simUI.create*", que permite a criação de uma janela de informação ao utilizador (UI), que no decorrer da simulação dará informações, instantâneas, de alguns parâmetros que estão a ser simulados e poderá até ser utilizada para adicionar novos *inputs* à simulação, servindo assim como uma ferramenta de apoio ao utilizador.

```
cp={0}
dados ={}
dad ={}
id=0
fruta=0
ppl=0
loc=0
turbo=0
Pbase={0}
Probo={0}
localizacao=0
zona=1
anda=0
counter=0
erro=0
sentido=1
```

Figura 32 - Código criado para Definição de variáveis.

Importa agora salientar que todo o código apresentado até ao momento foi inserido na função *sysCall_init(),* pois de acordo com o referido em [147], todos os objetos a utilizar na simulação devem ser declarados e definidos nesta função. Isto porque os *Child scripts* do tipo *Non-threaded* são constituídos por quatro funções, tal tinha sido mencionado no capítulo 3.3.4.

Feita esta parte inicial, todo o código que se segue foi inserido na função *sysCall actuation(),* que contém o código relacionado com a atuação dos vários componentes do modelo. Na fase inicial desta função, começou-se por criar um conjunto de variáveis que terá por intuito "medir" a posição ou velocidade de determinados objetos, que importam saber a sua exata posição no decorrer da simulação. Para isso foram utilizadas algumas funções específicas da biblioteca API, observadas na Figura 33. A execução dessas funções passou por definir qual a variável que tem o controlo de determinado objeto (operação feita acima) e qual a nova variável que irá medir a posição ou velocidade desse mesmo objeto.

```
--Parametros a ler na janela de Informacao ao Utilizador (UI)
simUI.setLabelText(ui,1,string.format("Num. Frutas ate ir a base: %.2f",-1*(fruta-5)))
simUI.setLabelText(ui,2,string.format("Distancia da base [m]: %.2f",(Probo[1]-Pbase[1])))
simUI.setLabelText(ui,3,string.format("Velocidade[m/s]: %.2f",turbo))
velo1,velo2=sim.getVelocity(estrutura)

-- variavel que mede a velocidade do robo para apresentar na janela UI
if velo1[1] <0 then
    turbo = -1*velo1 [1]
else turbo = velo1 [1]
end


Posicao1= sim.getJointPosition (eixo1)  -- variavel que mede a posicao do prisma 1
Posicao2= sim.getJointPosition (eixo2)  -- variavel que mede a posicao do prisma 2
Posicao3= sim.getJointPosition (eixo3)  -- variavel que mede a posicao do prisma 3
m1=sim.getJointPosition(junta1)

-- variavel que mede a posicao da junta principal da garra mecanica
Prota=sim.getJointPosition(rota)

-- variavel que mede a velocidade da roda frente lado esquerdo
velocidade=sim.getJointVelocity(WheelFL)

Probo=sim.getObjectPosition(robo,-1)  -- variavel que mede a posicao do robo
Pstop=sim.getObjectPosition(stop,-1)  -- Distancia maxima que o robo deve percorrer
Pbase=sim.getObjectPosition(base,-1)  -- variavel que mede a posicao da base

-- variavel que mede o sensor de forca da garra mecanica
resu,forca,torrqe=sim.readForceSensor(sensforc)
```

Figura 33 - Código criado para a definição das varáveis que irão medir posições.

Algumas destas novas variáveis criadas serão utilizadas na janela UI. Esta janela está representada na Figura 34, onde é possível ver as informações que irá fornecer. Além disso, vai ainda permitir que o operador solicite, por alguma eventualidade, o regresso do rover robótico à base.

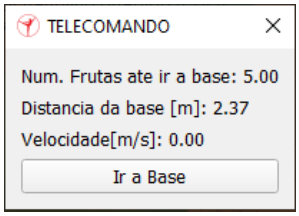


Figura 34 - "Telecomando" criado para a simulação.

O próximo passo foi a criação de um código que verificasse se nas zonas A e B da câmara vídeo está a passar algum objeto que deva ser detetado e as ordens que devem ser cumpridas caso isso aconteça. Sendo que uma das ordens que deverá cumprir é a determinação da velocidade de operação do rover robótico, sendo que para este algoritmo são propostas três velocidades:

- Velocidade “mínima”;
- Velocidade “média”;
- Velocidade “máxima”.

Assim o código inicial deste algoritmo irá garantir o cumprimento dos requisitos lógicos definidos na fase inicial do fluxograma da Figura 28. Este código foi realizado com recurso a ciclos do tipo *for e* operadores do tipo *if* e *else*, podendo ser observado na Figura 35.

```
if zona==1 then
 pr=0
 h1=0
 h2=0
 h3=0

     for u=20, 433, 7 do
     conf3=sim.getVisionSensorDepthBuffer(tv,x,u,30,30)
     conf4=sim.getVisionSensorImage(tv,x,u,30,30,0)
         for j=1, 900, 1 do
             pr=conf3[j]+pr
         end
         deep=pr/900

         for i=0, 2697, 3 do
             h1=conf4[i+1]+h1
             h2=conf4[i+2]+h2
             h3=conf4[i+3]+h3
         end
             r=h1/900
             g=h2/900
             b=h3/900
```

Figura 35 - Código criado para a verificação dos requisitos iniciais.

```
        if deep<= 0.77 and r>=0.65 and b<=0.2 and x >=260 then
            marcha=1
            zona=0
            xx=260
            break
        elseif deep<= 0.77 and r>=0.65 and b<=0.2 then
            marcha=1
            zona=0
            xx=250
            break
        end
        pr=0
        h1=0
        h2=0
        h3=0
    end

    if x>0 then
        x=x-20
    elseif x==0 then
        marcha=0
        zona=0
    end

end
```

Figura 35 - Código criado para a verificação dos requisitos iniciais (cont).

De referir que este código irá correr logo no início da simulação e irá também ser corrido sempre que seja recolhido algum objeto. Referir ainda que esta verificação é feita analisando partes parciais da imagem que está a ser capturada, tendo sido realizados vários testes para perceber qual o tamanho ideal da imagem a analisar, verificando-se que o tamanho de 30 x 30 *pixels* era o ideal. Isto porque se fosse uma imagem mais pequena corria-se o risco de detetar objetos que não interessavam (ex. pedras) e se a imagem fosse demasiado grande tornaria a simulação demasiado pesada e seria analisada uma área superior à dos objetos a recolher o que também não tinha interesse.

Assim sempre que seja capturada uma imagem parcial de 30x30 *pixels* são feitas duas verificações:

- Se o valor médio dos *pixels* capturados possui uma determinada gama de cores previamente definida;
- Se o valor médio dos *pixels* capturados está a uma profundidade mínima daquilo que foi definido.

Caso a imagem analisada não cumpra estes requisitos o ciclo *for* repetirá este procedimento para os *pixels* que se seguem e sempre deste modo até analisar todos os *pixels* da imagem, (esta situação está ilustrada na Figura 36). Caso em algum momento a imagem parcial a analisar cumpra os requisitos será então dada a condição de "Verdadeiro" e a simulação segue para o próximo passo.

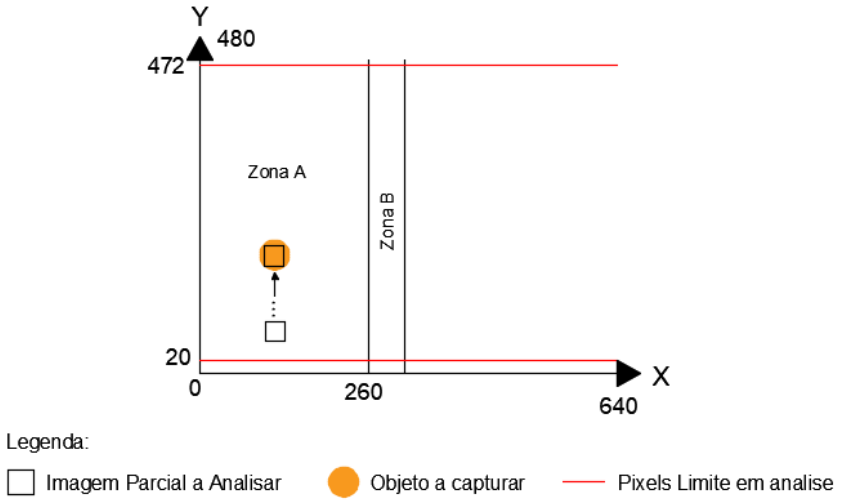


Figura 36 - Representação gráfica do procedimento que é feito aos *pixels* da imagem.

No próximo passo da simulação vão existir dois cenários que poderão acontecer:

- No primeiro cenário seriam detetados objetos na zona A ou na zona B e neste caso o algoritmo passaria para o procedimento ”Marcha=1” em que o rover robótico começaria a andar à velocidade “mínima” até o objeto ficar dentro da zona B e a garra completamente alinhada com o objeto a recolher;

- No segundo cenário não são detetados objetos nem na zona A nem na zona B e neste caso o algoritmo passa para o procedimento ”Marcha=0” em que o rover robótico começará a andar à velocidade “média” e analisará se durante o andamento surgem objetos na zona A, caso isso venha a acontecer o algoritmo passará para procedimento ”Marcha=1” que como já referido, o rover robótico reduz a velocidade e recolhe o objeto.

Os dois procedimentos referidos podem ser analisados nas Figuras 37 e 38.

```
--se houver objetos na zona A manda abrandar a velocidade do robo
if marcha == 0  then
    for p=20, 433, 7 do
    conf1=sim.getVisionSensorDepthBuffer(tv,1,p,30,30)
    conf2=sim.getVisionSensorImage(tv,1,p,30,30,0)
        for j=1, 900, 1 do
            hl=conf1[j]+hl
        end
        deep=hl/900

        for ww=0, 2697, 3 do
            h1=conf2[ww+1]+h1
            h2=conf2[ww+2]+h2
            h3=conf2[ww+3]+h3
        end
        r1=h1/900
        g1=h2/900
        b1=h3/900

        if fruta>=5 or ppl>=1 then
            anda=0
            estado=20
            localizacao=Probo[1]
            marcha=2
        elseif deep<= 0.77 and r1>=0.65 and b1<=0.2 then
            anda=-2
            marcha=1
            xx=250
            break
        end

        if marcha==0 then
        anda=-3
        end

        if (Pstop[1]-Probo[1])<=0.1 then
        anda=0
        estado=30
        marcha=2
        break
        end
        hl=0
        h1=0
        h2=0
        h3=0
    end
end
```

Figura 37 - Procedimento para "marcha=0".

```
--se houver objetos na zona B manda parar o robo  se o robo estiver cheio vai a base
if marcha == 1  then
    for p=20, 432, 5 do
    tab1=sim.getVisionSensorDepthBuffer(tv,xx,p,30,30)
    tab2=sim.getVisionSensorImage(tv,xx,p,30,30,0)
        for j=1, 900, 1 do
            m=tab1[j]+m
        end
        deep=m/900

        for ww=0, 2697, 3 do
            c1=tab2[ww+1]+c1
            c2=tab2[ww+2]+c2
            c3=tab2[ww+3]+c3
        end
        r=c1/900
        g=c2/900
        b=c3/900

        if fruta>=5 or ppl>=1 then
        anda=0
        estado=20
        marcha=2
        localizacao=Probo[1]

        elseif deep<= 0.77 and r>=0.65 and b<=0.2 then
            estado=1
            anda=0
            marcha=2
            break
        end

        if marcha==1 then
        anda=-2
        end

        if (Pstop[1]-Probo[1])<=0.1 then
        anda=0
        estado=30
        marcha=2
        break
        end

        c1=0
        c2=0
        c3=0
        m=0

    end
end
```

Figura 38 - Procedimento para "marcha=1".

Referir ainda que quer dentro do procedimento ”Marcha=0” ou no procedimento ”Marcha=1” será verificado se o número máximo de objetos a recolher foi atingido ou se o utilizador deu ordem para o rover robótico regressar à base ou ainda se foi atingida a distância máxima a percorrer pelo rover robótico. Pois caso algum destes casos acorra será ativada uma das seguintes subrotinas:

- Recolha de Objetos;
- Retornar à Base;
- Distancia máxima a percorrer.

De seguida é explicado o funcionado de cada uma destas subrotinas. No entanto, o código utilizado não será aqui transcrito por não trazer mais valor a este capítulo, podendo, como já referido ser consultado todo o algoritmo nos anexos deste documento.

4.2.1.1. Recolha de objetos

Foi definido um conjunto de movimentos que terão de ser feitos pelo rover robótico para a recolha de objetos. Estes movimentos estão descritos no fluxograma da Figura 39.

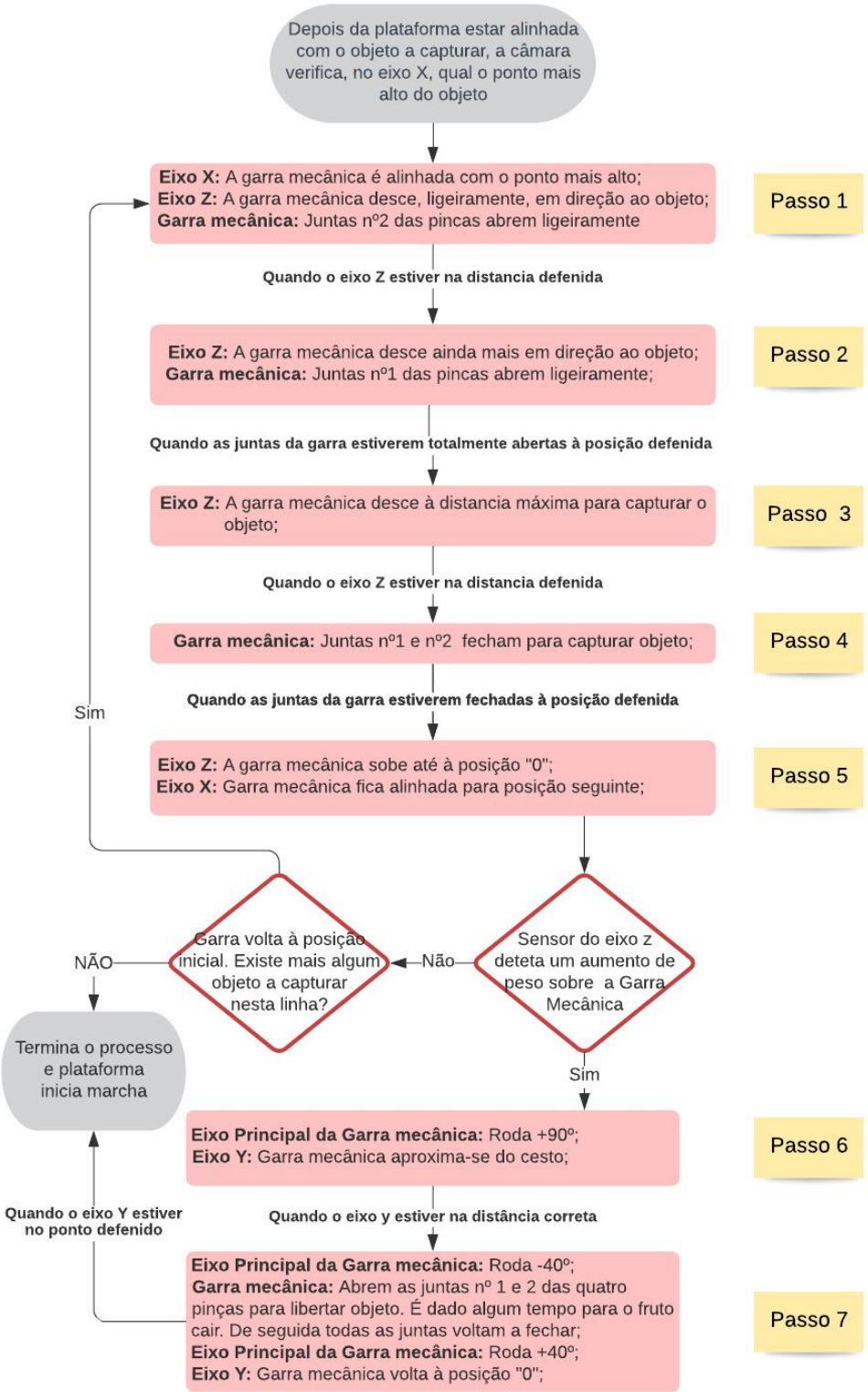


Figura 39 - Fluxograma para a recolha de objetos.

Do conjunto de movimentos definidos no ponto anterior, importa justificar algumas opções tomadas. Por exemplo, no início do fluxograma é dito que a câmara verifica o ponto mais alto no eixo X. Esta ação é feita porque é sabido que o rover robótico já parou de forma alinhada ao objeto (eixo Y), mas importa agora que o prisma nº 1 (que trabalha no eixo X) alinhe a garra mecânica com o objeto. Esta ação é exemplificada na Figura 40, em que o rover já está alinhado no eixo Y, e o ponto mais alto do objeto a recolher foi identificado com uma cruz.

Identificado o ponto mais alto, a garra dirige-se até ele (eixo X) e posteriormente a garra mecânica começará a descer (eixo Z) ficando completamente alinhada para recolher o objeto.

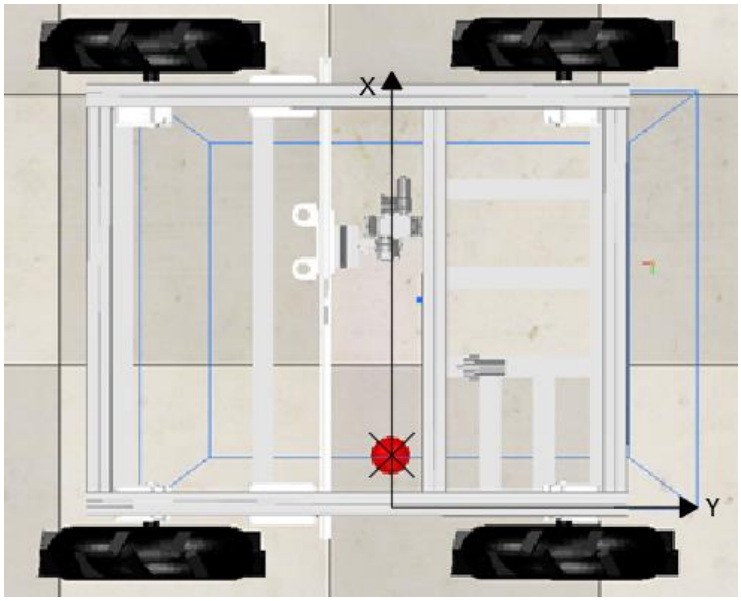


Figura 40 - Localização dos objetos a recolher.

Do fluxograma apresentado existe ainda outro aspeto que importa esclarecer, que é o caso da transição do passo 1 para o passo 2. Verificou-se que a garra mecânica passa demasiado perto da estrutura do rover, como observado na Figura 41. Assim torna-se necessário descer a garra mecânica até ultrapassar esta barra da estrutura e só depois abrir completamente as pinças, sendo que as pinças não poderão também abrir já numa posição demasiado baixa porque senão corre-se o risco de ao abrir tocarem no objeto. Assim, foi necessário encontrar um ponto ótimo para as pinças da garra abrirem.

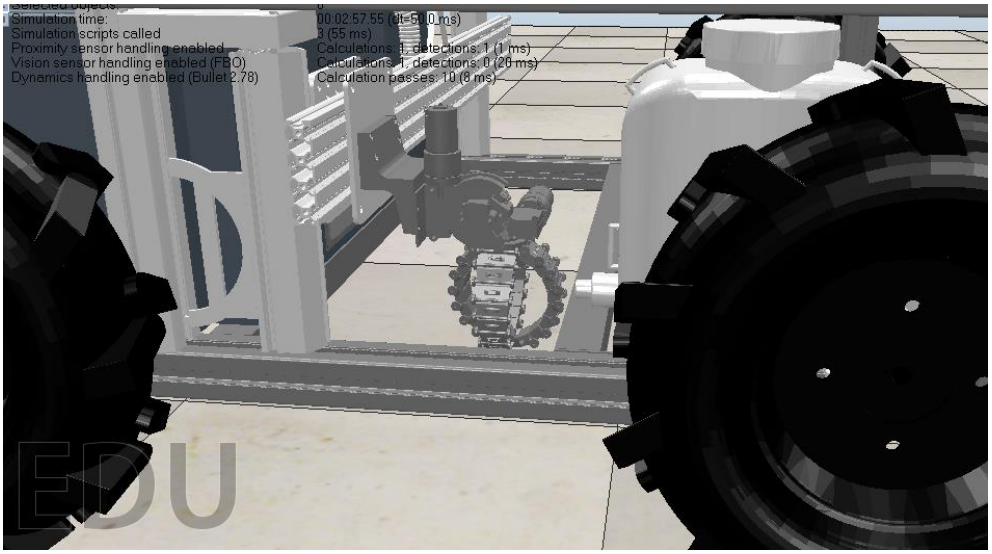


Figura 41 - Representação da garra mecânica aplicada no suporte do rover robótico.

Esta questão poderá ser facilmente ultrapassada se o avanço utilizado na fixação do eixo X à garra mecânica for melhorado ou então se o avanço dessa barra da estrutura do rover robótico for reduzida.

4.2.1.2. Retornar à base

Esta subrotina foi criada para que sempre que o cesto tenha atingido o número máximo de objetos a recolher ou tenha sido dada a ordem pelo operador para a rover robótico retorne à base, ao regressar à base, o rover robótico irá descarregar todos os objetos recolhidos e irá regressar ao local onde estava anteriormente à velocidade "máxima".

4.2.1.3. Distancia máxima a percorrer

Antes de iniciar a simulação é definido a localização de um *dummy* que representará o alcance máximo a que o rover robótico poderá ir. Quando o rover robótico recolher todos os objetos existentes até esse ponto, é dada ordem ao rover robótico para que este regresse à base e assim que chegue é dada por concluída a simulação.

4.2.2. Definição da velocidade a utilizar no rover robótico

Apresentado todo o algoritmo a utilizar nesta experiência, importa agora apresentar as velocidades de operação que foram aplicadas ao rover robótico. De acordo com o analisado na literatura científica as velocidades de operação utilizadas neste tipo de trabalhos são, por norma, baixas como K. Fue et al. 2020 e 2018 [149] e [150] relatam nos seus trabalhos. Os valores relatados na literatura científica são diversos, por exemplo Botterill et al. [151] no desenvolvimento de um sistema de poda automática de videiras utilizou uma velocidade de 0,25

[m/s], já K. G. Fue et al. [150] na criação de uma camara 3D para a deteção de fibras de algodão utilizou uma velocidade máxima de 0,17 [m/s]. Enquanto Zion et al. [20] na criação de um sistema robótico, multibraços, para a colheita de melões utilizou uma velocidade máxima de 0,1 [m/s].

Assim com base nos valores observados anteriormente, começou-se por realizar simulações com os valores mais elevados, isto é de 0,25 [m/s], que seria o pior cenário.

Para aplicar esta velocidade ao rover robótico foi necessário realizar alguns cálculos visto que a função da biblioteca do API que permite inserir a velocidade nas juntas rotacionais está em velocidade angular [rad/s], foi assim necessário converter a velocidade linear pretendida em velocidade angular a aplicar nas rodas do rover robótico. Para isso foi utilizada a Equação (1) da velocidade angular:

$$v = 2\pi \times R \times f \tag{1}$$

Em que:

- v = Velocidade Linear [m/s]
- R = Raio da roda do rover robótico que é de 0,211 [m];
- f = Frequência de rotação;

Aplicando assim os valores conhecidos na Equação (2), vem:

$$0{,}25 = 2\pi \times 0{,}211 \times f \Leftrightarrow f = 0{,}133 \text{ [Hz]} \tag{2}$$

Calculada a frequência [Hz] da roda, esta foi convertida para RPM e só depois para [rad/s], obtendo assim a velocidade angular que se terá de aplicar nas juntas rotacionais, neste caso será de 0,83 rad/s.

No entanto e após a realização de várias simulações observou-se que esta velocidade de operação não era elevada, pelo contrário verificou-se que podia ainda ser aumentada. Assim e após a realização de várias simulações conclui-se que o melhor valor a utilizar seria de 0,63 [m/s], isto para a velocidade "média" ou seja para quando o rover robótico está a locomover-se e a tentar detetar objetos na zona A da camara.

Assim que o rover robótico deteta algum objeto passa para a velocidade "mínima" que será de 0,42 [m/s] e irá parar quando estiver completamente alinhada com o objeto a recolher. Em relação à velocidade "máxima" foi definido um valor de 0,85 [m/s], trata-se de um valor elevado, mas uma vez que esta velocidade só será utilizado quando o rover robótico tiver de regressar à base ou seja em que não terá que ser feita nenhuma análise de imagem não foi verificado nenhum problema.

4.2.3. Recriação de um ambiente 3D semelhante ao de um campo de cultivo

Recriado um rover robótico idêntico ao existente no *software* CoppeliaSim e criado também um algoritmo para o controlo do rover robótico, importa agora recriar um ambiente semelhante aos campos de cultivo, onde o rover robótico irá operar.

Como já referido, este rover robótico foi projetado para a utilização em pomares de pessegueiros, nomeadamente nos pomares da região da Beira Interior, visto que só esta região representa cerca de 49,2% da produção a nível nacional [152].

Dado este facto, o modelo 3D a recriar será semelhante aos pomares desta zona. Segundo Simões [153] os compassos* mais frequentes nos pomares de pessegueiros da Beira Interior são 5 m x 2,5 m, 5 m x 3 m, 4,5 m x 2,5 m e 4,5 m x 2,75m. Sendo que a primeira medida está relacionada com a distância que deve existir entre cada planta e a segunda refere-se à distância entre as fileiras que o sulco forma, a entrelinha. Estas distâncias encontram-se representadas na Figura 42.

* Compasso: Distância necessária entre cada uma das plantas

Figura 42 – Definição de um Compasso, adaptado de [152].

Para a criação do modelo 3D foi utilizado o compasso de 4,5 m x 2,5 m por se verificar ser o "pior cenário", no sentido de que a entrelinha é a de menor largura e a que tem as árvores mais próximas umas das outras.

De seguida foi selecionada a textura a aplicar no plano de uso da simulação, tendo sido selecionada uma textura de tons castanho-escuro e com desenhos de relevos, não tendo, no entanto, quaisquer inclinações. O modelo final pode ser visto na Figura 43.

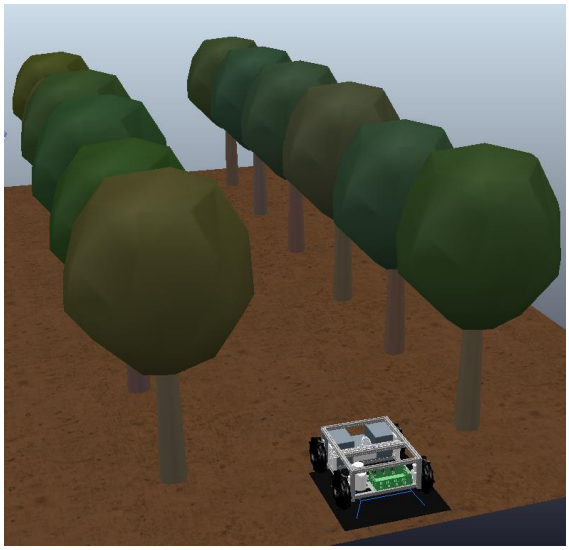


Figura 43 - Layout final utilizado nas simulações robóticas.

Para finalizar, foi necessário criar objetos que simulem os pêssegos que seriam recolhidos. Para determinar o tamanho e peso desses objetos foi tido em conta a classificação que M. P. Simões et al. [152] utiliza na avaliação do calibre e peso dos pêssegos e nectarinas. Esta classificação pode ser observada na Figura 44.

Designação	Intervalo medida equatorial (mm)	Peso (g) Intervalo de valores / média	
AAAA	≥ 90	>332	
AAA	80 a 90	245 - 332	290
AA	73 a 80	165 - 245	220
A	**67 a 73**	155 - 195	170
B	61 a 67	120 - 155	130
C	56 a 61	94 - 120	106
D	51 a 56	<94	

Figura 44 - Classificação dos pêssegos, consoante tamanhos e pesos, adaptado de [152].

Uma vez que os pêssegos podem ter vários tamanhos, optou-se por colocar na simulação objetos de vários calibres e pesos, com tamanhos entre os 50 e 90 mm de diâmetro e pesos entre as 100 e 150 g, sendo o peso maior atribuído aos objetos maiores, como observado na Figura 45. Todos estes objetos foram inseridos como formas do tipo primitiva e de geometria esférica.

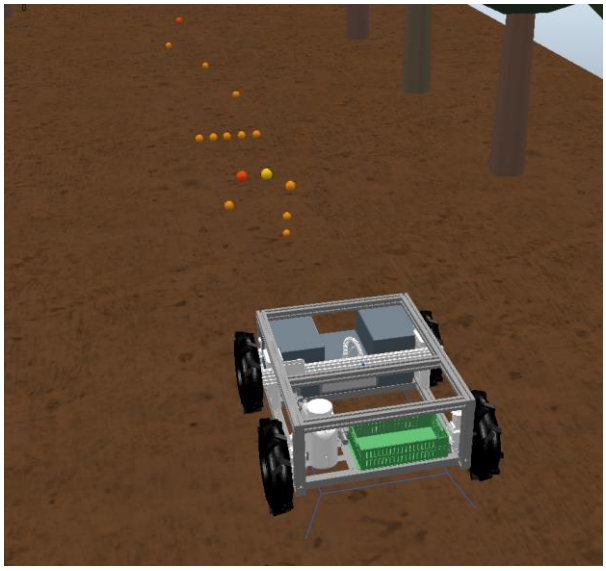


Figura 45 - Introdução de objetos a recolher na simulação.

Em relação à cor a atribuir a estes objetos, foram utilizados os tons laranja, visto ser a cor aproximada dos pêssegos. No entanto, foram utilizadas várias tonalidades de laranja para testar a capacidade do algoritmo.

4.3. Nota Conclusiva

Como indicado inicialmente, este rover robótico cumpre dois trabalhos, a pulverização de precisão ou a recolha de frutas caídas no solo da exploração agrícola que estão em fim de vida. No entanto e como pode ser observado ao longo deste capítulo, foi dado um maior ênfase para a recolha de frutas caídas, isto porque é uma atividade que engloba um conjunto de manobras, a realizar pelo rover robótico, mais complexas do que a atividade de pulverização. Ou seja, a primeira fase que é a deteção será muito idêntica para ambos os processos e a metodologia apresentada na deteção de frutos é facilmente replicada para a deteção de infestantes, tendo apenas que se ajustar alguns parâmetros, como a cor que os *pixels* do sensor de visão devem detetar. A segunda fase é que seria relativamente diferente, pois no caso da pulverização de precisão seria apenas necessário colocar bico pulverizador no local exato da infestante, através dos eixos X e Z. No caso da recolha de objetos seria também necessário colocar a garra mecânica no local do objeto a recolher, utilizando os mesmos eixos, mas obrigaria a um trabalho secundário que é a recolha e a colocação das frutas na embalagem que seguiria abordo.

Esclarecidos estes pontos e chegada a esta fase, tem-se todos os elementos necessários para a realização das diversas simulações computacionais e perceber se o raciocínio tomado foi o indicado ou não, existindo ainda assim alguns parâmetros que terão sempre de ser ajustados para os vários testes realizados. Todos estes aspetos serão discutidos no próximo capítulo.

5. Análise e Discussão de Resultados

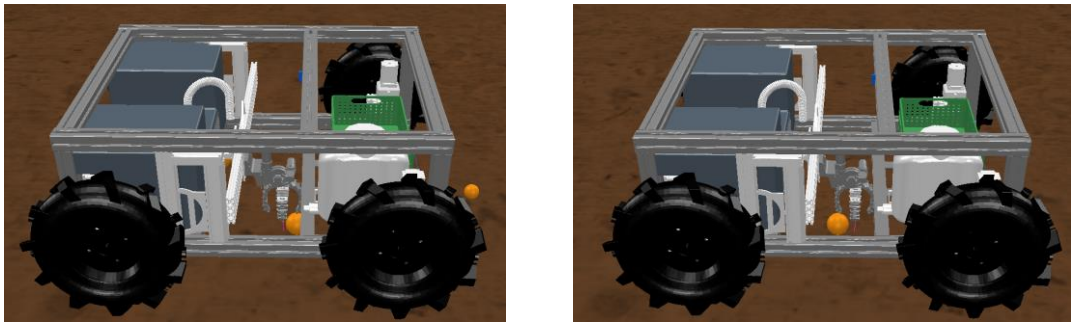
Neste capítulo é analisado e discutido o comportamento do rover robótico ao desempenhar as duas tarefas a que se propõe. Posteriormente, é apresentado um conjunto de 3 testes para cada uma das tarefas em que o rover robótico irá atuar. Concretamente, os testes experimentais têm como propósito avaliar a capacidade do algoritmo proposto em detetar objetos com diferentes parâmetros, e recolhê-los ou pulverizá-los, dependendo da operação a desempenhar. Por fim, é realizada uma breve análise aos resultados obtidos neste estudo.

5.1. Velocidade de Operação

Ao iniciar estas simulações já se tinha a noção que seria um processo algo moroso, visto a velocidade de operação ser baixa e o tamanho das áreas agrícolas alta. Inicialmente e para tentar melhorar este aspeto, tentaram-se utilizar velocidades de operação mais elevadas. No entanto, rapidamente se verificou que ao fazer este procedimento colocava-se em risco a deteção dos objetos, visto o tempo de visão da câmara ser inferior. Assim, a principal conclusão que se tirou foi que a velocidade de operação nunca deveria ser superior a 0,63 [m/s]. Sendo que para os valores referidos na literatura científica e já referenciados no subcapítulo 4.2.2, são até velocidades bastante elevadas.

Para além desta questão, importa ainda referir que o alinhamento da ferramenta em uso com o objeto ficava, muitas vezes, desajustado. No sentido em que o rover robótico ficaria sempre com um ligeiro avanço em relação ao objeto, esta situação está representada na Figura 46 e como se observa seria sempre mais grave quando a tarefa em curso fosse a recolha de objetos pois neste processo importa ter um bom alinhamento da garra mecânica com o objeto a recolher.

Inicialmente tentou-se corrigir este ponto acrescentando ao algoritmo uma condição que sempre que isto acontecesse o rover robótico se locomove-se para trás, até estar no alinhamento perfeito. No entanto, verificou-se que o tempo que se perdia neste processo era idêntico ao de colocar o rover robótico numa velocidade de operação inferior.



(a) (b)

Figura 46 - Exemplos da garra mecânica não estar alinhada com objeto.

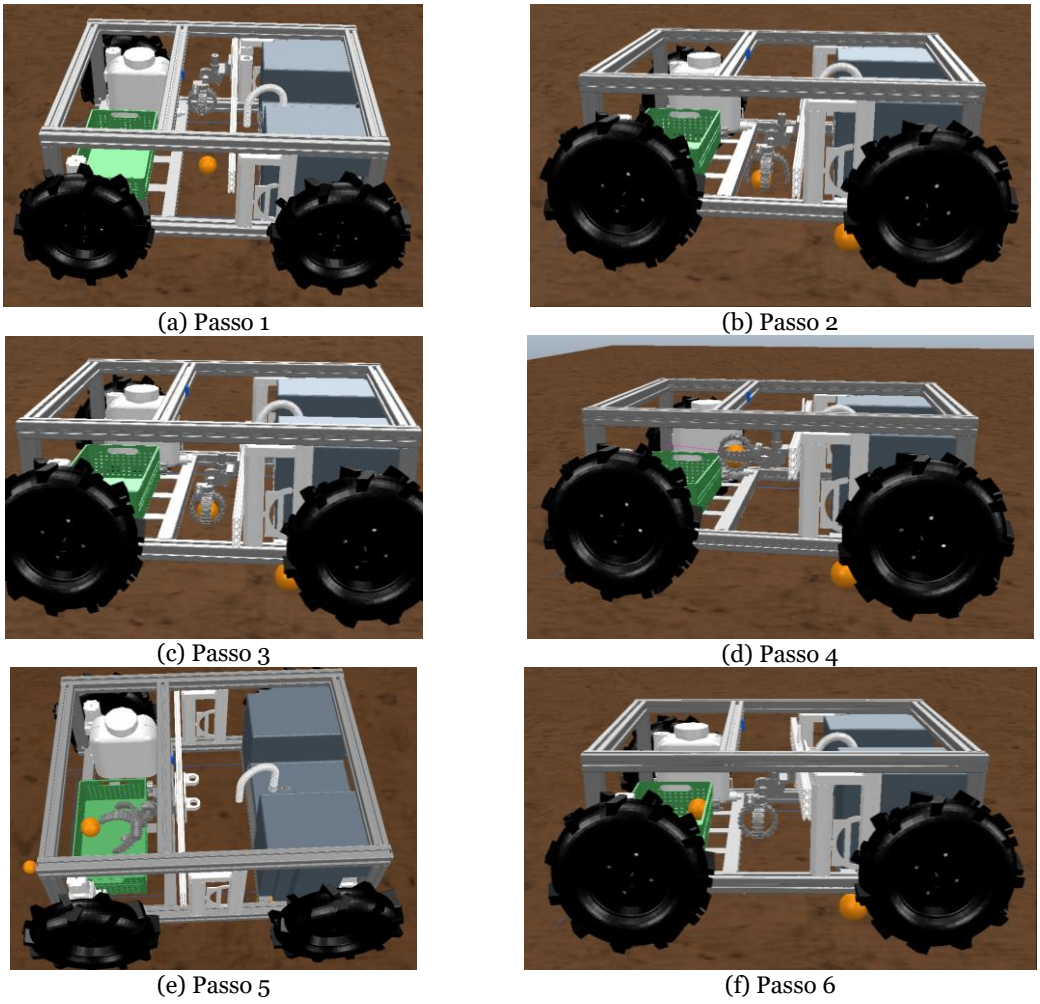
Por todos estes motivos optou-se pela estratégia de visualizar a câmara como duas zonas, como referido na Figura 27. Desta forma foi possível otimizar as velocidades de operação do rover robótico, utilizando para isso duas velocidades. Na fase de deteção de objetos, o rover robótico move-se à velocidade "média" de 0,63 [m/s] que é suficiente para detetar objetos e assim que estes são detetados o rover robótico passa a locomover-se à velocidade "mínima" de 0,42 [m/s], conseguindo assim realizar uma melhor análise ao objeto detetado e garantir ainda o abrandamento de velocidade necessário para que o rover robótico fique alinhado com o objeto a recolher ou a pulverizar.

5.2. Processamento de Imagem

Como já referido, o sensor de visão que está no rover robótico original permite várias resoluções e para melhorar a fluidez da simulação foi utilizada a resolução mais baixa do sensor de visão. No entanto, e decorrente do que se encontra em fóruns de utilizadores do CoppeliaSim e em outras plataformas, esta resolução continua a ser muito alta. Para este simulador normalmente são utilizadas resoluções inferiores e isso pode criar algum atraso. Aliada a esta questão está ainda o código criado, pois se este realizar muitos ciclos de código sobre o sensor de visão, torna a simulação ainda mais pesada. Foi também por isso que se tentou otimizar ao máximo o código a aplicar no processamento de imagem de forma a melhorar a fluidez da simulação.

5.3. Tempo de Operação

Para além dos pontos mencionados anteriormente que podem tornar a simulação mais morosa, existe ainda a questão de que quando a tarefa a realizar for a recolha de objetos o tempo de operação será ainda maior. Pois verificou-se que o tempo de recolher o objeto era também algo moroso no sentido de ser necessário realizar várias manobras, descritas no capítulo 4.2.1. e demonstradas na Figura 47.



(a) Passo 1 (b) Passo 2

(c) Passo 3 (d) Passo 4

(e) Passo 5 (f) Passo 6

Figura 47 – Sequência do rover robótico no processo de recolha de um objeto.

De acordo com a Figura 47 observa-se que no passo 1, o rover robótico para alinhado com o objeto, a garra mecânica corre no eixo x até ficar também alinhada com o objeto. No passo 2 o prisma do eixo z baixa a garra mecânica e são abertas as pinças da garra. Já no passo 3 o objeto é recolhido e por sua vez o eixo z volta a subir a garra mecânica até à posição inicial, a garra mecânica irá fazer uma rotação de 90º e o prisma do eixo y irá enviar a garra mecânica para a frente em direção à embalagem das frutas, como se observa nos passos 4 e 5. Sendo que no passo 5 já se vê o objeto a ser despejado na embalagem e no passo 6 a garra mecânica e todos os prismas voltam à posição inicial.

Para maximizar o tempo de operação na realização de todas estas manobras tentou-se aplicar algumas técnicas, desde melhorar o algoritmo em si, ou seja, quando é que cada uma das funções é ativada. Tentou-se ainda aumentar a velocidade de operação dos prismas ou mesmo alterar os parâmetros no controlo PID, que o *software* já dispõe para quando as juntas são colocadas no modo controlo de posição.

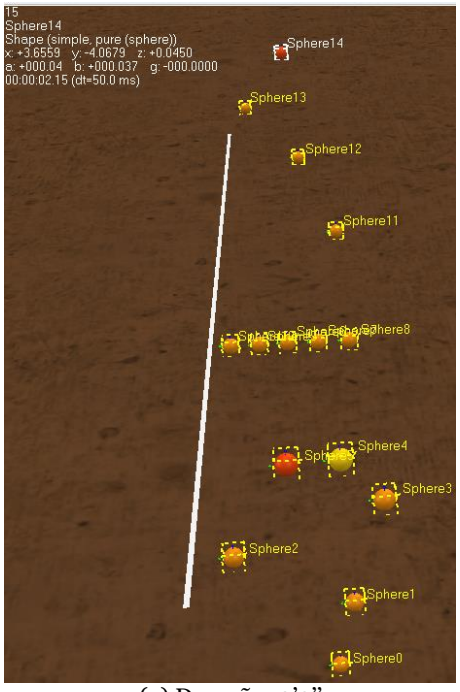
No entanto, as melhorias conseguidas foram mínimas e importa ainda referir que no caso de alterar as velocidades de operação e controlo PID dos eixos prismáticos é algo ambíguo, porque não adianta colocar um valor muito alto, se depois os prismas do rover robótico original não poderem operar nessas características.

5.4. Motor de Simulação Física

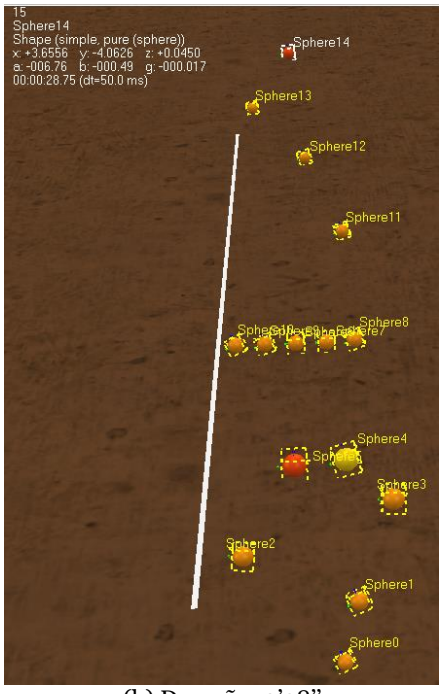
Como tinha sido referido no capítulo 3.3.3, o CoppeliaSim dispõe de quatro módulos de simulação dinâmica que servem para calcular, de uma forma realista, as interações que podem acorrer entre corpos rígidos. Nas várias simulações realizadas foram testados os quatro módulos pois e de acordo com Miranda [137], estes módulos funcionam principalmente por aproximações, e é interessante utilizar mais do que um para confirmar os resultados.

Das várias simulações realizadas observou-se que os resultados obtidos para os diferentes módulos de simulação eram semelhantes, à exceção do módulo *Bullet 2.78*, pois quando este era utilizado para a tarefa de recolha de objetos observou que, ao iniciar a simulação, os objetos a recolher por serem esféricos e o plano de uso completamente plano, começavam automaticamente a rodar pelo campo acabando por desaparecer. Este problema está representado na Figura 48 em que é possível verificar que nos primeiros 30 segundos os objetos não se movem muito, mas a partir daí os objetos começam a ganhar alguma aceleração e ao fim de 1 minuto já se notam diferenças e a partir daí as diferenças tendem a ser cada vez maiores visto que a aceleração dos objetos está constantemente a aumentar.

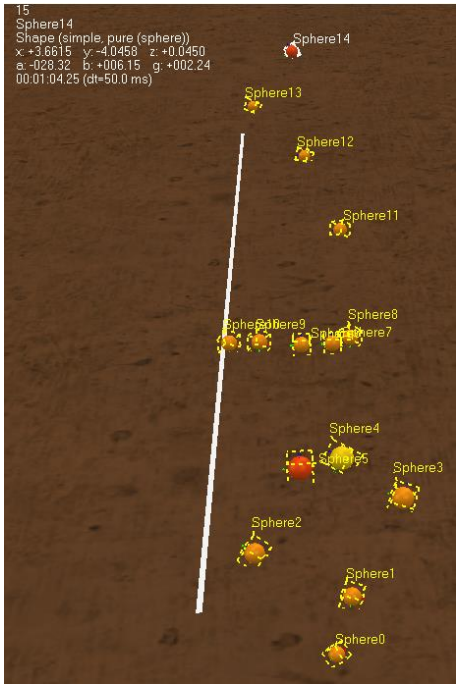
Sendo interessante verificar que os objetos tem tendência, na maioria dos casos, a rebolar para o lado esquerdo da cena, como demonstra a linha colocada no solo. Ainda se verificou se o plano de uso teria alguma inclinação, o que não era o caso. Isto acontecia mesmo colocando os parâmetros de fricção dos objetos e do plano de uso em valores máximos. Como tal este módulo foi desconsiderado para a operação de recolha de objetos por se entender que não correspondia à realidade da simulação.



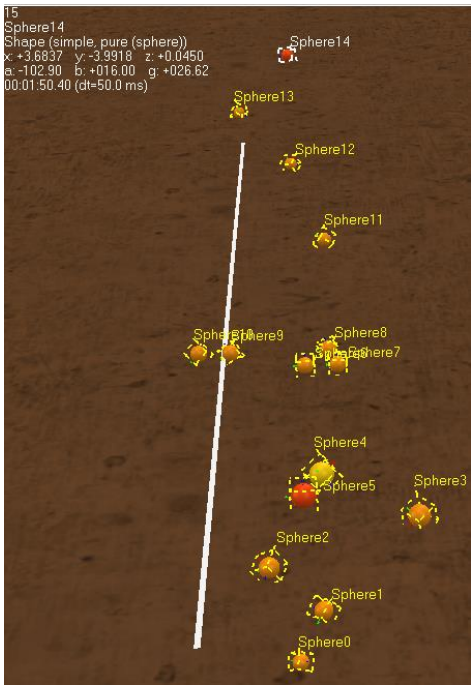
(a) Duração: 0'2"



(b) Duração: 0'28"



(c) Duração: 1'04"



(d) Duração: 1'50"

Figura 48 – Comportamento dos objetos a recolher com o módulo de simulação dinâmica *Bullet 2.78*.

Um outro aspeto que foi verificado nas várias simulações realizadas em que se alternava entre os motores de física que o simulador dispõe é que o processo de deteção de objetos era feito com êxito mas para o caso da operação de recolha de objetos, por norma, era preciso realizar alterações nos parâmetros das juntas da garra mecânica e dos eixos dos prismas para que a recolha fosse feita com êxito. No entanto e uma vez que os motores físicos simulam aproximações à realidade é normal que surjam algumas diferenças especialmente quando se está a realizar uma etapa tão

meticulosa e específica. Além disso Armesto [154] refere esta condição e até sugere o uso de um "*fake pick up*" para ultrapassar este problema. Assim, é dado como irrelevante este aspeto, pois o importante é que caso esta simulação seja replicada à realidade que estes parâmetros sejam ajustados à realidade dos diferentes motores e prismas.

Concluída a análise aos vários módulos de simulação dinâmica que o *software* dispõe optou-se por utilizar o módulo *Newton Dynamics* por se entender ser o módulo que melhor representava a realidade de um campo de cultivo e do comportamento do rover robótico.

5.5. Diferenciação das Cores nos Pixéis da Imagem

Como referido a deteção de objetos foi feita apenas em algumas zonas da imagem e um dos parâmetros utilizados foi a verificação da gama de cores que estava a ser lida em determinados *pixels*.

A gama de cores utilizada no CoppeliaSim é a do sistema RGB, que é um sistema de cores aditivas em que o Vermelho (*Red*), o Verde (*Green*) e o Azul (*Blue*) são combinados de várias formas de modo a reproduzir um largo espectro cromático. Este sistema é constituído por um conjunto de três códigos de cores em que cada um desses códigos corresponde à quantidade de vermelho de verde e de azul que essa cor tem [155].

Por norma, a gama de cores mais usual vai de 0 a 255 em que 0 é completamente escuro e 255 completamente intenso. No entanto, o CoppeliaSim funciona com uma gama de cores que vai do 0 a 1 sendo o mesmo princípio da gama 0 a 255.

Assim, para realizar a deteção de objetos foi necessário analisar a gama de cores dos objetos a utilizar em cada uma das tarefas, os valores medidos pelo sensor de visão para cada um destes objetos está descrito na Tabela 7.

Tabela 8 - Codigo de cores utilizado.

Item	Código de Cor		
	Vermelho	**Verde**	**Azul**
Plano de uso	0,376	0,258	0,164
Pêssego amarelado	0,933	0,772	0,043
Pêssego alaranjado	0,976	0,584	0,082
Pêssego avermelhado	0,992	0,309	0,081
Infestante de Tonalidade n.º1	0.234	0.775	0.074
Infestante de Tonalidade n.º 2	0.031	0.473	0.030
Infestante de Tonalidade n.º 3	0.191	0.509	0.331
Infestante de Tonalidade n.º 4	0.208	0.740	0.207

Para o caso da tarefa a realizar ser a recolha de objetos é percetível, numa primeira análise, que existe uma grande diferença no código da cor Vermelha entre o plano de uso e dos vários tons dos objetos a recolher, sendo que também existe alguma diferença no código verde e azul. No entanto, para o algoritmo criado, as condições que foram definidas para que seja dada a condição de "Verdadeiro" é existir um valor médio nos *pixels* da imagem parcial a analisar com código de vermelho superior a 0.8 e um código de azul inferior a 0.2. Foram consideradas estas condições porque para as várias tonalidades que o pêssego possa ter, o código de cor vermelho terá sempre um valor alto e o azul apenas para garantir que não seja recolhido um qualquer objeto que seja muito claro, isto porque a cor branca tem uma gama de cores perto do valor 1 nos três códigos.

Já para o caso da tarefa de pulverização de infestantes as principais diferenças que se notam é para o código da cor vermelha que é inferior e para o código da cor verde que é superior, sendo na cor verde onde se nota a maior diferença, dado ser a cor natural das infestantes. Assim para ser dada a condição de "verdadeiro" os requisitos a cumprir são código vermelho menor de 0.24, verde superior a 0.32 e o azul inferior a 0.34.

5.6. Tamanho dos Objetos a Recolher

Realizar deteção de objetos considerando apenas a gama de cores é algo pouco seletivo, no sentido de existir a possibilidade de aparecer um objeto com a mesma gama de cores, sendo esta possibilidade ainda maior quando se está a operar num ambiente não estruturado, em que não se consegue controlar as várias variáveis que podem surgir.

Assim para o caso da tarefa de recolha de objetos foi adicionada uma outra condição, em que sempre que a imagem parcial que está a ser analisada e que cumpra com a gama de cores pré selecionada, esta imagem terá também que estar a uma determinada profundidade mínima para que seja dado o resultado lógico de verdadeiro.

De referir que o valor, médio, de profundidade medido pelo sensor de visão é de 0,9 cm em terreno plano sem quaisquer objetos a passar no plano de uso, mas quanto um objeto passar pelo plano de uso e quanto mais alto for este objeto menor será o valor de profundidade medido pelo sensor.

Assim e uma vez que o intuito do algoritmo é detetar objetos de vários tamanhos, o valor atribuído para que esta condição se torne verdadeira é de 0,51 cm, ou inferior. Sendo que este valor garante que os objetos que terão o diâmetro menor (que é de 0,5 cm) sejam detetados e se o objeto tiver um diâmetro maior, estará sempre garantida a condição.

De referir que para o caso da tarefa de pulverização de infestantes esta condição não foi incluída por se considerar de difícil aplicação prática uma vez que o ambiente não é estruturado e as infestantes tem sempre tamanhos indefinidos, o que não acontece com o caso da tarefa de objetos a recolher.

5.7. Recolha de Objetos: Caso de estudo 1

Definidas todas as variáveis no algoritmo, no rover robótico e nas definições do próprio *software*, foi realizado um conjunto de testes para avaliar a robustez do algoritmo criado. Neste primeiro caso de estudo foi testada a capacidade do algoritmo na recolha de objetos. Para isso foi colocado um conjunto de 9 objetos, a recolher, espalhados de forma aleatória ao longo de um trajeto de 12 metros que o rover robótico irá percorrer.

Adicionalmente foram definidos três possíveis cenários:

- Cenário 1: Objetos com um diâmetro igual e com diferentes cores;
- Cenário 2: Objetos com vários diâmetros e a mesma gama cores;
- Cenário 3: Diferentes tamanhos e gama de cores.

De salientar ainda que para cada um dos cenários foram realizados três testes diferentes, em que cada um dos teste os objetos eram colocados em posições aleatórias tal como o parâmetro que estaria em análise.

5.7.1. Cenário 1

Na realização deste primeiro cenário foram colocados no plano de uso os 9 objetos com um diâmetro de 70 mm. A escolha deste tamanho é justificada porque ser o valor médio que os pêssegos podem ter, tal como descrito no capítulo 4.2.3.

A gama de cores utilizada foi a demonstrada na Tabela 7, e as posições que foram dadas aos objetos para cada um dos testes é a representada na Figura 49. Referir que os marcos demonstrados a amarelo nas várias imagens representam o local onde o rover robótico iniciou marcha e onde esta terminou.

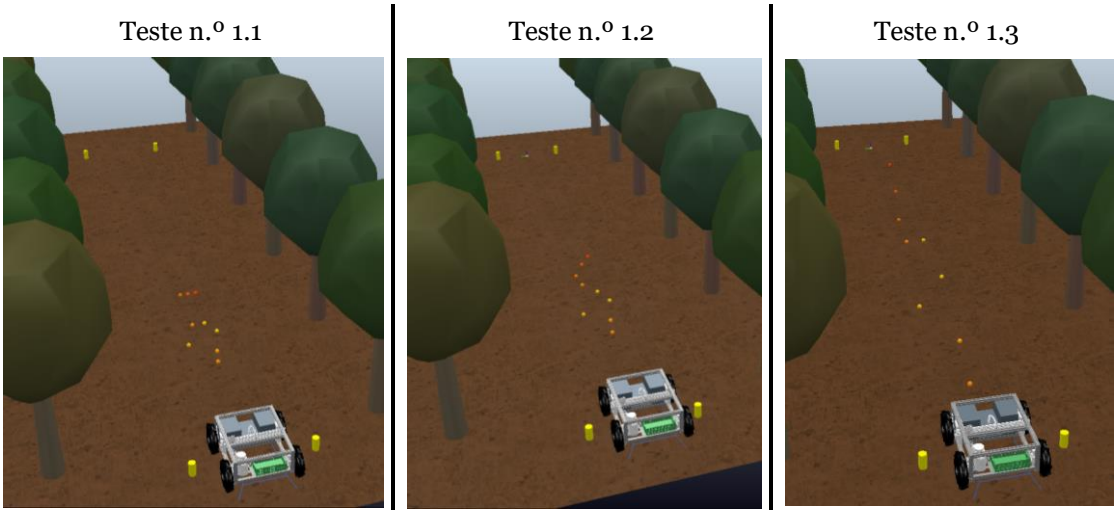


Figura 49 - Testes realizados para o cenário 1.

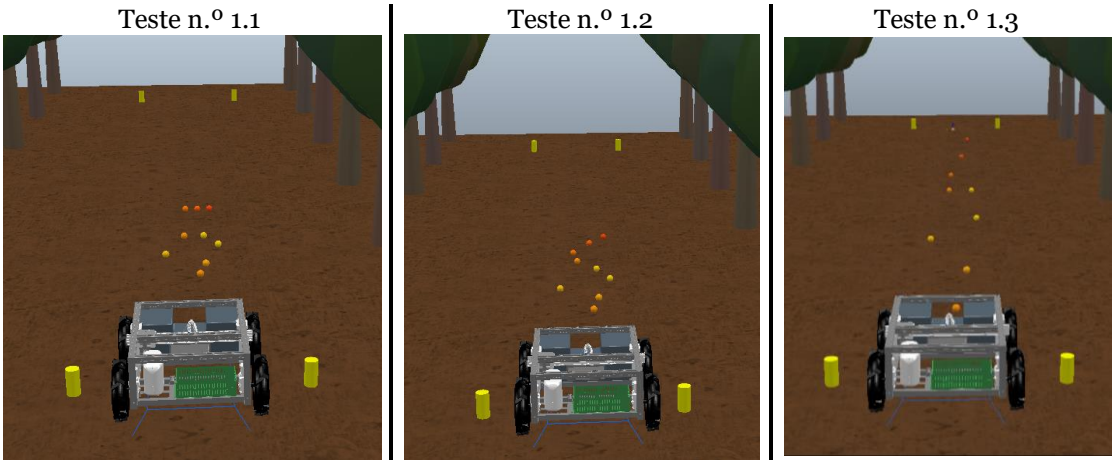


Figura 49 - Testes realizados para o cenário 1 (cont.).

5.7.2. Cenário 2

Neste segundo cenário foram atribuídos aos 9 objetos diâmetros que variam entre os 50 e 90 mm. Em relação à gama de cores, foi aplicada a cor do pêssego alaranjado (descrita na Tabela 7), para todos os 9 objetos. As posições utilizadas para os três testes é a representada na Figura 50.

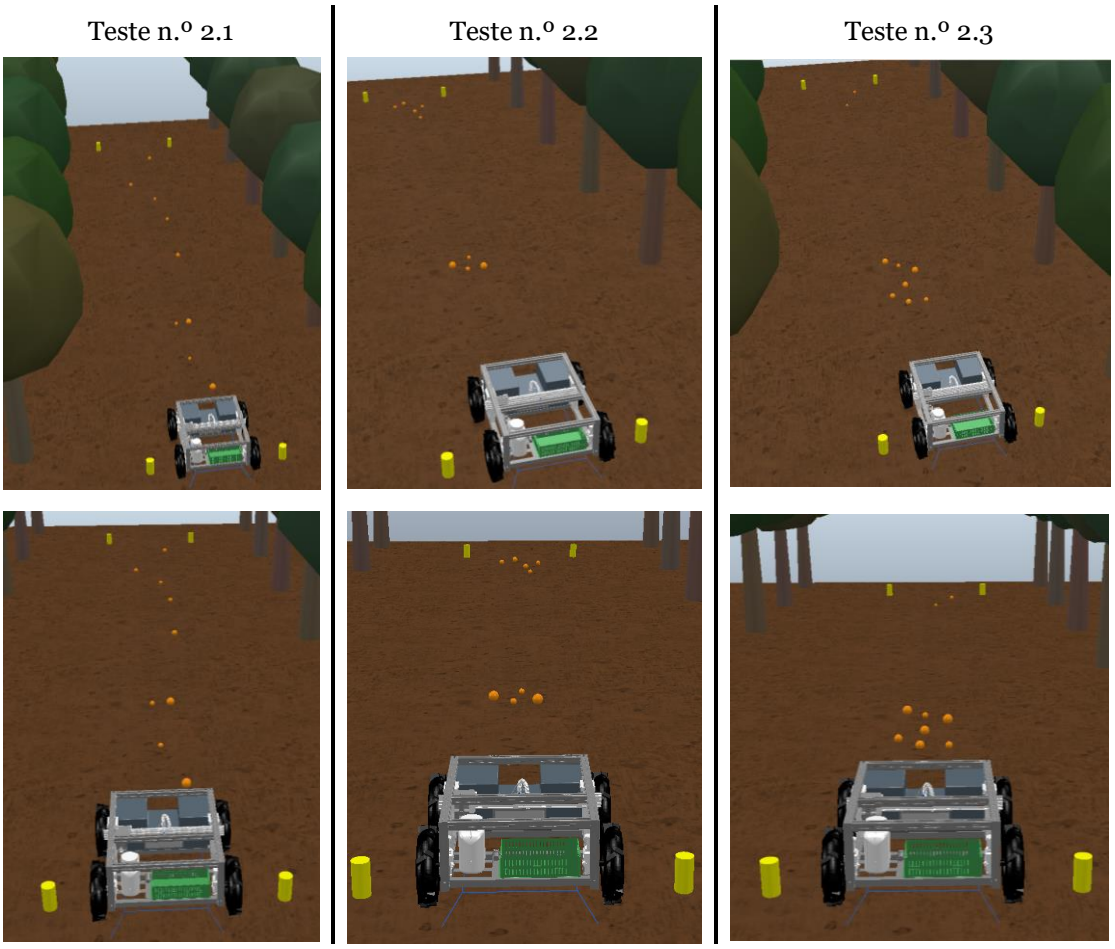


Figura 50 - Testes realizados para o cenário 2.

5.7.3. Cenário 3

Neste último cenário foram atribuídos aos 9 objetos várias gamas de cores, todas elas entre os valores apresentados na Tabela 7 e vários diâmetros que variam entre os 50 e 90 mm. As posições utilizadas para os três testes é representada na Figura 51.

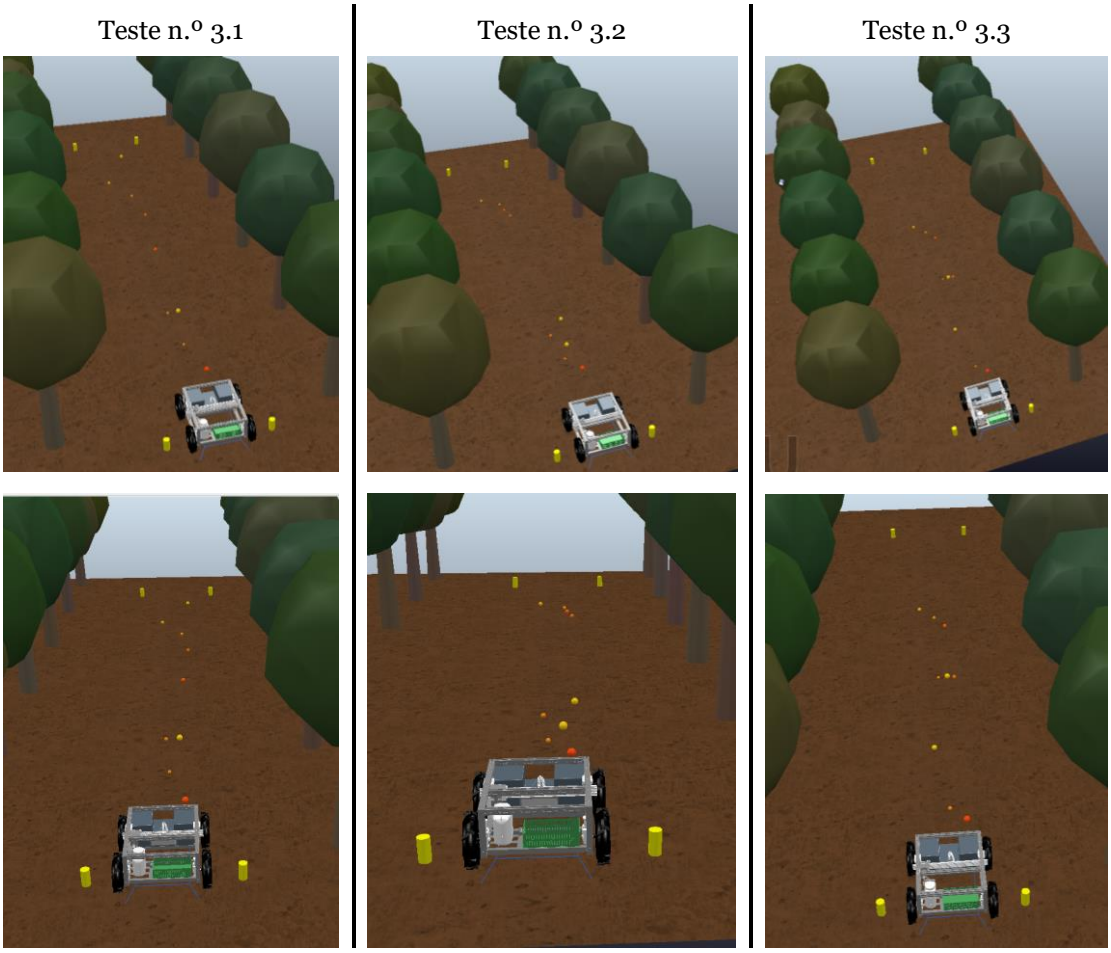


Figura 51 - Testes realizados para o cenário 3.

5.8. Pulverização Controlada: Caso de estudo 2

À semelhança do realizado no caso de estudo anterior, neste foi também criado um conjunto de cenários em que o rover robótico terá de operar, sendo que para cada cenário foram também feitos 3 testes. A diferença deste caso de estudo é que os objetos a detetar terão propriedades idênticas às das infestantes e em vez de serem recolhidos, serão pulverizados. Os cenários que foram definidos são:

- Cenário 1: Infestantes com formas e cores iguais;
- Cenário 2: Infestantes com formas iguais mas com diferentes gamas de cores;

- Cenário 3: Infestantes com formas e gamas de cores diferentes;

De referir que a trajetória a percorrer continuará a ser de 12 metros e o número de objetos a pulverizar será igualmente 9. Na Tabela 7 e na Figura 52 podem observar-se as gamas de cores utilizadas e as formas que estes objetos poderão ter.

(a) Infestante n.º 1 (b) Infestante n.º 2 (c) Infestante n.º 3

Figura 52 – Tipos de formas utilizadas na representação de infestantes.

5.8.1. Cenário 1

Para este primeiro cenário, o objeto que foi utilizado foi o da "Infestante n.º 1", por ser o objeto de maior densidade e por isso o que representa maior facilidade, à partida, em ser detetado. Relativamente à cor a utilizar, optou-se pela "tonalidade n.º 2". Na Figura 53 pode-se observar as posições em que foram colocados os objetos para os três testes realizados.

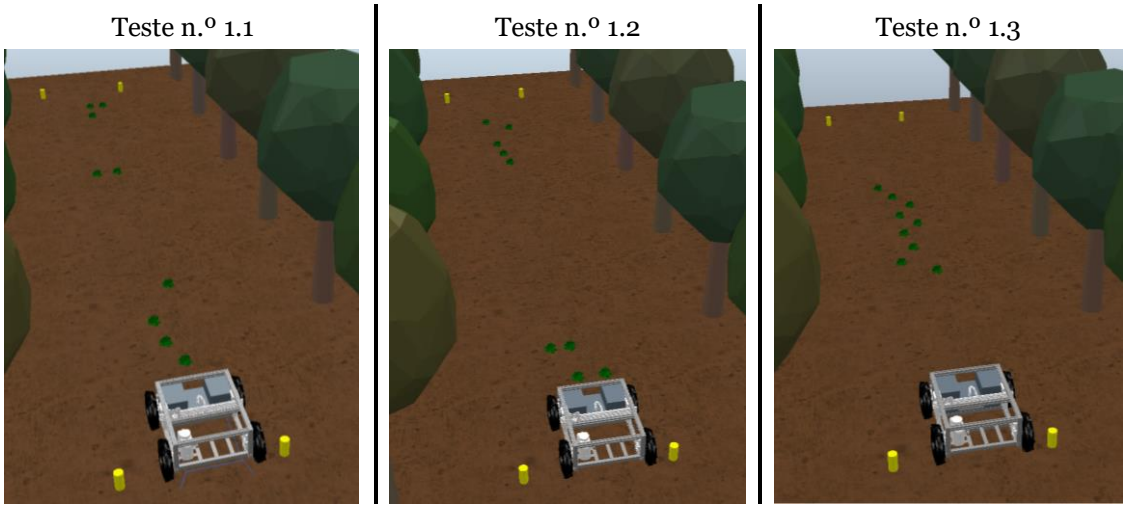


Figura 53 - Testes realizados para o cenário 1.

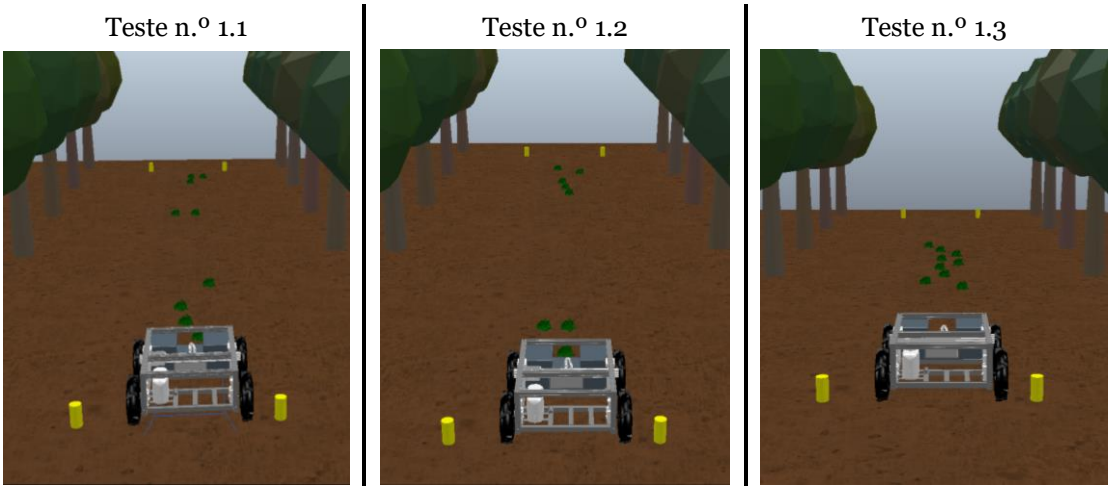


Figura 53 - Testes realizados para o cenário 1 (cont.).

5.8.2. Cenário 2

Neste segundo cenário voltou-se a utilizar o objeto "Infestante n.º 1", com a diferença que agora foram utilizadas as 4 tonalidades de cores apresentadas na Tabela 7. As posições utilizadas para os três testes são as representadas na Figura 54.

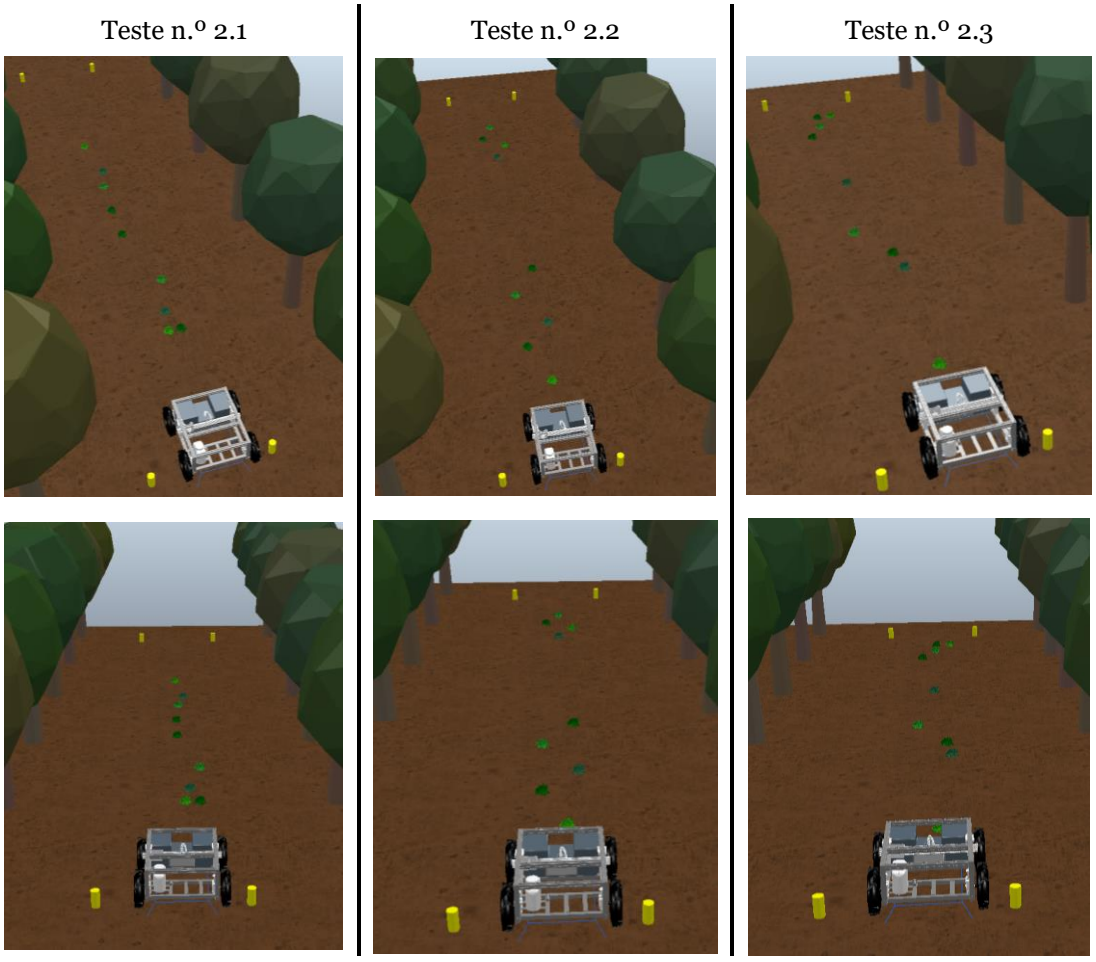


Figura 54- Testes realizados para o cenário 2.

5.8.3. Cenário 3

Para este último cenário foram colocados 3 objetos de cada um dos tipos de formas apresentadas na Figura 52. Foram ainda utilizadas as 4 gamas de cores nos diversos objetos. As posições utilizadas para os três testes são as representadas na Figura 55.

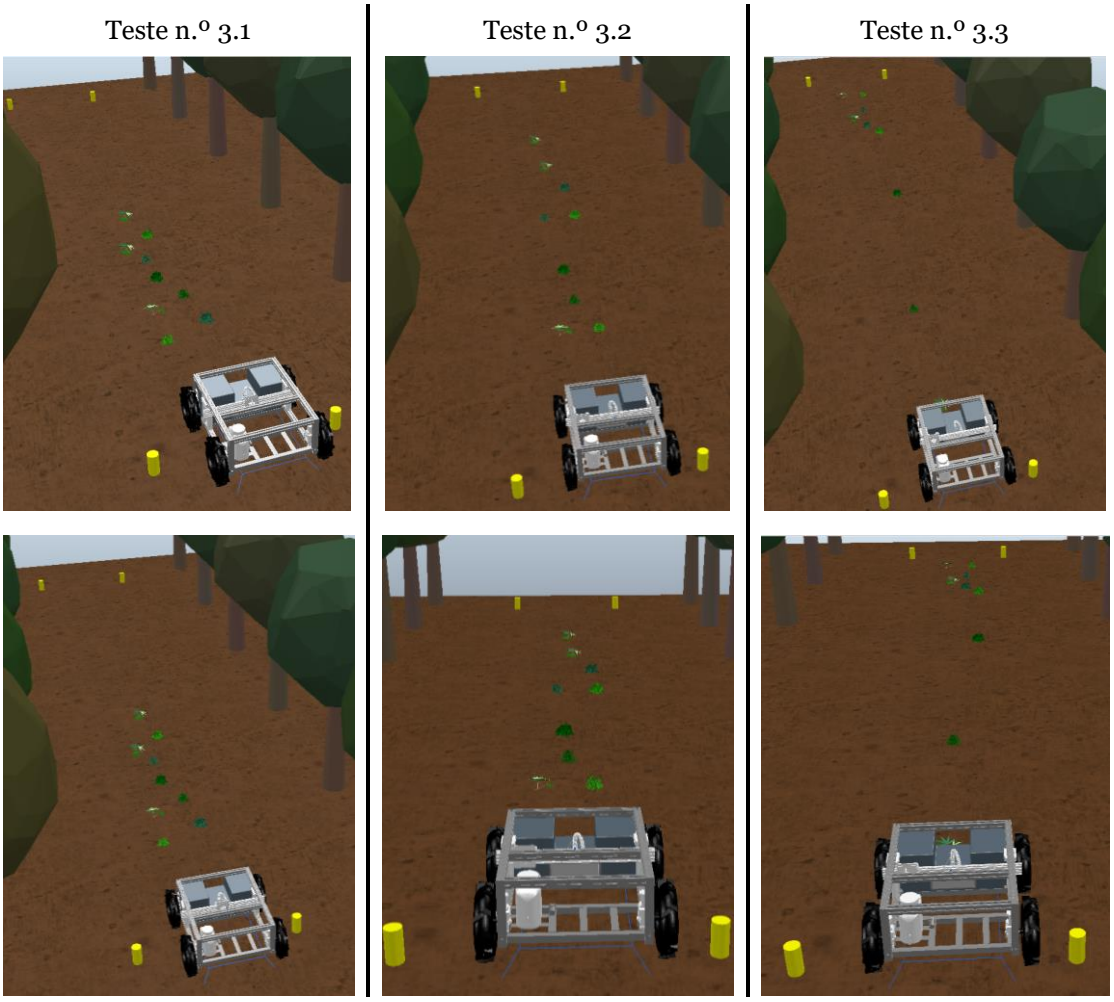


Figura 55 - Testes realizados para o cenário 3.

5.9. Análise de Resultados

Demonstradas as principais questões que surgiram no decorrer das várias simulações realizadas e demonstradas também as simulações que foram feitas para os dois casos de estudo, pode-se concluir que o algoritmo proposto teve bastante êxito.

Em ambos os casos de estudo, o algoritmo conseguiu sempre detetar os objetos que se pretendiam recolher, bem como realizar os passos que se seguiam à deteção do objeto.

5.9.1. Resultados do caso de estudo 1

Neste caso de estudo a deteção dos objetos foi conseguida nos diferentes cenários que se propunham. Incluindo no cenário 3, que seria o que representa maior dificuldade porque todos os objetos eram diferentes, quer a nível de cor, quer a nível de tamanho. Referir que a única dificuldade identificada foi na recolha dos objetos maiores, especialmente com diâmetros superiores a 85 mm, tendo sido necessário aumentar ligeiramente o ângulo de abertura das juntas da garra mecânica, como verificado na Figura 56.

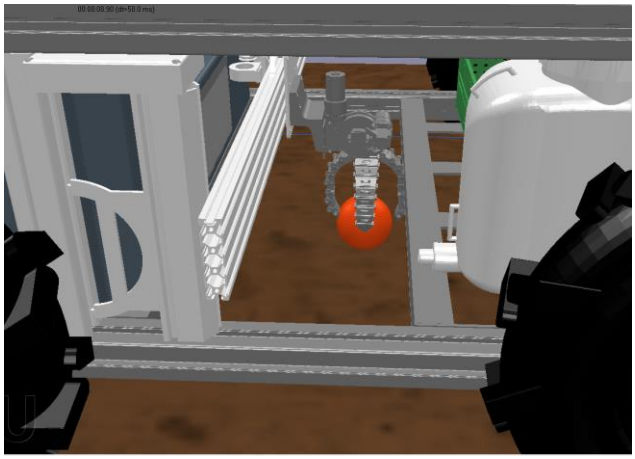


Figura 56 - Exemplo de recolha de objeto com dimensões maiores.

Para as várias simulações realizadas em cada cenário foram determinados os tempos de operação que foram necessários para o rover robótico concluir a tarefa de recolha dos objetos. Estes tempos estão identificados na Tabela 8. Depois de observados os tempos, foi ainda feita outra simulação que residiu na limitação do número máximo de objetos a recolher para 5 objetos, o que obrigaria o rover a realizar uma ida à base a meio do trabalho de recolha de objetos.

Tabela 9 -Tempos decorridos na realização das várias simulações do caso de estudo 1.

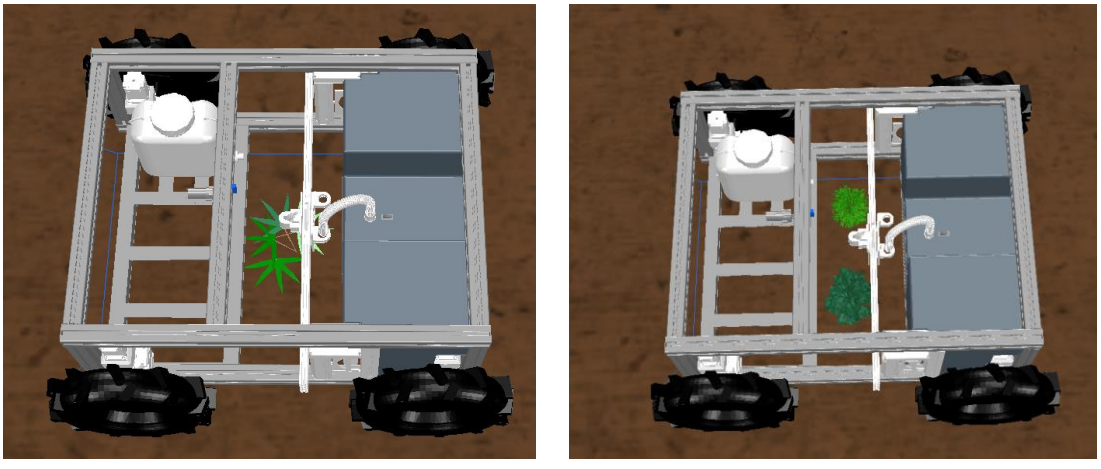
Caso de Estudo 1		Tempo sem paragem	Tempo com paragem
Cenário 1	Teste n.º 1.1	3'10"	3'14"
	Teste n.º 1.2	3'10"	3'16"
	Teste n.º 1.3	3'10"	3'26"
Cenário 2	Teste n.º 2.1	3'11"	3'19"
	Teste n.º 2.2	3'10"	3'38"
	Teste n.º 2.3	3'08"	3'14"
Cenário 3	Teste n.º 3.1	3'13"	3'21"
	Teste n.º 3.2	3'10"	3'16"
	Teste n.º 3.3	3'14"	3'20"

Como observado na Tabela 8, o facto de obrigar a uma ida à base não aumentou excessivamente o tempo total de operação. Referir que estes tempos de operação incluem o tempo que o rover robótico demora a recolher todos os objetos a chegar ao local de término e ainda o regresso do rover robótico à base.

5.9.2. Resultados do caso de estudo 2

Para este segundo caso de estudo a deteção foi também conseguida, independentemente do cenário em questão. No entanto e para conseguir obter estes resultados, houve a necessidade de redefinir os parâmetros da gama de cores a detetar, pois inicialmente tinham sido definidos valores mais baixos para o código vermelho e valores mais elevados para o código verde. No entanto, esses valores tiveram de ser reduzidos para os descritos no subcapítulo 5.5. Esta situação ocorreu devido aos cenários 2 e 3 em que existiam mais gamas de cores para detetar.

Outro aspeto que se verificou na operação do rover robótico foi que, caso a infestante a detetar fosse demasiado grande, corria-se o risco de o rover a pulverizar por duas vezes. Esta situação verificou-se mais para a "infestante n.º 3" (Figura 57 a), dado ter um tamanho relativamente superior às outras duas. No entanto, esta situação acontecia apenas esporadicamente. Para corrigir esta situação, optou-se por adicionar uma condição ao algoritmo em que sempre que era pulverizada uma infestante, o algoritmo iria ignorar os 100 *pixels* em redor da infestante pulverizada. Assim, para os vários testes realizados, o algoritmo já funcionou corretamente, mesmo quando se tinha duas infestantes relativamente próximas, o algoritmo conseguia detetar as duas e pulverizava ambas (Figura 57 b).



(a) Pulverização da infestante n.º 3. (b) Pulverização de várias infestantes.

Figura 57 - Deteção de infestantes.

Para este caso de estudo foram também determinados os tempos de operação necessários para o rover robótico percorrer todo o percurso, detetar todas as infestantes e voltar à base. Estes tempos podem ser observados na Tabela 9.

Tabela 10 - Tempos decorridos na realização das várias simulações do caso de estudo 2.

Caso de Estudo 2		Tempo
Cenário 1	Teste n.º 1.1	1'08"
	Teste n.º 1.2	1'06"
	Teste n.º 1.3	1'05"
Cenário 2	Teste n.º 2.1	1'12"
	Teste n.º 2.2	1'09"
	Teste n.º 2.3	1'06"
Cenário 3	Teste n.º 3.1	1'09"
	Teste n.º 3.2	1'05"
	Teste n.º 3.3	1'05"

Para este caso de estudo não foi realizado o teste com uma paragem a meio, por se ter verificado no caso do estudo anterior que a diferença de tempo não era substancial e também porque para a tarefa em questão, um depósito da solução a utilizar dará para muitas utilizações, não fazendo por isso sentido simular esta ação.

5.10. Nota Conclusiva

Concluídas as simulações e analisados os resultados obtidos para cada um dos casos de estudo, é possível retirar algumas ilações, nomeadamente que quer o processo de recolha quer o processo de pulverização de objetos não é tão moroso como se julgou inicialmente, pois os tempos de operação são até relativamente curtos.

No entanto, é possível verificar que os tempos de operação para o caso da tarefa de recolha de objetos são bastante superiores aos tempos da tarefa de pulverização, o que até era já esperado dada a análise que tinha sido efetuada no subcapítulo 5.3.

Para além destes pontos, é necessário não esquecer que a distância percorrida foi curta quando comparada com um campo de cultivo, já o número de objetos a recolher é de difícil análise e aleatório. De qualquer forma, estes valores bem como outros aspetos serão melhor dissecados no próximo capítulo.

6. Conclusões

Neste último capítulo são apresentadas as principais conclusões do trabalho desenvolvido, são descritas as principais dificuldades sentidas e são perspetivadas algumas ideias para trabalhos futuros.

6.1. Considerações Finais

Chegado aqui é possível retirar inúmeras conclusões. Em primeiro lugar que sem dúvida que este tipo de simuladores são uma mais-valia para o avanço dos sistemas robóticos, especialmente quando não existe a possibilidade de aplicar a simulação à realidade.

No entanto, ficam sempre algumas dúvidas sobre os resultados obtidos por se pensar que o comportamento de determinado componente ou por determinada interação poderem vir a ser diferentes na realidade. Mas, no entanto, a essência em si do que pode acontecer em determinada simulação é apresentada e muitas vezes são apenas pequenos pormenores técnicos que se sabe que terão de ser ajustados quando a simulação for realizada em ambiente real.

Esta questão é abordada na literatura científica, por exemplo Nehmzow [156], quando descreve algumas das desvantagens no uso de simuladores, menciona a complexidade da construção de um ambiente que simule com precisão, todas as combinações possíveis das interações, robô-tarefa-ambiente. Indicando que é difícil que um simulador genérico consiga simular com fiabilidade todas as interações possíveis entre estes três componentes.

No caso concreto deste estudo, surgem também algumas dúvidas sobre alguns resultados obtidos, nomeadamente no caso da velocidade de operação que terá sempre de ser adaptada ao terreno em causa e verificar se na realidade o aumento de velocidade afetará significativamente a imagem e consequentemente a deteção de objetos, em particular, frutos. Ainda em relação à velocidade, importa referir que todas estas simulações foram realizadas considerando um plano de uso sem inclinações e é sabido que os campos agrícolas têm muitas inclinações e irregularidades, pelo que estes aspetos devem ser tomados em conta na determinação da velocidade de operação. Além disso, o próprio autor deste rover robótico, Veiros [12], refere que dos testes de simulação realizados ao rover robótico, este mostrou ter alguns problemas em transpor obstáculos de maior dimensão e problemas de tração em solo molhado.

De referir que não foram utilizadas inclinações ou irregularidades no plano de uso, porque no algoritmo criado não foi previsto nenhum módulo que realizasse a deteção de obstáculos, perda de tração ou de sentido.

Pois os objetivos deste trabalho foram abrangentes, pois foi necessário implementar o modelo do rover robótico, com todos os seus aspetos construtivos e especificações técnicas no simulador, criar um ambiente 3D semelhante ao de um campo agrícola, criar um sistema de deteção de objetos e desenvolver um algoritmo de controlo para que todos os eixos funcionassem em harmonia na recolha dos objetos

Em relação ao processamento de imagem importa referir que a deteção de objetos foi feita inicialmente verificando apenas a cor e profundidade de um único *pixel* e, na verdade, para as várias simulações feitas a deteção foi realizada quase sempre com êxito. Mas por se considerar que este método poderia não ser muito fidedigno, principalmente se o algoritmo fosse aplicado à realidade. Decidiu-se assim aumentar a amostra do valor a ser verificado, aumentando-se o número de pixéis a verificar, tendo-se chegado a um valor de referência de 30x30 *pixels* em que a deteção foi sempre realizada com bastante êxito para as duas tarefas desempenhadas.

No entanto, a deteção dos objetos continua a ser um dos pontos que desperta maior curiosidade em saber como seria o comportamento do rover robótico com o algoritmo criado caso este fosse simulado na realidade, verificando também o comportamento do *hardware* e da câmara existente. Na simulação experimental utilizou-se a resolução mais baixa por se considerar que resoluções maiores não traziam mais-valias à simulação, pelo contrário. Mas, e na realidade, seria de facto assim? E as taxas de sucesso seriam idênticas às obtidas no simulador?

É sabido que o sistema proposto continua a ter algumas lacunas na deteção de objetos, no sentido de que se for realizada uma análise de imagem apenas pela gama de cores e pela profundidade é algo ambíguo. Caso surja, por exemplo, uma pedra que tenha a mesma gama de cores e que esteja na altura certa, pois o terreno terá também várias irregularidades, poderá correr-se o risco de esta vir a ser validada pelo algoritmo e recolhida.

Mas é também sabido que a realização de um bom algoritmo para a deteção de objetos que utilize fatores de forma para extrair parâmetros da imagem, seja distâncias *euclideanas* ou *mahalanobis* ou até utilizando um sistema de inteligência artificial como *machine learning* é um trabalho de desenvolvimento exigente. Além disso, dentro do Grupo Operacional PrunusBot Assunção et al. [84] desenvolveu um algoritmo com taxas de sucesso bastante elevadas na deteção de pêssegos e que poderá vir a ser adicionado ao algoritmo desenvolvido neste trabalho.

Para a tarefa de recolha de objetos existe ainda um aspeto que importa referenciar, sendo ele a utilização de sensores que meçam a direção e a força a aplicar no objeto a recolher de forma a evitar feri-lo ou a deixar marcas da garra mecânica, como indicam Eizicovits & Berman [19]. No entanto, como no trabalho em causa os frutos a recolher já não terão valor comercial, pois são

frutos em fim de vida, não faria sentido considerar a inclusão destes sensores. Sendo que o único sensor que se inclui neste trabalho foi no eixo Z. Para que fosse verificado se durante a eventual recolha de um objeto existe, ou não, um aumento do peso no eixo Z no processo de subida da garra mecânica.

Em relação à tarefa de pulverização de infestantes, surgem também algumas dúvidas em saber se a metodologia proposta para a deteção de infestantes é a mais correta. No simulador, os resultados obtidos foram adequados, mas mais uma vez levanta-se a dúvida em saber se num ambiente real essa deteção iria funcionar igualmente bem. Nomeadamente, se a técnica utilizada em ignorar os 100 *pixels* em redor da zona que tem sido pulverizada é suficiente ou pelo contrário pode ser reduzida.

6.2. Sugestões de Trabalhos Futuros

Como referido anteriormente, um dos principais trabalhos para dar continuação a este projeto será aplicar os algoritmos aqui desenvolvidos no protótipo do rover robótico, fazendo todos os ajustes necessários e não esquecendo que o algoritmo utiliza variáveis que medem a localização em tempo real das juntas prismáticas, da garra mecânica, do sensor de força no eixo Z e até para a medição de velocidade das rodas. Assim, poderia ser necessário incluir alguns sensores no protótipo do rover robótico.

O mesmo acontece com o facto de saber a localização exata do rover robótico em tempo real, pois na simulação foram utilizados "*dummys*" para determinar a sua localização, a localização da base (local onde se inicia a simulação e onde são descarregados os objetos) e do ponto limite que o mesmo pode percorrer, calculando assim as distâncias máximas a percorrer, bem como a sua localização e velocidade. Numa aplicação real e para tirar o benefício destas funções, seria necessário dotar o rover robótico de um sistema GPS.

Com a replicação destes algoritmos à realidade poder-se-ia assim analisar as diferenças entre os resultados obtidos no simulador e os resultados reais, principalmente para os processos de deteção de objetos. No caso da recolha de objetos poder-se-ia também analisar o real comportamento da garra mecânica e dos diferentes prismas. Já na pulverização localizada poder-se-ia também verificar se a utilização da função da profundidade cumulativamente com a função da gama de cores num ambiente real traria ou não vantagens à deteção de infestantes.

Referir ainda que para o caso da pulverização poderia ser interessante adicionar um mecanismo que avaliasse o nível do depósito da solução para saber quando o rover robótico deveria ir à base. O mesmo poderia ser feito para o caso da recolha de objetos, adicionando um sensor de peso/força entre o rover robótico e a embalagem. Com esta informação e dependendo da tarefa a executar poderia ser posteriormente dada ordem para o rover robótico voltar à base ou lançar um alerta ao

utilizador. Estas questões seriam de implementação rápida uma vez que o algoritmo já possui uma subrotina para quando é atingido o número máximo de objetos a recolher, pelo que seria apenas necessário adicionar esta nova condição.

Para além dos pontos referidos existe ainda a questão de o algoritmo proposto não contemplar nenhum módulo que possa realizar a deteção de obstáculos. Seria interessante a criação de um algoritmo que realizasse estas funções e que fosse incluído nos algoritmos propostos.

Outro trabalho que poderia ser realizado reside na criação de um algoritmo em que fosse apenas necessário definir a área total do campo onde o próprio rover robótico se ia movendo, recolhendo objetos ou pulverizando e desviando-se dos obstáculos que iam surgindo, e ao mesmo tempo fizesse um mapeamento de toda a área de atuação. No fundo seria a criação de um algoritmo para a condução autónoma do rover robótico em que o utilizador não teria a preocupação de verificar constantemente o ponto de situação do trabalho em curso.

Por fim, poder-se-ia inserir neste algoritmo um sistema mais robusto para a deteção dos objetos a recolher ou das infestantes a pulverizar.

Referências Bibliográficas

[1] E. Morimoto, "Obstacle Avoidance System for Autonomous Transportation Vehicle Based on Image Processing," *Agric. Eng. Int. CIGR Ejournal*, vol. IV, pp. 1–11, 2002.

[2] J. Hassall and N. Australia, "Future Trends in Precision Agriculture A report for," no. 0906, 2010.

[3] S. Bachche, "Deliberation on design strategies of automatic harvesting systems: A survey," *Robotics*, vol. 4, no. 2, pp. 194–222, 2015, doi: 10.3390/robotics4020194.

[4] D. L. Shaner, "Lessons Learned From the History of Herbicide Resistance," *Weed Sci.*, vol. 62, no. 2, pp. 427–431, 2014, doi: 10.1614/ws-d-13-00109.1.

[5] W. McAllister, D. Osipychev, A. Davis, and G. Chowdhary, "Agbots: Weeding a field with a team of autonomous robots," *Comput. Electron. Agric.*, vol. 163, no. May, p. 104827, 2019, doi: 10.1016/j.compag.2019.05.036.

[6] M. Livingston, J. Fernandez-Cornejo, and G. B. Frisvold, "Economic Returns to Herbicide Resistance Management in the Short and Long Run: The Role of Neighbor Effects," *Weed Sci.*, vol. 64, no. SP1, pp. 595–608, 2016, doi: 10.1614/ws-d-15-00047.1.

[7] CODY MATTHEW EVANS, "Characterization of a novel five-way-resistant population of waterhemp (Amaranthus tuberculatus)," University of Illinois at Urbana- Champaign, 2016.

[8] José manuel muñoz lobo Viana, "Mobilização do Solo - Manual do formador."

[9] C. L. Mohler, J. C. Frisch, and J. M. Pleasant, "Evaluation of Mechanical Weed Management Programs for Corn (*Zea mays*)," *Weed Technol.*, vol. 11, no. 1, pp. 123–131, Mar. 1997, doi: 10.1017/S0890037X00041452.

[10] H. Mousazadeh, "A technical review on navigation systems of agricultural autonomous off-road vehicles," *J. Terramechanics*, vol. 50, no. 3, pp. 211–232, 2013, doi: 10.1016/j.jterra.2013.03.004.

[11] P. Gaspar, "PrunusBot - Sistema robótico aéreo autónomo de pulverização controlada e previsão de produção frutícola." https://inovacao.rederural.gov.pt/grupos-operacionais/13-projectos-groupos-operacionais/188-prunusbot-sistema-robotico-aereo-autonomo-de-pulverizacao-controlada-e-previsao-de-producao-fruticola (accessed Aug. 21, 2021).

[12] A. F. R. Veiros, "Sistema robótico terrestre para apoio a atividades de manutenção de solo em pomares de prunóideas," Universidade da Beira Interior, 2020.

[13] S. Y. Nof, *Handbook of Automation*, 1sd ed. Springer, 2009.

[14] A. Bechar, "Robotics in horticultural field production," *Stewart Postharvest Rev.*, vol. 6, no. 3, pp. 1–11, 2010, doi: 10.2212/spr.2010.3.11.

[15] Y. Nagasaka, N. Umeda, Y. Kanetai, K. Taniwaki, and Y. Sasaki, "Autonomous guidance for rice transplanting using global positioning and gyroscopes," *Comput. Electron. Agric.*, vol. 43, no. 3, pp. 223–234, 2004, doi: 10.1016/j.compag.2004.01.005.

[16] M. Iida *et al.*, "Advanced Harvesting System by using a Combine Robot," *IFAC Proc. Vol.*, vol. 46, no. 4, pp. 40–44, Jan. 2013, doi: 10.3182/20130327-3-JP-3017.00012.

[17] Z. Zhang, N. Noguchi, K. Ishii, L. Yang, and C. Zhang, "Development of a Robot Combine Harvester for Wheat and Paddy Harvesting," *IFAC Proc. Vol.*, vol. 46, no. 4, pp. 45–48, Jan. 2013, doi: 10.3182/20130327-3-JP-3017.00013.

[18] S. A. Hiremath, G. W. A. M. van der Heijden, F. K. van Evert, A. Stein, and C. J. F. Ter Braak, "Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter," *Comput. Electron. Agric.*, vol. 100, pp. 41–50, Jan. 2014, doi: 10.1016/J.COMPAG.2013.10.005.

[19] D. Eizicovits and S. Berman, "Efficient sensory-grounded grasp pose quality mapping for gripper design and online grasp planning," *Rob. Auton. Syst.*, vol. 62, no. 8, pp. 1208–1219, Aug. 2014, doi: 10.1016/J.ROBOT.2014.03.011.

[20] B. Zion, M. Mann, D. Levin, A. Shilo, D. Rubinstein, and I. Shmulevich, "Harvest-order planning for a multiarm robotic harvester," *Comput. Electron. Agric.*, vol. 103, pp. 75–81, Apr. 2014, doi: 10.1016/J.COMPAG.2014.02.008.

[21] A. Bechar and C. Vigneault, "Agricultural robots for field operations: Concepts and components," *Biosyst. Eng.*, vol. 149, pp. 94–111, 2016, doi: 10.1016/j.biosystemseng.2016.06.014.

[22] A. Steinfeld, "Interface lessons for fully and semi-autonomous mobile robots," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2004, no. 3, pp. 2752–2757, 2004, doi: 10.1109/robot.2004.1307477.

[23] A. Stentz, C. Dima, C. Wellington, H. Herman, and D. Stager, "A system for semi-autonomous tractor operations," *Auton. Robots*, vol. 13, no. 1, pp. 87–104, 2002, doi: 10.1023/A:1015634322857.

[24] K. Tervo and H. N. Koivo, "Adaptation of the Human-Machine Interface to the Human Skill and Dynamic Characteristics," *IFAC Proc. Vol.*, vol. 47, no. 3, pp. 3539–3544, Jan. 2014, doi: 10.3182/20140824-6-ZA-1003.02614.

[25] G. Rodriguez and C. R. Weisbin, "A new method to evaluate human-robot system performance," *Auton. Robots*, vol. 14, no. 2–3, pp. 165–178, 2003, doi: 10.1023/A:1022279618991.

[26] T. B. Sheridan and R. R. O. N. Daniel, *Telerobotics , Automation and Human Supervisory Control * Model Reference Adaptive Control : From Theory to Practice **. Cambridge, 1992.

[27] R. Ceres, J. L. Pons, A. R. Jiménez, J. M. Martín, and L. Calderón, "Design and implementation of an aided fruit-harvesting robot (Agribot)," *Ind. Rob.*, vol. 25, no. 5, pp. 337–346, 1998, doi: 10.1108/01439919810232440.

[28] A. Bechar and Y. Edan, "Human-robot collaboration for improved target recognition of agricultural robots," *Ind. Rob.*, vol. 30, no. 5, pp. 432–436, 2003, doi: 10.1108/01439910310492194.

[29] K. C. Ng and M. M. Trivedi, "A neuro-fuzzy controller for mobile robot navigation and multirobot convoying," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 28, no. 6, pp. 829–840, 1998, doi: 10.1109/3477.735392.

[30] L. Grimstad, C. D. Pham, H. T. Phan, and P. J. From, "On the design of a low-cost, light-weight, and highly versatile agricultural robot," *Proc. IEEE Work. Adv. Robot. its Soc. Impacts, ARSO*, vol. 2016-March, 2016, doi: 10.1109/ARSO.2015.7428210.

[31] L. Haibo, D. Shuliang, L. Zunmin, and Y. Chuijie, "Study and Experiment on a Wheat Precision Seeding Robot," 2015, doi: 10.1155/2015/696301.

[32] T. Bak and H. Jakobsen, "Agricultural Robotic Platform with Four Wheel Steering for Weed Detection," *Biosyst. Eng.*, vol. 87, no. 2, pp. 125–136, Feb. 2004, doi: 10.1016/J.BIOSYSTEMSENG.2003.10.009.

[33] T. Hague, J. A. Marchant, and N. D. Tillett, "Ground based sensing systems for autonomous agricultural vehicles," *Comput. Electron. Agric.*, vol. 25, no. 1–2, pp. 11–28, Jan. 2000, doi: 10.1016/S0168-1699(99)00053-8.

[34] Z. Li and X. Wang, "Application Research on High Resolution Radar Target Aggregation," *Int. J. Intell. Syst. Appl.*, vol. 2, no. 1, pp. 1–7, 2010, doi: 10.5815/ijisa.2010.01.01.

[35] N. Tremblay, Z. Wang, and Z. G. Cerovic, "Sensing crop nitrogen status with fluorescence indicators. A review," *Agron. Sustain. Dev.*, vol. 32, no. 2, pp. 451–464, 2012, doi: 10.1007/s13593-011-0041-1.

[36] D. C. Slaughter, D. K. Giles, and D. Downey, "Autonomous robotic weed control systems: A review," *Comput. Electron. Agric.*, vol. 61, no. 1, pp. 63–78, Apr. 2008, doi: 10.1016/J.COMPAG.2007.05.008.

[37] S. M. Pedersen, S. Fountas, H. Have, and B. S. Blackmore, "Agricultural robots - System analysis and economic feasibility," *Precis. Agric.*, vol. 7, no. 4, pp. 295–308, 2006, doi: 10.1007/s11119-006-9014-9.

[38] S. Nissimov, J. Goldberger, and V. Alchanatis, "Obstacle detection in a greenhouse environment using the Kinect sensor," *Comput. Electron. Agric.*, vol. 113, pp. 104–115, Apr. 2015, doi: 10.1016/J.COMPAG.2015.02.001.

[39] T. Burks *et al.*, "Engineering and horticultural aspects of robotic fruit harvesting: Opportunities and constraints," *Horttechnology*, vol. 15, no. 1, pp. 79–87, 2005, doi: 10.21273/horttech.15.1.0079.

[40] O. Yekutieli and F. Garbati Pegna, "Automatic Guidance of a Tractor in a Vineyard," no. 701, pp. 252–260, 2002, doi: 10.13031/2013.10014.

[41] D. D. Bochtis, C. G. C. Sørensen, and P. Busato, "Advances in agricultural machinery management: A review," *Biosyst. Eng.*, vol. 126, pp. 69–81, Oct. 2014, doi: 10.1016/J.BIOSYSTEMSENG.2014.07.012.

[42] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning - Using the voronoi diagram for a clearance-based shortest path," *IEEE Robot. Autom. Mag.*, vol. 15, no. 2, pp. 58–66, 2008, doi: 10.1109/MRA.2008.921540.

[43] J. R. Canning, D. B. Edwards, and M. J. Anderson, "Development of a fuzzy logic controller for autonomous forest path navigation," *Trans. Am. Soc. Agric. Eng.*, vol. 47, no. 1, pp. 301–310, 2004.

[44] X. J. Jing, "Behavior dynamics based motion planning of mobile robots in uncertain dynamic environments," *Rob. Auton. Syst.*, vol. 53, no. 2, pp. 99–123, Nov. 2005, doi: 10.1016/J.ROBOT.2005.09.001.

[45] N. Kondo and K. C. Ting, "Robotics for Plant Production," *Artif. Intell. Rev.*, vol. 12, no. 1–3, pp. 227–243, 1998, doi: 10.1007/978-94-011-5048-4_12.

[46] V. Bloch, A. Bechar, and A. Degani, "Development of an environment characterization methodology for optimal design of an agricultural robot," *Ind. Rob.*, vol. 44, no. 1, pp. 94–103, 2017, doi: 10.1108/IR-03-2016-0113.

[47] G. Belforte, R. Deboli, P. Gay, P. Piccarolo, and D. Ricauda Aimonino, "Robot Design and Testing for Greenhouse Applications," *Biosyst. Eng.*, vol. 95, no. 3, pp. 309–321, Nov. 2006, doi: 10.1016/J.BIOSYSTEMSENG.2006.07.004.

[48] N. Tavares, P. D. Gaspar, M. L., M. L. Aguiar, R. Mesquita, and M. P. Simões, "Robotic arm and gripper to pick fallen peaches in the orchards," 2022.

[49] C. Vigneault and A. Bechar, "Agricultural robots for field operations. Part 2: Operations and systems," *Biosyst. Eng.*, vol. 153, pp. 110–128, 2017, doi: 10.1016/j.biosystemseng.2016.11.004.

[50] A. Bechar, J. Meyer, and Y. Edan, "An objective function to evaluate performance of human-robot collaboration in target recognition tasks," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 39, no. 6, pp. 611–620, 2009, doi: 10.1109/TSMCC.2009.2020174.

[51] R. A. Brooks, "Intelligence without representation," *Artif. Intell.*, vol. 47, no. 1–3, pp. 139–159, Jan. 1991, doi: 10.1016/0004-3702(91)90053-M.

[52] B. G. Woolley, G. L. Peterson, and J. T. Kresge, "Real-time behavior-based robot control,"

Auton. Robots, vol. 30, no. 3, pp. 233–242, 2011, doi: 10.1007/s10514-010-9215-y.

[53] Y. Oren, A. Bechar, and Y. Edan, "Performance analysis of a human-Robot collaborative target recognition system," *Robotica*, vol. 30, no. 5, pp. 813–826, 2012, doi: 10.1017/S0263574711001020.

[54] I. Tkach, A. Bechar, and Y. Edan, "Switching between collaboration levels in a human-robot target recognition system," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 41, no. 6, pp. 955–967, 2011, doi: 10.1109/TSMCC.2011.2119480.

[55] A. Bechar, V. Bloch, R. Finkelshtain, S. Levi, A. Hoffman, and et al. (2014) Egozi, H., "Visual servoing methodology for selective tree pruning by human-robot collaborative system," 2014.

[56] R. L. Parish, "Current developments in seeders and transplanters for vegetable crops," *Horttechnology*, vol. 15, no. 2, pp. 346–351, 2005, doi: 10.21273/horttech.15.2.0346.

[57] H. Mao, L. Han, J. Hu, and F. Kumi, "Development of a pincette-type pick-up device for automatic transplanting of greenhouse seedlings," *Appl. Eng. Agric.*, vol. 30, no. 4, pp. 547–556, 2014, doi: 10.13031/aea.30.10550.

[58] Y. Nagasaka, K. Taniwaki, R. Otani, and K. Shigeta, "Automated rice transplanting operation using GPS and FOG sensors," *Japan Agric. Res. Q.*, vol. 35, no. 1, pp. 25–29, 2001, doi: 10.6090/jarq.35.25.

[59] Y. Nagasaka, H. Saito, K. Tamaki, M. Seki, K. Kobayashi, and K. Taniwaki, "An Autonomous Rice Transplanter Guided by Global Positioning System and Inertial Measurement Unit," *J. F. Robot.*, vol. 33, no. 1, pp. 1–17, 2009, doi: 10.1002/rob.

[60] Y. Nagasaka, H. Kitagawa, A. Mizushima, N. Noguchi, H. Saito, and K. Kobayashi, "Unmanned Rice-Transplanting Operation Using a GPS-Guided Rice Transplanter with Long Mat-Type Hydroponic Seedlings," *Agric. Eng. Int. CIGR Ejournal*, vol. IX, pp. 1–10, 2007.

[61] K. H. Ryu, G. Kim, and J. S. Han, "AE—Automation and Emerging Technologies: Development of a Robotic Transplanter for Bedding Plants," *J. Agric. Eng. Res.*, vol. 78, no. 2, pp. 141–146, Feb. 2001, doi: 10.1006/JAER.2000.0656.

[62] M. Lopes, P. D. Gaspar, and M. Simões, "Current Status and Future Trends of Mechanized Fruit Thinning Devices and Sensor Technology," *Int. J. Mech. Ind. Aerosp. Sci.*, vol. 12, no. 1, pp. 43–57, 2019, doi: 10.5281/zenodo.3607801.

[63] J. R. Morris, "Development and commercialization of a complete vineyard mechanization system," *Horttechnology*, vol. 17, no. 4, pp. 411–420, 2007, doi: 10.21273/horttech.17.4.411.

[64] J. A. Brandt and B. C. French, "Mechanical Harvesting and the California Tomato Industry: A Simulation Analysis," *Am. J. Agric. Econ.*, vol. 65, no. 2, pp. 265–272, 1983, doi: 10.2307/1240872.

[65] Y. Edan and G. E. Miles, "Design of an agricultural robot for harvesting melons," *Trans. Am. Soc. Agric. Eng.*, vol. 36, no. 2, pp. 593–603, 1993.

[66] Y. Edan, D. Rogozin, T. Flash, and G. E. Miles, "Robotic melon harvesting," *IEEE Trans. Robot. Autom.*, vol. 16, no. 6, pp. 831–835, 2000, doi: 10.1109/70.897793.

[67] M. Umeda, S. Kubota, and M. Lida, "Development of 'STORK', a watermelon-harvesting robot," pp. 143–147, 1999.

[68] M. Monta, N. Kondo, and K. C. Ting, "End-Effectors for Tomato Harvesting Robot," *Artif. Intell. Rev.*, vol. 12, no. 1–3, pp. 11–25, 1998, doi: 10.1023/a:1006595416751.

[69] M. Ceccarelli, G. Figliolini, E. Ottaviano, A. S. Mata, and E. J. Criado, "Designing a robotic gripper for harvesting horticulture products," *Robotica*, vol. 18, no. 1, pp. 105–111, 2000, doi: 10.1017/S026357479900226X.

[70] T. Kataoka, T. Hiroma, and Y. Ota, "Development of a harvesting hand for apples," *Adv. Robot.*, vol. 13, no. 3, pp. 293–294, 1998, doi: 10.1163/156855399X00612.

[71] A. Arefi, A. M. Motlagh, and R. F. Teimourlou, "A segmentation algorithm for the automatic recognition of tomato at harvest," *J. Food, Agric. Environ.*, vol. 8, no. 3-4 PART 2, pp. 815–819, 2010.

[72] C. W. Bac, J. Hemming, and E. J. Van Henten, "Robust pixel-based classification of obstacles for robotic harvesting of sweet-pepper," *Comput. Electron. Agric.*, vol. 96, pp. 148–162, Aug. 2013, doi: 10.1016/J.COMPAG.2013.05.004.

[73] E. Barnea, R. Mairon, and O. Ben-Shahar, "Colour-agnostic shape-based 3D fruit detection for crop harvesting robots," *Biosyst. Eng.*, vol. 146, pp. 57–70, Jun. 2016, doi: 10.1016/J.BIOSYSTEMSENG.2016.01.013.

[74] D. M. Bulanon, T. F. Burks, and V. Alchanatis, "Fruit visibility analysis for robotic citrus harvesting," *Trans. ASABE*, vol. 52, no. 1, pp. 277–283, 2009.

[75] F. Dong, W. Heinemann, and R. Kasper, "Development of a row guidance system for an autonomous robot for white asparagus harvesting," *Comput. Electron. Agric.*, vol. 79, no. 2, pp. 216–225, Nov. 2011, doi: 10.1016/J.COMPAG.2011.10.002.

[76] N. Irie, N. Taguchi, T. Horie, and T. Ishimatsu, "Asparagus harvesting robot coordinated with 3-D vision sensor," *Proc. IEEE Int. Conf. Ind. Technol.*, 2009, doi: 10.1109/ICIT.2009.4939556.

[77] W. Ji, D. Zhao, F. Cheng, B. Xu, Y. Zhang, and J. Wang, "Automatic recognition vision system guided for apple harvesting robot," *Comput. Electr. Eng.*, vol. 38, no. 5, pp. 1186–1195, Sep. 2012, doi: 10.1016/J.COMPELECENG.2011.11.005.

[78] S. S. Mehta and T. F. Burks, "Vision-based control of robotic manipulator for citrus harvesting," *Comput. Electron. Agric.*, vol. 102, pp. 146–158, Mar. 2014, doi: 10.1016/J.COMPAG.2014.01.003.

[79] S. S. Mehta, W. MacKunis, and T. F. Burks, "Robust visual servo control in the presence of fruit motion for robotic citrus harvesting," *Comput. Electron. Agric.*, vol. 123, pp. 362–375, Apr. 2016, doi: 10.1016/J.COMPAG.2016.03.007.

[80] T. T. Nguyen, K. Vandevoorde, N. Wouters, E. Kayacan, J. G. De Baerdemaeker, and W. Saeys, "Detection of red and bicoloured apples on tree with an RGB-D camera," *Biosyst. Eng.*, vol. 146, pp. 33–44, Jun. 2016, doi: 10.1016/J.BIOSYSTEMSENG.2016.01.007.

[81] T. Yuan, C.-G. Xu, Y.-X. Ren, Q.-C. Feng, Y.-Z. Tan, and W. Li, "Detecting the information of cucumber in greenhouse for picking based on NIR image," *Guang Pu Xue Yu Guang Pu Fen Xi/Spectroscopy Spectr. Anal.*, vol. 29, no. 8, pp. 2054–2058, 2009, doi: 10.3964/j.issn.1000-0593(2009)08-2054-04.

[82] S. Hayashi, K. Ganno, Y. Ishii, and I. Tanaka, "Robotic harvesting system for eggplants," *Japan Agric. Res. Q.*, vol. 36, no. 3, pp. 163–168, 2002, doi: 10.6090/jarq.36.163.

[83] Y. Cui, Y. Gejima, T. Kobayashi, K. Hiyoshi, and M. Nagata, "Study on Cartesian-Type Strawberry-Harvesting Robot," *Sens. Lett.*, vol. 11, pp. 6–7, 2013.

[84] E. Assunção, P. D. Gaspar, R. Mesquita, A. Veiros, and H. Proença, "Resultados preliminares de deteção de imagens de pêssegos aplicando o método Faster R-CNN," *Rev. da Assoc. Port. Hortic.*, pp. 1–7, 2020.

[85] J. Cunha, P. D. Gaspar, E. Assunção, and R. Mesquita, "Prediction of the Vigor and Health of Peach Tree Orchard," in *Computational Science and Its Applications -- ICCSA 2021*, 2021, pp. 541–551.

[86] C. Wouter Bac, J. van H. Eldert, H. Jochen, and E. Yael, "Harvesting Robots for High-value Crops: State-of-the-art Review and Challenges Ahead," *J. F. Robot.*, vol. 33, no. 1, pp. 1–17, 2014, doi: 10.1002/rob.

[87] E. C. Oerke and H. W. Dehne, "Safeguarding production—losses in major crops and the role of crop protection," *Crop Prot.*, vol. 23, no. 4, pp. 275–285, Apr. 2004, doi: 10.1016/J.CROPRO.2003.10.001.

[88] R. N. Jorgensen *et al.*, "Hortibot: An accessory kit transforming a slope mower into a robotic tool carrier for high-tech plant nursing - part I," *2006 ASABE Annu. Int. Meet.*,

vol. 0300, no. 06, 2006, doi: 10.13031/2013.20886.

[89] K. H. Choi, S. K. Han, S. H. Han, K. H. Park, K. S. Kim, and S. Kim, "Morphology-based guidance line extraction for an autonomous weeding robot in paddy fields," *Comput. Electron. Agric.*, vol. 113, pp. 266–274, Apr. 2015, doi: 10.1016/J.COMPAG.2015.02.014.

[90] M. Gonzalez-de-Soto, L. Emmi, M. Perez-Ruiz, J. Aguera, and P. Gonzalez-de-Santos, "Autonomous systems for precise spraying – Evaluation of a robotised patch sprayer," *Biosyst. Eng.*, vol. 146, pp. 165–182, Jun. 2016, doi: 10.1016/J.BIOSYSTEMSENG.2015.12.018.

[91] R. Oberti *et al.*, "Selective spraying of grapevines for disease control using a modular agricultural robot," *Biosyst. Eng.*, vol. 146, pp. 203–215, Jun. 2016, doi: 10.1016/J.BIOSYSTEMSENG.2015.12.004.

[92] X. E. Pantazi, D. Moshou, and C. Bravo, "Active learning system for weed species recognition based on hyperspectral sensing," *Biosyst. Eng.*, vol. 146, pp. 193–202, Jun. 2016, doi: 10.1016/J.BIOSYSTEMSENG.2016.01.014.

[93] A. J. Pérez, F. López, J. V. Benlloch, and S. Christensen, "Colour and shape analysis techniques for weed detection in cereal fields," *Comput. Electron. Agric.*, vol. 25, no. 3, pp. 197–212, Feb. 2000, doi: 10.1016/S0168-1699(99)00068-X.

[94] J. F. Thompson, J. V. Stafford, and P. C. H. Miller, "Potential for automatic weed detection and selective herbicide application," *Crop Prot.*, vol. 10, no. 4, pp. 254–259, Aug. 1991, doi: 10.1016/0261-2194(91)90002-9.

[95] J. Torres-Sospedra and P. Nebot, "Two-stage procedure based on smoothed ensembles of neural networks applied to weed detection in orange groves," *Biosyst. Eng.*, vol. 123, pp. 40–55, Jul. 2014, doi: 10.1016/J.BIOSYSTEMSENG.2014.05.005.

[96] D. C. Slaughter, D. K. Giles, and C. Tauzer, "Precision offset spray system for roadway shoulder weed control," *J. Transp. Eng.*, vol. 125, no. 4, pp. 364–371, 1999, doi: 10.1061/(ASCE)0733-947X(1999)125:4(364).

[97] K. Nishiwaki, K. Amaha, and R. Otani, "DEVELOPMENT OF NOZZLE POSITIONING SYSTEM FOR PRECISION SPRAYER," *EWSHM - 7th Eur. Work. Struct. Heal. Monit.*, no. December, pp. 24–31, 2004.

[98] B. Åstrand and A.-J. Baerveldt, "An agricultural mobile robot with vision-based perception for mechanical weed control," *Auton. Robots*, vol. 13, no. 1, pp. 21–35, 2002, doi: 10.1023/A:1015674004201.

[99] B. Åstrand and A. J. Baerveldt, "A vision based row-following system for agricultural field machinery," *Mechatronics*, vol. 15, no. 2, pp. 251–269, Mar. 2005, doi: 10.1016/J.MECHATRONICS.2004.05.005.

[100] N. Schor, S. Berman, A. Dombrovsky, Y. Elad, T. Ignat, and A. Bechar, "Development of a robotic detection system for greenhouse pepper plant diseases," *Precis. Agric.*, vol. 18, no. 3, pp. 394–409, 2017, doi: 10.1007/s11119-017-9503-z.

[101] E. Assuncao, C. Diniz, P. D. Gaspar, and H. Proenca, "Decision-making support system for fruit diseases classification using Deep Learning," *2020 Int. Conf. Decis. Aid Sci. Appl. DASA 2020*, pp. 652–656, 2020, doi: 10.1109/DASA51403.2020.9317219.

[102] Y. Chen, Y. Shan, L. Chen, K. Huang, and D. Cao, "Optimization of Pure Pursuit Controller based on PID Controller and Low-pass Filter," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-Novem, pp. 3294–3299, 2018, doi: 10.1109/ITSC.2018.8569416.

[103] M. Rasib *et al.*, "Are Self-Driving Vehicles Ready to Launch? An Insight into Steering Control in Autonomous Self-Driving Vehicles," *Math. Probl. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/6639169.

[104] N. Siddique and H. Adeli, "Nature Inspired Computing: An Overview and Some Future Directions," *Cognit. Comput.*, vol. 7, no. 6, pp. 706–714, 2015, doi: 10.1007/s12559-015-9370-8.

[105] X. S. Yang and Y. X. Zhao, *Nature-Inspired Computation in Navigation and Routing Problems: Methods, Methods and Applications*. 2020.

[106] X. S. Yang and X. S. He, *Mathematical Foundations of Nature-Inspired Methods*. 2019.

[107] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Nat. Comput.*, vol. 17, no. 3, pp. 585–609, 2018, doi: 10.1007/s11047-018-9685-y.

[108] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hierarchical Trajectory Planning of an Autonomous Car Based on the Integration of a Sampling and an Optimization Method," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 2, pp. 613–626, 2018, doi: 10.1109/TITS.2017.2756099.

[109] Y. Valle, V. Ganesh Kumar, and S. Mohagheghi, "Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems," *IEEE*, vol. 12, no. 2/2002, pp. 21–23, 2008, [Online]. Available: http://orgprints.org/1358/.

[110] O. K. Erol and I. Eksin, "A new optimization method: Big Bang–Big Crunch," *Adv. Eng. Softw.*, vol. 37, no. 2, pp. 106–111, Feb. 2006, doi: 10.1016/J.ADVENGSOFT.2005.04.005.

[111] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation*, vol. 76, no. 2, 2001.

[112] M. Ahangaran and P. Ramesani, "Harmony search algorithm: strengths and weaknesses," *J. Comput. Eng. Inf. Technol.*, vol. 2, no. 1, 2013.

[113] W. Sun and X. Chang, "An improved harmony search algorithm for power distribution network planning," *J. Electr. Comput. Eng.*, vol. 2015, 2015, doi: 10.1155/2015/753712.

[114] M. A. K. Niloy *et al.*, "Critical Design and Control Issues of Indoor Autonomous Mobile Robots: A Review," *IEEE Access*, vol. 9, pp. 35338–35370, 2021, doi: 10.1109/ACCESS.2021.3062557.

[115] J. Liao, Z. Chen, and B. Yao, "Model-Based Coordinated Control of Four-Wheel Independently Driven Skid Steer Mobile Robot with Wheel-Ground Interaction and Wheel Dynamics," *IEEE Trans. Ind. Informatics*, vol. 15, no. 3, pp. 1742–1752, 2019, doi: 10.1109/TII.2018.2869573.

[116] J. Liao, Z. Chen, and B. Yao, "Performance-Oriented Coordinated Adaptive Robust Control for Four-Wheel Independently Driven Skid Steer Mobile Robot," *IEEE Access*, vol. 5, pp. 19048–19057, 2017, doi: 10.1109/ACCESS.2017.2754647.

[117] J. Yi, H. Wang, J. Zhang, D. Song, S. Jayasuriya, and J. Liu, "Kinematic modeling and analysis of skid-steered mobile robots with applications to low-cost inertial-measurement-unit-based motion estimation," *IEEE Trans. Robot.*, vol. 25, no. 5, pp. 1087–1097, 2009, doi: 10.1109/TRO.2009.2026506.

[118] V. Nazari and M. Naraghi, "A vision-based intelligent path following control of a four-wheel differentially driven skid steer mobile robot," *2008 10th Int. Conf. Control. Autom. Robot. Vision, ICARCV 2008*, no. December, pp. 378–383, 2008, doi: 10.1109/ICARCV.2008.4795549.

[119] J. Xu and W. R. Norris, "Hierarchical fuzzy control design and stability analysis for steering-control based vehicles," *ACM Int. Conf. Proceeding Ser.*, 2019, doi: 10.1145/3351917.3351961.

[120] M. B. Montaner and A. Ramirez-Serrano, "Fuzzy knowledge-based controller design for autonomous robot navigation," *Expert Syst. Appl.*, vol. 14, no. 1–2, pp. 179–186, Jan. 1998, doi: 10.1016/S0957-4174(97)00059-6.

[121] B. Zhao, H. Wang, Q. Li, J. Li, and Y. Zhao, "PID Trajectory Tracking Control of Autonomous Ground Vehicle Based on Genetic Algorithm," *Proc. 31st Chinese Control Decis. Conf. CCDC 2019*, pp. 3677–3682, 2019, doi: 10.1109/CCDC.2019.8832531.

[122] A. Ghorbani, S. Shiry, and A. Nodehi, "Using genetic algorithm for a mobile robot path planning," *Proc. - 2009 Int. Conf. Futur. Comput. Commun. ICFCC 2009*, pp. 164–166, 2009, doi: 10.1109/ICFCC.2009.28.

[123] Z. Jiang and B. Xiao, "LQR optimal control research for four-wheel steering forklift based-on state feedback," *J. Mech. Sci. Technol.*, vol. 32, no. 6, pp. 2789–2801, 2018, doi: 10.1007/s12206-018-0536-7.

[124] F. M. Barbosa, L. B. Marcos, M. M. da Silva, M. H. Terra, and V. Grassi, "Robust path-following control for articulated heavy-duty vehicles," *Control Eng. Pract.*, vol. 85, pp. 246–256, Apr. 2019, doi: 10.1016/J.CONENGPRAC.2019.01.017.

[125] P. Dai, J. Taghia, S. Lam, and J. Katupitiya, "Integration of sliding mode based steering control and PSO based drive force control for a 4WS4WD vehicle," *Auton. Robots*, vol. 42, no. 3, pp. 553–568, 2018, doi: 10.1007/s10514-017-9649-6.

[126] X. He, Y. Liu, C. Lv, X. Ji, and Y. Liu, "Emergency steering control of autonomous vehicle for collision avoidance and stabilisation," *Veh. Syst. Dyn.*, vol. 57, no. 8, pp. 1163–1187, 2019, doi: 10.1080/00423114.2018.1537494.

[127] J. Guo, Y. Luo, and K. Li, "ROBUST H∞ FAULT-TOLERANT LATERAL CONTROL OF FOUR-WHEEL-STEERING AUTONOMOUS VEHICLES," *Int. J. Autom. Tecnol.*, vol. 21, no. 4, pp. 293–300, 2019, doi: 10.1007/s12239.

[128] M. A. Khan, M. F. Aftab, E. Ahmed, and I. Youn, "ROBUST DIFFERENTIAL STEERING CONTROL SYSTEM FOR AN INDEPENDENT FOUR WHEEL DRIVE ELECTRIC VEHICLE," *Int. J. Autom. Tecnol.*, vol. 20, no. 2, pp. 293–300, 2019, doi: 10.1007/s12239.

[129] I. Engedy and G. Horváth, "Artificial neural network based local motion planning of a wheeled mobile robot," *11th IEEE Int. Symp. Comput. Intell. Informatics, CINTI 2010 - Proc.*, pp. 213–218, 2010, doi: 10.1109/CINTI.2010.5672245.

[130] W. Yu, *Recent advances in intelligent control systems*. 2009.

[131] S. Ahmadzadeh and M. Ghanavati, "Navigation of mobile robot using the PSO particle swarm optimization," *Acad. Appl. Stud.*, 2012.

[132] H. Meira, "Desenvolvimento de Add-in de Ecodesign para Software de CAD Solidworks e Sincronização com Base de Dados," 2010.

[133] H. Secchi, "Robôs Móveis," 2008.

[134] S. D. N. Faria, "Sensor Fusion for Mobile Robot Localization using UWB and ArUco Markers," Faculdade de Engenharia da Universidade do Porto, 2021.

[135] G. E. Teixeira, "Mobile Robotics simulation for ROS based robots using Visual Components," Faculdade de Engenharia da Universidade do Porto, 2019.

[136] F. M. Teixeira, "Simulação de um Sistema Robótico de Co- transporte," Instituto Superior de Engenharia do Porto, 2020.

[137] L. Miranda, "Analysis and simulation of AGVS routing strategies using V-REP," 2017.

[138] ARGoS, "The ARGoS Website," 2021. https://www.argos-sim.info/authors.php (accessed Oct. 18, 2021).

[139] R. R. Shamshiri *et al.*, "Simulation software and virtual environments for acceleration of agricultural robotics: Features highlights and performance comparison," *Int. J. Agric. Biol. Eng.*, vol. 11, no. 4, pp. 12–20, 2018, doi: 10.25165/j.ijabe.20181104.4032.

[140] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, *Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators*, vol. 10965 LNAI. Springer International Publishing, 2018.

[141] S. D. Teixeira, "Simulação e Melhoramento do PiTank com Sistema de Inteligência Artificial," Universidade do Porto, 2019.

[142] T. Ribeiro, "Deep Reinforcement Learning for Robot Navigation Systems," Universidade do Minho, 2019.

[143] L. A. Moura, "Análise Comparativa de Técnicas para Controle de um Manipulador Robótico Utilizando o Sensor Kinect," UNIVERSIDADE FEDERAL DE ITAJUBÁ, 2019.

[144] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots: a survey based

on user feedback," pp. 1–15, 2014, [Online]. Available: http://arxiv.org/abs/1402.7050.

[145] L. Nogueira, "Comparative Analysis Between Gazebo and V-REP Robotic Simulators," *Semin. Interno Cognição Artif. - SICA.*, pp. 1678–1683, 2014, doi: 10.1109/ICMREE.2011.5930657.

[146] Coppelia Robotics Ltd, "Robot simulator CoppeliaSim." https://www.coppeliarobotics.com/ (accessed Oct. 21, 2021).

[147] Coppelia Robotics Ltd, "CoppeliaSim User Manual," 2021. https://www.coppeliarobotics.com/helpFiles/.

[148] C. R. Ltd, "Forum - Unanswered topics." https://forum.coppeliarobotics.com/search.php?search_id=unanswered (accessed Oct. 21, 2021).

[149] K. Fue, W. Porter, E. Barnes, and G. Rains, "An Extensive Review of Mobile Agricultural Robotics for Field Operations: Focus on Cotton Harvesting," *AgriEngineering*, vol. 2, no. 1, pp. 150–174, 2020, doi: 10.3390/agriengineering2010010.

[150] K. G. Fue, G. C. Rains, and W. M. Porter, "REAL-TIME 3-D MEASUREMENT OF COTTON BOLL POSITIONS USING MACHINE VISION UNDER FIELD CONDITIONS," 2018, no. 2007, pp. 43–54.

[151] T. Botterill *et al.*, "A Robot System for Pruning Grape Vines," *J. F. Robot.*, vol. 33, no. 1, pp. 1–17, 2017, doi: 10.1002/rob.

[152] M. P. Simões *et al.*, *+Pessego. Guia Prático da Produção*, vol. I. 2016.

[153] M. P. A. F. Simões, "A fertilização azotada em pessegueiros : influência no estado de nutrição, produção e susceptibilidade a Phomopsis amygdali," *A Fertil. azotada em pessegueiros influência no estado Nutr. produção e susceptibilidade a Phomopsis amygdali*, pp. 1–296, 2009, Accessed: Oct. 12, 2021. [Online]. Available: https://repositorio.ipcb.pt/handle/10400.11/155.

[154] Leopoldo Armesto, "DYOR," 2021. http://dyor.roboticafacil.es/en/ (accessed Oct. 27, 2021).

[155] I. Universitária, "O que são padrões de cores RGB e CMYK? – Imprensa Universitária," *Universidade Federal do Ceará*. https://imprensa.ufc.br/pt/duvidas-frequentes/padrao-de-cor-rgb-e-cmyk/ (accessed Oct. 27, 2021).

[156] U. NEHMZOW, *Robot Behaviour: Design, Description, Analysis and Modelling*. Springer, 2009.

ANEXOS

Este capítulo serve apenas de apoio, para apresentar as características técnicas do sensor de visão utilizado e ainda para a apresentação de forma integral dos dois algoritmos que foram criados para a realização das tarefas de recolha de objetos e de controlo de infestantes.

Anexo 1 – Ficha Técnica do Sensor de Visão Utilizado

Raspberry Pi Camera v2

***Part number:** RPI 8MP CAMERA BOARD*

- 8 megapixel camera capable of taking photographs of 3280 x 2464 pixels
- Capture video at 1080p30, 720p60 and 640x480p90 resolutions
- All software is supported within the latest version of Raspbian Operating System

The Camera v2 is the new official camera board released by the Raspberry Pi foundation.

The Raspberry Pi Camera Module v2 is a high quality 8 megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. It's capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p60/90 video. It attaches to Pi by way of one of the small sockets on the board upper surface and uses the dedicated CSi interface, designed especially for interfacing to cameras.

- 8 megapixel native resolution sensor-capable of 3280 x 2464 pixel static images
- Supports 1080p30, 720p60 and 640x480p90 video
- Camera is supported in the latest version of Raspbian, Raspberry Pi's preferred operating system

The board itself is tiny, at around 25mm x 23mm x 9mm. It also weighs just over 3g, making it perfect for mobile or other applications where size and weight are important. It connects to Raspberry Pi by way of a short ribbon cable.
The high quality Sony IMX219 image sensor itself has a native resolution of 8 megapixel, and has a fixed focus lens on-board. In terms of still images, the camera is capable of 3280 x 2464 pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video.

Applications

- CCTV security camera
- motion detection
- time lapse photography

Anexo 2 – Algoritmo para Caso de Estudo 1

```
function sysCall_init()

    WheelFL=sim.getObjectHandle('UJuntaFL')    --Junta da roda da frente lado esquerdo
    WheelFR=sim.getObjectHandle('UJuntaFR')    --Junta da roda da frente lado direito
    WheelRL=sim.getObjectHandle('UJuntaRL')    --Junta da roda da traseira lado esquerdo
    WheelRR=sim.getObjectHandle('UJuntaRR')    --Junta da roda da traseira lado direito
    tv=sim.getObjectHandle('Vision')           --Sensor de visao (Camara Video)

    eixo1=sim.getObjectHandle('Prisma1')    --Junta prismatica do eixo n.?1
    eixo2=sim.getObjectHandle('Prisma2')    --Junta prismatica do eixo n.?2
    eixo3=sim.getObjectHandle('Prisma3')    --Junta prismatica do eixo n.?3


    junta1=sim.getObjectHandle('1junta')     --Junta n. 1 da pinca 1
    junta2=sim.getObjectHandle('2junta')     --Junta n. 1 da pinca 2
    junta3=sim.getObjectHandle('3junta')     --Junta n. 1 da pinca 3
    junta4=sim.getObjectHandle('4junta')     --Junta n. 1 da pinca 4

    junta11=sim.getObjectHandle('11junta')   --Junta n. 2 da pinca 1
    junta21=sim.getObjectHandle('21junta')   --Junta n. 2 da pinca 2
    junta31=sim.getObjectHandle('31junta')   --Junta n. 2 da pinca 3
    junta41=sim.getObjectHandle('41junta')   --Junta n. 2 da pinca 4

    rota=sim.getObjectHandle('Roda')     --Junta principal da garra mecanica

    robo=sim.getObjectHandle('Robo')     --Dummy sobre localizacao do Robo
    stop=sim.getObjectHandle('Stop')     --Dummy sobre distannia maxima a percorrer
    base=sim.getObjectHandle('Base')     --Dummy sobre localizacao da Base

    estrutura=sim.getObjectHandle('EstruturaRobo')  --Estrutura total do robo

    esfera = {}
    for i=0,8,1 do                       --Declaracao de objetos a capturar
    esfera[i]=sim.getObjectHandle('Sphere'..i)
    sim.setShapeMass(esfera[i],0.17)
    end

    sensforc=sim.getObjectHandle('Force')

    cp={0}
    dados ={}
    dad ={}
    id=0
    fruta=0
    ppl=0
    loc=0
    turbo=0
    Pbase={0}
    Probo={0}
    localizacao=0
    zona=1
```

```
    anda=0
    counter=0
    erro=0
    sentido=1
    cu=0
    --Criacao da janela de informacao ao utilizador
    ui=simUI.create('<ui enabled="true" modal="false" title="TELECOMANDO" closeable="true" layout="vbox" placement="relative" position="20,20">' ..
    '<label text="Num. Frutas ate ir a base: " id="1"/>'..
    '<label text="Distancia da base: " id="2"/>'..
    '<label text="Velocidade: " id="3"/>'..
    '<button text="Ir a Base" on-click="DevolveBase" checkable="true" id="4"/>'..
    '</ui>')
    x=260
end
```

```
function sysCall_actuation()

    --Parametros a ler na janela de Informacao ao Utilizador (UI)
    simUI.setLabelText(ui,1,string.format("Num. Frutas ate ir a base: %.2f",-1*(fruta-5)))
    simUI.setLabelText(ui,2,string.format("Distancia da base [m]: %.2f",(Probo[1]-Pbase[1])))
    simUI.setLabelText(ui,3,string.format("Velocidade[m/s]: %.2f",turbo))
    velo1,velo2=sim.getVelocity(estrutura)

    -- variavel que mede a velocidade do robo para apresentar na janela UI
    if velo1[1] <0 then
        turbo = -1*velo1 [1]
    else turbo = velo1 [1]
    end


    Posicao1= sim.getJointPosition (eixo1)  -- variavel que mede a posicao do prisma 1
    Posicao2= sim.getJointPosition (eixo2)  -- variavel que mede a posicao do prisma 2
    Posicao3= sim.getJointPosition (eixo3)  -- variavel que mede a posicao do prisma 3
    m1=sim.getJointPosition(junta1)

    -- variavel que mede a posicao da junta principal da garra mecanica
    Prota=sim.getJointPosition(rota)

    -- variavel que mede a velocidade da roda frente lado esquerdo
    velocidade=sim.getJointVelocity(WheelFL)

    Probo=sim.getObjectPosition(robo,-1)  -- variavel que mede a posicao do robo
    Pstop=sim.getObjectPosition(stop,-1)  -- Distancia maxima que o robo deve percorrer
    Pbase=sim.getObjectPosition(base,-1)  -- variavel que mede a posicao da base


    -- variavel que mede o sensor de forca da garra mecanica
    resu,forca,torrqe=sim.readForceSensor(sensforc)

   if zona==1 then
    pr=0
    h1=0
    h2=0
    h3=0

        for u=20, 433, 7 do
        conf3=sim.getVisionSensorDepthBuffer(tv,x,u,30,30)
        conf4=sim.getVisionSensorImage(tv,x,u,30,30,0)
            for j=1, 900, 1 do
                pr=conf3[j]+pr
            end
            deep=pr/900

            for i=0, 2697, 3 do
                h1=conf4[i+1]+h1
                h2=conf4[i+2]+h2
```

```
                h3=conf4[i+3]+h3
            end
                r=h1/900
                g=h2/900
                b=h3/900

            if deep<= 0.77 and r>=0.65 and b<=0.2 and x >=260 then
                marcha=1
                zona=0
                xx=260
                break
            elseif deep<= 0.77 and r>=0.65 and b<=0.2 then
                marcha=1
                zona=0
                xx=250
                break
            end
            pr=0
            h1=0
            h2=0
            h3=0
        end

        if x>0 then
            x=x-20
        elseif x==0 then
            marcha=0
            zona=0
        end

    end


    --se houver objetos na zona A manda abrandar a velocidade do robo
    hl=0
    h1=0
    h2=0
    h3=0
    if marcha == 0  then
        for p=20, 433, 7 do
        conf1=sim.getVisionSensorDepthBuffer(tv,1,p,30,30)
        conf2=sim.getVisionSensorImage(tv,1,p,30,30,0)
            for j=1, 900, 1 do
                hl=conf1[j]+hl
            end
            deep=hl/900

            for ww=0, 2697, 3 do
                h1=conf2[ww+1]+h1
                h2=conf2[ww+2]+h2
                h3=conf2[ww+3]+h3
```

```
                h2=conf2[ww+2]+h2
                h3=conf2[ww+3]+h3
            end
            r1=h1/900
            g1=h2/900
            b1=h3/900

            if fruta>=5 or ppl>=1 then
                anda=0
                estado=20
                localizacao=Probo[1]
                marcha=2
            elseif deep<= 0.77 and r1>=0.65 and b1<=0.2 then
                anda=-2
                marcha=1
                xx=250
                break
            end

            if marcha==0 then
            anda=-3
            end

            if (Pstop[1]-Probo[1])<=0.1 then
            anda=0
            estado=30
            marcha=2
            break
            end
            h1=0
            h1=0
            h2=0
            h3=0
        end
    end


    --se houver objetos na zona B manda parar o robo  se o robo estiver cheio vai a base
    m=0
    c1=0
    c2=0
    c3=0
    if marcha == 1  then
        for p=20, 432, 5 do
        tab1=sim.getVisionSensorDepthBuffer(tv,xx,p,30,30)
        tab2=sim.getVisionSensorImage(tv,xx,p,30,30,0)
            for j=1, 900, 1 do
                m=tab1[j]+m
            end
            deep=m/900

```

```
208     if marcha == 1  then
209         for p=20, 432, 5 do
210         tab1=sim.getVisionSensorDepthBuffer(tv,xx,p,30,30)
211         tab2=sim.getVisionSensorImage(tv,xx,p,30,30,0)
212             for j=1, 900, 1 do
213                 m=tab1[j]+m
214             end
215             deep=m/900
216 
217             for ww=0, 2697, 3 do
218                 c1=tab2[ww+1]+c1
219                 c2=tab2[ww+2]+c2
220                 c3=tab2[ww+3]+c3
221             end
222             r=c1/900
223             g=c2/900
224             b=c3/900
225 
226             if fruta>=5 or ppl>=1 then
227             anda=0
228             estado=20
229             marcha=2
230             localizacao=Probo[1]
231 
232             elseif deep<= 0.77 and r>=0.65 and b<=0.2 then
233                 estado=1
234                 anda=0
235                 marcha=2
236                 break
237             end
238 
239             if marcha==1 then
240             anda=-2
241             end
242 
243             if (Pstop[1]-Probo[1])<=0.1 then
244             anda=0
245             estado=30
246             marcha=2
247             break
248             end
249 
250             c1=0
251             c2=0
252             c3=0
253             m=0
254 
255         end
256     end
257 
258 
```

```
    -- Este "if" deteta a zona com menor profundidade e alinha a garra com a localizacao do objeto a apanhar
    if estado==1 then

        m=0
        for p=20, 432, 1 do
        tab=sim.getVisionSensorDepthBuffer(tv,280,p,30,30)

            for j=1, 900, 1 do
                m=tab[j]+m
            end
            dep=m/900
            cp[p]= dep
            m=0
        end

    counter=counter+1

        if counter >4 then
        linha=cp[20]
        for k=21, 432, 1 do
            if cp [k]<linha then
                linha=cp [k]
                id=k
            end
        end

            if id<62 then
                id7=id-15
            else
            id7=id
            end
            loc= ((id7-15)*0.58)/447
            if id>360 then
                loc=loc+0.037
            end

            sim.setJointTargetPosition(eixo2,loc)
            sim.setJointTargetPosition(eixo1,0.055)
            sim.setJointTargetPosition(junta11,-0.55)
            sim.setJointTargetPosition(junta21,-0.55)
            sim.setJointTargetPosition(junta31,-0.55)
            sim.setJointTargetPosition(junta41,-0.75)
            estado=2
        end
    end

```

```
    -- Este "if" manda abrir as pincas da garra e baixar a garra
    if Posicao1 >= 0.053 and Posicao2 >= loc-0.015 and estado==2 then
        sim.setJointTargetPosition(junta1,-0.3)
        sim.setJointTargetPosition(junta2,-0.3)
        sim.setJointTargetPosition(junta3,-0.3)
        sim.setJointTargetPosition(junta4,-0.3)

        sim.setJointTargetPosition(eixo1,0.115)
        estado=3
    end


    if m1<= -0.26 and estado==3 then   -- Baixar mais a garra
        sim.setJointTargetPosition(eixo1,0.15)
        estado=4
    end

    -- manda fechar as pincas de modo a centrar bem o objeto com a garra antes de o apanhar
    if Posicao1 >= 0.143 and estado==4 then
        sim.setJointTargetPosition(junta1,-0.2)
        sim.setJointTargetPosition(junta2,-0.2)
        sim.setJointTargetPosition(junta3,-0.2)
        sim.setJointTargetPosition(junta4,-0.2)

        sim.setJointTargetPosition(junta11,-0.1)
        sim.setJointTargetPosition(junta21,-0.1)
        sim.setJointTargetPosition(junta31,-0.1)
        sim.setJointTargetPosition(junta41,-0.1)
        estado=5
    end

    -- baixar ainda mais a garra e manda apanhar o objeto
    if estado==5 and m1 >= -0.23 and Posicao1 >=0.147 then
        sim.setJointTargetPosition(eixo1,0.172)
        estado=6
    end

    if  Posicao1 >0.166 and estado==6 then  -- levanta a garra
        sim.setJointTargetPosition(junta1,0.05)
        sim.setJointTargetPosition(junta2,0.05)
        sim.setJointTargetPosition(junta3,0.05)
        sim.setJointTargetPosition(junta4,0.05)

        sim.setJointTargetPosition(junta11,0)
        sim.setJointTargetPosition(junta21,0)
        sim.setJointTargetPosition(junta31,0)
        sim.setJointTargetPosition(junta41,0)
        estado=7
  end
```

```
355     if  m1 >-0.04 and estado==7 then
356         sim.setJointTargetPosition(eixo1,0)
357         estado=8
358     end
359 
360     -- levanta a garra ? posicao inicial e roda a 90 graus
361     if estado==8 and Posicao1 <= 0.022 and forca[3]<-1.8 then
362         sim.setJointTargetPosition(eixo2,0.38)
363         sim.setJointTargetPosition(rota,-1.8)
364         estado=9
365     elseif estado==8 and forca[3]<0 and forca[3]>-1.4 and erro==3 then
366         if sentido==1 then
367             sentido=2
368             erro=2
369             pop=0
370             kr=0
371             kg=0
372             kb=0
373             estado=100
374         elseif sentido==2 then
375             erro=0
376             anda=-1
377             estado=100
378             zona=1
379             x=200
380         end
381     elseif estado==8 and forca[3]<0 and forca[3]>-1.4 then
382         zona=1
383         x=280
384         estado=100
385         erro=1+erro
386         counter=0
387         sentido=1
388     end
389 
```

```
390     ------------------------------------------------------------------------------------------------------------
391     -- Este passo apenas acontece caso o sensor de forca nao detete um aumento de peso ao levantar a garra mecanica
392     -- pois caso isso nao aconteca ? iniciado um processo de verifica??o se existe mais algum objeto nessa zona antes de avancar
393     if erro==2  then
394     sim.setJointTargetPosition(eixo1,0)
395     zona=0
396
397         if id> 385 then
398             sentido=2
399         elseif  sentido==1 then
400             pixeli=id+15
401             pixelf=433
402             incr=1
403             pixelincr=pixeli+20
404             pixelincr1=pixelincr+1
405         end
406
407         if sentido==2 and id < 80then
408         erro=0
409         anda=-1
410         sentido=1
411         zona=1
412         x=200
413         elseif sentido==2 then
414             pixeli=0
415             pixelf=id-35
416             incr=1
417             pixelincr=pixeli+19
418             pixelincr1=pixelincr+1
419         end
420
421         hk=0
422         h13=0
423         h23=0
424         h33=0
425         pop={}
426         kr={}
427         kg={}
428         kb={}
429
430         if Posicao1<0.033 then
431         for p=pixeli, pixelf, incr do
432             conf13=sim.getVisionSensorDepthBuffer(tv,260,p,30,30)
433             conf23=sim.getVisionSensorImage(tv,260,p,30,30,0)
434                 for j=1, 900, 1 do
435                     hk=conf13[j]+hk
436                 end
437                 deepk=hk/900
438                 pop[p]=deepk
439
```

```
439
440                 for ww=0, 2697, 3 do
441                     h13=conf23[ww+1]+h13
442                     h23=conf23[ww+2]+h23
443                     h33=conf23[ww+3]+h33
444                 end
445                 r13=h13/900
446                 g13=h23/900
447                 b13=h33/900
448
449                 kr[p]=r13
450                 kg[p]=g13
451                 kb[p]=b13
452
453                 hk=0
454                 h13=0
455                 h23=0
456                 h33=0
457         end
458
459         lnha=pop[pixelincr]
460         lnha1=kr[pixelincr]
461         lnha2=kb[pixelincr]
462         for o= pixelincr1, pixelf, incr do
463             if pop[o]<lnha then
464                 lnha= pop[o]
465                 di=o
466             end
467             if kr[o]>lnha1 then
468                 lnha1= kr[o]
469                 di2=o
470             end
471             if kb[o]<lnha2 then
472                 lnha2= kb[o]
473                 di3=o
474             end
475         end
476
477         if di==nil then
478         di=pixelincr
479         end
480
481         if lnha<= 0.77 and lnha1>=0.65 and lnha2<=0.2 then
482             if di<62 then
483                 di=di-15
484             end
485             loc= ((di-15)*0.58)/447
486             if di>360 then
487                 loc=loc+0.037
488             end
489
```

```
            sim.setJointTargetPosition(eixo2,loc)
            sim.setJointTargetPosition(eixo1,0.055)
            sim.setJointTargetPosition(junta11,-0.5)
            sim.setJointTargetPosition(junta21,-0.5)
            sim.setJointTargetPosition(junta31,-0.5)
            sim.setJointTargetPosition(junta41,-0.5)
            estado=2
            erro=3
        elseif sentido==1 then
            sentido=2
        elseif sentido==2 then
        erro=0
        anda=-1
        sentido=1
        zona=1
        x=200
        end
        end
    end
-------------------------------------------------------------------------------
    -- manda a garra para a frente para descarregar objeto
    if estado==9 and Prota <= -1.74  then
        sim.setJointTargetPosition(eixo3,0.16)
        estado=10
    end

    -- manda baixar a junta rotativa da garra e abrir pincas para largar objeto
    if estado==10 and Posicao3 > 0.13  then
        sim.setJointTargetPosition(rota,-1)
        sim.setJointTargetPosition(junta1,-1)
        sim.setJointTargetPosition(junta2,-1)
        sim.setJointTargetPosition(junta3,-0.5)
        sim.setJointTargetPosition(junta4,-0.5)

        sim.setJointTargetPosition(junta11,-0.6)
        sim.setJointTargetPosition(junta21,-0.6)
        sim.setJointTargetPosition(junta31,-0.6)
        sim.setJointTargetPosition(junta41,-0.6)
        estado=11
    end


    if estado==11 and m1 <= -0.8  then  -- encolhe pinca
        sim.setJointTargetPosition(junta1,0)
        sim.setJointTargetPosition(junta2,0)
        sim.setJointTargetPosition(junta3,0)
        sim.setJointTargetPosition(junta4,0)

        sim.setJointTargetPosition(junta11,0)
        sim.setJointTargetPosition(junta21,0)
```

```
539         sim.setJointTargetPosition(junta11,0)
540         sim.setJointTargetPosition(junta21,0)
541         sim.setJointTargetPosition(junta31,0)
542         sim.setJointTargetPosition(junta41,0)
543
544
545         sim.setJointTargetPosition(rota,-1.8)
546         estado=12
547         counter=0
548     end
549
550     -- recua a garra
551     if estado==12 and Prota <= -1.72  then
552         sim.setJointTargetPosition(eixo3,0)
553         estado=13
554     end
555
556     -- volta ao estado inicial
557     if estado==13 and Posicao3 <=0.025  then
558         sim.setJointTargetPosition(rota,0)
559         estado=14
560     end
561
562     --este passo serve apenas para ganhar tempo de a garra chegar ao estado inicial
563     if estado==14 and Prota >=-0.04  then
564         fruta=fruta+1
565         estado=100
566         if erro==3 then
567         erro=2
568         pop=0
569         kr=0
570         kg=0
571         kb=0
572         else
573         zona=1
574         x=340
575         end
576     end
577
578     --Se o cesto estiver cheio manda andar em sentido contrario, para a base
579     if estado ==20 and velocidade ==0 then
580     anda=4
581     estado=21
582     end
583
584     --eliminar objetos
585     if estado ==21 and (Pbase[1]-Probo[1])> -0.2 then
586     anda=0
587
588         for i=0,8,1 do
589         Pesfera=sim.getObjectPosition(esfera[i],-1)
```

```
        for i=0,8,1 do
        Pesfera=sim.getObjectPosition(esfera[i],-1)

            if Pesfera [3]>0.15 then
            sim.setShapeMass(esfera[i],400)
            sim.setModelProperty (esfera[i],sim.modelproperty_not_visible)
            sim.setModelProperty(esfera[i],sim.modelproperty_not_respondable)
            end
        end
    simUI.setButtonText(ui,4,"Ir a Base")

    fruta=0
    ppl=0
    estado=22
    end

    if estado ==22 and velocidade ==0 then
    anda=-4
    estado=23
    end


    if estado ==23 and (localizacao-Probo[1]) <=0.7 then
        anda=-3
        x=280
        localizacao=0
        zona=1
        estado=100
    end


    --Volta para a base e acaba
    if estado ==30 and velocidade ==0 then
    anda=4
    estado=31
    end

    if estado ==31 and (Probo[1]-Pbase[1])< 0.9 then
    anda= 0

    print ("*******Finish*******")
    simUI.setButtonText(ui,4,"*******Finish*******")
    estado=33
    end
        -- funcao que faz as rodas andar
        sim.setJointTargetVelocity(WheelFL,anda)
        sim.setJointTargetVelocity(WheelFR,anda)
        sim.setJointTargetVelocity(WheelRL,anda)
        sim.setJointTargetVelocity(WheelRR,anda)
end
```

```
639 function sysCall_sensing()
640     -- put your sensing code here
641 
642 end
643 
644 function sysCall_cleanup()
645     -- do some clean-up here
646 end
647 
648 
649 function DevolveBase(ui)
650 
651     ppl = ppl +1
652 
653     if ppl>=1 then
654         simUI.setButtonText(ui,4,"a ir..")
655         fruta=5
656     end
657 
658     print (ppl)
659 end
660 
```

Anexo 3 – Algoritmo para Caso de Estudo 2

```
function sysCall_init()
    -- do some initialization here

    WheelFL=sim.getObjectHandle('UJuntaFL')
    WheelFR=sim.getObjectHandle('UJuntaFR')
    WheelRL=sim.getObjectHandle('UJuntaRL')
    WheelRR=sim.getObjectHandle('UJuntaRR')
    tv=sim.getObjectHandle('Vision')

    eixo1=sim.getObjectHandle('Prisma1')
    eixo2=sim.getObjectHandle('Prisma2')


    robo=sim.getObjectHandle('Robo')    --Dummy sobre localizacao do Robo
    stop=sim.getObjectHandle('Stop')    --Dummy sobre distannia maxima a percorrer
    base=sim.getObjectHandle('Base')    --Dummy sobre localizacao da Base

    estrutura=sim.getObjectHandle('EstruturaRobo')  --Estrutura total do robo


    cp={0}
    dados ={}
    dad ={}
    id=0
    fruta=0
    ppl=0
    loc=0
    turbo=0
    Pbase={0}
    Probo={0}
    localizacao=0
    zona=1
    anda=0
    counter=0
    erro=0
    sentido=1
    cu=0
    ejeta=0

    --Criacao da janela de informacao ao utilizador
    ui=simUI.create('<ui enabled="true" modal="false" title="TELECOMANDO" closeable="true" layout="vbox" placement="relative" position="20,20">' ..
    '<label text="Distancia da base: " id="2"/>'..
    '<label text="Velocidade: " id="3"/>'..
    '<button text="Ir a Base" on-click="bevolveBase" checkable="true" id="4"/>'..
    '</ui>')

    x=260
end
```

```
function sysCall_actuation()
    --Parametros a ler na janela de Informacao ao Utilizador (UI)
    simUI.setLabelText(ui,2,string.format("Distancia da base [m]: %.2f",(Probo[1]-Pbase[1])))
    simUI.setLabelText(ui,3,string.format("Velocidade[m/s]: %.2f",turbo))

    velo1,velo2=sim.getVelocity(estrutura)

    Probo=sim.getObjectPosition(robo,-1)  -- variavel que mede a posicao do robo
    Pstop=sim.getObjectPosition(stop,-1)  -- Distancia maxima que o robo deve percorrer
    Pbase=sim.getObjectPosition(base,-1)  -- variavel que mede a posicao da base

    if velo1[1] <0 then  -- variavel que mede a velocidade do robo para apresentar na janela UI
        turbo = -1*velo1 [1]
    else turbo = velo1 [1]
    end

    Posicao1= sim.getJointPosition (eixo1)  -- variavel que mede a posicao do prisma 1
    Posicao2= sim.getJointPosition (eixo2)  -- variavel que mede a posicao do prisma 2

    -- variavel que mede a velocidade da roda frente lado esquerdo
    velocidade=sim.getJointVelocity(WheelFL)

    -- Verifica presença de ervas
   if zona==1 then
    h1=0
    h2=0
    h3=0

        for u=20, 433, 7 do
        conf4=sim.getVisionSensorImage(tv,x,u,30,30,0)

            for i=0, 2697, 3 do
                h1=conf4[i+1]+h1
                h2=conf4[i+2]+h2
                h3=conf4[i+3]+h3
            end
                r=h1/900
                g=h2/900
                b=h3/900

            if r<=0.24 and g>=0.32 and b<=0.33 and x >=240 then
                marcha=1
                zona=0
                xx=x
                break
            elseif  r<=0.24 and g>=0.32 and b<=0.33 then
                marcha=1
                zona=0
                xx=250
                break
            end
```

```
            pr=0
            h1=0
            h2=0
            h3=0
        end

        if x>0 then
            x=x-20
        elseif x==0 then
            marcha=0
            zona=0
        end

    end


    --se houver ervas na zona A manda abrandar a velocidade da plataforma
    h1=0
    h2=0
    h3=0
    if marcha == 0  then
        for p=20, 433, 7 do
        conf2=sim.getVisionSensorImage(tv,1,p,30,30,0)

            for ww=0, 2697, 3 do
                h1=conf2[ww+1]+h1
                h2=conf2[ww+2]+h2
                h3=conf2[ww+3]+h3
            end
            r1=h1/900
            g1=h2/900
            b1=h3/900

            if ppl>=1 then
                anda=0
                estado=20
                localizacao=Probo[1]
                marcha=2
            elseif r1<=0.24 and g1>=0.32 and b1<=0.33 then
                anda=-2
                marcha=1
                xx=250
                break
            end

            if marcha==0 then
            anda=-3
            end

            if (Pstop[1]-Probo[1])<=0.1 then
            anda=0
```

```
150             anda=0
151             estado=30
152             marcha=2
153             break
154             end
155             h1=0
156             h1=0
157             h2=0
158             h3=0
159         end
160     end
161 
162 
163     --se houver ervas na zona B manda parar a plataforma  se estiver cheia vai a base
164     c1=0
165     c2=0
166     c3=0
167     if marcha == 1  then
168         for p=20, 432, 5 do
169         tab2=sim.getVisionSensorImage(tv,xx,p,30,30,0)
170 
171             for ww=0, 2697, 3 do
172                 c1=tab2[ww+1]+c1
173                 c2=tab2[ww+2]+c2
174                 c3=tab2[ww+3]+c3
175             end
176             r=c1/900
177             g=c2/900
178             b=c3/900
179 
180             if ppl>=1 then
181             anda=0
182             estado=20
183             marcha=2
184             localizacao=Probo[1]
185 
186             elseif r<=0.24 and g>=0.32 and b<=0.33 then
187                 estado=1
188                 anda=0
189                 marcha=2
190                 break
191             end
192 
193             if marcha==1 then
194             anda=-2
195             end
196 
197             if (Pstop[1]-Probo[1])<=0.1 then
198             anda=0
199             estado=30
200             marcha=2
```

```
            marcha=2
            break
            end

            c1=0
            c2=0
            c3=0
            m=0

        end
    end


    -- Este "if" deteta a zona com a gama de cores mais favoravel
    -- e alinha a garra com a localizacao do objeto
    if estado==1 then

        h13=0
        h23=0
        h33=0
        kr={}
        kg={}
        kb={}
        for p=5,432, 1 do
            conf23=sim.getVisionSensorImage(tv,xx,p,30,30,0)

                for ww=0, 2697, 3 do
                    h13=conf23[ww+1]+h13
                    h23=conf23[ww+2]+h23
                    h33=conf23[ww+3]+h33
                end
                r13=h13/900
                g13=h23/900
                b13=h33/900

                kr[p]=r13
                kg[p]=g13
                kb[p]=b13

                h13=0
                h23=0
                h33=0
        end

            lnha1=kg[20]
            for o= 21, 432, 1 do
                if kr[o]<0.27 and kg[o]>lnha1 then
                    lnha1= kr[o]
                    id=o
                end
            end
```

```
251 
252             if id<62 then
253                 id7=id-15
254             else
255             id7=id
256             end
257             loc= ((id7-15)*0.58)/447
258             if id>360 then
259                 loc=loc+0.037
260             end
261 
262                 sim.setJointTargetPosition(eixo2,loc)
263                 sim.setJointTargetPosition(eixo1,0.16)
264                 estado=2
265     end
266 
267 
268     -- Este "if" deteta quando a pulverização estiver efetuada
269    if estado==2 and Posicao1 >0.1596  then
270         sim.setJointTargetPosition(eixo1,0)
271         estado=3
272         ejeta=ejeta+1
273     end
274 
275 
276     if estado==3  then
277 
278         if id> 333 then
279             sentido=2
280         elseif  sentido==1 then
281             pixeli=id+100
282             pixelf=433
283             incr=1
284             pixelincr=pixeli+20
285             pixelincr1=pixelincr+1
286         end
287 
288         if sentido==2 and id < 100 then
289         ejeta=0
290         estado=4
291         anda=-2
292         sentido=1
293         elseif sentido==2 then
294             pixeli=0
295             if id >399 then
296             pixelf=id-170
297             else pixelf=id-100
298             end
299             incr=1
300             pixelincr=pixeli+19
301             pixelincr1=pixelincr+1
```

```
301             pixelincr1=pixelincr+1
302             print (pixelf)
303         end
304         hk=0
305         h13=0
306         h23=0
307         h33=0
308         kr={}
309         kg={}
310         kb={}
311 
312         if Posicao1<0.033 then
313         for p=pixeli, pixelf, incr do
314             conf23=sim.getVisionSensorImage(tv,xx,p,30,30,0)
315 
316                 for ww=0, 2697, 3 do
317                     h13=conf23[ww+1]+h13
318                     h23=conf23[ww+2]+h23
319                     h33=conf23[ww+3]+h33
320                 end
321                 r13=h13/900
322                 g13=h23/900
323                 b13=h33/900
324 
325                 kr[p]=r13
326                 kg[p]=g13
327                 kb[p]=b13
328 
329                 hk=0
330                 h13=0
331                 h23=0
332                 h33=0
333         end
334 
335         lnha2=kg[pixelincr]
336         for o= pixelincr1, pixelf, incr do
337             if kr[o]<0.31 and kg[o]> lnha2 then
338                 lnha2= kg[o]
339                 lnha3= kr[o]
340                 lnha4= kb[o]
341                 di=o
342             end
343         end
344 
345 
346         if di==nil then
347         di=pixelincr
348         end
349 
350         if lnha2==nil then
351         lnha2=0.32
```

```
350         if lnha2==nil then
351         lnha2=0.32
352         end
353 
354         if lnha3==nil then
355         lnha2=0.24
356         end
357 
358 
359         if lnha2>= 0.32 and lnha3<=0.24 and lnha4<=0.33 and ejeta<2 then
360             if di<62 then
361                 di=di-15
362             end
363             loc= ((di-15)*0.58)/447
364             if di>360 then
365                 loc=loc+0.037
366             end
367 
368             sim.setJointTargetPosition(eixo2,loc)
369             sim.setJointTargetPosition(eixo1,0.16)
370             estado=2
371         elseif sentido==1 then
372             sentido=2
373             ejeta=1
374         elseif sentido==2 then
375         ejeta=0
376         estado=4
377         anda=-2
378         sentido=1
379         end
380         end
381     end
382 
383     --Se no local em que a plataforma esta parada nao tiver mais objetos
384     -- o algoritmo volta ao inicio
385     if estado == 4 and velocidade<-1.99 then
386         estado=100
387         zona=1
388         x=100
389     end
390 
391     --Se receber ordem para ir a base
392     if estado ==20 and velocidade ==0 then
393     anda=4
394     estado=21
395     end
396 
397     --Regressa à base e volta a partir
398     if estado ==21 and (Pbase[1]-Probo[1])> -0.2 then
399     anda=0
400     simUI.setButtonText(ui,4,"Ir a Base")
```

```
401         ppl=0
402         estado=22
403         end
404 
405         if estado ==22 and velocidade ==0 then
406         anda=-4
407         estado=23
408         end
409 
410         if estado ==23 and (localizacao-Probo[1]) <=0.7 then
411             anda=-3
412             x=280
413             localizacao=0
414             zona=1
415             estado=100
416         end
417 
418         --Volta para a base e acaba
419         if estado ==30 and velocidade ==0 then
420         anda=4
421         estado=31
422         end
423 
424         if estado ==31 and (Probo[1]-Pbase[1])< 0.9 then
425         anda= 0
426         print ("*******Finish*******")
427         estado=33
428         end
429 
430         -- funcao que faz as rodas andar
431         sim.setJointTargetVelocity(WheelFL,anda)
432         sim.setJointTargetVelocity(WheelFR,anda)
433         sim.setJointTargetVelocity(WheelRL,anda)
434         sim.setJointTargetVelocity(WheelRR,anda)
435         end
436 function sysCall_sensing()
437         -- put your sensing code here
438 end
439 function sysCall_cleanup()
440         -- do some clean-up here
441 end
442 function DevolveBase(ui)
443 
444         ppl = ppl +1
445 
446         if ppl>=1 then
447             simUI.setButtonText(ui,4,"a ir..")
448         end
449 
450         print (ppl)
451 end
```