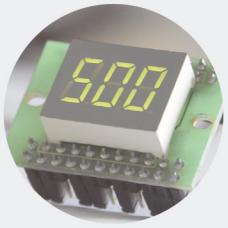
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






EFACEC ? RGV SIMULATOR - SIMULADOR DE VEÍCULOS GUIADOS POR CARRIL

EDGAR ANDRÉ AGUIAR MAGALHÃES
novembro de 2016

POLITÉCNICO
DO PORTO

EFACEC – RGV SIMULATOR

SIMULADOR DE VEÍCULOS GUIADOS POR CARRIL

Edgar André Aguiar Magalhães

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

2016

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Edgar André Aguiar Magalhães, Nº 1100413, 1100413@isep.ipp.pt

Orientação científica: Eng. Lino Figueiredo, lbf@isep.ipp.pt

Empresa: Eng. Vítor Vaz, vvaz@efacec.com – EFACEC, Handling Solutions, S.A.

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Automação e Sistemas

2016

Agradecimentos

Gostava de aproveitar esta secção para agradecer às diversas pessoas que permitiram que este projeto se tornasse realidade.

Ao meu chefe, Eng. Vítor Vaz, e ao meu orientador Eng. Lino Figueiredo, pela oportunidade, interesse e apoio prestado na conceção deste projeto.

O maior agradecimento vai para os meus Pais. Eles que me apoiaram, sempre me deram o carinho, atenção e me guiaram durante toda a minha vida. Não há palavras para descrever e agradecer as oportunidades que me deram e o amor que demonstraram e demonstram. A sensação de ter sempre alguém a apoiar-nos nos bons e maus momentos faz-nos sentir seguros de que nada é impossível, basta existir vontade.

Deixar, de igual forma, uma palavra de agradecimento à minha namorada. Agradeço-te por tudo, Sara, que fazes de mim quem sou e que me motivas para ser mais e melhor a cada dia que passa.

Aos meus amigos, aos de sempre, e aos novos que, desde a minha entrada no ISEP, estiveram dispostos a ajudar, confraternizar, partilhar conhecimentos e, acima de tudo, fazer *masterplans*, rir e brincar.

À EFACEC em geral e em particular, à Efacec, Handling Solutions, SA, que me proporcionou, desde Setembro de 2013, a melhor experiência profissional que poderia ter adquirido. Aos colegas de trabalho, pela paciência e pela ajuda na criação do projeto.

Resumo

Os constantes avanços tecnológicos levam à construção de sistemas mais complexos e mais flexíveis. O mundo da Automação Industrial não é exceção e os sistemas desenvolvidos neste âmbito são complexos, ao ponto de se tornarem difíceis de prever o funcionamento, ainda na fase de conceção.

A presente Tese propõe a criação de um simulador que preveja o funcionamento dos circuitos de Veículos Guiados por Carril (RGV) e que auxilie o especialista na configuração e correção do programa *Master Controller*, o "cérebro" dos RGVs. O simulador permitirá determinar o grau de *performance* dos sistemas de RGVs em fase de projeto, e ajudará a criar sistemas mais robustos por permitir o teste intensivo antes da instalação. A criação do simulador tornará a permanência de pessoal especializado no local da instalação mais curta e possibilitará a libertação desses recursos de forma a alocá-los a novos projetos, reduzindo assim os custos da empresa.

Para desenvolver o simulador, utilizou-se o Blender, uma plataforma de criação de aplicações 3D que tem *Application Programming Interface* (API) em Python. Foram desenvolvidos modelos que representam os equipamentos reais e módulos que auxiliam a troca de informação entre computadores e Controladores Lógicos Programáveis (PLC) da marca Siemens, que leem desenhos AutoCAD e os traduzem em mundos simulados.

O simulador emula o comportamento dos equipamentos e comunica com o *Master Controller*¸ de forma a testar as suas decisões sobre o circuito. A solução desenvolvida foi e é usada no dia-a-dia pelos especialistas destes equipamentos e ajuda-os de diversas formas, desde a configuração à correção de possíveis erros no programa.

Palavras-Chave

Automação Industrial, RGV, Simulador 3D, Blender, Python, PLC, Siemens

Abstract

Advances in technology often lead to the construction of more complex and flexible systems. The world of Industrial Automation is no exception and systems developed in this area are complex, to the point of being difficult to predict the operation, when in design phase.

This thesis proposes the creation of a simulator that predicts the reaction of Rail Guided Vehicles (RGV) circuits and to assist the automation engineer in the configuration and debug of the Master Controller program, the RGV's "brain". The simulator will determine the level of performance of RGVs systems still in design phase, and will help to create more robust and flexible systems as it allows intensive testing before installation. This simulator will allow to reduce the permanence on-site of specialized staff, freeing the experts to be allocated to new projects, thus reducing the company's costs.

To develop the simulator, it was used Blender, a platform for creating 3D applications that have Python as Application Programming Interface (API). 3D models were developed to represent the actual equipment and Python modules to allow the exchange of information between computers and Siemens Programmable Logic Controllers (PLC) and to read and translate AutoCAD drawings into simulated worlds.

The simulator emulates the behavior of equipment and communicates with the Master Controller in order to test its decisions on the circuit. The developed solution was and is used in day-to-day basis by RGV’s experts helps them in many ways, from parameters setting to correction of possible errors in the program.

Keywords

Industrial Automation, RGV, 3D Simulator, Blender, Python, PLC, Siemens

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AGV – *Autonomous Guided Vehicles*

AP – *Access Point*

API – *Application Programming Interface*

AS/RS – *Automated Storage and Retrieval System*

BGE – *Blender Game Engine*

CGI – *Computer Generated Imagery*

CPU – *Communications Processor Unit*

CSV – *Comma Separated Values*

DB – Base de Dados

DEE – Departamento de Engenharia Eletrotécnica

DLL – *Dynamic-Link Library*

DWG – *Drawing*

EHS – EFACEC, Handling Solutions, S.A.

ERP – *Enterprise Resorce Planning*

FIFO – *First In First Out*

GIL – *Global Interpreter Lock*

HUD – *Head-Up Display*

ID – Investigação e Desenvolvimento

LED – *Light Emitting Diode*

MEEC – Mestrado em Engenharia Eletrotécnica e Computadores

MPI – *Multi-Point Interface*

PLC – Controlador Lógico Programável

SCADA – *Supervisory Control and Data Acquisition*

UI – *User Interface*

WCS – *Warehouse Control System*

WIK – *Work In Progress*

WMS – *Warehouse Management System*

XML – *Extensible Markup Language*

1. INTRODUÇÃO

No âmbito da unidade curricular de Dissertação, do 2º ano do Mestrado em Engenharia Eletrotécnica e Computadores (MEEC), do Departamento de Engenharia Eletrotécnica (DEE), do Instituto Superior de Engenharia do Porto, surgiu a possibilidade de realizar um projeto na empresa EFACEC, Handling Solutions, S.A. (EHS). O projeto consiste em criar um simulador que permita testar e reduzir o tempo de desenvolvimento de programas para circuitos de veículos guiados por carril (*Rail Guided Vehicle* – RGV).

A EFACEC é uma empresa multinacional criada em 1948. Hoje possui mais de 4600 colaboradores e está presente em mais de 65 países, nos cinco continentes. A EFACEC produz soluções nas áreas de energia, logística e ambiente, entre outras.

A unidade de Logística – Handling Solutions, na qual foi realizado o projeto, produz soluções para armazenamento automático de *stock*, centros de distribuição automáticos, sistemas de preparação de encomendas e sistemas de transporte de bagagens. O portfólio de produtos disponibilizados pela unidade inclui transelevadores, miniloads, transportadores, *Autonomous Guided Vehicles* (AGV), RGVs, máquinas de triagem e sistema de gestão de armazéns.

O objetivo do presente projeto é o de criar uma plataforma de simulação, onde se possa testar possíveis cenários que acontecerão na realidade de forma a reduzir o tempo de permanência em obra de pessoal especializado, aumentando a eficácia e eficiência de projetos com RGVs. Desta forma, antes do material estar instalado, já se tem a certeza que o programa não contém erros a nível lógico.

1.1. CONTEXTUALIZAÇÃO

Nos dias de Hoje o tempo e a eficácia são cada vez mais valorizados. Todos os segundos contam quando se pretende chegar com maior rapidez da produção de um qualquer produto ao consumidor final. Com o aparecimento de soluções automatizadas tenta-se chegar a uma máquina perfeita, alheia a erros humanos. A evolução dos equipamentos e tecnologias e a sua integração na indústria tem incentivado os profissionais das áreas da Engenharia Eletrotécnica e Informática, bem como outros especialistas em Automação, a desenvolverem soluções mais complexas, flexíveis, eficazes e eficientes.

O desenvolvimento de uma ferramenta flexível é moroso e complexo pois o utilizador tem de prever e responder de forma controlada a todos os cenários possíveis. Isto afeta o tempo de desenvolvimento e acompanhamento de tal forma ao ponto de a solução deixar de ser rentável.

O desafio proposto é desenvolver uma ferramenta que permita, remota e antecipadamente, prever o comportamento de um sistema complexo de gestão de veículos guiados por carril, permitindo assim ao especialista perceber as correções e melhorias que tem de desenvolver antes da instalação física. Para isso será necessário recorrer a ferramentas de criação de aplicações 3D, que são utilizadas normalmente para a criação de jogos, modelação 3D, filmes de animação e *Computer Generated Imagery* (CGI) para filmes de produção, ferramentas essas que partilham um dos principais objetivos deste projeto: reduzir o tempo e custo da solução.

1.2. OBJETIVOS

O objetivo geral desta Tese consiste em desenvolver uma plataforma de teste e simulação para um programa complexo de gestão de RGVs. A simulação terá que emular o comportamento dos veículos quando recebem ordens para a execução das suas tarefas.

Com todos os dados disponíveis, terá também que os analisar e apresentar ao utilizador para que este se aperceba das correções e melhorias que o programa necessita.

O processo de criação do simulador pode ser dividido em:

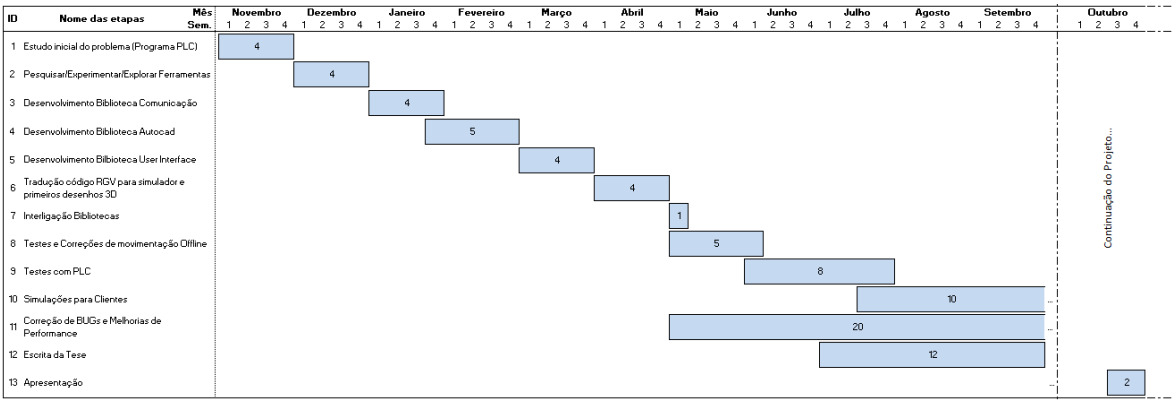
- Estudar o programa do Controlador Lógico Programável (PLC) a testar;
- Pesquisar, experimentar e explorar ferramentas para a criação do projeto;
- Desenvolver uma biblioteca de comunicação com um PLC real;
- Desenvolver uma biblioteca para extrair informação de um *layout* de AUTOCAD;
- Desenvolver uma biblioteca para fazer a interface com o utilizador;
- Traduzir o programa PLC de um RGV para o simulador;
- Desenvolver o núcleo de simulação, interligando todas as bibliotecas;

1.3. CALENDARIZAÇÃO

Com a duração de um ano, o projeto apresentado nesta tese passou por diversas fases. A procura de uma ferramenta grátis e *open-source*, capaz de responder ao problema proposto, a aprendizagem da ferramenta de desenvolvimento e de uma nova linguagem de programação, a delineação do problema de forma a compreender a proporção do projeto e organizar as fases seguintes, o desenvolvimento dos vários módulos e os constantes testes para verificar e autenticar o funcionamento do simulador.

De notar que o simulador é um *Work In Progress* (WIK) com desenvolvimentos diários que, a partir do momento em que se começou a escrever a presente Tese, não foram incluídos.

Tabela 1 Calendarização do projeto (disponível em Anexo A)



1.4. ORGANIZAÇÃO DO DOCUMENTO

Este documento está dividido em 5 capítulos.

No segundo capítulo é apresentado o Estado da Arte, que inclui a apresentação do mundo da logística e de armazéns automáticos, procura de simuladores semelhantes ao proposto, ferramentas para desenvolvimento e respetivas linguagens de programação admitidas.

No terceiro capítulo, intitulado de Sistema, é apresentada a respetiva delineação e arquitetura, esquemas de ligações entre dispositivos e alguns dos algoritmos a utilizar.

No quarto capítulo é apresentado o desenvolvimento do sistema e resultados obtidos.

No capítulo final são apresentadas as conclusões, trabalhos correntes e futuros.

Segundo o princípio de confidencialidade presente no Código de Conduta da EHS, na presente tese está proibida a colocação de qualquer informação que ponha em causa o a empresa. Por essa razão, as funções criadas são expostas através de diagramas e fluxogramas.

2. ESTADO DA ARTE

Neste capítulo, é feita uma introdução ao mundo da logística, nomeadamente aos armazéns automáticos, desenvolvidos pela EHS. É apresentado um simulador existente para a ajuda a programação de PLCs. De seguida são apresentadas algumas plataformas de desenvolvimento de aplicações 3D e as respetivas linguagens de programação admitidas nos seus projetos. Por fim é apresentada uma comparação das plataformas existentes e das diferentes linguagens de programação.

2.1. LOGÍSTICA

O termo Logística tem origem militar que, na sua perspetiva, aplica-se ao processo de movimentar, fornecer e manter forças militares no terreno. [4] Foi adotado pela indústria para descrever a gestão do fluxo de materiais numa organização, desde a matéria-prima a produtos acabados entregues ao consumidor final.

Susan Malyk, no seu artigo "*Customer Service in Supply Chain Management*" descreveu a logística como: "*having the right item in the right quantity at the right time at the right place for the right price in the right condition to the right customer*". [5] Isto indica que

todos os passos devem estar bem definidos: os serviços que se vão usar para obter os produtos para o cliente, o tempo que o produto vai estar parado em armazém, o tempo e custo do transporte, entre outros.

A logística é muito mais do que armazenamento automático mas, para o propósito desta Tese, os próximos subcapítulos serão dedicados a este tema.

2.2. ARMAZÉNS AUTOMÁTICOS

Em muitos casos, o custo relacionado com o inventário pode rivalizar com os custos de transporte e até tornar-se o maior custo logístico de uma empresa. Como é examinado de forma contínua, torna-se o segmento com maior oportunidade de melhoria. [1]

Um sistema de reaquisição e armazenamento automático, conhecido no mundo logístico por *Automated Storage and Retrieval System* (AS/RS), é composto por um conjunto de equipamentos computorizados que movimentam produtos, armazenando-os em estantes com uma ou mais alas. Estes equipamentos são responsáveis tanto pelo armazenamento como pela reaquisição de produtos para expedição.

As vantagens destes sistemas são inúmeras. [3] Providenciam aos utilizadores um aumento de controlo sobre o inventário, bem como uma melhor monitorização dos produtos em movimento. Estes sistemas costumam ser modulares, compostos por subsistemas que, além de redundantes, são facilmente substituídos em caso de avaria para minimizar o tempo de inatividade, aumentando assim o tempo de serviço do armazém. Outra vantagem é a redução de custos com pessoal, reduzindo os requisitos de trabalho forçado, aumentando assim a segurança no local de trabalho e removendo os trabalhadores de condições de trabalho precárias (como em ambientes de armazenamento de alimentos frios, normalmente a temperaturas negativas). Porém, a vantagem mais significativa em sistemas AS/RS é a possibilidade de reduzir em larga escala os custos de armazenamento de inventário, aumentando o aproveitamento de espaço disponível, criando uma maior densidade de armazenamento.

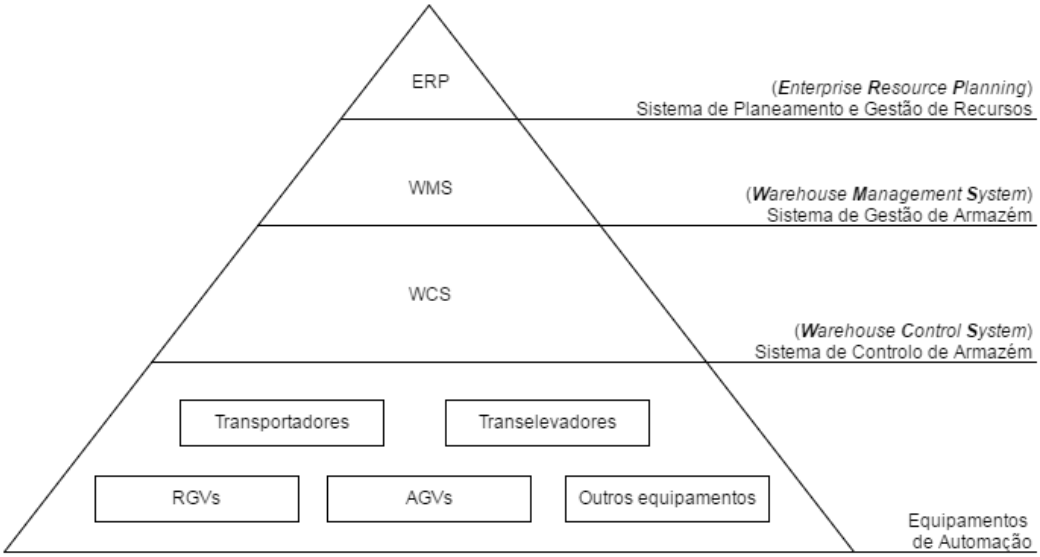


Figura 1 Pirâmide de Hardware/Software de um armazém/centro de distribuição

Dentro de um armazém automático existem vários níveis de *Hardware/Software* que acompanham, ajudam a controlar o fluxo de tarefas em execução e a planear as tarefas por executar. A Figura 1 proporciona uma visão geral dos sistemas normalmente utilizados para gestão de um armazém ou centro de distribuição. Como se pode observar, o esquema tem uma forma de pirâmide, o que indica que as ordens são geradas de cima para baixo. A figura será analisada nos subcapítulos abaixo.

2.2.1. WAREHOUSE MANAGEMENT SYSTEM (WMS)

Um *Warehouse Management System* (WMS) é um *software* responsável por apoiar um centro de armazenamento e distribuição. [2] Estes sistemas permitem um controlo centralizado de tarefas como a monitorização de inventário e localizações de *stock*. Os sistemas WMS podem ser aplicações independentes ou fazer parte de um sistema *Enterprise Resorce Planning* (ERP).

Estes usam bases de dados configuradas para apoiar operações de armazenamento, contendo detalhes como:

- Dados sobre cada unidade de armazenamento: peso, dimensões, códigos de barras dos produtos, entre outras;
- Localizações de armazenamento;
- Linhas de expedição;
- Indicadores de produtividade do pessoal.

Estas aplicações ajudam no planeamento e organização dos trabalhos no dia-a-dia, dando assim controlo ao utilizador de direcionar os meios para onde são necessários e dirigir tarefas que, de outro modo, seriam demasiado complexas de gerir. [2]

Instalações que possuem sistemas AS/RS costumam conter também um *Warehouse Control System* (WCS) que é integrado com o sistema WMS de forma a proporcionar ao utilizador uma visão abrangente do armazém.

2.2.2. WAREHOUSE CONTROL SYSTEM (WCS)

Um WCS é uma aplicação de *software* responsável por orquestrar a atividade e fluxos de produtos dentro de um armazém ou centro de distribuição. O WCS coordena o manuseamento de materiais nos subsistemas como transportadores, transelevadores, RGVs, AGVs, entre outros. Em cada ponto de decisão, o WCS é responsável por determinar o fluxo mais eficiente dando assim ordens aos subsistemas para atingir o resultado desejado.

Algumas implementações de WCS incluem também sistemas *Supervisory Control and Data Acquisition* (SCADA) que representam de uma forma gráfica e em tempo real o *layout* da instalação, facilitando assim a operação e monitorização por parte do utilizador. A Figura 2 representa um exemplo de um sistema SCADA.

Figura 2 Exemplo de sistema SCADA (*iSystemsNow*)

Nestes sistemas é possível ter uma visão global da instalação, monitorar e rastrear unidades de carga, verificar o estado e enviar comandos para cada equipamento. Isto facilita a deteção dos problemas e acelera a sua resolução.

2.2.3. EQUIPAMENTOS DE AUTOMAÇÃO

Um armazém automático consiste num conjunto de *software* e equipamentos automáticos que melhoram a eficiência das operações de logística. Estes sistemas de automação contam com uma grande variedade de equipamentos, dentro dos quais:

TRANSPORTADORES

- *Gabarit*: responsáveis por assegurar o bom acondicionamento da carga antes de entrar no armazém;
- Elevadores: para transportar cargas em vários níveis diferentes;
- Mesas rotativas: para alterar a orientação de movimento da carga;
- Gravíticos: para acumular cargas de expedição;
- *Buffer*: responsáveis por acumular cargas em fluxos assíncronos;

RGVS

Um veículo guiado por carril, que se movimenta num circuito fechado, com o formato de um ou mais transportadores, responsável por transportar uma ou várias cargas entre dois pontos com velocidade superior à de uma cadeia de transportadores em série.

ROBÔS INDUSTRIAIS

- Paletizadores: tem como função acomodar produtos numa palete, de forma a otimizar o espaço utilizado;
- Despaletizadores: tem como função retirar artigos de uma palete para seguirem para linhas de *Order Picking* ou armazenamento especial;
- *Order Picking*: tem como tarefa reunir um conjunto de artigos, numa quantidade específica antes da expedição de forma a satisfazer as necessidades do cliente.

EQUIPAMENTOS PARA ARMAZENAMENTO DE CARGAS EM ESTANTE:

- Transelevadores: uma máquina que se movimenta dentro de uma ou mais alas segundo os eixos X e Y para depositar ou reaver uma unidade de carga numa estante.
- AGVs: veículos autónomos que se movimentam no mesmo espaço que os trabalhadores e empilhadores manuais, transportando cargas para estantes em armazéns semiautomáticos.

Uma instalação deste tipo, devido à grande variedade de equipamentos, tem muitas variáveis associadas que podem alterar completamente o comportamento do sistema pelo que é extremamente difícil prever o funcionamento final, na fase de projeto. Por esta razão existem empresas que se dedicam a fazer simulações de instalações de logística como a SIMCORE, localizada em França, de forma a determinar a *performance* do sistema antes da instalação.

O problema destas simulações é que não incluem o modelo real da máquina pois não possuem o programa que está de facto a ser executado no controlador da máquina.

2.3. SIMULADORES

As plataformas de simulação providenciam uma forma económica de entender e avaliar a *performance* tanto de sistemas abstratos como de sistemas reais. Além de eficientes e fáceis de utilizar, os simuladores têm de ser capazes de acompanhar a crescente complexidade dos sistemas atuais. Estes devem também ser extensíveis e modulares, de forma a se adaptarem facilmente ao comportamento e *performance* de novos sistemas.

A ideia de desenvolver um simulador para testar programas PLC remotamente não é nova. Algumas empresas dedicam-se exclusivamente a esta tarefa, como é o caso do TrySIM, um *software* desenvolvido pela empresa Cephalos, na Alemanha. [6]

As vantagens do TrySIM identificam-se com as deste projeto pois trata-se de uma ferramenta que permite testar a lógica de um programa PLC em grande parte, antes do comissionamento, tornando-o muito mais curto, o que conduz a uma redução significativa dos custos de instalação.

O TrySIM, como qualquer simulador que tenha como objetivo testar um programa PLC, usa uma interface para ler/escrever de ou a partir de um PLC, como esquematizado na Figura 3.

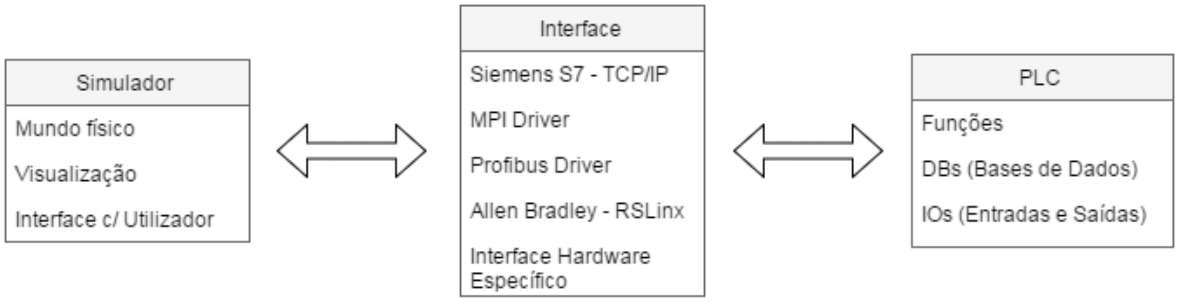


Figura 3 Interação Simulador com o PLC

Portanto, criar um simulador deste género tem como desafios desenvolver:

- Um mundo físico que se adapte ao projeto a simular;
- Uma visualização correta e cuidadosa do projeto;
- Uma interface com o utilizador que permita a interação deste no mundo simulado, simulando a resposta do sistema a situações reais;
- Desenvolver uma interface robusta que permita escrever/ler sinais com um PLC real.

Um projeto desta dimensão pode levar anos a desenvolver. Existem ferramentas disponíveis no mercado algumas até gratuitas, para facilitar a elaboração de cada desafio.

Um simulador pode ser comparado com um jogo de computador. Ambos simulam um mundo virtual, interagem com o utilizador e produzem resultados que podem ser analisados em tempo real ou armazenados para uma análise posterior. Ao contrário dos simuladores, para jogos de computador existe um vasto leque de ferramentas que ajudam na sua criação, normalmente denominadas por motores de jogo ou *Game Engines*. Ferramentas como o Blender, o Unity, o Unreal Engine, entre outras, são exemplos de plataformas que facilitam a criação tanto de protótipos, como de produtos finais, reduzindo assim o tempo de desenvolvimento dos mesmos.

Utilizar um motor de jogo tem inúmeras vantagens [7]:

- A maior parte do código de baixo-nível está desenvolvido e testado. O utilizador tem apenas de desenvolver o código específico relativo à sua aplicação;

- Tarefas como gestão de memória, carregamento de *assets*, iluminação, entre outros está desenvolvido. O utilizador só tem apenas de escrever algumas linhas de código (em algumas plataformas, é tudo feito de forma visual);
- Se o motor for multiplataforma, o utilizador tem de fazer pouco ou nada para transcrever o código para outra plataforma.

Ainda assim, nem tudo são vantagens. Utilizar um motor de jogo como uma "caixa-preta" tem desvantagens:

- Se tiver de alterar algum aspeto, o utilizador terá que se familiarizar com a base de código utilizada pelo motor;
- Se existir um erro no motor, a menos que seja *open-source*, não existe solução a não ser a de esperar por uma nova versão oficial com a respetiva correção;
- A plataforma não foi desenvolvida especificamente para a aplicação que o utilizador está a desenvolver, portanto pode não ser tão eficiente como desenvolver o código de raiz;
- Os motores de jogo normalmente não são grátis.

Embora a lista de desvantagens seja maior do que a de vantagens, a escolha entre criar uma plataforma de raiz ou utilizar uma existente recai sobre um simples fator: tempo de desenvolvimento. Desenvolver uma plataforma de raiz requer tempo, um fator que é muito valorizado pela indústria.

Quando o objetivo é otimizar o tempo de instalação de projetos, passar meses ou até anos a desenvolver uma plataforma de simulação não é opção.

2.4. MOTORES DE JOGO (*GAME ENGINES)*

Um motor de jogo é constituído por três sub-motores: lógico, físico e visual. O motor lógico é responsável pelo processamento da simulação: criação de objetos no mundo, cálculos de performance do sistema, realização da interface com o utilizador, entre outras funções. É também no motor lógico que vai assentar todo o código da aplicação.

O motor visual é o responsável por criar a imagem de acordo com posição da câmara atual de todos os objetos visíveis na janela de exibição bem como desenhar um *Head-Up Display* (HUD). Já o motor físico tem como função determinar a posição e velocidade

dos objetos segundo as leis da física – que podem ser completamente diferentes das do mundo real.

A Tabela 2 representa uma comparação de quatro fatores importantes aquando da escolha da plataforma de desenvolvimento para o projeto: plataformas, *application programming interface* (API), *open-source* e o preço de uma licença para uso comercial.

O Unity3D, desenvolvido pela *Unity Techonlogies*, é o motor com mais plataformas-alvo disponíveis. Este tenta cobrir um vasto leque de mercados, desde PCs, consolas, a plataformas *online* e até plataformas móveis. Permite criar aplicações em C# ou Javascript, tendo como inconveniente não ser *open source* e o valor de aquisição/aluguer.

Tabela 2 Comparação de Motores de Jogo

Motor		**Unity3D**	**Unreal Engine**	**Unigine**	**Blender**
Produtor		Unity	Epic Games	Unigine Corp	Blender Foundation
Plataformas	*Windows*	✓	✓	✓	✓
	Linux	✓	✓	✓	✓
	OS X	✓	✓	✓	✓
	Xbox 360	✓	✓	✗	✗
	Xbox One	✓	✓	✗	✗
	PlayStation 3	✗	✓	✓	✗
	PlayStation 4	✓	✓	✗	✗
	PlayStation Vita	✓	✓	✗	✗
	Adobe Flash Player	✗	✓	✗	✗
	Unity Web Player	✓	✗	✗	✗
	Wii	✓	✗	✗	✗
	Wii U	✓	✓	✗	✗
	Nintendo 3DS line	✓	✗	✗	✗
	Windows Phone 8	✓	✗	✓	✗
	Android	✓	✓	✓	✗
	BlackBerry 10	✓	✗	✗	✗
	iOS	✓	✓	✓	✗
	AppleTV	✓	✗	✗	✗
Open Source		✗	✓	✗	✓
API		C# Javascript	C++	C++ UnigineScript	Python (C++)
Preço de licença comercial		1500$ ou 75$/mês	19$/mês + 5%/venda	10 000$	Grátis, para qualquer utilização

O Unreal Engine, desenvolvido pela *Epic Games*, embora não cubra todas as plataformas do Unity3D, tem a vantagem de ser *open source* e ter um valor mensal mais baixo – que,

dependendo da aplicação, pode não compensar, visto que a *Epic Games* cobra 5% de cada venda do produto final.

Estes dois motores têm como alvo principal o desenvolvimento de jogos para um vasto número de plataformas. Já o Unigine, apesar de servir para o mesmo efeito e ter provas dadas nesse segmento, é mais orientado para simulação. [8] Possui uma linguagem de programação proprietária, UnigineScript e pode utilizar-se também C++. Tem o inconveniente de não ser *open source* e o elevado valor de aquisição de uma licença comercial.

O Blender, desenvolvido pela *Blender Foundation*, é um *software* gratuito e *open source* de criação 3D. Foi criado em 2002, com a intenção de criar uma ferramenta gratuita, para modelação, animação, simulação, renderização e composição. Em 2007 foi introduzido também o motor de jogo, o *Blender Game Engine* (BGE), que acabou por tirar partido de todo o trabalho feito nas outras áreas até à data. O Blender é hoje uma plataforma robusta para criação de simulações precisas e tem, a cada 3 meses, uma atualização oficial que corrige possíveis erros que possam surgir. Conta também com uma vasta comunidade *online* que se entreajuda para resolver e ajudar a que os seus projetos singrem. Todo o código é disponibilizado e pode ser alterado e compilado novamente para criar uma ferramenta completamente diferente. O entusiasmo da comunidade é tal que existem *websites* dedicados só à apresentação de novos projetos, como o *BlenderNation.com*, onde são organizadas conferências um pouco por todo o mundo para juntar os *developers* ou simples entusiastas para apresentarem os próprios projetos e iniciativas. A API do Blender usa como linguagem de programação Python mas, devido ao facto de ser *open-source*, há a possibilidade de integrar no código-fonte da própria aplicação, em C++.

Sendo este um projeto do departamento de Investigação e Desenvolvimento (ID) da EHS, o custo da plataforma é um fator determinante. Reduzindo o custo da ferramenta, liberta-se maior quantidade de recursos para alocar ao desenvolvimento.

Então, o ponto de partida para este projeto seria encontrar uma ferramenta que preenchesse os seguintes requisitos:

- Plataforma-alvo: Windows;

- Preço da licença comercial: O mais reduzido possível;
- Curva de aprendizagem reduzida;
- Adequado a implementar um protótipo num prazo reduzido.

Tendo em conta os requisitos do projeto, e dado que a venda a público não é, pelo menos à partida, um dos objetivos, as duas plataformas que restam são o Blender e o Unreal Engine. Além do custo da licença, a necessidade de criação de um protótipo num prazo reduzido, inclina a decisão para o Blender, visto que a sua API é desenvolvida em Python. O Python, apesar de ser uma linguagem interpretada (assunto discutido no próximo subcapítulo), conta com uma ampla comunidade *online* e um leque de bibliotecas já desenvolvidas que acelera o desenvolvimento de qualquer projeto.

2.5. LINGUAGENS DE PROGRAMAÇÃO

Existem muitas formas de descrever uma linguagem de programação: de alto ou baixo nível, multi-plataforma, entre outras. Mas uma forma diferenciadora que surge muitas vezes é: linguagem compilada ou interpretada. [9]

Quando um programador cria um programa, fá-lo em texto – o seu código-fonte tem depois de ser traduzido para linguagem-máquina para que o dispositivo o possa processar. O programador escreve as linhas de código de forma a ser lógico para um humano entender. Mas essa ordem de linhas pode não ser a mais lógica para uma máquina processar pelo que é necessário proceder à tradução. O código deve estar num formato que a *Central Processing Unit* (CPU) consiga ler, e os compiladores e interpretadores tratam disso mesmo.

As linguagens compiladas[1] são escritas num código que pode ser diretamente processado na CPU, isto porque o compilador já tratou o código fonte e o traduziu para código-máquina, mesmo antes de este ser processado. Este processo pode requerer muitas iterações antes de se transformar em código-máquina otimizado, mas o resultado é

[1] i.e. C, C++, C#, Objective-C, Pascal, entre outras.

sempre código pronto a ser executado de forma eficiente. [9] O processo de compilação, demonstrado na Figura 4, ocorre apenas uma vez.

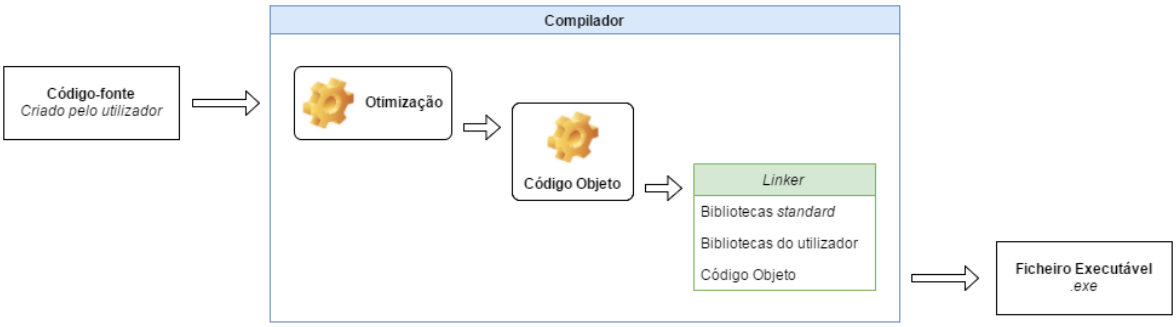


Figura 4 Processo de Compilação

O compilador otimiza o código fonte, criando um objeto intermédio, conhecido por Código Objeto, que depois é submetido a um *Linker* que interliga as bibliotecas *standard* da CPU e as bibliotecas do utilizador, resultando num ficheiro executável.

As linguagens interpretadas[1], por outro lado, são aquelas que não necessitam de estar traduzidas em código-máquina antes de serem executadas. A tradução ocorre ao mesmo tempo em que o programa está a ser executado. Muitas instruções destas linguagens são executadas diretamente, no entanto, quando é necessário executar algum código específico a tradução é feita em tempo real.

Tabela 3 Comparação de velocidade entre Python e C++ [11]

Linguagem	Tempo s (Algoritmo)	x Mais Lento
C++ (Otimizado)	1,124	-
C++ (Puro)	3,156	2,81
Python 3.5	19,004	16,91
Python 2.7	21,112	18,78

A maior vantagem de uma linguagem interpretada é a sua portabilidade [10], o que quer dizer que as aplicações criadas nesta linguagem podem ser executadas em diferentes plataformas. O facto de serem traduzidas em tempo-real significa que vão ser otimizadas para a plataforma na qual estão a ser executadas. Isso significa também que não existem

[1] i.e. JavaScript, Python, Ruby, entre outras.

passos intermédios, logo menos memória é necessária e o utilizador não tem de se preocupar com código específico de cada plataforma.

Um interpretador é um programa que converte código-fonte em código-máquina todas as vezes que é executado, uma linha de cada vez. O programa resultante demora mais tempo a ser executado do que um programa compilado, isto porque tem mais passos a acontecerem em tempo-real. Os programas compilados já passaram pelo passo de tradução antes de serem executados, pelo que são necessariamente mais rápidos. A Tabela 3 apresenta uma comparação entre o tempo levado a executar um determinado algoritmo em C++ e Python. Segundo os dados apresentados, o Python é em média quinze a dezasseis vezes mais lento que o C++. [11]

Tabela 4 Comparação entre compilador e interpretador

Aspeto	Compilador	Interpretador
Entrada	Carrega todo o programa.	Carrega apenas uma linha de código de cada vez.
Saída	Gera um objeto-código intermédio.	Não possui a definição de objeto-código.
Velocidade	Executa o programa mais rápido.	É mais lento a executar o programa.
Memória	Requer mais memória de forma a criar o objeto-código.	Requer menos memória, visto que não cria objeto-código.
Carga de Trabalho	Compila o programa uma única vez.	Tem de converter expressões de alto nível em expressões para CPU durante a execução.
Erros	Mostra os erros depois de verificar todo o programa.	Mostra erros apenas quando a linha de código é executada.

Os interpretadores são úteis para rever, executar e testar funcionalidades de uma aplicação durante a fase de desenvolvimento pois são capazes de executar código de alto-nível e gerar relatórios de erro detalhados. Também permitem aos programadores fazer pequenas mudanças no código, passo-a-passo que resulta num melhor controlo e confiança nas alterações.

A Tabela 4 mostra um resumo da comparação de alguns aspetos, entre os dois tipos de linguagens. Não existe melhor, mas sim duas opções disponíveis quando se está a iniciar um projeto. A escolha deve ser feita de forma a tirar o máximo partido da tecnologia que se utiliza.

Para este projeto, foi decidido utilizar o Blender, que tem API desenvolvida em Python, uma linguagem interpretada *open-source* que conta com uma extensa comunidade *online*

que produz e disponibiliza milhares de bibliotecas, contribuindo para o seu contínuo desenvolvimento e aperfeiçoamento.

3. SISTEMA

Neste capítulo é apresentado o sistema de RGVs, a arquitetura e modo de funcionamento. É também apresentada uma solução para o problema principal – criar um simulador para testar circuitos de RGVs – a arquitetura de *software* e respetivo diagrama de blocos, a arquitetura de *hardware* e possíveis configurações de ligação.

3.1. SISTEMA DE RGVS

O sistema de RGVs é constituído por um conjunto de veículos que se movimentam num circuito fechado, controlados por um "cérebro" externo que lhes atribui tarefas e gere o tráfego. Os RGVs funcionam segundo o princípio de *Master-Slave*, sendo que o *Master Controller* é o responsável por atribuir ordens e gerir o fluxo de trabalho e o *Slave* de cumprir esses comandos.

Um circuito de RGVs é constituído por um circuito em carril fechado, agulhagens, um conjunto de veículos, um controlador de veículos, transportadores de *interface* e portas de acesso.

3.1.1. O LAYOUT

O circuito de RGVs, bem como toda a instalação na fase de projeto, é desenhado em AUTOCAD e representa, a pormenor, tudo o que vai ser instalado. Este *layout* tem de ser traduzido e introduzido no controlador Master de modo a que este tenha perceção do formato físico do circuito que está a controlar.

Dos equipamentos representados no esquema da instalação, para o sistema de RGVs são relevantes:

- O formato do circuito (comprimento de retas e curvas);
- O formato do veículo (comprimento);
- Os transportadores que realizam *interface* com os veículos;
- As portas de acesso ao circuito.

3.1.2. O VEÍCULO RGV

Um RGV é um veículo autónomo, guiado por carril, que tem como objetivo transportar unidades de carga entre dois ou mais pontos a alta velocidade. Este tem dois eixos de movimentação: a translação, ou X e o transportador, ou Z.

O veículo é constituído por uma estrutura tubular, com quatro rodas sendo que duas destas são apoiadas no carril de alimentação e as outras duas no piso. Conta também com um ou mais objetos do tipo transportador que são os responsáveis por transferir e transportar as unidades de carga em segurança. O transportador pode ter o formato de rolos, correntes ou até garfos, embora este último requeira a utilização de mais um eixo de movimentação – a elevação, ou Y.

A Figura 5 representa uma visão em perspetiva de um RGV equipado com um transportador de rolos. Dos equipamentos que constituem o veículo destacam-se:

- Controlador – normalmente um PLC de baixa gama;
- *Encoder* absoluto;
- *Access Point* (AP).

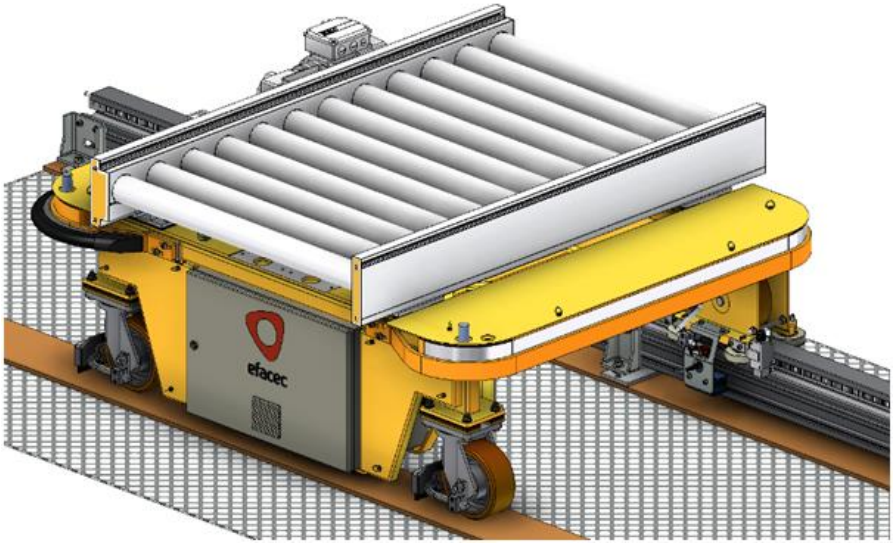


Figura 5 O veículo RGV

O Controlador é responsável por orquestrar toda a lógica do veículo, desde um simples acender e apagar um *Light Emitting Diode* (LED) até ao cálculo de velocidade a cada instante, na movimentação entre dois ou mais pontos. Este é constituído normalmente por um PLC da marca Siemens, da série S7-1200 – PLC publicitado pela Siemens para aplicações de baixa escala.

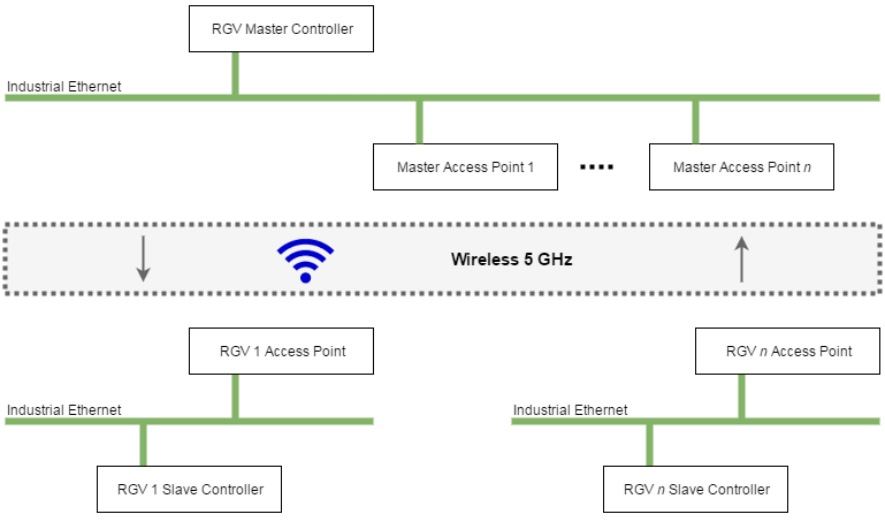


Figura 6 Esquema de ligação entre dispositivos

O PLC do veículo deve adquirir a sua posição (em milímetros) no circuito através de um *encoder* absoluto montado numa das rodas de translação. O *encoder* é, por norma, um leitor de código de barras que lê valores a partir de uma fita colocada ao longo de todo o circuito. A fita é uma sucessão de códigos de barras consecutivos que representam um valor absoluto em milímetros do seu comprimento à semelhança de uma régua. A posição

do veículo é enviada para o controlador *Master*, junto com outras informações relevantes como estado e modo de funcionamento.

A comunicação entre o veículo e o *Master Controller* pode ser estabelecida de duas formas: através do carril da alimentação, utilizando duas ou mais pistas, criando desta forma uma ligação física, ou ser feita de forma *wireless* através de um *Access Point*. A Figura 6 representa o conceito de ligação entre o *Master* e os RGVs para uma arquitetura de ligação sem fios. O AP é um cliente sem fios responsável por colocar o veículo na mesma rede do *Master*.

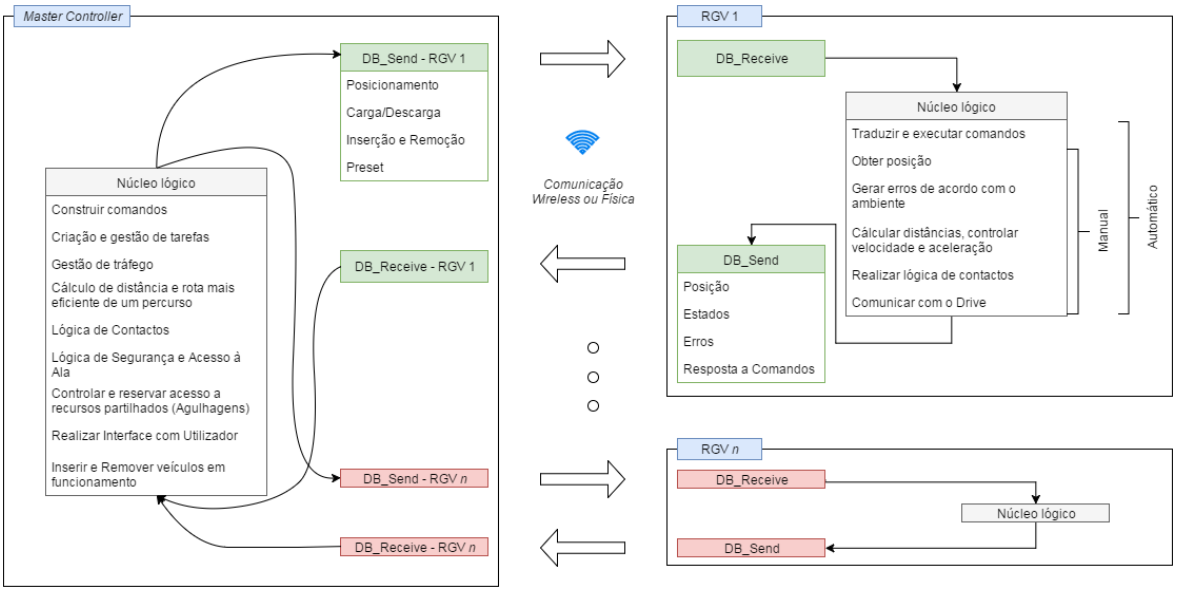


Figura 7 Arquitetura Sistema de RGVs

A ligação estabelecida entre o *Master* e o veículo é do tipo *ISO-on-TCP*, protocolo usado pela Siemens para comunicar entre os seus PLCs. A Figura 7 representa um diagrama da arquitetura do sistema de RGVs, nomeadamente na troca de informação entre o(s) veículo(s) e o *Master*.

O veículo tem dois modos de funcionamento: Manual e Automático. Quando se encontra em Manual, este é controlado por um comando local que dá ordens aos vários eixos para se moverem.

Em Automático, o veículo recebe ordens do *Master Controller* e procede à tradução das mesmas em ações físicas como posicionamentos, movimentos de carga e descarga, mudança de estado, entre outros. O *Master* tem o poder de fazer um veículo entrar em

modo de erro ou *Stop* quando deteta alguma anomalia no funcionamento do mesmo. Este é também o responsável por limpar os erros, enviando um comando de *Reset* a todos os veículos, colocando-os em funcionamento normal.

A comunicação consiste no envio e receção de bases de dados (DBs) entre o *Master* e os vários *Slaves*. A DB de envio do veículo para o *Master* tem informações relevantes como posição no circuito, estado dos sensores, estado de erros, resposta a comandos recebidos, estado de execução de comandos, entre outros. Já a DB de envio do *Master* para cada veículo possui posição de destino, ordens de *Stop* ou *Reset*, carga ou descarga ou simples posicionamentos.

3.1.3. OS TRANSPORTADORES DE INTERFACE

Os transportadores da instalação são responsáveis por movimentar unidades de carga, a velocidade reduzida entre dois ou mais pontos da instalação. Estes fazem parte do sistema de armazenamento automático e comunicam com os restantes equipamentos de forma a entregar e receber unidades sem perda de *tracking*.

Os transportadores que realizam *interface* com os RGVs podem comportar-se como Entrada, Saída ou as duas em simultâneo. Em modo de Entrada, na perspetiva do veículo, o transportador entrega uma unidade de carga ao circuito. Por outro lado, em modo de Saída, recebe unidades e transporta-as para outras localizações da instalação.

Transportadores Mistos são os que se comportam como Entrada em certas alturas e como Saída nas restantes. Para que isto aconteça, existe uma troca de informação entre o PLC dos transportadores e *Master Controller*, informando-o sobre o modo de funcionamento atual.

3.1.4. AS AGULHAGENS

Uma agulhagem, na ótica do sistema de RGVs, é um equipamento que altera um percurso reto para curvo, e vice-versa, adaptando o circuito às exigências dos movimentos em curso.

Na Figura 8 é possível observar o conceito de agulhagem. Está ilustrado um exemplo prático de aplicação destes sistemas com duas agulhagens que cria um atalho para chegar a um ponto do circuito.

As agulhagens são usadas para dividir o circuito em vários segmentos, criar rotas mais eficientes para os veículos e criar zonas de parque para os estacionar quando não são necessários, tornando o circuito mais eficiente.

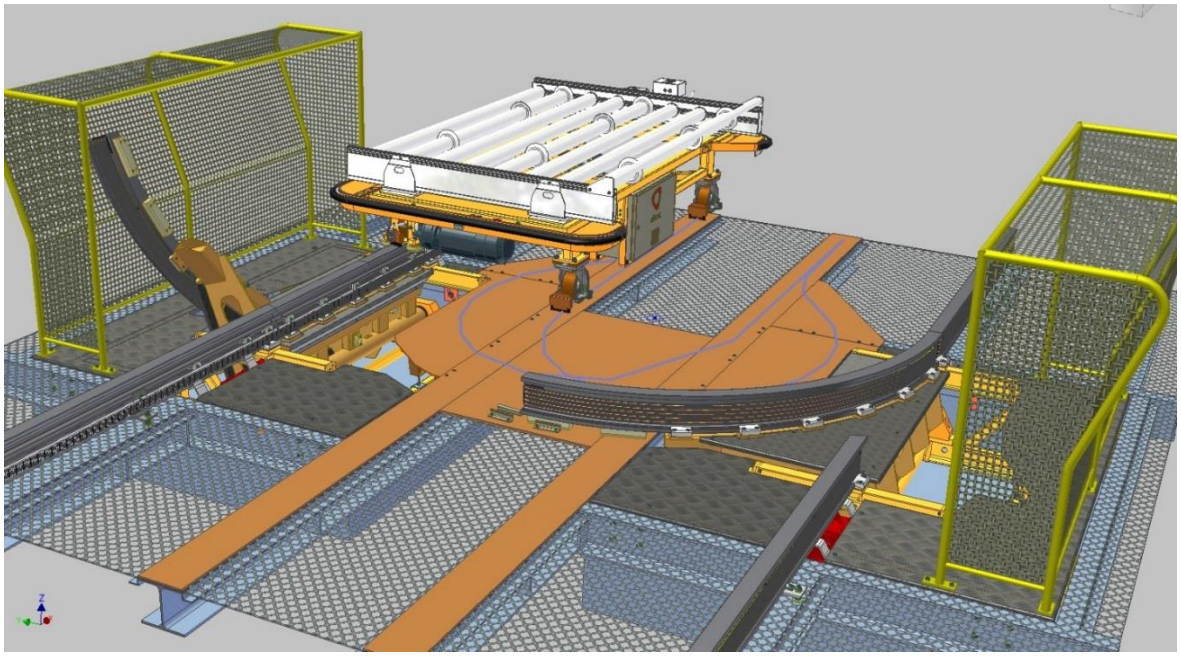


Figura 8 Sistema de Agulhagens

O programa da agulhagem é um *software* independente que pode ser executado num PLC separado ou dentro do *Master Controller*.

3.1.5. O MASTER CONTROLLER

O controlador *Master* é um *software* inteligente, normalmente alojado num PLC, que efetua todas as funções principais do sistema e tem em seu controlo todos os veículos. As tarefas principais deste controlador são:

- Supervisionar as seguranças de sistema e comunicar com PLC de segurança;
- Comunicar com um ou mais PLCs dos transportadores;
- Comunicar com a gestão do armazém, o WMS;
- Comunicar com todos os RGVs inseridos no circuito;

- Gerar tarefas de acordo com as cargas presentes nos transportadores de interface, seguindo um critério de *First In First Out* (FIFO);
- Atribuir tarefas de acordo com o posicionamento dos veículos;
- Inserir e Remover veículos em funcionamento;
- Gerir o tráfego no circuito de forma a não existir choques e a fazer parar os veículos em pontos específicos de parque quando não há fluxo de trabalho;
- Controlar e reservar acesso a recursos partilhados como Agulhagens.

O *Master* tem de ter conhecimento do comprimento do circuito, das posições dos transportadores de *interface*, das posições de início e fim de agulhagens entre outras informações relevantes.

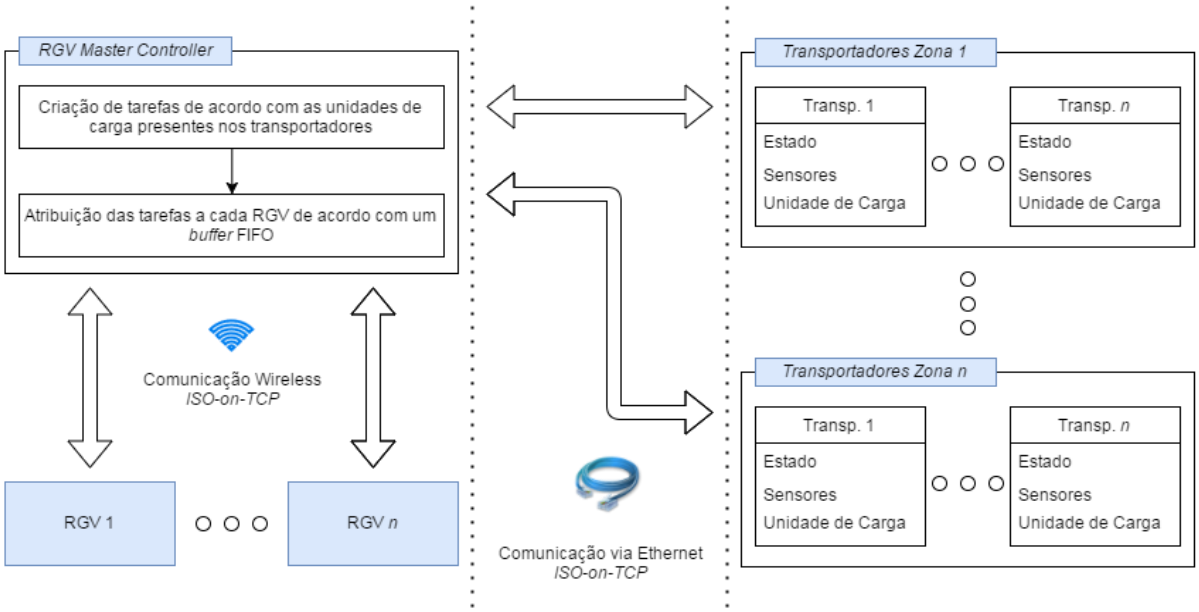


Figura 9 Arquitetura da criação e atribuição de tarefas

Devido à extensão de alguns circuitos, que podem ter vários quilómetros de comprimento, a instalação de transportadores pode estar dividida de forma a serem controlados por vários PLCs. O *Master* tem assim de comunicar com cada um desses PLCs de forma a obter os dados dos transportadores que realizam *interface* com os seus RGVs.

A gestão e atribuição de tarefas é uma função complexa que depende de muitos fatores, dentro dos quais:

- A distância do veículo ao ponto de interface:
    - Deve ser selecionado o veículo mais próximo sempre que possível;
- Estado do transportador de origem e destino:
    - Só devem ser atribuídas tarefas que tenham condições físicas e lógicas para serem executadas;
- O tempo de espera:
    - Deve atribuir-se a tarefa com maior tempo de espera;
- Pontos prioritários:
    - Deve atender-se tarefas de pontos prioritários antes das restantes;
- Otimizar movimento dos RGVs:
    - Deve aproveitar-se o movimento dos RGVs em vazio, na movimentação para o ponto de origem de uma tarefa, para realizar tarefas intermédias, aumentando assim a eficiência do circuito.

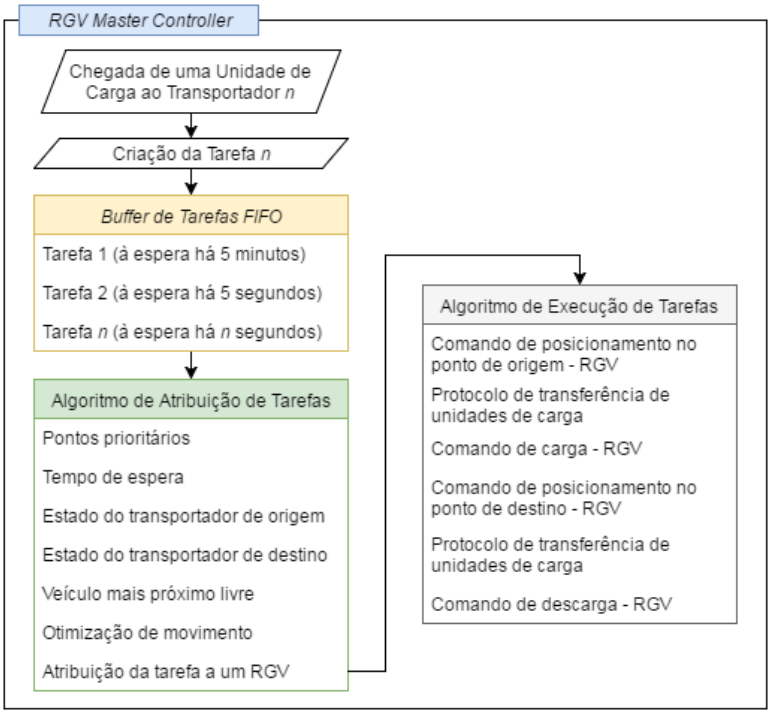


Figura 10 Criação, gestão e execução de tarefas

Uma tarefa é uma ação gerada pelo *Master Controller* aquando da chegada de uma unidade de carga a um transportador de *interface*, em modo de Entrada. As tarefas são introduzidas num *buffer*, ordenado por tempo de espera, seguindo um critério de FIFO.

A Figura 10 representa a criação, gestão e execução de uma tarefa, da perspetiva do *Master Controller*. O *buffer* ordenado é usado pelo algoritmo de atribuição que é responsável por escolher o veículo em melhores condições para executar a tarefa. Uma vez atribuída a tarefa, esta é entregue ao algoritmo de execução que gere o envio de comandos de posicionamento/carga/descarga ao veículo e realiza o protocolo de transferência de unidades com os transportadores.

O grau de eficiência e eficácia de um sistema de RGVs é medido pela capacidade de movimentação de unidades de carga por hora, mais conhecida por Cadência. Existem outros indicadores de *performance* como o grau de ocupação física de um veículo mas, normalmente, todos dependem, em parte da Cadência do sistema.

3.1.6. A NECESSIDADE DE SIMULAR

Os constantes avanços tecnológicos levam invariavelmente à construção de sistemas com várias camadas de complexidade. Os simuladores proporcionam uma forma de entender e analisar sistemas do mundo real reproduzindo o comportamento dos mesmos através de *hardware* e *software*. Então, pode-se definir que uma simulação é o processo de desenhar um modelo de um sistema real e efetuar experiências com esse modelo com o propósito de entender o comportamento do sistema ou avaliar várias estratégias de configuração para a operação do mesmo.

Um dos fatores importantes num simulador é a fidedignidade. Esta pode ser provada através da realização de um teste no mundo real que produza resultados aproximados aos simulados. Outro fator relevante é a repetibilidade – a capacidade do sistema produzir os mesmos resultados, nas mesmas condições de entrada.

Como visto nos subcapítulos anteriores, a determinação da Cadência em fase de projeto é complexa, devido à quantidade de fatores envolvidos. Alterar um simples fator como a velocidade máxima de um veículo pode produzir um valor de cadência resultante completamente diferente.

Simulações como as da SIMCORE (2.2), são úteis para determinar o funcionamento do conceito mas, por não possuírem o modelo real dos equipamentos, não são válidas para determinar cadências. O desenvolvimento de um simulador que se comporte de acordo

com um modelo real (executando o mesmo programa presente no controlador do mesmo) possibilita determinar com precisão o grau de *performance* do sistema em fase de projeto, bem como ajudar o especialista a conceber o programa.

3.2. SOLUÇÃO DE SIMULAÇÃO

O simulador proposto nesta Tese tem como objetivo principal testar o programa real de um PLC, o *Master Controller*. A Figura 11 representa a solução de simulação proposta tendo em conta todos os equipamentos existentes num circuito de RGVs.

O *Master Controller* é executado num PLC real ou simulado, possibilitando assim a deteção e correção de erros no respetivo programa. Os programas dos transportadores de *interface*, como o dos RGVs, são emulados, simulando o comportamento real dos mesmos.

Os transportadores de *interface* são máquinas simples que têm como tarefa entregar ou receber unidades de carga dos RGVs, seguindo um protocolo pré-definido, para sincronizar a transferência física – troca de unidades de carga – e a transferência lógica – troca de dados informáticos.

Os RGVs, apesar de possuírem um PLC de baixa gama, têm um programa complexo que tem de ser parcialmente traduzido para a simulação de forma a se poder simular *n* veículos sem a necessidade de se ter *n* PLCs.

Estes equipamentos emulados enviam e recebem informações do *Master* através de bases de dados de memória que contêm informações como receção de comandos e envio de estados.

Os programas das agulhagens, por outro lado, são simulados de forma diferente. O simulador emula o comportamento físico, estando o processamento da agulhagem no programa do *Master Controller* havendo apenas uma troca dos sinais de Entrada/Saída.

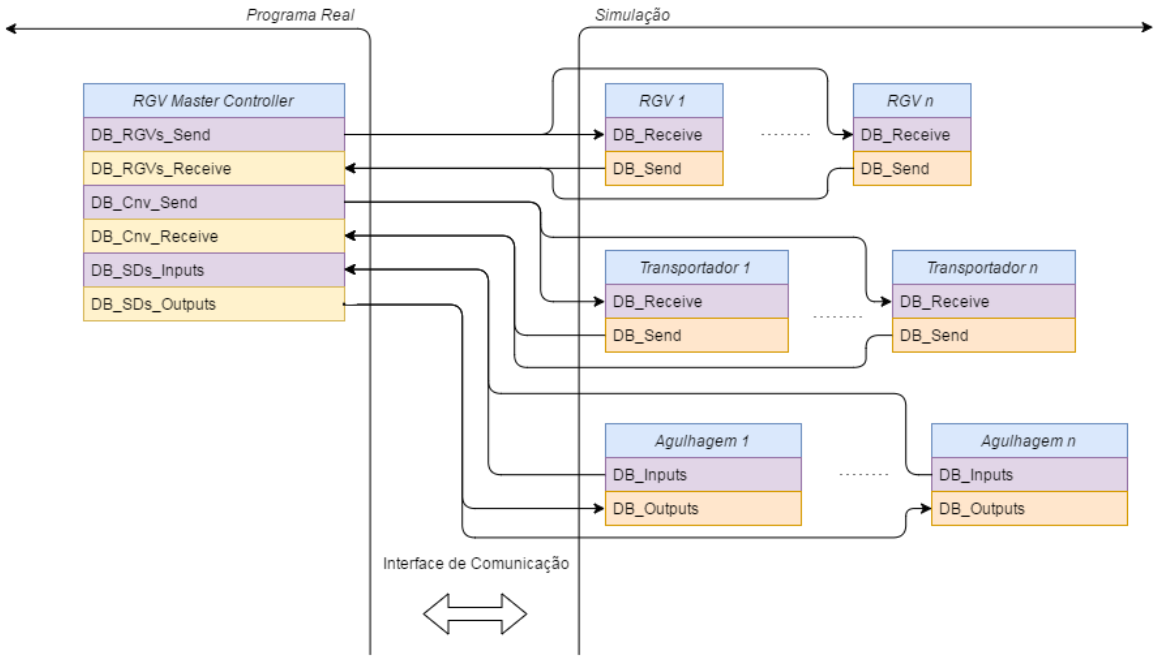


Figura 11 Equipamentos em simulação

Os equipamentos simulados trocam informação com o *Master Controller* através de uma *interface* de comunicação que tem a capacidade de ler e escrever em zonas de memória do PLC *Master Controller*.

A EHS realiza várias obras por ano com circuitos de RGVs pelo que é imperativo que o simulador seja genérico e flexível a diferentes instalações. Assim, para satisfazer esse requisito, tem de se proporcionar uma forma de configuração do sistema.

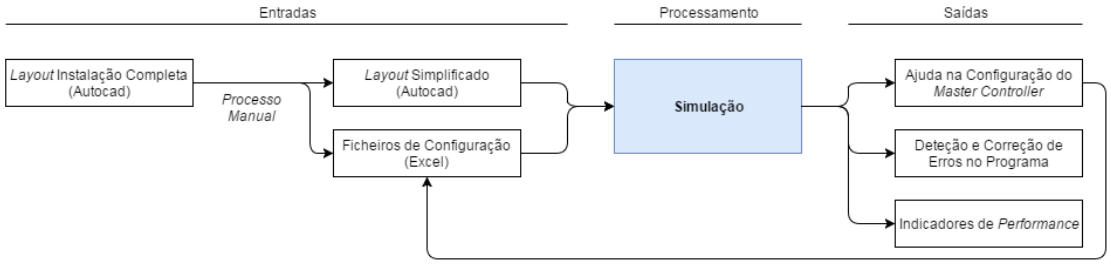


Figura 12 Arquitetura geral do sistema

A Figura 12 representa a arquitetura geral do sistema. Como se pode ver na figura, o sistema tem como entradas o *layout* e ficheiros de configuração e como saídas os benefícios já mencionados no capítulo 1 – deteção e correção de erros remota e indicação de *performance*.

O sistema do simulador permitirá também ajudar na configuração do *Master Controller* pois determinará a posição em *mm*, de início e fim de segmento e de agulhagem bem como a posição de todos os pontos de posicionamento.

3.2.1. LAYOUT

O *layout* usado como entrada para a simulação foi criado com o intuito de uniformizar os sistemas de RGVs e é concebido a partir do *layout* geral da instalação, de forma manual. Aqui estão representados apenas os equipamentos e informações que dizem respeito ao sistema de RGVs.

A Figura 13 representa o conceito de um circuito de RGVs com agulhagens e vários segmentos. Como se pode observar na figura, um segmento é uma fração do circuito que tem um único ponto de entrada e um outro de saída. As posições dentro do segmento são calculadas tendo em conta o valor mínimo de fita definido pelo especialista, e o valor máximo é calculado com base no comprimento das várias retas e curvas constituintes do mesmo.

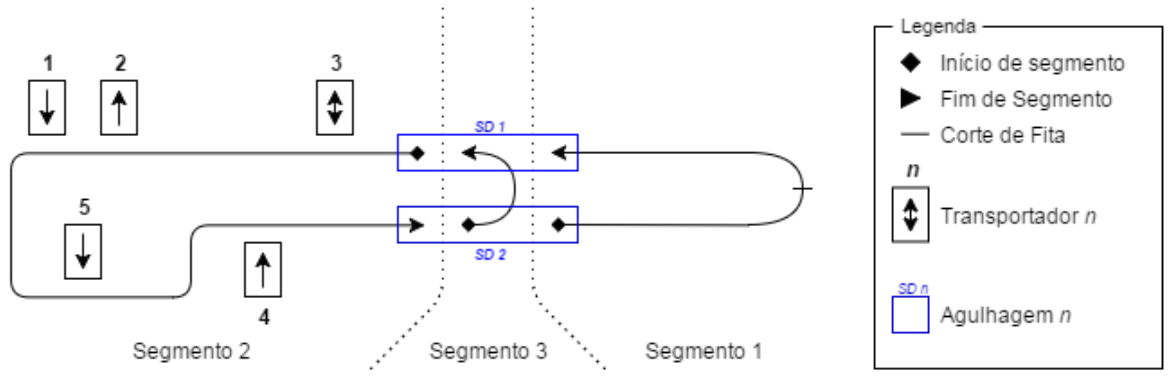


Figura 13 *Layout* – Conceito de Circuito de RGV

No exemplo apresentado, o circuito está divido em três segmentos. O primeiro é usado como parque, o segundo como segmento de "trabalho" e o terceiro como atalho. Para modificar o seu formato, o circuito conta com duas agulhagens que alteram de estado entre curva e reta.

O *layout* do circuito é utilizado pelo especialista como "Mapa" e é usado para configurar o *Master Controller*. Este é desenhado em AutoCAD, produzindo um ficheiro do formato *Drawing* (DWG) com o aspeto da Figura 14.

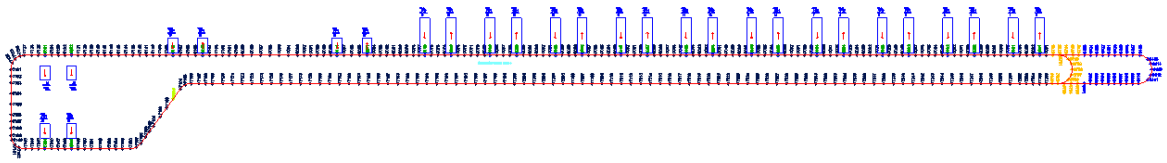


Figura 14 ***Layout*** **– Representação de um Circuito de RGVs em AutoCAD**

O simulador terá de ser capaz de carregar este *layout* e representá-lo no espaço 3D. Para os vários equipamentos são definidos objetos diferentes, como é o caso dos transportadores, representados na Figura 15 (a).

SEGMENTOS

Os segmentos são representados na forma de curvas e retas, que podem ser independentes ou fazer parte de uma *polyline*. Uma *polyline* é uma entidade de desenho que representa uma sequência de fragmentos de linha que podem ser curvos ou retos.

Quando são utilizadas curvas e retas independentes é necessário definir uma sequência e é impossível, sem a ajuda de outros objetos, determinar o início e fim de segmento.

Ao utilizar uma *polyline* para definir o segmento, a sequência, início e fim de segmento está automaticamente definida, ficando apenas a tradução de linhas em curva ou reta por determinar.

TRANSPORTADORES DE *INTERFACE*

Estes objetos simbolizam, da esquerda para a direita, os transportadores de Entrada, Saída e Mistos que estão também providos de três parâmetros adicionais que os identificam no circuito, apresentados na tabela de atributos da Figura 15 (a).

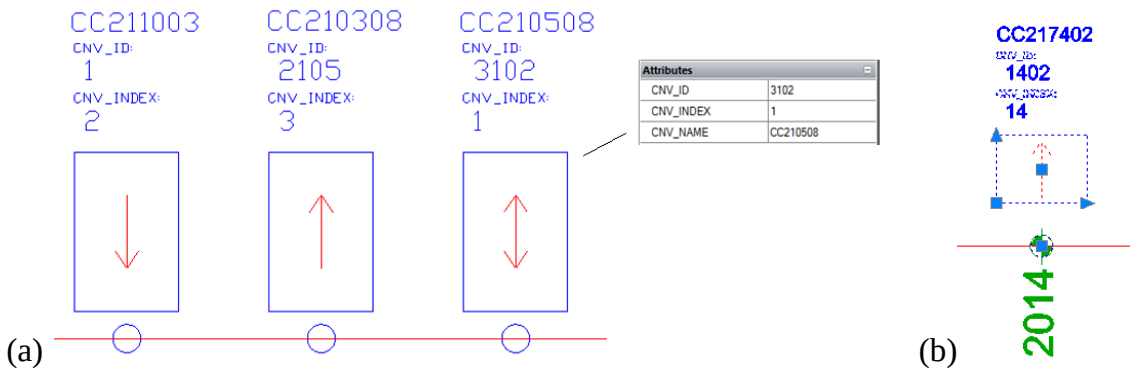


Figura 15 ***Layout*** **–Transportadores de** ***Interface*** **(a) configuráveis (b)**

O objeto transportador tem como objetivo ser genérico para todos os tipos de transportadores da instalação. Na Figura 15 (b) é possível verificar que o objeto é configurável e constituído por:

- Um retângulo, que pode ser editado para ter comprimento e largura do transportador real;
- Um círculo, cujo centro indica a posição de inserção do transportador no circuito;
- Uma seta, que indica o tipo de transportador.

PONTOS DE POSICIONAMENTO

Para identificar posições no circuito, sejam elas de paragem, sinalização ou simples indicações de início e fim de fita de posicionamento, são usados objetos Ponto, como representados na Figura 16 (a). Estes possuem um parâmetro de identificação, apresentado na Figura 14 (b), usado pelo *Master Controller*, para o envio da posição final ao veículo.

O simulador terá de ter a capacidade de converter as posições XYZ de inserção destes objetos em posição de fita (em *mm*), tendo em conta o início e fim de cada segmento e a posição de início e fim de fita.

(a)

(b)

Figura 16 ***Layout* – Pontos de Posicionamento (a) e Atributos (b)**

AGULHAGENS

As agulhagens são objetos constituídos por uma curva e uma reta, como representado na Figura 17. O simulador irá utilizar a informação de início e fim destes elementos para determinar qual o segmento seguinte, de acordo com o estado da agulhagem.

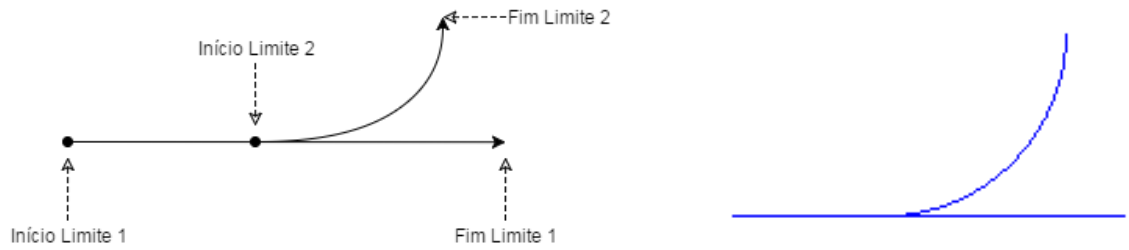


Figura 17 *Layout* – Conceito e Representação de uma Agulhagem

Para distinguir os vários objetos no *layout*, estes estão divididos por tipo, em camadas:

- Transportadores: contém todos os transportadores de *interface* do circuito;
- Pontos de Paragem;
- Pontos de Parque;
- Pontos de Manutenção;
- Pontos de *Interface*;
- Agulhagens;
- Segmentos.

3.2.2. ARQUITETURA DE SOFTWARE

O simulador é um projeto de grande dimensão e, por essa razão, está dividido em vários módulos que são responsáveis por realizar tarefas específicas que ficam, desta forma, melhor organizadas.

A Figura 18 apresenta a arquitetura geral do sistema. As entradas estão assinaladas a azul, sendo estas o *layout* em AutoCAD, as configurações do *layout* e simulador em Excel e a configuração da Interface com o Utilizador, em *Extensible Markup Language* (XML).

O desenho é entregue ao Interpretador de *Layout*, responsável por traduzir todos os objetos AutoCAD, criando um objeto *Percurso* que contenha a informação sobre os segmentos, agulhagens, transportadores e pontos de posicionamento.

A função principal do simulador, que tem como tarefa gerir todo o sistema, está alojada no módulo Núcleo de Simulação. É aqui que é efetuada toda a lógica relacionada com o simulador bem como a criação e gestão dos objetos 3D.

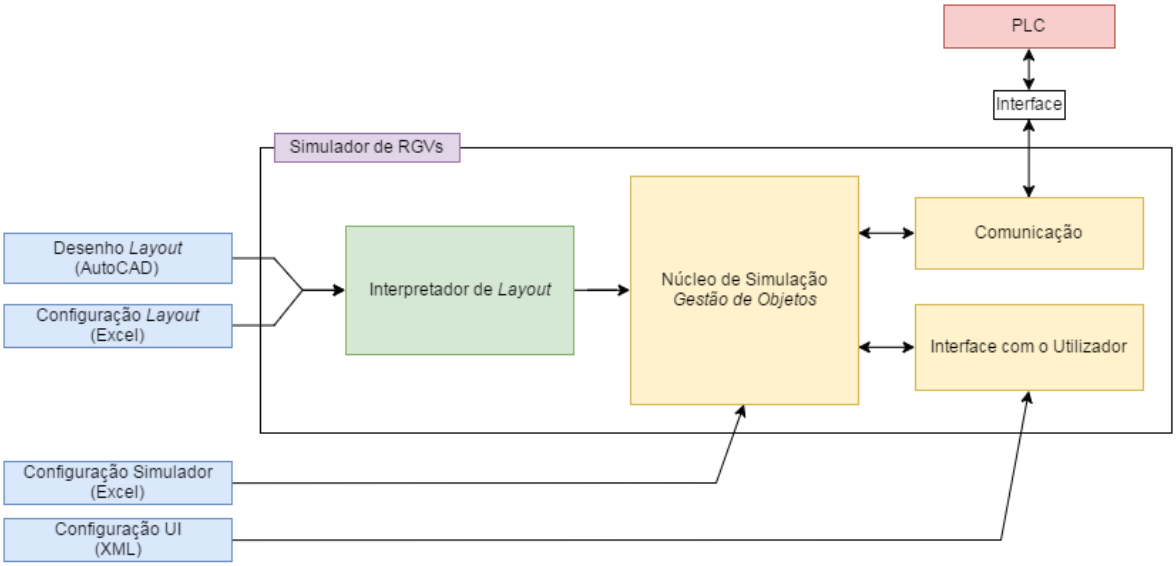


Figura 18 Arquitetura de *Software*

Para permitir a interação com o utilizador existe um módulo que faz a ponte entre o utilizador e a simulação – *Interface* com o Utilizador.

O sistema usa como saída e entrada informações do PLC externo, assinalado a vermelho. As informações são geridas por um módulo de comunicação que, por sua vez, utiliza uma *interface* que tem a capacidade de ler e escrever em zonas de memória de um PLC.

O funcionamento de cada módulo, bem como o dos algoritmos envolvidos, é apresentado no Capítulo 4 - Implementação.

3.2.3. ARQUITETURA DE HARDWARE

Ao nível do *hardware*, o sistema conta com duas configurações possíveis, apresentadas na Figura 19 e Figura 20.

Na primeira configuração, o computador onde é executado o simulador de RGVs comunica com um PLC real através de uma interface, utilizando como meio físico comunicação *Wireless* ou *Ethernet*.

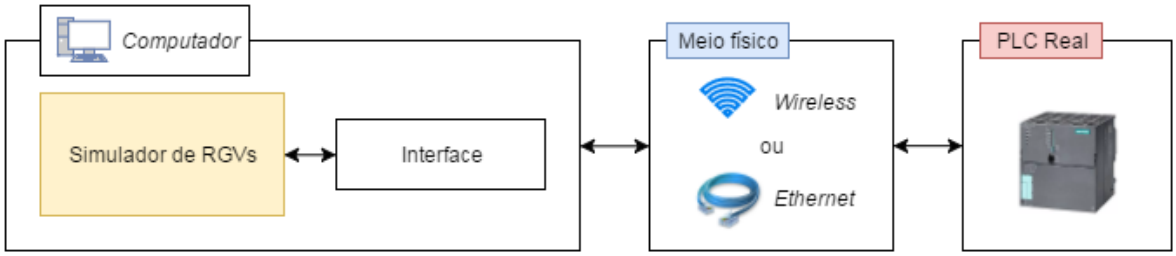


Figura 19 Arquitetura de *Hardware*: Configuração 1

Na segunda configuração, é tudo executado dentro do computador. Como se pode observar na Figura 20, nesta configuração é utilizado um PLC simulado. A simulação é disponibilizada pelo *software* SIMATIC S7–PLCSIM, proprietário da Siemens.

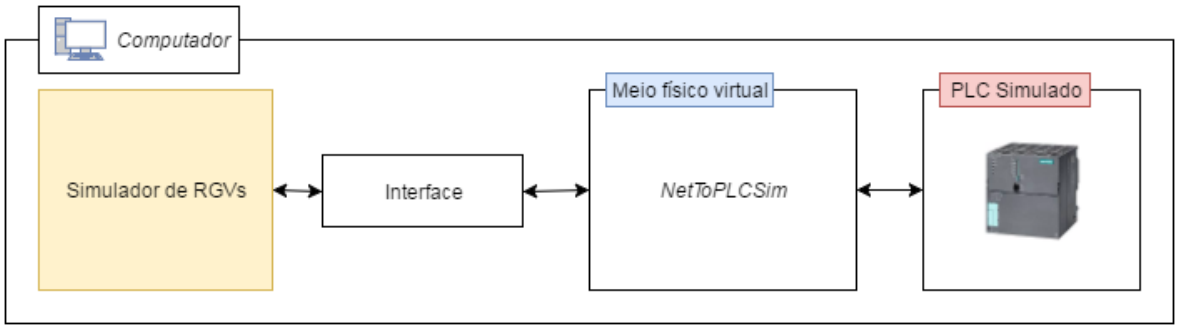


Figura 20 Arquitetura de *Hardware*: Configuração 2

Embora a simulação traduza o comportamento de um PLC real, não é possível interagir com programas externos. Para que isto se torne possível, é utilizado o NetToPLCSim, um projeto de *software open source*, que utiliza o protocolo *S7 Online* da Siemens para adquirir dados do PLCSIM.

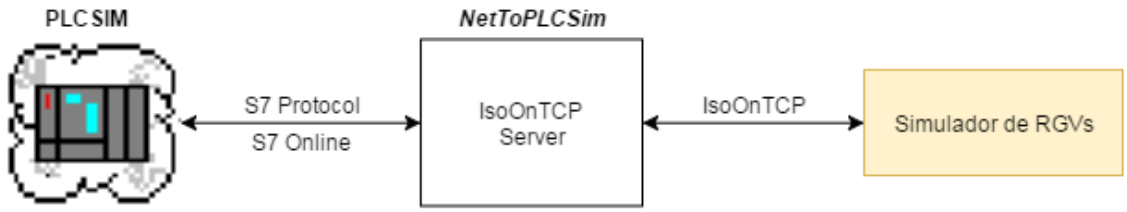


Figura 21 Funcionamento do NetToPLCSim

O NetToPLCSim é então usado para criar um meio físico "virtual", tornando esta configuração transparente do ponto de vista do Simulador de RGVs.

4. IMPLEMENTAÇÃO

Neste capítulo é apresentada a implementação do sistema, a sequência de desenvolvimento e o funcionamento de cada módulo, explicando alguns dos algoritmos mais importantes.

4.1. INTERPRETADOR DE *LAYOUT*

O Interpretador de *Layout* é um módulo do Simulador de RGVs encarregado por traduzir o desenho AutoCAD de forma ao simulador ter conhecimento do formato físico do circuito bem como de todos os objetos que têm de ser criados. A Figura 22 representa o funcionamento do módulo, identificando as entradas e saídas do mesmo.

O *layout* concluído é guardado em formato DXF, um ficheiro de texto que contém informação sobre todas as entidades do desenho AutoCAD. Para ler o ficheiro de *layout*, em DXF, o interpretador utiliza uma biblioteca *open source*, o *dxfgrabber*. Esta lê o ficheiro de texto e organiza tudo num objeto que depois é utilizado pelo módulo para transformar o desenho 2D numa representação 3D do circuito de RGVs.

Existe também, como entrada do módulo, um ficheiro de configuração que indica informações sobre o *offset* de posicionamento, *offset* de início de fita e início de cada um dos segmentos. Este ficheiro é criado em Excel e o módulo de interpretação conta com a ajuda de uma biblioteca *open source* especializada em ler e escrever em ficheiros deste tipo, o *openpyxl*.

O módulo cria um objeto Percurso que contém toda a informação necessária para o simulador criar os objetos 3D nas coordenadas corretas:

- Segmentos: Sequência de objetos Segmento;
- Pontos de Inserção de RGVs: Lista de pontos para inserir objetos do tipo RGV;
- Pontos de Posicionamento: Lista de pontos existentes no *layout*;
- Transportadores de Interface: Sequência de objetos Transportador;
- Comprimento do circuito;
- Máximo e mínimo de fita de posicionamento.

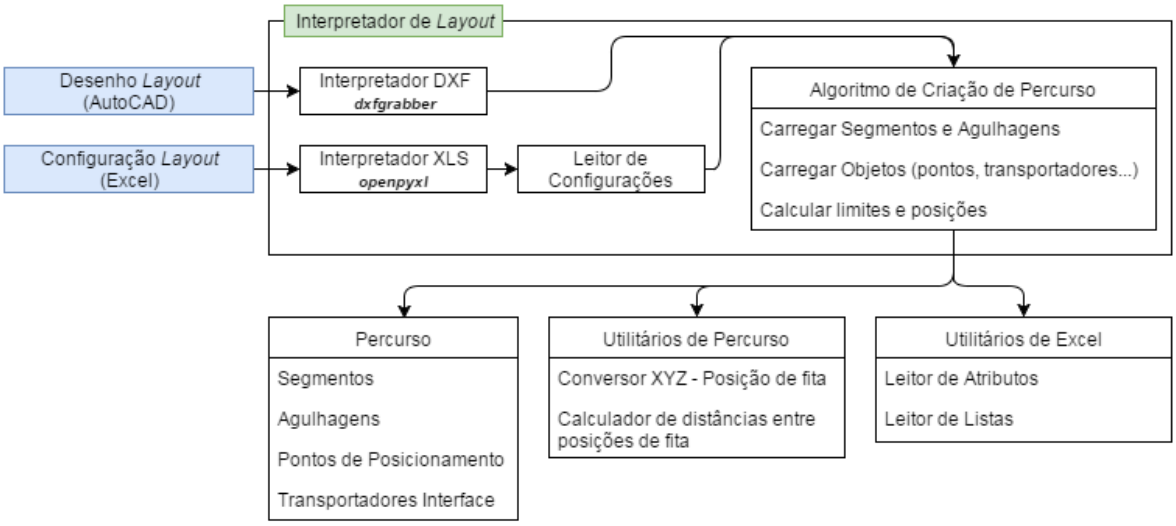


Figura 22 Simulador – Interpretador de *Layout*

Cada um dos vetores referidos contém um conjunto de objetos específicos que têm informações sobre cada tipo de objeto.

4.1.1. OBJETO SEGMENTO

Para cada *layer* que contenha a palavra "SEGMENT_XXX" no seu nome, onde XXX indica o número do segmento, o interpretador cria um novo objeto Segmento.

Um segmento é uma sequência de curvas e retas com um só ponto de entrada e outro de saída. Os segmentos podem ser desenhados com retas e curvas independentes ou criados a partir de uma *polyline*.

Como apresentado na Figura 24 (b), o objeto tem as informações necessárias para a construção de uma representação gráfica de um segmento, dentro das quais a sequência de entidades e vértices. A sequência de entidades é uma lista ordenada com as curvas e retas que constituem o segmento.

O início do segmento é determinado no ficheiro de configuração pelo especialista. Caso contrário, o início de cada segmento é sempre o valor final do anterior, somado de um milímetro. O fim é calculado tendo em conta o comprimento das várias entidades do segmento.

A Figura 23 apresenta um exemplo de numeração de início e fim de fita de posicionamento num circuito com três segmentos. Como se pode observar, o segmento 1, que contém o início e fim de fita, denominado como "Zero" tem início superior ao final. O segmento 2 segue-se com início igual ao final do anterior somado de um milímetro. Já o segmento 3 pode ter início calculado automaticamente, ou ser definido pelo especialista. No caso apresentado, o especialista optou por escolher definir o início do segmento 3 com 8000 mm.

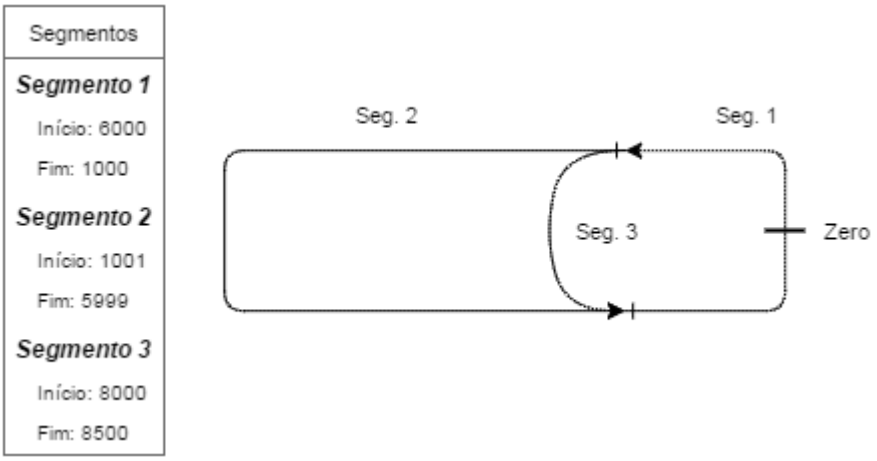
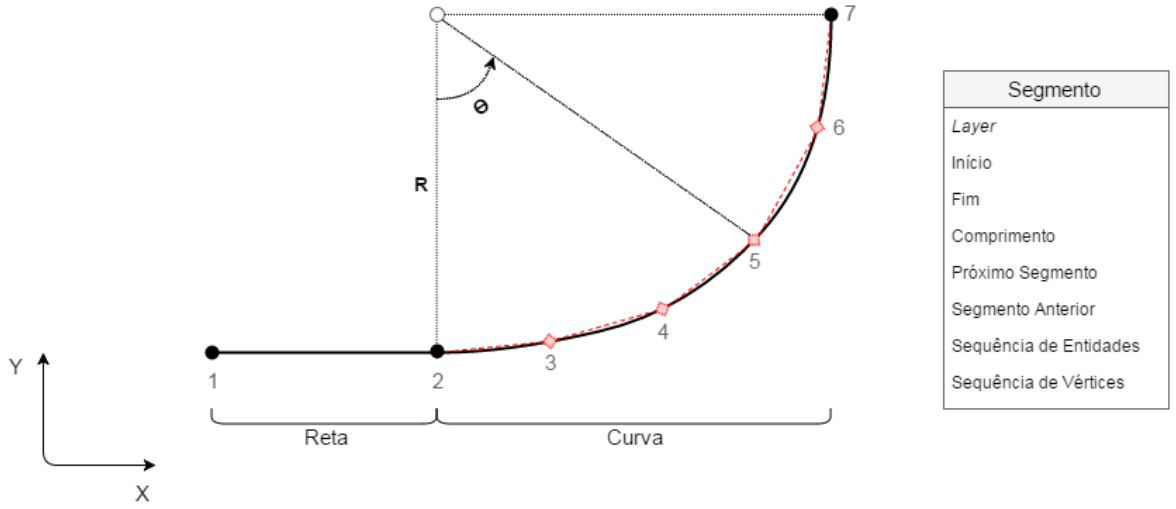


Figura 23 Objeto Segmento: Exemplo de Numeração

O algoritmo de criação de Percurso cria um vetor com vértices, com posições em formato XYZ, que são depois usados pelo simulador para orientar as rodas do veículo. Se, por um lado, uma reta tem apenas dois vértices, uma curva pode ter um vasto conjunto.

Uma sequência de vértices tem o aspeto da Figura 24 (a). Tal como representado, as retas têm dois vértices, o início e fim. Já uma curva pode ter *n*, formando uma aproximação com segmentos de reta, como a representado a vermelho.



(a) (b)

Figura 24 Objeto Segmento: Objeto e Exemplo

Se considerarmos o exemplo da Figura 24 (a), o segmento é constituído por duas entidades, uma reta e uma curva. O ponto n.º 1 indica o início do segmento e o n.º 7 o fim. No exemplo, a reta é composta por dois vértices, o início e o fim, enquanto a curva é composta por seis. A posição XY destes é calculada de acordo com uma fração do ângulo total da curva.

A sequência de vértices é a responsável por guiar as rodas dos veículos segundo o carril. A posição de um qualquer ponto na curva é dada pelo ângulo entre a reta desde o centro da curva ao ponto inicial 2, e a reta desde o centro ao ponto a calcular multiplicado pelo raio da mesma. Sendo assim, a posição do ponto 5 é dada pela equação (1).

$$P5\ (mm) = \theta \times R \qquad (1)$$

Tendo $\theta\ (rad) = \frac{\pi}{4}$ e $R\ (mm) = 500$, então P5 (mm) ≈ 393.

Esta é a equação mais tarde utilizada para determinar a posição em milímetros da roda de um RGV em curva.

4.1.2. OBJETO AGULHAGEM

Para cada *layer* que contenha a palavra "SD_XXX" é criado um objeto do tipo Agulhagem, onde o número "XXX" representa o índice lógico da agulhagem.

Uma agulhagem é um equipamento responsável por alternar o seu estado entre curva e reta de forma a modificar a morfologia do circuito para cumprir as necessidades dos movimentos em curso.

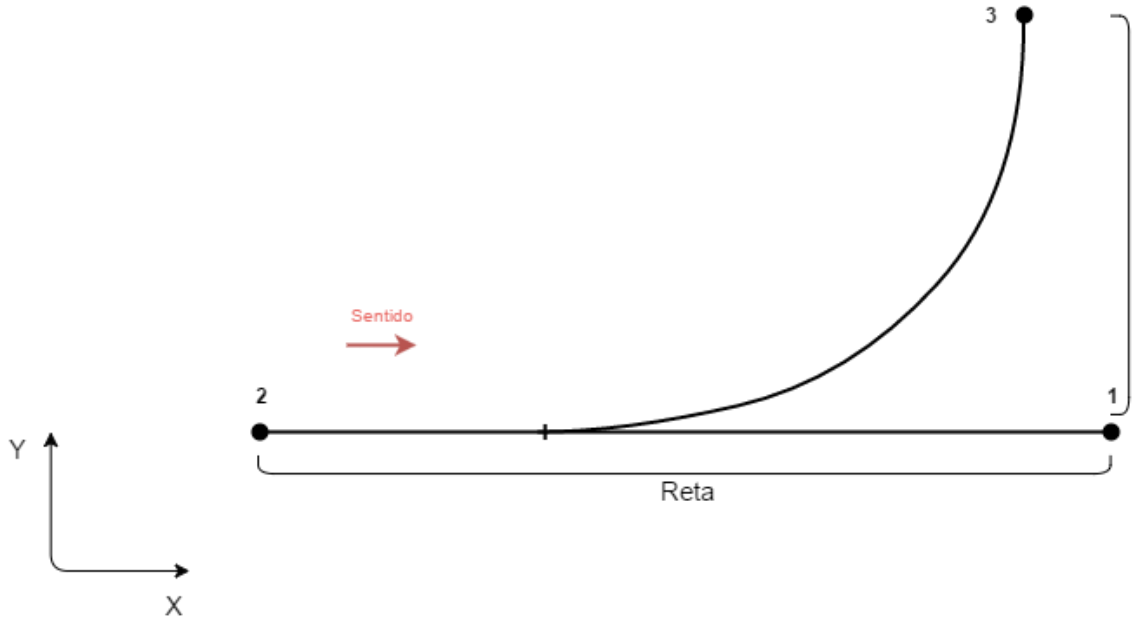
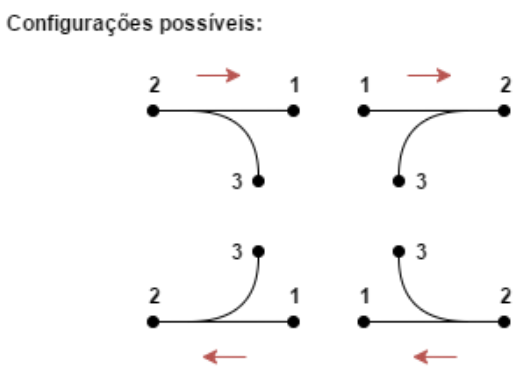


Figura 25 Objeto Agulhagem: Conceito

A representação de uma agulhagem em AutoCAD é formada por uma reta e uma curva. A Figura 25 apresenta o conceito do objeto e a Figura 26 as configurações possíveis no circuito, bem como os seus atributos.

O objeto criado contém informação sobre a *layer* onde é desenhado, o estado atual, se em curva ou reta e os segmentos associados.



Agulhagem
Layer
Estado (Curva/Reta)
Próximo Segmento em Curva
Próximo Segmento em Reta
Segmento Anterior
Sequência de Vértices Reta
Sequência de Vértices Curva

Figura 26 Objeto Agulhagem: Configurações Possíveis

Os pontos 1, 2 e 3 apresentados são usados para identificar os segmentos que a agulhagem liga. De acordo com o sentido de movimento, as agulhagens podem ter uma entrada e duas saídas ou duas entradas e uma saída.

As sequências de vértices são usadas para desenhar a agulhagem no espaço 3D.

4.1.3. OBJETO PONTO

Para cada ponto de posicionamento no *layout* é criado um objeto do tipo Ponto. Estes são organizados por tipo, estando separados por várias *layers* para diferentes propósitos:

- Paragem;
- Transferência;
- Parque;
- Manutenção;
- Recursos Partilhados (agulhagens).

Ponto
ID
Tipo
Segmento
Entidade
Posição XYZ
Posição na Entidade
Posição no Segmento
Posição no Circuito
Offset

Figura 27 Objeto Ponto: Atributos

Embora para o simulador o tipo de ponto não seja relevante, já que a ordem de posicionamento é dada por uma entidade exterior, esta informação é útil para o *Master Controller* e usada na sua configuração.

Ainda assim, o simulador tem a possibilidade de representar os pontos no espaço 3D para que o especialista tenha uma perceção visual da localização dos mesmos. Para que isto seja possível, o objeto tem os atributos apresentados na Figura 27.

4.1.4. OBJETO TRANSPORTADOR DE INTERFACE

Para cada transportador presente no *layout* é criado um objeto do tipo Transportador de Interface. Como se pode observa na Figura 28, este é constituído por três entidades:

- Um círculo (1) que representa o ponto de inserção do transportador no circuito, ponto este que é usado para posicionar o RGV na transferência de uma unidade de carga;
- Um retângulo que retrata o tamanho físico do transportador e cujo centro (2) indica a posição de inserção do transportador no mundo 3D;
- Setas no centro (representadas a vermelho) que indicam o tipo de transportador.

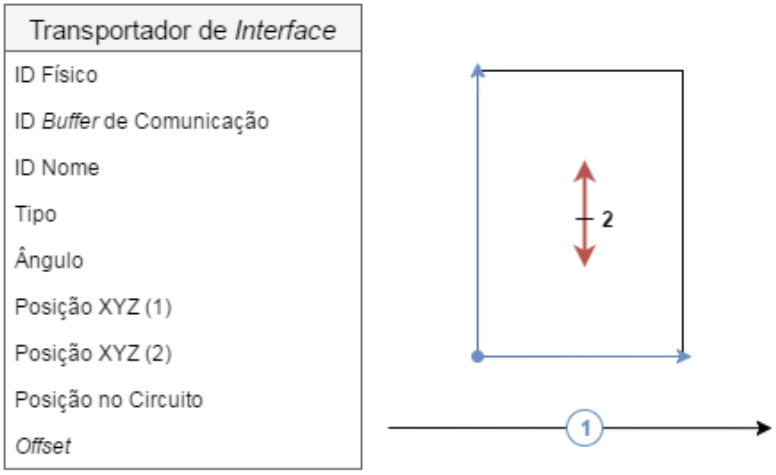


Figura 28 Objeto Transportador de *Interface*

Durante a instalação de um armazém automático, é recorrente que as posições dos transportadores não coincidam exatamente com as teóricas. No caso de ser necessário realizar testes de cadência de um programa de RGVs já instalado e em funcionamento,

cada transportador possui um parâmetro de *offset* que o faz deslocar em X e Y de forma a coincidir com a posição real instalada.

```
PARA CADA Segmento:
        Resultados = FUNÇÃO (Procurar Ponto Inicial Coincidente)
        Próx. Segmento = MÍNIMO (Resultados)
        Segmento.Próx = Próx. Segmento
        Próx. Segmento.Ant = Segmento
FIM PARA CADA

Definir posição do corte da fita
Definir posição de início do Segmento Inicial

PARA CADA Segmento:
        SE Segmento Inicial:
                CONTINUAR
        SENÃO:
                Segmento.Inicio = Segmento.Ant.Fim + 0.001
                Segmento.Fim = Segmento.Início + Segmento.Comprimento
FIM PARA CADA
```

Algoritmo 1 Definição de sequência de segmentos

O objeto tem três parâmetros de identificação externa e um parâmetro de identificação para o simulador. Estes são úteis para o *Master Controller*, exportados pelo simulador e usados na configuração do mesmo.

O Percurso é assim uma estrutura complexa que contém a informação necessária para transformar o *layout* AutoCAD num mundo simulado 3D. Este tem também a responsabilidade de criar a sequência de segmentos e obter a tabela de ligações das agulhagens.

Caso os segmentos tenham sido criados com *polylines*, o início e fim estão intrinsecamente definidos no objeto. Desta forma é possível definir uma sequência de segmentos usando o pseudo-código apresentado no Algoritmo 1. Por outro lado, caso o utilizador tenha desenhado os segmentos com curvas e retas independentes é impossível determinar o início e fim sem ajuda de objetos adicionais. Nesta situação o utilizador tem de inserir a sequência de segmentos manualmente.

4.1.5. UTILITÁRIOS DE PERCURSO

Para converter posições XYZ em posições absolutas do circuito, foi criada uma classe que contém ferramentas para fazer cálculos relacionados com posições e distâncias.

CÁLCULO DE POSIÇÃO

A função de cálculo de posição converte uma qualquer posição XYZ em posição absoluta de fita de posicionamento. O algoritmo usa polígonos anteriormente calculados no interpretador de *layout* para identificar se a posição está dentro de uma determinada entidade, seguindo o pseudo-código do Algoritmo 2.

```
PARA CADA Segmento:
        PARA CADA Entidade
                SE FUNÇÃO (Ponto dentro do polígono da entidade)
                        Devolver Entidade
                FIM SE
        FIM PARA CADA
FIM PARA CADA
```

Algoritmo 2 Procura de entidade para uma determinada posição

O funcionamento do algoritmo está exemplificado na Figura 29. Como é possível observar-se, a posição a calcular encontra-se dentro da entidade cinco, apresentada a verde.

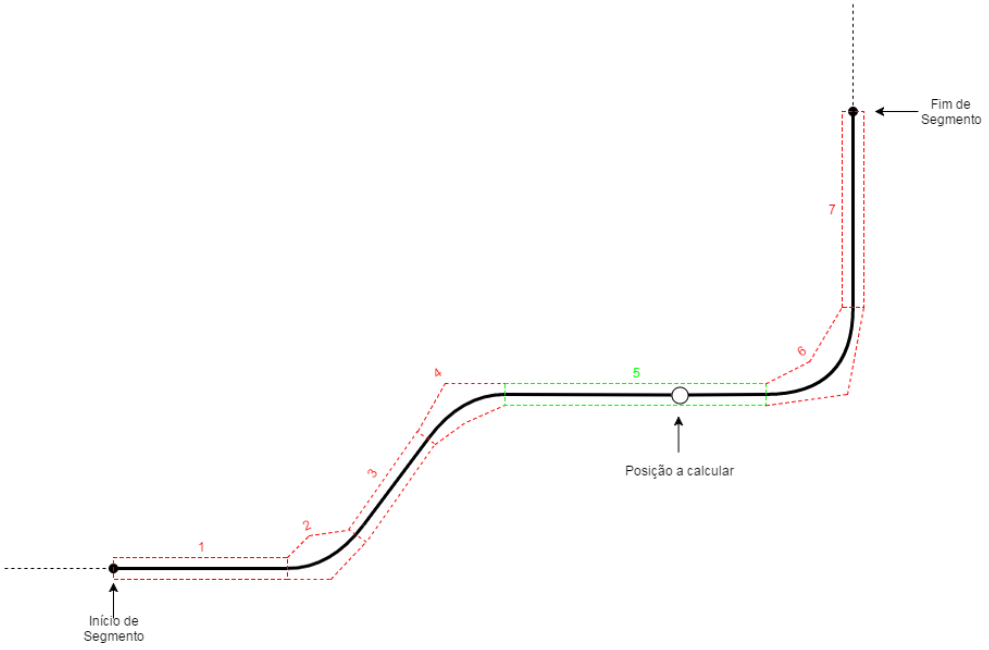


Figura 29 Funcionamento do algoritmo de procura de entidade

Somado ao valor do início de segmento, a posição do ponto a calcular é dada pela soma dos comprimentos das entidades um a quatro, somada à posição do ponto na entidade cinco.

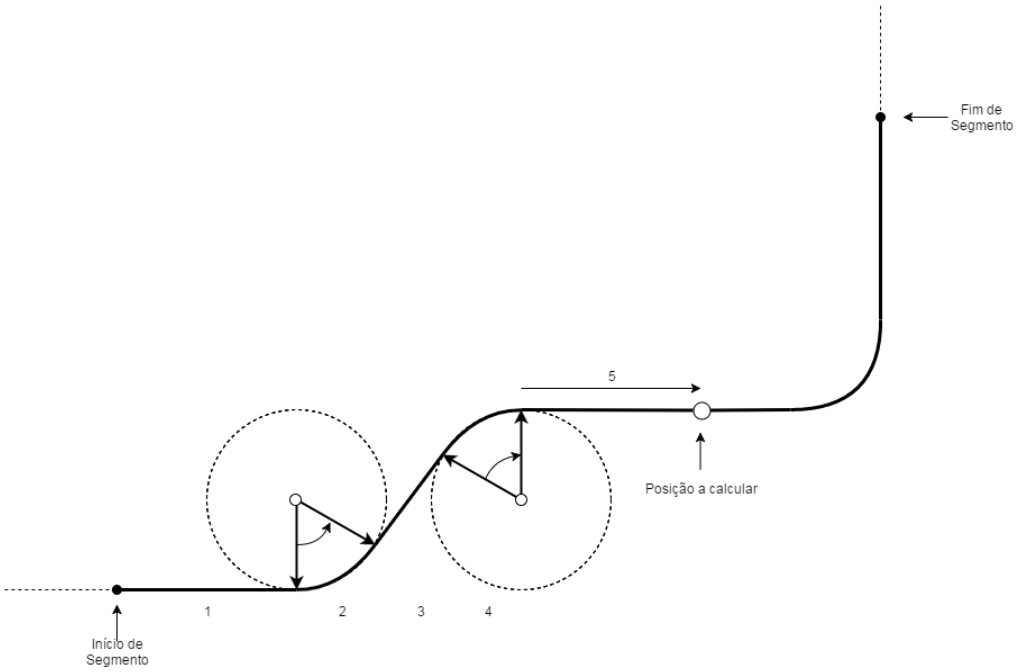


Figura 30 Cálculo de Posição dentro de um segmento

O cálculo da posição é feito seguindo o pseudo-código apresentado no Algoritmo 3. Como se pode observar, para calcular uma posição, é chamada a função de procura de entidade, calculada a posição dentro da mesma, seja reta ou curva e somado o comprimento de todas as entidades para trás. Por fim é somado o valor de início de segmento, obtendo assim a posição absoluta no circuito.

```
Posição         = 0
Entidade        = FUNÇÃO (Procura de entidade por ponto)
SE Entidade >= 0:
        SE Entidade é do tipo Curva:
                FUNÇÃO (Cálculo de posição em curva)
        SENÃO
                FUNÇÃO (Cálculo de posição em reta)
        FIM SE

        PARA tEntidade = 0 ATÉ tEntidade = Entidade:
                Posição += tEntidade.Comprimento
        FIM PARA

        Posição += Segmento.Início
FIM SE
```

Algoritmo 3 Cálculo de Posição

4.2. COMUNICAÇÃO

O módulo de comunicação é responsável por realizar todas as tarefas de diálogo com o PLC, dentro das quais ler e escrever em zonas de memória, criar e cessar a ligação e implementar tudo isto de uma forma cíclica não bloqueando no entanto o ciclo normal da simulação.

A função de comunicação é executada em paralelo ao código da simulação, numa *thread* temporizada que é executada ciclicamente com um intervalo configurável, normalmente entre 50 a 200 milissegundos.

4.2.1. LIBNODAVE

Para realizar o diálogo diretamente com o PLC, o módulo utiliza a biblioteca *libnodave*. Esta é uma biblioteca criada em C++ para processadores de arquitetura 32 bit. O estado final da biblioteca compilada é um ficheiro *Dynamic-Link Library* (DLL) que contém todas as funções necessárias para a sua utilização.

O Python é capaz de carregar funções em DLLs, formando um *wrapper* à biblioteca original, utilizando a biblioteca *CTypes*.

O *libnodave* é responsável por criar um *socket* de comunicação com o PLC e criar uma abstração ao utilizador do protocolo utilizado pelos equipamentos da Siemens. Este recebe ordens e tradu-las em comandos para o PLC utilizando a mnemónica específica do *S7 Protocol*.

Das funções disponíveis na biblioteca destacam-se:

- Criar e cessar ligação a um PLC;
- Colocar PLC em STOP ou RUN;
- Ler e escrever *bytes* em zonas de memória do PLC.

Tudo isto é realizado de forma independente da *interface* de ligação ao PLC. Estas podem ser:

- *Multi-Point Interface* (MPI) – protocolo usado pelos PLCs mais antigos da Siemens, da série S7-200. Também compatível com série S7-300;
- *ISO-on-TCP*;

- *S7 Online* – protocolo usado pelos programas informáticos de programação de PLCs para verificar o funcionamento em tempo real das funções desenvolvidas.

O simulador utiliza como *interface* uma ligação *ISO-on-TCP* para recriar a comunicação entre os veículos e o *Master Controller*.

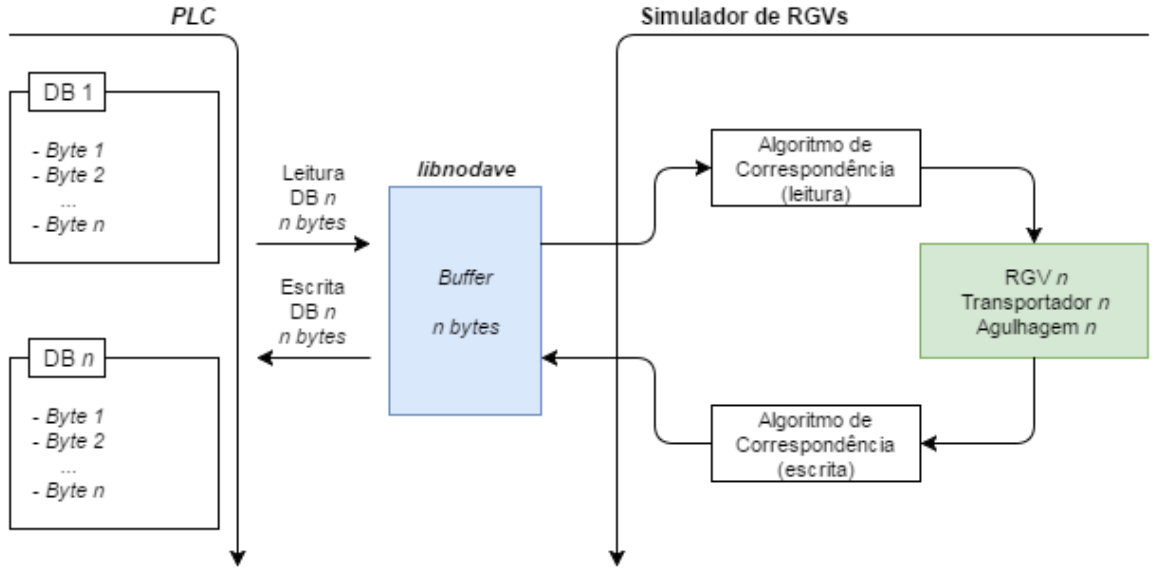


Figura 31 Leitura e Escrita de variáveis usando a biblioteca *libnodave*

O funcionamento da leitura e escrita de variáveis utilizando o *libnodave* está representado na Figura 31. A biblioteca cria um *buffer* de memória que representa um bloco de dados para leitura ou escrita na zona de memória do PLC.

4.2.2. MEMÓRIA E MAPEAMENTO

Depois de uma operação de leitura, o *libnodave* disponibiliza o bloco de *bytes* lido através do buffer de memória. Este é então entregue a um algoritmo de correspondência que trata de fazer a ligação entre o endereço absoluto e o endereço simbólico.

As variáveis, na zona de memória do PLC, estão organizadas em tipos elementares que são usados posteriormente para criar estruturas complexas. Os tipos estão apresentados na Tabela 5, juntamente com o tamanho em *bits* e *bytes*.

Tipos de Variáveis

Tamanho (*bits*)	Tamanho (*bytes*)	Variáveis		
1	-	BOOL		
8	1	BYTE	CHAR	
16	2	WORD	INT	
32	4	DWORD	DINT	REAL

Tabela 5 Comunicação: Tipos de variáveis e tamanho

Cada variável possui um endereço de memória absoluto e outro simbólico. O endereço absoluto é utilizado pelo PLC para mapear a variável na sua memória, enquanto o simbólico é utilizado pelo programador, para facilitar o desenvolvimento de programas.

O objetivo do módulo de comunicação é obter os valores corretos e liga-los às variáveis certas. Para isso foi criado um algoritmo de correspondência que faz a ligação entre o *buffer* de memória lido do PLC e a variáveis em Python.

A Figura 32 apresenta um exemplo de leitura de uma base de dados de memória de um PLC. Como se pode observar, a DB lida está duplicada do lado do simulador, com as mesmas variáveis.

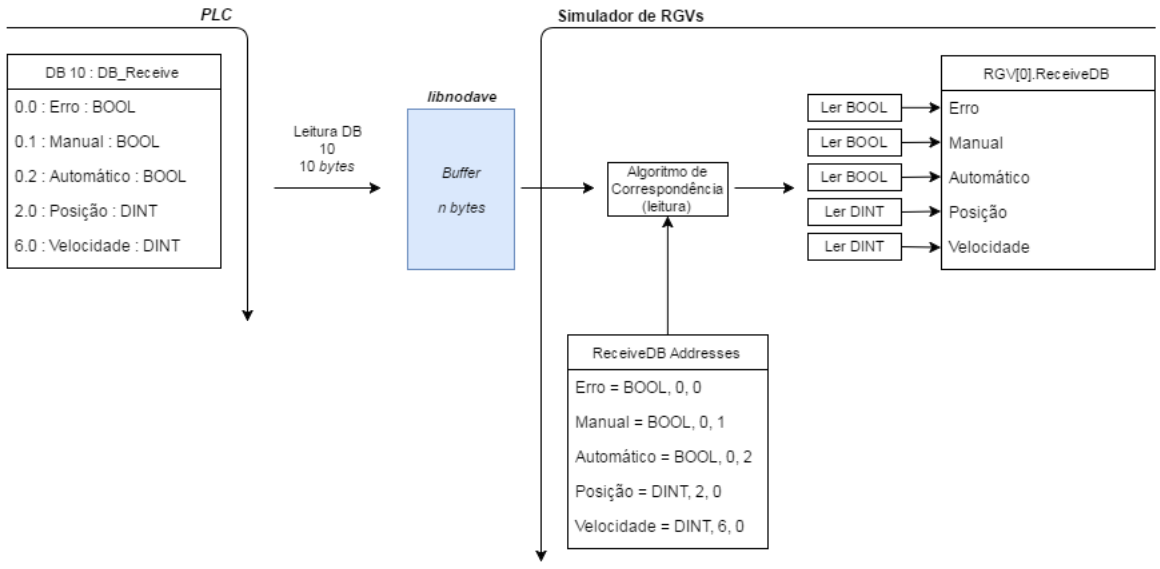


Figura 32 Leitura e Mapeamento de Variáveis

O bloco de memória lido é guardado no *buffer* do *libnodave* e posteriormente entregue ao Algoritmo de Correspondência. Este, de acordo com as variáveis a preencher, e com

a ajuda da respetiva DB de endereços, que contém informação sobre o tipo de variável e endereço, transfere a quantidade de informação necessária do *buffer* para a variável em questão.

4.2.3. OPERAÇÃO

Uma Operação é uma objeto que traduz a intenção de executar uma tarefa de escrita ou leitura de forma cíclica ou acíclica. O objeto, apresentado na Figura 33, é utilizado pelo ciclo de comunicação, que coordena a execução das operações de acordo com o tempo configurado das mesmas.

O tipo da Operação pode ser de leitura, escrita ou ambas. Pode ser cíclica, configurando o campo de "Base de Tempo" ou acíclica, atuando os *bits* de "Ler" e "Escrever".

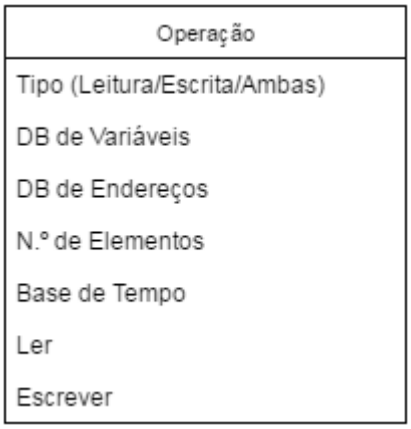


Figura 33 Operação: Conceito de objeto

As Operações estão preparadas para ler vetores, como todos os RGVs presentes na instalação, pelo que têm também um campo para indicar o número de elementos a ler.

4.2.4. CICLO DE COMUNICAÇÃO

As Operações são entregues ao "Ciclo de Comunicação", um algoritmo que coordena todo o processo que é executado em paralelo com o código da simulação de forma cíclica, com uma determinada frequência configurável.

O processo de coordenação da execução das operações está apresentado no pseudo-código presente no Algoritmo 4.

O ciclo executa a contagem de tempo para cada operação e, quando essa é superior à base de tempo definida na mesma, ordena uma operação de escrita ou leitura, de acordo com o tipo definido.

```
PARA CADA Operação:
        Operação.ContadorTempo += Tempo desde a última operação
        SE Operação.ContadorTempo >= Operação.BaseTempo:
                Operação.Ler = (Tipo == OperaçãoLER OU Tipo == OperaçãoLERESCREVER)
                Operação.Escrever = (Tipo == OperaçãoEscrever OU Tipo == OperaçãoLERESCREVER)
        FIM SE
        SE Operação.Ler OU Operação.Escrever:
                Operação.ContadorTempo = 0
                SE Operação.LER:
                        Buffer = PLC.LER(Operação.DB_Endereços)
                        FUNÇÃO (Algoritmo de Correspondência Leitura)
                                (Buffer)
                                (Operação.DB_Endereços)
                                (Operação.DB_Variáveis)
                                (Operação.N.º_Endereços)
                FIM SE
                SE Operação.ESCREVER:
                        FUNÇÃO (Algoritmo de Correspondência Escrita)
                                (Buffer)
                                (Operação.DB_Endereços)
                                (Operação.DB_Variáveis)
                                (Operação.N.º_Endereços)
                        PLC.ESCREVER(Operação.DB_Endereços)
                FIM SE
                SE Erro na Leitura ou Escrita:
                        FUNÇÃO (Reiniciar ligação, n tentativas)
                        SE NÃO PLC.Ligado:
                                Sair do Ciclo de Comunicação
                        FIM SE
                FIM SE
FIM PARA CADA
```

Algoritmo 4 Ciclo de Comunicação

Nas Operações de leitura, primeiro é lido o bloco de dados do PLC, guardado no *buffer* e posteriormente entregue ao Algoritmo de Correspondência que, juntamente com a DB de endereços presente na Operação, escreve na DB de variáveis correspondente.

Já nas Operações de escrita, primeiro é executado o Algoritmo de Correspondência, que coloca o valor das variáveis no *buffer* que posteriormente é enviado para o PLC.

O ciclo de comunicação é também responsável por detetar se existiu algum erro na leitura ou escrita e executar um determinado número de tentativas configurável de reconexão.

4.3. NÚCLEO DE SIMULAÇÃO

O núcleo da simulação é um programa complexo que gere todos os objetos presentes no mundo 3D. Para simplificar e organizar a implementação, a tarefa de gerir objetos específicos estão delegadas a gestores dedicados às mesmas, como representado na Figura 34.

Dada a complexidade do sistema, existem vários parâmetros que necessitam de ser ajustados conforme a instalação. Para isso existem os ficheiros de configuração de entrada no sistema, sendo que a configuração geral contém o endereço de todos os outros ficheiros de configuração.

4.3.1. INICIALIZAÇÕES DO MUNDO 3D

A função de inicializações trata de criar todos os modelos 3D de acordo com o *layout* previamente carregado. Para isso, a função começa por desenhar o carril, utilizando a sequência de vértices calculada pelo Interpretador de *Layout*. De seguida desenha todos os pontos de posicionamento sobre o carril.

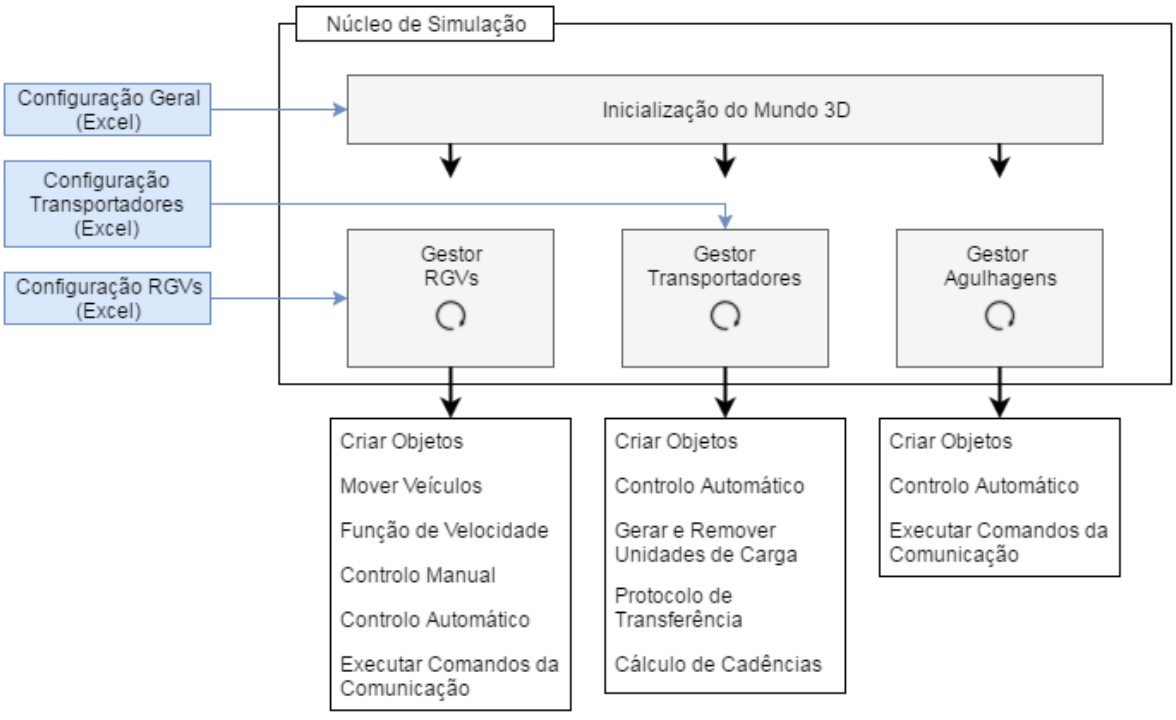


Figura 34 Simulador: Núcleo de Simulação

Posteriormente, começa por criar o gestor de agulhagens, desenhando o carril do segmento reta e curvo. O gestor de transportadores é iniciado de seguida, inserindo no

mundo, nas posições XYZ, objetos do tipo transportador, com a orientação designada no *layout* e de acordo com os parâmetros definidos na configuração de transportadores.

Por fim é iniciado o gestor de RGVs, inserindo no mundo, nas posições definidas no *layout* AutoCAD, veículos de acordo com a orientação do carril e com os critérios definidos no ficheiro de configuração de RGVs.

4.3.2. MODELOS 3D

Para representar os equipamentos da instalação foram criados modelos 3D que possuem algumas das características das máquinas reais.

Foram criados modelos para os veículos, transportadores, unidade de carga e pontos de posicionamento.

RGV – VEÍCULO

O veículo modelado, presente na Figura 35, procura ser uma representação simplificada da máquina representada na Figura 5. Em aplicações 3D, para se obter uma maior *performance*, devem-se construir modelos com um reduzido número de vértices.

Com as mesmas dimensões do veículo real, o RGV modelado representa assim algumas das características mais relevantes da máquina real, dentro das quais:

- Quatro rodas, representadas por quatro cubos;
- Um transportador, representado por um bloco de cilindros;
- Dois sensores que representam a presença de unidade de carga;
- Um indicador de estado, representado por um cilindro que altera a cor de acordo com o estado do veículo;
- Um texto dinâmico que representa o número de identificação do RGV na instalação.

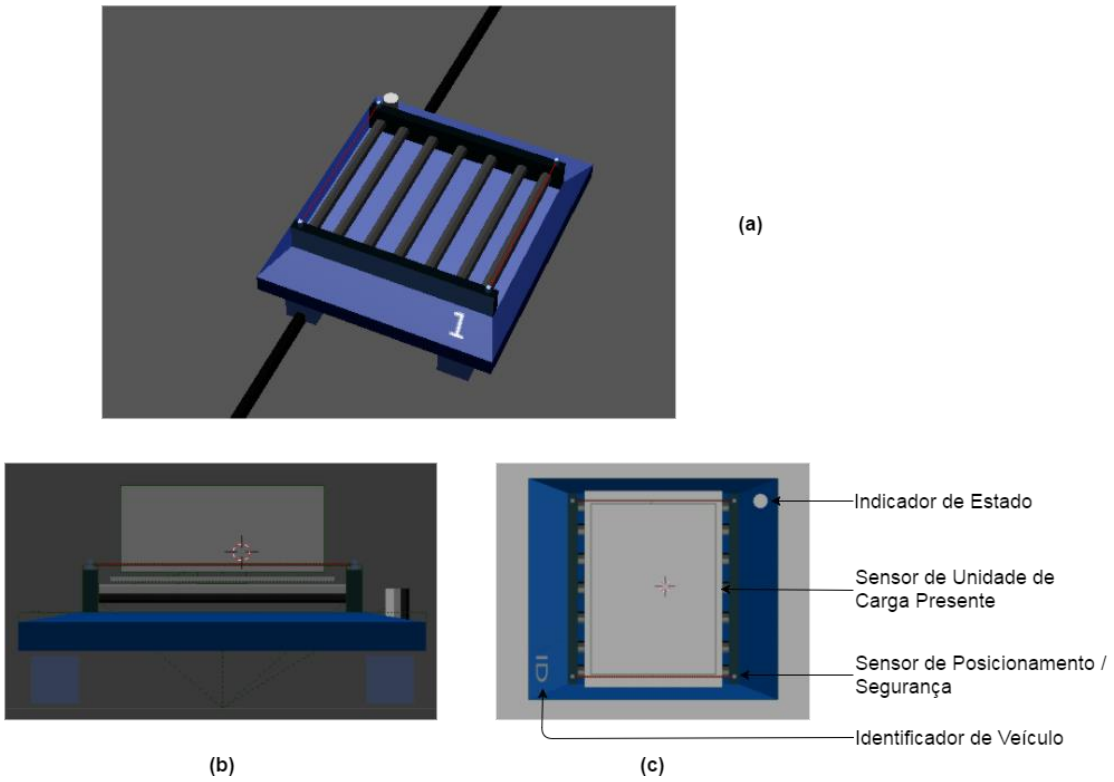


Figura 35 Simulador: Modelo de Veículo RGV (vistas gerais)

TRANSPORTADOR DE INTERFACE

O equipamento transportador é representado pelo objeto modelado na Figura 36. Este é constituído por:

- Dois sensores de posicionamento, que também atuam como segurança, representados por dois cilindros de baixo diâmetro;
- Um sensor de unidade de carga presente.

Este é um objeto genérico usado para todos os tipos de transportadores presentes na instalação.

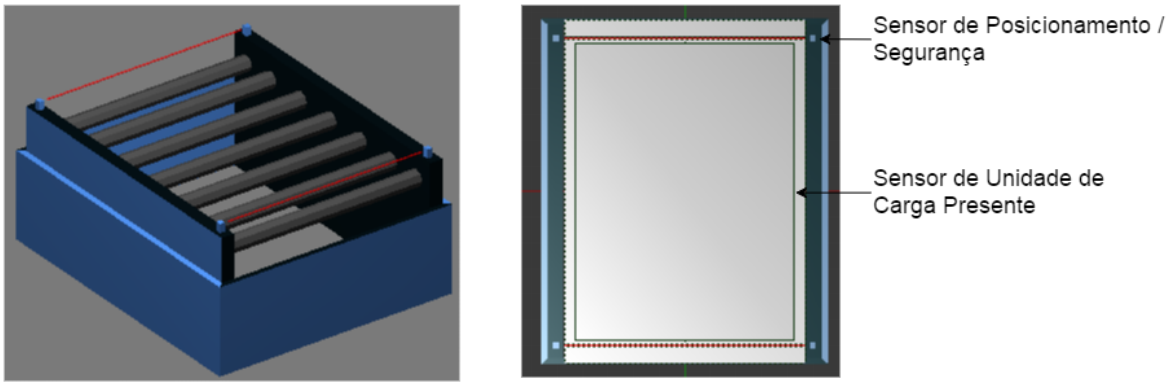


Figura 36 Simulador: Modelo de Transportador

UNIDADE DE CARGA

Uma unidade de carga é representada por um modelo de uma euro palete e um bloco verde que simboliza o produto transportado, tal como apresentado na Figura 37.

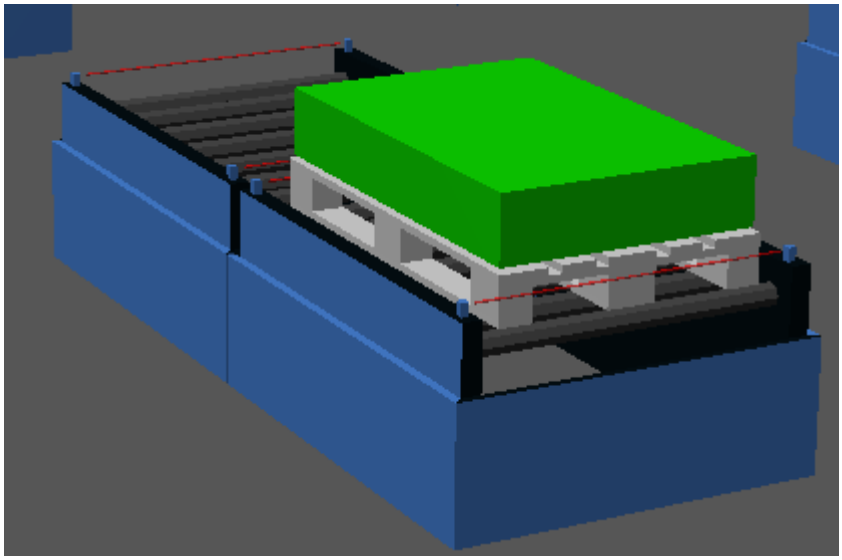
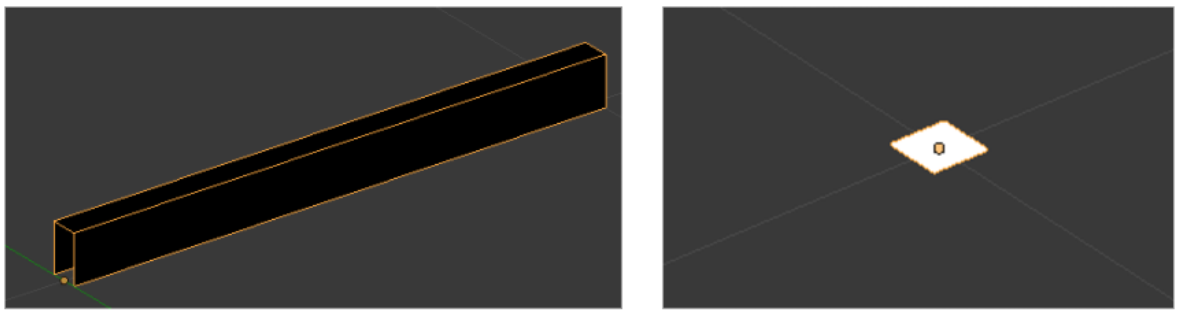


Figura 37 Simulador: Modelo de Unidade de Carga

CARRIL E PONTOS DE POSICIONAMENTO

O carril é retratado por um paralelepípedo, presente na Figura 38 (a), com uma unidade de comprimento que posteriormente é manipulado em escala e orientação para seguir a sequência de vértices definida no Percurso.



(a) (b)

Figura 38 Simulador: Modelo de Carril e Ponto de Posicionamento

Os pontos de posicionamento, Figura 38 (b), são simbolizados por um simples quadrado que assenta no objeto de carril.

AGULHAGEM

A agulhagem é, sem dúvida, o equipamento que menos se assemelha à realidade. Para a representar de forma precisa teriam de se apagar secções de carril de acordo com o estado da agulhagem.

A abordagem adotada simplifica o processo, representando a agulhagem por um carril com maior largura e menor altura, como presente na Figura 39. Como se pode observar, as agulhagens presentes na figura, podem estar em modo reta (a), transição (b) ou curva (c).

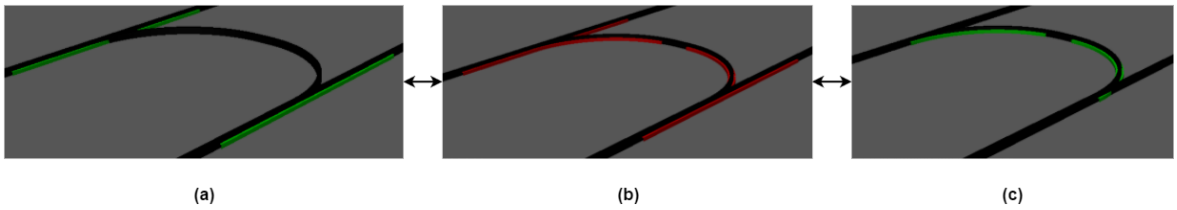


Figura 39 Simulador: Modelo de Agulhagem

4.3.3. GESTOR DE RGVS

Tal como referido anteriormente, cada gestor do núcleo de simulação tem como tarefa gerir os seus objetos específicos. O gestor de RGVs tem como função reproduzir o comportamento de um veículo real quando recebe comandos do *Master Controller*, controlar cada veículo individualmente em modo manual e orientar todos os objetos constituintes dos mesmos.

CONFIGURAÇÕES

O gestor de RGVs tem como configurações:

- Roda Castor: identifica qual a roda assenta sob o carril: i.e. em Inglaterra é frontal, esquerda, em Portugal é frontal, direita;
- Velocidade do transportador e velocidade máxima de translação em modo Manual;
- Dinâmica do veículo: velocidade máxima de translação, rampas de aceleração e desaceleração;
- Zonas de redução: zonas do circuito em que o veículo está limitado em velocidade.

CONTROLO DO VEÍCULO

O controlo dos veículos é feito no gestor dos RGVs e segue o fluxograma presente na Figura 40.

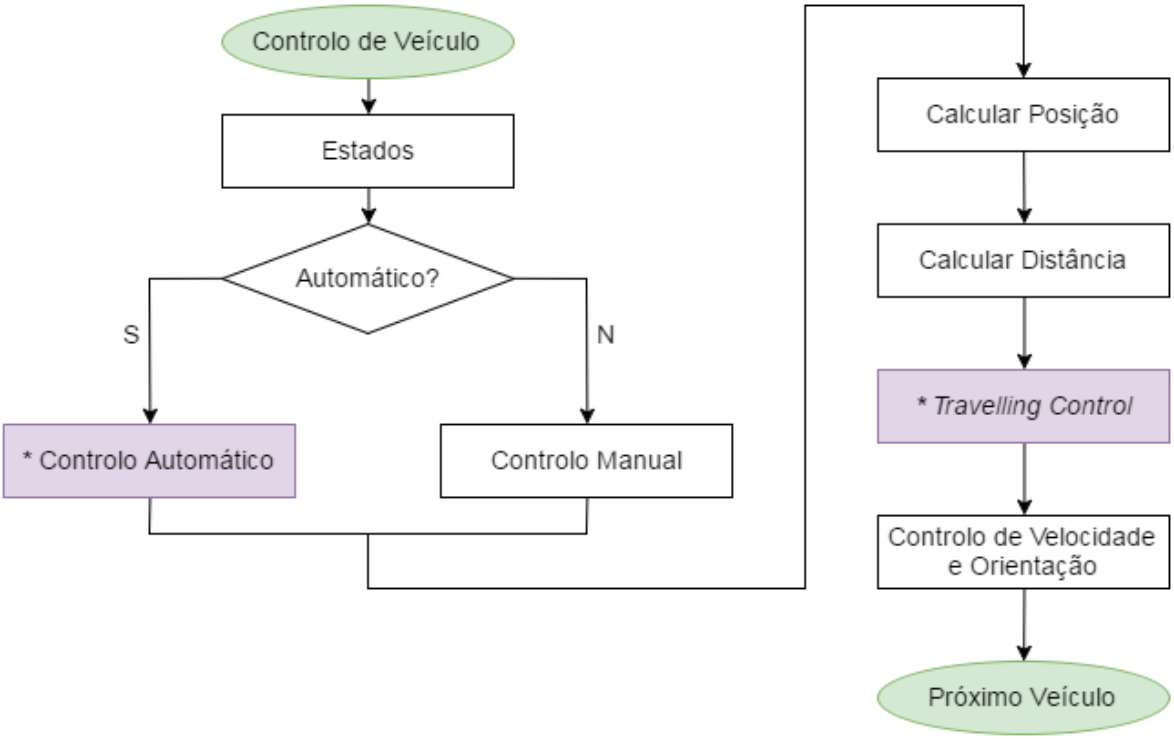


Figura 40 Simulador: Fluxograma de Controlo de Veículo

O fluxograma apresentado é executado para cada veículo, em cada ciclo do programa. A função começa por definir os estados do veículo. Aqui, a função atualiza o estado dos sensores presentes no RGV para determinar se o este possui unidade de carga.

Posteriormente é realizado o controlo automático, que é uma cópia integral do programa presente num RGV real, de modo a imitar o comportamento real da máquina.

CONTROLO MANUAL

O controlo manual de cada veículo é feito através das teclas de direção do teclado do computador. Selecionando o RGV a controlar e abrindo a janela de controlo da *User Interface* (UI), presente na Figura 41:

- CIMA: Velocidade máxima translação (manual)
- BAIXO: Velocidade máxima translação (manual) (invertida)
- DIREITA: Velocidade máxima transportador (manual)

- EQUERDA: Velocidade máxima transportador (manual) (invertida)

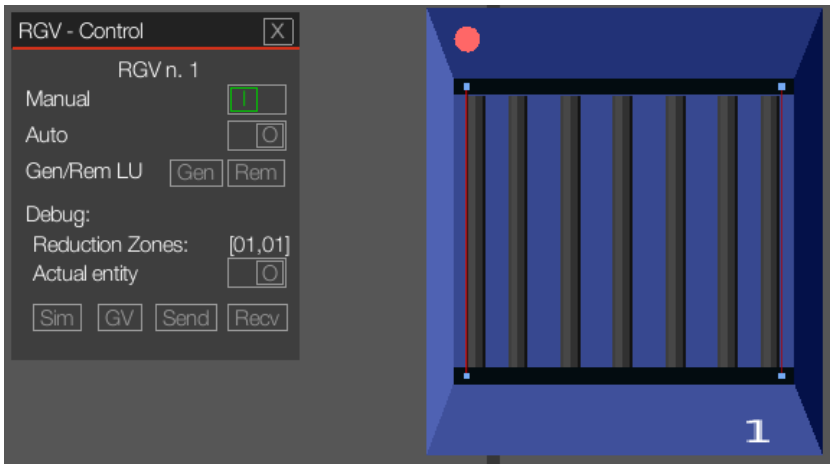


Figura 41 Simulador: Controlo Manual de Veículo

CÁLCULO DE VELOCIDADE E ORIENTAÇÃO

Como referido anteriormente, um RGV é constituído por duas rodas que assentam no carril, duas que seguem o movimento do veículo e um corpo que normalmente tem um transportador.

Cada RGV possui quatro objetos do tipo Roda, que contêm as variáveis presentes na tabela da Figura 42. Como se pode observar na imagem, o RGV está a seguir a sequência de vértices que representa o carril.

As rodas do RGV, numeradas de 1 a 4, estão a ser orientadas de acordo com a sequência de vértices.

- A roda 1 tem orientação igual ao ângulo da semirreta 2-3
- A roda 3 copia a orientação da roda 1
- A roda 2 tem orientação igual ao ângulo da reta 1-2
- A roda 4 copia a orientação da roda 2

O RGV sabe qual o vértice que tem de usar para se orientar por possuir memória do último vértice de destino. O vértice de destino é atualizado assim que a distância ao mesmo for menor que um determinado parâmetro. Nessa altura, dependendo do sentido de movimentação do veículo, o vértice seguinte será o próximo ou o anterior.

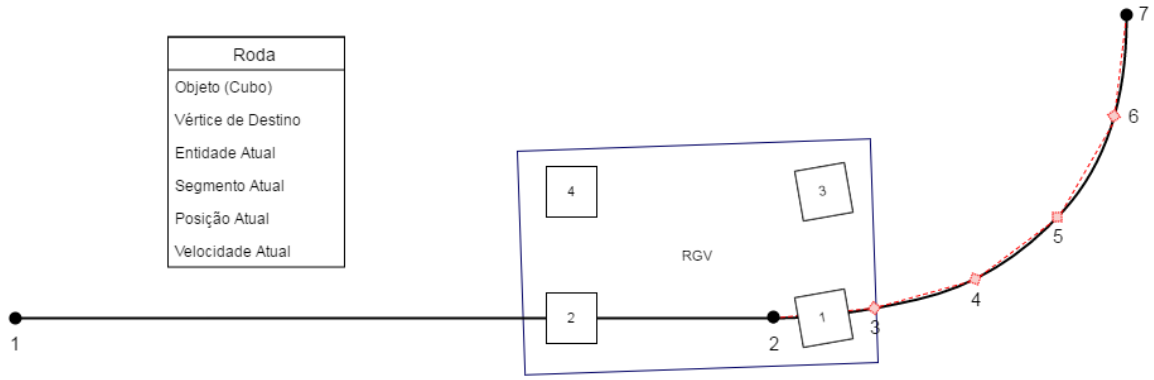


Figura 42 Simulador: Funcionamento da Função de Velocidade e Orientação

Em pontos de decisão, como fim de segmentos ou agulhagens, o simulador verifica o estado das mesmas para decidir qual o vértice a seguir. A Figura 43 apresenta um fluxograma que explica o funcionamento da orientação de uma roda ao próximo vértice.

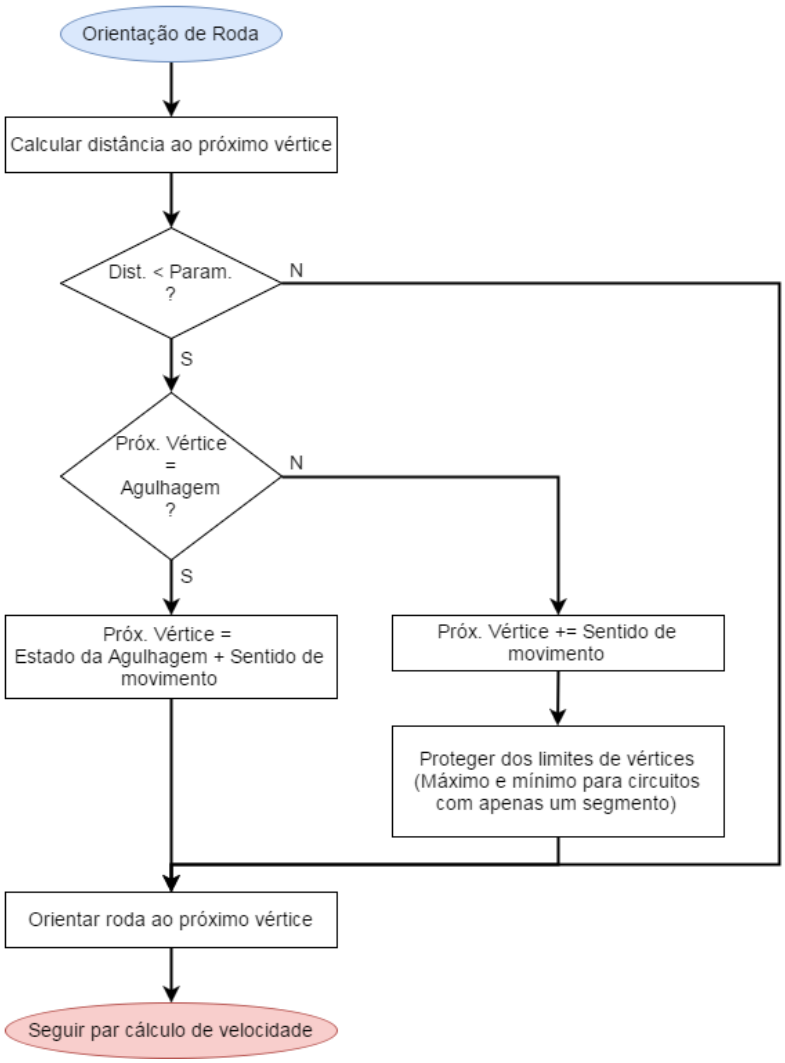


Figura 43 Simulador: Fluxograma de cálculo de orientação para uma roda

Como apresentado na Figura 44, a função de cálculo de velocidade e orientação traduz uma velocidade absoluta, em milímetros por segundo, em velocidade XY para os objetos "roda" de um determinado RGV. De acordo com a sequência de vértices do circuito, a função também orienta as rodas, fazendo com que estas sigam o carril.

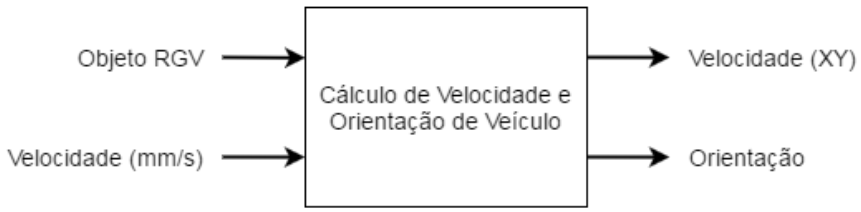


Figura 44 Simulador: Função de Cálculo de Velocidade e Orientação

Na Figura 45 é possível observar o processo de cálculo de velocidade e todas as funções envolvidas. Destas, apenas as que se encontram na área "Simulador" foram desenvolvidas, as restantes pertencem ao sistema RGV e estão traduzidas de forma a plagiar o seu comportamento real.

O Controlo de Translação é a função responsável por controlar a velocidade do RGV em modo Automático, desde a posição de início de movimento até à posição final. Depois de calculada a velocidade a que o RGV se deveria movimentar, em modo Automático, existem várias funções que a limitam, protegendo-o de zonas críticas que, à velocidade máxima, poderiam danificar o equipamento.

Zonas como curvas e agulhagens são secções do circuito a que o RGV necessita de se mover a uma velocidade inferior. Estas são designadas por Zonas de Redução e estão definidas em cada RGV.

De acordo com a sequência de vértices, as rodas do RGV devem orientar-se de acordo com o vértice seguinte. O cálculo de orientação e de velocidade é feito para cada uma das rodas pois, sendo independentes, estão em diferentes secções da sequência de vértices. Esta tarefa é executada na função de cálculo de velocidade e orientação. Posteriormente é executada uma verificação para determinar se o RGV se encontra dentro de uma curva, entidades que possuem maior densidade de vértices. Aqui a velocidade deve ser limitada pois o simulador faz mais cálculos por segundo, devido à constante orientação das "rodas".

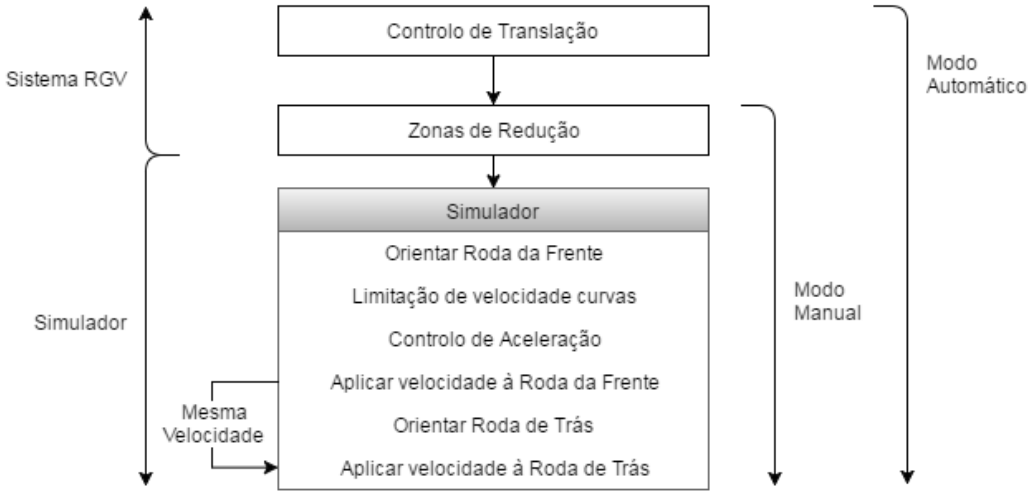


Figura 45 Simulador: Processo de Controlo de Velocidade e Orientação

De seguida é feito um controlo de aceleração que limita a aceleração do veículo. Isto é particularmente útil para o controlo manual, em que é definida uma velocidade máxima assim que é pressionada uma das teclas de direção. Esta função limita assim a velocidade, calculando uma rampa com aceleração parametrizada no ficheiro de configuração de RGVs.

Tendo a velocidade e orientação final calculada, o simulador calcula então a velocidade a aplicar ao objeto, nos eixos X e Y de acordo com as seguintes equações (2) e (3).

$$Velocidade\ X = \cos(\theta) \times Velocidade\ (mm/s) \quad (2)$$

$$Velocidade\ X = cos(\theta) \times Velocidade\ (mm/s) \quad (3)$$

4.3.4. GESTOR DE TRANSPORTADORES

O gestor de transportadores é o responsável por gerir todos os transportadores da instalação, gerar e remover todas as unidades de carga, calcular e gerar um relatório da cadência geral e de cada fluxo existente e gerar um ficheiro Excel com a posição de todos os pontos de posicionamento, pontos de *interface*, valores calculados automaticamente para cada segmento.

Embora pareça complexo à partida, este gestor está construído de forma a ser modular e o mais genérico possível, tornando-o flexível e configurável para qualquer instalação.

CONFIGURAÇÕES

Os transportadores de *interface* são, sem dúvida, os equipamentos numa instalação de RGVs que mais configurações podem ter, devido ao elevado número de combinações de disposição dos mesmos. Para tornar o simulador genérico e flexível a qualquer tipo de instalação, o gestor de transportadores possui a sua própria folha de configuração, em Excel.

A Figura 46 apresenta uma possível configuração para uma instalação de transportadores de *interface*. As configurações possíveis são:

- Grupos de transportadores;
- Destinos a gerar para transportadores de entrada;
- Fluxos de movimento de unidades de carga;
- Parâmetros especiais.

Os transportadores da instalação estão normalmente divididos em grupos. Transportadores de *interface* com o armazém são normalmente designados por Alas, já os equipamentos que fazem *interface* com os transportadores de saída de armazém são normalmente denominados por Expedição. Pode então definir-se grupo, um conjunto de um ou mais transportadores que fazem *interface* com uma determinada entidade.

Os grupos são estruturas com os seguintes atributos:

- Nome: Identificação do grupo;
- Transportadores: Lista contendo as identificações de cada transportador;
- Destinos (próprio): Lista de valores que definem o destino de uma unidade de carga para um dos próprios transportadores;
- Destinos (gerados): Lista de valores que definem o destino das unidades de carga geradas dentro do grupo.

A *performance* do sistema, além da cadência geral, é determinada pelas sub-cadências de cada fluxo. Um fluxo é um percurso que uma unidade de carga pode habitualmente percorrer. Este tem as seguintes propriedades:

- Nome: Identificação do fluxo;
- Grupo de Origem;

- Grupo de Destino;
- Número de unidades de carga transportadas;
- Cadência;

A cadência de um fluxo, bem como a cadência da instalação geral, é definida pelo quociente entre o número de unidades de carga transportadas por um determinado intervalo de tempo, normalmente uma hora.

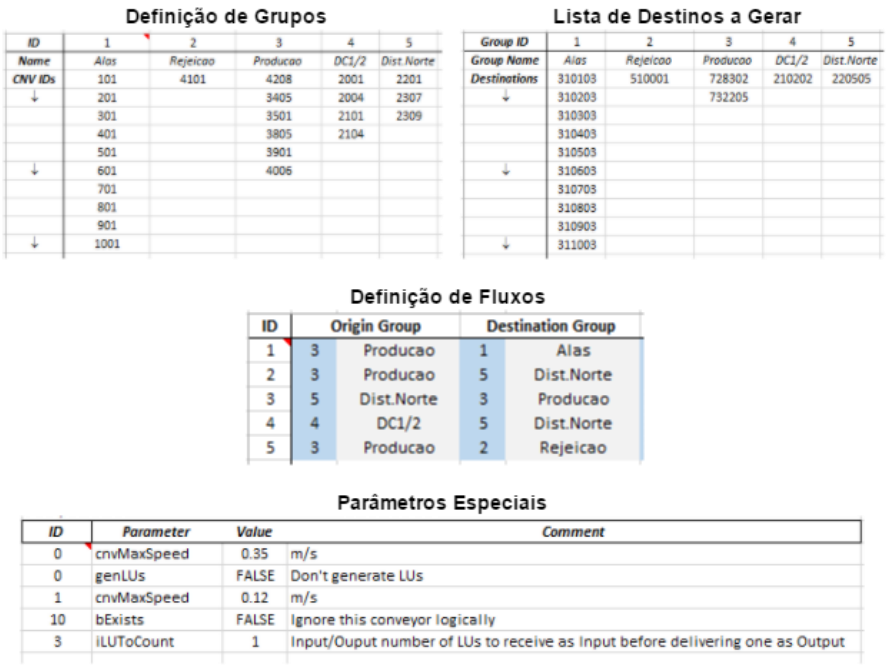


Figura 46 Simulador: Folhas de Configuração dos Transportadores

Nos parâmetros gerais podem definir-se valores predefinidos para um transportador ou para todos. Esta folha é útil para definir velocidades diferentes para cada transportador, ativar a funcionalidade de gerar unidades de carga continuamente, entre outras propriedades.

COMBINAÇÕES DE TRANSPORTADORES

Quando é gerado um transportador de *interface* é necessário criar um ou mais transportadores que crie ou remova unidades de carga ou que atue como simples transportadores de *buffer*.

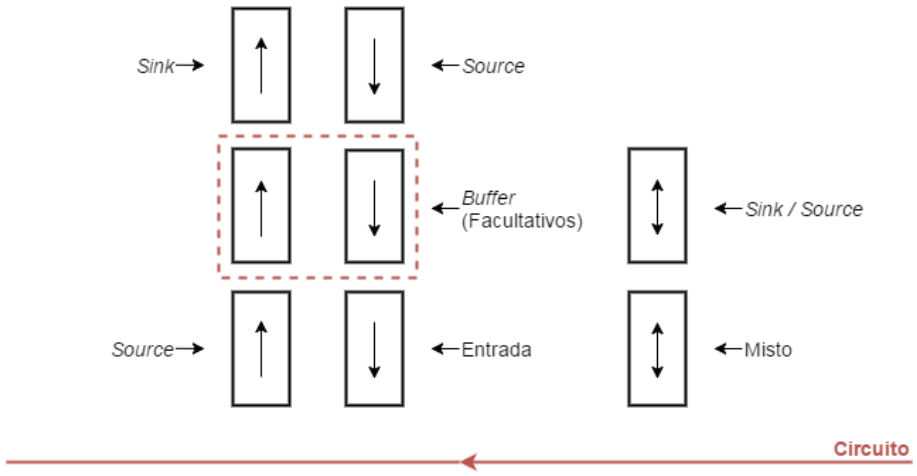


Figura 47 Simulador: Combinações de Transportadores

Como visto anteriormente, os transportadores de *interface* estão divididos em classes: Entrada, Saída ou Mistos. Quando é gerado um transportador de entrada é necessário criar um transportador do tipo *Source*, responsável por gerar unidades de carga. Estes transportadores também atuam como *buffer*, quando têm a *flag* de gerar paletes continuamente ativa.

Já num transportador de saída é necessário criar um transportador do tipo *Sink*, responsável por remover as unidades de carga do mundo 3D. As configurações possíveis de transportadores estão exemplificadas na Figura 47.

Os transportadores mistos, assim como referido anteriormente, atuam como entrada em certas ocasiões e como saída nas restantes. A troca entre estes dois modos é feita quando um número de unidades de carga é entregue ou recebido, configurável.

Os destinos de cada unidade de carga são gerados nos transportadores de saída, assim que a unidade fica pronta para carregar. De acordo com os dados presentes nos transportadores, o *Master Controller* gera uma tarefa que posteriormente atribui a um dos RGVs disponíveis para a executar.

CICLO

O ciclo do Gestor de transportadores segue o fluxograma presente na Figura 48. Como se pode observar, o ciclo executa o cálculo de cadências e de seguida controla cada transportador, atualizando os seus estados e seguindo uma função diferente para cada tipo de transportador.

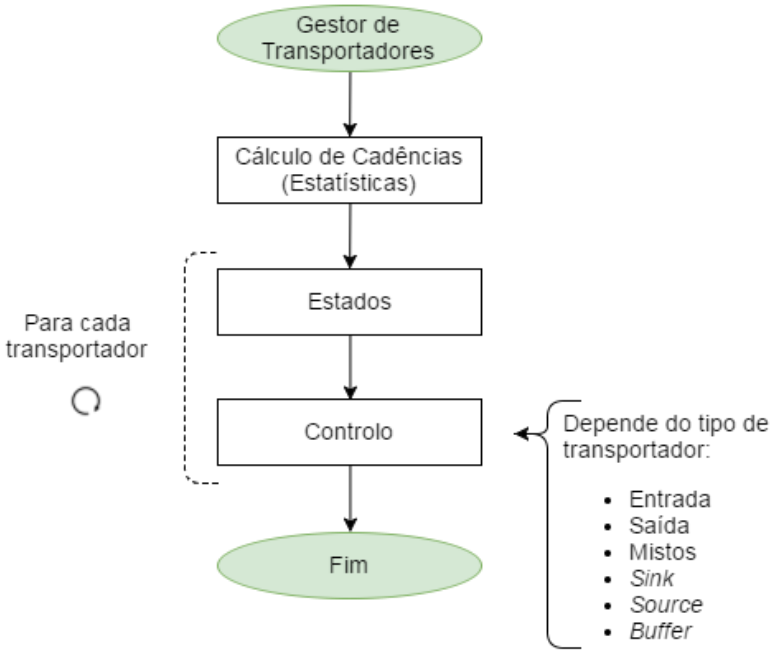


Figura 48 Simulador: Gestor de Transportadores

SAÍDAS

O gestor de transportadores, tal como referido previamente, é também responsável por gerar um ficheiro que contém informação sobre toda a instalação, posteriormente usado pelo especialista para configurar o *Master Controller*, e um registo de cadências que é usado de seguida por uma folha de Excel para criar uma análise pormenorizada dos fluxos de unidades de carga da instalação.

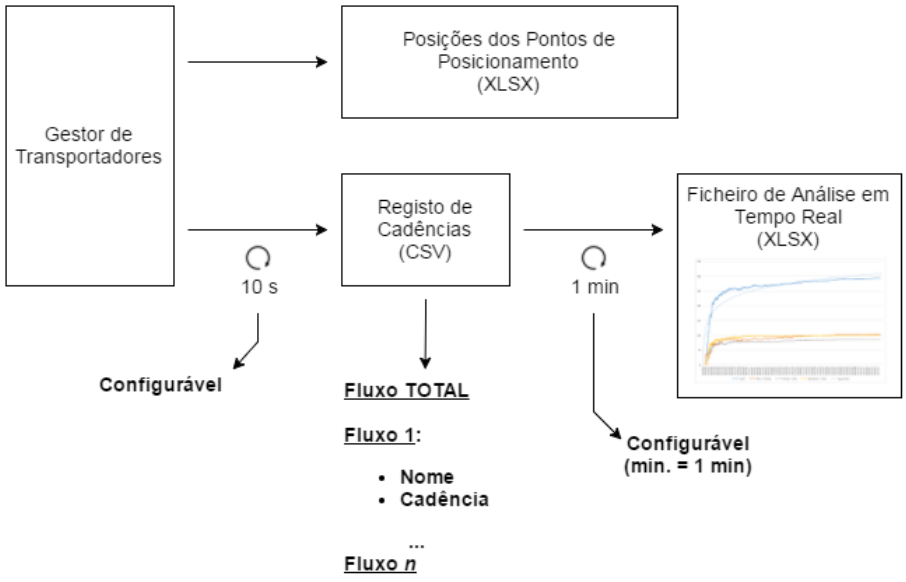


Figura 49 Simulador: Saídas do Gestor de Transportadores

A criação do ficheiro de posições é feita de forma assíncrona, e normalmente só executada uma vez por instalação. Já o registo de cadências consiste num ficheiro *Comma Separated Values* (CSV) que é atualizado de forma cíclica, com um intervalo de tempo configurável, posteriormente importado por uma outra folha Excel, que analisa os dados gravados. A folha de Excel tem a capacidade de atualizar os dados com um intervalo de tempo configurável, com o limite máximo de uma atualização por minuto.

É assim possível gerar relatórios detalhados da *performance* da instalação, em tempo real, dando possibilidade ao especialista de afinar os parâmetros que a influenciam remotamente.

4.3.5. GESTOR DE AGULHAGENS

O gestor das agulhagens é responsável por gerir os respetivos objetos curva e reta que identificam o seu estado atual. As agulhagens podem estar em três estados: Curva, Reta ou Não Definido. O estado intermédio representa o intervalo temporal de mudança de curva para reta ou vice-versa, de uma agulhagem real.

O funcionamento segue o esquema presente na Figura 50. Como se pode observar, o estado intermédio tem um intervalo de tempo de espera de três segundos, configurável.

Figura 50 Simulador: Funcionamento das Agulhagens

Tal como referido acima, a função de cálculo de velocidade e orientação do veículo usa o estado das agulhagens para determinar o segmento e vértice a seguir.

O gestor simboliza assim visualmente o estado da agulhagem, escondendo ou mostrando os objetos curva/reta e alterando a sua cor entre verde, significando que está estabilizada e pronta para utilização, e vermelho, em caso de erro ou mudança de estado.

4.4. *INTERFACE* COM O UTILIZADOR

Apesar da UI do Blender ser muito apelativa e *user-friendly*, não é possível utilizá-la no motor de jogo. Por outro lado, o motor de jogo oferece um *wrapper* à biblioteca OpenGL que permite desenhar formas 2D depois da imagem do motor ser gerada.

Na ausência de uma biblioteca para o efeito, utilizaram-se as ferramentas de desenho 2D para construir uma *interface* com o utilizador. O processo de criar uma plataforma deste tipo é moroso e complexo e é difícil de gerar algo com a mesma perfeição de uma biblioteca desenhada para o efeito. Ainda assim, sendo a única hipótese disponível, partiu-se para o desenvolvimento de raiz de uma *interface* customizada.

O módulo da *interface* foi criado com o objetivo de ser genérico, flexível e que a criação de novas interações fosse de simples execução. Para isso foi desenhada a arquitetura de sistema presente na Figura 51.

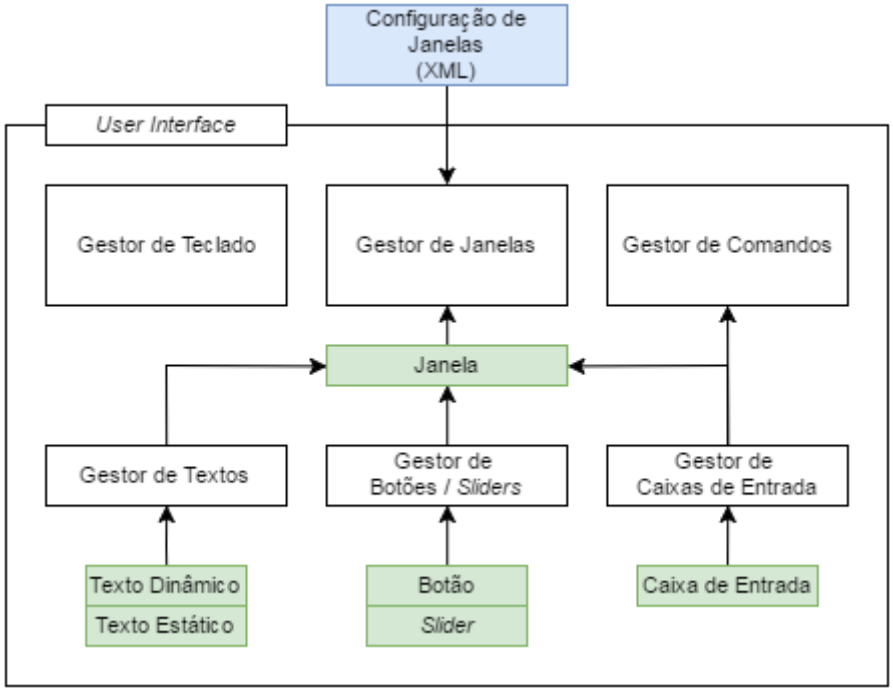


Figura 51 UI: Arquitetura do Módulo

CONFIGURAÇÃO

Como é possível observar, o módulo está dividido em vários gestores que coordenam funções específicas. A configuração de janelas consiste num ficheiro XML que contém a estrutura de cada janela.

- Janela
    - Título;
    - Identificação;
    - Posição XY;
    - Tamanho XY;
    - Botões: Lista de objetos Botão e Seletor;
    - Textos: Lista de objetos Texto;
    - Caixas de Entrada: Lista de objetos Caixa de Entrada;
    - Eventos: Lista de funções a executar quando ocorrem determinados eventos;

Na Figura 52 é possível visualizar uma janela, que contém várias das entidades referidas na configuração.

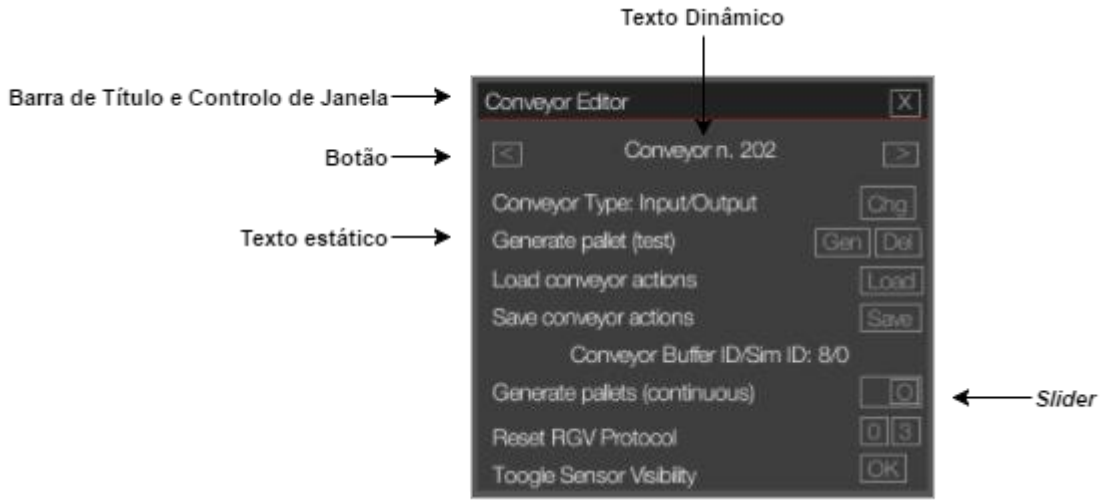


Figura 52 UI: Janela de Exemplo

Como é possível observar, a janela possui uma barra de título e controlo que a identifica, a move e, carregando no botão X, a fecha. Tem um tamanho definido na configuração e uma vasta lista de entidades Texto e de botões, bem como de um *slider*.

GESTOR DE TECLADO

O gestor de teclado é responsável por adquirir todos as entradas feitas a partir do mesmo, e traduzi-las para a *interface*. O gestor é capaz de filtrar as teclas pressionadas, distinguindo os eventos:

- “Tecla pressionada”: detetando o flanco ascendente;
- “Tecla ativa”, detetando o estado atual de uma tecla;
- “Tecla solta”, detetando o flanco descendente.

Estes eventos são úteis para determinadas funcionalidades do sistema, como é o caso do controlo manual do RGV, que atribui velocidade máxima enquanto tem a tecla de direção cima ou baixo ativa.

Todas as classes do sistema podem aceder ao estado do teclado, ficando a observar apenas eventos de teclas do seu interesse.

GESTOR DE JANELAS

O gestor de janelas faz exatamente o que o seu nome indica – gere objetos do tipo Janela. Nele está alojado um vetor de janelas ativas na *interface* e é essencial para determinar qual a janela ativa a cada momento.

O gestor é responsável por executar as funções de cada janela, de controlo e desenho e de proporcionar uma *interface* flexível, podendo arrastar as janelas no ecrã, e *user-friendly*.

As janelas possuem eventos que podem ser usados no programa para carregar elementos de texto ou executar funções específicas. Os eventos disponíveis são:

- *OnLoad*: Função a executar no momento de carregar a janela;
- *OnClose*: Função a executar no momento em que se fecha a janela;
- *OnLoaded*: Função a executar aquando da janela carregada.

GESTORES DE TEXTO, BOTÕES E CAIXAS DE ENTRADA

Tal como referido anteriormente, uma janela pode possuir objetos do tipo Texto, Botão e *Slider* ou Caixa de Entrada. Cada objeto destes é gerido por um gestor próprio que possui um funcionamento conceitual que segue o fluxograma da Figura 53.

Dado que todos os objetos no gestor são idênticos, ou pelo menos do mesmo tipo, é feito o controlo (interação com o sistema) e de seguida efetuado o desenho no ecrã.

O gestor de textos, na mesma linha de ideia do apresentado, gere objetos de texto. Um texto tem como atributos:

- Texto:
    - Posição XY;
    - Variável a apresentar (pode ser texto estático);
    - *Flag* Estático.

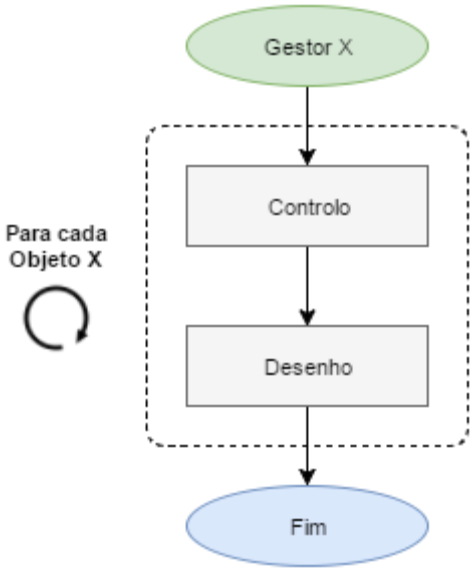


Figura 53 UI: Funcionamento de um gestor de objetos

A posição XY indica o posicionamento do texto dentro da janela. A variável a apresentar indica a variável do sistema que se quer representar e, por último, a *flag* de estático indica se o programa deve atualizar o valor da variável a cada ciclo.

O gestor de botões segue a mesma lógica, com um controlo um pouco diferente. Além de possuir interação com o rato, que indica se está sobre o mesmo ou se foi efetuado um clique, possui também controlo de *slider* que requer a leitura da variável a cada ciclo, antes da interação.

A estrutura de um botão é:

- Botão:
    - Posição XY
    - Texto: texto a desenhar sobre o botão;
    - Função: função a executar aquando de um clique;

    - Argumentos: argumentos da função;

Já um *slider* tem como estrutura:

- *Slider*:
    - Posição XY
    - Classe: estrutura da qual se quer ler/escrever;
    - Variável: variável da estrutura a ler/escrever.

Caixas de entrada são objetos nos quais é possível introduzir texto ou valores numéricos. São úteis para executar comandos ou definir início de segmentos, posição final para veículos, entre outras utilidades. Uma caixa de entrada tem a seguinte estrutura:

- Caixa de Entrada:
    - Posição XY
    - Tamanho XY
    - Função a executar quando se carrega na tecla Enter.

Os objetos deste tipo utilizam as funções do gestor de teclado para executar as suas funções.

GESTOR DE COMANDOS

O gestor de comandos é uma entidade que possui uma caixa de entrada na qual é possível definir valores de parâmetros específicos, dar ordens de mudança a agulhagens e posições finais de movimento a veículos-.

Como o Python é uma linguagem interpretada, é possível executar linhas de código durante a execução do programa. O gestor de comandos tira partido disso mesmo e permite ao utilizador, especializado no sistema, realizar operações de *debug* ou de movimentação em modo *offline*.

Ainda assim, o gestor permite que utilizadores menos especializados, interajam com o sistema. Para isso foi criada uma nomenclatura que permite executar tarefas como rodar uma agulhagem, definir ponto de posicionamento para um ou todos os RGVs e alguns parâmetros específicos. Um comando tem a estrutura apresenta na Figura 54.

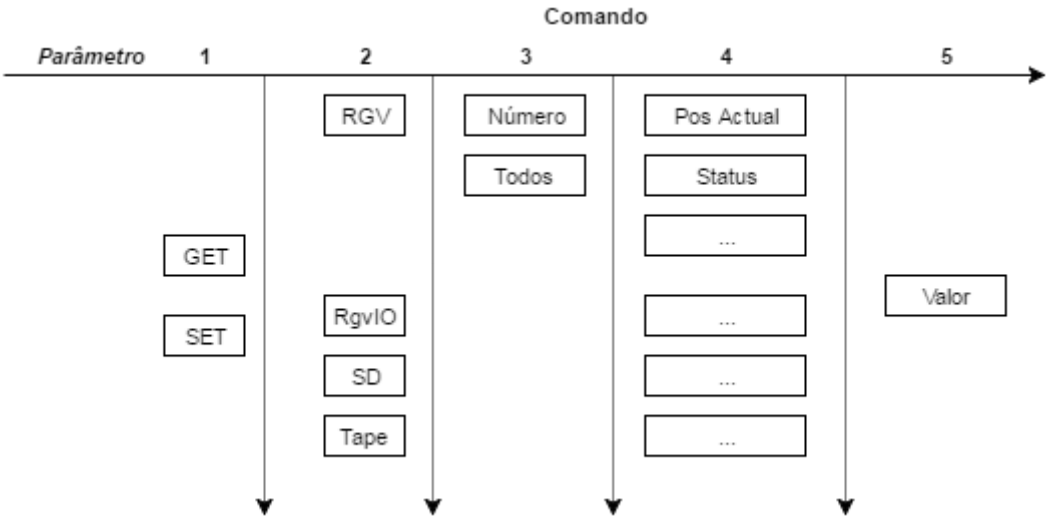


Figura 54 UI: Construção de Comando

A *interface* com o utilizador é assim um módulo simples, flexível e modular que permite a fácil integração de novos blocos de interação. Embora não tenha uma apresentação profissional, as funcionalidades essenciais estão criadas e facilita naturalmente a *interface* com o utilizador que de outra forma seria inexistente.

A Figura 55 apresenta um *screenshot* do produto final, a realizar simulação de um circuito de RGVs.

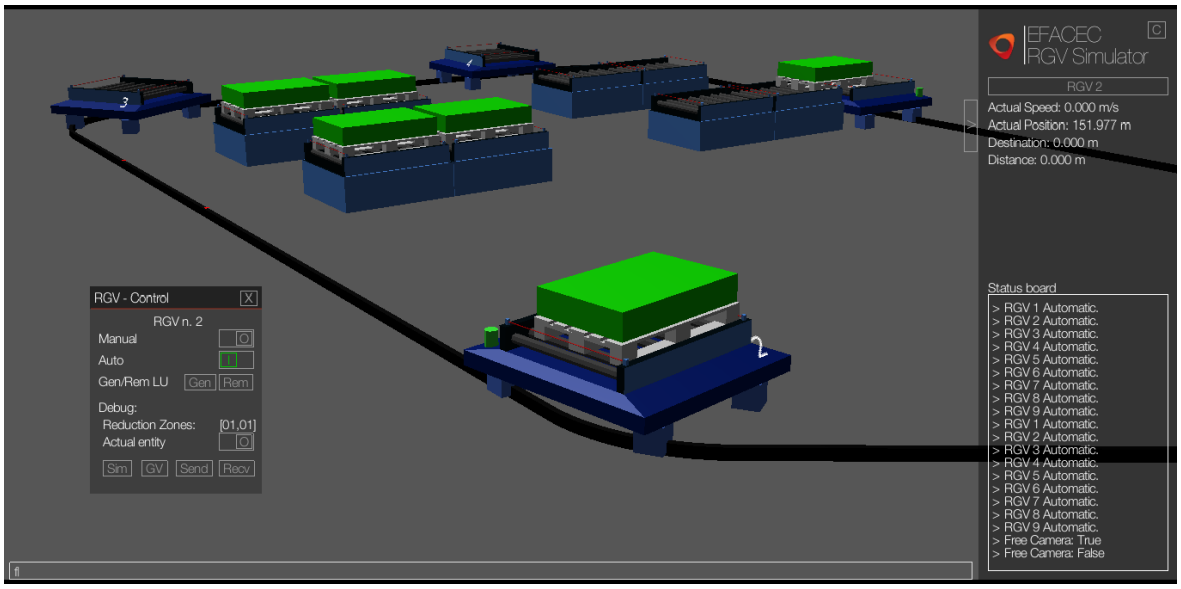


Figura 55 Simulador: *Screenhot* da apresentação final

5. CONCLUSÕES

Neste capítulo são apresentadas as conclusões do projeto e sugestões para trabalhos futuros, com o intuito de o melhorar e ampliar. Estando em constante desenvolvimento há pouco menos de um ano, este tem evoluções diárias que não estão, em parte, refletidas neste documento.

A utilização do Python e do Blender como ferramentas para a execução do projeto foi benéfica, em termos de tempo de desenvolvimento, pois permitem criar protótipos num curto espaço de tempo. Ainda assim, por serem de tão simples utilização, têm muito código que não é específico para o projeto que se está a desenvolver. Por essa razão, para aplicações que requerem altos níveis de *performance*, a utilização destas ferramentas não é recomendável. O Blender foi utilizado por ser a alternativa *open-source* mais barata que, contudo, se mostrou mais fiável do que inicialmente pensado.

O Python, embora seja uma linguagem com uma curva de aprendizagem bastante curta, tem as suas limitações. A ausência de *threads* reais é uma delas, e a mais determinante em muitos casos. Devido à existência do *Global Interpreter Lock* (GIL), as *threads* são

executadas como interrupções num microcontrolador, de forma sequencial, e não em paralelo, como em C ou C++.

Este problema causou contratempos na execução do projeto pois, mesmo colocando o módulo de comunicação numa *thread* separada, a simulação sofria atrasos no processamento lógico. O problema foi contornado, substituindo a *thread* tradicional por uma *thread* temporizada que executa a comunicação de forma cíclica num determinado intervalo de tempo.

Depois do protótipo desenvolvido e com provas dadas de que é útil e fidedigno, o departamento de investigação e desenvolvimento está a equacionar a tradução do projeto para uma plataforma mais eficiente e com melhor *performance*, que é o caso do Unreal Engine, com API em C++. Isto permitirá tirar melhor partido das *threads* e resultará num código mais eficiente.

O simulador criado neste projeto foi e é utilizado pelos especialistas para fazerem *debug* do código complexo e prepararem as instalações antecipadamente e de forma remota. Reduz assim os custos com pessoal deslocado e o tempo de permanência em obra. Este é também utilizado para fazer testes de cadência em fase de projeto, bem como em instalações já em funcionamento, para permitir à EHS fechar os contratos com os clientes, assegurando-os do bom funcionamento dos equipamentos instalados.

A *interface* com o utilizador, apesar de não ter o aspeto nem as funcionalidades de uma *interface* profissional, permite a interação com o utilizador e facilita o *debug* e a configuração de circuitos com RGVs. Em geral, o aspeto do simulador pode e deve ser melhorado, substituindo os modelos 3D aproximados por modelos reais, já que estes são já criados pelo departamento de mecânica.

A criação do simulador permitiu o desenvolvimento de ferramentas que permitem agilizar projetos que requerem comunicação entre plataformas informáticas e controladores lógicos programáveis da Siemens. Estando dividido em módulos, estes são flexíveis o suficiente para se adaptarem a outras aplicações, como é o caso do SCADA automático para circuitos de RGVs.

Este projeto consiste numa aplicação informática que desenha o circuito, os veículos e os transportadores de *interface* de acordo com os respetivos estados e permite a interação com o circuito, possibilitando o envio de comandos. O projeto é auxiliado pelos módulos de comunicação que coordena a leitura e envio de dados para o PLC, e interpretação de *layout*, que traduz o ficheiro AutoCAD num objeto Percurso, de forma a poder desenhar-se os equipamentos nas posições corretas. Na Figura 56 está representado um exemplo desta aplicação para um circuito de RGVs.

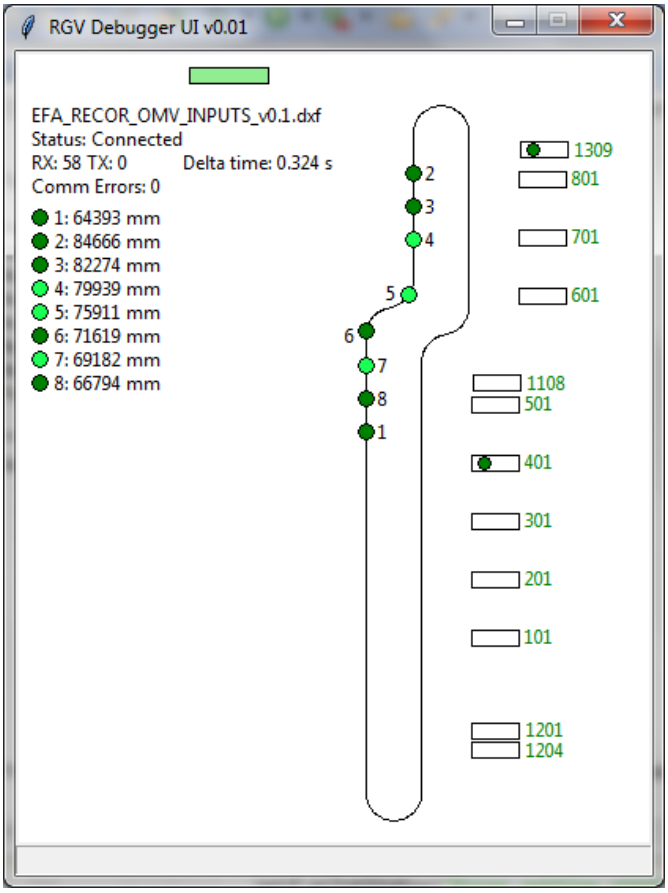


Figura 56 RGV *Debugger*: Aplicação desenvolvida com os módulos do simulador

Em geral, pode afirmar-se que os objetivos da presente Tese foram cumpridos com sucesso. Na criação de sistemas desta dimensão, o mais importante é o teste intensivo pós-desenvolvimento e resolução de diversos *bugs* que podem e vão aparecendo com os testes. O sistema apresentado foi testado por várias pessoas e é utilizado atualmente pelos especialistas de forma a acelerarem o seu trabalho, libertando-os mais rapidamente para abraçar novos projetos.

5.1. DESENVOLVIMENTOS FUTUROS

Sendo um projeto com constantes desenvolvimentos futuros, ter uma visão de futuro é extremamente importante. Em sistemas de grandes dimensões é fácil perder-se horas no desenvolvimento de módulos que não têm grande utilidade. Como tal, para tirar um melhor partido dos recursos disponíveis, é recomendável existir uma lista de desenvolvimentos que seriam realmente úteis, como é o caso de:

- Integrar uma *interface* com o utilizador dedicada que permitirá ter um aspeto e funcionalidades mais profissionais;
- Tradução do projeto para a plataforma Unreal Engine, com API C++;
- Introduzir mais equipamentos de forma a se poder simular uma instalação completa;
- Adicionar um modo de vídeo que permitisse guardar simulações para mais tarde serem analisadas.

Ainda assim, na indústria, as ideias têm um preço e só se justifica desenvolver algo caso o mesmo seja rentável. Os tempos e a evolução do mercado da Automação ditam o passo a que se podem desenvolver projetos deste tipo, que se verificou ser rentável, visto que reduziu os custos da empresa e aumentou a permanência dos especialistas em escritório, dedicados ao aperfeiçoamento e desenvolvimento de novos produtos.

Referências

[1] *Inventory Management 101: Time to revisit the principles*, Junho de 2016, disponível: *http://www.logisticsmgmt.com/article/inventory_management_101_time_to_revisit_the_principles*

[2] Ramaa. A, K. N. Subramanya, T. M. Rangaswamy, Dept. of Industrial Engg and Management, RVCE, Bangalore, *Impact of Warehouse Management System in a Supply Chain*, Junho de 2016, disponível: *http://research.ijcaonline.org/volume54/number1/pxc3882062.pdf*

[3] *Automated Storange and Retrieval Systems*, Junho de 2016, disponível: *http://www.inc.com/encyclopedia/automated-storage-and-retrieval-systems-as-rs.html*

[4] Michael B. Stroh, *What is Logistics?*, Junho de 2016, disponível: *http://www.logisticsnetwork.net/articles/What%20is%20Logistics.pdf*

[5] Hossein Bidgoli, Susan Malik, *The Handbook of Technology Management: Supply Chain Management, Marketing and Advertising, and Global Management, Edition 1*, 2010

[6] Cephalos GmbH, *TrySim*, Julho de 2016, disponível: *http://www.trysim.de/en/*

[7] Jennifer Marsh, *Advantages of the Unity Game Engine – The Ultimate Tool for Game Development*, Julho de 2016, disponível: *https://blog.udemy.com/unity-game-engine/*

[8] *Unigine Corp*, Indústrias-alvo do Unigine, Julho de 2016, disponível: *https://unigine.com/en/industries*

[9] Laurence Bradford, *Source Code and Language Differences*, Agosto de 2016, disponível: *http://learntocodewith.me/programming/source-code/*

[10] Circuits Today, *Compilers vs Interpreters – An overview of the differences*, Agosto de 2016, disponível: *http://www.circuitstoday.com/compilers-vs-interpreters-an-overview-of-the-differences*

[11] Ivan Zahariev, C++ vs. Python vs. Perl vs. PHP performance benchmark, Setembro de 2016, Disponível: *https://blog.famzah.net/2016/02/09/cpp-vs-python-vs-perl-vs-php-performance-benchmark-2016/*

Anexo A. Calendarização do Projeto