• U C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTAMENTO DE
ENGENHARIA MECÂNICA

Programação *off-line* de robôs a partir de modelos CAD

Dissertação apresentada para a obtenção do grau de Mestre em Engenharia Mecânica na Especialidade de Projecto Mecânico/Sistemas de Produção

Autor

Rui Miguel Faustino Honório

Orientador

Professor Doutor Pedro Mariano Simões Neto

Júri

Presidente	**Professor Doutor Ana Paula Bettencourt Martins Amaro** **Professor Auxiliar da Universidade de Coimbra**
Vogais	**Professor Doutor Joaquim Norberto Cardoso Pires da Silva** **Professor Auxiliar c/ Agregação da Universidade de Coimbra** **Professor Doutor Pedro Mariano Simões Neto** **Professor Auxiliar Convidado da Universidade de Coimbra**

Colaboração Institucional

SOLIEN – Soluções de Engenharia Integradas, Lda.

Coimbra, Setembro, 2013

“If I find 10,000 ways something won't work, I haven't failed. I am not discouraged, because every wrong attempt discarded is another step forward.”

Thomas A. Edison, Encyclopaedia Britannica.

Aos meus pais e irmã, com todo o meu carinho.

Agradecimentos

A realização desta dissertação marca o fim de mais uma importante etapa da minha vida. Gostaria de agradecer a todos os que contribuíram de alguma forma para a sua concretização.

À Solien e Faculdade de Ciências e Tecnologia da Universidade de Coimbra manifesto apreço pela oportunidade e por todos os meios colocados à disposição para a realização deste trabalho. Agradeço igualmente a primazia de formação prestada e conhecimentos transmitidos ambicionando que esta dissertação dignifique, em última instância ambas as instituições.

Agradeço ao meu orientador, Pedro Mariano Simões Neto, pela excelência no apoio, profissionalismo, e sabedoria na tomada de decisões, pela disponibilidade que sempre manifestou, mesmo em alturas difíceis, e pela empatia com que recebeu as minhas ideias.

Ao meu grande amigo Diogo Ferreira da Cunha que teve um papel fundamental na concretização desta dissertação com a partilha de conhecimentos, amizade, esforço e dedicação.

Ao engenheiro e investigador Nuno Marques Mendes pela ajuda incondicional e as longas horas passadas no laboratório em volta do robô.

À minha grande e estimada amiga Diana Sofia Dourado Salgueiro pelo apoio, carinho e motivação que me deu nos últimos anos.

Ao meu colega e amigo João Manuel Pires Duarte pela amizade, incentivo e motivação que me prestou nestes curtos cinco anos.

Aos meus amigos de mecânica que iniciaram comigo esta "caminhada" agradeço a força, a amizade e confiança que depositaram em mim.

Por último, manifesto um sentido e profundo reconhecimento à minha família pelo apoio incondicional ao longo destes anos. Expresso sentimento idêntico em relação a todos os meus amigos de longa data.

A todos que me ajudaram a ser quem sou, que depositam confiança em mim e para os quais sou uma esperança, resta-me afincadamente não vos desiludir. Muito obrigado...

Resumo

O objectivo deste trabalho é o desenvolvimento de um aplicativo intuitivo de programação *off-line* de robôs acessível a PME's inserido num *software* de CAD amplamente utilizado, *Autodesk Inventor*. Procura-se a melhor maneira de integrar o movimento de um robô num desenho CAD, como transformar e traduzir os dados de ambientes virtuais em ambientes reais, simular as posições e orientações do robô nestas plataformas e métodos de geração automática de programas de robô. Em suma, este trabalho tem como objectivo apresentar um novo sistema de programação de robôs baseado em CAD, acessível a qualquer pessoa com conhecimentos básicos em robótica.

Palavras-chave: CAD, OLP, Programação *off-line*, Programação intuitiva de robôs, Robótica.

Abstract

The goal of this work is the development of an intuitive application for off-line robot programming accessible to SMEs, as a component of a widely used and common CAD *software*: Autodesk Inventor. The best way to integrate a robot's movement in a CAD drawing is investigated, in particular on how to transform and translate the data from virtual environments to real environments, simulate the positions and orientation of a robot in these platforms, and the methods of automatic generation of robot programs. In short, this work intends to present a new robot programming system based in CAD, accessible to anyone with basic robotics knowledge.

Keywords CAD, Intuitive robot programming, Off-line programing, OLP, Robotics.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ÍNDICE DE ALGORITMOS

SIMBOLOGIA E SIGLAS

Simbologia

$a, \ldots, z$ – Escalares no espaço de reais $\mathbb{R}$

$\{A\}$ – Representa um sistema de coordenadas com três vectores ortogonais unitários entre si.

p_n – Componente n do vector, $n = x, y, z$

${}^{A}P$ – Vector relativo ao sistema de coordenadas $\{A\}$.

${}^{A}P_{BORG}$ – Vector que localiza a origem (ORG) do sistema de coordenadas $\{B\}$ descrito em termos de $\{A\}$.

$\hat{X}_A$, $\hat{Y}_A$, $\hat{Z}_A$ – Vectores unitários que definem as direcções principais do sistema de coordenadas $\{A\}$.

${}^{A}\hat{X}_B$, ${}^{A}\hat{Y}_B$, ${}^{A}\hat{Z}_B$ – Vectores unitários que definem as direcções principais do sistema de coordenadas $\{B\}$ relativo ao sistema de coordenadas $\{A\}$.

${}^{A}_{B}R$ – Matriz rotação 3x3 que descreve uma rotação particular de $\{A\}$ relativo a $\{B\}$.

${}^{A}_{B}T$ – Descreve a frame $\{B\}$ relativo à frame $\{A\}$

Siglas

3DS – 3D Studio file

API – Application Programming Interface

CAD – Computer Aided Design

DLL – Dynamic-link library

FCTUC – Faculdade de Ciências e Tecnologia da Universidade de Coimbra

HMF – HOOPS Metafile File

HSF – HOOPS Stream File

IGS – Initial Graphics Specification

OLP – Off-line robot Programming

PME – Pequena e Média Empresa

SET – Settings file extension

STEP – Standard for The Exchange of Product model data

STL – STereoLithography

UCS – *User Coordinate Systems*

1. INTRODUÇÃO

No século VXIII deu-se início a um fenómeno conhecido como revolução industrial com um impacto profundo no processo produtivo. Foi a mecanização dos processos de produção que impulsionou esta nova Era. Actualmente vive-se uma economia altamente competitiva onde, e de modo a conseguir-se obter produtos mais baratos, a redução dos custos de produção ganha maior importância. De um modo análogo à teoria Darwinista, para a perseverança de uma indústria há, cada vez mais, a necessidade de esta ser altamente eficiente. Surgem novas necessidades como a redução de ciclos de trabalho e dos elevados custos de mão-de-obra qualificada. Deste modo, no início do século XX, a automatização industrial destacou-se como uma solução e adquiriu um elevado grau de importância. Da mesma forma, também a robotização acompanhou esta evolução permitindo ao mundo industrial produzir uma maior quantidade de produtos, de maior qualidade a um preço mais competitivo. Apesar dos robôs serem actualmente órgãos fundamentais de sistemas industriais, a programação deles por vezes pode ser uma tarefa complexa e requere boas qualificações de programação. Por consequência, surgiram programas que tornam este processo mais intuitivo e acessível. No entanto, devido ao seu elevado custo e à sua complexidade, estas ferramentas só estão ao alcance de uma minoria. Tendo em consideração o presente contexto político-económico faz sentido facultar ferramentas acessíveis a pequenas e médias empresas (PME's) por forma a aumentar a sua competitividade.

Desde a década de 80 que, por necessidade, houve um grande desenvolvimento de ferramentas de modelação de objectos, que se foram tornando progressivamente mais baratas e, deste modo, acessíveis a PME´s. Em particular, o surgir da tecnologia *Computer Aided Design* (CAD) representa uma enorme vantagem no que diz respeito à concepção de novos produtos.

Nesta dissertação procura-se desenvolver uma ferramenta de programação off-line de robôs (OLP) recorrendo à tecnologia CAD tendo como referência as necessidades de uma PME que integra robôs em sistemas de produção, a empresa Solien.

1.1. Contextualização e motivação

A Solien é uma empresa de serviços de engenharia em elevado crescimento que se dedica à elaboração de estudos e projectos de inovação e desenvolvimento, controlo de produção e industrialização de processos. A Solien dotou-se de uma política industrial cujo objectivo é aumentar a sua competitividade numa economia aberta e concorrencial.

No contexto actual, o desenvolvimento de produtos personalizados a preços competitivos é um factor diferenciador. Dos muitos projectos que a Solien integra, grande parte deles são projectos que envolvem robôs manipuladores, onde a selecção da marca ou modelo do robô a integrar na obra, muitas vezes, não depende do engenheiro de projecto da Solien mas sim do cliente. De facto, cada marca de robô tem a sua linguagem e metodologia de programação obrigando a que o programador tenha de ter uma grande capacidade de adaptação. A forma de programação que actualmente se recorre é de ensinamento por *teaching*.

O *teaching* consiste em ensinar directamente o robô guiando a sua extremidade às posições desejadas, bem como atribuir as funções/tarefas que ele vai assumir em cada ponto. No entanto, a qualidade da programação do robô depende da experiência do operador e, no caso de ser necessário fazer posteriores correcções, o processo torna-se muito difícil. Este processo pode ser muito moroso visto que, apesar de intuitivo, o operador tem que lidar com diferentes parâmetros e diversos sistemas de coordenadas.

Foi justamente por esta razão que a Solien propôs o desafio de criar uma ferramenta que permitisse programar os robôs de uma forma simples, económica e de fácil aplicação. Na verdade, a modelação da obra está sempre presente em suporte CAD durante o ciclo do produto, desde o projecto à produção (T. Godinho e J. N. Pires et al., 2006). Este facto motivou o desenvolvimento de um *add-in* (*software* acessório que amplia os recursos de um aplicativo existente) que permita automatizar e simplificar parte do processo de programação.

Com esta dissertação pretende-se desenvolver este *add-in* inserido no *software* de CAD utilizado por esta empresa, *Autodesk Inventor*, que permita delegar o processo de programação do robô ao gabinete de projecto através de um método designado de programação off-line de robôs.

1.2. Desafios

Os desafios propostos ao desenvolvimento desta aplicação são:

- Solução acessível economicamente a PME's;
- Ser uma solução de rápido *setup*;
- Ter a capacidade de programar diferentes marcas de robôs;
- Ser uma solução intuitiva e de fácil utilização;
- Capacidade de ser implementada por um utilizador sem conhecimentos em robótica ou em programação;
- Solução que trabalhe com base num modelo CAD;
- Capacidade de simulação das posições do robô;
- Dispor de ferramentas de ajuste fino.

1.3. Programação de Robôs

Em aplicações industriais, existem, tipicamente, duas categorias principais de métodos de programação de robôs: a programação on-line e a programação *off-line* (OLP).

A programação on-line é normalmente realizada com recurso a uma interface do controlador do robô (*teach pedent*) que permite ao utilizador mover a ferramenta do robô (*end-effector*) manualmente para as posições e orientações desejadas e ainda atribuir as funções/tarefas que ele vai desempenhar em cada uma destas posições (Zengxi Pan et al., 2011).

Apesar do conceito ser aparentemente simples, para projectos em que a geometria e disposição das peças são complexas, o processo pode-se tornar extremamente difícil (Pedro Neto et al., 2012). Para além disso, uma vez gerado o programa, torna-se complicado fazer futuras correcções.

Este método apresenta algumas vantagens em relação a outros:

- É um processo relativamente simples e intuitivo;
- As posições ensinadas ao robô podem realmente ser alcançadas;
- Baixo investimento inicial;

Por outro lado, existem desvantagens como:

- A fase de *teaching* não pode ser executada durante o funcionamento do robô, implicando muitas vezes paragens de produção longas;
- Obriga a que o operador, na fase de programação, esteja perto do robô, podendo muitas vezes ficar exposto a ambientes hostis.
- Dependendo das trajectórias ou mesmo da própria complexidade do sistema, o processo de programação pode ser relativamente moroso comprometendo os custos de produção;
- Movimentar um robô através de um *teach pendent* é relativamente complicado pois o operador tem de lidar com diferentes sistemas de coordenadas, parâmetros, etc.;
- A qualidade do programa do robô está limitada pelo conhecimento técnico do operador;

A programação *off-line* permite que os robôs sejam programados sem que se pare ou perturbe o seu funcionamento e consequentemente a produção. Os "*software"* de OLP recorrem a dados de modelos CAD 3D para simular e gerar os programas de robôs. Toda a programação é desenvolvida num computador independente, tornando-se escusado estar próximo do robô. Neste método, a célula do robô é completamente modelada em 3D onde o utilizador pode testar as propriedades de ajuste fino dos movimentos do robô, verificar se de facto é possível que o robô cumpra as tarefas que lhe são destinadas ou ainda prever eventuais colisões com objectos antes mesmo de o programa ser gerado e carregado no robô (Zengxi Pan et al., 2011). Sempre que necessário, podem ser feitos pequenos ajustes ao código gerado.

As principais vantagens são (Pedro Neto et al., 2012):

- Programação sem interromper ou parar a produção. Os robôs podem ser reprogramados para novas tarefas sem interromper o processo de produção, ou seja, em paralelo, reduzindo as quebras e/ou paragens de produção;
- Os programas de robô anteriores podem ser facilmente alterados e reutilizados;

- Aumento de segurança de trabalho, permitindo que o operador, durante o processo de programação, não se encontre próximo do robô evitando em alguns casos a exposição a ambientes hostis;
- Os programas do robô podem ser testados através de ferramentas de simulação possibilitando prever o comportamento real, bem como optimizar os processos de trabalho;
- Os programas de robô podem ser criados recorrendo a ficheiros CAD;

No entanto, existem algumas desvantagens:

- O processo de calibração requer operadores experientes. A falta de cuidado nesta fase pode levar a erros elevadíssimos quando o robô for operar. Muitas vezes tem mesmo de se recorrer a sensores extra como procedimento adicional de calibração;
- Significativo investimento inicial em "*software"* e na formação dos trabalhadores;
- Problemas com a integrabilidade e a compatibilidade do sistema;
- Requer que se conheça antecipadamente todo o processo de trabalho;
- Os sistemas OLP dependem da precisão do autómato;

A integração de sistemas OLP em PME's é fortemente restringida tanto pela sua complexidade como pelo seu custo elevado (Zengxi Pan et al., 2011). A evolução da tecnologia CAD e da tecnologia computacional tem proporcionado o desenvolvimento de "*software*" OLP alternativos, por parte de investigadores, transformando uma tarefa morosa e penosa numa tarefa rápida e intuitiva (D. J. Kasik et al., 2005;T. Godinho e J. N. Pires et al., 2006).

Os ficheiros neutros (STL, IGS, STEP e SET), utilizados na tecnologia CAD, revelaram uma vantagem para o desenvolvimento destas aplicações, permitindo a uniformização de dados, e contribuindo para uma ferramenta capaz de interagir entre soluções de diferentes fabricantes. Estes ficheiros, surgiram como um modo de interligar, a importação e exportação de ficheiros, nas demais ferramentas CAD (V. Bottazzi, J. Fonseca et al., 2006).

A simulação tem vindo a ser reconhecida como uma ferramenta importante para a programação off-line. Cada vez mais, os complexos sistemas de robôs precisam de

ferramentas de programação rápidas e sofisticadas que permitam modelar ambientes com o máximo de precisão (Freese, M.; Singh, S. P. N.; Ozaki, F. & Matsuhira, N. et al., 2010). O rápido desenvolvimento de computadores de elevada capacidade gráfica e de processamento tornou possível simular sistemas dinâmicos que envolvem robô e objectos sob a influência de forças externas (L. Zlajpah et al., 2008). Os sistemas de simulação avançada permitiram ainda elaborar complexas técnicas de sistemas de cooperação multi-robô que podem ser aplicadas à programação *off-line* (Yahui Gan et al., 2013). Actualmente estão disponíveis um grande número de "*software*" que permitem, por exemplo, detectar colisões ou realizar planeamentos de trajectórias (Lonnie J. Love et al., 2005; Neto P., Pires J.N. e Moreira A.P et al., 2010). Desta forma, têm vindo a ser conduzidas pesquisas de simulação de sistemas OLP de diversas áreas como optimização de processos de soldadura e na de integração de ferramentas na cooperação de robôs (M. Bruccoleri, C. D'Onofrio e U. La Commare et al., 2007; N. Papakostas et al., 2011).

A simulação com recurso a modelos CAD tem vindo a revelar-se uma área prometedora (Neto P., Mendes N et al., 2013). Com frequência, estão presentes as modelações dos projectos em formato CAD. Tal facto motivou várias pesquisas, tanto em sistemas capazes de extrair informação do movimento de robôs a partir destes modelos, como no desenvolvimento de plataformas que integram nestes sistemas CAD ferramentas intuitivas de programação *off-line* ((Pires J. N. et al., 2004; Neto P, Pires J. N. et al., 2010; Neto P., Mendes N., Araújo R., Pires J.N. e Moreira A.P. et al., 2012).

1.4. Abordagem proposta

De modo a solucionar os desafios discutidos na secção 1.2 propõe-se, nesta dissertação, o desenvolvimento de uma aplicação que permita ao utilizador programar em ambiente *off-line* qualquer marca ou modelo de robô. Assim, ir-se-á realizar primeiramente um *benchmarking* de produtos similares ao que se pretende desenvolver.

Esta aplicação será implementada no programa de modelação CAD *Autodesk Inventor*, da *Autodesk*, em forma de *add-in* e será programada no *software Microsoft Visual Basic 2010* em *VB.net*.

Com o compromisso de ser intuitiva terá o número mínimo de interfaces possível. Esta solução terá que ter ainda a capacidade de simular as posições do robô e de dispor de ferramentas de ajuste fino numa só janela.

Terá também que ser de fácil e rápida implementação, sem quaisquer custos, para isso sugere-se que seja compilada num instalador em formato *wizard*.

A orientação de objectos será com base em transformações matriciais no espaço e nos parâmetros de Euler, sendo a posição e orientação da ferramenta do robô exclusivamente responsável pelo reposicionamento (adaptabilidade) das partes móveis do robô.

Do produto final dever-se-á ser capaz exportar um ficheiro texto com o código do programa gerado que, posteriormente, se importará para o robô garantindo a sua pronta operacionalidade, por isso, é imperativo que este *add-in* disponha de uma secção de geração de código onde o utilizador o possa parametrizar.

2. BENCHMARKING

Para criar ou desenvolver um novo produto é fundamental realizar uma prospecção de mercado de forma a saber se já existem produtos iguais ou semelhantes, quais as principais características e o que difere de uns para outros. O *benchmarking* é geralmente associado à avaliação das características de performance de uma família de hardwares de computador. No entanto, esta técnica, é frequentemente utilizada na avaliação de "*software*". Em "*software*", *benchmarking* é o acto de executar um conjunto de testes a um grupo específico de "*software*" de modo a avaliar a performance relativa de determinada característica comum entre eles. Por vezes é necessário incluir na análise características extra como o preço, a manutenção ou mesmo o tipo de licenças. No nosso caso, estes factores são fundamentais dado que um dos principais objectivos do desenvolvimento deste *software* OLP é a redução significativa de custos, principalmente com o pessoal.

2.1. *Software* OLP

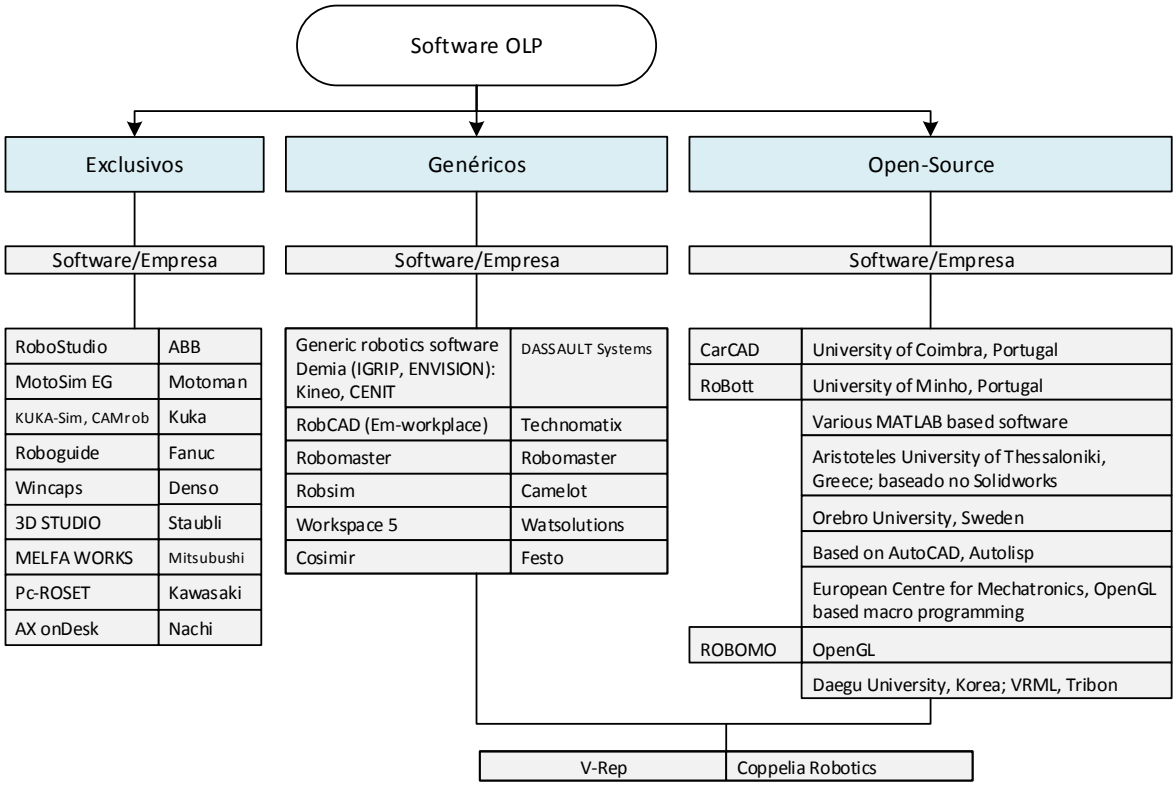
A complexidade dos sistemas de manipulação de robôs motivou, aos fabricantes de robôs, investigadores académicos e mesmo a independentes, o desenvolvimento de plataformas computacionais que permitem simplificar o processo de OLP.

O universo destes "*software*" pode ser dividido em três grupos:

- "*Software*" de fabricantes de robôs, que são desenvolvidos apenas para programar ou simular os seus robôs, designados daqui em diante por "*software"* exclusivos;
- Os "*software*" de empresas independentes/privadas, que geralmente permitem programar ou simular mais do que uma marca de robôs, designados doravante por "*software"* genéricos;
- "*Software*" open source (código aberto) ou académicos.

Na Tabela 2-1 representa-se esquematicamente alguns "*software*" de OLP.

Tabela 2-1 Universo de "*software"* OLP de robôs.



2.2. Estudo comparativo

Na Tabela 2-2 compara-se as principais características de alguns "*software*" OLP de robôs. No anexo A apresenta-se uma descrição mais pormenorizada de cada *software* estudado.

Tabela 2-2 Principais características de "*software*" OLP de robôs

Benchmarking de "software" OLP

Fabricante		Motoman	FANUC	KUKA	ABB	Dassault Systèmes	Fastems	Erxa	Coppelia Robotics	Visual Components	Mastercam
Software		Motosim EG	ROBOGUIDE	KUKA.Sim Pro	RobotStudio	DELMIA OLP	FestSimu	RoboWave hyprogram	V-rep	3DAutomate	Robotmaster
Simulador/OLP		●/●	●/●	●/●	●/●	●/●	●/●	●/●	●/●	●/●	●/●
Licença Comercial/Open Source		●/○	●/○	●/○	●/○	●/○	●/○	●/○	●/●	●/○	●/○
Software adicionais/ licença comercial	ArcWelding	○/○	●/●	●/●	●/●	○/○	○/○	○/○	○/○	○/○	○/○
	Bending	○/○	○/○	●/●	●/●	○/○	○/○	●/●	○/○	○/○	○/○
	Machining	○/○	○/○	●/●	●/●	○/○	●/○	○/○	○/○	○/○	○/○
	Machining Tending	○/○	●/●	●/●	●/●	○/○	○/○	○/○	○/○	○/○	○/○
	Palletizing	○/○	●/●	●/●	●/●	○/○	○/○	○/○	○/○	○/○	○/○
	Cutting	○/○	○/○	●/●	●/●	○/○	○/○	○/○	○/○	○/○	○/○
	Paiting	○/○	●/●	●/●	●/●	○/○	○/○	○/○	○/○	○/○	○/○
Software adicionais para manutenção de sistemas		●	●	●	●	○	○	○	○	○	○
Multi-robô		○	●	○	●	●	N/A	●	●	●	●
Interface friendly user		○	●	●	●	●	●	●	●	●	●
Suporta CAD (IGES)		○	●	●	●	●	●	●	●	●	●
Gera Caminho		●	●	N/A	●	N/A	●	●	●	●	●
para calibração		○	●	N/A	●	●	●	●	●	●	●
Caminho		●	●	N/A	●	N/A	●	●	●	●	●
Opera com Multimarca		○	○	○	○	●	●	●	●	●	●
Detecta Colisões		●	●	●	●	●	●	●	●	●	●
Tempo de ciclo		●	●	●	●	N/A	●	●	●	●	●
ScreenRecorder		●	●	●	●	N/A	N/A	●	●	●	●
Debug		●	●	●	●	●	●	●	●	●	●
Gestão I/O		●	●	●	●	●	●	●	●	●	●
Simulação em tempo real		●	●	●	●	●	●	●	●	●	●

2.3. Discussão e comparação

Os "*software*" OLP de robôs, para além de permitirem programar *off-line*, apresentam ainda grandes vantagens como a simulação, geração de trajectórias, detecção de colisões, previsão do tempo de ciclo, gestão I/O, entre outras. No entanto, dos três

grupos de "*software*" OLP, os "*software*" *open-source* ainda estão em desenvolvimento e não têm verdadeiramente aplicação prática ou intuitiva e, tanto os "*software*" de produtores de robôs como os de empresas independentes, são maioritariamente comercializados segundo um sistema de licenças de tempo limitado ou fornecidos de forma parcial, isto é, o cliente compra o sistema base (*software* principal) com funções limitadas e posteriormente adquire packs de expansão como soldadura, manipulação de objectos, pintura, gestão, entre outros, revelando-se um enorme investimento em produtos que rapidamente ficam desactualizados. Os encargos inerentes a este género de produtos só são admissíveis a empresas cujo núcleo de produção seja maioritariamente programação de robôs.

Como tal facto não se aplica à Solien propõe-se desenvolver uma nova ferramenta com características semelhantes aos "*software*" analisados.

2.4. Estudo de caso

No seguimento do *benchmarking* de "*software*" surgiu a oportunidade de explorar o *software* de OLP, *Motosim EG* da *Motoman*, aplicado a um projecto que decorria na Solien, na expectativa de perceber como realmente funciona este tipo de produtos.

Este projecto Figura 2-1 tem a finalidade de verificar se dois modelos de peças, A e B, apresentam defeitos a nível de estanquicidade. Este equipamento é composto por quatro módulos.

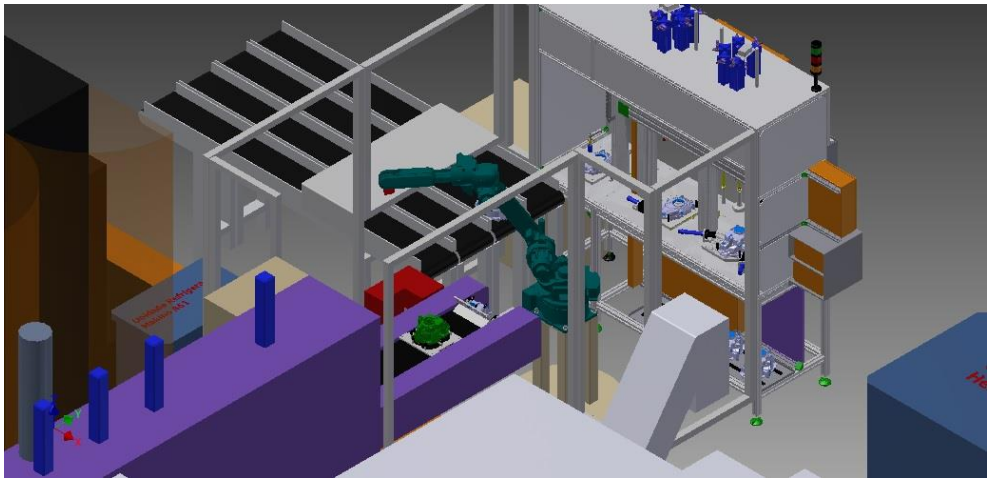


Figura 2-1 Modelo CAD da célula em estudo.

O primeiro módulo é constituído por um tapete transportador que atravessa um túnel de lavagem prevenindo que impurezas "parasitas" adulterem os testes ou danifiquem o equipamento. O segundo trata-se de um robô MOTOMAN MH6-10E que é responsável pelo transporte das peças para cada módulo de trabalho. O terceiro tem a função de averiguar a estanquicidade das peças. A triagem das peças é realizada no quarto e último módulo que é constituído por quatro tapetes transportadores, dois para cada tipo de peça (A ou B) e cada um deles para peças com ou sem defeito, como está representado esquematicamente na Figura 2-2.

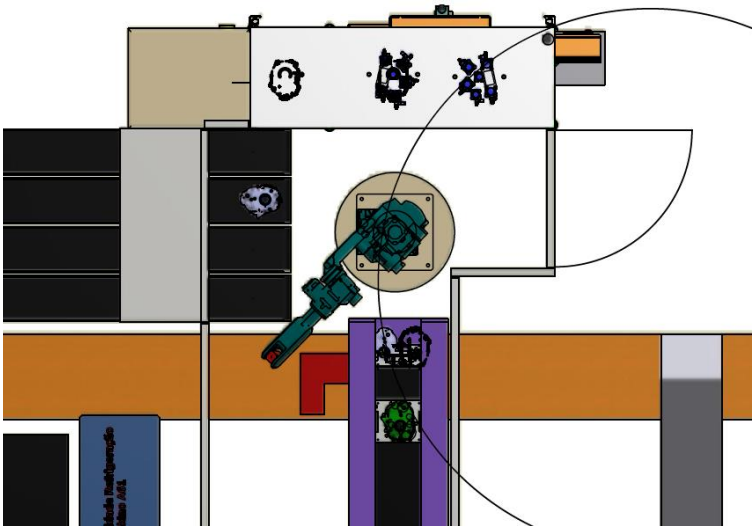


Figura 2-2 Modelo 2D CAD da célula em estudo (vista de cima).

2.4.1. Implementação, resultados e discussão

A primeira fase deste processo foi a conversão dos ficheiros de modelação CAD para um formato suportado pelo *Motosim EG* (3DS). A conjugação dos poucos recursos de modelação gráfica deste *software* e a complexidade da ferramenta de trabalho proposta pela Solien Figura 2-3 gerou a necessidade de desenvolver um modelo CAD simplificado da ferramenta.

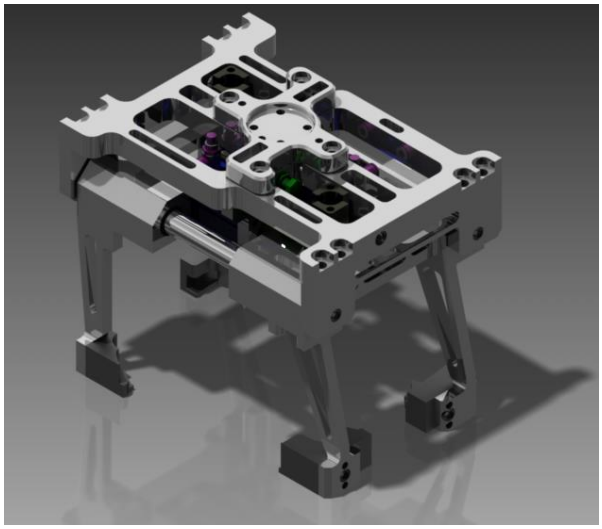


Figura 2-3 Modelo CAD da frame tool da célula do caso em estudo.

Uma vez convertidos os ficheiros e desenvolvido o modelo simplificado da ferramenta, definiram-se as várias posições do robô e verificou-se se ocorriam colisões. Após a validação da programação através de simulação importou-se o código gerado Figura 2-4 para o robô.

```
63  C0055=188999,-46656,-9603,0,
64  C0056=106189,-72136,-9603,0,25477,-290
65  C0057=-142241,-72136,-9603,0,25477,-290
66  //INST
67  ///DATE 2013/04/10 11:37
68  ///ATTR SC,RW
69  ///GROUP1 RB1
70  NOP
71  MOVJ C0000 VJ=10.00
72  DOUT OT#(99) ON
73  MOVJ C0001 VJ=5.00
74  MOVJ C0002 VJ=2.00
75  DOUT OT#(9) ON
```

Figura 2-4 Fragmento do código gerado pelo programa Motosim EG.

Na Figura 2-5 apresentam-se alguns estágios da simulação realizada.

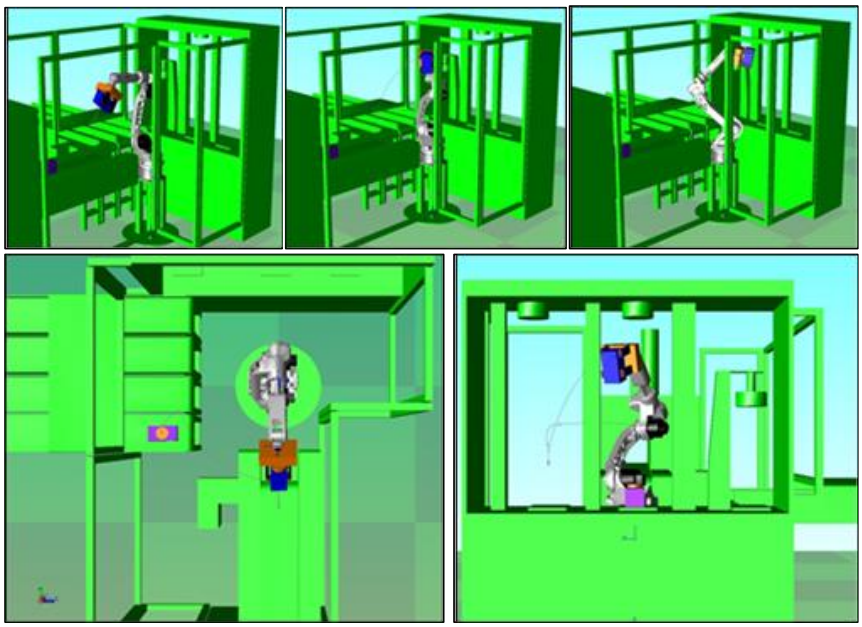


Figura 2-5 Simulação no Motosim EG do robô a executar o programa.

A conjugação de sistemas OLP com sistemas de simulação traduz enormes vantagens pois para além de permitir resolver problemas e encontrar soluções na programação de robôs de um modo rápido e criativo, permite ainda prever eventuais colisões prevenindo potenciais acidentes no local, tanto para os equipamentos como para os trabalhadores. Desta forma, aquando a implementação do código gerado garante-se que tudo vai estar dentro das normas e que apenas serão necessários pequenos ajustes.

A aplicação, tanto do *RobotStudio* como do *MotoSim EG*, a este projecto permitiu perceber as vantagens e desvantagens deste tipo de produtos no contexto industrial. Por exemplo, ao contrário do *RobotStudio* da *ABB*, que também foi objecto de análise, o *MotoSim EG* é um *software* desactualizado, pouco intuitivo, extremamente difícil de trabalhar e requere uma formação extremamente morosa e específica. O simulador *Motosim EG* apresenta enormes dificuldades no que diz respeito à importação de ficheiros neutros. Apenas admite importar um leque limitado de formatos de ficheiros CAD (HMF, HSF e 3DS).

Apesar de alguns erros de calibração devido a falhas de comunicação com o projectista (por exemplo alterações das dimensões de elementos do projecto, situação

comum), o programa gerado foi implementado com sucesso, e o feedback dos engenheiros responsáveis positivo. Este estudo intensificou o interesse dos responsáveis da Solien por este tipo de produtos, reconhecendo as enormes vantagens de ter uma ferramenta como esta no seu processo de produção.

3. TRANSFORMAÇÕES NO ESPAÇO

Em robótica é frequente ser necessário representar a posição e orientação de um objecto em diferentes sistemas de coordenadas.

Neste capítulo faz-se uma breve referência a alguns conceitos necessários para o desenvolvimento deste *add-in* como, as descrições e transformações no espaço, parâmetros de Euler e quaterniões.

3.1. Descrição de posição

Sempre que exista um sistema de coordenadas podemos localizar qualquer ponto no espaço através de um vector de posição 3×1 Figura 3-1.

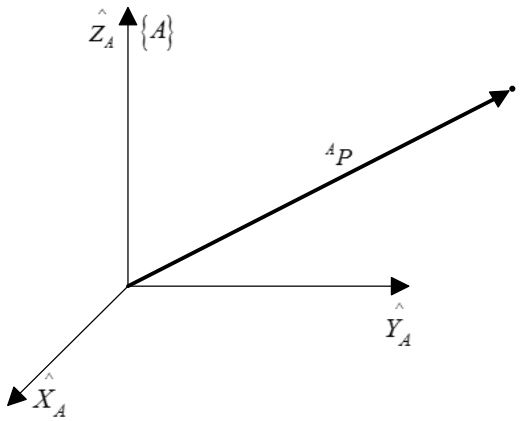


Figura 3-1 Vector relativo a uma frame (exemplo).

Consideremos ${}^{A}P$ um componente com valores numéricos que indica a distância à origem do sistema de coordenadas $\{A\}$. As projecções de cada componente do ponto P representam a distância a que o ponto está da origem em relação aos respectivos eixos de $\{A\}$:

$$ {}^{A}P = \begin{bmatrix} Px \\ Py \\ Pz \end{bmatrix}. \tag{3.1} $$

3.2. Descrição de uma orientação

Para a descrição de uma orientação no espaço é necessário adicionar um outro sistema de coordenadas $\{B\}$ e dar uma descrição do novo sistema de coordenadas em relação ao sistema de referência $\{A\}$, como indicado na Figura 3-2.

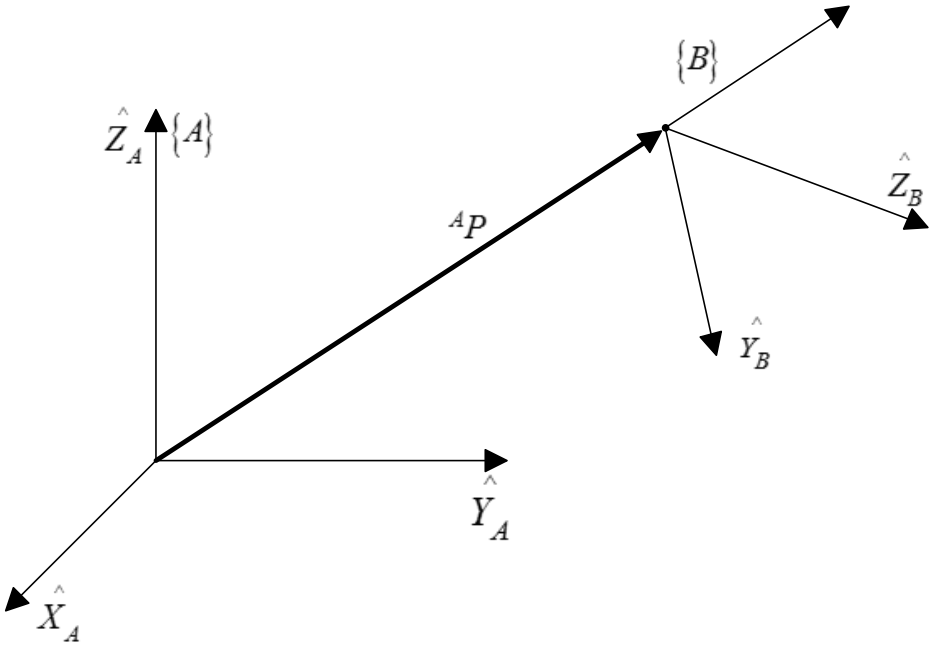


Figura 3-2 Vector relativo a uma frame (exemplo).

Uma forma de descrever o sistema de coordenadas $\{B\}$, é escrever os vectores unitários dos seus eixos ($\hat{X}_b$, $\hat{Y}_b$ e $\hat{Z}_b$), em termos do sistema de coordenadas de $\{A\}$, denotado por $^A\hat{X}_b$, $^A\hat{Y}_b$ e $^A\hat{Z}_b$. Assim, dizemos que o sistema $\{B\}$ está rodado relativamente a $\{A\}$ segundo a matriz rotação ${}^A_B R$, em que cada coluna representa a rotação de cada eixo de $\{B\}$ relativamente a cada eixo de $\{A\}$:

$$ {}^A_B R = \begin{bmatrix} {}^A\hat{X}_b & {}^A\hat{Y}_b & {}^A\hat{Z}_b \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \quad (3.2) $$

A descrição dada de $\{A\}$ para $\{B\}$ é dada pela sua transposta,

$$ {}^{A}_{B}R = {}^{B}_{A}R^{T}. \tag{3.3} $$

pois se,

$$ {}^{A}_{B}R^{T}\ {}^{A}_{B}R = \begin{bmatrix} {}^{A}\hat{X}_b \\ {}^{A}\hat{Y}_b \\ {}^{A}\hat{Z}_b \end{bmatrix} \begin{bmatrix} {}^{A}\hat{X}_b & {}^{A}\hat{Y}_b & {}^{A}\hat{Z}_b \end{bmatrix} = I_3, \tag{3.4} $$

onde I_3 é uma matriz identidade 3×3. Ficamos com,

$$ {}^{A}_{B}R = {}^{B}_{A}R^{-1} = {}^{B}_{A}R^{T}. \tag{3.5} $$

Designemos por frame um conjunto de quatro vectores (três para a rotação e um para a posição) que descreve a posição e orientação de um objecto no espaço. Seja $\{B\}$ o sistema de coordenadas do objecto descrito por ${}^{A}_{B}R$ e ${}^{A}P_{BORG}$, onde ${}^{A}P_{BORG}$ é o vector que localiza a origem do objecto, ou melhor, o vector que localiza a origem do sistema de coordenadas de $\{B\}$ relativamente a $\{A\}$:

$$ \{B\} = \left\{ {}^{A}_{B}R,\ {}^{A}P_{BORG} \right\}. \tag{3.6} $$

Suponhamos que temos um ponto descrito relativamente a $\{B\}$ através de um vector ${}^{B}P$. Para expressar este ponto relativamente à frame $\{A\}$, Figura 3-3:

$$ {}^{A}P = {}^{B}P + {}^{A}P_{BORG}. \tag{3.7} $$

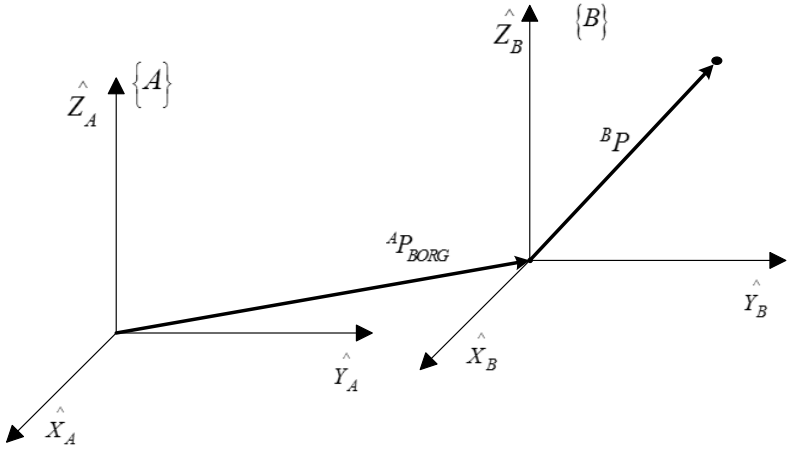


Figura 3-3 Vector descrito relativamente à frame B e frame B relativa à A.

Consideremos que se pretende descrever um ponto no espaço relativamente ao objecto, $\{B\}$, e que $\{B\}$ tem uma rotação relativa a $\{A\}$ Figura 3-4.

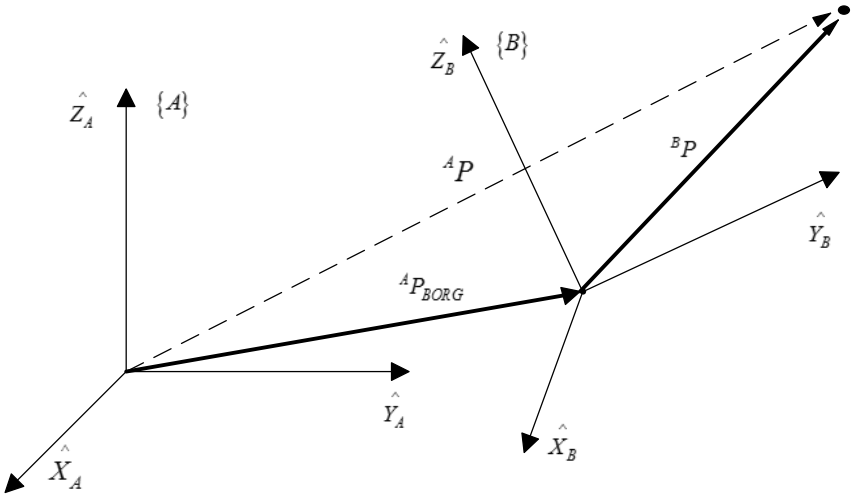


Figura 3-4 Mapeamento de uma translação.

Já vimos que o ponto P é descrito relativamente a $\{B\}$ pelo vector ${}^{B}P$. Se pré-multiplicar-mos ${}^{A}_{B}R$ por ${}^{B}P$ e adicionarmos ${}^{A}P_{BORG}$, ficamos com ${}^{A}P$. Vejamos:

$$ {}^{A}P = {}^{A}_{B}R\,{}^{B}P + {}^{A}P_{BORG}. \tag{3.8}$$

Seja ${}^{A}_{B}T$ a descrição da frame $\{B\}$ relativa à frame $\{A\}$. Especificamente as colunas de ${}^{A}_{B}R$ são vectores que definem as direcções principais da frame $\{B\}$ e ${}^{A}P_{BORG}$ localiza a origem da frame $\{B\}$. De (3.8),

$$ {}^{A}P = {}^{A}_{B}T + {}^{B}P. \tag{3.9}$$

Deste modo, podemos escrever:

$$\begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & {}^{A}_{B}R\,{}^{B}P & & {}^{A}P_{BORG} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix}. \tag{3.10}$$

3.3. Transformações compostas

Na Figura 3-5, temos ${}^{C}P$ e queremos encontrar ${}^{A}P$. Sabemos $\{C\}$ relativamente a $\{A\}$ e $\{B\}$ relativamente a $\{A\}$.

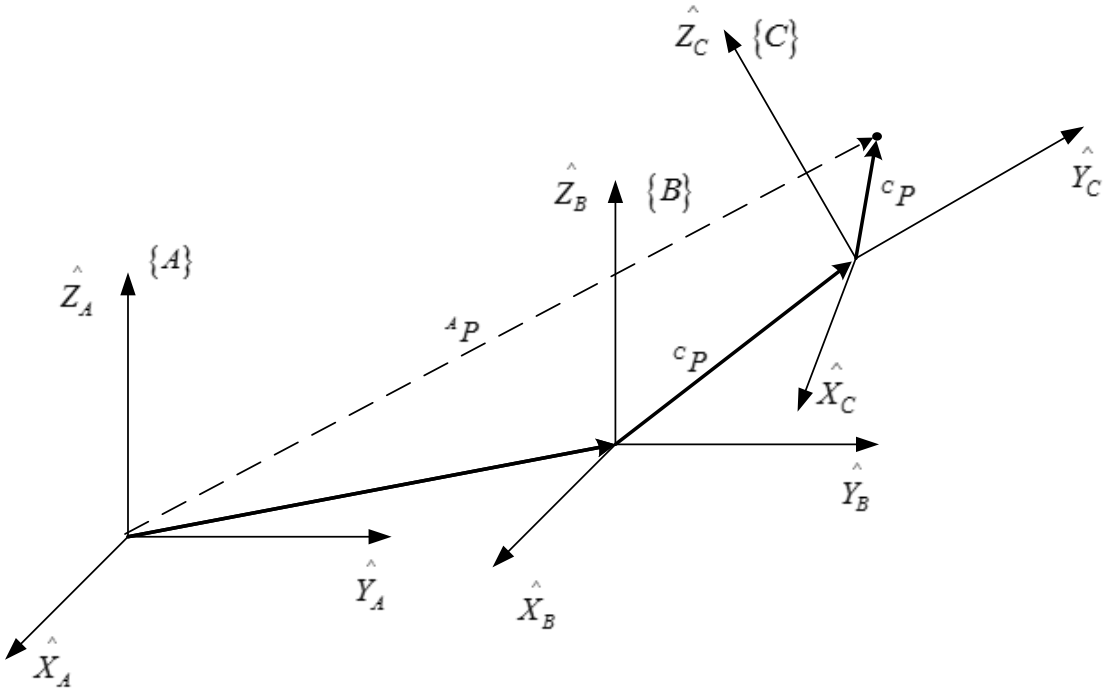


Figura 3-5 Frames compostas: cada uma é conhecida relativamente à anterior.

Transformando ${}^{C}P$ em ${}^{B}P$,

$$ {}^{B}P = {}^{B}_{C}T \ {}^{C}P; \tag{3.11} $$

e ${}^{B}P$ em ${}^{A}P$,

$$ {}^{A}P = {}^{A}_{B}T \ {}^{B}P. \tag{3.12} $$

De 3.11 e 3.12 obtemos,

$$ {}^{A}P = {}^{A}_{B}T \ {}^{B}_{C}T \ {}^{C}P, \tag{3.13} $$

ou seja,

$$ {}^{A}_{C}T = {}^{A}_{B}T \ {}^{B}_{C}T. \tag{3.14} $$

Deste modo, podemos escrever:

$$ {}^{A}_{C}T = \left[\begin{array}{ccc|c} & {}^{A}_{B}R\,{}^{B}_{C}R & & {}^{A}_{B}R\,{}^{B}P_{CORG} + {}^{A}P_{BORG} \\ 0 & 0 & 0 & 1 \end{array}\right]\left[\begin{array}{c} {}^{B}P \\ 1 \end{array}\right]. \tag{3.15} $$

3.4. Inversa da transformada

Consideremos uma frame $\{B\}$ conhecida relativa a $\{A\}$ por ${}^{A}_{B}T$. Para obter ${}^{B}_{A}T$, que é $\{A\}$ relativamente a $\{B\}$ fazemos,

$$ {}^{B}_{A}R = {}^{A}_{B}R^{T}. \tag{3.16} $$

Seguidamente alteramos a descrição de ${}^{A}P_{BORG}$ em $\{B\}$ através de,

$$ {}^{A}P = {}^{A}_{B}R\,{}^{B}P, \tag{3.17} $$

$$ {}^{B}({}^{A}P_{BORG}) = {}^{B}_{A}R\,{}^{A}P_{BORG} + {}^{B}P_{AORG}. \tag{3.18} $$

Como a igualdade (3.18) tem de ser zero,

$$ {}^{B}P_{AORG} = -{}^{B}_{A}R\,{}^{A}P_{BORG} = -{}^{A}_{B}R^{T}\,{}^{A}P_{BORG}. \tag{3.19} $$

Recorrendo a (3.16), (3.18) e (3.19),

$$ {}^{B}_{A}T = \left[\begin{array}{ccc|c} & {}^{A}_{B}R^{T} & & -{}^{A}_{B}R^{T}\,{}^{A}P_{BORG} + {}^{A}P_{BORG} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]. \tag{3.20} $$

Ou seja,

$$ {}^{B}_{A}T = {}^{A}_{B}T^{-1}. \tag{3.21} $$

3.5. Parâmetros de Euler

3.5.1. Ângulos de Euler

Um método para representar matrizes de rotação é através dos ângulos de Euler. Uma vez que a ordem de rotação influencia o resultado obtido, a solução adoptada nesta dissertação é a de $X-Y-Z$, equação (3.22), Figura 3-6.

$$ {}^{A}_{B}R_{X'Y'Z'}(\alpha,\beta,\gamma) = \begin{bmatrix} c\beta c\gamma & -c\beta s\gamma & s\beta \\ s\alpha s\beta c\gamma + c\alpha s\gamma & -s\alpha s\beta s\gamma + c\alpha c\gamma & -s\alpha c\beta \\ -c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha s\beta s\gamma + s\alpha c\gamma & c\alpha c\beta \end{bmatrix}, \tag{3.22} $$

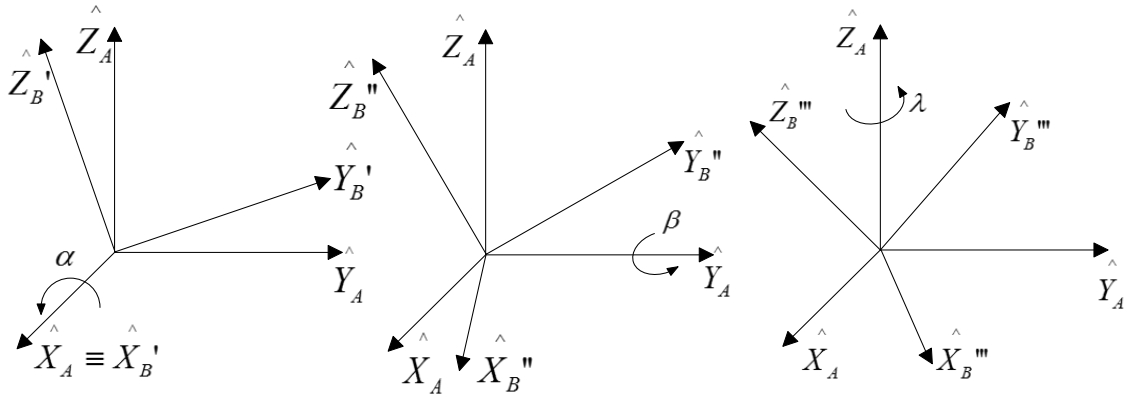


Figura 3-6 Ordem de rotação: $X-Y-Z$ (exemplo).

Dada uma matriz de rotação,

$$ {}^{A}_{B}R_{X'Y'Z'}(\alpha,\beta,\gamma) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{3.23} $$

a solução para extrair os ângulos de Euler $X-Y-Z$ de uma matriz de rotação quando $\beta \neq \pm\left(\frac{\pi}{2}\right)$ é:

$$\alpha = A\tan 2(r_{23}, r_{33}), \tag{3.24}$$

$$\beta = A\tan 2(-r_{13}, \sqrt{r_{11}^{\ 2} + r_{12}^{\ 2}}), \tag{3.25}$$

$$\gamma = A\tan 2(\sin(\alpha) r_{31} - \cos(\alpha) r_{21}, \sin(\alpha) r_{22} - \cos(\alpha) r_{32}). \tag{3.26}$$

3.5.2. Gimbal lock

Uma particularidade associada ao método dos ângulos de Euler é que quando $\beta = \pm(\pi / 2)$, o processo de cálculo dos ângulos de Euler torna-se mais complexo. Situações particulares em que o eixo x está alinhado com o eixo z resultam numa perda de um grau de liberdade e torna o problema matematicamente insolúvel. Nesta situação tanto α como γ não podem ser calculados separados mas sim em conjunto:

$$\alpha \pm \gamma A\tan 2(r_{32}, r_{22}) \tag{3.27}$$

3.6. Quaterniões

Apesar do fenómeno Gimbal lock fazer com que os ângulos de Euler não se apliquem em algumas situações não significa que estejam errados. Este facto motivou ao surgimento de novos métodos para representar matrizes de rotação que permitissem ultrapassar tal obstáculo, por exemplo, soluções de representação de orientações de corpos rígidos através de quaterniões.

Em termos de angulo θ e eixos equivalentes $\hat{K} = [k_x k_y k_z]^T$, os parâmetros de Euler são dados por,

$$\begin{aligned} \varepsilon_1 &= k_x \sin\frac{\theta}{2}, \\ \varepsilon_2 &= k_y \sin\frac{\theta}{2}, \\ \varepsilon_3 &= k_z \sin\frac{\theta}{2} \\ \varepsilon_4 &= \cos\frac{\theta}{2}. \end{aligned} \tag{3.28}$$

e ε_1, ε_2, ε_3, ε_4 não são independentes,

$$\varepsilon_1^{\,2} + \varepsilon_2^{\,2} + \varepsilon_3^{\,2} + \varepsilon_4^{\,2} = 1 \tag{3.29}$$

A matriz rotação que R_ε é,

$$R_\varepsilon = \begin{bmatrix} 1-2\varepsilon_2^{\,2}-2\varepsilon_3^{\,2} & 2(\varepsilon_1\varepsilon_2-\varepsilon_3\varepsilon_4) & 2(\varepsilon_1\varepsilon_3+\varepsilon_2\varepsilon_4) \\ 2(\varepsilon_1\varepsilon_2+\varepsilon_3\varepsilon_4) & 1-2\varepsilon_1^{\,2}-2\varepsilon_2^{\,2} & 2(\varepsilon_2\varepsilon_3-\varepsilon_1\varepsilon_4) \\ 2(\varepsilon_1\varepsilon_3-\varepsilon_2\varepsilon_4) & 2(\varepsilon_2\varepsilon_3+\varepsilon_1\varepsilon_4) & 1-2\varepsilon_1^{\,2}-2\varepsilon_2^{\,2} \end{bmatrix}. \tag{3.30}$$

Dada uma matriz de rotação os ângulos de Euler equivalentes são:

$$\begin{aligned} \varepsilon_1 &= \frac{r_{32}-r_{23}}{4\varepsilon_4}, \\ \varepsilon_2 &= \frac{r_{13}-r_{31}}{4\varepsilon_4}, \\ \varepsilon_3 &= \frac{r_{21}-r_{12}}{4\varepsilon_4}, \\ \varepsilon_4 &= \frac{1}{2}\sqrt{1+r_{11}+r_{22}+r_{33}}. \end{aligned} \tag{3.31}$$

3.7. Considerações adicionais

O estudo que aqui se apresentou das transformações no espaço, parâmetros de Euler e dos quaterniões é, necessariamente, incompleto. O objectivo deste capítulo era apresentar os conceitos fundamentais que foram necessários para o desenvolvimento deste *software*.

Como referências principais para o tema e que serviram de suporte a este trabalho aponta-se [John J. Craig et al., 2005] e [Pedro Neto et al., 2012].

4. PROGRAMAÇÃO OFF-LINE BASEADA EM CAD

Um dos objectivos principais deste projecto consistiu no desenvolvimento de ferramentas simples e amigáveis para programar robôs. Neste sentido, foi codificado, no sistema Visual Basic, um conjunto de funções de manipulação de objectos e de programação de robôs.

Neste capítulo são apresentados os contributos computacionais do trabalho realizado: o *software* e um pacote de rotinas para a manipulação de robôs.

4.1. Interacção com CAD

Para a interacção do programa desenvolvido com o *Autodesk Inventor* recorre-se a um wizard disponibilizado pela *Autodesk*. Um wizard consiste numa interface que apresenta ao utilizador uma sequência de opções bem definidas, passo a passo. É utilizado frequentemente para simplificar tarefas normalmente complexas ou de código fechado. No anexo B apresenta-se as principais características deste *wizard*.

O código fonte do *Autodesk Inventor* é fechado de forma a proteger os seus direitos intelectuais. Contudo, com vista a permitir o desenvolvimento de novas ferramentas adaptadas às necessidades de cada utilizador a *Autodesk* desenvolveu uma API (*Application Programming Interface*) que consiste numa biblioteca com especificações de rotinas, estruturas de dados, classes de objectos e variáveis que especifica como o utilizador há-de interagir com os componentes do *software*. A interacção com API, ilustrado na Figura 4-1, pode ser realizada através de linguagens de programação como *Visual Basic* (*VB*), *Visual C#* e *Visual C++*. Desta forma, para o desenvolvimento da aplicação proposta recorre-se ao *VB*, VB.net.

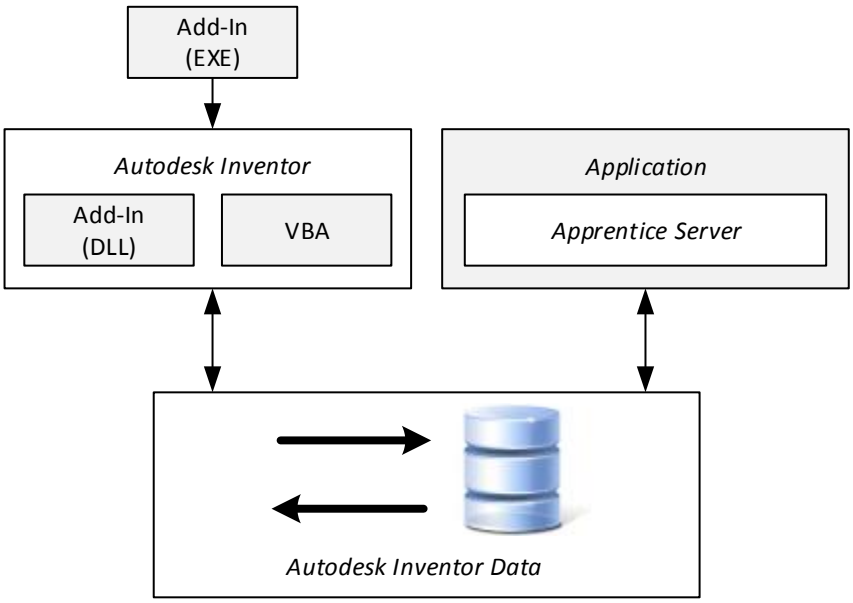


Figura 4-1 Interacção do *add-in* com a API do *Autodesk Inventor*.

4.2. Add-in

Comecemos por analisar a interface do *Inventor* numa perspectiva de utilizador final. A instalação do *add-in* dá-se no instante em que se corre o instalador e é responsável por integrar no *Autodesk Inventor* todo o código desenvolvido nesta dissertação. Após a instalação, sempre que o utilizador correr o *Inventor* estarão disponíveis as ferramentas desenvolvidas que permitem programar robôs de um modo *off-line*.

Estas ferramentas estão acessíveis através de *ribbon tabs*, *ribbon panels*, e *botões*. Os vários elementos da interface do *Inventor* estão expostos na Figura 4-2.

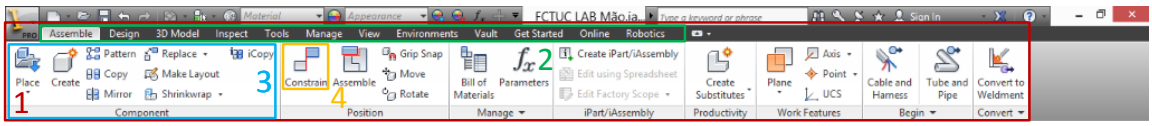


Figura 4-2 Interface de comandos do Autodesk Inventor.

1. *Ribbon*- É uma faixa onde a barra de ferramentas é apresentada através de uma barra mais larga contendo um grupo de Ribbon tab.
2. *Ribbon Tab*- Uma ribbon tab é um separador inserido numa Ribbon que permite organizar as mais diversas ferramentas.

3. *Ribbon Panel* – É um painel que permite organizar um grupo de comandos (botões) dentro de uma ribbon tab.
4. *Buttons* – São botões que permitem aceder a um ou mais controlos.

4.3. Abordagem geral do *software*

As ferramentas desenvolvidas para esta dissertação estão disponíveis através da *ribbon tab* com o nome "Robotics" e estão divididas em duas classes: ferramentas auxiliares e ferramentas de programação de robôs, respectivamente, "Operations" e "*OLP"*, Figura 4-3.

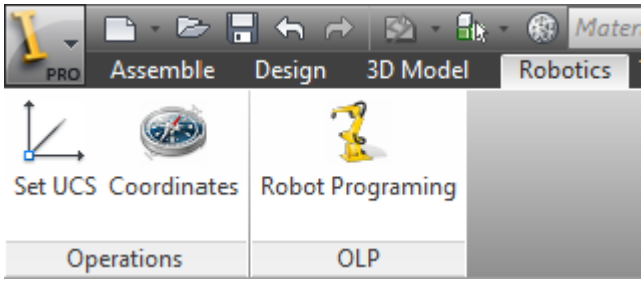


Figura 4-3 Interface das ferramentas desenvolvidas para este *software.*

Movimentar um robô no espaço nem sempre é uma tarefa fácil. Quando se trabalha em modelos CAD tridimensionais complexos é difícil ter a percepção de distâncias ou de dimensões. Desta forma, propôs-se um sistema de medida, Figura 4-4**,** que permite saber a posição espacial de um *workpoint* (pontos de trabalho) ou de um UCS (*User Coordinate System*) em qualquer ponto do desenho que tenha sido previamente definido pelo utilizador, Algoritmo 4-1. Esta ferramenta é designada por "Coordinates".

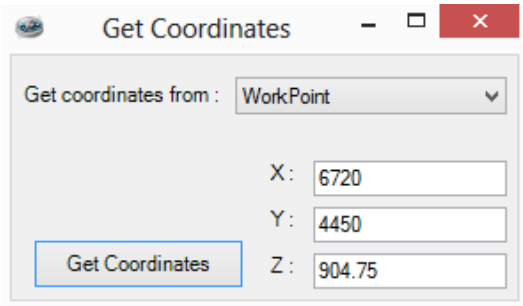


Figura 4-4 Interface "Coordinates" que permite localizar no espaço workpoints ou UCS.

Algoritmo 1. Obtenção de coordenadas (código em VB).

```
1   'oDoc refere-se a um documento do tipo Assembly
2   Dim oDoc As AssemblyDocument = AddinGlobal.InventorApp.ActiveDocument
3   'Referência ao item seleccionado na ListBox
4   Dim SelectedItem As String = cmbCoordinates.SelectedItem.ToString()
5   Dim oWP As WorkPoint 'O mesmo para UCS's
6   'Carrega o primeiro workpoint seleccionado
7   oWP = oDoc.SelectSet.Item(1)
8   'Obtém a coordenada
9   Dim wpx As Double 'O mesmo para y e z
10  wpx = Math.Round(oWP.Point.X, 3) * 10 '"*10" converte cm para mm;O mesmo para y e z
```

Algoritmo 4-1 Excerto de código: Obtenção de coordenadas de *workpoints* ou UCS.

Na robótica é frequentemente necessário representar a posição e orientação de objectos em diferentes sistemas de coordenadas o que torna interessante poder criar tais sistemas no momento da programação. De facto o *Autodesk Inventor* tem uma ferramenta que permite a criação de diferentes UCS, contudo, quando nos encontramos em ambiente *assembly* (modo de montagem próprio do *Inventor*) esta criação é difícil pois, por um lado, a única forma de o posicionarmos é através da atribuição de uma coordenada espacial e, por outro, não é permitido associá-lo a um objecto. Para ultrapassar este problema sugere-se uma ferramenta que permite para além de colocar UCS's com precisão no *assembly*, associá-los a objectos, Algoritmo 4-2. Esta ferramenta é designada por "Set UCS".

Algoritmo 2. Criação de um UCS (código em VB).

```
1   'oDoc refere-se a um documento do tipo Assembly
2   Dim oDoc As AssemblyDocument = AddinGlobal.InventorApp.ActiveDocument
3   Dim oCompDef As AssemblyComponentDefinition
4   oCompDef = oDoc.ComponentDefinition
5   'Define UCS
6   Dim oUCSDef As UserCoordinateSystemDefinition
7   oUCSDef = oCompDef.UserCoordinateSystems.CreateDefinition
8   'Verifica se existe algum workpoint seleccionado
9   Dim oWP As WorkPoint
10  oWP = oDoc.SelectSet.Item(1)
11  ' Vector com a posição do UCS
12  Dim oVectorX As Vector = AddinGlobal.InventorApp.TransientGeometry.CreateVector()
13  oVectorX.X = oWP.Point.X  ' o mesmo para Y e Z
14  'Aplica a translação, que é a origem do UCS
15  oUCSDef.Transformation.SetTranslation(oVectorX)
16  'Cria o UCS
17  Dim oUCS As UserCoordinateSystem
18  oUCS = oCompDef.UserCoordinateSystems.Add(oUCSDef)
```

Algoritmo 4-2 Excerto de código: Criação de um UCS.

A complexidade dos sistemas robóticos motivou o desenvolvimento de um novo sistema de programação de robôs baseada em CAD acessível a qualquer pessoa com conhecimentos básicos em robótica.

A abordagem proposta para a programação intuitiva de robôs está representada esquematicamente na Figura 4-5.

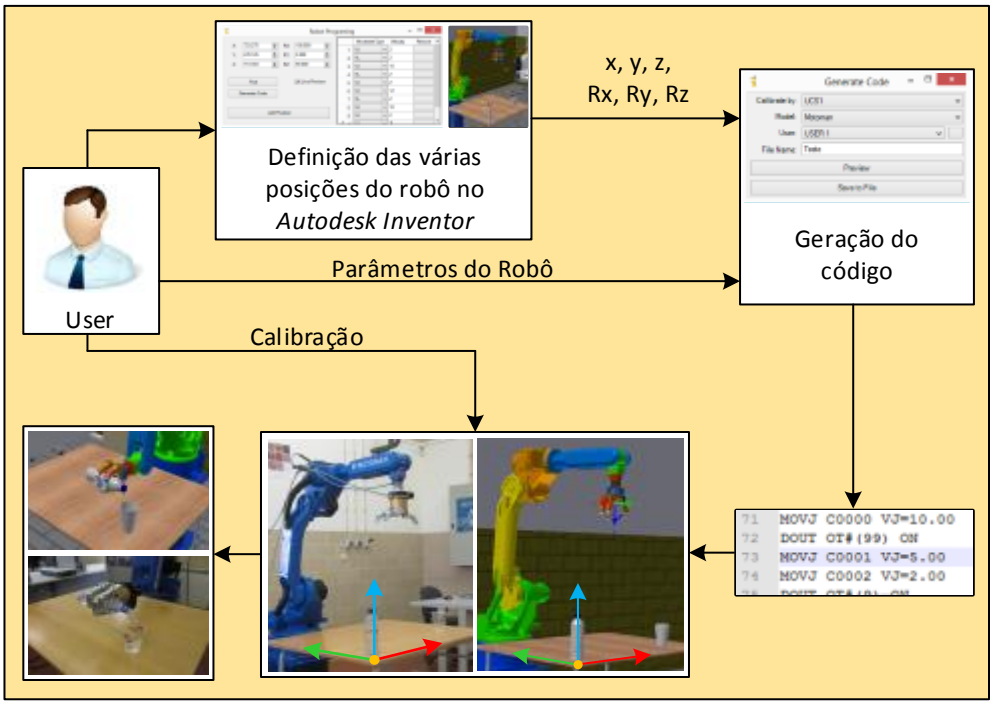


Figura 4-5 Arquitectura e funcionalidades do programa proposto.

4.3.1. Definição da Frame Tool

De um modo simplificado, em ambiente *Inventor*, cada objecto é caracterizado por um sistema de coordenadas que define a sua posição e orientação no espaço. Da mesma forma, também os robôs possuem este tipo de caracterização, nomeadamente no que respeita à orientação da sua ferramenta (frame tool), representado na Figura 4-6 como uma mão, sendo importante a coerência na orientação de ambos os sistemas de coordenadas (virtual/real). É possível ajustar o sistema de coordenadas da frame tool a partir de um *assembly* recorrendo a um método apresentado no anexo C. Uma etapa deste método passa por implementar um UCS na frame tool. Este UCS permite ao utilizador reconhecer em cada instante a orientação espacial da frame tool.

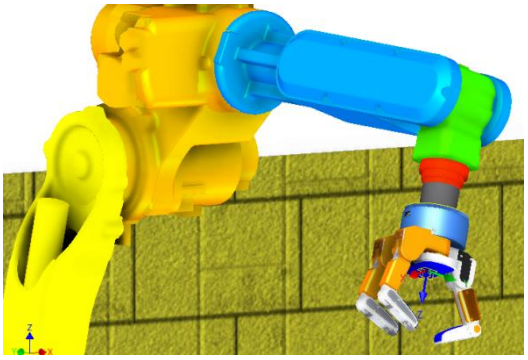


Figura 4-6 Modelo CAD da frame tool.

4.4. Interacção com o Robô

No *Autodesk Inventor* é possível orientar objectos no espaço através da função *drag* (arrastar), contudo quando se trata da movimentação do robô no espaço esta tarefa já não é assim tão simples pelo facto de se ter de lidar com vários graus de liberdade levando a que por vezes o robô adquira posições irreais, Figura 4-7**.**

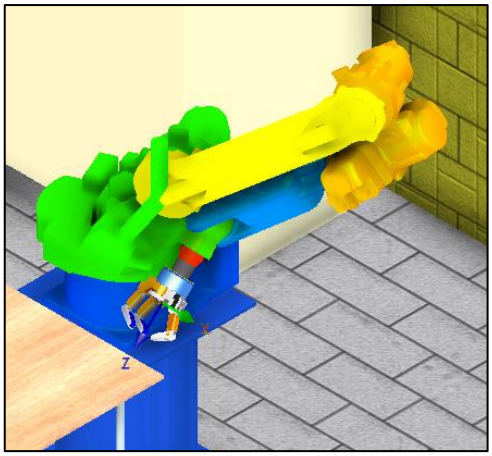


Figura 4-7 Exemplo de uma posição do robô inválida.

De modo a ultrapassar este problema propõe-se uma ferramenta, Figura 4-8, que permite movimentar o robô de um modo controlado onde o utilizador apenas tenha de se preocupar com a posição e orientação da frame tool, Figura 4-9.

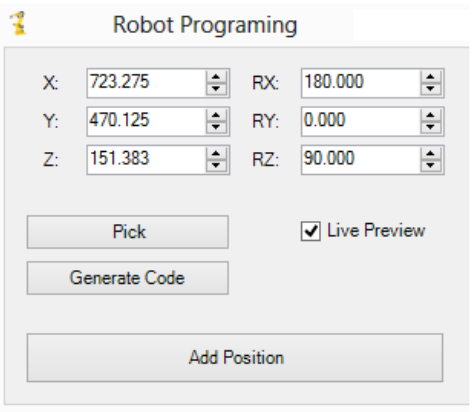


Figura 4-8 Interface "*Robot Programing*" que permite movimentar o robô.

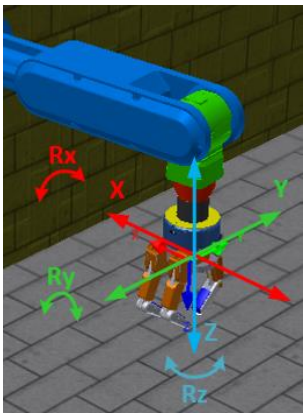


Figura 4-9 Esquema dos movimentos possíveis da frame tool.

Para isso, define-se que a frame tool é hierarquicamente responsável por comandar todas as outras partes do robô, isto é, as partes do robô adaptam-se á posição da frame tool. Anteriormente referiu-se, em contexto figurativo, que no *Inventor* todos os objectos são caracterizados por sistemas de coordenadas, mas na realidade são representados na forma matricial sendo os seus valores relativos ao sistema de coordenadas do *assembly*, Figura 4-10.

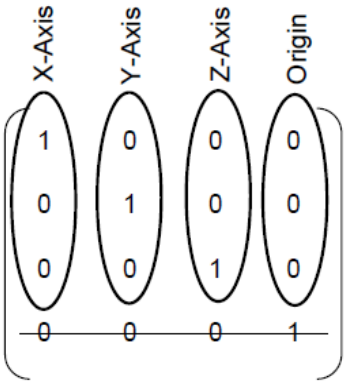


Figura 4-10 Matriz que descreve um objecto no espaço.

O reconhecimento da frame tool e de objectos em geral é realizado através do Algoritmo 4-3 que é chamado cada vez que um objecto é seleccionado através do botão "Pick".

Este algoritmo trata de carregar e disponibilizar todas as propriedades do objecto seleccionado (no nosso caso a frame tool) a outros algoritmos necessários para o funcionamento do programa.

Algoritmo 3. Interacção com objectos (código em VB).

```
1   Dim oDoc As AssemblyDocument = AddinGlobal.InventorApp.ActiveDocument
2   bStillSelecting = True
3   ' Criação de uma interacção de eventos com o objecto.
4   Dim oInteractEvents = AddinGlobal.InventorApp.CommandManager.CreateInteractionEvents
5   ' Assegura que a interacção está activa.
6   oInteractEvents.InteractionDisabled = False
7   ' Define uma referência ao evento seleccionado.
8   oSelectEvents = oInteractEvents.SelectEvents
9   oSelectEvents.SingleSelectEnabled = True
10  ' Define o filtro através de valor.
11  oSelectEvents.AddSelectionFilter(SelectionFilterEnum.kAssemblyOccurrenceFilter)
12  ' Inicia a interacção de eventos com o objecto.
13  oInteractEvents.Start()
14  ' Torna a janela invisível enquanto não estiver nenhum objecto seleccionado
15  While bStillSelecting
16   Me.Visible = False
17   AddinGlobal.InventorApp.UserInterfaceManager.DoEvents()
18  End While
19  ' Torna a janela visível
20  Me.Visible = True
21  ' Para a interacção de enventos com objectos.
22  oInteractEvents.Stop()
23  ' Limpeza de memória.
```

Algoritmo 4-3 Excerto de código: Interacção com objectos (no nosso caso frame tool).

O Algoritmo 4-4 é responsável pela translação da frame tool no espaço através da equação 3.1. Esta transformação consiste na actualização dos valores da última coluna da matriz da frame tool, coluna esta que representa a sua origem, Figura 4-10. Estes cálculos são sempre em relação á origem do *assembly*.

Algoritmo 4. Translação da frame tool (código em VB).

```
1   Dim oTG As TransientGeometry = AddinGlobal.InventorApp.TransientGeometry
2   Dim oTransformMatrix As Matrix = oTG.CreateMatrix
3   ' Matriz identidade
4   oTransformMatrix.SetToIdentity()
5   Dim oTmpMatrix As Matrix = oTG.CreateMatrix
6   ' Adquire o corrente estado ground do objecto seleccionado.
7   Dim groundState As Boolean = selectedEntity.Grounded
8   ' Muda o estado do objecto para Ground.
9   selectedEntity.Grounded = True
10  ' Actualização dos valores da matriz de transformação.
11  oTmpMatrix = selectedEntity.Transformation
12  oTmpMatrix.SetTranslation(oTG.CreateVector(Me.NumericX.Value,Me.NumericY.Value,_
13  Me.NumericZ.Value))
14  selectedEntity.Transformation = oTmpMatrix
15  ' Reset do estado ground do objecto para o seu estado original.
16  selectedEntity.Grounded = groundState
```

Algoritmo 4-4 Excerto de código: Translação da frame tool.

Como apresentado na secção 3.2. a orientação de objectos no espaço pode ser realizada com recurso aos ângulos de Euler. À medida que o utilizador incrementa os ângulos de rotação que os eixos ordenados da frame tool fazem com os do *assembly* é actualizada a sua matriz de rotação através do Algoritmo 4-5.

Algoritmo 5. Rotação da frame tool (código em VB).

```
1   Dim oTG As TransientGeometry = AddinGlobal.InventorApp.TransientGeometry
2   Dim dPi As Double = Math.Atan(1) * 4
3   ' Adquire o corrente estado ground do objecto seleccionado.
4   Dim groundState As Boolean = selectedEntity.Grounded
5   ' Muda o estado do objecto para Ground.
6   selectedEntity.Grounded = True
7   ' Referência ao zero do objecto
8   Dim oPartOrigin As Point = oTG.CreatePoint(0, 0, 0)
9   Dim oTransMatrix As Matrix = selectedEntity.Transformation
10  ' Defenição de uma matriz identidade 3x3
11  oPartOrigin.TransformBy(oTransMatrix)
12  oTransMatrix.Cell(1, 1) = 1
13  oTransMatrix.Cell(2, 1) = 0
14  oTransMatrix.Cell(3, 1) = 0
15  oTransMatrix.Cell(1, 2) = 0
16  oTransMatrix.Cell(2, 2) = 1
17  oTransMatrix.Cell(3, 2) = 0
18  oTransMatrix.Cell(1, 3) = 0
19  oTransMatrix.Cell(2, 3) = 0
20  oTransMatrix.Cell(3, 3) = 1
21  'Transformação (rotação) da peça seleccionada (relativamente à origem do assembly)
22  Dim oMatrix As Matrix = oTG.CreateMatrix()
23  oMatrix.SetToRotation((NumericRX.Value * dPi) / 180.0, oTG.CreateVector(1, 0, 0),_
24  oPartOrigin)'O mesmo para Y e para Z
25  oTransMatrix.TransformBy(oMatrix)
26  ' Aplicação da transformação
27  selectedEntity.Transformation = oTransMatrix
28  ' Reset do estado ground do objecto para o seu estado original.
29  selectedEntity.Grounded = groundState
```

Algoritmo 4-5 Excerto de código: Rotação da frame tool.

Suponhamos que o utilizador estaria a movimentar o robô através desta ferramenta, mas que por algum motivo recorresse à ferramenta *drag* para complementar esse movimento, os valores dispostos nas *combobox's* (*X*, Y, Z, Rx, Ry e Rz) que informam o utilizador da posição e orientação da frame tool ficariam desactualizados. Para prevenir isso é utilizado o Algoritmo 4-6 sempre que tal aconteça. Este algoritmo é responsável por fazer todas as transformações relativas à origem do *assembly* e de, posteriormente, actualizar os valores nas *combobox*.

Algoritmo 6. Update (código em VB).

```
1   Dim dPi As Double = Math.Atan(1) * 4
2   'Adquire o novo vector posição do objecto seleccionado.
3   initializing = True
4   Me.NumericX.Value = selectedEntity.Transformation.Translation.X
5   Me.NumericY.Value = selectedEntity.Transformation.Translation.Y
6   Me.NumericZ.Value = selectedEntity.Transformation.Translation.Z
7   initializing = False
8   Dim oTransform As Inventor.Matrix = selectedEntity.Transformation
9   'Adquire os novos valores de rotação do objecto seleccionado através da aplicação
10  'das equações 3.24, 3.25, 3.26
11  initializing = True
12  Me.NumericRX.Value = (Math.Atan2(oTransform.Cell(3,2),oTransform.Cell(3,3))*180)/dPi
13  Me.NumericRY.Value = (Math.Atan2(-oTransform.Cell(3, 1),Math.Sqrt(Math.Pow(_
14  oTransform.Cell(3,2),2)+Math.Pow(oTransform.Cell(3,3),2)))*180)/dPi
15  Me.NumericRZ.Value = (Math.Atan2(oTransform.Cell(2,1),oTransform.Cell(1,1))*180)/dPi
16  initializing = False
```

Algoritmo 4-6 Excerto de código: Update; actualiza os valores da posição e orientação dos objectos.

Em alternativa, para a movimentação da frame-tool, pode-se recorrer a uma função disponível no *Inventor* designada por *UCS constrain* (restrição de UCS) que permite fazer coincidir a orientação do UCS da frame tool com um outro UCS previamente colocado no espaço, este método é descrito a pormenor no ANEXO D. Esta funcionalidade é útil para situações muito específicas que requerem elevada precisão, Figura 4-11.

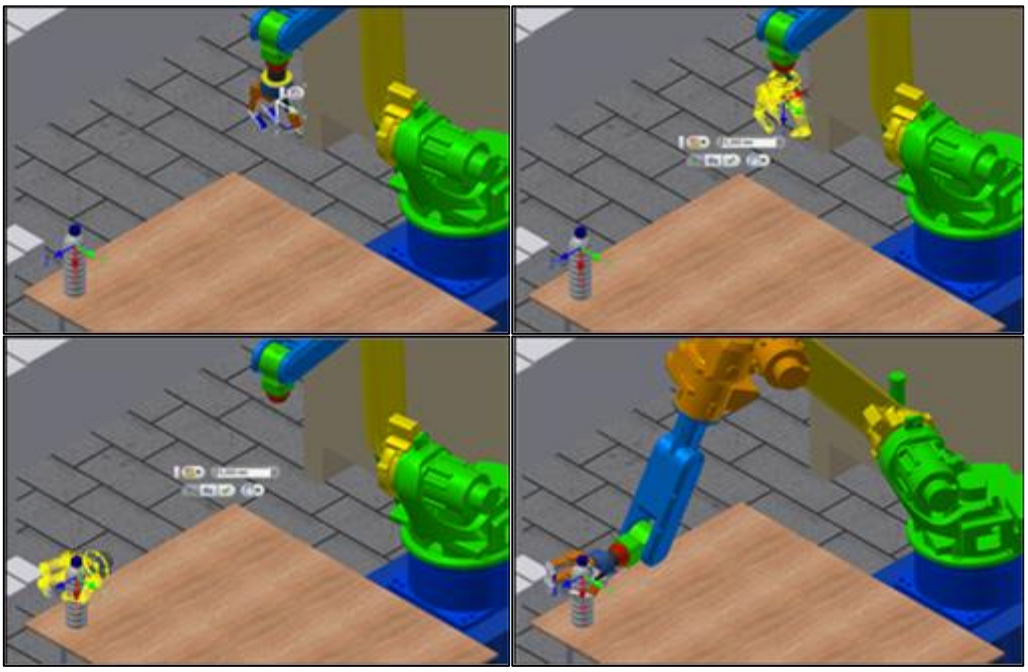


Figura 4-11 Método de movimentar o robô através de *constraints*

4.5. Definição das tarefas do Robô

A partir do momento em que a frame tool é seleccionada cabe ao utilizador definir as posições que quer que o robô assuma. Para isso, apenas tem de movimentar a frame tool no espaço através das ferramentas discutidas na secção anterior e adicionar a sua posição à lista através do botão "Add Position", Figura 4-12.

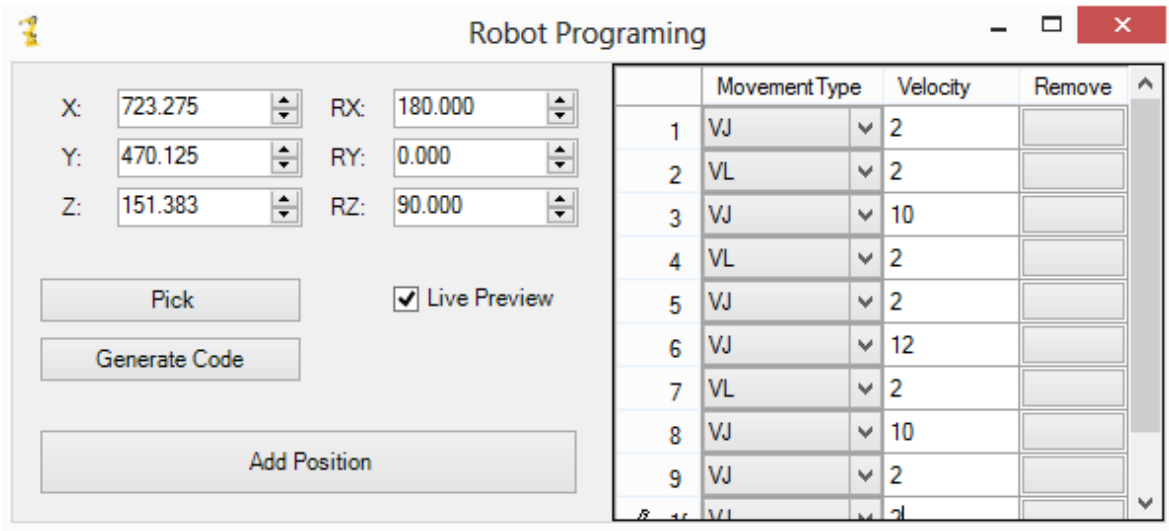


Figura 4-12 Interface "*Robot Programing*" que permite movimentar, programar e simular.

Cada vez que este botão é pressionado é colocado em memória, em forma de lista, a matriz que descreve a frame tool no espaço relativamente ao UCS do *assembly* nesse instante através do Algoritmo 4-7.

Algoritmo 7. Guardar a posição de objectos (código em VB).

```
1   If Not selectedEntity Is Nothing Then
2     Dim stepPoint As StepPoint = New StepPoint()
3     Dim dPi As Double = Math.Atan(1) * 4
4     'Guarda em memória os valores da posição actual do objecto.
5     stepPoint.TX = selectedEntity.Transformation.Translation.X'O mesmo para Y e para Z
6     'Guarda em memória os valores da orientação actual do objecto.
7     Dim oTransform As Inventor.Matrix = selectedEntity.Transformation
8     stepPoint.RX = (Math.Atan2(oTransform.Cell(3, 2), oTransform.Cell(3, 3))*180)/dPi
9     stepPoint.RY = (Math.Atan2(-oTransform.Cell(3, 1), Math.Sqrt(Math.Pow(_
10    oTransform.Cell(3, 2), 2) + Math.Pow(oTransform.Cell(3, 3), 2))) * 180) / dPi
11    stepPoint.RZ = (Math.Atan2(oTransform.Cell(2, 1), oTransform.Cell(1, 1))*180)/dPi
12    Me.StepPointDataGridView.Rows(Me.StepPointBindingSource.Add(stepPoint)).Selected_
13                                = True
14    Dim oRotTransform As Inventor.Matrix = selectedEntity.Transformation
15    stepPoint.R11 = oRotTransform.Cell(1, 1)
16    stepPoint.R12 = oRotTransform.Cell(1, 2)
17    stepPoint.R13 = oRotTransform.Cell(1, 3)
18    stepPoint.R21 = oRotTransform.Cell(2, 1)
19    stepPoint.R22 = oRotTransform.Cell(2, 2)
20    stepPoint.R23 = oRotTransform.Cell(2, 3)
21    stepPoint.R31 = oRotTransform.Cell(3, 1)
22    stepPoint.R32 = oRotTransform.Cell(3, 2)
23    stepPoint.R33 = oRotTransform.Cell(3, 3)
24  End If
```

Algoritmo 4-7 Excerto de código: Guarda a posição de objectos do *Autodesk Inventor.*

É nesta fase que é definido o tipo de movimento e a velocidade que o robô vai assumir em cada ponto, Figura 4-12.

4.5.1. Simulação

O facto das matrizes, que descrevem a frame tool no espaço relativamente ao UCS do *assembly*, estarem armazenados em memória em forma de lista permite que, seleccionando a *check list* "*Live Preview*", se possa rever todas as posições que o utilizador definiu e se seja conveniente alterar. Da mesma forma o utilizador pode saltar de posição para posição e analisar com detalhe, prevenindo eventuais colisões do robô com objectos.

4.6. Programação do Robô

A programação do robô é definida por três fases distintas uma de calibração, outra de definição de parâmetros e finalmente a geração do código. Estas três fases são descritas de seguida em maior detalhe.

4.6.1. Calibração

Em etapas anteriores gerou-se as posições espaciais da frame tool do robô, no entanto estas posições são relativas a um sistema de coordenadas do *assembly* que na verdade poderá estar em qualquer lugar, no nosso caso está representado na Figura 4-13 por frame $\{A\}$.

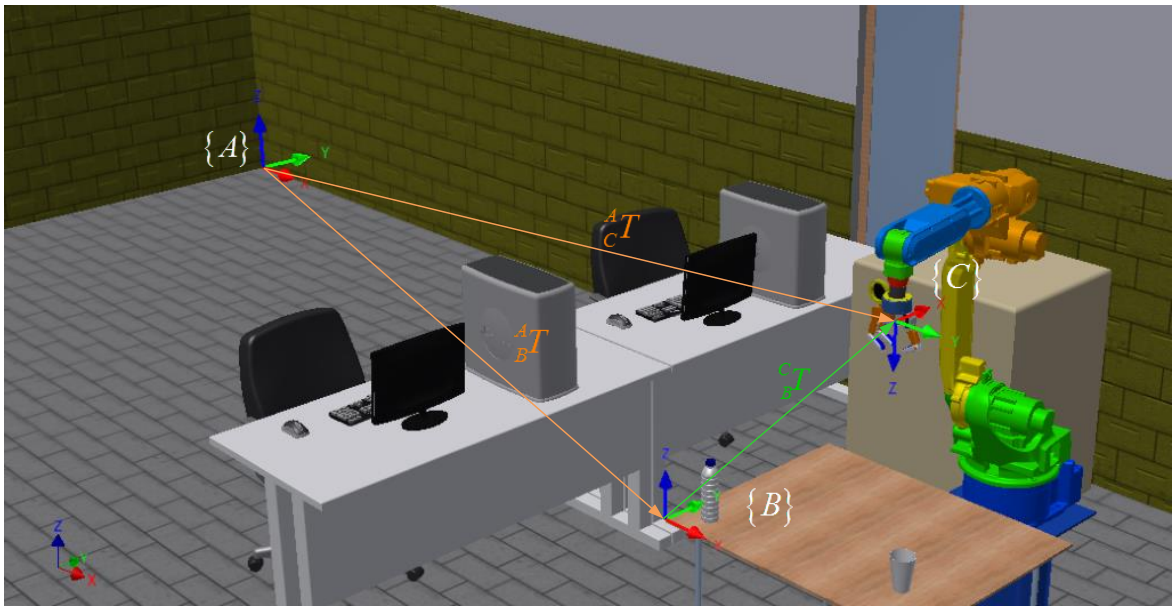


Figura 4-13 Frames compostas: Representação da abordagem proposta.

Todas as posições e orientações da frame tool do robô, frame $\{C\}$, extraídas do *Inventor* têm de ser conhecidas relativamente a um sistema de coordenadas de referência conhecido no espaço real. Para tal cabe ao utilizador definir um sistema de coordenadas (UCS), frame $\{B\}$, no modelo CAD e posteriormente comunicar ao robô

onde este se encontra através do *teach pendent*. Recorde-se que para a definição de um UCS no *assembly* recorre-se à ferramenta desenvolvida referida na secção 3.6, *Set UCS*.

Para o cálculo destas transformações compostas propõe-se o Algoritmo 4-8 que tem como referência as equações indicadas na secção 3.1.3..

A matriz de transformação da frame $\{B\}$ relativamente à frame $\{A\}$, ${}^{A}_{B}T$, é obtida através da API do *Inventor*. Contudo, nós queremos saber a matriz de transformação da frame $\{C\}$ relativamente à frame $\{B\}$, ${}^{B}_{C}T$. Note-se que a frame $\{C\}$ é uma frame comum entre o modelo virtual e o real. Para obtermos ${}^{B}_{C}T$ recorremos à equação 3.14,

$$ {}^{B}_{C}T = {}^{B}_{A}T\,{}^{A}_{C}T. \tag{4.1} $$

Para descobrir ${}^{B}_{A}T$ ou calculamos a matriz de rotação que define a frame $\{A\}$ relativamente a frame $\{B\}$, ${}^{B}_{A}R$, e o vector que define a origem da frame $\{A\}$ relativamente a $\{B\}$, ${}^{B}P_{AORG}$:

$$ {}^{B}_{A}T = \left[\begin{array}{ccc|c} & {}^{B}_{A}R & & {}^{B}P_{AORG} \\ 0 & 0 & 0 & 1 \end{array}\right], \tag{4.2} $$

ou calculamos através da API do *Inventor* a inversa de ${}^{B}_{A}T$:

$$ {}^{B}_{C}T = ({}^{A}_{B}T)^{-1}. \tag{4.3} $$

Seja ${}^{A}P$, um vector/ponto definido na frame $\{A\}$. Se quisermos expressar este ponto em relação a $\{B\}$ temos de calcular,

$$ {}^{B}P = {}^{B}_{A}R\ {}^{A}P + {}^{A}P_{BORG}. \tag{4.4} $$

Como a matriz de rotação característica, ${}^{B}_{A}R = {}^{A}_{B}R^{T}$,e como conhecemos ${}^{A}_{B}T$, podemos calcular ${}^{B}P_{AORG}$ invertendo a matriz,

$$ {}^{B}_{A}T = \left[\begin{array}{ccc|c} & {}^{A}_{B}R^{T} & & -{}^{A}_{B}R^{T}\,{}^{A}P_{BORG} \\ 0 & 0 & 0 & 1 \end{array}\right]. \tag{4.5} $$

Conjugando as equações (4.2) e (4.5) obtemos,

$$ {}^{B}P_{UORG} = -{}^{A}_{B}R^{T}\,{}^{A}P_{BORG}. \tag{4.6} $$

Recorrendo às equações (4.1) e (4.5) ou (4.1) e (4.3) finalmente obtemos ${}^{B}_{C}T$.

Algoritmo 8. Transformações compostas (código em VB).

```
1   'Trata a Matriz transformação do objecto para cada ponto relativamente ao UCS
2   'seleccionado
3   For Each stepPoint As StepPoint In mStepPointList
4       ' Transformação do vector posição relativamente ao UCS "BAT"
5       Dim transf_invert As Matrix = selectedUCS.Transformation
6       transf_invert.Translation.X = (selectedUCS.Transformation.Cell(1, 4))
7       transf_invert.Translation.Y = (selectedUCS.Transformation.Cell(2, 4))
8       transf_invert.Translation.Z = (selectedUCS.Transformation.Cell(3, 4))
9       Dim x_trans As Double
10      x_trans = transf_invert.Translation.X * (-1)'O mesmo para Y e Z
11      transf_invert.Cell(1, 4) = x_trans
12      transf_invert.Cell(2, 4) = y_trans
13      transf_invert.Cell(3, 4) = z_trans
14     'Constroi a matriz de rotação que define a orientação do Objecto no espaço
15      'relativamente ao sistema de coordenadas do assembly, e.g. .Cell(1, 1)
16      Dim testeRotMatrix As Inventor.Matrix = oTG.CreateMatrix
17      testeRotMatrix.Cell(1, 1) = stepPoint.R11
18     'Cálculo da inversa de "BAT" -> "ABT"
19      Dim transf_invert_rot As Matrix = selectedUCS.Transformation
20      transf_invert_rot.Invert()
21     'Definição de "ACT" e "ABT"
22      Dim matrix_A As Double()() = New Double(3)() {}
23      Dim matrix_B As Double()() = New Double(3)() {}
24      For i As Integer = 0 To 3
25          matrix_A(i) = New Double(3) {}
26          matrix_B(i) = New Double(3) {}
27      Next
28      For i = 0 To 3
29      For j = 0 To 3
30          matrix_A(i)(j) = testeRotMatrix.Cell(i + 1, j + 1)
31          matrix_B(i)(j) = transf_invert_rot.Cell(i + 1, j + 1)
32      Next
33      Next
34      Dim ACT As New GeneralMatrix(matrix_A)
35      Dim BAT As New GeneralMatrix(matrix_B)
36      Dim BCT As GeneralMatrix
37     'Produto de "ACT" com "ABT" = "CBT"
38      BCT = BAT * ACT
39     'Matriz BCT, e.g. .Array(0)(0)
40      Dim matrix_rot(,) As Double = New Double(3, 3) {}
41      matrix_rot(0, 0) = BCT.Array(0)(0) '[.Array(0)(0) (...) .Array(2)(2)]
42      'Ângulos de Euler XYZ
43      Dim array_angle() As Double = New Double(2) {}
44      Dim array_ang() As Double = New Double(2) {}
45      Dim ft As New Euler_Angles.EULER_ang
46      array_angle = ft.Rotation_XYZ(matrix_rot)
47     array_ang(0) = Math.Round(array_angle(0), 2)
48     array_ang(1) = Math.Round(array_angle(1), 2)
49     array_ang(2) = Math.Round(array_angle(2), 2)
50     BCT.Array(0)(3) = Math.Round(BCT.Array(0)(3), 3) * 10
51     BCT.Array(1)(3) = Math.Round(BCT.Array(1)(3), 3) * 10
52     BCT.Array(2)(3) = Math.Round(BCT.Array(2)(3), 3) * 10
53   Next
```

Algoritmo 4-8 Excerto de código: Transformações compostas.

4.6.2. Definição de parâmetros

De modo a permitir a geração de código para vários modelos de robô, propõe-se uma interface, Figura 4-14, que permite ao utilizador definir alguns parâmetros como: o tipo de modelo de código (*Motoman*, *ABB*, *Fanuc* ou *Kuka*), definição do nome do ficheiro que se quer gerar, definição do sistema de coordenadas a que o robô se vai referir (UCSi; $i \in \mathbb{R}$) e o nome a que se encontra associado a ele no robô (USERi; $i \in \mathbb{R}$) (calibração).

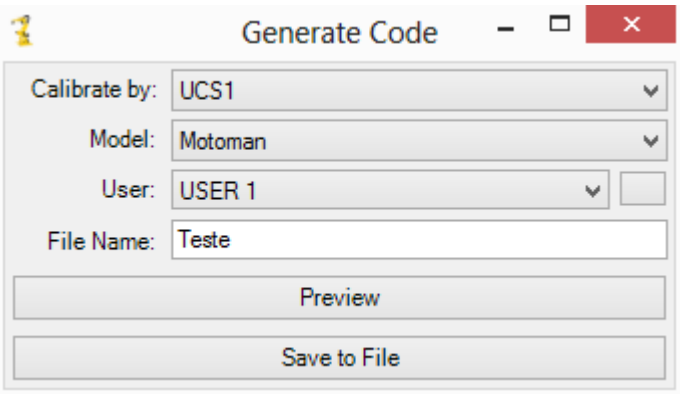


Figura 4-14 Interface "Generate Code" que permite parametrizar o código que se quer gerar.

4.6.3. Geração de código

O Algoritmo 4-9 é responsável pela geração do código do robô respeitando os diferentes parâmetros das diferentes marcas de robôs. Este algoritmo recorre às informações recolhidas em etapas anteriores para compor o código. Contudo, devido à extensividade do algoritmo apenas se demonstra a parte responsável por programar um robô da marca *Motoman*.

Algoritmo 9. Geração de código do robô - Motoman (código em VB).

```
'O mesmo princípio é aplicado para ABB, Fanuc e Kuka
For Each stepPoint As StepPoint In mStepPointList
    '********Algoritmo 4-8********
    Dim Rot As Inventor.Point = oTG.CreatePoint(array_ang(0),array_ang(1),_
                                                array_ang(2))
    Dim point As Inventor.Point = oTG.CreatePoint(UFT.Array(0)(3), UFT.Array(1)(3),_
                                                  UFT.Array(2)(3))
    Dim pointRef As String = "P" + pointCount.ToString("0000")
    Dim pointRefMov As String = "P" + pointCount.ToString("000")
    'Enumeração dos pontos de entrada
    pointCount = pointCount + 1
    dictionary.Add(pointRef, point)
    dictionaryR.Add(pointRef, Rot)
    instructionOutput.AppendLine("MOVJ " + pointRefMov + " " +stepPoint._
                                 MovementType.ToString + "=" +stepPoint.Velocity._
                                 ToString("0.00"))
Next
' Trata e imprime a velocidade e o tipo de movimento
Dim pair As KeyValuePair(Of String, Inventor.Point)
For Each pair In dictionary
    Dim rots As Inventor.Point = dictionaryR.Item(pair.Key)
    'Imprime a linha que contém "P0001 = TX, TY, TZ, RX, RY, RZ"
    pointOutput.AppendLine(pair.Key.ToString + "=" + pair.Value.X.ToString + "," +_
                           pair.Value.Y.ToString + "," + pair.Value.Z.ToString +_
                           "," + rots.X.ToString + "," + rots.Y.ToString + "," +_
                           rots.Z.ToString)
Next
'Escreve o intro do ficheiro Motoman
Dim SelectedUser As String = Me.cmbUser.SelectedItem
Dim ProjectName As String = Me.txtFileName.Text
ProjectName = ProjectName.Replace(" ", "")
Dim Intro As StringBuilder = New StringBuilder()
Intro.AppendLine("/JOB" & vbCrLf & "//NAME " + ProjectName.ToString.ToUpper &_
                 vbCrLf &"//POS" & vbCrLf & "///NPOS 0,0,0," + pointCount.ToString+_
                 ",0,0" & vbCrLf & "///" + SelectedUser.ToString & vbCrLf &_
                 "///TOOL 0" & vbCrLf & "///POSTYPE USER" & vbCrLf & "///RECTAN" &_
                 vbCrLf & "///RCONF 0,0,0,0,0,0,0,0")
'Escreve o corpo do ficheiro Motoman
Dim Midle As StringBuilder = New StringBuilder()
Midle.AppendLine("//INST" & vbCrLf & "///DATE 2013/07/20 11:26" & vbCrLf & "///ATTR_
                 "SC,RW" & vbCrLf & "///GROUP1 RB1" & vbCrLf & "NOP")
'Escreve o Fim do ficheiro Motoman
Dim finale As StringBuilder = New StringBuilder()
finale.AppendLine("END")
'Compila todos os valores que estão em memoria e escreve por esta ordem
Return Intro.ToString + pointOutput.ToString + Midle.ToString + instructionOutput._
       ToString + finale.ToString
```

Algoritmo 4-9 Excerto de código: Geração de código do robô.

O código gerado, pode ser salvo através do botão *Save to File*; o utilizador tem ainda a opção de pré visualizar o código através do botão *Preview*, Figura 4-15.

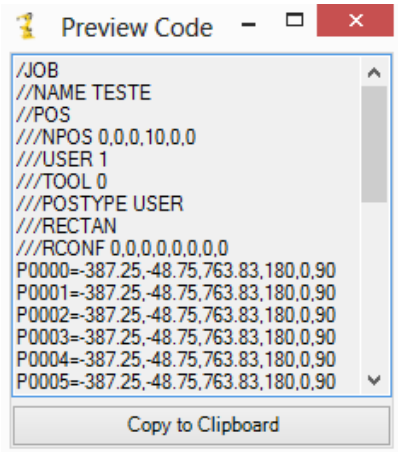


Figura 4-15 Interface “Preview Code” que permite pré-visualizar o código gerado.

5. ESTUDO DE CASO

De forma a provar o conceito, propôs-se um desafio onde se pretende encher um copo com água através de um robô programado *off-line* com recurso ao *add-in* desenvolvido para esta dissertação. Estes dois elementos, o copo e a garrava de água, estão dispostos em cima de uma mesa que se encontra em frente ao robô, Figura 5-1. A frame tool do robô é uma mão mecânica que é responsável pelo transporte da garrafa.

Figura 5-1 Disposição do robô e dos objectos em estudo.

5.1. Implementação e resultados

A primeira fase deste desafio foi modelar em CAD tanto os objectos como o espaço onde se encontram (laboratório de robótica da FCTUC), Figura 5-2.

Figura 5-2 Comparação do modelo em estudo virtual com o real.

Seguidamente caracterizou-se o sistema de coordenadas da frame tool, Figura 5-3, processo que teve a duração de dois minutos.

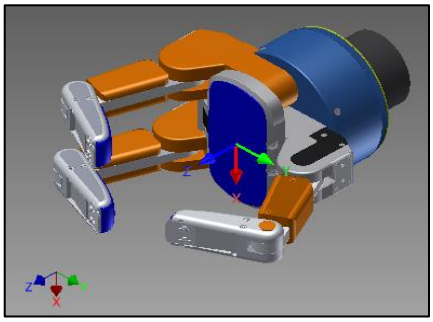


Figura 5-3 Modelo CAD da frame tool calibrado.

A definição das tarefas do robô Figura 5-4 levou cerca de oito minutos a realizar tendo a tarefa mais delicada (aproximação da frame tool à garrafa) demorado sensivelmente dois minutos, Figura 5-5.

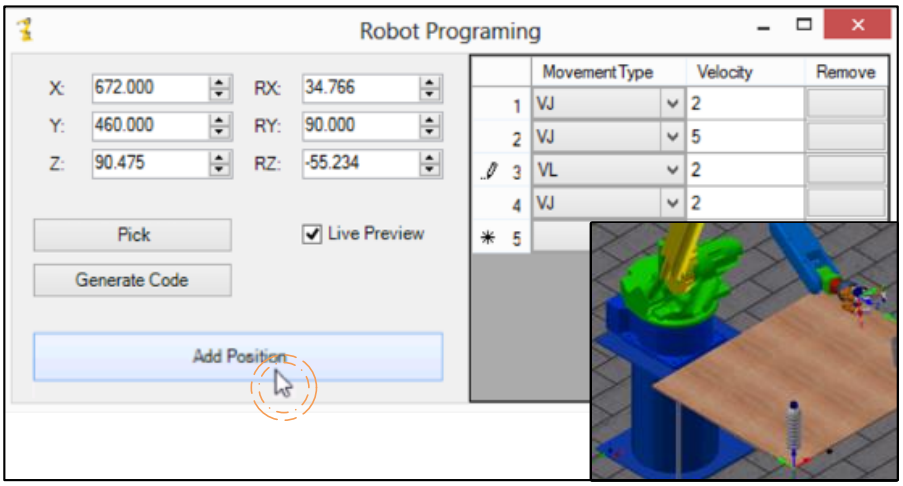


Figura 5-4 Exemplo da definição das tarefas do robô.

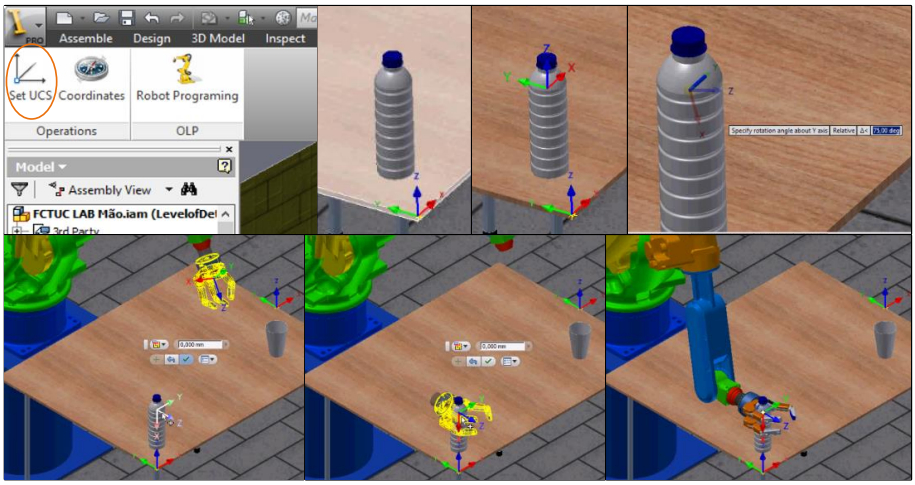


Figura 5-5 Procedimento utilizado para definição da tarefa de aproximação da frame tool à garrafa.

Na Figura 5-6 está representado o trajecto que se impôs ao robô: aproximar-se da garrafa, agarrá-la, transportá-la até ao ponto de despejo, encher o copo e fazer o processo inverso até voltar á posição inicial.

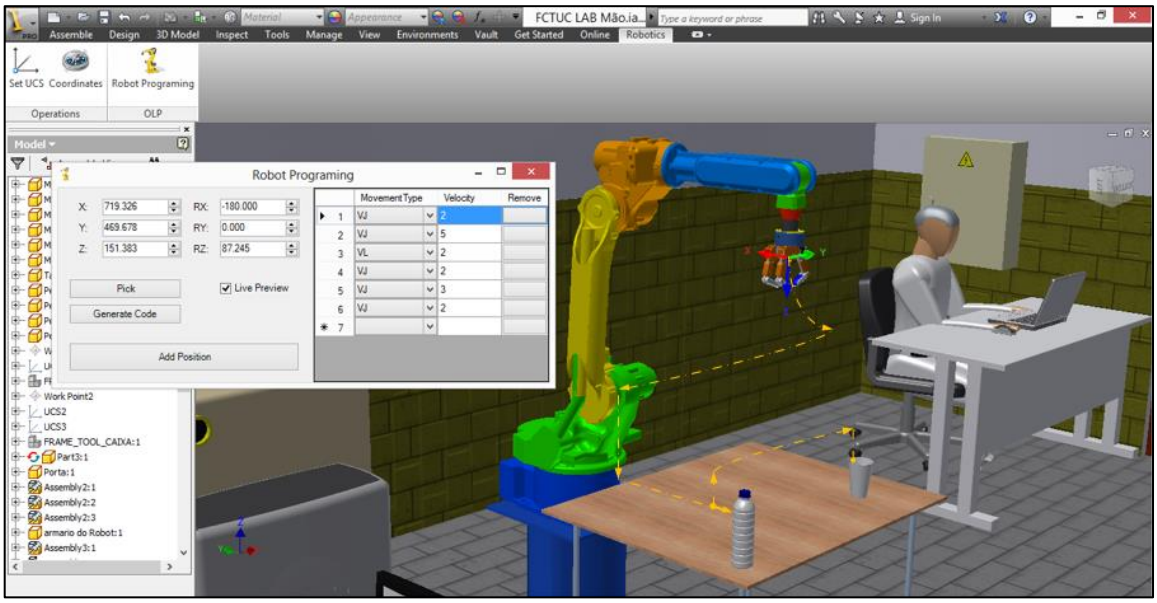


Figura 5-6 Representação da trajectória do robô.

Depois de se definir os parâmetros, Figura 5-7**,** e se gerar o código, importou-se o programa para o robô, Figura 5-8.

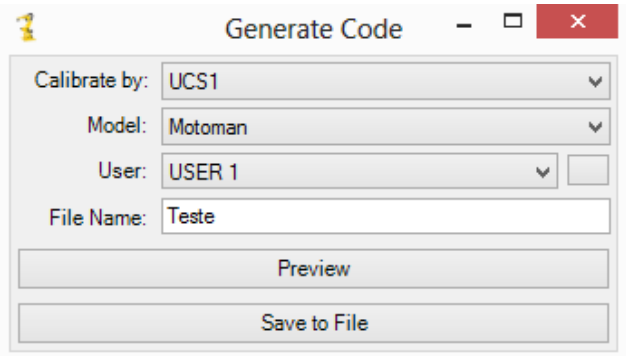


Figura 5-7 Parametrização do programa do robô.

```
63  C0055=188999,-46656,-9603,0,
64  C0056=106189,-72136,-9603,0,25477,-290
65  C0057=-142241,-72136,-9603,0,25477,-290
66  //INST
67  ///DATE 2013/04/10 11:37
68  ///ATTR SC,RW
69  ///GROUP1 RB1
70  NOP
71  MOVJ C0000 VJ=10.00
72  DOUT OT#(99) ON
73  MOVJ C0001 VJ=5.00
74  MOVJ C0002 VJ=2.00
75  DOUT OT#(9) ON
```

Figura 5-8 Fragmento do código gerado pelo software desenvolvido.

Os resultados da experiência foram excelentes não tendo havido necessidade de qualquer ajuste ou correcção. Na Figura 5-9, Figura 5-10 e Figura 5-11 representa-se os vários estágios do robô neste processo.

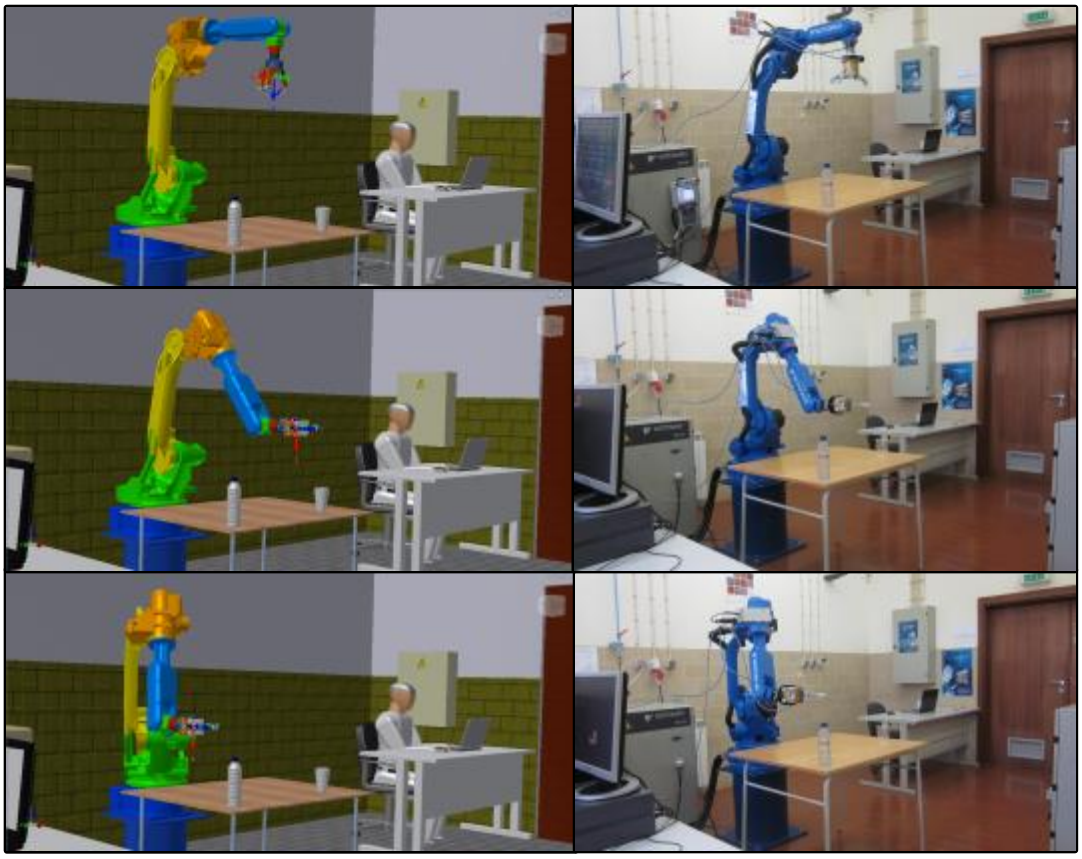


Figura 5-9 Comparação do modelo CAD com o modelo real: Robô a corre o programa gerado.

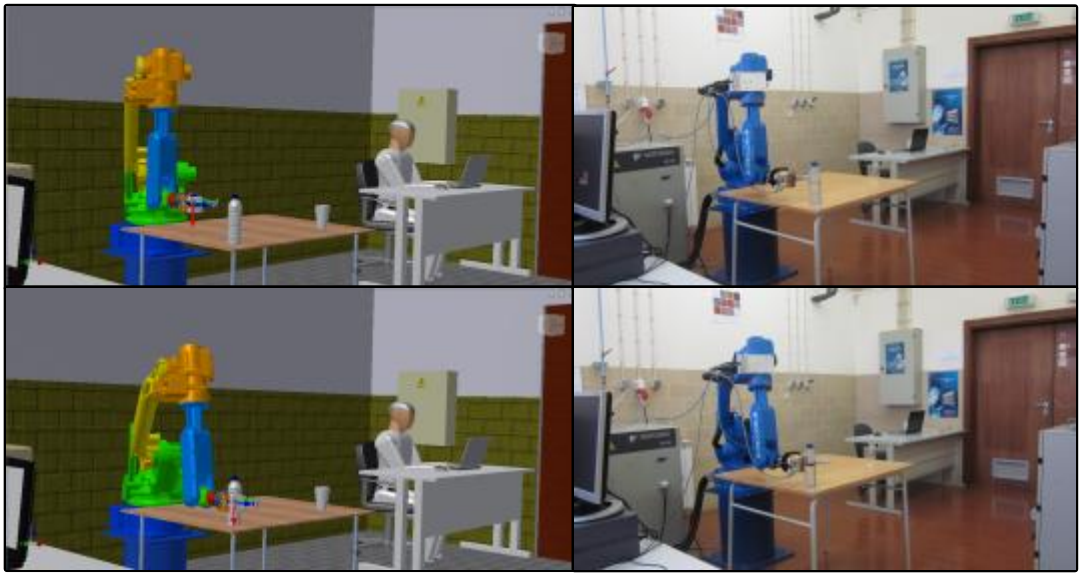


Figura 5-10 Comparação do modelo CAD com o modelo real: Robô a corre o programa gerado.

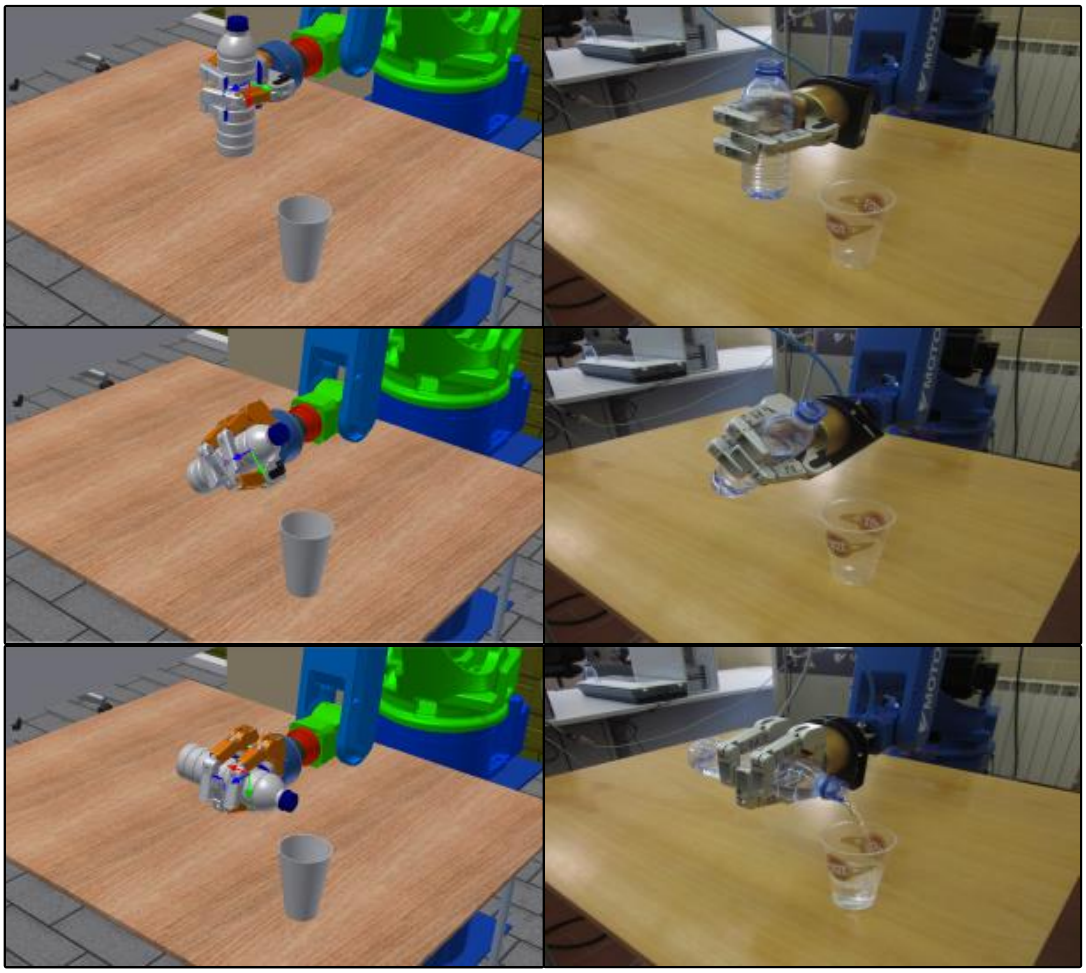


Figura 5-11 Comparação do modelo CAD com o modelo real: Robô a corre o programa gerado.

6. CONCLUSÕES

Nesta dissertação foi apresentada uma nova plataforma de OLP baseada em CAD onde a célula robótica e a programação *off-line* de robôs estão integradas na mesma interface gráfica, com o recurso a um *software* de CAD frequentemente utilizado, *Autodesk Inventor*. Foi proposto um método para extrair, dos modelos, CAD posições e orientações do robô. Depois de tratados e transformados, estes dados permitiram desenvolver outros métodos como a simulação, que possibilita antever o percurso do robô em ambiente virtual prevenindo eventuais erros na definição de trajectórias, e a geração de programas de robô para diferentes marcas.

Os diferentes testes realizados nesta aplicação provaram que os objectivos propostos foram conseguidos:

- Solução acessível a PME's;
- Solução de rápido *setup*;
- Capaz de programar diferentes marcas de robôs;
- Solução intuitiva e de fácil utilização;
- Capaz de ser implementada por um utilizador sem conhecimentos em robótica ou em programação;
- Solução que trabalha com base num modelo CAD;
- Capaz de simular as posições do robô;
- Dispõe de ferramentas de ajuste fino.

6.1. Trabalho Futuro

Existem algumas ferramentas que podem ser desenvolvidas no futuro. Sugere-se a criação de uma biblioteca de robôs que permita reduzir o tempo de *setup*, onde o utilizador possa importar para o modelo CAD os robôs já modelados. Outra ferramenta relaciona-se com o melhoramento da interacção entre o simulador e o utilizador. Esta interacção pode ser melhorada com o desenvolvimento de métodos de geração de trajectórias que permitam identificar o percurso que o robô vai tomar de ponto de trabalho

a ponto de trabalho. Outras funcionalidades interessantes que poderiam ser desenvolvidas têm a ver com a implementação de algoritmos de pintura e/ou soldadura que, por exemplo, definissem a trajectória e ângulos de ataque necessários para pintar/soldar uma superfície.

REFERÊNCIAS BIBLIOGRÁFICAS

Bruccoleri, M. and D'Onofrio, C. and La Commare, U. (2007), Industrial Informatics, 2007 5th IEEE International Conference on, Off-line Programming and simulation for automatic robot control software generation. DOI : 10.1109/INDIN.2007.4384806

Craig, J. J. (2004). Introduction to robotics: mechanics and control, Prentice Hall, 3rd Edition, New Jersey

David J. Kasik, William Buxton, and David R. Ferguson. 2005. Ten CAD Challenges. IEEE Comput. Graph. Appl. 25, 2 (March 2005), 81-92. DOI=10.1109/MCG.2005.48 http://dx.doi.org/10.1109/MCG.2005.48

Freese, M.; Singh, S. P. N.; Ozaki, F. & Matsuhira, N. (2010), Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator., in Noriaki Ando; Stephen Balakirsky; Thomas Hemker; Monica Reggiani & Oskar von Stryk, ed., 'SIMPAR' , Springer, pp. 51-62 .

Godinho, Tiago, Pires, JN, "A Autodesk na programação de robôs industriais", Journal Robótica, nº65, Novembro, 2006, pp 36-39

J. Norberto Pires, T. Godinho, P. Ferreira, CAD interface for automatic robot welding programming pp. 71 - 76 in *Industrial Robot: An International Journal*, Emerald Group, 2004. DOI 10.1108/01439910410512028

Leon Žlajpah, Simulation in robotics, Mathematics and Computers in Simulation, Volume 79, Issue 4, 15 December 2008, Pages 879-897, ISSN 0378-4754, http://dx.doi.org/10.1016/j.matcom.2008.02.017.

Love, L. J., "Robot Simulation," pp. 21-1–21-7 in *Robotics and Automation Handbook*, CRC Press, Inc., 2005

N. Papakostas, K. Alexopoulos, A. Kopanakis, Integrating digital manufacturing and simulation tools in the assembly design process: A cooperating robots cell case, CIRP Journal of Manufacturing Science and Technology, Volume 4, Issue 1, 2011, Pages 96-100, ISSN 1755-5817, http://dx.doi.org/10.1016/j.cirpj.2011.06.016.

Neto P., "Intuitive Human-Robot Interaction using CAD Drawings, Gestures and Programming by Demonstration", Ph.D., University of Coimbra, 2012. ISBN: 978-972-8954-27-7

Neto P., Mendes N., Araújo R., Pires J.N. and Moreira A.P.: "High-level robot programming based on CAD: dealing with unpredictable environments," Industrial Robot, Emerald, Vol. 39, No. 3, pp. 294-303, 2012

Neto P., Mendes N: "Direct off-line robot programming via a common CAD package," Robotics and Autonomous Systems, Elsevier, Vol. 61, No. 8, pp. 896-910, 2013, http://dx.doi.org/10.1016/j.robot.2013.02.005

Neto P., Pires J.N. and Moreira A.P.: "Robot path simulation: a low cost solution base on CAD," 4th IEEE International Conference on Robotics, Automation and Mechatronics, RAM 2010, pp. 333-338, Singapore, Singapore, 2010

Neto, P; Pires, J.N.; Moreira, A.P., "CAD-based off-line robot programming," *Robotics Automation and Mechatronics (RAM), 2010 IEEE Conference on* , vol., no., pp.516,521, 28-30 June 2010
doi: 10.1109/RAMECH.2010.5513141

Vitor Bottazzi and Jaime Fonseca (2006). Off-line Programming Industrial Robots Based in the Information Extracted From Neutral Files Generated by the Commercial CAD Tools, Industrial Robotics: Programming, Simulation and Applications, Low Kin Huat (Ed.), ISBN: 3-86611-286-6, InTech, DOI: 10.5772/4906.

Yahui Gan, Xianzhong Dai and Dongwei Li (2013). Off-Line Programming Techniques for Multirobot Cooperation System, International Journal of Advanced Robotic Systems, Antonio Visioli (Ed.), ISBN: 1729-8806, InTech, DOI: 10.5772/56506.

Z. Pan, J. Polden, N. Larkin, S. V. Duin, J. Norrish, Recent progress on programming methods for industrial robots, Robotics and Computer-Integrated Manufacturing 28 (2012) 87–94.

ANEXO A

"*Software*" exclusivos

ROBOTSTUDIO - RobotStudio da ABB é, actualmente um dos *software* mais utilizado e tem como principais características o MultiMove, CAD Import, AutoPath, AutoReach, Path Optimization, Collision Detection, Virtual FlexPendant, True Upload and Download, Rapid editor, Debugging, EventManager, VSTA, MechanismModeler, ScreenRecorder, Virtual Time, AutoConfiguration, Instruction templates. Apesar de muito completo, este *software* é destinado apenas aos robôs da ABB.

A função MultiMove permite trabalhar com vários robôs virtuais em simultâneo a partir de um único controlador. É possível importar ficheiros em vários formatos CAD como IGES, STEP, VRML, VDAFS, ACIS e CATIA. Uma característica importante do RobotSudio é a AutoPath, característica que esta permite economizar muito tempo. A AutoPath permite, a partir de um modelo CAD, gerar automaticamente as posições necessárias ao robô para este seguir uma curva ou trajectória em apenas alguns minutos. De modo a verificar se o robô consegue, ou não, chegar á posição que se pretende a ABB VirtualRobot™ adicionou uma nova funcionalidade designada por AutoReach. A Path Optimization (optimização de caminho), detecta e avisa automaticamente sobre os programas que incluem movimentos em estreita proximidade com singularidades. Desta forma podem ser tomadas medidas que evitem tais situações. A simulação por motorização é uma ferramenta visual da RobotStudio que permite optimizar o movimento do robô, i.e. velocidade, aceleração, singularidades e eixos traduzindo-se em redução de tempo de ciclos.

A praticabilidade do detector de colisões é basilar impedindo danos ao equipamento ou mesmo ao robô. É possível criar macros com o *add-in* VSTA em C# ou Visual basic.net bem como criar os nossos próprios recursos e integrá-los com o RobotStudio GUI. Com o MechanismModeler o usuário pode modelar as suas próprias pinças ou ferramentas para uma simulação. O RobotStudio ainda permite que se grave uma simulação do robô para demonstração e gestão I/O. Finalmente, o VirtualTime torna possível estimar os tempos de ciclo do programa do robô.

MOTOSIM EG - O Motosim EG é um *software* dedicado aos robôs da Motoman. O simulador Motosim EG permite importar ficheiros CAD apenas em formatos .hmf, .hsf e .3ds e só é compatível com alguns modelos de robôs da Motoman. Não obstante o Motosim EG permita minimizar custos, para o correcto funcionamento, este requer um *software* complementar para calibração.

As principais características são a análise de sucesso de chegada, gestão I/O, detector de colisões, posicionamento do robô e optimização do caminho. Com o Motosim EG o utilizador também pode calcular trajectórias, planeamento de trajecto, e ainda fazer alterações para simplificar a programação. Este *software* fornece ao utilizador diversas informações como velocidade, aceleração e trajectórias.

ROBOGUIDE - O Roboguide produzido pela FANUC e é compatível apenas com os robôs desta marca. Este *software* cria automaticamente programas de referência de modo a calibrar a simulação com o sistema real do robô. Similar a outros "*software*" o RoboGuide também permite detectar colisões, planear trajectória, gestão I/O e exportar vídeos das simulações. O RoboGuide utiliza ficheiros de formato CAD do tipo IGES e possui a funcionalidade Profiler Function que analisa o tempo de cada linha de comando de modo a obter o melhor tempo de ciclo. Contudo, assim como outros *software*, o RoboGuide também requer packs extra para estender algumas funcionalidades importantes como por exemplo soldadura, rebarba, pintura e um plug-in que, automaticamente, posiciona o robô em relação á peça de trabalho.

KUKA.SIM PRO - O *software* de simulação e OLP fornecido pela KUKA, KUKA.Sim Pro é um programa que permite criar layouts 3D de sistemas com robôs KUKA. É possível simular e analisar facilmente os layouts e conceitos desejados. Sistemas transportadores push-pull e sistemas transportadores com velocidade constante também podem ser simulados no KUKA.Sim Pro. Este *software* permite a importação de formatos CAD padrão, estão disponíveis STL, 3DS, VRML1, Robface e Google Sketchup.

Opcionalmente, estão disponíveis importações CAD individuais adicionais para o KUKA.Sim Pro: CATIA V5, CATIA V4, Siemens NX, JT, STEP, Parasolid, ProE, SolidWorks, ACIS, IDEAS, IGES. No que respeita à simulação, o KUKA.Sim Pro tem a capacidade de verificar eventuais colisões, tempo de ciclos e criar o programa do robô,

contudo, estas duas últimas funcionalidades requerem o apoio de um outro *software* da KUKA, KUKA.OfficeLite.

"Software" genéricos

FASTSIMU - Fastsimu, produzido pela Fastems, é uma plataforma de *software* composta por ferramentas visuais de elevada qualidade de interface incluindo uma gama diversificada de ferramentas especiais para manipulação de caminhos de ferramenta. O Fastsimu permite a importação modelos de acessórios e peças em formato CAD 3D. Permite ainda detectar superfícies, furos e bordas dos modelos e gera caminhos de ferramenta. De modo a minimizar as diferenças entre o modelo real e o modelo virtual o Fastsimu fornece ferramentas para calibração.

O programa no que respeita à simulação inclui, movimentos do robô, análise de acessibilidade, detecção de colisão, mudança de ferramenta automática e relatório de tempo de ciclos. Os formatos padrão para importação de modelos são: VRML, STL, DXF, 3DS, e, opcionalmente, um dos seguintes: Inventor, JT, Open, Parasolid, SolidWorks, CATIA, STEP, ProE, IDEAS NX, IGES.

O Fastsimu é um *software* independente do fabricante do robô e fornece informação tanto de velocidades como acelerações.

3DAUTOMATE - 3DAutomate é um *software* desenvolvido pela Visual Components para simulação e programação offline de robôs. Trata-se de um programa pago em tudo, semelhante ao Fastsimu.

3DAutomate é uma ferramenta multifunções e multimarca. Tem a capacidade de planear e gerar caminhos, verificar colisões, tempos de ciclos, gestão I/O e exportar vídeos das simulações, posicionamento do robô e optimização do caminho, entre outras. É possível importar ficheiros em vários formatos CAD como IGES, STEP, VRML, VDAFS,ACIS e CATIA.

ROBOWAVE HYPROGRAM - RoboWave hyprogram é um *software* intuitivo de OLP que permite a programação de qualquer robô. O facto de ser intuitivo

permite que o usuário se concentre no processo de trabalho real em vez de ensinar cada movimento ao robô.

O RoboWave hyprogram, produzido pela Erxa, tem como principais características planear e gerar caminhos, verificar colisões, tempos de ciclos, gestão I/O e exportar vídeos das simulações, posicionamento do robô e optimização do caminho, entre outras. Suporta ficheiros de formatos CAD como IGES, STEP, VRML, VDAFS, ACIS e CATIA.

DELMIA OLP - O DELMIA OLP é um *software* desenvolvido pela Dassault Systèmes para simulação e programação offline de robôs. Programadores de robôs, familiarizados com o ensino de robôs em ambiente real podem facilmente aprender a ensinar em simulação evitando perdas de horas de trabalho ou erros. Os utilizadores podem transferir os programas de simulação optimizados, através dos programas gerados *off-line*. Uma arquitectura *off-line* aberta permite ao utilizador programar em JAVA, XSLT (Extensible Stylesheet Language Transformations), Visual Basic, etc.

DELMIA OLP é uma ferramenta multifunções e multimarca. Tem a capacidade de planear, verificar colisões, tempos de ciclos, gestão I/O e exportar vídeos das simulações, posicionamento do robô, entre outras. É possível importar ficheiros em vários formatos CAD como IGES, STEP, VRML, VDAFS, ACIS e CATIA.

"Software" open source

CARCAD – O CARCAD é uma *standalone application*, desenvolvida na FCTUC pelo investigador Pedro Neto e Nuno Mendes, que permite criar trajectórias e programas de robô de forma intuitiva. Esta aplicação trabalha sobre a interface gráfica do *Autodesk Inventor*. No que respeita à importação de modelos o CARCAD suporta ficheiros de formatos CAD como IGES, STEP, SAT, ACIS, CATIA e DXF.

V-REP - O V-rep é um simulador open source muito completo com diversas funcionalidades. O V-rep, produzido pela Coppelia Robotics, tem uma excelente API (Application Programming Interface) e ainda possibilita a programação através de seis métodos diferentes (Freese, M.; Singh, S. P. N.; Ozaki, F. & Matsuhira, N. et al., 2010). Este *software* tem como principais características: simulação dinâmica, cinemática, dinâmica de partículas, detecção de colisões, cálculo de distância mínima. Suporta ainda

simulação de corte com as mais diversificadas ferramentas, sensor de proximidade e de visão, construção em bloco, planeamento de caminho, grava os dados das simulações, etc. No que respeita à importação de modelos o V-rep suporta ficheiros de formatos CAD como IGES, STEP, VRML, VDAFS, ACIS, CATIA, DXF, OBJ, 3DS, STL e COLLADA.

ANEXO B

A Wizard for Inventor .NET *Add-in*

It finally comes out a user-friendly multi-page and real-sense wizard for helping program Inventor .NET Addins. The following features are provided as now:

• Developers can specify which Inventor product and version to work with.

• If the particular Inventor version has some flavours such as Inventor Standard or Inventor Professional, they will be listed out too for people to choose.

• The specified Inventor executable can be launched automatically from the Visual Studio debugger if opted.

• The Inventor API COM Interop assembly is defaulted to the one residing in the Inventor installation folder and can be changed to any other good.

• The specified Inventor API COM Interop assembly will be referenced into the project automatically.

• An Inventor file such as a part file (.ipt) and an assembly file (.iam) can be specified for the Inventor to load automatically at start up.

• The Inventor Application Addin Server (Inventor.ApplicationAddInServer) is supported and can be generated with a different class name from the project name.

• The GUID of the Inventor Application Addin Server is provided a default value and can be changed to anything else valid in case the naming convention for the GUIDs is followed.

• An Inventor Addin manifest file if applicable will be created automatically with all necessary nodes properly filled out during the Visual Studio project creation.

• A copy of the same Inventor Addin manifest file will be deployed into the right spot (the proper user roaming folder) automatically during the project creation.

• A link to the copy of Inventor Addin manifest file will be added to the project in case review or update is necessary in the future.

• The display name and the description of the Inventor Addin in the manifest file are provided with some default values and can be changed to something more meaningful in the wizard.

• The Addin data version, UI version, supported version greater than and less than, user unloadable, and hidden or not can all be specified in the same wizard page.

• The Addin load behavior such as 'Load on Inventor Startup' and 'Load on Demand' can be chosen from all good options at the same time.

• Not only the ClassId and the ClientId nodes of the Addin manifest file will be filled out properly and automatically but also the Assembly node will have the full path and name of the resultant Inventor addin DLL.

• For Inventor versions that do not support the manifest file (.AddIn) or COM Free Registration, the COM Interop Registration approach will be used intelligently.

• The Inventor Ribbon UI is supported.

• The Inventor Ribbon environment such as Part or Drawing can be specified.

• A Ribbon Panel with many good options will be created and added into the specified Inventor Ribbon environment automatically.

• The Ribbon Panel name and client id are provided with some good default values and can be changed to anything else more meaningful.

• A Ribbon Button with many good options again will be created and added into the Ribbon Panel just created as above.

• The Ribbon Button display name, description, and tooltip can be specified in the same wizard page.

• The Ribbon Button internal name will be created automatically based on a good naming convention.

• The Ribbon Button will be provided a nice default icon and it can be changed to anything more meaningful once again.

• The Ribbon Button creation is optional and not forcible.

• The addin assembly name can be different from the project name at the project creation time.

• The resultant Inventor Addin assembly can be registry free depending on the Inventor product and version selection.

• A lot good and necessary namespaces are imported into the auto-generated source files automatically and ready to use right away.

• Many reusable global objects such as the Inventor Application object and the Addin Server class ID are cached in a good central class.

• Inventor Ribbon Button callbacks are also organized into a central place class automatically.

• An Inventor Button helper class is provided and it contains a lot reusable good properties, methods, and signatures.

• No redundant namespaces will be added for the addin server class and those helpers. Only good and necessary namespaces will be added to avoid confusion in case the assembly needs to be referenced into some other projects with the same programming language or different.

• Dozens of Inventor API Event Handler item wizards are provided.

• C# and VB.NET languages are supported.

• Both Visual Studio fully fledged version and Express are supported.

• Both Visual Studio 2008 and 2010 are supported.

• Both 32 bit and 64 bit environment are supported.

• Many Inventor versions are supported and fully tested such as 2013, 2012, 2011, 2010, and 2009.

• Many different Windows versions are supported and tested such as Windows 7, Windows Vista, and Windows XP.

• Manifest Navigator Widget is provided.

• RegEdit Launcher Widget is provided.

• Inventor Locator Widget is provided.

• Data Collector Coder is provided.

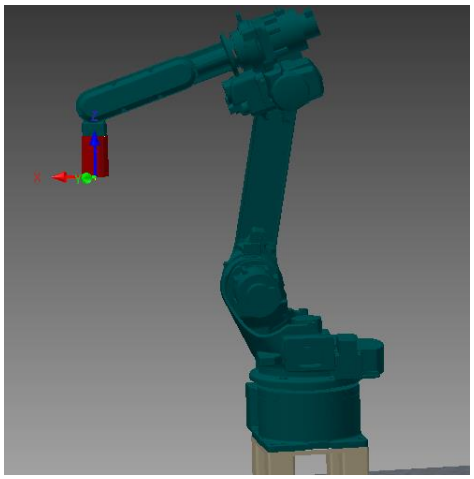
It can be found and downloaded below.

https://docs.google.com/file/d/0B13_dQJqHUAjVEN6SU11YV9jTXc/edit

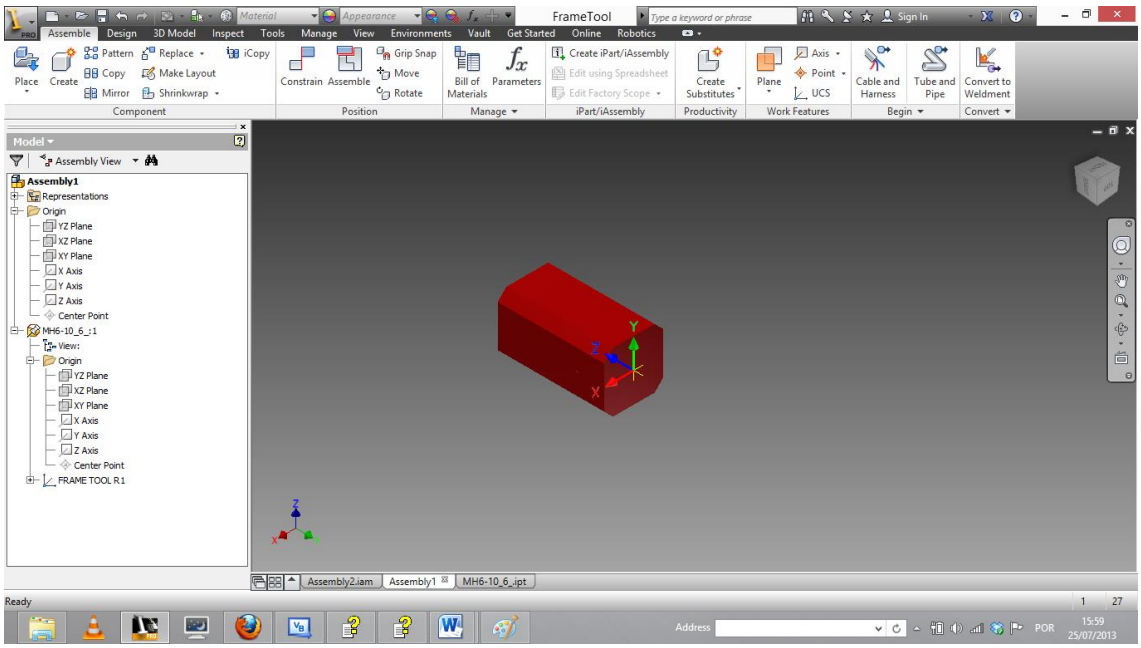
ANEXO C

Calibração da Frame Tool

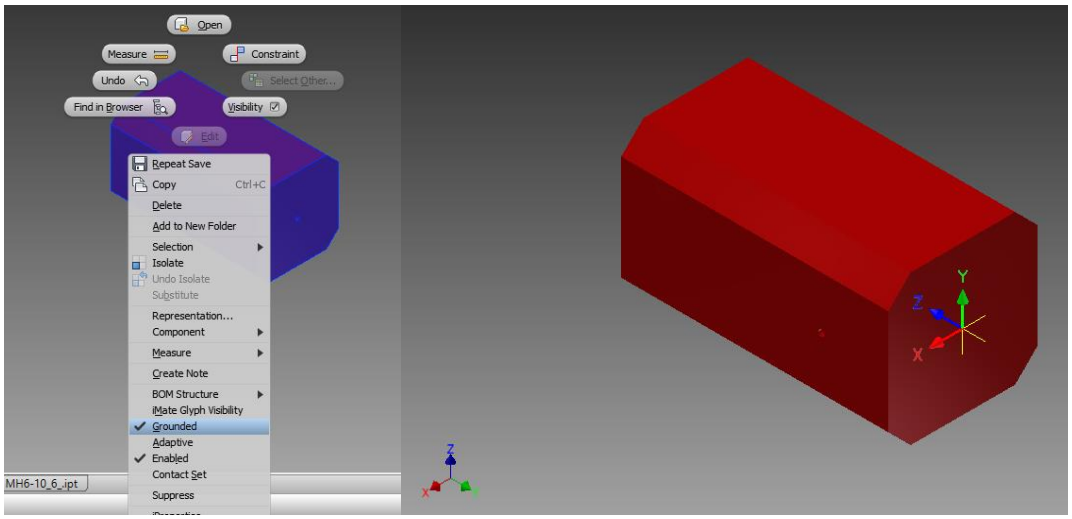
É necessário colocar um UCS na ferramenta do robô. A aplicação do UCS na frame tool (ferramenta do robô) tem de ser feito na *part*. O UCS deve ser posicionado no zero relativo da frame tool permitindo-nos posteriormente referir à posição de trabalho.



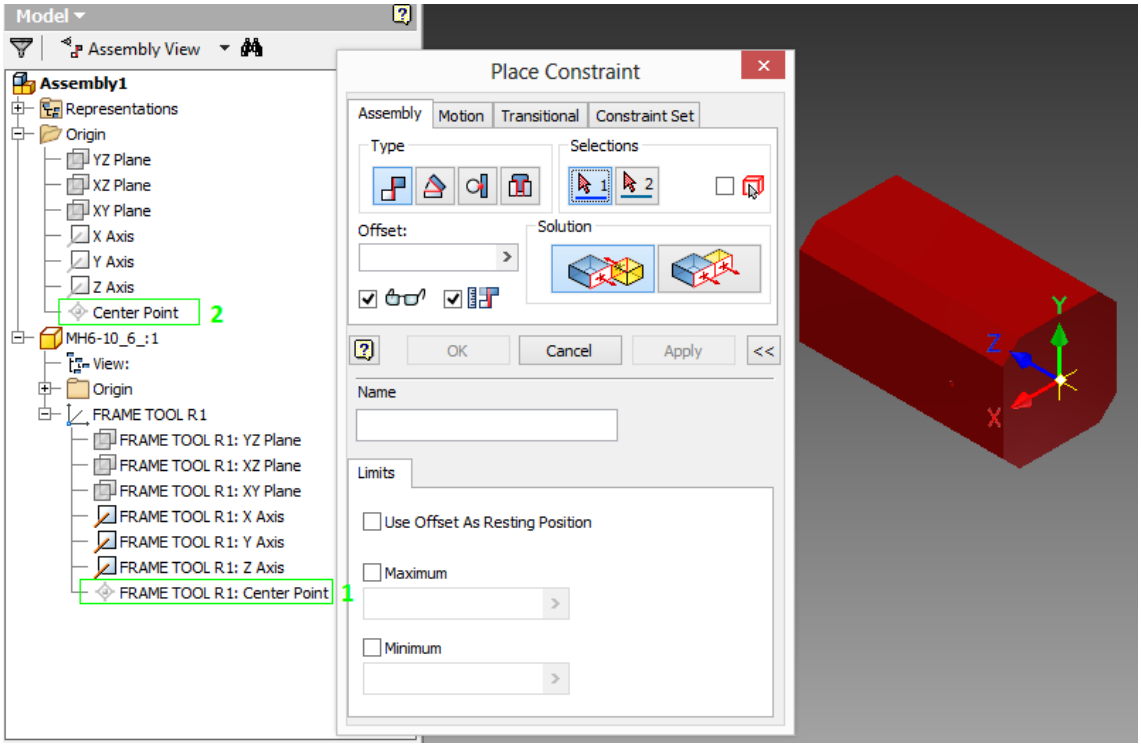
Se a frame tool for uma *part*, cria-se um novo *assembly* e importa-se a frame tool.ipt.



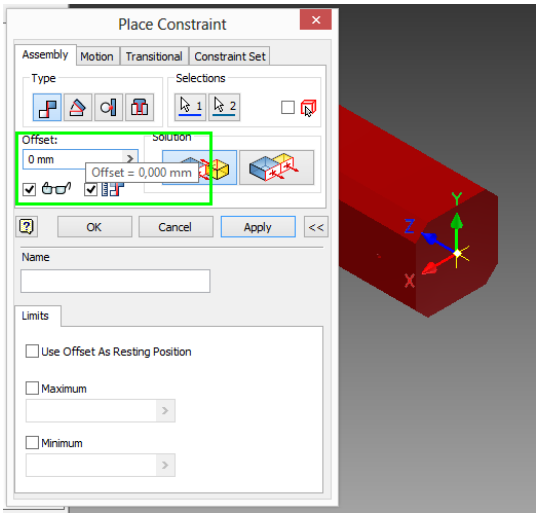
Para mover a origem para o zero da Frame_Tool, tem de se verificar se a peça não está com a propriedade "ground" activa. Caso esteja deve-se desactivar.

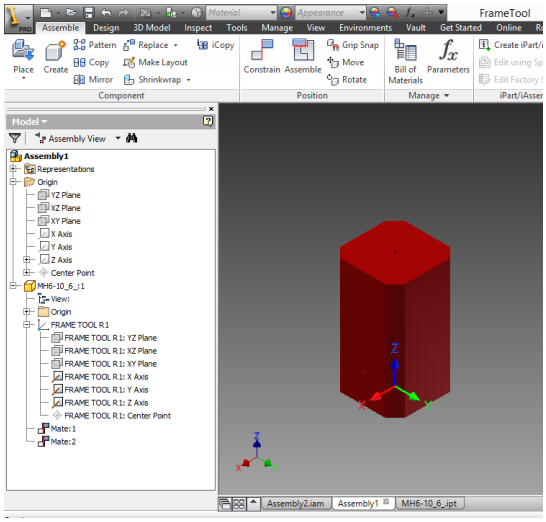


Aplicar uma *constraint* na origem do UCS (1) que foi definido anteriormente na *part* e na origem do *assembly* (2).



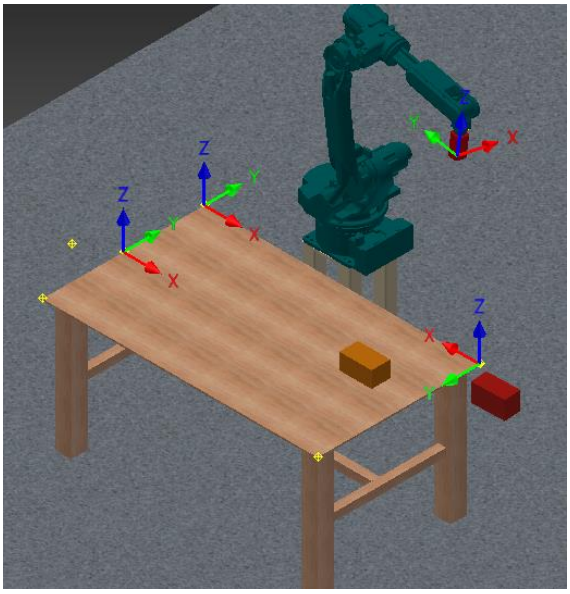
Deve-se garantir que o *offset* é zero; Aplicar.


Place Constraint
Assembly
Motion
Transitional
Constraint Set
Type
Selections
Offset:
0 mm
Offset = 0,000 mm
Solution
OK
Cancel
Apply
<<
Name
Limits
Use Offset As Resting Position
Maximum
Minimum
Y
Z
X

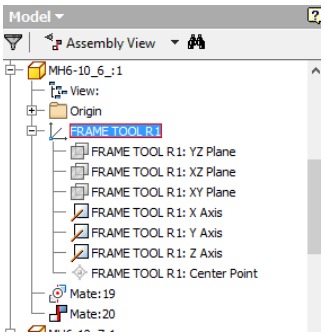

FrameTool
Assemble
Design
3D Model
Inspect
Tools
Manage
View
Environments
Vault
Get Started
Online
Place
Create
Pattern
Replace
iCopy
Copy
Make Layout
Mirror
Shrinkwrap
Component
Constrain
Assemble
Grip Snap
Move
Rotate
Position
Bill of Materials
Parameters
Manage
Model
Assembly View
Assembly1
Representations
Origin
YZ Plane
XZ Plane
XY Plane
X Axis
Y Axis
Z Axis
Center Point
MH6-10_6_:1
View:
Origin
FRAME TOOL R1
FRAME TOOL R1: YZ Plane
FRAME TOOL R1: XZ Plane
FRAME TOOL R1: XY Plane
FRAME TOOL R1: X Axis
FRAME TOOL R1: Y Axis
FRAME TOOL R1: Z Axis
FRAME TOOL R1: Center Point
Mate:1
Mate:2
Assembly2.iam
Assembly1
MH6-10_6_.ipt

ANEXO D

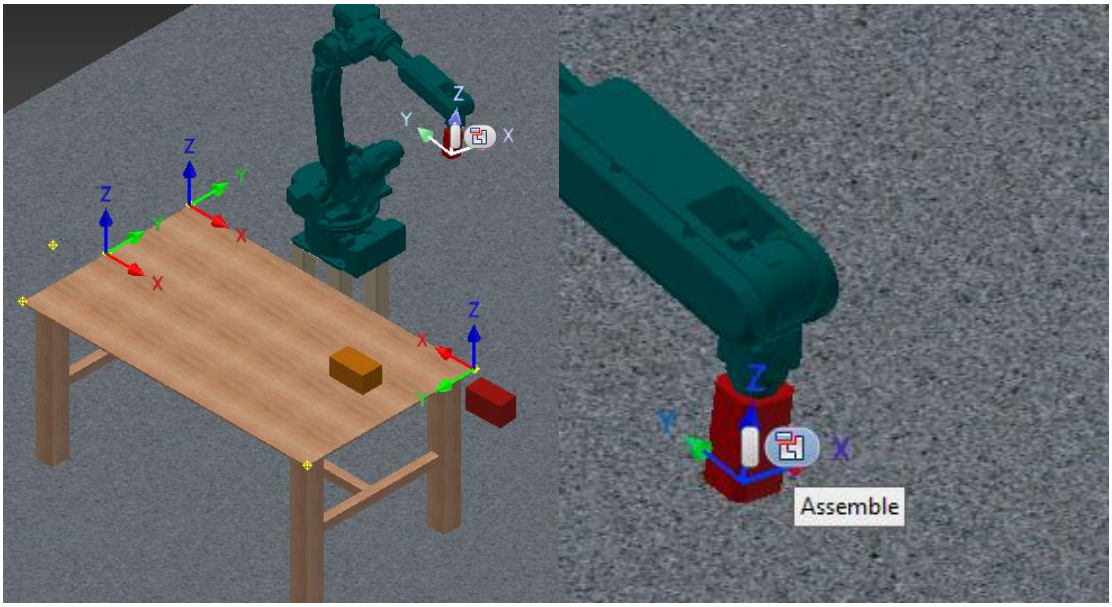
Posicionar o Robô aos pontos de trabalho



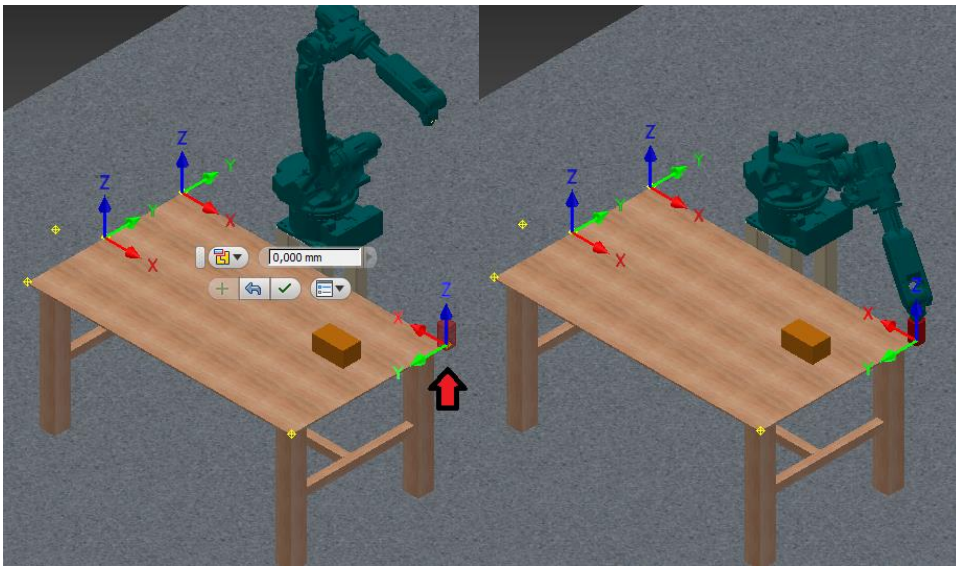
Na árvore, selecciona-se o UCS que se refere á FrameTool, no nosso caso "FRAME TOOL R1"



Depois de seleccionada é imediatamente disponibilizada a ferramenta de posicionamento de peças (Constrain) próximo do UCS da *FrameTool*.



Para o robô se deslocar para uma posição específica selecciona-se o UCS de destino



Para mover a *FrameTool* para uma outra posição (segunda, terceira, etc.) repete-se o procedimento visto em cima