Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE

Arthur Cruz de Araújo

Sistema de rastreamento de um manipulador robótico e deteção do estado de seu efetuador final

Campina Grande, Paraíba
Abril de 2017

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE

Arthur Cruz de Araújo

Sistema de rastreamento de um manipulador robótico e deteção do estado de seu efetuador final

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Orientador:
Professor Antonio Marcus Nogueira Lima, Dr. Sc.

Campina Grande, Paraíba
Abril de 2017

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE

Arthur Cruz de Araújo

Sistema de rastreamento de um manipulador robótico e deteção do estado de seu efetuador final

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Aprovado em / /

Professor Antonio Marcus Nogueira Lima, Dr. Sc.
Universidade Federal de Campina Grande
Orientador

Professor Gutemberg Gonçalves dos Santos Júnior, Dr. Sc.
Universidade Federal de Campina Grande
Convidado

Campina Grande, Paraíba
Abril de 2017

Este trabalho é dedicado a meus pais.

Agradecimentos

Agradeço a minha família pelo suporte desde sempre, possibilitando minha dedicação exclusiva à graduação, além do constante acompanhamento ao longo da mesma, especialmente nos períodos de afastamento para intercâmbio, como é o que caso que permeia a realização deste trabalho.

Agradeço a minha namorada Carla pelo apoio contínuo, companheirismo e carinho, invariantes à distância.

Agradeço ao Professor Dr. Ville Kyrki, da Aalto University, Finlândia, pela oportunidade de desenvolver este Trabalho de Conclusão de Curso em um de seus laboratórios, durante o programa de intercâmbio *Erasmus Mundus*.

Agradeço também a Rajkumar Muthusamy, que não só propôs o trabalho, como também orientou e acompanhou o seu desenvolvimento. Agradeço ainda ao amigo e colega de laboratório Jaakko Mattila, cujas ideias sempre levavam a boas discussões sobre nossas pesquisas.

Agradeço ao Professor Dr. Antonio Marcus Nogueira Lima, que aceitou orientar este Trabalho de Conclusão de Curso, apesar da distância e das diferenças de fuso horário, e ainda o vem fazendo desde o fim do intercâmbio, auxiliando na redação deste documento.

"To succeed, planning alone is insufficient.
One must improvise as well."
Isaac Asimov

Resumo

A popularidade de sistemas de múltiplos robôs tem impulsionado a pesquisa sobre interação colaborativa robô-robô: reconhecer as ações de outro robô é bastante útil em situações de cooperação e ao assistir manipuladores na realização de tarefas de apanhar e depositar objetos. Muito tem sido feito no estudo de rastreamento e detecção de mãos humanas para fins de interação. De forma análoga, tais fatores motivaram o desenvolvimento de um sistema de visão com o objetivo primordial de conferir a um dado robô certo nível de ciência a respeito do estado de outros robôs, sem conhecimento prévio das dinâmicas de suas juntas, por exemplo. É proposto um sistema de visão 3D baseado em modelos que tem como principais metas a detecção da garra de um manipulador robótico em uma cena, seu rastreamento à medida em que ela se move e a determinação contínua do seu estado (aberta ou fechada). Este sistema é composto fundamentalmente por um algoritmo de registro – uma combinação dos métodos de Sample Consensus Initial Alignment (SAC-IA) e Iterative Closest Point (ICP); um filtro de partícula para rastreio da garra do manipulador e um classificador de seu estado baseado numa medida de similaridade entre nuvens de pontos. Fundamentado sobre o Sistema Operacional de Robôs (ROS), combina métodos de processamento de nuvens de pontos aos algoritmos citados, enquanto faz uso dos modelos disponibilizados para identificar se a garra está aberta, fechada ou se está segurando um objeto. A plataforma de teste utilizada no projeto compõe-se de um braço robótico Kinova Jaco^2 com 6-DOF e três dedos, além de um módulo de câmera RGB-D Microsoft Kinect. Em geral, os resultados experimentais se mostraram satisfatórios: o rastreamento apresentou boa performance para trajetórias comportadas e pôde-se detectar, sem erros, não só o estado da garra, demonstrando robustez a auto-oclusões, como também a presença de objetos. Uma das limitações observadas foi a baixa sensibilidade do rastreador a rotações da garra. Ainda, o uso de apenas dois modelos de referência também pode ser visto como uma limitação. Respeitadas as suposições e limitações do sistema, a detecção de objetos e da sua apreensão funcinonam bem. Vídeos demonstrativos foram disponibilizados online para melhor compreensão do funcionamento do sistema. Trabalhos posteriores podem explorar a generalização do detector para manipuladores similares, bem como contemplar a inserção deste sistema em um cenário colaborativo de robôs.

Palavras-chave: robótica, sistema decentralizado, múltiplos robôs, visão, registro, rastreamento, modelos, reconhecimento.

Abstract

The popularity of multi robot systems has propelled research on collaborative robot-robot interaction: recognizing other robot's actions is very useful in cooperation and for assisting single robots in stable pick and place tasks. Lots of work has been done in human hand tracking and recognition for interaction. Analogously, those factors motivated the development of a vision system with the prime goal of providing a robot with awareness about others robots' states, without knowledge concerning their joints' dynamics, for example. A 3D model-based is proposed, with the main goals of detecting a manipulator's gripper on a scene, tracking it as it moves and continuously determine its state (opened or closed). The system comprises mainly of a registration pipeline – a combination of the Sample Consensus Initial Alignment (SAC-IA) and the Iterative Closest Point (ICP) algorithms; a particle filter used for tracking the gripper and a state classifier based on a measure of similarity between pointclouds. It runs under the Robot Operating System (ROS) and combines point cloud processing methods with the aforementioned algorithms, while using cloud models of the gripper to identify its state, i.e. whether it is opened or closed, as well as if it's grasping an object. The system was implemented and evaluated with a test platform composed of a 6-DOF Kinova Jaco^2 robotic arm, with a three-fingered gripper, and a Microsoft Kinect RGB-D (red, green and blue with per-pixel depth information) camera. In general, the experimental results were satisfactory: tracking had a good performance for well-behaved trajectories and the detection of not only the state of the gripper, which showed robustness to self-occlusions, but also the presence of an object, were successful. One of the observed limitations was the tracker's low sensitivity to rotations of the gripper. Also, using only two reference models could as well be seen as a limitation. The system's assumptions and limitations being respected, object detection would perform well, and also its grasping. Beyond the discussion of results, demonstration videos are available online for better understanding. Future works might explore generalization of the detector to similar manipulators, as well as contemplate the insertion of this system in a collaborative scenario.

Keywords: robotics, decentralized system, multi-robot systems, vision, registration, tracking, model-based, recognition.

Sumário

1 INTRODUÇÃO 13

2 OBJETIVO 15

3 FUNDAMENTAÇÃO 16
3.1 Nuvem de Pontos 16
3.2 Projeto Baseado em Modelos 17
3.3 Sample Consensus Initial Alignment 19
3.3.1 Point Feature Histogram 19
3.3.2 Fast Point Feature Histogram 20
3.3.3 O Algoritmo SAC-IA 21
3.4 Iterative Closest Point 22
3.4.1 O Algoritmo ICP 23
3.5 Particle Filter Algorithm 24
3.5.1 Rastreamento Bayesiano não-linear 24
3.5.2 O Algoritmo PFA 27
3.6 Cálculo de Similaridade entre Nuvens de Pontos 28
3.7 As Atribuições dos Algoritmos Utilizados 29

4 O SISTEMA PROPOSTO 30
4.1 Ambiente de Teste 30
4.2 Arquitetura Geral do Projeto no Sistema ROS 31
4.3 Aquisição de dados - *Freenect* 32
4.4 Geração de Cena 33
4.5 Registro da nuvem de pontos 34
4.6 Rastreamento do manipulador e do efetuador final 38
4.7 Detecção do estado da garra 41
4.7.1 Detecção de objetos a partir de modelos 43
4.8 Interface com usuário 44

5 RESULTADOS 47
5.1 Limitações e Potenciais Aprimoramentos 50
5.2 Vídeos Demonstrativos 51

6 CONCLUSÃO 52

REFERÊNCIAS 53

Lista de figuras

Figura 1 – Modelos CAD para a garra aberta e fechada do manipulador Jaco^2. A parte que se refere à palma foi removida do modelo original em concordância com o braço robótico utilizado. 17
Figura 2 – Nuvens de pontos extraídas dos modelos CAD na Fig. 1. 18
Figura 3 – Sistema de coordenadas de referência para o modelo aberto da garra. O sistema utilizado para o modelo fechado segue as mesmas convenções. . 18
Figura 4 – Diagrama de região de influência para um PFH. O ponto de análise (vermelho) e seus k vizinhos (azul) estão interconectados em uma malha. 20
Figura 5 – Diagrama de região de influência para um FPFH. O ponto de análise (vermelho) está conectado apenas a seus k vizinhos (envolvidos pelo círculo cinza). As conexões marcadas com 2 contribuirão duas vezes para o FPFH. 21
Figura 6 – Da esquerda para a direita: duas visões parciais do modelo de um coelho antes do alinhamento; resultados obtidos com a aplicação da solução encontrada pelo SAC-IA. 22
Figura 7 – As duas etapas que definem o rastreamento bayesiano não-linear. Para evitar confusão quanto aos índices, o arco tracejado representa o fim de um loop temporal, onde k seria incrementado. 26
Figura 8 – Um diagrama ilustrativo para o processo envolvido no Filtro de Partícula. 27
Figura 9 – Montagem experimental usando o sensor Kinect e o manipulador Jaco^2 6-DOF. 30
Figura 10 – Diagrama ilustrando a comunicação via mensagens entre nós no sistema ROS. 31
Figura 11 – Arquitetura geral do sitema no ROS. 32
Figura 12 – Uma captura de tela da cena salva pelo nó de geração de cena. 33
Figura 13 – Pose inicial definida para o braço robótico Jaco^2. 34
Figura 14 – Apresentação das nuvens de pontos referentes à cena e ao modelo de garra aberta antes do registro. 35
Figura 15 – Apresentação de alguns passos de pré-processamento: à esquerda, é apresentada a cena resultante após aplicação dos filtos *pass-through*, da remoção de ruído e da subamostagem; à direita, o resultado da posterior segmentação de plano e custerização, onde o cluster correspondente à garra está destacado em vermelho. 36
Figura 16 – Alinhamento Inicial para o modelo de garra aberta usando o algoritmo SAC-IA. 36
Figura 17 – Alinhamento final para o modelo de garra aberta pelo algoritmo ICP. . 37

Figura 18 – Registro final para o modelo de garra aberta visualizado no RViz. . . . 38
Figura 19 – Capturas de tela ilustrando o funcionamento do Filtro de Partícula. Inicialização (esquerda), primeira iteração (centro) e vigésima-quinta (direita). 39
Figura 20 – Captura de tela no RViz do nó de rastreamento em execução, ressaltando a hierarquia entre os sistemas de coordenadas. 40
Figura 21 – Duas amostras de nuvem de pontos mostrando uma visão parcial da garra na cena, referenciadas no frame da pose atual. 40
Figura 22 – Os quatro casos extremos para classificação do estado da garra como aberta ou fechada. Os pontos em amarelo correspondem às nuvens dos modelos e os pontos pretos às nuvens de cena. 41
Figura 23 – Ecoando os dados publicados pelo nó detector do estado da garra (NDEG). 42
Figura 24 – Comportamento das medidas de similaridade com os modelos disponíveis ao longo do contínuo abrir e fechar da garra. 43
Figura 25 – Uma máquina de estados idealizada para a garra do manipulador. . . . 44
Figura 26 – Visualização da interface desenvolvida em um cenário com a garra completamente aberta e ausência de objetos. 46
Figura 27 – Visualização da interface desenvolvida em um cenário onde um objeto encontra-se firmemente agarrado pelo manipulador. 46
Figura 28 – Uma captura de tela exemplificando o nó de rastreamento em execução. 47
Figura 29 – Um exemplo de rastreamento da garra fechada, junto aos gráficos de similaridades e à interface gráfica desenvolvida. 48
Figura 30 – Detecção de objeto a ser apreendido a partir da abordagem baseada em modelos. 49
Figura 31 – Efeito da apreensão do objeto sobre os valores das similaridades. 49
Figura 32 – Demonstrando a limitação de baixa sensibilidade do algoritmo de rastreamento a rotações da garra. 50
Figura 33 – Demonstrando a perda do rastro da garra quando do seu deslocamento carregando um objeto. 51

Lista de tabelas

Tabela 1 – Valores de similaridades obtidos para os casos extremos de classificação. 41

Lista de abreviaturas e siglas

RRI	Robot-robot interaction
ICP	Iterative Closest Point
RANSAC	Random Sample Consensus
ROS	Robot Operating System
CAD	Computer Aided Design
BDS	Berkeley Software Distribution
PCL	Point Cloud Library
RGB-D	Red, Green, Blue and Depth
SAC-IA	Sample Consensus Initial Alignment
PFH	Point Feature Histogram
FPFH	Fast Point Feature Histogram
SPFH	Simplified Point Feature Histogram
SHT	Single Hypothesis Tracking
MHT	Multiple Hypothesis Tracking
FP	Filtro de Partícula
DOF	Degree of Freedom
NDEG	Nó de Detecção do Estado da Garra

1 Introdução

O advento de sistemas colaborativos de múltiplos robôs relaciona-se com a abordagem introduzida em Arkin (1998) no que diz respeito ao controle baseado em comportamentos. Robôs com manipuladores hábeis podem, trabalhando coletivamente, não só completar ações de forma mais eficiente do que um robô individual faria, como também realizar atividades mais complexas (Kragic e Christensen, 2002), impossíveis de serem executadas por um único manipulador.

No projeto de tais sistemas robóticos colaborativos, visão computacional mostra-se fundamental, especialmente em se tratando de ambientes desestruturados (Hashimoto, 2003). Em um cenário RRI — *robot-robot interaction* (Wang, Johnston e Williams, 2012), a partir da observação da mão de um determinado robô e posterior interpretação do estado de seu respectivo manipulador, outro robô pode ser controlado de modo a operar em harmonia com o primeiro, em direção ao cumprimento de uma dada tarefa.

A preocupação em torno deste tipo de interação entre robôs se justifica na demanda das mais diversas aplicações, tais como a apreensão (*grasping*) de objetos ou transporte e montagem de estruturas. Para um sistema de visão, mais especificamente, é útil direcionar o foco do desenvolvimento para o efetuador (*end-effector*) do manipulador robótico, assim evitando lidar com toda a estrutura do braço. Tarefas como as supracitadas dependem tão somente de informações referentes ao efetuador final, doravante denominado garra (*gripper*) do manipulador, como sua posição, velocidade e estado dos dedos.

Naturalmente, o primeiro passo para o rastreio (*tracking*) do manipulador é a determinação de sua pose, isto é, sua posição e orientação. A primeira etapa do sistema aqui apresentado é o que se entende por registro do objeto na cena. O registro é formulado como um problema de otimização que busca a matriz de transformação que melhor associe a nuvem de pontos 3D relativa ao modelo do manipulador à nuvem da cena observada. Assim, vários métodos podem ser abordados, tais como gradiente descendente, ICP, RANSAC, entre outros. (Tam et al., 2013)

Feito o registro do manipulador, é fundamental que se acompanhe o seu movimento, o que caracteriza a segunda etapa do sistema descrito. Muito tem sido feito no que diz respeito ao rastreio 2D (Wang e Popović, 2009; Stenger et al., 2006) e 3D (Bray et al., 2004) de uma mão humana, inclusive em sistemas baseado em modelos (Heap e Hogg, 1996; Oikonomidis, Kyriazis e Argyros,). Manter o rastro do manipulador se traduz, matematicamente, na renovação contínua da matriz de transformação que alinha o modelo à cena. Assim, tomando o resultado do registro como ponto de partida, o algoritmo de rastreio 3D foca na constante atualização da pose do objeto em questão.

O conhecimento da pose do manipulador ao longo do tempo é essencial para a detecção do estado de sua garra, o que, por sua vez, pode ser muito útil em aplicações RRI, onde robôs demandem ciência uns dos outros. Tarefas colaborativas entre robôs, como a manipulação de um objeto pesado ou frágil, podem demandar de um manipulador o conhecimento preciso do estado de outros. Um sistema de visão pode beneficiar até problemas mais simples, como deslocamento de objetos.

Em resumo, propõe-se um sistema de visão capaz de determinar o estado do *gripper* de um dado robô manipulador, um problema que vai do registro de um modelo 3D em uma dada cena, isto é, da determinação de sua pose inicial, até o rastreio contínuo não só da evolução desta pose com o tempo, mas também, como consequência, do estado de sua garra. Tal sistema de visão deve tornar o robô mais ciente do ambiente que o cerca, expandindo sua capacidade de interação com demais robôs.

No Capítulo 2, os objetivos estabelecidos para este Trabalho de Conclusão de Curso são apresentados. O Capítulo 3 descreve conceitos, paradigmas e algoritmos considerados essenciais para o sistema proposto. No Capítulo 4, o ambiente de teste utilizado em laboratório é descrito e o sistema desenvolvido é detalhado, expandido em um conjunto de subsistemas. Os resultados experimentais, limitações e potenciais aprimoramentos são discutidos no Capítulo 5. Por fim, o Capítulo 6 conclui este documento.

2 Objetivo

O objetivo deste Trabalho de Conclusão de Curso é o desenvolvimento de um sistema de visão computacional 3D capaz de rastrear o manipulador de um braço robótico, com o acompanhamento do estado de sua garra, isto é, se está aberta ou fechada. A partir de uma câmera 3D, a orientação da garra deve ser continuamente estimada, assim como seu estado.

Especificamente, podem ser destacadas as seguintes metas:

- Familiarizar-se com o sistema operacional ROS (Robot Operating System);
- Realizar o registro do manipulador robótico na cena 3D, utilizando-se de modelos tridimensionais CAD;
- Buscar algoritmos para rastreamento contínuo de um modelo 3D;
- Propor algoritmo de detecção do estado da garra do manipulador (aberto/fechado)
- Desenvolver uma interface de monitoramento para usuário.

3 Fundamentação

Este capítulo apresenta os principais conceitos e paradigmas envolvidos no sistema proposto, no intuito de prover maior embasamento teórico ao que será explanado posteriormente. São aqui descritos o conceito de nuvem de pontos, tipo estrutural de dados tratados no problema, a motivação para a adoção de uma abordagem com base em modelos e os algoritmos utilizados nos diferentes subsistemas envolvidos. Então, uma breve discussão é feita a respeito das atribuições de cada algoritmo utilizado e de como eles estão relacionados.

3.1 Nuvem de Pontos

Uma nuvem de pontos (*point cloud*) pode ser definida como um conjunto finito de pontos expressos num mesmo sistema de coordenadas. A disponibilidade de sistemas de imageamento tridimensionais (3D) de baixo custo tem alavancado significativamente a pesquisa no campo da robótica (Mateo et al., 2012; Litomisky, 2012; Schnabel, Wahl e Klein, 2007).

Em geral, estes sistemas fornecem um conjunto relativamente grande de pontos de medição da cena visualizada; este conjunto, denominado de nuvem de pontos, deve ser especificado segundo um determinado sistema de coordenadas, podendo representar, por exemplo, a disposição tridimensional de um objeto no espaço, bem como do ambiente no qual o objeto está imerso. É importante ressaltar que nuvens de pontos não podem ser diretamente utilizadas assim que adquiridas pelos sensores, mas normalmente requerem algum método de reconstrução 3D de superfície.

Neste trabalho, o processamento efetuado sobre os dados da nuvem de pontos é feito utilizando-se a Point Cloud Library - PCL (Rusu e Cousins, 2011). A PCL é uma biblioteca escrita em C++ que oferece suporte a uma série de algoritmos de processamento 3D, como filtragem 3D, métodos de segmentação e reconhecimento de objetos. Os dados de entrada para estes algoritmos são medições oriundas dos sensores de imagem do tipo RGB-D (informação de cores RGB e profundidade por pixel).

Uma motivação adicional para o uso da PCL é sua integração com o sistema ROS, utilizado neste projeto. Além disso, a documentação disponível oferece exemplos elucidativos que podem certamente ser usados como pontos de partida no desenvolvimento de qualquer projeto. No sistema aqui descrito, a PCL é usada extensivamente, desde o processamento primordial dos dados crus do sensor RGB-D até a identificação do estado da garra robótica.

3.2 Projeto Baseado em Modelos

A motivação por trás da adoção de um sistema com base em modelos é trabalhar com um nível elevado de abstração e se utilizar de representações independentes de plataforma (Estevez e Marcos, 2012). Nesta abordagem, faz-se uso de modelos para representar os elementos de um sistema e as relações entre eles (Causo et al., 2010). Especificamente para este projeto, modelos CAD são adotados para a representação do manipulador robótico e de sua garra. O modelo CAD para o braço robótico Jaco^2 com a garra aberta é disponibilizado online pela fabricante Kinova Robotics e, a partir deste, diferentes modelos podem ser gerados, utilizando-se de algum modelador CAD, como o *FreeCAD*, por exemplo. Os modelos CAD para garra aberta e fechada são apresentados na Fig. 1.

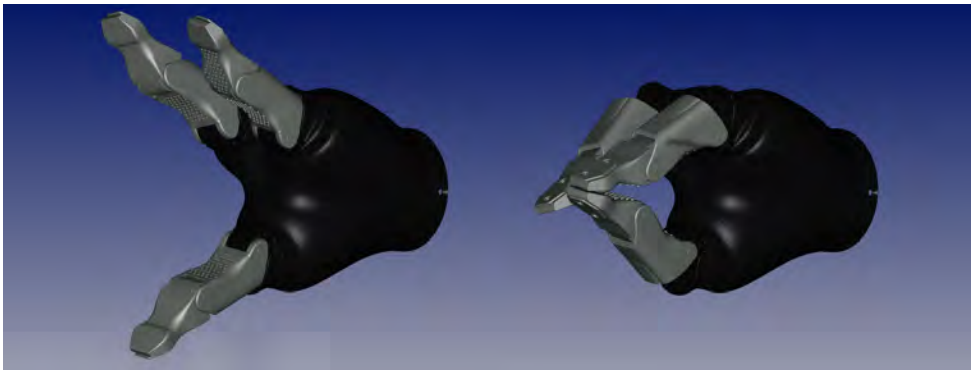


Figura 1 – Modelos CAD para a garra aberta e fechada do manipulador Jaco^2. A parte que se refere à palma foi removida do modelo original em concordância com o braço robótico utilizado.

Como o processamento de dados é feito via PCL, os modelos CAD não podem ser usados diretamente. Assim, os modelos obtidos da garra em ambos os estados devem ser transformados em conjuntos de pontos tridimensionais, levando ao que se chama de nuvens de pontos estruturados. As nuvens de pontos geradas a partir dos modelos da Fig. 1 são expostas na Fig. 2. Este procedimento pode ser implementado por modeladores paramétricos gratuitos, como o *FreeCAD* e o *MeshLab*.

O uso de um modelo total para a garra robótica fornece informação tridimensional completa para seu rastreio. Isto contribui para mitigar a dificuldade na estimativa da orientação de um objeto 3D quando se detém apenas de visão parcial do mesmo (PCL, 2016), o que favorece a abordagem baseada em modelos à livre deles.

Associado a cada nuvem de pontos há um sistema de coordenadas (frame) que não deve ser desconsiderado, especialmente em se tratando de tarefas de rastreio e detecção. É importante garantir que tais sistemas sejam conhecidos e bem definidos, assim como as matrizes de transformação de coordenadas, definidas a partir do sistema da câmera. Na Fig. 3, é apresentado o sistema de coordenadas da garra do manipulador, definido de

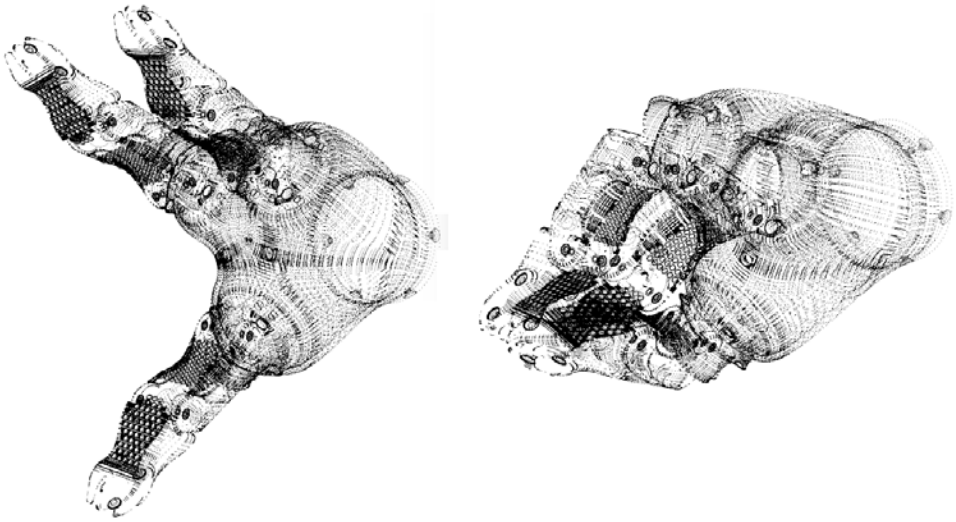


Figura 2 – Nuvens de pontos extraídas dos modelos CAD na Fig. 1.

modo que sua origem se localizasse no centro da região de apreensão, a cerca de cinco centímetros (na direção z) da palma da mão. Esta escolha é motivada pela intuição de simplificar o posterior rastreio da garra e a determinação de seu estado. Quanto aos eixos, o z é posto no sentido que se afasta da palma da mão, o y para cima e o eixo x para a esquerda da garra.

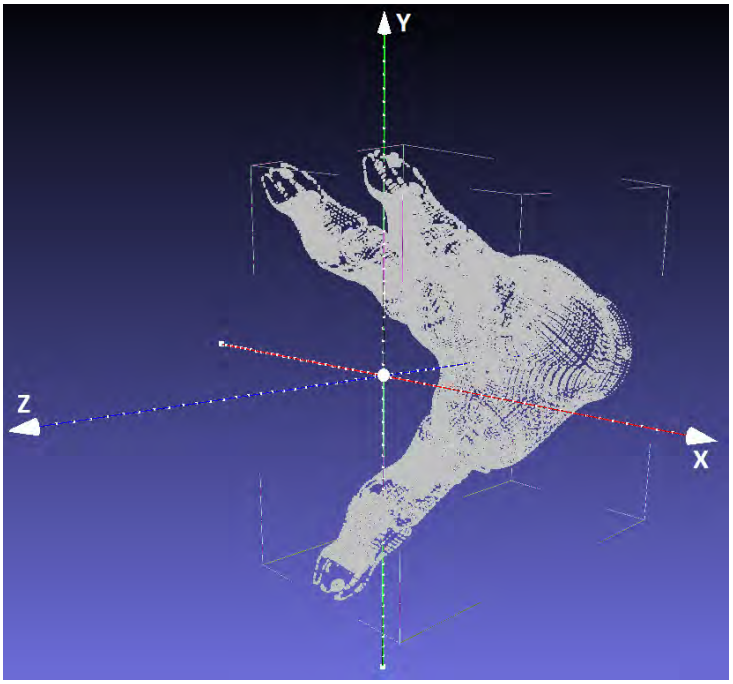


Figura 3 – Sistema de coordenadas de referência para o modelo aberto da garra. O sistema utilizado para o modelo fechado segue as mesmas convenções.

3.3 Sample Consensus Initial Alignment

O Sample Consensus Initial Alignment (SAC-IA) é um método de otimização que visa o alinhamento consistente entre duas ou mais nuvens de pontos 3D correspondentes a visões parciais distintas, possivelmente sobrepostas, de uma cena. Esta tarefa é o que se denomina registro 3D de uma nuvem de pontos e pode ser formulada como um problema de otimização, uma vez que busca a matriz de transformação, isto é, rotação e translação, que melhor alinhe duas nuvens de pontos, minimizando, portanto, uma determinada função de erro.

Este método fundamenta-se na definição de um tipo específico de características (*features*) denominado *Point Feature Histogram* (PFH). O algoritmo SAC-IA, na verdade, opera nas nuvens de pontos extraindo os chamados *Fast Point Feature Histogram* (FPFH), que constituem um aprimoramento, no sentido de complexidade computacional, dos PFH. Convém, portanto, descrevê-los.

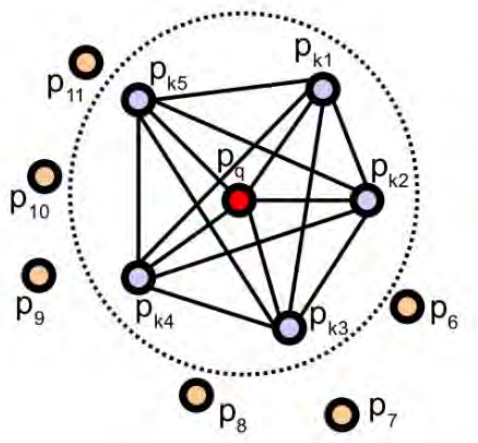
3.3.1 Point Feature Histogram

A definição dos Point Feature Histograms (PFH) remonta ao objetivo de identificar um espaço de características no qual pontos 3D dispostos em superfícies geométricas primitivas possam ser facilmente identificados. Assim, o poder discriminatório deste espaço de características (*feature space*) deve ser alto o suficiente para que os pontos da mesma superfície sejam conjuntamente classificados (Rusu et al., 2008).

Estes histogramas consitutem características locais invariantes à pose e representam, para um dado ponto p, as propriedades de modelo da superífice a ele subjacente. Basicamente, o cálculo de um PFH em um ponto p requer o conhecimento de suas coordenadas 3D e de estimativas das normais à superfície. Assim, é feito da seguinte forma: i) para cada ponto p, define-se a vizinhança-k de p como o conjunto formado pelos pontos vizinhos que estejam envolvidos em uma esfera de raio r centrada em p; ii) para cada par de pontos p_i e p_j, com $i \neq j$, na vizinhança-k de p, e considerando suas respectivas estimativas de normais n_i e n_j, define-se um sistema Darboux de coordenadas uvn ($u = n_i$, $v = (p_j - p_i) \times u$, $w = u \times v$) e são calculadas as variações angulares entre as normais segundo as seguintes equações:

$$\begin{aligned} \alpha &= v \cdot n_j \\ \phi &= (u \cdot (p_j - p_i))/\|p_j - p_i\| \\ \theta &= arctan(w \cdot n_j, u \cdot n_j) \end{aligned} \tag{3.1}$$

A Fig. 4 apresenta um diagrama que descreve a região de influência sobre o cálculo de um PFH para um dado ponto p_q (query point). Marcado em vermelho, o ponto p_q é disposto no centro de uma circunferência (esfera em 3D) de raio r e todos os seus k vizinhos, distantes de p_q não mais que r, estão interconectados em uma malha.



Fonte: Rusu, Blodow e Beetz, 2009

Figura 4 – Diagrama de região de influência para um PFH. O ponto de análise (vermelho) e seus k vizinhos (azul) estão interconectados em uma malha.

Para cada característica da tupla (α, ϕ, θ) definem-se limites teóricos (máximo e mínimo) e a faixa dinâmica de cada uma é discretizada em d subdivisões, o que gera um espaço 3D com d^3 divisões. Finalmente, a representação do PFH se dá pela criação de um histograma sobre o espaço de divisões, contando o número de ocorrências de cada um delas ao longo do conjunto de tuplas obtido.

3.3.2 Fast Point Feature Histogram

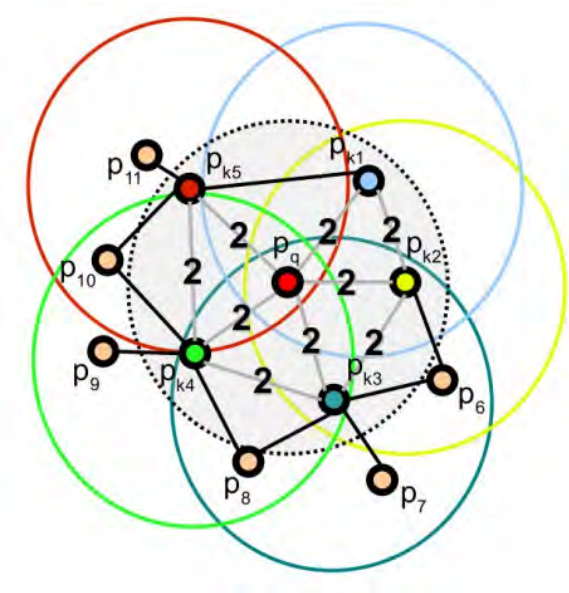
A complexidade computacional do algoritmo Point Feature Histogram para uma dada nuvem com n pontos é de $O(n \cdot k^2)$ (Rusu, Blodow e Beetz, 2009), onde k é o número de vizinhos para cada ponto na nuvem. Para aplicações em tempo real, densas vizinhanças de pontos podem se toram um gargalo significativo no registro 3D. Os Fast Point Feature Histograms (FPFH) nada mais são que versões simplificadas dos PFH, a partir dos quais a complexidade computacional é reduzida para $O(n \cdot k)$, mas ainda mantendo a maior parte do poder discriminativo dos Point Feature Histograms.

Para simplificar a construção do histograma, o procedimento é feito da seguinte maneira: i) em um primeiro passo, para cada ponto p, serão computadas apenas as relações (Equação 3.1) entre p e seus vizinhos, construindo assim o Simplified Point Feature Histogram (SPFH); ii) em seguida, para cada ponto, define-se novamente sua vizinhança-k e, utilizando-se dos valores SPFH como pesos, calcula-se o histograma final de p (FPFH) como segue:

$$FPFH(p) = SPFH(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k} \cdot SPFH(p_k) \tag{3.2}$$

onde ω_k representa a distância entre p e p_k em um dado espaço métrico.

O diagrama da região de influência exposto na Fig. 5 ilustra como ocorre a computação do FPFH: cálculo inicial dos histogramas simplificados e posterior ponderação dos mesmos para obtenção do histograma final.



Fonte: Rusu, Blodow e Beetz, 2009

Figura 5 – Diagrama de região de influência para um FPFH. O ponto de análise (vermelho) está conectado apenas a seus k vizinhos (envolvidos pelo círculo cinza). As conexões marcadas com 2 contribuirão duas vezes para o FPFH.

3.3.3 O Algoritmo SAC-IA

O algoritmo SAC-IA é, primeiramente, um método de Sample Consensus, como, por exemplo, o RANSAC (*Random Sample Consensus*) e, portanto, tem como objetivo a manutenção das mesmas relações geométricas entre os pontos correspondentes, sem que seja preciso testar todas as possíveis combinações do conjunto de correspondências (Rusu, Blodow e Beetz, 2009).

Assim, a abordagem pelo consenso de amostra (*sample consensus*) é adotada, para evitar não só alta complexidade computacional, mas também a eventualidade de se encontrar preso em um mínimo local. Faz-se um levantamento de um grande número de correspondências candidatas, que podem ser classificadas segundo o seguinte esquema:

1. Selecione s amostras de pontos em P, de modo que as distâncias entre cada par seja superior a um parâmetro de distância mínima d_{min} defnido pelo usuário.

2. Para cada ponto de amostra, encontre a lista dos pontos em Q cujos histogramas são similares ao da amostra. Destes, selecione um aleatoriamente e o assuma como sendo o ponto correspondente à amostra.

3. Compute a transformação rígida definida pelos pontos de amostra e seus correspondentes e calcule uma métrica de erro que traduza a qualidade da transformação.

Os três passos descritos são repetidos e a transformação que apresentar a melhor métrica de erro será armazenada e utilizada para um alinhamento inicial. Posteriormente, algoritmos de otimização não-linear podem ser utilizados para ajuste fino. Um exemplo, exposto na Fig. 6, pode ser encontrando em Rusu, Blodow e Beetz, 2009, onde foram combinados o SAC-IA com o refino do algoritmo de otimização local não-linear de Levenberg-Marquardt.

Fonte: Rusu, Blodow e Beetz, 2009

Figura 6 – Da esquerda para a direita: duas visões parciais do modelo de um coelho antes do alinhamento; resultados obtidos com a aplicação da solução encontrada pelo SAC-IA.

Em resumo, o SAC-IA executa buscas rápidas em um exaustivo espaço de correspondências FPFH para encontrar uma boa solução de alinhamento, que pode, por sua vez, ser posteriormente ajustada usando um método de otimização não-linear. No contexto deste projeto, o algoritmo fornece a primeira suposição em relação à localização do manipulador robótico na cena 3D. Levando em consideração que a determinação da pose da garra deve ser precisa, para os devidos fins, insiste-se na necessidade do citado refinamento da matriz de transformação rígida encontrada.

3.4 Iterative Closest Point

Como já sugerido na seção anterior, a aplicação do algoritmo SAC-IA não basta para o registro satisfatório de nuvens de pontos, mas provê um bom alinhamento inicial, passível de aprimoramento. Para refinar este resultado, o algoritmo Iterative Closest Point (ICP) foi adotado, um método bastante utilizado na litetura para minimizar a diferença entre duas nuvens de pontos (Chetverikov, Stepanov e Krsek, 2005; Kaneko, Kondo e Miyamoto, 2003; Luck, Little e Hoff, 2000).

O Iterative Closest Point sempre converge monotonicamente para o mínimo local mais próximo, segundo uma métrica de distância média quadrática, além de apresentar uma rápida taxa de convergência durante as primeiras iterações. Portanto, dado um conjunto inicial adequado de rotações e translações, aqui fornecidos pelo SAC-IA, é possível minimizar globalmente a métrica de distâcia média quadrática para todos os seis graus de de liberdade (Besl e McKay, 1992).

Este algoritmo pode ser usado para o registro de distintas representações geométricas de dados, como conjuntos de pontos, de segmentos de linha, de triângulos, de curvas e de superfícies, implícitas ou paramétricas, o que cobre boa parte das aplicações que demandariam registro 3D. Na prática, o método demanda apenas que os dados sejam descritos como conjunto de pontos, isto se já não se encontrarem neste formato.

Fundamentalmente, o método ICP requer que se compute, para um dado ponto, o ponto mais próximo a ele, pertencente a determinada forma geométrica. As distintas formas para cálculo deste ponto, a depender do tipo de forma geométrica em que se encontra (conjunto de pontos, linhas, curvas, etc.) são discutidas em Besl e McKay, 1992. Ainda, é apresentado um método para obtenção do vetor (quaternion) de registro pelos mínimos quadrados, correspondente a uma dada matriz de transformação.

3.4.1 O Algoritmo ICP

Seja P um conjunto com N_P pontos corresnpodentes à cena e X um conjunto com N_X primitivas (pontos, linhas, etc.) relativas ao modelo. Inicializando $P_0 = P$ e sendo k o índice de iteração do algoritmo, este pode ser descrito nas seguintes operações:

1. Para os pontos de P_k, calcular os pontos mais próximos em X: $Y_k = C(P_k, X)$, onde Y_k é o conjunto de pontos mais próximos e C é o operador correspondente.

2. Calcular o quaternion $\vec{q_k}$ relativo à transformação (registro), no sentido de mínimos quadrados, entre P_0 e Y_k: $(\vec{q_k}, d_k) = Q(P_0, Y_k)$, onde d_k é a métrica de distância entre os conjuntos e Q é o operador correspondente.

3. Aplicar o registro obtido: $P_{k+1} = \vec{q_k}(P_0)$.

4. Finalizar o algoritmo quando a variação no erro médio quadrático for inferior a um limiar pré-definido $\tau > 0$, parâmetro que define a precisão do registro: $d_k - d_{k+1} < \tau$.

3.5 Particle Filter Algorithm

Os métodos descritos anteriormente (SAC-IA e ICP) foram utilizados para fins de registro de um modelo em uma cena. No entanto, uma vez registrado, é preciso que se mantenha o rastro do objeto de interesse à medida em que este se desloca no ambiente. Para lidar com o rastreio do modelo, o algoritmo do Filtro de Partícula (*Particle Filter Algorithm - PFA*) foi escolhido, um método sequencial de Monte Carlo (Arulampalam et al., 2002) baseado na representação de distribuições de probabilidades através de partículas.

Segundo Erol et al., os métodos propostos para rastreamento podem ser classificados em dois tipos. O primeiro deles se baseia em uma busca local, atendo-se apenas à melhor estimativa a cada frame e, por isso, é chamado rastreamento de única hipótese (Single Hypothesis Tracking - SHT). O segundo tipo, exemplificado pelo Filtro de Partícula adotado, é o chamado rastreamento de múltiplas hipóteses (Multiple Hypothesis Tracking - MHT), cuja ideia básica é a de manter várias estimativas a cada frame, de modo que, caso a melhor delas falhe, o sistema ainda seja capaz de manter o rastro a partir das restantes.

O príncipio de um algoritmo MHT é melhor capturado pela filtragem Bayesiana, que visa, basicamente, o cálculo de uma função de densidade de probabilidade (fdp) *a posteriori* para o estado de um sistema a partir de toda informação disponível, incluindo observações realizadas.

A seguir, é definido o problema de rastreamento sob a perspectiva Bayesiana, para melhor fundamentar o algoritmo do Filtro de Partícula.

3.5.1 Rastreamento Bayesiano não-linear

Para inferir dados de um sistema dinâmico, pelo menos dois modelos são necessários: um que descreva a evolução do estado do sistema no tempo (*modelo do sistema*), isto é, a relação entre o estado atual e o(s) passado(s); e um modelo que relacione as medições ruidosas realizadas ao estado do sistema (*modelo de medição*). Assume-se que estes modelos são disponíveis em forma probabilística (Arulampalam et al., 2002).

Esta formulação se mostra ideal para uma abordagem Bayesiana do problema, que enseja, como citado anteriormente, a construção da fdp a posteriori do estado do sistema, com base no passado e em novas medições feitas. Em princípio, pode-se obter uma estimativa ótima do estado do sistema a partir da sua fdp e, como uma nova estimativa é geralmente requerida para todo novo conjunto de medições, convém utilizar-se de um filtro que explore recursividade.

O filtro em questão consiste de dois estágios: o de predição e o de atualização. O estágio de predição faz uso do modelo do sistema para predizer a fdp do próximo estado e, como o sistema é afetado por perturbações, esta predição pode transladar, deformar e

espalhar a fdp. Já o estágio de atualização fará uso das medições recém-tomadas para, através do teorema de Bayes, modificar a pdf do estágio de predição.

Para a formulação do problema, como descrito em Arulampalam et al., define-se $\{\mathbf{x}_k, k \in \mathbb{N}\}$ como a sequência de estados do sistema, cuja evolução no tempo discreto k se dá por

$$\mathbf{x}_k = \mathbf{f}_k\left(\mathbf{x}_{k-1},\ \mathbf{v}_{k-1}\right) \tag{3.3}$$

onde $\mathbf{f}_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_v} \rightarrow \mathbb{R}^{n_x}$ é uma função possivelmente não linear do estado $\mathbf{x}_{k-1}$ e da sequência i.i.d. de ruído de processo dada por $\{\mathbf{v}_{k-1}, k \in \mathbb{N}\}$. A equação 3.3 define, portanto, o chamado *modelo do sistema*. Ainda, define-se $\mathbf{z}_k$ como o conjunto de medições realizadas no tempo k, modeladas segundo

$$\mathbf{z}_k = \mathbf{h}_k\left(\mathbf{x}_k,\ \mathbf{n}_k\right) \tag{3.4}$$

onde $\mathbf{h_k} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_n} \rightarrow \mathbb{R}^{n_z}$ é uma função possivelmente não linear do estado $\mathbf{x}_k$ e da sequência i.i.d. de ruído de medição dada por $\{\mathbf{n}_{k-1}, k \in \mathbb{N}\}$. Analogamente, a equação 3.4 define o *modelo de medição*. Por fim, sejam n_x, n_z, n_v e n_n as dimensões dos vetores de estado, de medição, de ruído de processo e de ruído de medição, respectivamente.

O objetivo do rastreamento é buscar por estimativas filtradas do vetor de estados $\mathbf{x}_k$ com base no conjunto de todas as medições disponíveis até o tempo k, isto é, o conjunto $\mathbf{z}_{1:k} = \{\mathbf{z}_i, i = 1, ..., k\}$. Do ponto de vista bayesiano, o problema consiste em calcular recursivamente um certo grau de crença a respeito do estado $\mathbf{x}_k$, para distintos valores, condicionados às medições realizadas até k, ou seja, busca-se construir a função de densidade de probabilidade $p\left(\mathbf{x}_k|\mathbf{z}_{1:k}\right)$. Para fins de inicialização, assume-se $p\left(\mathbf{x}_0|\mathbf{z}_0\right) \equiv p\left(\mathbf{x}_0\right)$ como a fdp *a priori*.

Assim, a distribuição *a posteriori* $p\left(\mathbf{x}_k|\mathbf{z}_{1:k}\right)$ pode ser obtida recursivamente em duas etapas: predição e atualização. Na primeira etapa (predição), calcula-se a distribuição *a priori* do estado do sistema no tempo k, isto é, $p\left(\mathbf{x}_k|\mathbf{z}_{1:k-1}\right)$. Vale enfatizar que esta distribuição é condicionada às medições passadas, mas não às tomadas em k e, portato, ainda caracteriza, para o tempo k, uma distribuição *a priori*, já que é anterior ao conhecimento das observações $\mathbf{z}_k$.

Dessa forma, a etapa de predição faz uso do *modelo do sistema* (3.3) para obter a fdp *a priori* por meio da equação de Chapman-Kolmogorov

$$\begin{aligned}
p\left(\mathbf{x}_k|\mathbf{z}_{1:k-1}\right) &= \int p\left(\mathbf{x}_k, \mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}\right) d\mathbf{x}_{k-1} \\
p\left(\mathbf{x}_k|\mathbf{z}_{1:k-1}\right) &\overset{(a)}{=} \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_{1:k-1}\right) p\left(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}\right) d\mathbf{x}_{k-1} \\
p\left(\mathbf{x}_k|\mathbf{z}_{1:k-1}\right) &\overset{(b)}{=} \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}\right) p\left(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}\right) d\mathbf{x}_{k-1}
\end{aligned} \tag{3.5}$$

Em (a), a equação original de Chapman-Kolmogorov é simplesmente expandida pela regra da cadeia, enquanto que, em (b), aplica-se o fato de que o modelo do sistema

(3.3) constitui um processo de Markov de primeira ordem, de modo que $p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_{1:k-1})$ equivale à distribuição $p(\mathbf{x}_k|\mathbf{x}_{k-1})$. Esta distribuição é o modelo probabilístico da evolução do estado, definido pela Equação 3.3 e pelas estatísticas conhecidas de $\mathbf{v}_{k-1}$.

Na segunda etapa (atualização), pressupõe-se a disponibilidade de um vetor de medições $\mathbf{z}_k$ no instante k. Com isso, as novas medições são utilizadas para modificar a fdp *a priori*, obtendo, portanto, a fdp *a posteriori* desejada: $p(\mathbf{x}_k|\mathbf{z}_{1:k})$. A atualização se dá através da regra de Bayes

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{3.6}$$

em que a constante de normalização

$$p(\mathbf{z}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{z}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k \tag{3.7}$$

depende da distribuição $p(\mathbf{z}_k|\mathbf{x}_k)$, que é definida pelo *modelo de medição* (3.4) e pelas estatísticas conhecidas de $\mathbf{n}_{k-1}$.

As relações de recorrência 3.5 e 3.6 constituem a base para a solução ótima bayesiana do problema. Na Figura 7 é apresentado um diagrama ilustrativo das etapas envolvidas no rastreamento bayesiano.

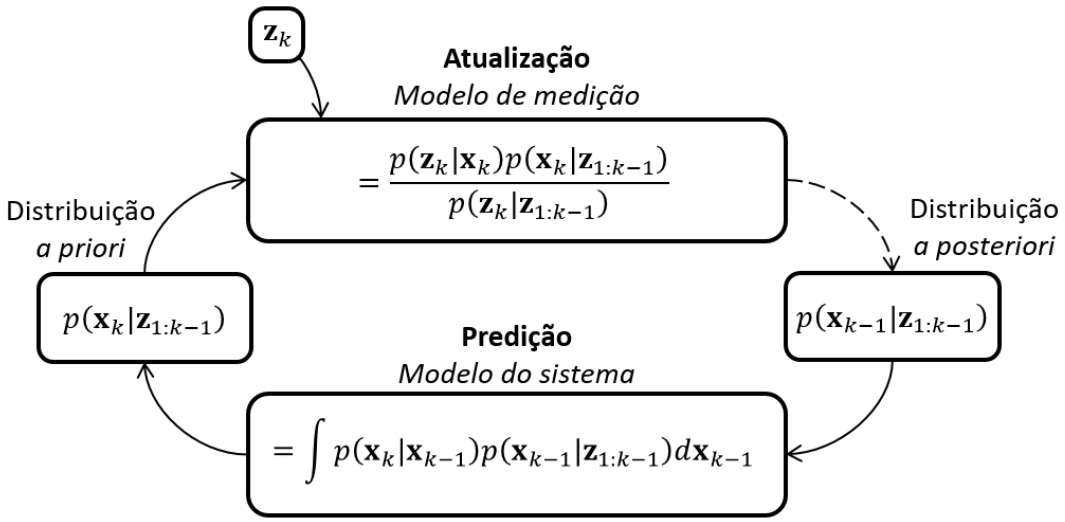


Figura 7 – As duas etapas que definem o rastreamento bayesiano não-linear. Para evitar confusão quanto aos índices, o arco tracejado representa o fim de um loop temporal, onde k seria incrementado.

Ao problema do rastreio não-linear associa-se, assim, uma solução Bayesiana ótima. A propagação recursiva da fdp *a posteriori* constitui, a princípio, uma solução conceitual que, em geral, não pode ser resolvida analiticamente. Para certos casos mais restritos, a solução ótima pode ser rastreável, como no filtro de Kalman e o no filtro em malha (*grid-based filter*). Nos casos gerais em que a solução não é rastreável, diversos métodos existem para aproximar a solução Bayesiana ótima, tais como o filtro de Kalman estendido, os filtros em malha aproximada e os filtros de partícula.

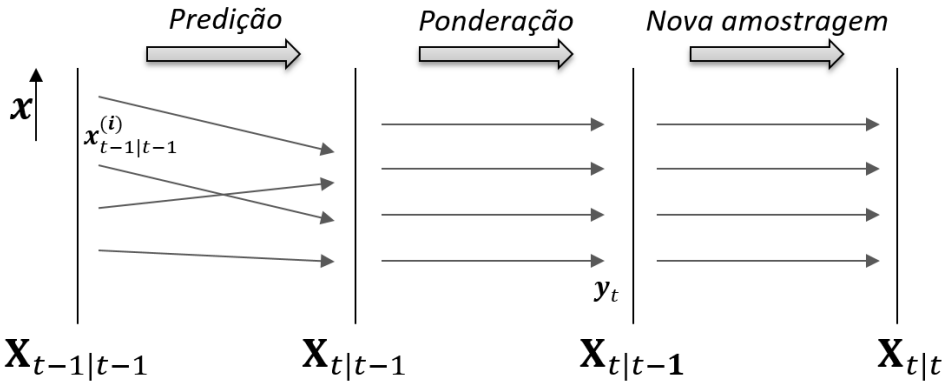
3.5.2 O Algoritmo PFA

O Filtro de Partícula (FP) foi o método adotado para implementação do rastreio da nuvem de pontos referente ao modelo da garra do manipulador. No que diz respeito à classificação descrita por Erol et al., o filtro de partícula constitui um algoritmo de rastreamento com base em múltiplas hipóteses, pois procura estimar múltiplas poses para o mesmo quadro temporal.

Além do FP ser uma opção recorrente no desenvolvimento de sistemas preditivos para estimação e rastreio de pose (Kerdvibulvech e Saito, 2009; Causo et al., 2010), ele também encontra suporte para desenvolvimento na PCL (Point Cloud Library), que provê um *framework* tipado para aplicação direta do PFA.

O algoritmo do PFA é uma técnica bastante conhecida para implementar o filtro Bayesiano recursivo discutido na seção anterior. A ideia básica do filtro é representar uma distribuição de probabilidade arbitrária a partir de amostras ponderadas (partículas), extraídas, por sua vez, de outra distribuição de probabilidade, denominada *densidade de importância*. Os pesos utilizados representam as probabilidades de ocorrência de cada amostra (Erol et al., 2007).

O filtro pode ser descrito em duas etapas bem definidas: a primeira, denominada amostragem por importância (importance sampling), consiste em coletar partículas (amostras) de uma dada distribuição (grau de crença) e ponderá-las de acordo com sua probabilidade de ocorrência; em seguida, fazendo uso de observações tomadas, realiza-se nova amostragem, atualizando-se a crença (distribuição). Na Figura 8 é apresentado um diagrama ilustrativo simplificado com as principais operações envolvidas no Filtro de Partícula.



Fonte: diagrama simplificado a partir de PCL, 2016

Figura 8 – Um diagrama ilustrativo para o processo envolvido no Filtro de Partícula.

3.6 Cálculo de Similaridade entre Nuvens de Pontos

A decisão pelo desenvolvimento de um sistema com base em modelos naturalmente levou, entre outros fatores, à necessidade de definir alguma medida de similaridade entre duas nuvens de pontos. Intuitivamente, pode-se esperar que, ao depender de um conjunto de modelos, o objetivo do detector de estados seja escolher, dentre eles, aquele cuja similaridade com a nuvem da cena seja maximizada, assim classificando-a entre os possíveis estados.

Buscou-se, portanto, uma maneira simples e eficiente de estimar a similaridade entre as duas nuvens de pontos. É importante ressaltar que o procedimento escolhido foi proposto levando em consideração as características do sistema aqui descrito. Por exemplo, não há necessidade deste algoritmo em se preocupar com o alinhamento entre as nuvens nem com extração de características, uma vez que estas tarefas são atribuídas a outros subsistemas. Assim, não se deve assumir que a estimativa de similaridade proposta se aplica a um cenário geral; na verdade, parte-se do pressuposto que os procedimentos de registro e rastreio já foram realizados.

Dada a particularidade da tarefa para a qual esta solução foi proposta, as razões para adotá-la serão detalhadas ao longo da discussão geral do sistema. Para dar maior suporte a esta discussão, é apresentada a seguir uma descrição matemática do procedimento para cálculo de similaridade.

Sejam C_1 e C_2 duas nuvens de pontos com N e M pontos $3D$, respectivamente. Ainda, seja $C_x(i)$ o i-ésimo ponto percentente à nuvem C_x. A similaridade da nuvem C_1 em relação à nuvem C_2 é definido como

$$S(C_1|C_2) \triangleq \frac{1}{N}\sum_{i=1}^{N}\delta(C_1(i), C_2) \tag{3.8}$$

onde $\delta(C_1(i), C_2)$ representa o pertencimento do ponto $C_1(i)$ à nuvem C_2, definido como

$$\delta(C_1(i), C_2) = \begin{cases} 1 & \text{se } \exists\, C_2(j) : d(C_1(i), C_2(j)) < r,\ 1 \leq j \leq M \\ 0 & \text{c.c.} \end{cases} \tag{3.9}$$

O operador $d(C_1(i), C_2(j))$ na Equação 3.9 corresponde simplesmente à distância Euclidiana entre seus argumentos (pontos 3D) e o raio r especifica a flexibilidade em decidir quanto ao pertencimento de um dado ponto. De maneira mais direta, a similaridade de C_1 em relação a C_2 é o percentual dos pontos em C_1 que possuem pelo menos um ponto vizinho em C_2. Centra-se, em cada ponto de C_1, uma esfera de raio r e, caso dentro desta esfera exista pelo menos um ponto de C_2, decide-se pelo pertencimento do ponto.

Embora tais pontos não pertençam às mesmas nuvens, são considerados vizinhos puramente no sentido geométrico, já que suas coordenadas são próximas o suficiente. Esta

maneira simplificada para comparar nuvens é limitada pela suposição de que elas já se encontrem no mesmo sistema de coordenadas, o que explica a necessidade do alinhamento prévio.

Vale destacar que esta definição de similaridade não é simétrica (Equação 3.10), devido principalmente ao fato de que as nuvens provavelmente não apresentarão o mesmo número de pontos nem terão pontos coincidentes no espaço.

$$S(C_1|C_2) \neq S(C_2|C_1) \tag{3.10}$$

No contexto deste sistema, esta assimetria mostrou-se uma vantagem pois, como a similaridade envolve iterar sobre os pontos de uma das nuvens, não haveria necessidade de fazê-lo com a nuvem do modelo, que fatalmente apresentaria um número bem maior de pontos. Assim, a similaridade medida é da nuvem observada na cena em relação à nuvem do modelo da garra, não o contrário.

3.7 As Atribuições dos Algoritmos Utilizados

Esta seção busca contextualizar a interação entre os algoritmos apresentados, destacando as atribuições de cada um deles e a relação de entrada-saída existente entre os dados por eles gerados. No Capítulo 4, isto será feito em maiores detalhes, à medida em que o sistema proposto é descrito.

Como apresentado anteriormente, o primeiro passo para o rastreio do manipulador é a determinação de sua pose na cena 3D, isto é, definir a matriz de transformação que alinha a nuvem do modelo à nuvem da cena, ao que se denomina registro.

Para registrar o modelo, são usados os algoritmos SAC-IA e ICP. O SAC-IA realiza o alinhamento inicial, que serve como entrada para o ICP, algoritmo utilizado para refino e obtenção do alinhamento final. Assim, o registro tem como saída uma matriz de transformação.

Uma vez feito o registro, deve-se manter o rastreio do movimento do efetuador. Assim, o algoritmo PFA é utilizado para realizar o rastreamento do modelo, a partir da constante atualização da matriz de transformação citada, tendo como estado o inicial o alinhamento resultante do registro.

A partir do rastreio, o conhecimento da pose do efetuador pode ser usado para isolar a nuvem de pontos corresponde a ele. Com isso, o algoritmo proposto para cálculo de similaridade entre nuvens de pontos é utilizado com a nuvem isolada do efetuador e a nuvem dos modelos de garra aberta e fechada. A medição possibilitará, portanto, a classificação do estado da garra.

4 O Sistema Proposto

Neste capítulo, é apresentado o sistema proposto para detecção e rastreamento da garra, e inferência de seu estado, segundo sua estruturação no sistema ROS. Primeiramente, é descrito o ambiente de teste utilizado ao longo do desenvolvimento do sistema. Em seguida, com o subsídio dos conceitos, algoritmos e ferramentas previamente apresentados, a arquitetura geral ROS é apresentada e cada nó descrito individualmente, enquanto que a interação entre eles é esclarecida.

4.1 Ambiente de Teste

Nesta seção, é descrita a montagem em laboratório na qual o sistema foi desenvolvido e testado, bem como os dispositivos utilizados. A partir disso, será mais fácil compreender a posterior descrição do sistema.

Como mostrado na Figura 9, a montagem consiste basicamente de um módulo sensor de profundidade Kinect, fixado em uma parede, e de um braço robótico Kinova Jaco^2 6-DOF com três dedos e base estacionária, disposto sobre uma mesa. Quanto à garra do manipulador, em foco no lado inferior direito da imagem, a parte correspondente à palma não foi utilizada, como se encontraria no modelo original da garra.

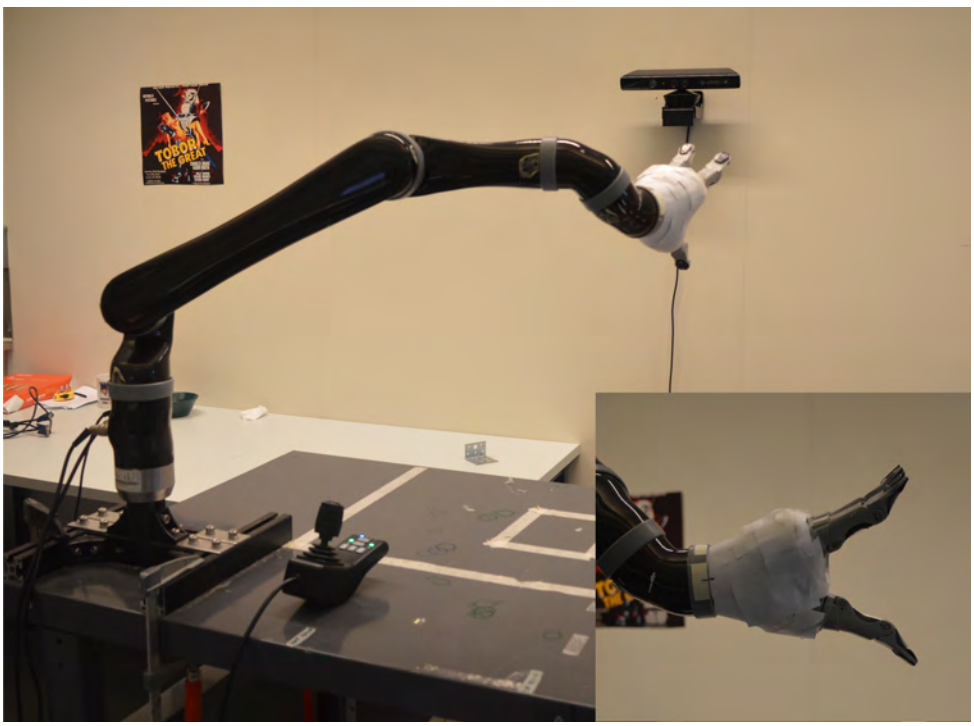


Figura 9 – Montagem experimental usando o sensor Kinect e o manipulador Jaco^2 6-DOF.

Não havia necessidade em se conhecer de antemão a posição relativa entre a câmera e o braço, já que o sistema seria capaz de localizar a garra por conta própria. A única preocupação na disposição da mesa e o sensor 3D foi assegurar-se de que a maior parte do alcance dinâmico da garra estivesse no campo de visão do Kinect.

O manipulador utilizado apresentava uma superfície bastante escura e brilhosa, que refletiria grande parte da luz que o alcançasse. Isto se tornou um problema para o tipo de sensor utilizado, já que o Kinect não seria capaz de captar a maior parte da informação do braço. Assim, decidiu-se por cobrir a garra com papel, tornando-o mais claro e menos reluzente.

4.2 Arquitetura Geral do Projeto no Sistema ROS

O Sistema ROS (Robot Operating System), ou Sistema Operacional de Robôs, (Quigley et al., 2009) é um sistema de código aberto (licença BSD) que fornece um framework de desenvolvimento para aplicações voltadas à robótica.Não se trata de uma sistema operacional no sentido comum, mas fornece um camada estruturada de comunicação sobre o sistema operacional hospedeiro. Apresenta suporte tanto para linguagem C++ quanto para Python, sendo a primeira a adotada neste trabalho.

A unidade computacional básica no sistema ROS é denominada nó (node), que é um processo a que se atribui uma determinada tarefa. A comunicação entre nós se dá pela passagem de mensagens tipadas, podendo elas ser um dado inteiro, booleano, uma estrutura de dados com tipos distintos, um conjunto de pontos tridimensionais, etc.

A troca dessas mensagens é feita em um tópico (topic), onde nós podem disponibilizar dados, publicando no tópico (publisher node), ou podem ser assinantes daquele tópico e extrair dados dele (subscriber node). Na Fig. 10, um simples diagrama ilustra a ideia.

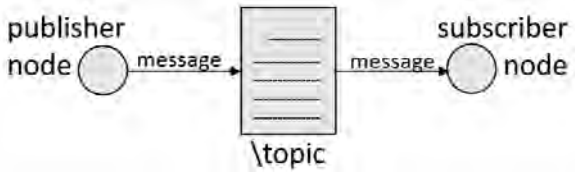


Figura 10 – Diagrama ilustrando a comunicação via mensagens entre nós no sistema ROS.

O sistema de visão computacional proposto foi concebido com cinco nós. Os dois primeiros nós servirão apenas na inicialização do sistema, sendo um deles para a geração da nuvem de pontos inicial correspondente à cena observada e o outro para o registro de um dado modelo na referida cena. Os demais nós estarão, por outro lado, continuamente interagindo e tratando dados: na sequência causal de processamento, um deles rastreará o manipulador robótico, outro se responsabilizará pela detecção do estado de sua garra e, por fim, o último nó proverá ao usuário interface de monitoramento em tempo real.

O diagrama da Figura 11 configura um esquemático geral dos nós implementandos e instanciados no sistema e de como ocorre a comunicação entre eles, seja por meio da escrita e leitura em arquivos, seja pela troca de mensagens entre publicantes e subscritos em um dado tópico.

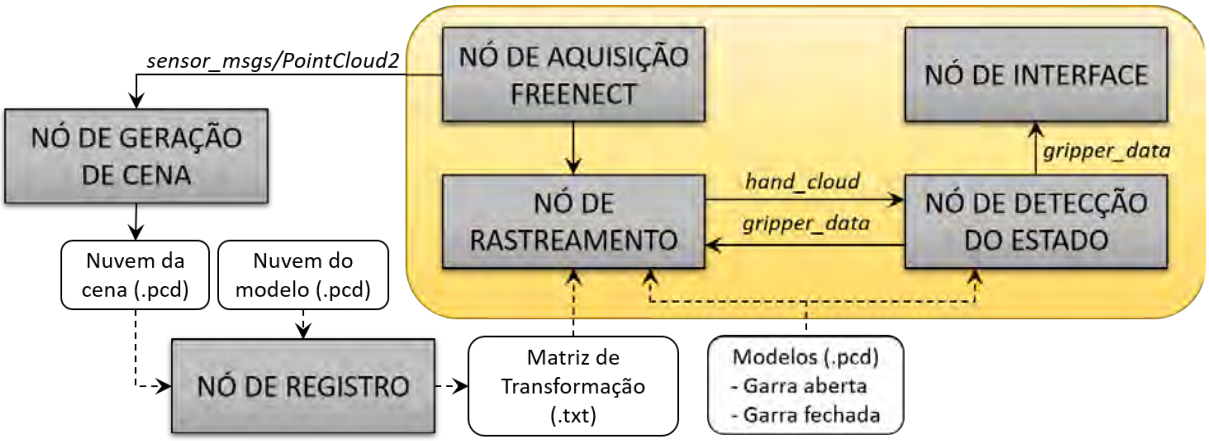


Figura 11 – Arquitetura geral do sitema no ROS.

No diagrama, os retângulos de cor cinza representam os nós, enquanto que os de cor branca e cantos arredondados representam os arquivos utilizados. Setas sólidas são utilizadas para designar tópicos, às quais se associa o tipo de mensagem neles trocada. Setas tracejadas, por sua vez, indicam simplesmente a leitura ou escrita em arquivos. Por fim, o painel de cor amarela envolve os nós que são executados continuamente no sistema, enquanto que os nós externos a ele são necessários tão somente para fins de inicialização.

4.3 Aquisição de dados - *Freenect*

O primeiro nó apresentado não foi desenvolvido neste projeto, estando, na verdade, disponível a partir da instalação dos drivers *libfreenect*, necessários para aquisição dos dados vindos do módulo sensor Kinect. O pacote freenect_launch do ROS fornece os arquivos de lançamento (launch files) necessários para a produção de nuvens de pontos a partir das leituras obtidas (Freenect). Após conectar o módulo Kinect por meio de uma porta USB, executa-se o arquivo de lançamento (launch file) do Freenect, a partir do seguinte comando:

```
roslaunch freenect_launch freenect.launch
```

Com isso, serão instanciados um conjunto de nós e de tópicos, onde se iniciará a publicação de toda informação sensorial vinda do Kinect. Embora os dados sejam disponibilzados (publicados) em variados formatos, fez-se uso apenas de nuvens de pontos

3D sem informação de cor, de modo que foi preciso subscrever-se apenas no tópico /camera/depth_registered/points.

O nó que publica neste tópico, designado por /camera/camera_nodelet_manager é referenciado aqui simplesmente como *nó Freenect*, já que foi o único nó necessário para a obtenção dos dados desejados. Enquanto o nó Freenect publica dados no tópico Freenect (/camera/depth_registered/points), dois outros nós estarão subscritos nele: o nó de geração de cena e o nó de rastreamento.

4.4 Geração de Cena

A atribuição do nó de geração de cena é relativamente simples. Estando subscrito no tópico Freenect, fará a leitura da nuvem de pontos disponibilizada no tópico, salvando-a então em um arquivo .pcd (*Point Cloud Data*). Em vista de sua simplicidade, este nó poderia ser facilmente integrado em outro, mas foi decidido defini-lo dessa forma, como uma maneira eficaz de gerar cenas, por exemplo, para testes.

Na Figura 12, é apresentado um *screenshot* retirado do arquivo .pcd de uma dada cena, onde percebe-se o manipulador Kinova Jaco, a mesa sobre a qual ele se encontra fixo e todo o contexto de fundo (background) da sala. Também é mostrado o sistema de coordenadas da câmera, no qual todos os pontos são descritos.

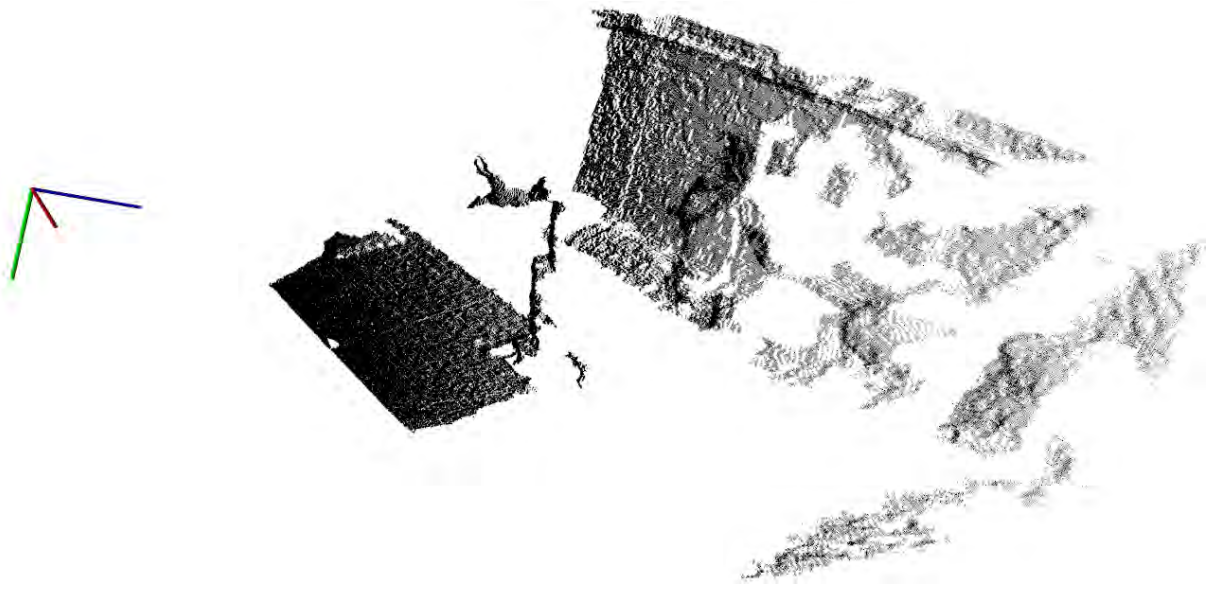


Figura 12 – Uma captura de tela da cena salva pelo nó de geração de cena.

4.5 Registro da nuvem de pontos

O nó de registro desempenha o primeiro passo fundamental no sistema proposto, que consiste em estimar a pose inicial da garra, para posterior utilização nos passos de rastreamento e detecção de estado. Os algoritmos e códigos utilizados desenvolvidos para a implementação deste nó foram adaptados dos exemplos disponíveis em PCL, 2016.

Foi imposto ao sistema uma condição de inicialização na qual a garra deve se encontrar completamente aberta. Dessa forma, a nuvem de pontos a ser registrada na cena seria aquela referente ao modelo de garra aberta, exposto no lado esquerdo da Figura 2.

A princípio, a posição inicial da garra poderia ser qualquer, contanto que estivesse dentro do campo de visão do sensor. Independentemente, o nó de registro seria capaz de localizá-la na cena. Entretanto, uma vez que o foco deste projeto não é propriamente o registro, ter que fazê-lo todas as vezes que o sistema fosse inicializado, supondo que diferentes posições iniciais fossem adotadas, seria bastante tedioso e demorado.

Por questões práticas, portanto, uma pose inicial foi escolhida para o manipulador, sendo, na verdade, uma leve variação da pose de inicialização programada no Jaco, de forma que se tornou fácil retorná-lo para a mesma posição de inicialização sempre que necessário. A Figura 13 apresenta a pose inicial escolhida.

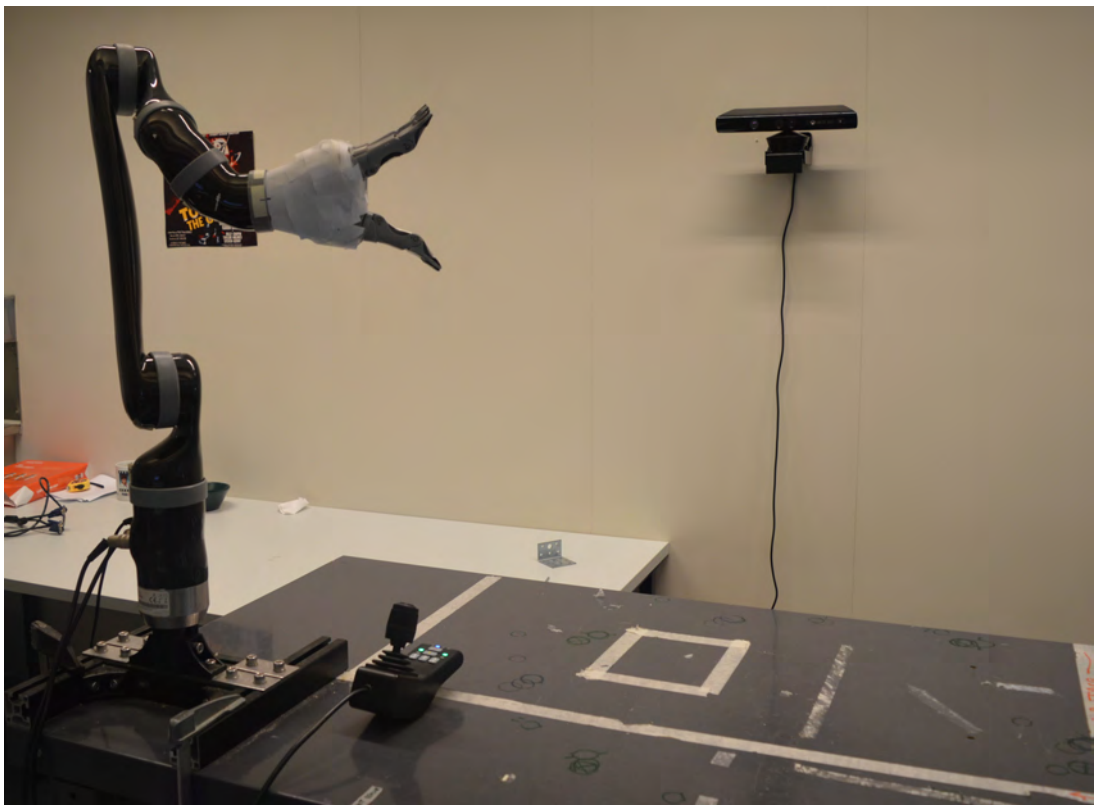


Figura 13 – Pose inicial definida para o braço robótico Jaco[2].

Conforme fundamentado anteriormente, o registro consiste em um problema de

otimização que, ao minimizar uma dada função de custo, encontra a matriz de transformação que deve ser aplicada sobre o *frame* (sistema de coordenadas) da nuvem de pontos do modelo de tal forma a alinhá-la com a nuvem de pontos correspondente à cena. Na Figura 14, é exposta a sobreposição das nuvens de cena e modelo. Devido à maneira que foi definido o frame do modelo (ver Figura 3), é de se esperar que a origem do frame do modelo coincida com a origem do frame da câmera.

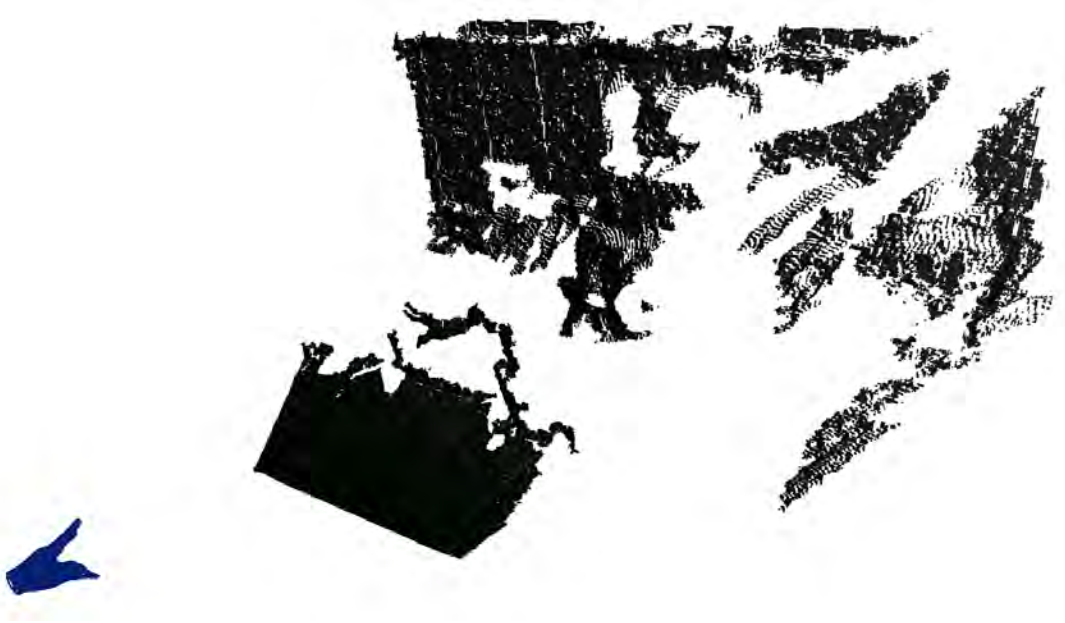


Figura 14 – Apresentação das nuvens de pontos referentes à cena e ao modelo de garra aberta antes do registro.

Antes de registrar o modelo na cena, alguns passos de pré-processamento são necessários, listados a seguir na ordem em que são executados. A Figura 15 mostra os resultados obtidos após o pré-processamento da cena apresentada na Figura 14.

- Extração da região de interesse: são aplicados filtros *pass-through* ao longo dos eixos x, y and z, removendo assim os pontos exteriores à região de interesse;
- Remoção de dados ruidosos;
- Aplicação de subamostragem (*downsampling*) sobre as nuvens para reduzir o número de pontos e, consequentemente, o esforço computacional;
- Segmentação e remoção de planos, utilizado para removar, por exemplo, a mesa da cena;
- Clusterização da cena remanescente, no intuito de focar no cluster referente à garra;

Uma vez feito o pré-processamento da nuvem de pontos da cena, bem como a subamostragem da nuvem do modelo, o algoritmo SAC-IA é então aplicado para obtenção

Figura 15 – Apresentação de alguns passos de pré-processamento: à esquerda, é apresentada a cena resultante após aplicação dos filtos *pass-through*, da remoção de ruído e da subamostagem; à direita, o resultado da posterior segmentação de plano e custerização, onde o cluster correspondente à garra está destacado em vermelho.

do alinhamento inicial. Mesmo com o pré-processamento, o resultado poderia, em algumas tentativas, se mostrar insatisfatório. Notou-se que estes casos correlacionavam com valores relativamente elevados da medida de alinhamento utilizada (*Euclidean Fitness Score*) e, a partir de experimentos, adotou-se um *score* máximo de $7 \cdot 10^{-4}$.

Assim, o algoritmo seria repetido algumas vezes até que resultasse em um score inferior ao limite imposto. Com isso, obteve-se um bom alinhamento inicial para o modelo, a ser refinado posteriormente. Um exemplo do resultado obtido pelo SAC-IA é apresentado na Figura 16.

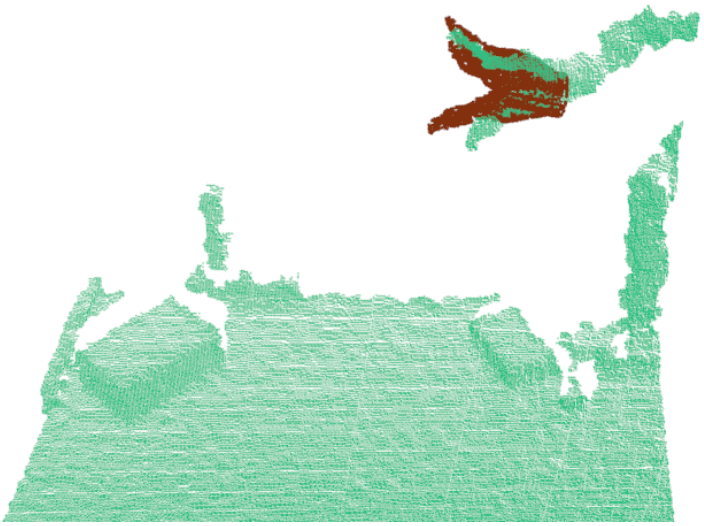


Figura 16 – Alinhamento Inicial para o modelo de garra aberta usando o algoritmo SAC-IA.

Em seguida, para aprimorar o alinhamento, o ICP foi aplicado, estimando a transformação que minimiza uma função de custo pelo erro médio quadrático. Em termos

práticos, a saída do algoritmo é uma matriz de trasformação final (rotação e translação), conforme descrito no diagrama da Figura 11 e exemplificado na Equação 4.1 a seguir. O nó de registro salva a matriz final em um arquivo de texto para que outros nós possam acessá-lo.

$$T = \begin{bmatrix} 0.19 & 0.20 & -0.96 & 0.16 \\ 0.43 & -0.90 & -0.10 & -0.34 \\ -0.88 & -0.39 & -0.26 & 1.19 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

O alinhamento final resultante da aplicação do ICP, mais especificamente da aplicação da transformação dada pela Equação 4.1 é apresentado nas Figuras 17 e 18, sendo esta exposta no visualizador 3D do ROS RViz.

Figura 17 – Alinhamento final para o modelo de garra aberta pelo algoritmo ICP.

Percebe-se qualitativamente que o registro obteve resultados satisfatórios. O alinhamento final pode eventualmente não ser tão preciso quanto o exemplo apresentado. No entanto, para os propósitos do sistema, não é mandatório que o registro seja perfeito, pois o nó de rastreamento ainda é capaz de compensar pequenos ajusto no alinhamento, como será discutido na seção seguinte.

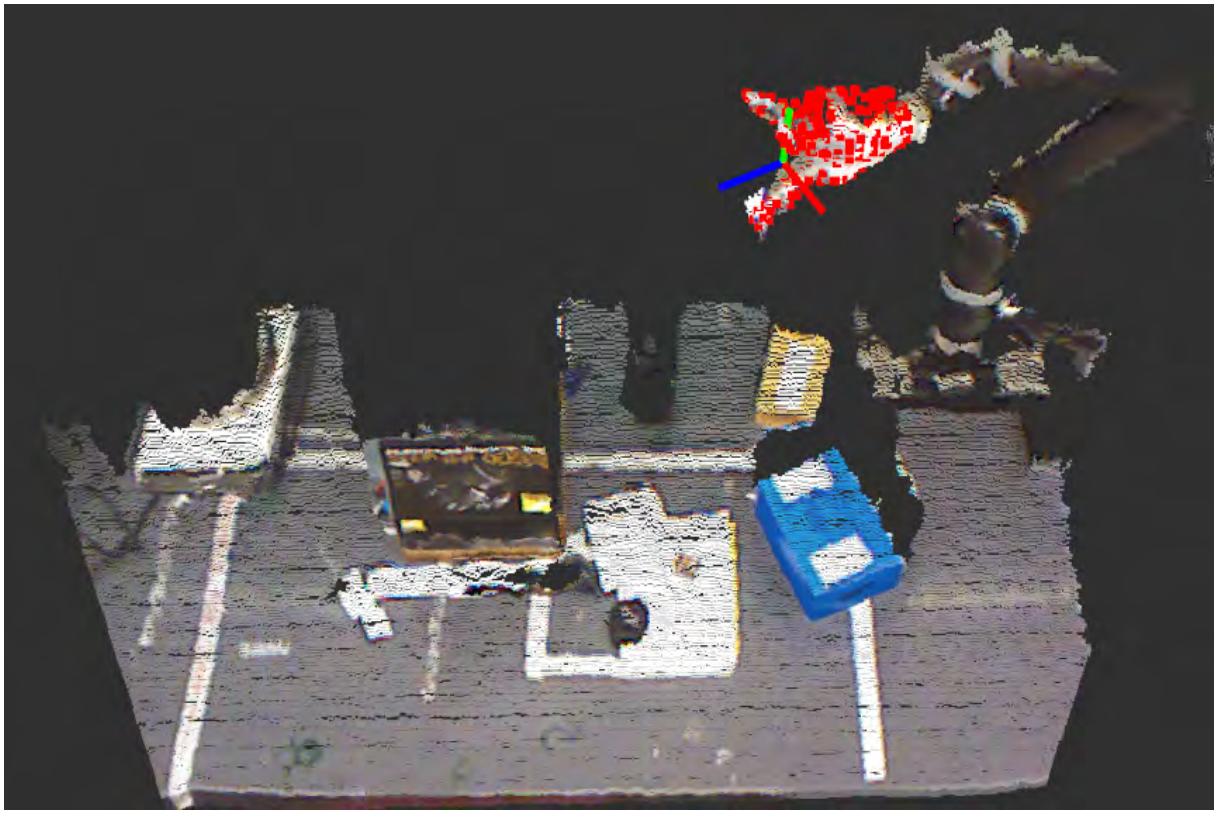


Figura 18 – Registro final para o modelo de garra aberta visualizado no RViz.

4.6 Rastreamento do manipulador e do efetuador final

Enquanto os nós previamente descritos são executados apenas na inicialização do sistema, o nó de rastreamento, assim como os demais, os são continuamente. Este nó busca manter o rastreio da matriz de transformação entre os frames da garra e da câmera ou, em outros termos, entre sua pose atual e inicial, esta resultante do nó de registro.

O nó de rastreamento faz uso de três entradas: um arquivo de texto contendo a transformação inicial oriunda do registro do modelo, as mensagens publicadas no tópico Freenect, relacionadas à cena observada, e os modelos em nuvem da garra aberta e fechada. Ainda, este nó também recebe retorno do nó de detecção do estado da garra.

A implementação do nó de rastreamento iniciou-se como uma adaptação do exemplo tutorial para rastreio de objeto diponibilizado em (PCL, 2016). Uma vez subscrito no tópico Freenect, o rastreador pré-processaria a cena lida de maneira similar à do nó de registro: extraindo a região de interesse, filtrando ruído, aplicando subamostragem, removendo planos e clusterizando a nuvem remanescente. O nó adota como nuvem de referência para rastreio o modelo de garra aberta, por convenção. Então, a partir do Filtro de Partícula, estimaria, para cada nova cena obtida, a nova matriz de transformação para a garra em relação ao frame da câmera.

A Figura 19 ilustra o funcionamento do filtro de partículas. Para uma apresentação mais didática do conceito, o algoritmo foi inicializado (lado esquero da figura) com alta covariância, de modo que as partículas estivessem bastante espalhadas ao redor da garra. Uma vez observada a primeira cena, já se percebe, após a primeira iteração (centro da figura), uma disposição mais concentrada das partículas em torno do frame da garra. No exemplo, na 25^a iteração (lado direito), o alinhamento está completo. Vale ressaltar que, embora o registro não tenha sido perfeito, foi bom o suficiente par que o filtro de partículas pudesse, em algumas iterações, finalizá-lo.

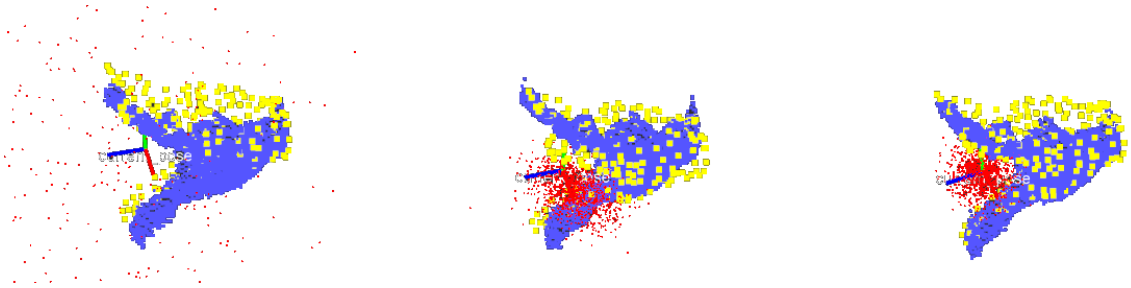


Figura 19 – Capturas de tela ilustrando o funcionamento do Filtro de Partícula. Inicialização (esquerda), primeira iteração (centro) e vigésima-quinta (direita).

Foi de interesse representar o sistema de coordenadas da pose atual (PA) em relação ao da pose inicial (PI). Como o filtro de partículas retornaria como saída a transformação relativa ao frame da câmera (C), a seguinte trasformação foi necessária:

$$^{PI}T_{PA} = {^{C}T_{PI}}^{-1} \cdot \ ^{C}T_{PA} \tag{4.2}$$

Dessa forma, o frame referente à pose atual seria filho do frame da pose inicial que, por sua vez, seria filho do frame da câmera. Esta relação é esclarecida na Figura 20, que também exemplifica o funcionamento do nó.

Em termos de saída, o nó de rastreamento publicará duas nuvens: a nuvem de partículas (pontos em vermelho na Figura 19), utilizado apenas para visualização no RViz, e a nuvem referente à visão parcial da garra (*hand_cloud*, pontos em azul), publicadas respectivamente nos tópicos *particles_cloud* e *tracked_scene_cloud*.

A nuvem *hand_cloud* será então explorada pelo nó de detecção do estado da garra. Se for detectada uma mudança no estado da garra, isto é, se a garra fechar ou abrir, esta será notificada ao nó de rastreamento, que modificará a nuvem de referência para rastreio, de modo que não busque rastrear o modelo errado. Na Figura 21, duas amostras de mensagens *hand_cloud* são apresentadas, seus pontos definidos em relação ao frame da pose atual.

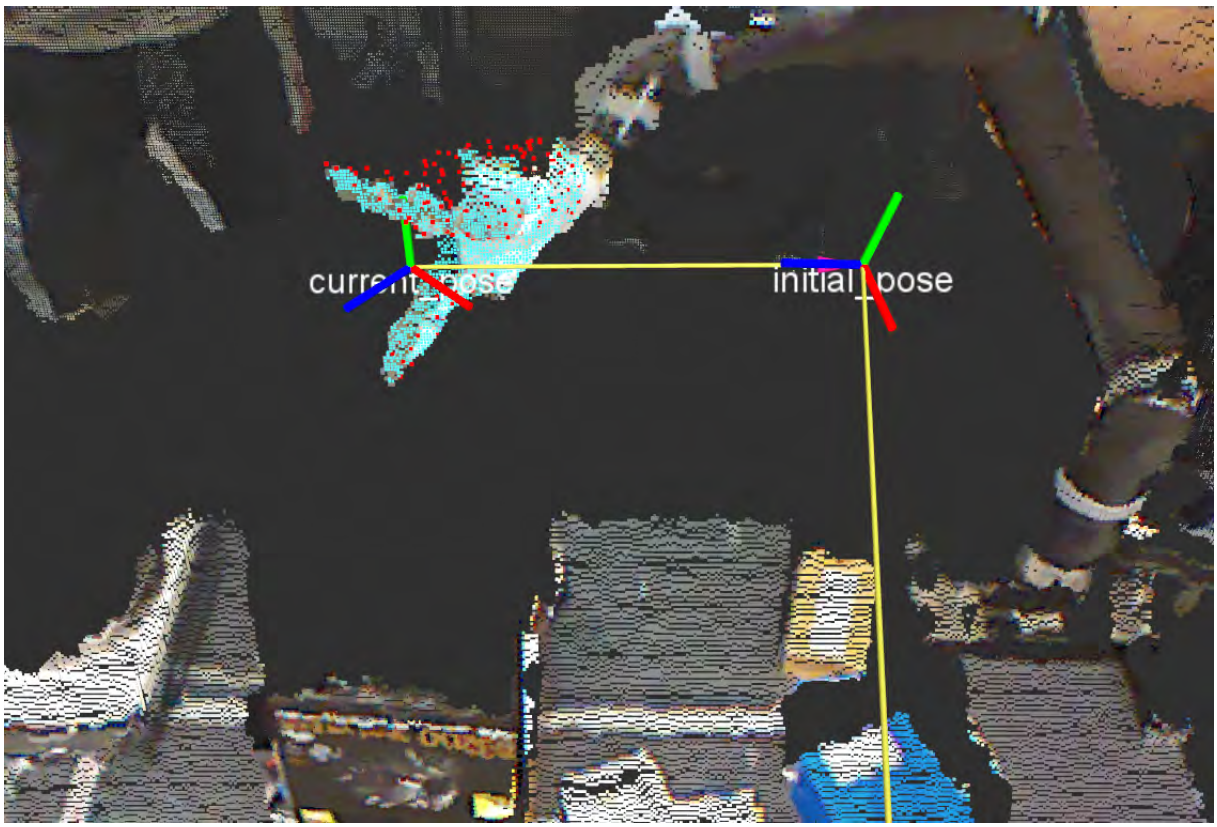


Figura 20 – Captura de tela no RViz do nó de rastreamento em execução, ressaltando a hierarquia entre os sistemas de coordenadas.

Figura 21 – Duas amostras de nuvem de pontos mostrando uma visão parcial da garra na cena, referenciadas no frame da pose atual.

4.7 Detecção do estado da garra

Como brevemente mencionado na seção anterior, o nó de detecção do estado da garra (NDEG) faz uso da nuvem de pontos fornecida pelo nó de rastreamento. De fato, o tópico *tracked_scene_cloud* é o único no qual o NDEG se subscreve.

O fato da nuvem da garra ser descrita no sistema de coordenadas da pose atual isola completamente o NDEG em relação ao resto do sistema, já que este não precisará lidar com transformações entre frames nem tomar conhecimento do contexto da cena. Portanto, enquanto o nó de rastreamento for capaz de executar sua tarefa, o NDEG também o será.

À primeira vista, a tarefa atribuída a este nó é simples: dada uma nuvem de pontos relativa à garra do manipulador, deve-se decidir se esta representa uma garra aberta ou fechada. A partir do conceito de similaridade entre nuvens de pontos definido na Equação 3.8, limiares de decisão foram escolhidos para as similaridades com cada um dos possíveis modelos.

A Figura 22 apresenta os casos de classificação mais extremos em termos da abertura da garra, isto é, completamente aberta e completamente fechada. Vale ressaltar que, como o punho da garra não provê informação a respeito do seu estado, a nuvem a ser classificada pode ser reduzida à porção que inclui dos dedos à sua palma. Na Tabela 1, são expostos os valores de similaridades para os quatro casos extremos, o que sugere uma faixa razoavelmente larga para seleção de limiares.

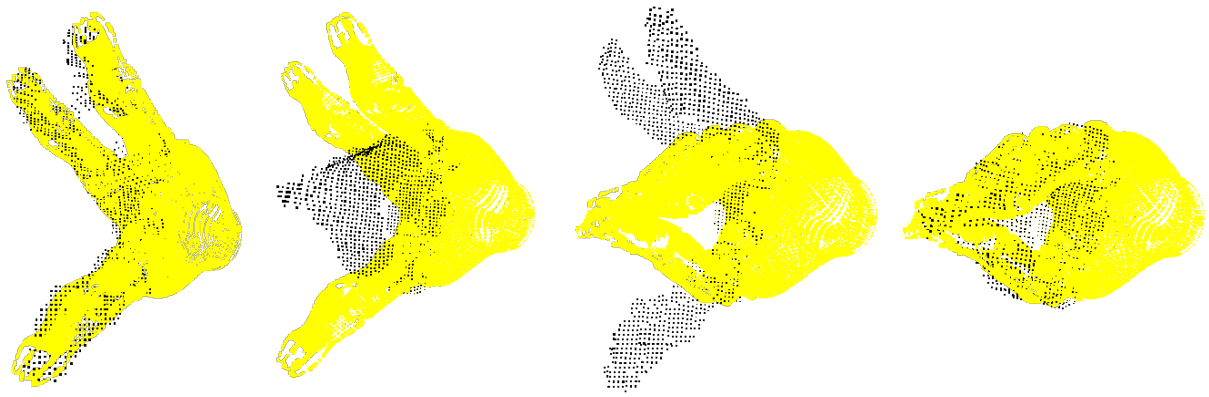


Figura 22 – Os quatro casos extremos para classificação do estado da garra como aberta ou fechada. Os pontos em amarelo correspondem às nuvens dos modelos e os pontos pretos às nuvens de cena.

Tabela 1 – Valores de similaridades obtidos para os casos extremos de classificação.

	Cena com garra aberta	Cena com garra fechada
Modelo garra aberta	0.98	0.34
Modelo garra fechada	0.31	0.95

O nó NDEG publica no tópico denominado `gripper_data` um tipo de mensagem que corresponde ao conjunto de variáveis listado a seguir:

- `m1_sim`, `m2_sim`: similaridades com os modelos de garra aberta e fechada, respectivamente;
- `thres_1`, `thres_2`: valores limiares para decisão sobre cada modelo;
- `thres_obj`: um valor limiar para decisão sobre eventual detecção de objeto;
- `model_n`: um identificador sobre o estado atual (1:aberto, 2:fechado);
- `cluster_n`: um contador do número de clusters presentes na nuvem;
- `obj_detect`: uma varíavel booleana para indicar a presença de um objeto.

A Figura 23 contém um exemplo do tipo de mensagem publicada pelo NDEG, em um cenário onde a garra se encontra completamente aberta e não há objetos ao seu alcance.

```
$ rostopic echo /gripper_state_node/gripper_data
m1_sim: 0.981783317354
m2_sim: 0.341323106424
thres_1: 0.95
thres_2: 0.95
thres_obj: 0.9
model_n: 1
cluster_n: 2
obj_detect: False
---
```

Figura 23 – Ecoando os dados publicados pelo nó detector do estado da garra (NDEG).

Para melhor apresentar a dinâmica dos valores de similaridade, a Figura 24 expõe os gráficos de seus comportamentos, a partir da ferramenta `rqt_plot` de visualização 2D do ROS, enquanto a garra do manipulador era continuamente aberta e fechada.

Embora denominado detector de estado, decidiu-se que este nó simplesmente calcularia e disponibilizaria toda a informação necessária para a determinação do estado da garra, mas não necessariamente faria a classificação em si. Na verdade, quaisquer nós teriam acesso às informações exemplificadas na Figura 23, interpretando-as conforme lhes conviesse. O nó de rastreamento, por exemplo, estará subscrito neste tópico e, observando as medidas de similaridade, poderá alternar sua nuvem de referência de rastreio entre os modelos disponíveis. O nó de interface com o usuário, por sua vez, extrairá sua própria lógica dos dados, no intuito de prover as informações de forma clara.

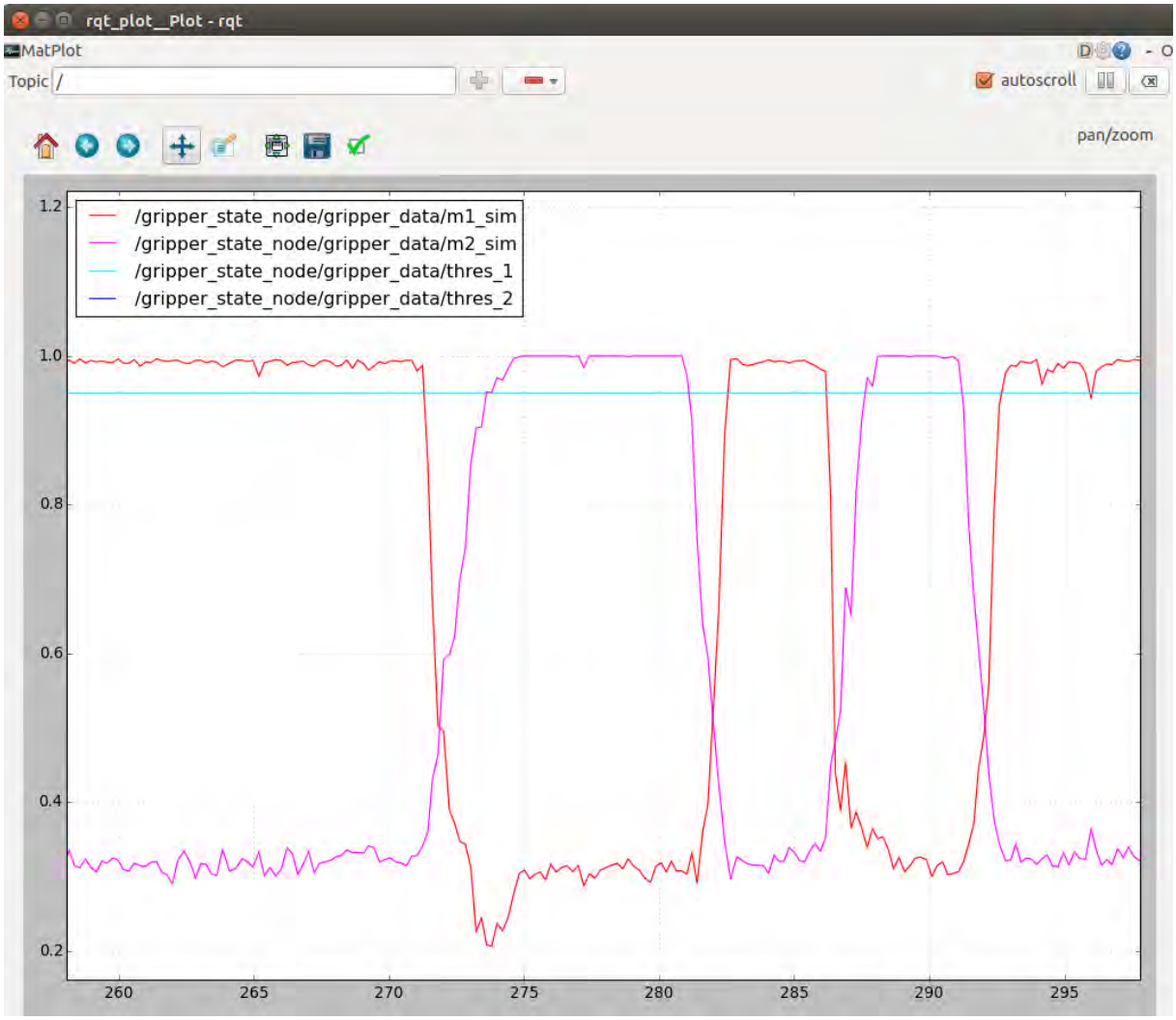


Figura 24 – Comportamento das medidas de similaridade com os modelos disponíveis ao longo do contínuo abrir e fechar da garra.

4.7.1 Detecção de objetos a partir de modelos

Fazer uso de modelos para classificação pode ser bastante conveniente para fins comparativos caso se disponha de modelos suficientes para contemplar o espaço de possibilidades a serem classificadas. Nesta abordagem orientada a modelos, a definição de similaridade pode ser utilizada para separar pontos que pertencem a um dado modelo (*inliers*) dos que não pertencem (*outliers*).

Em um ambiente controlado, isto se tornou pertinente para a detecção de objetos. Quaisquer pontos na nuvem disponibilizada pelo nó de rastreamento poderia facilmente ser avaliado quanto à pertença a qualquer um dos modelos. Partindo desse princípio, a informação de não pertencer a nenhum dos modelos pode ser interpretada, dadas certas condições, como a presença de um objeto ou obstáculo.

Esta é a razão pela qual o NDEG também publica informação referente à detecção

de objetos: a partir da clusterização que executa sobre a nuvem, o nó está constantemente avaliando os clusters obtidos e a qual modelo eles pertencem, gerando saídas de acordo. Esta informação será então utilizada pelo nó de interface com o usuário para posterior detecção de objeto e percepção de sua apreensão.

4.8 Interface com usuário

O monitoramento contínuo do estado do manipulador, fundamental no sistema proposto, motivou o desenvolvimento de uma interface de fácil leitura para o usuário, capaz de resumir informações gerais relativas ao estado da garra (*gripper*) bem como à avaliação sobre a apreensão de objetos.

Neste sentido, o Qt Application Framework se mostrou bastante útil, um framework de plataforma cruzada usado principalmente para desenvolvimento de aplicações. A principal razão para adotar o Qt foi sua capacidade de se integrar ao sistema ROS, em C++ (Qt, 2016).

Assim, ao nó de interface com usuário foi designada a construção da interface Qt, estando ele subscrito apenas no tópico `gripper_data`, que provê toda a informação a ele necessária.

O principal esforço de processamento neste nó é o de interpretar a nuvem recebida, extraindo dela informação lógica. Para isso, uma máquina de estados foi estruturada no intuito de cobrir uma quantidade razoável de estados possíveis da garra. Esta máquina de estados, apresentada na Figura 25, é composta por duas cadeias principais de eventos: o gripper do manipulador pode alternar entre o estado fechado (`Closed Gripper`) e o estado totalmente aberto (`Fully-opened Gripper`), passando pelo estado parcialmente aberto (`Opened Gripper`), e vice-versa; ainda, pode passar pela sequência de estados referentes à apreensão e soltura de um objeto (`Grasping` → `Object Grasped` → `Releasing`).

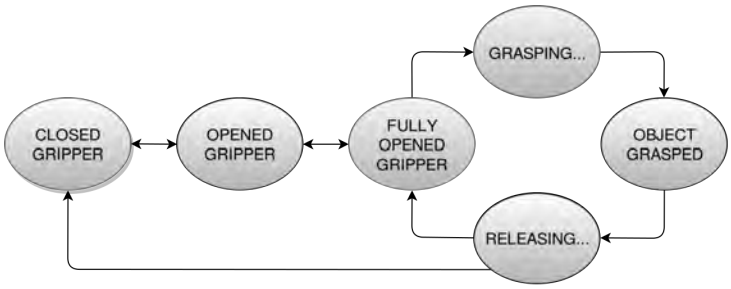


Figura 25 – Uma máquina de estados idealizada para a garra do manipulador.

Supondo que a garra esteja completamente aberta (`Fully-opened Gripper`), se a similaridade com o modelo aberto cair abaixo de um determinado limiar, a conclusão é de que a garra começou a se fechar. Então, se nenhum objeto foi detectado, a máquina entra

no estado de garra aberta (`Opened Gripper`), onde permanece até que a similaridade com algum dos modelos ultrapasse seu limiar, podendo voltar ao estado de início ou passar para o estado de garra fechada (`Closed Gripper`). Por outro lado, se algum objecto fosse detectado, a máquina ficaria no estado de preensão (`Grasping`) até que o número de clusters se tornasse unitário, indicando que o objeto estaria firmemente agarrado (`Grasped`). Então, quando o número de clusters modificasse novamente, a garra estaria começando a soltar o objeto (`Releasing`), até retornar ao estado inicial.

A janela de interface Qt é composta dos seguintes paineis:

- Painel de Conexão (Connection Panel), para estabelecer a conexão do nó de interface no sistema ROS;
- Painel de Registros (Logging Panel), para notificar sobre quaisquer atualizações em relação ao estado da garra;
- Painel de Rastreamento (Tracking Panel), apresentando as similaridades e qual modelo está sendo rasteado no momento;
- Painel de Estado (Gripper State Panel), exibindo a dinâmica da máquina de estado da garra.

Nas figuras a seguir são apresentados alguns exemplos da interface desenvolvida, cada um exibindo uma das cadeias de eventos mencionadas: enquanto na Figura 26 é apresentado um cenário onde o sistema acabou de ser inicializado, como exemplo da primeira cadeia de eventos, a Figura 27 exemplifica a cadeia de eventos referente à apreensão de objetos.

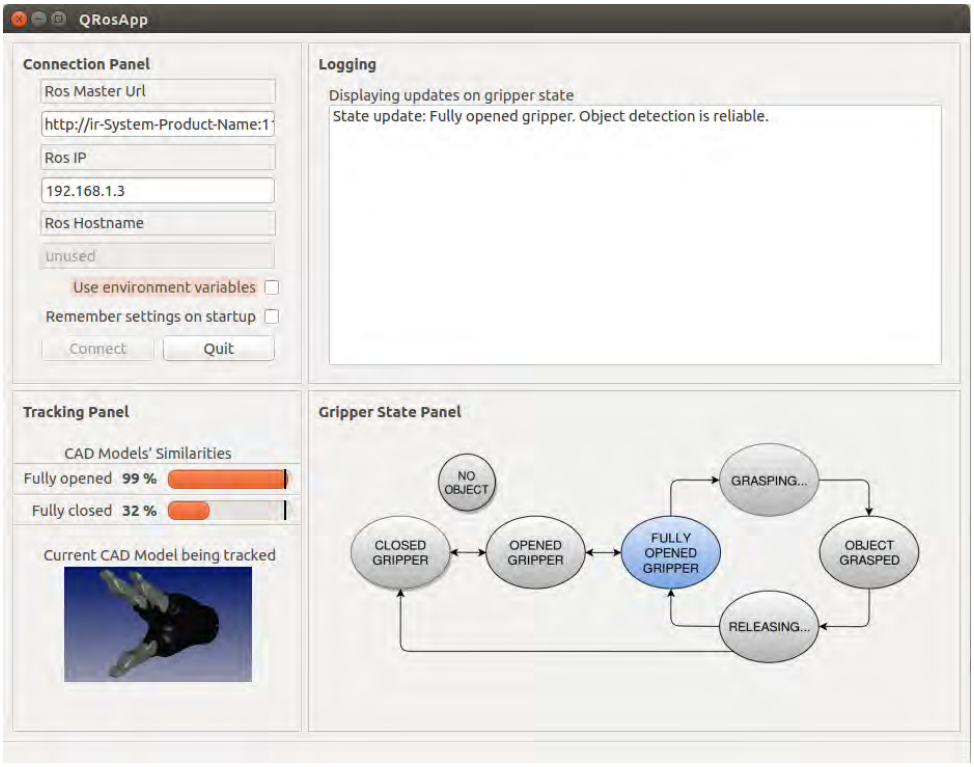


Figura 26 – Visualização da interface desenvolvida em um cenário com a garra completamente aberta e ausência de objetos.

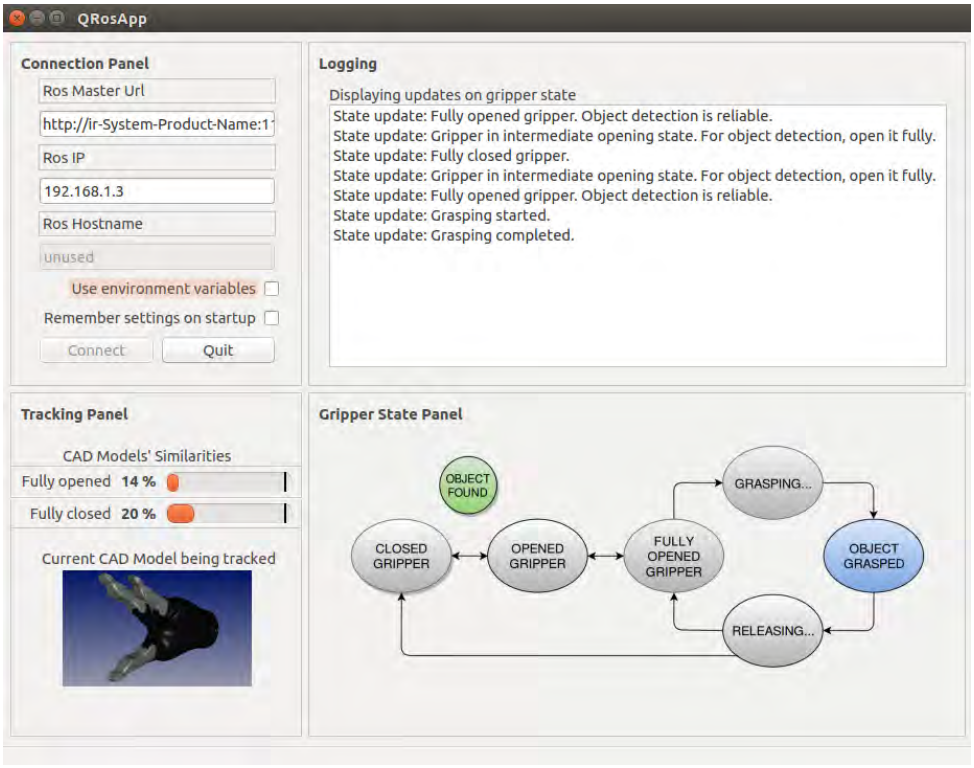


Figura 27 – Visualização da interface desenvolvida em um cenário onde um objeto encontra-se firmemente agarrado pelo manipulador.

5 Resultados

Neste capítulo, é apresentada uma discussão de caráter majoritariamente qualitativo a respeito dos resultados experimentais obtidos durante os testes do sistema desenvolvido. Na verdade, detalhes sobre os resultados já foram inevitavelmente discutidos de antemão ao longo do capítulo anterior, pois foi entendido que apresentar o funcionamento dos nós sem tratar das entradas e saídas envolvidas, dos tipos de mensagens trocadas e dos resultados esperados acabaria por acarretar em uma descrição bastante incompleta do sistema.

Inicialmente, algumas capturas de tela foram selecionadas para prover o máximo de informação possível sobre os resultados mais relevantes do sistema. Em seguida, serão discutidas as limitações que foram encontradas ao longo dos testes realizados. Por fim, como suplemento às imagens presentes neste documento, vídeos demonstrativos serão referenciados.

Um exemplo do rastreamento é apresentado na Figura 28, expondo um bom alinhamento entre a garra rastreada e a nuvem de pontos do modelo, além de reforçar a relação hierárquica entre os sistemas de coordenadas. No momento em que foi realizada esta captura de tela, a garra encontrava-se completamente aberta e imóvel.

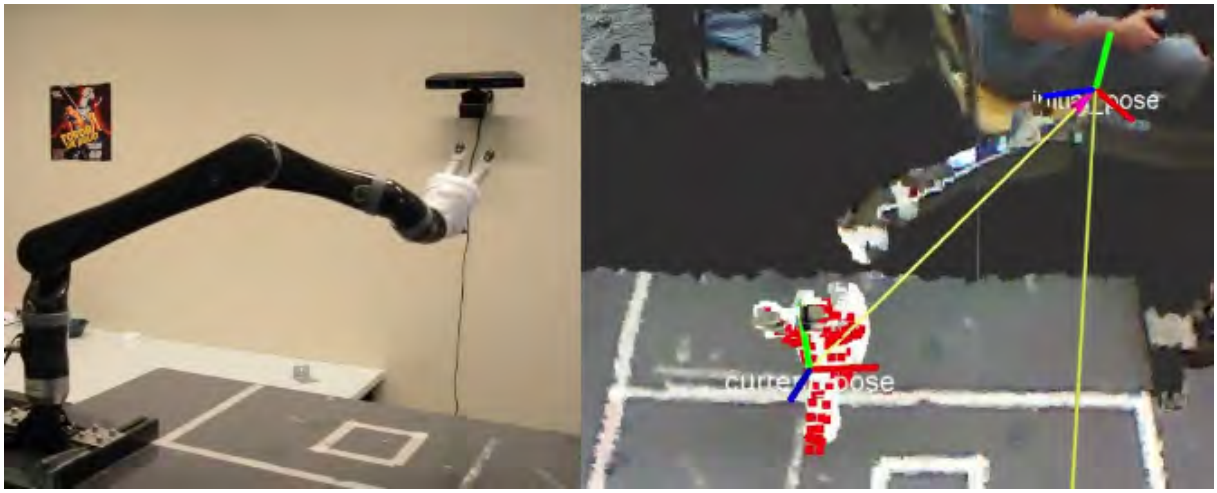


Figura 28 – Uma captura de tela exemplificando o nó de rastreamento em execução.

A Figura 29, por sua vez, busca passar a noção do movimento, além de exemplificar o funcionamento do detector de estado da garra. No visualizador RViz (parte superior esquerda), nota-se um pequeno atraso entre o movimento da garra e o modelo que a segue.

Ainda, percebe-se que o modelo sendo rastreado é o da garra fechada, em concordância com o estado do manipulador. Os gráficos de similaridade confirmam o estado atual

da garra e reportam os estados passados mais recentes. Por fim, a máquina de estados apresentada pelo nó de interface com usuário encontra-se também de acordo com o estado de garra fechada.

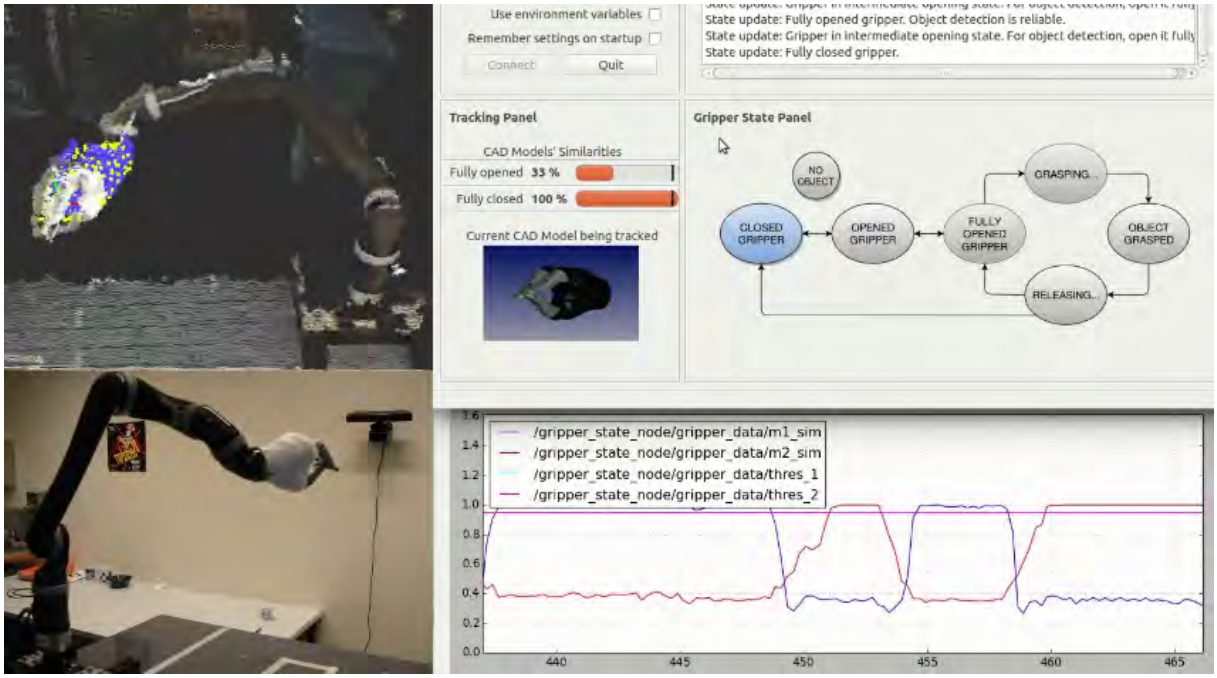


Figura 29 – Um exemplo de rastreamento da garra fechada, junto aos gráficos de similaridades e à interface gráfica desenvolvida.

Uma das vantagens consequentes da abordagem baseada em modelos foi a possibilidade de tirar proveito deles para detectar objetos. A Figura 30 exemplifica de forma clara o funcionamento da detecção. A partir da suposição de que um objeto só pode ser detectado se a garra estiver completamente aberta, a sua presença acarretará na detecção de um cluster que não pertence, no sentido da similaridade, à nuvem do modelo. Este cluster é, portanto, interpretado como um objeto.

Se um objeto for detectado, a cor do círculo no Painel de Estado mudará para verde e, se a garra começar a fechar enquanto o objeto estiver presente, a máquina de estados entrará no ciclo de preensão (grasping). É importante salientar que o a presença de um objeto não deve afetar as similaridades com os modelos: um objeto pode vir a obstruir a visão parcial da garra, mas isso apenas reduziria o número de pontos disponíveis na nuvem; a similaridade apresentaria valores similares, calculada então com um número menor de pontos.

Além disso, se, por exemplo, um dos dedos do manipulador ocluir outro de forma que este não seja clusterizado com a garra, a abordagem adotada ainda consideraria o dedo ocluso como parte da garra. Assim, a utilização de modelos também provê certa robustez a auto-oclusões do manipulador.

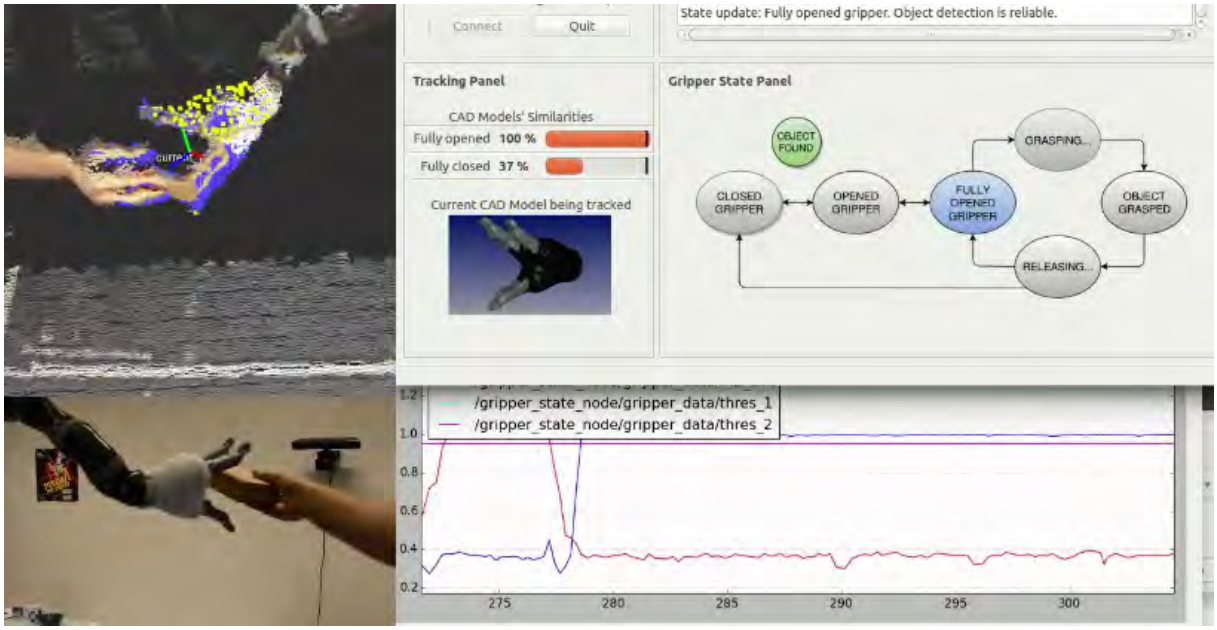


Figura 30 – Detecção de objeto a ser apreendido a partir da abordagem baseada em modelos.

Ainda no contexto da detecção de objetos, a Figura 31 apresenta a preensão de uma pequena caixa, sequência da figura anterior. O ponto a se destacar é que, uma vez agarrado o objeto, ambas as similaridades se reduziram bastante. De fato, perde-se a informação do modelo, já que a garra está, de certa forma, em um novo estado, incompatível com os modelos disponíveis.

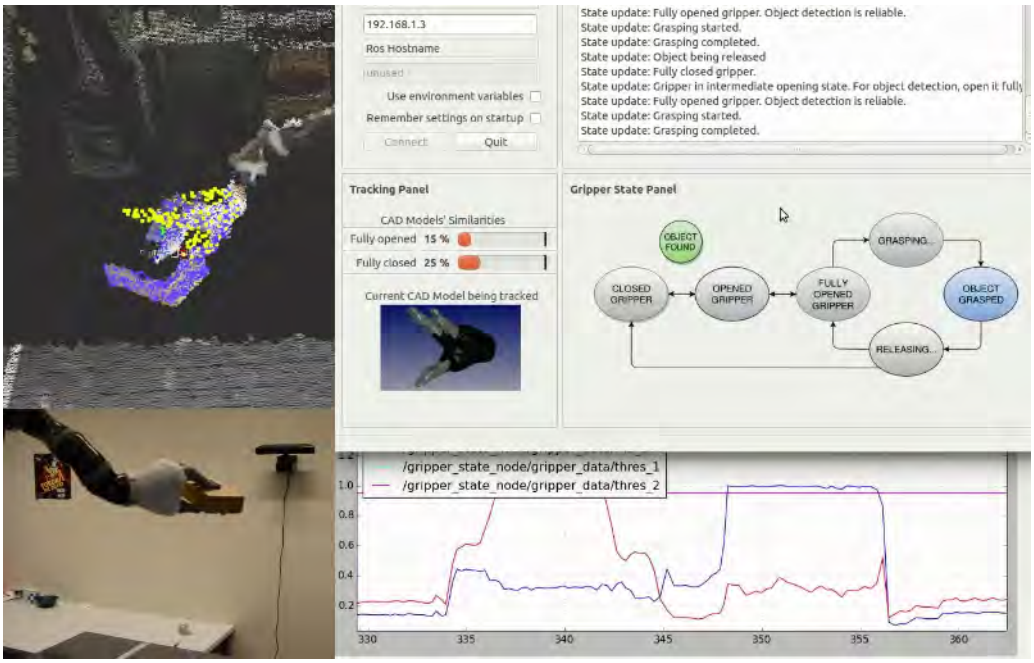


Figura 31 – Efeito da apreensão do objeto sobre os valores das similaridades.

5.1 Limitações e Potenciais Aprimoramentos

No desenvolvimento do sistema, foram percebidas duas limitações principais. A primeira delas diz respeito ao funcionamento do nó de rastreamento face a rotação da garra. A resposta do rastreio a translações em geral se mostrou rápida e precisa, mas em se tratando de rotações, a sensibilidade e velocidade do rastreamento eram prejudicadas.

É possível que uma parametrização mais minuciosa do filtro de partículas, por exemplo, amenize este problema ou, ainda, a proposição de uma maneira para aumentar a resposta do rastreador a rotações. A Figura 32 exemplifica esta limitação, em uma situação na qual a garra do manipulador foi submetida a uma rápida rotação em torno do eixo z, que levou à perda do rastro.

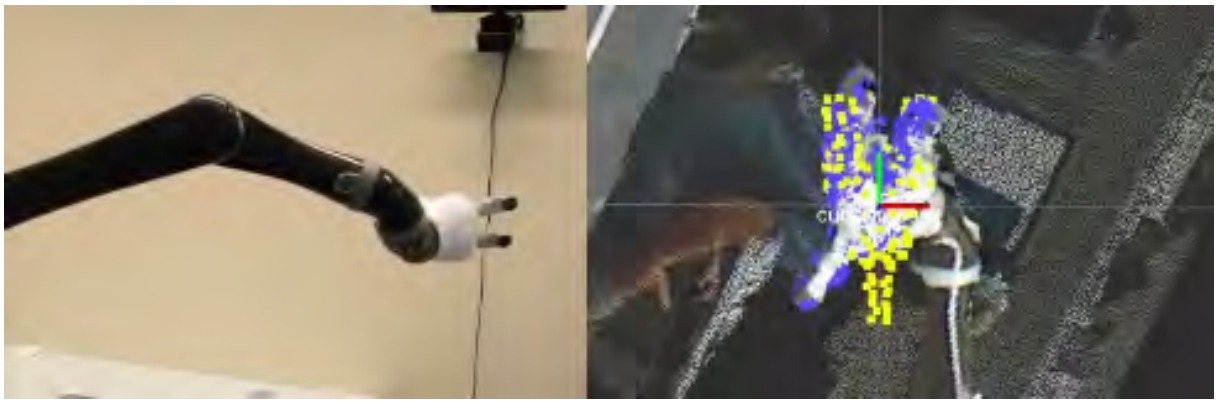


Figura 32 – Demonstrando a limitação de baixa sensibilidade do algoritmo de rastreamento a rotações da garra.

A segunda limitação observada se relaciona diretamente ao número de modelos disponíveis e, mais especificamente, com a ausência de um modelo que leve em conta a presença de objeto. Na verdade, a preensão de objetos não fez parte do escopo inicial deste trabalho, configurando assim apenas uma aplicação prática do sistema. De todo modo, rastrear a garra enquanto esta carrega um objeto, subsidiado apenas pelos modelos disponíveis, mostrou-se inviável. Uma possível solução para este problema seria a geração de um novo modelo a partir da visão parcial do objeto detectado, de modo que o nó de rastreamento dispusesse de uma referência mais condizente com a situação em questão. A Figura 33 apresenta um exemplo onde o rastro da garra foi perdido, após esta ter sido movida transportando o objeto agarrado.

Em geral, o sistema é passível de expansão pela exploração de um conjunto maior de modelos CAD, o que confereria ao nó de rastreamento uma informação mais coerente com o estado real do manipulador. Ainda, um possível aprimoramento seria a capacidade de gerar modelos 3D sob demanda, como, por exemplo, a partir da combinação da garra e de um objeto como uma única entidade a ser rastreada.

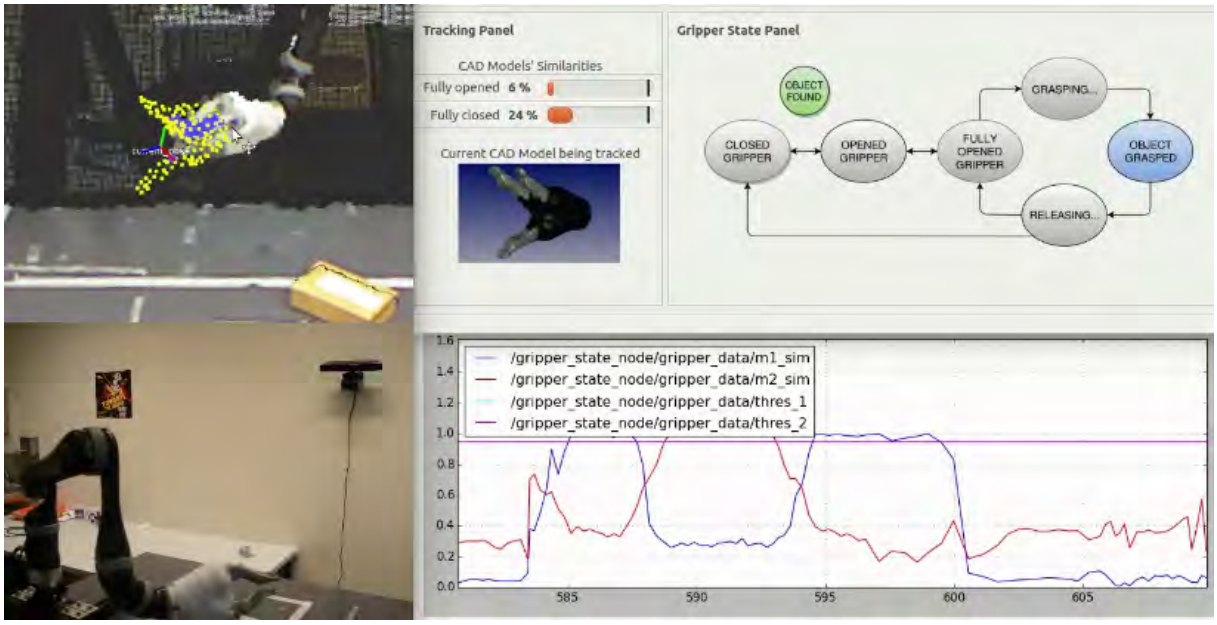


Figura 33 – Demonstrando a perda do rastro da garra quando do seu deslocamento carregando um objeto.

5.2 Vídeos Demonstrativos

Como o sistema proposto é bastante dinâmico, facilmente avaliado por meio de observação, notou-se que a melhor maneira de ampliar a compreensão do leitor sobre o seu funcionamento seria pela apresentação do sistema em execução. Portanto, um conjunto de vídeos demonstratios foram disponibilizados online, onde as principais características e limitações do sistema podem ser percebidos. Os vídeos cobrem os seguintes aspectos:

- Demonstração completa das etapas envolvidas no registro de um modelo;
- O rastreamento da garra completamente aberta, apresentando, inclusive, a baixa sensibilidade a rápidas rotações;
- Demonstração do funcionamento do filtro de partícula;
- Uma demonstração dos nós de rastreamento, de detecção do estado da garra e de interace com usuário em execução simultânea, trocando mensagens no sistema ROS, de acordo com o movimento descrito pelo braço robótico na cena;
- Demonstração da detecção e preensã de um objeto;
- Demonstração da perda de rastro quando do deslocamento da garra enquanto segurando um objeto.

Os vídeos demonstrativos, assimo como todo o projeto desenvolvido, podem acessados pelo GitHub (Araujo, 2016).

6 Conclusão

Em termos gerais, a proposta deste Trabalho de Conclusão de Curso foi a de desenvolver um sitema de visão computacional 3D baseado em modelos capaz de rastrear a garra de um manipulador robótico e identificar seu estado. De forma secundária, a identificação de um objeto ao alcance e deu sua preensão tornou-se uma tarefa de interesse para o projeto. Partindo disso, conclui-se que o sistema proposto atendeu aos objetivos definidos: uma vez respeitadas suas suposições e limitações, é capaz de realizar adequadamente o rastreio da garra, a detecção de seu estado e da preensão de um objeto.

O desenvolvimento deste trabalho foi beneficiado por um conjunto de ferramentas de fácil integração. Construir o sistema na plataforma ROS foi, além de muito conveniente, importante para potenciais aprimoramentos futuros e para o reuso do que foi desenvolvido em outros projetos.

Uma das principais ideias para trabalhos posteriores seria a inserção deste projeto em um cenário colaborativo de múltiplos robôs, no intuito de que estes sejam capazes de realizar tarefas em conjunto, subsidiados pelo sistema de visão proposto. Por exemplo, montar um ambiente com dois braços robóticos e avaliar o quão bem eles estimariam o estado um do outro. É também de interesse investigar o quão bem o sistema proposto generalizaria para manipuladores similares, sem a necessidade recorrer ao seu próprio modelo.

Apesar da limitação de tempo, devido à duração do intercâmbio ao longo do qual este trabalho foi desenvolvido, foi possível ainda alcançar bons resultados nos diversos aspectos do projeto. Devido a sua natureza e às circunstâncias do projeto, a apresentação de resultados neste relatório não foi trivial, uma vez que se limitou majoritariamente a discussões qualitativas e exposição de imagens. Portanto, embora tenha-se buscado selecionar cautelosamente perspectivas que melhor descrevessem os resultados obtidos, a melhor maneira de observá-los seria provavelmente através dos vídeos demonstrativos disponibilizados (Araujo, 2016) como suporte a este trabalho.

Referências

ARAUJO, A. C. de. *Source code and Demonstration Videos supporting this Bachelor Thesis*. 2016. Accessed: 2016-07-05. Disponível em: <https://github.com/arthurarj/bachelor_thesis.git>. Citado 2 vezes nas páginas 51 e 52.

ARKIN, R. C. *Behavior-Based Robotics*. 1. ed. Cambridge, MA, USA: MIT Press, 1998. Citado na página 13.

ARULAMPALAM, M. S. et al. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, Ieee, v. 50, n. 2, p. 174–188, 2002. Citado 2 vezes nas páginas 24 e 25.

BESL, P. J.; MCKAY, N. D. *Method for registration of 3-D shapes*. 1992. 586-606 p. Disponível em: <http://dx.doi.org/10.1117/12.57955>. Citado na página 23.

BRAY, M. et al. 3d hand tracking by rapid stochastic gradient descent using a skinning model. In: CITESEER. *In 1st European Conference on Visual Media Production*. CVMP, 2004. Citado na página 13.

CAUSO, A. et al. *Predictive Tracking in Vision-based Hand Pose Estimation Using Unscented Kalman Filter and Multi-viewpoint Cameras*. INTECH Open Access Publisher, 2010. ISBN 9789533070513. Disponível em: <https://books.google.com.br/books?id=_NuHoAEACAAJ>. Citado 2 vezes nas páginas 17 e 27.

CHETVERIKOV, D.; STEPANOV, D.; KRSEK, P. Robust euclidean alignment of 3d point sets: the trimmed iterative closest point algorithm. *Image and Vision Computing*, Elsevier, v. 23, n. 3, p. 299–309, 2005. Citado na página 22.

EROL, A. et al. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, Elsevier, v. 108, n. 1, p. 52–73, 2007. Citado 2 vezes nas páginas 24 e 27.

ESTEVEZ, E.; MARCOS, M. Model-based validation of industrial control systems. *IEEE Transactions on Industrial Informatics*, IEEE, v. 8, n. 2, p. 302–310, 2012. Citado na página 17.

FREENECT. *Freenect Launch ROS Package*. Accessed: 2016-07-02. Disponível em: <http://wiki.ros.org/freenect_launch>. Citado na página 32.

HASHIMOTO, K. A review on vision-based control of robot manipulators. *Advanced Robotics*, Taylor & Francis, v. 17, n. 10, p. 969–991, 2003. Citado na página 13.

HEAP, T.; HOGG, D. Towards 3d hand tracking using a deformable model. In: *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*. Killington, Vermont: IEEE, 1996. p. 140–145. Citado na página 13.

KANEKO, S.; KONDO, T.; MIYAMOTO, A. Robust matching of 3d contours using iterative closest point algorithm improved by m-estimation. *Pattern Recognition*, Elsevier, v. 36, n. 9, p. 2041–2047, 2003. Citado na página 22.

KERDVIBULVECH, C.; SAITO, H. Model-based hand tracking by chamfer distance and adaptive color learning using particle filter. *EURASIP Journal on Image and Video Processing*, Springer International Publishing, v. 2009, n. 1, p. 1, 2009. Citado na página 27.

KINOVA. *Kinova Robotics - Page for CAD model downloads*. 2016. Accessed: 2016-06-23. Disponível em: <http://www.kinovarobotics.com/service-robotics/products/grippers>. Citado na página 17.

KRAGIC, D.; CHRISTENSEN, H. I. Model based techniques for robotic servoing and grasping. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Switzerland: IEEE, 2002. v. 1, p. 299–304 vol.1. Citado na página 13.

LITOMISKY, K. Consumer rgb-d cameras and their applications. *Rapport technique, University of California*, v. 20, 2012. Citado na página 16.

LUCK, J.; LITTLE, C.; HOFF, W. Registration of range data using a hybrid simulated annealing and iterative closest point algorithm. In: IEEE. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. California, 2000. v. 4, p. 3739–3744. Citado na página 22.

MATEO, C. et al. Rgbd human-hand recognition for the interaction with robot-hand. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Portugal: IEEE, 2012. Citado na página 16.

OIKONOMIDIS, I.; KYRIAZIS, N.; ARGYROS, A. A. Efficient model-based 3d tracking of hand articulations using kinect. In: *BmVC*. United Kingdom: IEEE. v. 1, n. 2, p. 3. Citado na página 13.

PCL. *Point Cloud Library Documentation Tutorials*. 2016. Accessed: 2016-06-23. Disponível em: <http://www.pointclouds.org/documentation/tutorials>. Citado 4 vezes nas páginas 17, 27, 34 e 38.

QT. *A simple Qt template under ROS*. 2016. Accessed: 2016-06-23. Disponível em: <http://wiki.ros.org/qt_ros>. Citado na página 44.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: *ICRA Workshop on Open Source Software*. Japan: IEEE, 2009. Citado na página 31.

RUSU, R. B.; BLODOW, N.; BEETZ, M. Fast point feature histograms (fpfh) for 3d registration. In: IEEE. *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. Japan: IEEE, 2009. p. 3212–3217. Citado 3 vezes nas páginas 20, 21 e 22.

RUSU, R. B.; COUSINS, S. 3d is here: Point cloud library (pcl). In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. China: IEEE, 2011. p. 1–4. ISSN 1050-4729. Citado na página 16.

RUSU, R. B. et al. Learning informative point classes for the acquisition of object model maps. In: IEEE. *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*. Vietnam, 2008. p. 643–650. Citado na página 19.

SCHNABEL, R.; WAHL, R.; KLEIN, R. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, Blackwell Publishing Ltd, v. 26, n. 2, p. 214–226, 2007. ISSN 1467-8659. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2007.01016.x>. Citado na página 16.

STENGER, B. et al. Model-based hand tracking using a hierarchical bayesian filter. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 28, n. 9, p. 1372–1384, 2006. Citado na página 13.

TAM, G. K. et al. Registration of 3d point clouds and meshes: a survey from rigid to nonrigid. *IEEE transactions on visualization and computer graphics*, IEEE, v. 19, n. 7, p. 1199–1217, 2013. Citado na página 13.

WANG, R. Y.; POPOVIĆ, J. Real-time hand-tracking with a color glove. In: *ACM transactions on graphics (TOG)*. USA: [s.n.], 2009. v. 28, n. 3, p. 63. Citado na página 13.

WANG, W.; JOHNSTON, B.; WILLIAMS, M.-A. Social networking for robots to share knowledge, skills and know-how. In: SPRINGER. *International Conference on Social Robotics*. China, 2012. p. 418–427. Citado na página 13.