UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Felipe Victor de Sá Oliveira

Programação por demonstração de um manipulador robótico utilizando uma abordagem baseada em grafos

Recife

2016

Felipe Victor de Sá Oliveira

Programação por demonstração de um manipulador robótico utilizando uma abordagem baseada em grafos

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Glauco Gonçalves

Coorientador: Victor Medeiros

Recife

2016

Aos meus pais, com carinho e dedicação.

Agradecimentos

Agradeço primeiramente a Deus, por ter me fornecido a fé necessária para nunca desistir desta jornada tão difícil, com ele eu pude acreditar que tudo era possível.

Agradeço aos meus pais João Carlos e Aparecida pelo apoio total desde do primeiro dia de aula até o dia de hoje, que apesar de todas as dificuldades enfrentadas ao longo de todos esses anos, nunca deixaram de lutar para que eu conseguisse chegar até o final.

Agradeço ao meu segundo pai, meu irmão, Francisco Vanderlei, que nunca mediu esforços para me fornecer tudo que fosse necessário para que eu cursasse uma graduação, sem nunca precisar se preocupar com nada além dos estudos.

Agradeço ao meu irmão e divisor do mesmo ventre, meu gêmeo Fábio Oliveira, que sempre me serviu como exemplo de determinação e sabedoria, graças a ele eu puder acreditar mais em mim e na minha capacidade de aprender tudo aquilo que eu estivesse disposto.

Agradeço a minha melhor amiga, companheira, namorada, noiva e futura esposa, Maria Luisa, que foi fundamental em todo esse processo, me acalmando nos momentos de nervosismo e frustrações, me encorajando em todos os desafios, que acompanhou todo meu processo de amadurecimento como pessoa e como cidadão, que nunca e em nenhum momento, desacreditou da minha capacidade de ir cada vez mais longe.

Agradeço a todos meus amigos pelos momentos de distrações, alegrias e aconselhamentos durante todo esse tempo, em especial à Vitor Hazin, Fabrício Luna e Raphael Uchôa.

Agradeço aos meus orientadores, Professores Glauco Gonçalves e Victor Medeiros, pela paciência e confiança que depositaram em mim ao longo de todo desenvolvimento deste trabalho, sempre me motivando a fazer ainda melhor e graças a eles isso tudo foi possível.

Agradeço as alunos do Juá Labs por toda a troca de conhecimento realizada durante todo o período, em especial aos alunos Daniel, João, Filipe e Nichene que me ajudaram diretamente em fases fundamentais do projeto.

Por fim e não menos importante, agradeço aos meus professores da UFRPE do campus de Serra Talhada, que foram fundamentais para construir minha base de conhecimento ao longo do curso. Muito obrigado a todos!

“Milagres acontecem quando a gente vai à luta!”
(Sérgio Vaz)

Resumo

A cada dia que passa os robôs estão ficando mais inteligentes e autônomos. Fazer com que um robô aprenda uma nova tarefa sem ser necessário ter acesso à seu código fonte desperta bastante interesse, pois traz benefícios como a redução de custos com manutenção de programas e aumento da produtividade na indústria.

Sistemas baseados em Programação por Demonstração (*Programming by Demonstration* - PbD) têm como objetivo fazer com que um robô seja capaz de aprender e reproduzir a atividade demonstrada, sem a reprogramação direta de seu código fonte, ou seja, em PbD o robô deve se autoprogramar para se adequar à tarefa a ele imposta. Algumas formas de PbD empregam câmeras, sensores e software complexo para que o robô aprenda a partir da observação da tarefa, sem necessidade de um programador. Diversamente, uma abordagem mais barata, faz uso de controles complexos (*teaching pendants*) para programar o robô enquanto a tarefa é executada. Esta programação exige que o programador seja especialista não apenas na tarefa a ser executada mas tenha habilidade no uso do controle. Além disso, a introdução de imprecisões no momento da programação exige um maior tempo com o robô parado e sem produzir.

Neste trabalho, apresenta-se uma estratégia PbD baseada em teoria dos grafos para programação de robôs sem recursos de sensoriamento. Esta estratégia tem duplo objetivo: a) permitir que usuários sem conhecimento de programação possam programar robôs por meio de um joystick simples; e b) potencialmente reduzir o tempo de programação permitindo que o usuário forneça demonstrações de uma mesma tarefa para mitigar imprecisões introduzidas pelo usuário ou pelo ambiente durante as demonstrações.

Implementada em um manipulador robótico, 20 usuários diferentes experimentaram a estratégia proposta na execução de uma mesma tarefa, porém com quantidades diferentes de demonstrações. Como resultado, constata-se que o aumento do número de demonstrações reduziu em até 76% o total de erros na execução da tarefa com um tempo máximo de programação de 15 minutos. Desta forma, é possível observar que a técnica proposta tem potencial para simplificar o treinamento de um robô pelo uso de um mero joystick, sem necessidade de conhecimentos de programação baseada em texto, além de potencialmente reduzir o tempo com o robô parado.

Palavras-chave: Programação por Demonstração, Robótica, Teoria dos Grafos.

Abstract

Every to day the robots are becoming more intelligent and autonomous. Making a robot learn a new task without having to update its source code arouses great interest as it brings benefits such as reduced costs with maintenance programs and increase productivity in the industry.

Systems based on Programming by Demonstration PBD aims at making a robot able to learn and perform the demonstrated activity without the direct reprogramming of its source code, ie PBD robot should autoprogramar to suit the task he imposed. Some forms of PBD employ cameras, sensors and complex software for the robot to learn from observation of the task, without a programmer. By contrast, a cheaper approach makes use of complex teaching pendants controls to program the robot while the task is performed. This program requires that the programmer is an expert not only on the task at hand but has the ability to control the use. Furthermore, introducing inaccuracies in the time of programming requires a longer with the robot stopped and without producing.

In this paper, we present a PBD strategy based on graph theory for robots without sensing capabilities programming. This strategy has a double purpose: a) allow users without programming knowledge to program robots through a simple joystick; and b) potentially reduce programming time by allowing the user to provide demonstrations of the same task to mitigate inaccuracies introduced by the user or the environment during the demonstrations.

Implemented in a robotic manipulator, 20 different users experienced the proposed strategy in the execution of the same task, but with different amounts of demonstrations. As a result, it is found that increasing the number of statements reduced by up to 76% of the total errors in the task execution with a maximum time of 15 minutes schedule. Thus, it can be seen that the proposed technique has the potential to simplify training a robot by using a simple joystick, without the need for text-based programming skills, and potentially reduce the time to stop the robot.

Keywords: Programing by demonstration, Robotics, Graph Theory.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

PbD	Programming by Demonstration (Programação por Demonstração)
CPU	Unidade Central de Processamento
GUI	*Graphical User Interface* (Interface Gráfica do Usuário)
3D	Três Dimensões
USB	*Universal Serial Bus* (Porta Universal)
SVM	*Support Vector Machine* (Máquina de vetores de suporte)

Sumário

1 Introdução

Este Capítulo apresenta o problema abordado e os objetivos deste trabalho. Primeiramente, procura-se contextualizar como as máquinas revolucionaram a indústria como um todo, bem como mostrar a importância dos robôs quando inseridos em linhas de produção, além de apresentar uma visão geral sobre a Programação por Demonstração (*Programming by Demonstration* - PbD), que é o foco deste trabalho. Neste capítulo também serão apresentados a justificativa e motivação para realização do trabalho, assim como a organização do documento.

1.1 A importância da robótica

Desde a revolução industrial ocorrida na Europa em meados do século XVIII, atividades repetitivas e cansativas nas linhas de montagem que eram desempenhadas basicamente por seres humanos e suas pesadas ferramentas de trabalho, começaram a ser realizadas por máquinas cada vez mais complexas. Os produtos deixaram de ser manufaturados e passaram a ser maquino faturados, o que permitiu a produção em massa, aumentando a oferta de produtos no mercado e reduzindo seus preços (CAVALCANTE; SILVA, 2011). Também foi possível reduzir os altos custos com capital humano além do ganho significativo no tempo e na produtividade. A Revolução Industrial teve grande relevância para a sociedade atual e principalmente para o surgimento da revolução tecnológica vivida até os dias atuais. (CAVALCANTE; SILVA, 2011).

Com o avanço tecnológico ao longo dos anos, as fábricas se tornaram cada vez mais automatizadas. Os robôs começaram a ser inseridos nas linhas de montagem e braços robóticos que realizavam tarefas limitadas foram evoluindo e se tornando mais versáteis e precisos, executando múltiplas tarefas, inclusive. A Fabricante de Automóveis Tesla, por exemplo, utiliza em sua fábrica em Fremont, Califórnia, robôs como os ilustrados na Figura 1 que podem fazer até quatro tarefas: solda, rebites, colagem e instalação de um componente (MARKOFF, 2012). Esta versatilidade dos robôs revolucionou a forma de produção proporcionando ganhos significativos para os proprietários de fábricas ao redor do mundo. Segundo os dados observados por (MARKOFF, 2012): “esses equipamentos tendem a gerar grandes lucros a longo prazo, já que uma máquina com custo médio de 250 mil dólares substituindo dois empregados com salário de 50 mil ao ano deve economizar 3,5 milhões ao fim de seus 15 anos de vida útil”.

Figura 1 – Robôs da Fabricante de Automóveis Tesla em Fremont, Califórnia.

Fonte: Notícia do site Gizmodo/Austália[1].

1.2 Programação por demonstração

Um robô é basicamente um conjunto de equipamentos integrados, composto por hardware e software, que necessita ser programado para que consiga realizar uma atividade útil para os seres humanos. Mesmo para executar a mais básica operação, um robô precisa ter um software em execução. Este sistema tem como função monitorar os sensores do robô e acionar os seus motores fazendo com que ele se mova dentro dos limites dos seus graus de liberdade.

Para (BIGGS; MACDONALD, 2003), a área relacionada especificamente à programação de robôs pode ser dividida em *Automatic programming* (Programação automática) e *Manual Programming* (Programação Manual). Na primeira, não se tem acesso direto ao código fonte que programa o robô. (BIGGS; MACDONALD, 2003) destacam que ”esse tipo de sistema oferece pouco ou nenhum controle direto sobre o código do programa que o robô irá executar [...], o código do robô é gerado a partir de informação introduzida no sistema por uma variedade de formas indiretas”. Geralmente o robô permanece em execução enquanto é programado, oferecendo então uma programação *online*. Na programação manual, diferentemente, o usuário/programador precisa criar o programa que o robô deve interpretar e, consequentemente, executá-lo no robô para que adquira um comportamento. O robô não precisa estar presente enquanto é realizada a programação. No capítulo 2 serão apresentados mais detalhes sobre cada categoria.

A forma mais comum de Programação automática chama-se *Programming by Demonstration*. Esta abordagem permite que pessoas sem conhecimento formal em programação possam efetivamente programar um robô para que realize uma atividade útil em um curto intervalo de tempo e de forma intuitiva. PbD pode permitir que usuá-

rios finais ensinem, por meio de demostrações, as tarefas que o robô deve executar (FORBES et al., 2014). Toda a complexidade das numerosas linhas de código necessárias para o treinamento do robô é completamente abstraída para um plano real e observável, ou seja, onde o usuário consegue visualizar o robô fisicamente realizando os movimentos. A necessidade de PbD será cada vez mais percebida no futuro, conforme afirma (EKVALL, 2005): “a próxima geração de robôs será colocada em nossas casas e [...] a variedade de tarefas dos robôs nos ajudará significativamente e não será possível pré-programar robôs para todas essas tarefas”.

Sistemas baseados em Programação por Demonstração, utilizam diversos métodos como *interface* entre o usuário/programador e o robô para o qual se pretende transmitir um comportamento. (TEIXEIRA, 2009) destaca os dispositivos e estratégias que se pode usar em PbD, por exemplo, a fala, controles com botões, desenhos, gestos entre outros.

Aplicações de PbD avançadas são fortemente dependentes da integração de sensores internos e externos, como sensores de força/torque e visão (WAHL; THOMAS, 2002). Esses sensores são responsáveis por ser os “olhos” do robô e, a partir deles, os robôs conseguem “enxergar” tanto a tarefa que está sendo demonstrada quanto o ambiente em que está inserido. Porém em um cenário onde tais recursos sensoriais complexos não são possíveis, pode-se fazer uso de controles complexos denominados *teach-pendants* (como os que são ilustrados na Figura 2). Por estes controles, um usuário especialista demonstra a tarefa posicionando o robô e programando comandos simples como andar em linha, descrever um arco, ligar um atuador etc, que correspondem aos movimentos a serem executados pelo robô para realizar a tarefa (BIGGS; MACDONALD, 2003). Assim, o robô armazena em memória estes comandos e os executa para reproduzir a tarefa.

1.3 Justificativa e motivação

Os manipuladores robóticos são recursos fundamentais nas fábricas por todo o mundo e novos investimentos são feitos em pesquisas buscando melhorar ainda mais essa tecnologia. A complexidade do software usado para comandar essas máquinas e a demanda crescente por robôs flexíveis e reprogramáveis têm aumentado a necessidade por sistemas de programação por demonstração. (EKVALL, 2005). Sem a utilização de um sistema baseado em PbD, para que máquinas realizem as tarefas atribuídas a elas, faz-se necessária a contratação de profissionais especialistas em programação de computadores e microcontroladores e a necessidade de uma forte integração entre estes profissionais e a equipe de produção que conhece em detalhes as tarefas que o robô deve executar. Estas exigências levam ao aumento de custos

Figura 2 – Evolução dos *teach-pendants*

Fonte: (BOUCHARD, 2011) .

administrativos e de pessoal.

PbD simplifica esta tarefa de reprogramação do robô, permitindo que os profissionais especialistas nas atividades de produção possam diretamente programar estas máquinas (TEIXEIRA, 2009). Sistemas mais complexos de PbD utilizam numerosos recursos de hardware, como sensores e câmeras, para comunicação entre o usuário/programador e o robô. Sem esses recursos o robô fica impossibilitado de observar quais tarefas o usuário deseja transmitir a ele, bem como as características do ambiente em que está inserido.

Os *teach-pendants* permitem que os usuários programem os robôs, fornecendo as coordenadas e comandos específicos para realização dos movimentos necessários para execução da tarefa. São exemplos de comandos: descreva um movimento em linha desta coordenada a esta outra, descreva um movimento em arco passando por estas coordenadas etc. O sistema armazena esses comandos e, quando solicitado, reproduz os movimentos. Caso movimentos indesejados sejam transmitidos, o usuário tem a possibilidade de apagar os comandos e coordenadas e em seguida fornecer outras, ainda no momento da programação.

O PbD baseado em *teach-pendants* permite o uso de robôs mais simples, isto é, que não precisam de sensores, câmeras e software mais complexo. Por outro lado, exige um profissional que seja especialista tanto no uso do controle para programação do robô quanto na tarefa a ser executada. Além disso, a presença de imprecisões na execução da tarefa que passem desapercebidas pelo programador no momento da programação só serão detectadas no momento dos testes, o que exigirá um maior

tempo com o robô afastado da linha de produção, para reprogramação.

Neste trabalho, apresenta-se uma abordagem que utiliza técnicas da teoria dos grafos para realizar PbD de robôs. Esta abordagem pretende, ao mesmo tempo, reduzir o tempo de programação total, removendo a necessidade de reprogramações, e permitir que usuários sem conhecimento de programação possam programar o robô por meio de controles simples. A ideia básica desta abordagem é permitir que o usuário/programador, utilizando um joystick, possa oferecer diversas demonstrações da mesma tarefa e que o robô aprenda os movimentos demonstrados por seu usuário mesmo quando os movimentos são repetidos com imprecisão. De posse do conjunto de demonstrações, o sistema PbD extrai deste conjunto uma representação que reflete a melhor maneira de reproduzir a tarefa pretendida, mitigando imprecisões tanto das demonstrações fornecidas pelo usuário quanto do próprio robô.

1.4 Objetivos

O objetivo principal deste trabalho é desenvolver uma abordagem para realizar PbD para a programação de robôs. Utilizando técnicas de grafos, a abordagem proposta recebe um determinado número de demonstrações diretamente do usuário por meio de um joystick, e permite inferir a tarefa demonstrada mitigando imprecisões, sem que seja necessário ter acesso ao código fonte do programa do robô. Como objetivos específicos pretende-se:

- Definir um modelo de base de conhecimento para o robô para armazenamento dos movimentos do robô;
- Descrever, em detalhes, um algoritmo para inferência da tarefa a partir do modelo de memória; e
- Implementar e avaliar os modelos propostos.

1.5 Organização do trabalho

Além deste Capítulo introdutório, o presente trabalho possui mais 4 capítulos e está organizado da seguinte forma: o Capítulo 2 descreve todo o conceito relacionado a Programming by Demonstration (PbD) e também mostra trabalhos relacionados na área; o Capítulo 3 descreve toda a implementação realizada para criação do sistema PbD com base em grafos; o Capítulo 4 apresenta todos os experimentos e seus respectivos resultados. Por fim, Capítulo 5 expõe as conclusões obtidas neste trabalho.

2 Referencial Teórico

Este Capitulo tem como objetivo contextualizar as categorias e métodos para realizar a programação de um robô (Seção 2.1) e descrever com mais clareza a Programação por Demonstração (Seção 2.2), mostrando desde a concepção dos primeiros sistemas baseados em PbD, até serem inseridos no campo de estudo da robótica. Por fim, serão apresentados elementos da Teoria dos Grafos (Seção 2.3) necessários para a compreensão do modelo de base de conhecimento e dos algoritmos propostos neste trabalho.

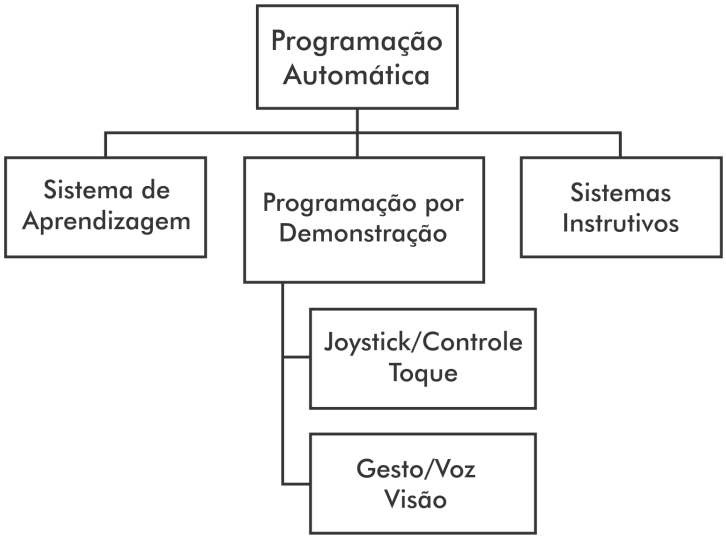
2.1 Programação de robôs

Um robô é um tipo de sistema embarcado, que é geralmente implementado como uma máquina eletromecânica controlada por programas de computador com o objetivo de realizar uma atividade útil aos seres humanos. Mesmo para executar a mais básica operação, um robô precisa ter software em execução para monitorar os seus sensores e acionar os seus atuadores fazendo com que execute as tarefas para as quais está programado, dentro dos limites dos seus graus de liberdade.

A área relacionada especificamente à programação de robôs pode ser dividida em *Manual Programming* (Programação Manual) e *Automatic programming* (Programação automática) (BIGGS; MACDONALD, 2003). Na primeira, o usuário/programador precisa diretamente desenvolver o programa e, consequentemente, executá-lo no robô para que adquira um comportamento. O robô não precisa estar presente enquanto é realizada a programação. Assim como a programação de computadores convencionais, a Programação Manual pode ser feita por meio de linguagens textuais como C, Java, Python etc, ou por meio de modelos gráficos que permitam a geração automática de código textual tais como linguagens de programação visuais baseadas em blocos. A Seção 2.1.1 apresenta detalhes sobre estas categorias.

Na Programação Automática, diferentemente, o usuário não precisa ter acesso direto ao código fonte do software de controle do robô, ou seja, o comportamento executado pelo robô é obtido a partir de inferências sobre sua base de conhecimento. Esta base, por sua vez, é composta pelo tratamento, realizado por algoritmos específicos, dos dados capturados pelos sensores do robô. Este tipo de programação abrange três categorias ilustradas na Figura 3 e detalhadas na Seção 2.1.2: *learning systems* (Sistemas de Aprendizagem), *Programming by Demonstration* (Programação por Demonstração) e *Instructive Systems* (Sistemas Instrutivos).

Figura 3 – Categorias da *Automatic programming* (Programação Automática)



Fonte: Adaptado de (BIGGS; MACDONALD, 2003).

2.1.1 Sistemas de Programação Manual

A forma mais comum de programar manualmente um robô é a utilização de linguagens de programação convencionais como C, Java, Python e etc, para o desenvolvimento do código. Os usuários desses sistemas precisam entender especificamente sobre a linguagem que se pretende utilizar, conhecer as APIs de programação do robô, bem como ser especialista na tarefa que se quer programar, ou ter acesso direto à equipe de produção que domine estas tarefas. A Programação Manual se torna mais adequada quando o ambiente em que o robô está inserido não requer mudanças contínuas, obrigando o robô a alterar seu comportamento para se ajustar às necessidades de um momento específico.

A Programação Manual é classificada pelor autores (BIGGS; MACDONALD, 2003) em duas categorias principais, são elas: *Text−based Systems* (Sistemas Baseado em Texto) e *Graphical Systems* (Sistemas Gráficos).

Em Sistemas Baseados em Texto, um programador precisa escrever códigos textualmente para implementar a inteligência do robô, desta forma, o processo para concepção de um comportamento robótico, demanda tanto profissionais em programação textual, quanto especialista na tarefa em particular que o dispositivo deve executar, nesta, a forma da programação utilizada para o desenvolvimento do programa, é o que necessariamente difere um sistema de outro.

Dois principais métodos levam em consideração a especificidade da linguagem utilizada, o primeiro caracteriza-se pela ausência de uma linguagem padronizada entre diferentes fabricantes de robôs, ou seja, neste método utiliza-se uma linguagem proprietária apropriada para o modelo do robô ou do fabricante, desta forma, se uma

fábrica utiliza robôs de muitos fabricantes diferentes, então ela terá que treinar seus programadores para cada um, ou pagar para o fabricante desenvolver os programas necessários (BIGGS; MACDONALD, 2003), o que representa uma significante adversidade para realizar a programação de um robô.

O segundo método, ao contrário do primeiro, utiliza uma linguagem aberta de propósito geral, o termo “Geral”, traduz uma linguagem de múltiplos propósitos de alto nível, um exemplo disto é a linguagem C++. Essa abordagem é especialmente utilizada em cenários envolvendo pesquisas, onde há uma seleção de uma linguagem para ser utilizada como base, afim de acolher as necessidades do projeto.

Por fim, outro método interessante na programação manual, caracteriza-se por sistemas programados por uma lista de instruções específicas, ou seja, o usuário seleciona manualmente em uma lista, quais comportamentos o robô deve adquirir em um dado momento. Desta forma, o usuário não precisa ser programador a nível de linhas de código, a evolução desta abordagem fez com que esse método se tornasse mais tarde, uma das categorias na Programação Automática. A principal desvantagem desse método é que o robô não é capaz de aprender novos comportamentos, ele é capaz apenas de adquirir comportamentos previamente definidos, os tornando inúteis em cenários onde há mudanças constantes de atividades.

Os softwares de programação baseados em componentes gráficos proporcionam aos programadores uma alternativa mais produtiva para a programação ao oferecer objetos gráficos para representação dos movimentos dos robôs. Não se tem acesso direto a linhas de código de modo textual, mesmo assim, essa abordagem não se caracteriza como uma forma automática de programação devido ao fato dos usuários/programadores ainda precisarem fornecer os comandos necessários ao sistema manualmente (como especificar ações e fluxos do programa, para que isso reflita em um comportamento característico do robô). Por ser um método mais simples de programação, caso o usuário adquira um treinamento adequado, o próprio especialista no desenvolvimento da tarefa no mundo real pode ser capaz de programar um robô para reproduzi-la, sem a necessidade de outro programador trabalhando em conjunto (BIGGS; MACDONALD, 2003). Para a programação manual, esta é a abordagem mais próxima de uma programação automática. A principal adversidade neste método é a demora resultante na configuração de cada tarefa que o robô deve executar, pois deve ser feito individualmente para cada atividade específica. Além disso, para um usuário se tornar um programador utilizando esse método, o mesmo precisa passar por um treinamento bastante específico para aprender a utilizar o programa.

2.1.2 Sistemas de Programação Automática

Os sistemas de aprendizagem, uma das categorias de programação automática, são aqueles que tem como característica a capacidade de aprender a realizar uma determinada atividade de maneira autônoma, tomando decisões por conta própria, a partir da habilidade de realizar autoexploração. Geralmente, são utilizadas técnicas de inteligência artificial conexionista, como redes neurais, para fornecer a inteligência necessária para que o robô aprenda a partir de erros do passado e tome decisões fundamentais com o objetivo de não repeti-las no futuro. Nessa abordagem o tempo se torna um dos problemas cruciais, uma vez que, para o robô aprender corretamente como proceder, ele precisa realizar inúmeras tentativas até conseguir alcançar o objetivo pretendido.

A programação por demonstração é, dentro da Programação Automática, o método mais comum para realizar a programação de um robô. Tem como principal característica a necessidade de um instrutor que ensine ao robô como realizar uma dada atividade. Diferentes tipos de interfaces podem ser utilizadas entre o usuário/programador e o sistema robótico com o intuito de transmitir o comportamento. Convencionalmente, utilizam-se *teach-pendants* para demonstrar os movimentos que o robô deve realizar (BIGGS; MACDONALD, 2003). Contudo, métodos mais naturais de comunicação também são possíveis, como gestos ou até mesmo a própria voz do usuário. Sistemas PbD também podem captar demonstrações, por meio do toque do usuário/programador no robô, movendo-o diretamente no sentido da atividade desejada. Por ser o foco deste trabalho detalharemos outros aspectos do PbD na Seção 2.2.

Por fim, os Sistemas Instrutivos tem como principal característica, a necessidade de receber instruções sequenciais do usuário, para que o robô realize uma atividade já conhecida. Essa técnica é mais adequada para comandar os robôs para realizar tarefas que eles já foram treinados ou programados para executar (BIGGS; MACDONALD, 2003). Desta forma, o usuário/programador apenas ordena que o robô realize uma atividade, e por sua vez, o robô entra em operação a partir do conhecimento prévio de como proceder. Os métodos mais utilizados para transmitir instruções para o robô são por meio de reconhecimento de gestos e voz.

A utilização dos métodos de Programação Automática são interessantes quando os robôs estão inseridos em ambientes onde há uma necessidade contínua de alternância no seu comportamento, como por exemplo, robôs que precisam executar inúmeras atividades em linhas de produção nas indústrias, onde a busca contínua no ganho de produtividade, demanda robôs cada vez mais versáteis e inteligentes o suficiente para aprender a executar novas atividade com rapidez e eficiência. Sistemas baseados em Programação Automática procuram facilitar a forma como um usuário ou, até mesmo, o ambiente transferem um comportamento desejado para um robô, reduzindo a neces-

sidade de um especialista em programação.

A Tabela 1 apresenta os principais tipos de sistemas, a categoria deles em relação ao tipo programação utilizada (Automática ou Manual) e as diferenças entre as tecnologias comumente utilizadas para cada um. Essas tecnologias são essenciais para fornecer aos sistemas os dados e/ou comandos necessários para a programação do robô.

Tabela 1 – Principais diferenças entres os tipos de programação

Tipo dos Sistemas	Categoria da Programação	Tecnologias utilizadas
Sistemas de Aprendizagem	Automática	Redes Neurais, Sensores.
Programação por Demonstração	Automática	Teach pendant, Joystick, Sensores, Câmeras.
Sistemas Instrutivos	Automática	Sensores, voz, gestos etc.
Sistemas Baseados em Texto	Manual	Linguagens como C++, Java e Python.
Sistemas Gráficos	Manual	Componentes Gráficos, Softwares Gráficos de Simulação.

2.2 Características dos sistemas PbD

O primeiro sistema baseado em Programação por Demonstração - PbD inicialmente não foi projetado para a aplicação em robôs. No ano de 1975, o sistema Pygmalion - um programa para estimular o pensamento criativo em pessoas - foi desenvolvido por David C. Smith (CYPHER; HALBERT, 1993), dando então os primeiros passos na ideia de um ser humano programar um sistema apenas demonstrando a ele como realizar determinada tarefa, sem ser necessário ter acesso ao código fonte do sistema. Desta forma os usuários de sistemas baseados em PbD, não necessariamente precisam ter conhecimento sobre programação, já que devem apenas demonstrar ao sistema como realizar uma determinada tarefa e, por sua vez, o próprio sistema deve aprender e criar um programa interno que realize a tarefa refletindo as ações do usuário.

Uma das motivações para o avanço no estudo da PbD foi a mudança de perfil dos usuários de computadores ao longo do tempo. Nas décadas de 60 e 70, a maioria dos usuários tinham conhecimento sobre programação e construíam seus próprios programas, porém, com o passar do tempo e a sofisticação dos aplicativos, os usuários passaram reconfigurar seus aplicativos e computadores sem programar e sem a

necessidade de um especialista. Segundo (CYPHER; HALBERT, 1993) “usuários de computadores contemporâneos são ’usuários finais’, o que significa que eles estão no final do processo de programação de computadores, longe do programador”. Desta forma, sistemas capazes de se reprogramar se mostraram alternativas interessantes para se adequar à mudança do perfil dos usuários finais de computadores.

O maior potencial para o uso de programação por demonstração é automatizar atividades repetitivas (CYPHER; HALBERT, 1993). Realizar manualmente uma tarefa repetitiva por um longo período de tempo ou por uma grande quantidade consecutiva de vezes além de ser tedioso se torna propício ao aparecimento de erros. Utilizar máquinas para reproduzir tais atividades além de favorecer a redução do tempo de produção de uma indústria, racionaliza os seus gastos com mão de obra, que podem ser direcionados a outras atividades.

Os sistemas baseados em PbD são caracterizados por (CYPHER; HALBERT, 1993) em quatro dimensões. A primeira dimensão é denominada *Uses and Users* (Usos e Usuários), e busca identificar qual o exato domínio da aplicação e quem são os seus usuários. A segunda dimensão é denominada *User Interaction* (Interação com Usuário) que leva em consideração a forma com que o usuário se comunica com o sistema para gerenciar funções no programa. A terceira dimensão denominada como *Inference* (Inferência), busca explicar como um sistema elege uma compreensão generalizada para as demonstrações realizadas pelo usuário, baseado em uma amostra limitada de exemplos. A última dimensão e não menos importante, é denominada de *Knowledge* (Conhecimento), que basicamente refere-se ao momento de tirar por conclusão, quais informações o sistema pode usar.

A compreensão destas características podem tanto ajudar na avaliação e comparação entre diferentes sistemas PbD, como, quando tomadas como modelo, auxiliar projetistas a definir a contribuição dos seus sistemas no campo de pesquisa da PbD (CYPHER; HALBERT, 1993).

2.2.1 *Programming by Demonstration* na Robótica

Programação de robôs por Demonstração (PbD), tem se tornado um tema central na robótica que se estende por diversas áreas de pesquisa tais como: Interação Humano Robô, Aprendizado de Máquina, Visão de Máquina e Controle Motor (BILLARD et al., 2008). No início da década de 80, PbD surgiu como uma alternativa promissora para o setor industrial que demandava uma grande quantidade dispositivos robóticos nas suas linhas de produção. Com as técnicas PbD, o processo manual de programação dos equipamentos por codificação poderia se tornar completamente automatizado.

PbD também se refere a aprendizagem por imitação e é um mecanismo pode-

roso para reduzir a complexidade dos espaços de busca para a aprendizagem (BILLARD et al., 2008). Nós, seres humanos, ao observarmos ações que geram resultados positivos ou negativos, intuitivamente reduzimos nossa busca por uma solução possível para um problema em observação, baseando-se em um modelo que gere um resultado significativo.

A ideia básica da Programação por demonstração é permitir que o robô observe um ser humano executar uma tarefa para extrair o máximo possível de informações a partir da demonstração e mapeá-la em uma representação abstrata e generalizada da tarefa demonstrada (ZöLLNER, 2004). As vantagens oferecidas pela utilização de PbD podem ser observadas quando se considera que o processo de reprogramação de um robô através de codificação direta a cada nova tarefa que ele pretende executar, demanda gastos com capital humano especializado e, a depender da experiência do programador, maior tempo de programação total por conta da introdução de imprecisões no código.

Sistemas PbD mais complexos utilizam diversos recursos de hardware, como sensores e câmeras, para possibilitar o robô observar a tarefa que terá que reproduzir, bem como dados inerentes ao ambiente o qual está inserido, quando esses recursos não são possíveis, outra forma de repassar informações necessárias para o robô sobre a atividade pretendida é com a utilização de um *teach-pendant*.

O uso deste artefato permite que o usuário que programa a tarefa, seja o mesmo especialista da atividade no mundo real, desta forma não há necessidade de um programador trabalhando em conjunto, para fornecer os comandos necessários robô, como exemplo, para demonstrar a um robô como ele deve realizar uma atividade de solda de uma peça metálica qualquer, o próprio soldador pode programar o robô por demonstração de como fazer a tarefa. Imprecisões fornecidas pelo usuário não observadas no momento da demonstração da tarefa, só serão percebidas da fase de teste, podendo acarretar perca de tempo e produtividade.

2.2.2 Sistemas robóticos baseados em PbD

Um sistema baseado em PbD é utilizado em (ZöLLNER, 2004) para observar, aprender e generalizar atividades executadas por seres humanos, para um manipulador robótico de dois braços. Neste trabalho um ciclo para Programação por Demonstração é proposto para transferir as demonstrações da tarefa para o robô, com objetivo de absorver o maior número possível de informações da demonstração. Com posse dessas informações o sistema cria uma abstração mais generalizada da tarefa com a finalidade de aproveitá-la em outros modelos diferentes de robôs. O clico PbD é dividido em três fases fundamentais. Na primeira fase é feita a percepção e interpretação da demonstração fornecida pelo usuário, para perceber a tarefa recursos como câme-

ras, sensores táteis e luvas com rastreadores magnéticos são utilizados. Na segunda fase é gerada a abstração da tarefa por meio de uma generalização. Na última fase do ciclo é feito o mapeamento das tarefas abstratas para os robôs específicos.

Em outro estudo realizado por (EKVALL, 2005), um sistema de aprendizagem por demonstração é integrado a um sistema de planejamento de nível de tarefa. O esquema geral do sistema, se baseia em um professor fornecer as demonstrações de como realizar uma determinada tarefa, e por sua vez o robô utiliza de entradas visuais para observar tais demonstrações, em seguida o robô planeja a tarefa e por fim ele a executa.

Três maneiras diferentes podem ser utilizadas para o aprendizado da tarefa. A primeira, denominada de *Imitation Learning* (Aprendizagem de Imitação), é comumente usada para representar a tarefa de aprendizagem em um nível baixo, considerando reprodução de trajetórias e/ou configurações conjuntas do robô (EKVALL, 2008). Com esta técnica é apenas possível reproduzir uma tarefa utilizando as mesmas coordenadas, sem mudar nada no trajeto. A segunda maneira, denominada de *Learning in Dialogue with Teacher* (Aprendizagem em Diálogo com Professor), um humano realiza o papel de um professor, demonstrando a tarefa enquanto explica o passo a passo. Restrições podem ser concebidas pelo humano professor, fazendo com que o robô seja disciplinado, aprendendo como deve ou não agir, prevenindo decisões erradas. Por último, tem-se a *Generalizing from Multiple Observations* (Generalização de Múltiplas Observações), um robô deve ser capaz de aprender a realizar uma nova tarefa a partir de um conjunto de demonstrações, extraindo deste conjunto, um modelo geral de execução da tarefa. Várias observações da mesma tarefa podem ser utilizadas para formar um modelo mais geral, e portanto, flexível da tarefa (EKVALL, 2008).

Em (FORBES et al., 2014), um arcabouço PbD é proposto. A partir de uma demonstração inicial, o robô recolhe mais informações de um conjunto de demonstrações providos por diversas pessoas diferentes e utilizando a demonstração inicial como semente. Na sequência, o robô busca cenários onde a demonstração classificada como semente não irá funcionar mas que provavelmente seja remediável, por fim o robô executa a ação no novo cenário usando as demonstrações selecionadas.

Dividida em duas diferentes fases denominadas de *Task Learning* (Tarefa de Aprendizagem) e *Task Refining* (Tarefa de Refino) respectivamente, (MOLLARD et al., 2015) apresentam uma abordagem para realizar programação robótica por demonstração que contempla *Feedback* e Transferência de Conhecimento. Para o *feedback*, palavra utilizada para expressar uma mensagem de retorno como resposta a alguma ação realizada, o sistema utiliza uma GUI para interação com o usuário por uma representação 3D. Desta forma, o usuário pode idealizar intuitivamente o planejamento da tarefa, bem como corrigir possíveis imprecisões antes mesmo da execução. Este

sistema de simulação 3D permite ao usuário reparar possíveis imprecisões.

A Tabela 2 expõe um resumo dos sistemas PbD observados, onde na segunda coluna apresenta o modo que é realizado a demonstração da tarefa para o robô, na terceira coluna apresenta qual a aplicabilidade desses sistemas e por fim quais são as técnicas utilizadas nos trabalhos.

Tabela 2 – Resumo dos sistemas PbD Observados

Referência	Modo da Demonstração	Aplicação	Técnicas Utilizadas
(ZöLLNER, 2004)	Gestual	Robô com dois braços, Robô Humanoide.	Redes Neurais,SVM, câmeras, sensores táteis,luvas com rastreadores magnéticos.
(EKVALL, 2005)	Gestual	Manipulador simples para atividades caseiras.	rastreador magnético, sensores de medição.
(EKVALL, 2008)	Entrada Visual	Manipulador de objetos simples.	Sistema de interface para programar a tarefa.
(FORBES et al., 2014)	Toque no robô	Manipulador de objetos simples.	Sistema gráfico de simulação.
(MOLLARD et al., 2015)	Simulação 3D	Robô para montagem de produtos.	GUI de simulação 3D para programar a tarefa.

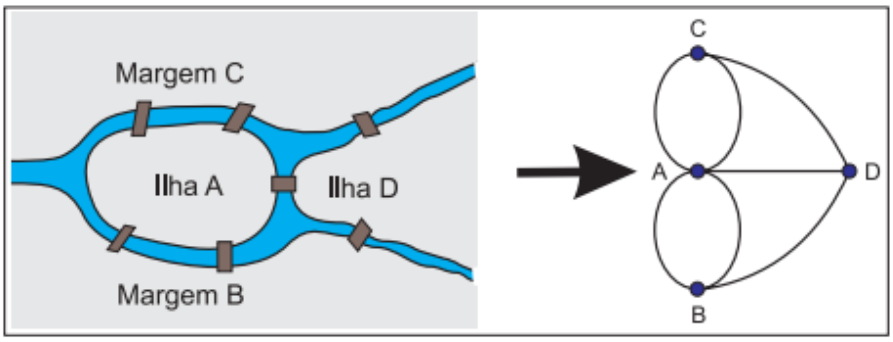
2.3 Teoria dos Grafos

Os primeiros estudos relacionados à teoria dos grafos foram conduzidos cerca de três séculos atrás. Na época, o estudo que se tornou mais relevante para o surgimento e exploração da teoria foi publicado por Leonhard Euler, no ano de 1736, sob o título ”A solução para o problema relativo à geometria da posição”, que investiga o problema conhecido como Sete Pontes de Königsberg.

Atualmente, chamada de Kaliningrado e pertencente ao território russo, a cidade de Königsberg é atravessada por um rio, formando duas ilhas, e possui sete pontes que foram criadas para conectar todo o complexo, ilustrado na Figura 4. O problema consiste em, a partir de um determinado ponto, passar por todas as pontes somente uma vez e retornar ao ponto inicial (CARVALHO, 2005).

A teoria dos grafos na matemática investiga a associação finita entre componentes de um determinado grupo, onde cada componente deste grupo é denominado

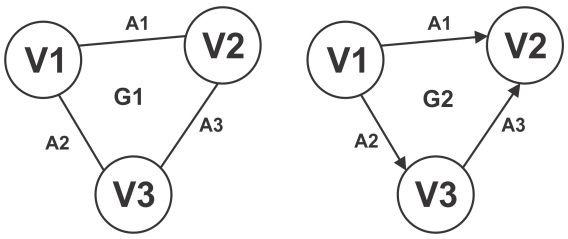
Figura 4 – Ilustração do problema Sete Pontes de Königsberg.



Fonte: Retirado do artigo (RODRIGUES, 2007).

de vértice ou nó, e cada relação entre um par de vértices desse mesmo grupo é denominado como aresta. Assim, pode-se entender o grafo como uma estrutura formada por dois tipos de objetos: vértices e arestas. Cada aresta é um par de vértices, ou seja, um conjunto com exatamente dois vértices (FEOFILOFF, 2012). Quando estes pares de vértices são ordenados, chama-se o grafo de direcionado, caso contrário, quando os pares de vértices não são ordenados, chama-se o grafo não-direcionado. Estes tipos de grafos são ilustrados na Figura 5. Em G1 as arestas A1, A2 e A3 não possuem direção, enquanto em G2, como exemplo, as arestas A1, A2 e A3, necessariamente possuem uma direção específica, sempre saindo de um nó e entrando em outro nó vizinho.

Figura 5 – Representação de grafos: G1 não-direcionado e G2 direcionado



Fonte: Autor

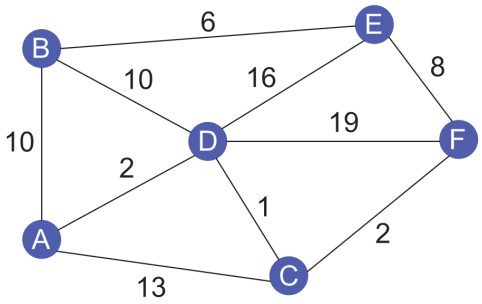
Quando as arestas pertencentes a um grafo possuem pesos, este é chamado grafo ponderado. Neste tipo de grafo, um peso ou conjunto de pesos é associado a cada aresta, representado da forma $w(i, j)$, ou seja, $w(1, 2)$ é o peso associado a aresta que une os nós 1 e 2 (CARVALHO, 2005).

Dentro da Teoria dos Grafos, existe um problema computacional conhecido como Problema do Caminho Mínimo, que pode ser caracterizado da seguinte forma: Dado um grafo ponderado, onde todas suas arestas possuem um peso associado, deve-se encontrar o caminho de distância mínima entre um vértice inicial A e um vértice final Z. O caminho de distância mínima entre um vértice inicial A e um vértice final Z é aquele cujo somatório de todos os pesos das arestas possui valor mínimo compa-

rado com todos os outros caminhos possíveis entre os vértices A e Z. Para solucionar esse problema, pode-se usar o Algoritmo de Dijkstra.

Considere o grafo ponderado ilustrado na Figura 6 representando um grafo com os vértices A,B,C,D,E e F. Para calcular todos os caminhos possíveis, entre o vértice inicial A e todos os outros vértices do grafo, o uso do Algoritmo de Dijkstra pode ser vantajoso.

Figura 6 – Representação de um Grafo Ponderado



Fonte: Autor

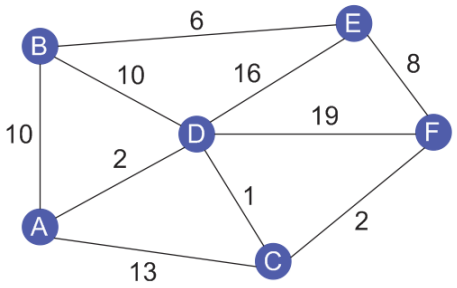
Ao iniciar sua execução, o algoritmo em questão julga que o custo mínimo entre o vértice definido como inicial e todos os outros vértices do grafo é preliminarmente considerada como infinito (∞), e na medida que vai avançando esse custo vai sendo regulado. Sempre quando um caminho é considerado menos custoso, entre dois vértices (A e D por exemplo), este último vértice é considerado como “Fechado” (o vértice D no exemplo).

Primeiramente, exceto para o vértice A definido como inicial, a distância entre todos os vértices do grafo é considerado infinita (∞), para o vértice A é zero. O procedimento executado pelo algoritmo pode ser representado em uma tabela onde a medida que for avançando os dados vão sendo alterados. Na Tabela mostrada na Figura 7 a primeira linha equivale à representação dos vértices do grafo ilustrado ao lado da Tabela, a segunda linha representa os custos entre os vértices e seus precedentes (o precedente de um vértice t é o vértice que precede t no caminho de custo mínimo de a para t), que está representado na terceira linha. Por último estão os dados pertinentes a informação se um vértice está “Fechado” ou não, caso não, recebem o caractere N, caso isso seja verdade recebe então o caractere S.

Em seguida, o vértice cujo o custo associado a sua aresta for menor, em comparação a todos os outros, é selecionado e marcado como “Fechado”. Para o exemplo mostrado na Figura 8 em diante, foi selecionado o vértice A como ponto de partida. Partindo dele, são recalculadas os demais custos para os vértices adjacentes que ainda não foram visitados, ou seja, possuem na tabela o caractere N na linha “Fechado”. Quando o custo calculado é menor que o custo anteriormente armazenado, é feita

Figura 7 – Tabela e Grafo representando o funcionamento do Algoritmo de Dijkstra

Vértice	A	B	C	D	E	F
Custo	0	∞	∞	∞	∞	∞
precedentes	'	'	'	'	'	'
Fechado	N	N	N	N	N	N



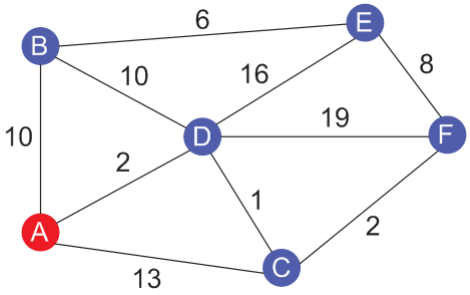
Fonte: Autor

então a substituição e o vértice por onde esse custo foi calculado, é atribuído como precedente na tabela.

Na Figura 8 o vértice A foi selecionado e marcado como ”Fechado”, todos os custos relacionados aos vértices adjacentes a ele são calculados e substituídos na Tabela, os vértices não adjacentes ainda continuam com o valor infinito (∞).

Figura 8 – Tabela e Grafo relacionado ao vértice A

Vértice	A	B	C	D	E	F
Custo	0	10	13	2	∞	∞
precedentes	A	A	A	A	'	'
Fechado	S	N	N	N	N	N



Fonte: Autor

Continuamente, a mesma lógica segue para o segundo passo, ou seja, selecionar os vértices adjacentes que ainda não foram visitados, marcar aquele que possuir menor curso associado como “fechado”, e a partir dele realizar os cálculos e substituir na tabela apenas aqueles que possuírem um custo inferir ao anteriormente armazenado na tabela.

Na Figura 9 o vértice D foi selecionado e marcado como ”Fechado”, os custos de C, E e F alterados na tabela. Para os dois últimos, qualquer valor é menor que infinito, os precedentes de C, E e F foram alterados para D.

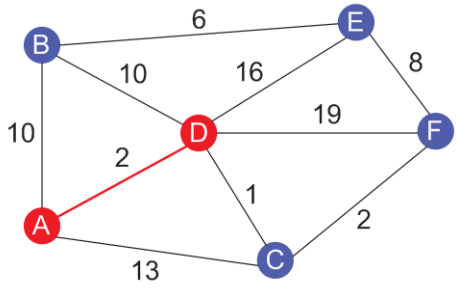
Na Figura 10 o vértice C foi selecionado e marcado como ”Fechado”, ele possui ligação disponível apenas com o vértice F, o custo e o precedente de F foram alterados.

Na Figura 11 o vértice F foi selecionado e marcado como ”Fechado”, o custo e o precedente de E foram alterados.

Na Figura 12 o vértice B foi selecionado e marcado como ”Fechado”, nenhum

Figura 9 – Tabela e Grafo relacionado ao vértice D

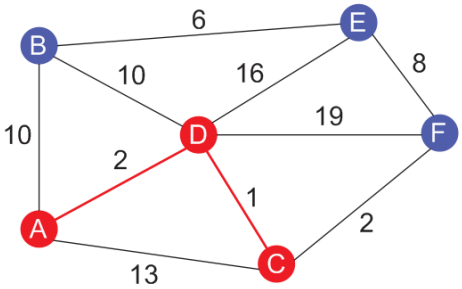
Vértice	A	B	C	D	E	F
Custo	0	10	3	2	18	21
precedentes	A	A	D	A	D	D
Fechado	S	N	N	S	N	N



Fonte: Autor

Figura 10 – Tabela e Grafo relacionado ao vértice C

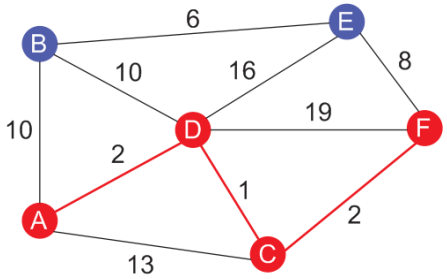
Vértice	A	B	C	D	E	F
Custo	0	10	3	2	18	5
precedentes	A	A	D	A	D	C
Fechado	S	N	S	S	N	N



Fonte: Autor

Figura 11 – Tabela e Grafo relacionado ao vértice F

Vértice	A	B	C	D	E	F
Custo	0	10	3	2	13	5
precedentes	A	A	D	A	F	C
Fechado	S	N	S	S	N	S



Fonte: Autor

custo ou precedente foi alterado na tabela.

Na Figura 13 o vértice E foi selecionado e marcado como ”Fechado”, nenhum custo ou precedente foi alterado na tabela.

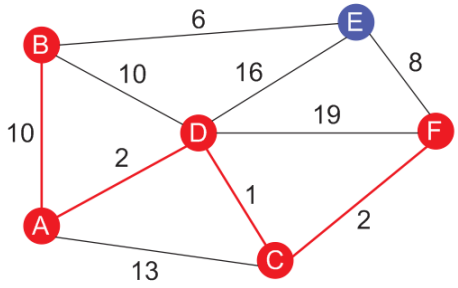
Ao fim da execução do algorítimo, quando todos os vértices do grafo são visitados e marcados como ”Fechados”, a distancia mínima entre o vértice definido como inicial para todos os outros do grafo, estão disponíveis na tabela, como um exemplo, o custo mínimo para o caminho do vértice A até o vértice F é de 5 indo pelo vértice C.

Como limitação do algoritmo de Dijkstra, deve-se observar que os pesos do grafo ponderado não podem conter valores negativos.

Seja $G(V, E)$ um grafo orientado e a um vértice de G, o algoritmo de Dijkstra,

Figura 12 – Tabela e Grafo relacionado ao vértice B

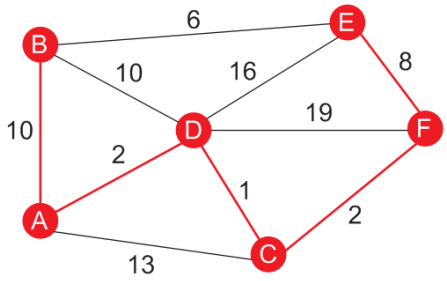
Vértice	A	B	C	D	E	F
Custo	0	10	3	2	13	5
precedentes	A	A	D	A	F	C
Fechado	S	S	S	S	N	S



Fonte: Autor

Figura 13 – Tabela e Grafo relacionado ao vértice E

Vértice	A	B	C	D	E	F
Custo	0	10	3	2	13	5
precedentes	A	A	D	A	F	C
Fechado	S	S	S	S	S	S



Fonte: Autor

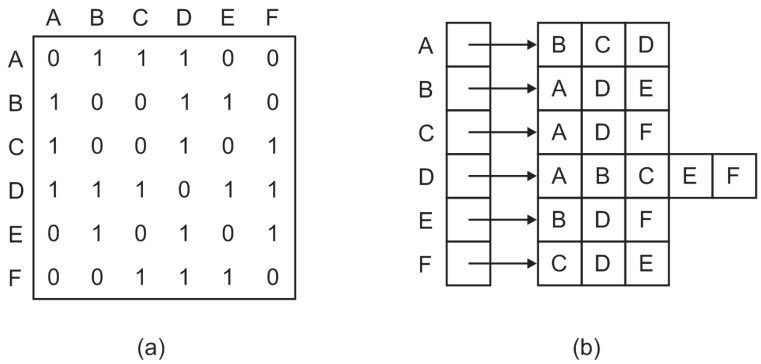
retirado de (CARVALHO, 2005), pode ser enunciado como:

1. Atribui-se zero à estimativa do custo do vértice a (a raiz da busca) e infinito às demais estimativas;
2. Atribui-se um valor qualquer aos precedentes;
3. Enquanto houver vértice aberto:
    a) Escolha k como o vértice aberto cujo custo seja o menor dentre os vértices abertos;
    b) fecha-se o vértice k
    c) Para todo vértice j aberto que seja sucessor de k faz-se:
        i. soma-se a estimativa do vértice k com o custo da aresta que une k a j;
        ii. caso a soma seja menor que a estimativa anterior para o vértice j, substitui-se o custo e anota-se k como precedente de j.

A forma como um grafo é representado no computador pode impactar diretamente no desempenho de um algoritmo que receba um grafo de entrada e no consumo de memória. Os grafos podem ser representados no computador por meio da matriz de adjacências ou da lista de adjacências. A matriz de adjacências de um grafo com

$|V|$ vértices é uma matriz $|V| \times |V|$ de 0s e 1s, na qual a entrada na linha i e coluna j é 1, se e somente se, a aresta (i, j) estiver no grafo (KHAN…, 2016). A Figura 14(a) ilustra a matriz de adjacências do grafo representado na Figura 6.

Figura 14 – Representação de um grafo por meio da uma Matriz de Adjacências (a) e de uma Lista de Adjacências (b)



Fonte: Autor

Outra forma de representar um grafo no computador é por meio de uma lista de adjacências. Essa representação mantém, para cada vértice do grafo, uma lista de todos os vértices adjacentes a ele. Tipicamente, tem-se $|V|$ listas de adjacências (KHAN…, 2016). A Figura 14(b) apresenta a representação por meio de listas de adjacências do grafo mostrado na Figura 6.

3 Proposta

Os sistemas PbD mais complexos dispõem, geralmente, de dispositivos sensoriais e câmeras de vídeo para que o robô obtenha dados sobre a tarefa que o usuário pretende transmitir a ele. Sem esses recursos, contudo, o robô se torna incapacitado de aprender tais demonstrações por observação, fazendo se então necessária a utilização de outros métodos. Com o uso de um *teach pendant*, o usuário/programador pode programar robôs na indústria para tarefas repetitivas transmitindo ao robô os comandos sequenciais necessários para realizar uma tarefa, não necessitando que o robô faça uso de sensores para capturar o movimento do usuário ou para certificar-se de sua própria posição no ambiente. Um exemplo de tarefa seria, por exemplo, capturar um objeto em um ponto A, manipular este objeto, e levá-lo a um ponto B qualquer.

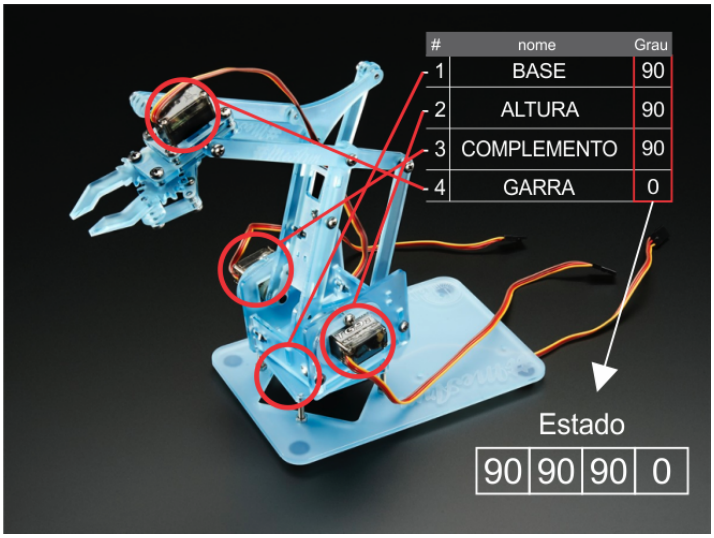
Comumente, o usuário precisaria demonstrar apenas uma vez ao robô a tarefa a ser executada, contudo o ensino da tarefa pode vir acompanhado por imprecisões na demonstração fornecida pelo usuário (devido à falta de experiência na execução da tarefa, por exemplo). Assim, a programação de tarefas complexas por meio de *teaching pendants* pode se tornar morosa devido à imprecisões introduzidas durante a fase de programação do robô, o que obrigaria a uma fase de reprogramação, após a fase de testes, aumentando o tempo que o robô permanece parado. Retomando o exemplo anterior, uma imprecisão pode ocorrer se o usuário, ao manipular o controle, movimentar o robô por um traçado não desejado ou pousar o objeto em um ponto diferente de B.

A estratégia proposta neste trabalho permite que o robô aprenda uma determinada tarefa mesmo na presença de imperfeições na execução da tarefa. Para mitigar o efeito destes erros, o usuário realiza diversas demonstrações de uma mesma tarefa para que o robô consiga inferir a melhor forma de executá-la a partir dos movimentos mais frequentes nas demonstrações. Além disso, a estratégia permite que a demonstração seja feita por um usuário/programador sem conhecimentos em programação, o que potencialmente reduz os custos de programação.

A estratégia proposta emprega uma abordagem baseada em grafos que deve ser aplicada a um robô equipado com um conjunto de atuadores que permitam que este execute tarefas. Para mapear os possíveis estados dos atuadores do robô utiliza-se um grafo em que cada nó representa uma configuração possível dentro do conjunto de graus de liberdade dos atuadores do robô. A Figura 15 ilustra a relação entre os graus dos servomotores do robô utilizado neste estudo e um estado do grafo. As arestas do grafo, por outro lado, representam movimentos possíveis executados pelo robô de acordo com as limitações dos seus atuadores, do joystick utilizado na programação e

da tarefa realizada. A transição de estados, ou seja, o mudança de um nó para outro nó vizinho dentro do grafo, passando necessariamente por uma aresta, representa então um movimento do robô.

Figura 15 – Relação entre graus dos motores e um estado do grafo



Fonte: Adaptado de (ELECTRONICS, 2016).

Enquanto o usuário manipula o robô na execução da tarefa fornecendo comandos de forma sequencial através do joystick, o algoritmo de aprendizado armazena o caminho feito enquanto percorre o grafo de um nó para outro, ou seja, de um estado para o outro, através das arestas até que a tarefa seja cumprida. Ao término da execução da tarefa, o usuário forneceu ao robô uma demonstração possível de como realizar a atividade proposta, a qual é armazenada pelo algoritmo. Novas demonstrações podem ser feitas da mesma forma.

3.1 Modelos Robóticos

O sistema PbD proposto nesse trabalho pode ser implantado em diversos modelos robóticos diferentes, por manter uma abordagem baseada em grafo, onde cada vértice equivale a um conjunto de graus configurados nos servomotores do dispositivo. Este conjunto se traduz numa posição física do estado atual do robô no mundo real, e cada aresta do grafo, equivalente à transição de um estado para outro adjacente, representando um movimento realizado pelo robô.

O primeiro requisito necessário para que o sistema seja implantado em um modelo robótico candidato é que os atuadores do dispositivo devem ser discretos, ou devem poder ser discretizados sem perdas para o robô. Em outras palavras, deve-se dispor de informações de todas as possíveis configurações que os atuadores do robô podem assumir. No caso de um laser, por exemplo, pode-se ter níveis como ligado

e desligado, ou ainda níveis discretos de intensidade. No caso de servomotores, por outro lado, deve-se ter disponível as posições que o motor pode assumir.

Outra característica relevante nos dispositivos robóticos, é que eles não necessariamente precisam ter sensores responsáveis por captar o ambiente em que estão inseridos ou para observar as demonstrações da tarefa. Para o funcionamento, um simples joystick (como o que está representado na Figura 16) que acione os atuadores do robô pode ser utilizado para indicar os movimentos que o robô deve executar. Ainda para funcionamento do sistema, o modelo robótico deve possuir CPU e memória para o processamento dos dados do programa.

Figura 16 – Exemplo de Joystick para uso no sistema proposto.

Fonte: Retirado de (GENERIC..., 2016)

O sistema PbD apresentado também permite que as entradas possam conter imprecisões. Estas podem advir tanto do usuário ao prover diretamente comandos ao robô através do joystick, quanto pelo próprio meio, que por ser eletrônico, pode estar sujeito a algum ruído que refletirá em um movimento indesejado.

3.2 Modelo da Base de Conhecimento

O modelo da base de conhecimento para o desenvolvimento do sistema proposto utiliza um grafo para representar todos os possíveis estados que o robô pode alcançar. Este grafo de estados é criado a partir do relacionamento dos possíveis estados que seus atuadores podem assumir, contendo todos os nós possíveis e todas as arestas compatíveis. Assim, para um robô com n atuadores, onde cada atuador possui um conjunto de estados $A_i = \{s_{i1}, s_{i2}, s_{i3}, ..., s_{im}\}$, os nós do grafo $G = (N, E)$ serão os elementos do conjunto $N = A_1 \times A_2 \times A_3 \times ... \times A_n$. O total de nós deste grafo é dado por $|N| = |A_1||A_2||A_3|...|A_n|$.

A Tabela 3 exemplifica como são criados os estados de um robô com quatro servomotores. No caso, as colunas BASE, ALTURA, COMPLEMENTO e GARRA têm relação direta as funções específicas dos atuadores no robô. A linha Valor indica o ângulo do servomotor no caso de BASE, ALTURA e COMPLEMENTO, e indica também o estado do servomotor GARRA, que é binário (aberta ou fechada). A especificação do

valor em cada estado depende do conjunto de graus de liberdade que os servomotores possuem.

Tabela 3 – Exemplo de estado de um manipulador robótico com 4 atuadores.

Nome	BASE	ALTURA	COMPLEMENTO	GARRA
Valor	90	90	90	0

As arestas do grafo de estados são modeladas conforme os possíveis movimentos de cada atuador. Assim, como exemplo, a aresta $e_{11\rightarrow12} = (s_{11,23,31}, s_{12,23,31})$, refere-se a mudança do estado 1 do atuador 1 para o estado 2 do mesmo atuador, conservando os estados dos outros dois atuadores.

A Figura 17 ilustra um recorte do grafo de estados de um robô com quatro atuadores. Nele, os três primeiros atuadores do robô, foi predefinido que podem apenas ser incrementados ou decrementados do valor 5 (equivalente à angulação de um servomotor, por exemplo) e o último atuador só pode possui dois estados. O nó vermelho A corresponde ao estado atual do robô e possui arestas para sete nós vizinhos, que serão um dos nós visitados após o próximo movimento.

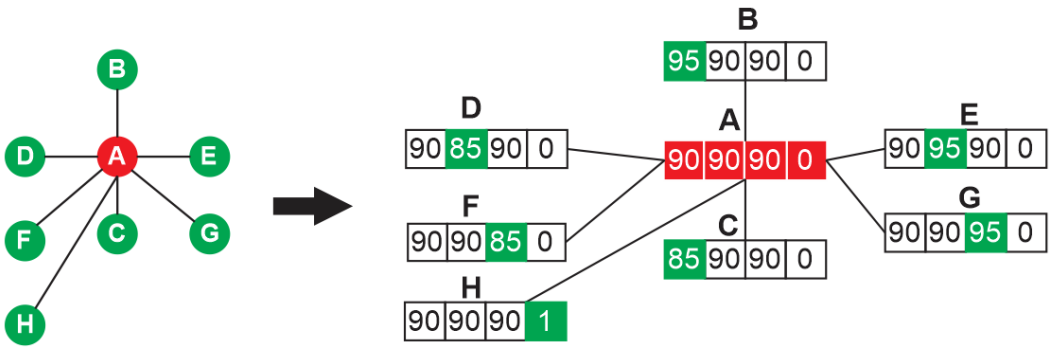
Os graus de liberdade dos servomotores do robô foram configurados para permanecer entre 60 e 120 graus, exceto para o servomotor da garra que apenas pode assumir dois estados (aberto ou fechado). Como exemplo, para o robô realizar um movimento total para esquerda, o servomotor responsável pela base do robô, precisa então ir para o grau 60, na mesa lógica inversa se o robô realizar um movimento total para direita, o servomotor da base irá para o grau 120. Por definição, o incremento e decremento dos graus dos servomotores do robô foram configurados passos de 5 graus por vez.

Para o servomotor da garra, o tratamento da configuração dos graus de liberdade foi implementado direto no Arduino, caso ele receba um valor 0, o grau de liberdade do servomotor é configurado para 60 graus, representado visualmente a garra fechada, caso ele receba um valor 1, o grau é então configurado para 120, representando visualmente a garra aberta.

Como cada vértice do grafo representa uma tupla formada pelo conjunto de graus de liberdade do robô, ele possui 3 servomotores que vão de 60 à 120 graus em passos de 5 graus por vez e um servomotor para garra que possui dois estados possíveis, para o sistema desenvolvido, 4394 vértices são necessários para representar todos os estados possíveis do robô.

Deve-se ressaltar ainda que todas as arestas $e \in E$ do grafo possuem um peso $w(e)$, cujo valor inicial é 0. Esses pesos serão fundamentais para o processo de treinamento, e serão discutidos com mais detalhes na Seção 3.3.2.

Figura 17 – A estrutura do grafo de estados de um robô com 4 atuadores.



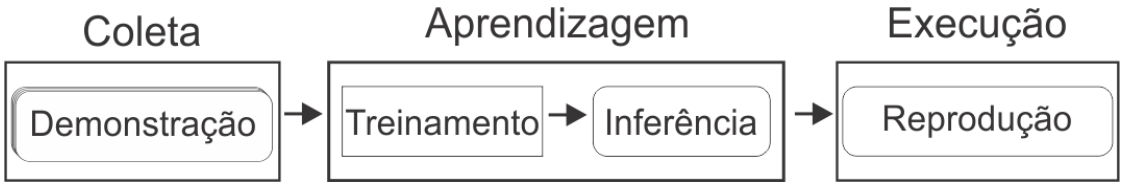
Fonte: Autor

Note-se que esta estratégia é suficientemente genérica para contemplar casos de um robô com muitos graus de liberdade, podendo o grafo ser completamente conectado, se for o caso. Porém, a depender do tipo do robô, do joystick disponível, e do cenário de aplicação, o grafo pode ter menos arestas. Por exemplo, em um cenário onde o joystick está limitado a mudar apenas um atuador por vez por meio de pequenos incrementos, o número de arestas no grafo fica reduzida, já que se pode podar as arestas correspondentes aos casos em que dois ou mais atuadores mudam simultaneamente. Esta poda é importante porque permite a redução do consumo de memória e tempo de processamento no grafo pelo uso de representações esparsas.

3.3 Processo proposto

O processo geral proposto para o sistema está dividido em três fases fundamentais, executadas sequencialmente: Coleta, Aprendizagem e Execução, conforme ilustrado na Figura 18. As próximas seções descrevem, em detalhes, cada uma destas etapas.

Figura 18 – Divisão das fases do processo de execução do sistema



Fonte: Autor

3.3.1 Coleta

Nesta fase, as demonstrações são fornecidas pelo usuário por meio de entradas recebidas pelo joystick. Estas entradas são mapeadas em estados do grafo e armaze-

nadas para servir como base de conhecimento para o sistema PbD.

Após a preparação do grafo, as demonstrações são coletadas da seguinte forma: a cada entrada fornecida pelo usuário no joystick, o robô muda os estados de seus atuadores no sentido de realizar um movimento da série de movimentos necessários para que o manipulador consiga cumprir determinada tarefa até o final. Ao executar a tarefa por completo, o usuário forneceu à base de conhecimento uma **demonstração** de como realizar a atividade em observação. Desta forma, cada demonstração é o resultado de uma sequência de movimentos fornecidos pelo usuário/programador, que expressam ao robô como ele deve se comportar, para cumprir uma determinada tarefa. Do ponto de vista da estratégia proposta, a tarefa executada pelo usuário é modelada na forma de um caminho sobre o grafo de estados.

Como podemos observar na fase de Coleta ilustrada na Figura 18, o usuário pode transmitir ao sistema tantas demonstrações da tarefa quanto deseje. O conjunto de todas estas demonstrações formam a **base de conhecimento** do robô que será utilizada na fase de Aprendizagem propriamente dita. Note-se que o pressuposto básico da aprendizagem do robô é que quanto maior a base de conhecimento (ou seja, quanto maior o número de demonstrações fornecidas pelo usuário), mais preciso será o robô ao decidir como desempenhar a tarefa.

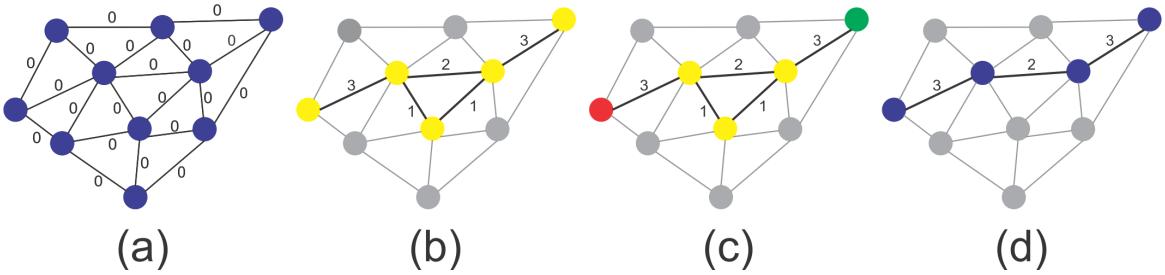
3.3.2 Aprendizagem

O processo de aprendizagem do robô é subdividido em duas etapas fundamentais. A primeira, denominada **Treinamento**, realiza a adaptação dos pesos do grafo de acordo com a base de conhecimento. Neste momento o robô não realiza nenhum movimento, o algoritmo apenas manipula os pesos das arestas do grafo. A segunda etapa do processo de aprendizagem, denominada **Inferência**, tem como finalidade selecionar uma sequência de movimentos para realizar a tarefa, desprezando as imprecisões do ensino.

A base de conhecimento formada após o término da fase de coleta (Figura 19(a)) é composta pelas demonstrações dadas pelo usuário que, em última análise, correspondem a diversos caminhos percorridos no grafo de estados de um nó inicial até um nó final. Assim, o Treinamento é realizado percorrendo novamente os caminhos e **reforçando as arestas percorridas**. Cada aresta do grafo de estados é incrementada em 1 para cada demonstração que utilize esta aresta (fase ilustrada na Figura 19(b)), com isso, as arestas mais visitadas indicam que esse movimento é desejado para a tarefa, pois foi realizado com mais frequência. As arestas não visitadas são, por fim, removidas do grafo, as arestas em cinza representam esta remoção na Figura 19(b).

Após o Treinamento, vem a etapa da **Inferência**. O primeiro passo na etapa

Figura 19 – Etapas do Processo de Aprendizagem para 3 demonstrações: (a) base de conhecimento inicial; (b) base de conhecimento após o treinamento; (c) base de conhecimento após a escolha dos nós final e inicial; (d) base de conhecimento após a escolha do maior caminho.



Fonte: Autor

de Inferência é escolher os nós inicial e final. Em ambos os casos, escolhe-se o nó por meio de eleição, isto é, o nó que foi mais utilizado como nó inicial (final) de cada uma das demonstrações na base de conhecimento é escolhido para ser o nó inicial (final) do caminho que o robô aprenderá. Em caso de empate na eleição, escolhe-se qualquer um dentre os nós mais votados. Note-se que, em alguns casos, pode-se evitar o cálculo do nó inicial fazendo com que o robô parta sempre de uma mesma condição inicial previamente configurada. A fase de seleção dos nós inicial e final é ilustrada na Figura 19(c), em que o nó vermelho representa o nó inicial e o nó verde representa o nó final.

Definidos os nós inicial e final, o caminho de interesse é aquele de maior distância, i.e., aquele cuja soma dos custos é a maior. A seleção do maior caminho no grafo é ilustrada na Figura 19(d), com os nós marcados de azul representando o caminho de maior distância entre os vértices inicial e final. Assim, pode-se perceber, que este é caminho que provavelmente contém o conjunto de movimentos que o robô deve descrever para bem executar a tarefa.

O problema computacional da escolha do caminho de interesse é conhecido como problema do maior caminho sem ciclos, que é um problema NP-Difícil (GAREY; JOHNSON, 1979). A Equação 3.1 descreve formalmente o problema, onde p é um caminho pertencente ao conjunto de todos os caminhos sem ciclos do grafo $\mathcal{P}(G)$.

$$\underset{p \in \mathcal{P}(G)}{\text{maximizar}} \sum_{e \in p} w(e) \tag{3.1}$$

Para solucionar este problema, utilizou-se a seguinte heurística: a partir do grafo obtido após o Treinamento e o passo da escolha dos nós inicial e final, cria-se um novo grafo G^I com os mesmos nós e arestas, mas com os pesos das arestas definidos como $w(e^I) = 1/w(e)$. A solução da Equação 3.2, que é o menor caminho em G^I, tenderá a

refletir o maior caminho em G.

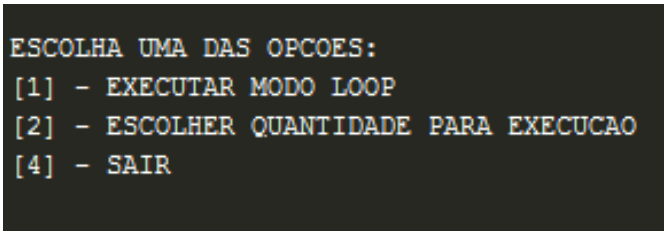
$$\underset{p^I \in \mathcal{P}(G^I)}{\text{minimizar}} \sum_{e^I \in p^I} w(e^I) \tag{3.2}$$

Esta heurística, embora não solucione o problema geral do maior caminho, se encaixa bem neste problema, dado que os caminhos obtidos a partir das demonstrações tendem a ser muito próximos e, dessa forma, a escolha pelo menor caminho evita considerar movimentos pouco repetidos (que terão peso alto no grafo G^I), já que estes podem ser considerados como erros, o que mitiga as imprecisões.

3.3.3 Execução

Após ter passado por todo o processo de Coleta e Aprendizagem, o sistema entra na fase final relacionada a execução da tarefa aprendida. Neste momento, de posse de um caminho obtido na etapa de Inferência, o robô possui uma sequência de estados selecionados que consiste de uma série de movimentos específicos e necessários, para que a tarefa possa ser **reproduzida** autonomamente. A Figura 20 mostra como é feita a interação entre o sistema e o usuário para decidir qual será o modo de execução da tarefa do manipulador robótico.

Figura 20 – Caixa de diálogo para modo de execução da tarefa



Fonte: Autor

O sistema PbD proposto possui duas opções para reprodução da atividade. Na primeira ele reproduz a atividade repetidas vezes, até que o usuário desligue o sistema manualmente, indicando que o robô deve parar. Este modo é ideal para realizar atividades por um longo período de tempo quando não se tem conhecimento do momento em que o robô deve parar especificamente.

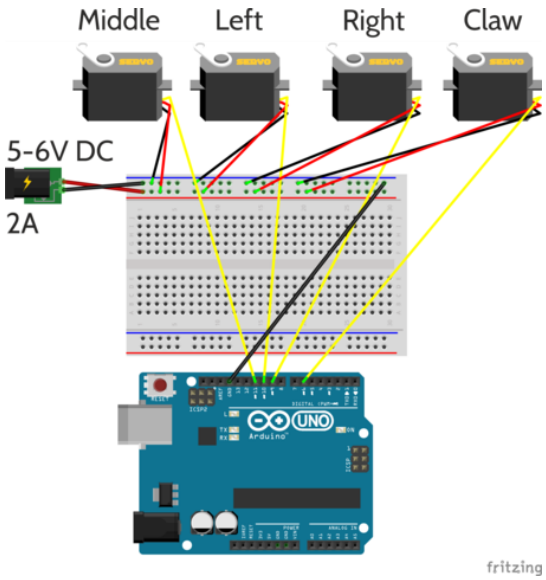
Na segunda opção o usuário deve informar ao robô a quantidade de vezes que a tarefa deve ser reproduzida. Ao contrário da primeira opção, esta é ideal para atividades em que o usuário sabe exatamente o momento em que o robô deve cessar suas atividades.

3.4 Implementação da proposta

Para implementação da proposta descrita utilizou-se um manipulador robótico com movimentos simplificados - sem utilidade para o meio industrial - denominado Me-Arm Robot Arm - Your Robot - V1.0 (ROBOTICS, 2016), com quatro servomotores do tipo *Micro Servo 9g SG90 TowerPro* responsáveis por fornecer os graus de liberdade ao robô. Este equipamento possui apenas 25 centímetros de tamanho e sua estrutura é montada em acrílico. A Figura 15 ilustra o robô utilizado.

Para fornecer os pulsos elétricos necessários para configuração dos servomotores do robô, uma placa Arduíno Uno foi utilizada. A Figura 21 apresenta os quatro servomotores. Três dos quatro servomotores atuam nos movimentos estruturais do robô e um servomotor é exclusivo para abertura e fechamento da garra (*Claw*). O primeiro (*Middle*), representa o grau de liberdade relacionado a base. Com esse servomotor o robô consegue mover-se tanto para esquerda quanto para direita, dentro dos limites previamente configurados. O segundo (*Left*) está relacionado à altura que o manipulador pode alcançar, desta forma pode-se elevar e descer o braço. O terceiro (*Rigth*) está relacionado a um grau de liberdade complementar, com ele o manipulador consegue um movimento para frente e para trás para auxiliar na manipulação de um objeto.

Figura 21 – Esquemático das ligações entre o Arduino e os Servomotores



Fonte: Retirado de (ROBOTICS, 2016)

O sistema PbD foi implementado em Python 2.7 com o auxílio da biblioteca python-igraph 0.7.0[1] para manipulação de grafos [2]. Este sistema é executado em um laptop HP COMPAQ Presario CQ43, sistema operacional Windows 7 de 32 bits, pro-

[1] http://igraph.org/python/
[2] Código disponível em: https://github.com/FelipeOliveiraTI/TCC_Felipe_2016.1.git

cessador Intel Pentium P6200 2.13GHz e memória RAM 4GB. O usuário interage com o sistema por meio do teclado e do joystick ligado via porta USB.

Durante o uso do robô, as fases do processo PbD são exibidas na tela para que o usuário possa acompanhar os modos de operação do robô. Os comandos para movimentação do robô são transmitidos do laptop para o Arduino via uma porta serial.

A Figura 22 mostra como foram integrados todos os componentes utilizados para realização do trabalho. Da esquerda para direita, a seta de cor preta representa a conexão entre o joystick e o laptop feito por meio de uma porta USB, o notebook ilustrado representa onde o sistema é executado, e por último a seta azul representa a conexão entre o computador e a placa Arduíno responsável por fornecer os pulsos para o giro dos servomotores.

Figura 22 – Arquitetura geral do sistema

Fonte: Autor

4 Avaliação e Resultados

Este capítulo descreve as avaliações da estratégia proposta e seus respectivos resultados a partir de sua implementação em um manipulador robótico.

4.1 Avaliação de custo de memória e processamento

O programa dispõe de duas fases fundamentais relativas ao processamento da funcionalidade de Aprendizagem da tarefa, a primeira se chama Treinamento e a segunda Inferência, foi observado um custo computacional diferente entre essas duas fases. O experimento realizado se baseou em repetir 30 vezes cada fase do processo, afim de obter o dados relativos ao tempo de execução de cada uma, com um conjunto de 10 demonstrações previamente fornecidas ao programa.

A Tabela 4 mostra os dados colhidos do experimento, onde a primeira coluna representa os dados relativos ao tempo médio de execução de cada fase em segundos, a segunda coluna se refere ao desvio padrão, as colunas terceira e quarta estão os dados referentes ao Mínimo e Máximo dos tempos respectivamente, e por último o Intervalo de confiança.

Tabela 4 – Resultados da análise do custo computacional

	Média	Desvio Padrão	Mínimo	Máximo	Intervalo de Confiança
Treinamento	2,07	0,007	2,04	2,07	(2,063 : 2,068)
Inferência	0,23	0,007	0,22	0,27	(0,227 : 0,232)
Total	2,30	0,011	2,27	2,34	(2,291 : 2,299)

O tempo total médio observado apenas para execução da fase de Treinamento, foi de pouco mais de 2 segundos, comparando com tempo total do processo de programação do robô por demonstração, que é em média de 15 minutos para cada conjunto de dez demonstrações, essa média observada na fase de treinamento é mínima, contribuindo significativamente com a diminuição do tempo que o robô deve ficar parado aguardando o treinamento.

Para a fase de Treinamento, o custo computacional é linear com o número dos vértices, já para a fase de Inferência, o custo computacional é quadrático para o número de vértices do grafo, provocado pela execução do Algoritmo de Dijkstra. Contudo, podemos observar que teoricamente, o tempo relativo a fase de Inferência deveria ser superior em relação à fase de Treinamento, pois um custo computacional quadrático tende a ser mais elevado que um linear, porém, podemos observar na tabela, que

os resultados são invertidos, ou seja, os custos relativos a fase de Treinamento são superiores aos da fase de Inferência.

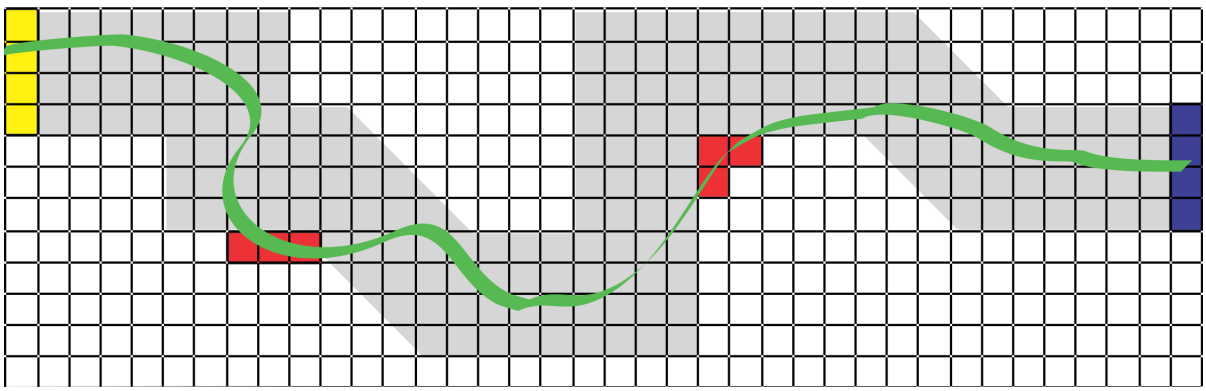
Este fato pode ser explicado, devido a maneira que o sistema foi desenvolvido, pois para implementação da fase de Treinamento, foi utilizado a biblioteca Python-igraph, esta é um wrapper para o igraph do C/C++, enquanto para a fase de Inferência, uma função responsável por executar o algoritmo foi chamada diretamente, executando em uma biblioteca em C.

4.2 Avaliação do aumento de demonstrações

O objetivo desta avaliação é verificar como o aumento do número de demonstrações impacta na diminuição das imprecisões da tarefa realizada. Desta forma, pode-se indicar se a abordagem PbD proposta neste trabalho é suficiente para fazer com que um robô seja capaz de aprender como executar uma tarefa dentro dos pressupostos determinados no Capítulo 3.

A tarefa a ser executada compreende em controlar o robô para descrever um circuito pré-definido em um plano, de modo que o manipulador robótico seja capaz de traçar este caminho com um marcador de texto comum acoplado à sua garra. A Figura 23 mostra o circuito utilizado. Os quadrantes marcados de cinza representam o caminho por onde o usuário deve manipular o robô, simulando a pintura sem o marcador de texto. O resultado do treino do robô é a linha do marcador de texto (na cor verde). Os quadrantes marcados de azul e amarelo representam os pontos de início e fim da tarefa, respectivamente. Os quadrantes marcados de vermelho representam os erros realizados pelo robô ao pintar um quadrado fora do caminho específico. Na representação em questão, foram calculados 6 erros. Esses erros foram tomados como métrica de avaliação.

Figura 23 – Representação do circuito a ser seguido na tarefa.



Fonte: Autor

4.2.1 Método do experimento

O experimento foi realizado com 20 participantes, com cada um deles apresentando demonstrações da mesma tarefa ao robô. Como mostrado na Figura 23, o usuário, utilizando um joystick, manipula o robô para seguir o caminho em cinza impresso em papel ajustado a uma base fixa utilizada em todos os testes. Cada usuário executou um número de demonstrações diferente dentre quatro níveis: 3, 5, 7 e 10 demonstrações. Os usuários não sabiam o número de demonstrações que fariam, a priori, apenas o condutor do experimento tinha essa informação, e este solicitava mais demonstrações até que fossem atingidos o total previsto para o teste. Ao fim das demonstrações do usuário, o algoritmo era executado e o resultado do robô (a trilha desenhada no papel) era armazenada para computação dos resultados. Note-se que ao final do experimento tem-se exatamente 5 usuários para cada número de demonstrações.

Na escolha dos participantes não foram consideradas características dos usuários como sexo, escolaridade ou idade. Porém, para nivelar os usuários no uso do joystick, cada usuário teve cinco oportunidades para realizar a tarefa, antes da execução própria do experimento. Isto foi necessário, porque a habilidade do usuário influencia nestes resultados.

4.2.2 Resultados Obtidos

Os resultados das demonstrações de cada usuário em cada categoria podem ser vistos nas figuras do Apêndice A. A Tabela 5 sumariza estes resultados apresentando as estatísticas básicas dos erros em cada categoria. A partir da segunda coluna têm-se, respectivamente: o total dos erros computados, a média de erros, o desvio padrão desta média, os valores mínimos e máximos e, por último, na sétima coluna, estão os valores referentes ao Intervalo de Confiança, considerando 95% de confiança.

Tabela 5 – Estatísticas dos erros computados referentes a cada categoria do experimento

Categoria	Total	Média	Desvio Padrão	Mínima	Máximo	Intervalo de Confiança
3	50	10,0	6,60	3	18	(4,01 ; 15,99)
5	22	4,40	3,05	0	8	(1,63 ; 7.17)
7	26	5,20	4,27	1	10	(1,33 ; 9,07)
10	12	3,67	3,78	0	9	(0,23 ; 7,10)

Deve-se observar, em primeiro lugar, a alta variabilidade nos resultados. O coeficiente de variação é de mais de 60% em todas as categorias, chegando a ultrapassar 100% no caso com 10 demonstrações. Isto ocorre devido a discrepância na habilidade dos usuários. O resultado do aprendizado do robô para um dos usuários que

fez apenas 3 demonstrações, por exemplo, alcançou apenas 3 erros. Enquanto que o aprendizado do robô para um outro usuário com 10 demonstrações foi de 9 erros. Em outras palavras, um usuário com apenas 3 demonstrações ensinou ao robô um caminho melhor do que aquele obtido por um usuário que teve a chance de fazer mais demonstrações.

Em segundo lugar, pode-se notar uma melhora significativa no modo de execução da tarefa realizada pelo robô, provocado pelo aumento no número de demonstrações fornecidas pelo usuário ao sistema. A diferença mais considerável foi em relação as categorias com 3 e 10 demonstrações.

Devido à pequena quantidade de amostras (5 por categoria) utilizadas no experimento, não é possível afirmar com certeza estatística que houve redução na média. Os intervalos de confiança sobrepostos reforçam esta observação. Contudo, pode-se observar que as trajetórias descritas pelo robô treinado com apenas 3 demonstrações obteve uma média de 10 erros computados, enquanto que para categoria 10, foi observado uma média de 3,6 de erros, refletindo uma diminuição de 64% entre as categorias.

Ainda para a categoria 3, onde os cinco usuários forneceram cada um, três demonstrações de como executar a tarefa, foi computado um total de 50 erros. Por outro lado, os circuitos obtidos pelos aprendizados com 10 demonstrações apresentam 12 erros computados no total, traduzindo uma melhora de 76% na execução da tarefa, quando comparado com a categoria 3.

Ao longo do experimento, observou-se ainda o tempo de execução das demonstrações de cada participante. E o tempo de treino máximo foi de 15 minutos. Além disso, outro dado interessante, é que o número de vértices que compõem o caminho inferido no processo de Aprendizagem, foi em média de 28 nós para um conjunto de 10 demonstrações fornecidas. Assim, o robô precisa executar essa quantidade movimentos para reproduzir a atividade pretendida. Podemos observar que essa quantidade de movimentos está relacionada a esta tarefa em particular, já que esta quantidade de movimentos depende diretamente da tarefa que o robô deve executar.

5 Conclusão

O presente trabalho apresentou e avaliou uma estratégia para a programação por demonstração de robôs baseada em teoria dos grafos. A técnica se aplica à programação de robôs para tarefas repetitivas na indústria e assume que o usuário demonstra a tarefa que o robô deve executar através de um joystick, não necessitando que o robô faça uso de sensores para capturar o movimento do usuário ou para certificar-se de sua própria posição no ambiente.

A programação de tarefas complexas por meio de *teach pendants* pode se tornar morosa devido à imprecisões introduzidas durante a fase de programação do robô. Assim a estratégia proposta neste trabalho permite que o robô execute tarefas mesmo na presença de imperfeições que o próprio programador possa cometer durante a execução da tarefa. Para mitigar o efeito destes erros, o usuário realiza diversas demonstrações de uma mesma tarefa para que o robô consiga inferir a melhor forma de executá-la a partir dos movimentos mais frequentes nas demonstrações.

Para composição da base de conhecimento do robô, assume-se que esse possui atuadores discretos. Desta maneira é possível modelar os graus de liberdade dos atuadores do robô em nós de um grafo de estados. Adicionalmente, as arestas deste grafo são criadas de acordo com as limitações de movimento relativas à própria estrutura do robô. O aprendizado ocorre quando, a cada demonstração feita pelo usuário, as arestas relacionadas aos movimentos são reforçadas. Assim, ao encontrar o maior caminho direto, tem-se o conjunto de movimentos desejados para que o robô execute a tarefa. Dada a complexidade do problema, utiliza-se uma heurística para solucioná-lo em tempo viável.

Aplicou-se a estratégia proposta a um manipulador robótico simples composto por três servomotores para a realização da tarefa de seguir com uma caneta um circuito desenhado em um papel. Para verificar a eficácia da estratégia e os ganhos obtidos com o aumento do número de demonstrações, foram realizados experimentos com um total de 20 participantes. Cada participante executou um número diferente de demonstrações da mesma tarefa e se verificou que a quantidade de erros média diminuiu em 64% com o aumento do número de demonstrações de 3 para 10. Este experimento também demonstrou que o tempo de treino foi no máximo de 15 minutos para um participante que realizou 10 demonstrações. Desta forma, é possível observar que a técnica proposta tem potencial para simplificar o treinamento de um robô pelo uso de um mero joystick, sem necessidade de conhecimento de programação baseada em texto, além de potencialmente reduzir o tempo com o robô parado ao evitar que o usuário tenha

que reprogramar o robô no caso de imprecisões.

5.1 Limitações da proposta

Para o sistema proposto funcionar, o robô precisa possuir motores discretos, ou seja, eles precisam ter informações pertinentes aos graus de configurações dos ângulos de todos os motores, sem isto, fica impossível gerar a base de conhecimento do sistema baseado em grafo. Outra limitação é o fato do robô precisar ter um computador trabalhando em conjunto diretamente, pois nele fica a CPU para realizar a execução do programa bem como a unidade de memória responsável por mantê-lo.

5.2 Trabalhos futuros

Como trabalhos futuros espera-se investigar o uso desta técnica em tipos diferentes de robôs para a execução de tarefas mais complexas possivelmente comparando-a ao uso de *teach pendants*. Espera-se ainda investigar heurísticas alternativas que ampliem as possibilidade de aplicação da estratégia.

Uma outra proposta seria a implantação de uma placa Raspberry Pi no robô, para que o programa desenvolvido execute diretamente nele, sem a necessidade de um computador trabalhando em conjunto, integração de um novo joystick mais sofisticado para tornar o controle do robô mais intuitivo, além de aumentar o número de comandos enviados para o robô diretamente do controle.

Referências

BIGGS, G.; MACDONALD, B. A survey of robot programming systems. In: *Proceedings of the Australasian conference on robotics and automation*. [s.n.], 2003. p. 1–3. Disponível em: <http://ftp.societyofrobots.com/robottheory/Survey_of_Robot_Programming_Systems.pdf>. Citado 6 vezes nas páginas 13, 14, 17, 18, 19 e 20.

BILLARD, A. et al. Robot programming by demonstration. In: *Springer handbook of robotics*. Springer, 2008. p. 1371–1394. Disponível em: <http://link.springer.com/10.1007/978-3-540-30301-5_60>. Citado 2 vezes nas páginas 22 e 23.

BOUCHARD, S. *The Evolution of the Robotic Teach Pendant*. 2011. Disponível em: <http://blog.robotiq.com/bid/29774/The-Evolution-of-the-Robotic-Teach-Pendant>. Citado na página 15.

CARVALHO, M. Teoria dos Grafos, uma introdução. *Centro*, 2005. Citado 3 vezes nas páginas 25, 26 e 30.

CAVALCANTE, Z. V.; SILVA, M. L. S. da. A Importância da Revolução Industrial no mundo da tecnologia. 2011. Disponível em: <http://rgomes.yolasite.com/resources/A%20IMPORT%C3%82NCIA%20DA%20REVOLU%C3%87%C3%83O%20INDUSTRIAL%20NO%20MUNDO%20DA%20TECNOLOGIA.pdf>. Citado na página 12.

CYPHER, A.; HALBERT, D. C. *Watch what I Do: Programming by Demonstration*. [S.l.]: MIT Press, 1993. ISBN 978-0-262-03213-1. Citado 2 vezes nas páginas 21 e 22.

EKVALL, S. Grasp recognition for programming by demonstration. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005. p. 748–753. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1570207>. Citado 3 vezes nas páginas 14, 24 e 25.

EKVALL, S. Robot learning from demonstration: a task-level planning approach. *International Journal of Advanced Robotic Systems*, v. 5, n. 3, p. 223–234, 2008. Disponível em: <http://cdn.intechopen.com/pdfs/4292.pdf>. Citado 2 vezes nas páginas 24 e 25.

ELECTRONICS, R. R. *Rapid Electronics and Education Blog: Screwdriver + Enthusiasm = MeArm*. 2016. Disponível em: <http://rapideleceducationblog.blogspot.com.br/2016/01/screwdriver-enthusiasm-mearm.html>. Citado na página 33.

FEOFILOFF, P. Exercícios de Teoria dos Grafos. 2012. Disponível em: <http://arquivoescolar.org/bitstream/arquivo-e/193/1/ETG.pdf>. Citado na página 26.

FORBES, M. et al. Robot programming by demonstration with crowdsourced action fixes. In: *Second AAAI Conference on Human Computation and Crowdsourcing*. [s.n.], 2014. Disponível em: <http://www.aaai.org/ocs/index.php/HCOMP/HCOMP14/paper/view/8975>. Citado 3 vezes nas páginas 14, 24 e 25.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN 0-7167-1044-7. Citado na página 38.

GENERIC USB Gamepad / Joystick - CeX (UK): - Buy, Sell, Donate. 2016. Disponível em: <https://uk.webuy.com/product.php?sku=SACCJG50A#.V3fTLXUrJkU>. Citado na página 34.

KHAN Academy. 2016. Disponível em: <http://pt.khanacademy.org>. Citado na página 31.

MARKOFF, J. New Wave of Deft Robots Is Changing Global Industry. *The New York Times*, ago. 2012. ISSN 0362-4331. Disponível em: <http://www.nytimes.com/2012/08/19/business/new-wave-of-adept-robots-is-changing-global-industry.html>. Citado na página 12.

MOLLARD, Y. et al. Robot programming from demonstration, feedback and transfer. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015. p. 1825–1831. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7353615>. Citado 2 vezes nas páginas 24 e 25.

ROBOTICS, M. *MeArm Robotics - Instructions*. 2016. Disponível em: <https://www.mearm.com/pages/instructions>. Citado na página 40.

RODRIGUES, F. A. Caracterização, classificação e análise de redes complexas. *Instituto de Fısica de Sao Carlos, Universidade de Sao Paulo, Sao Carlos*, 2007. Citado na página 26.

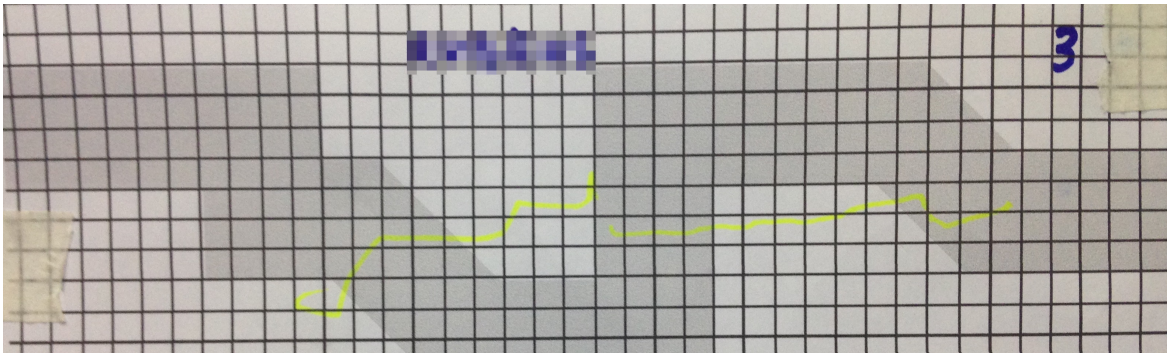
TEIXEIRA, C. E. P. *Ensinamento Rápido de Manipuladores Industriais*. Tese (Doutorado) — Universidade do Porto, 2009. Citado 2 vezes nas páginas 14 e 15.

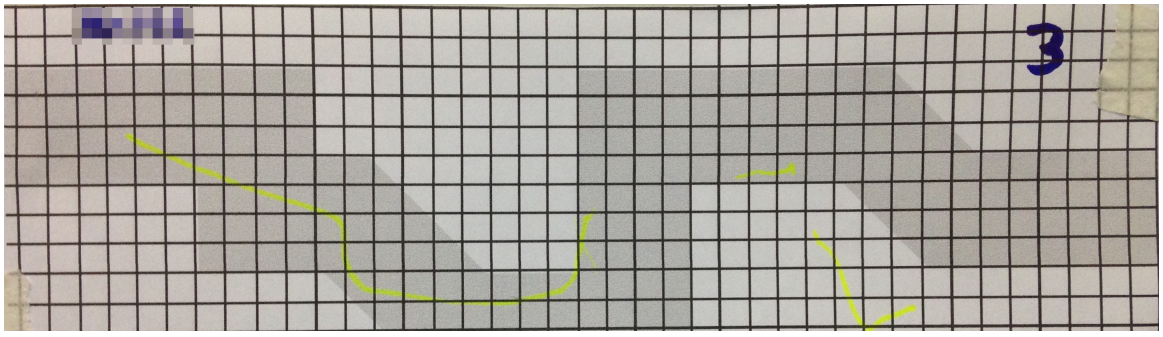
WAHL, F. M.; THOMAS, U. Robot programming-from simple moves to complex robot tasks. *Institute for Robotics and Process Control, Technical University of Brawnschweig*, 2002. Disponível em: <http://www2.cs.siu.edu/~hexmoor/classes/CS404-S10/Wahl.pdf>. Citado na página 14.

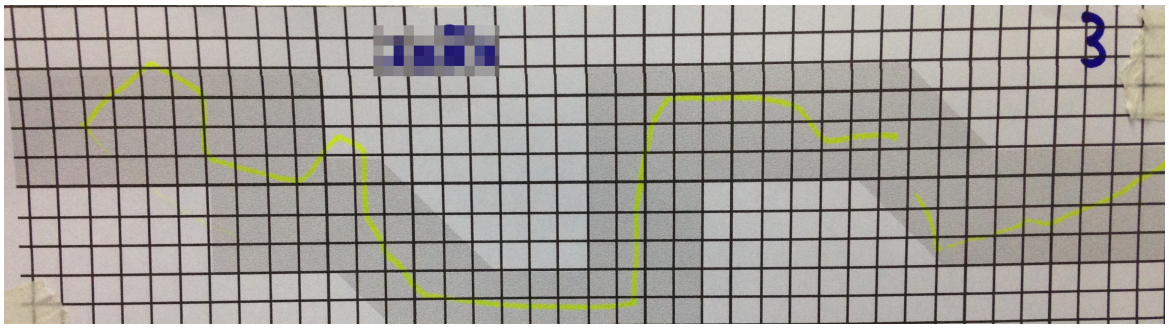
ZöLLNER, R. Programming by demonstration: dual-arm manipulation tasks for humanoid robots. In: *IROS*. [s.n.], 2004. p. 479–484. Disponível em: <https://www.researchgate.net/profile/Ruediger_Dillmann/publication/4121774_Programming_by_Demonstration_Dual-Arm_Manipulation_Tasks_for_Humanoid_Robots/links/0046351c964f63c967000000.pdf>. Citado 2 vezes nas páginas 23 e 25.

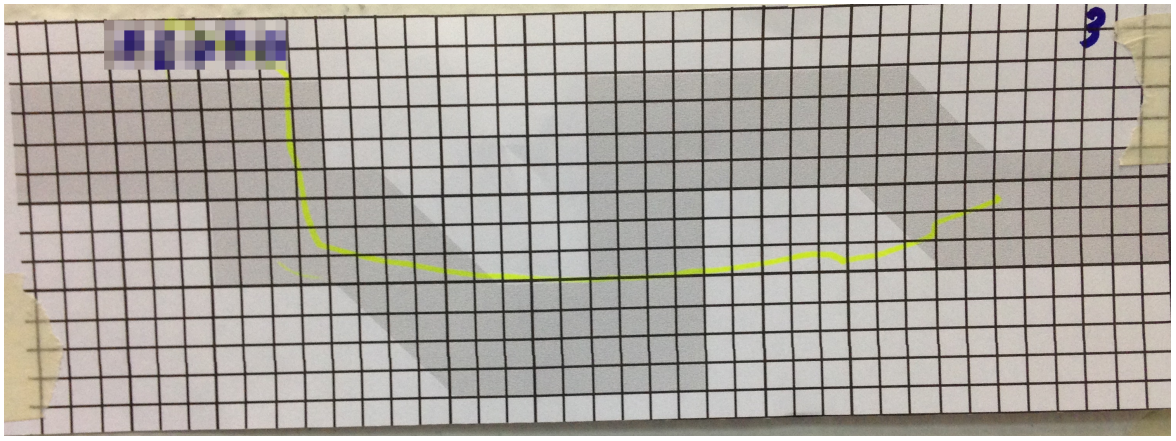
A Apêndice

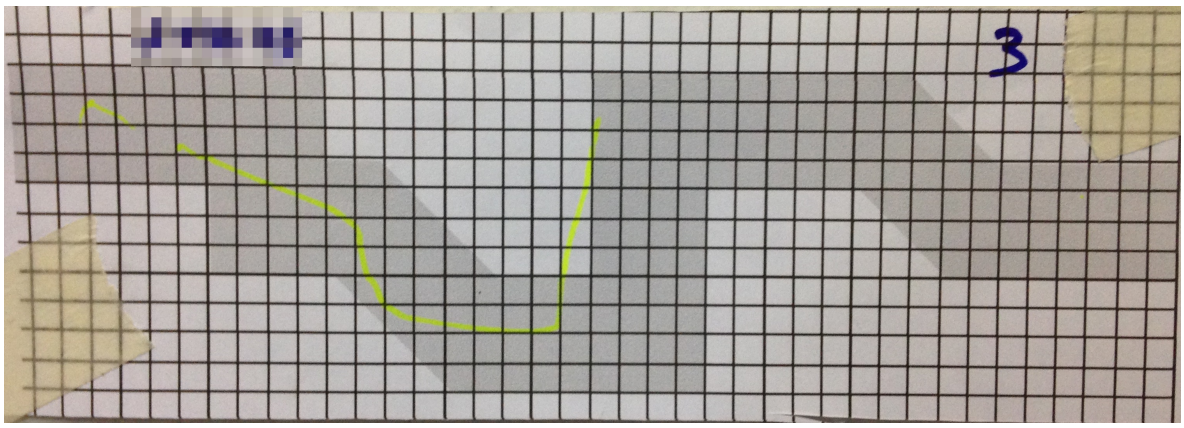
A.1 Testes da Categoria 3



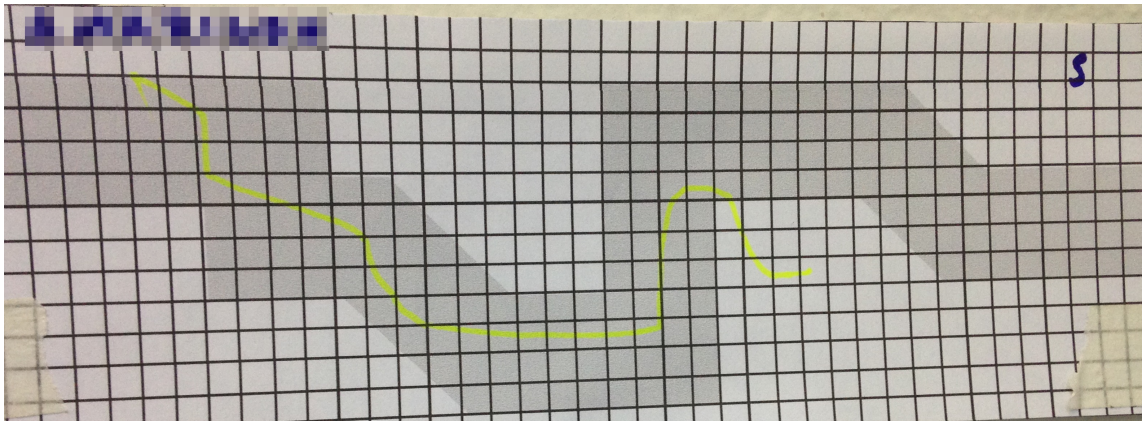


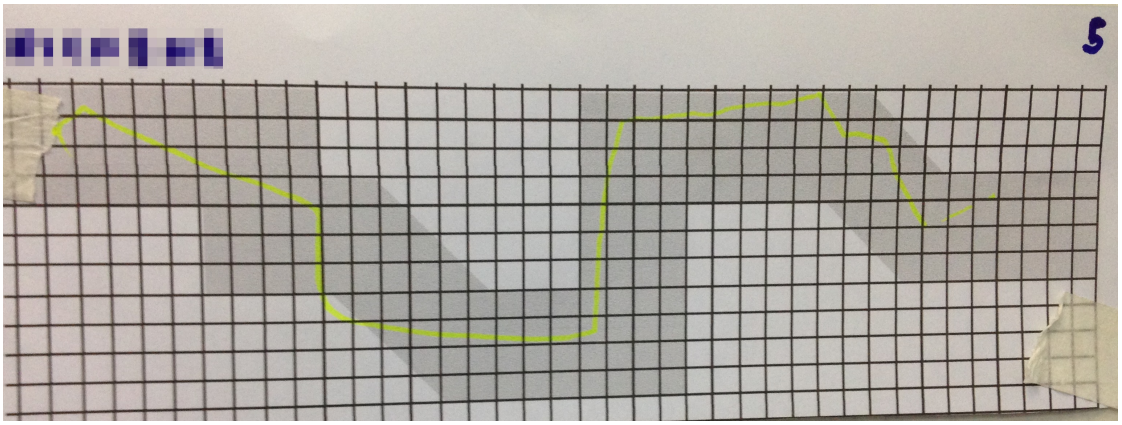


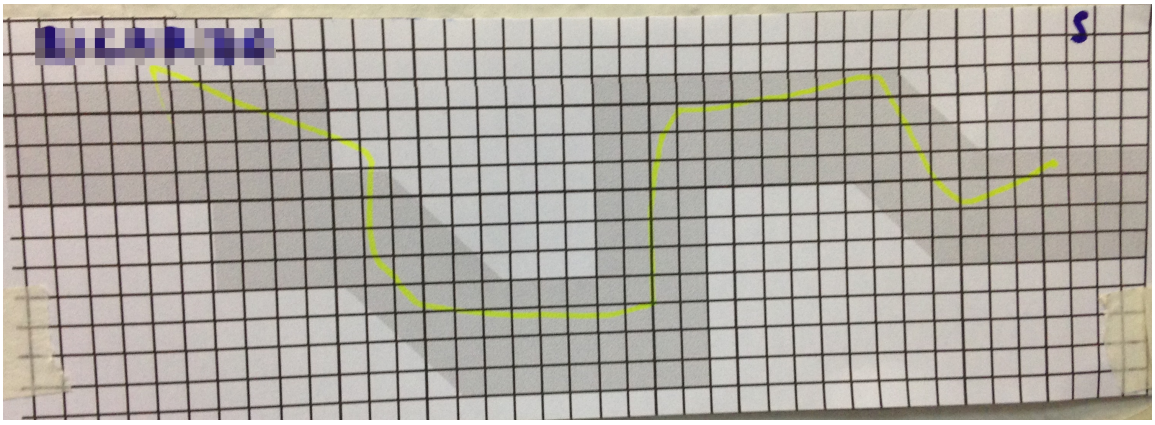


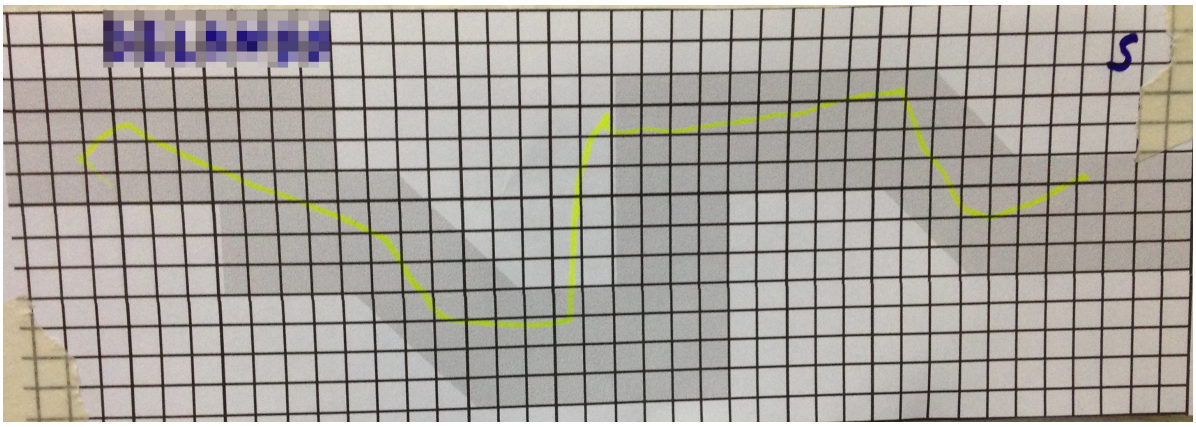


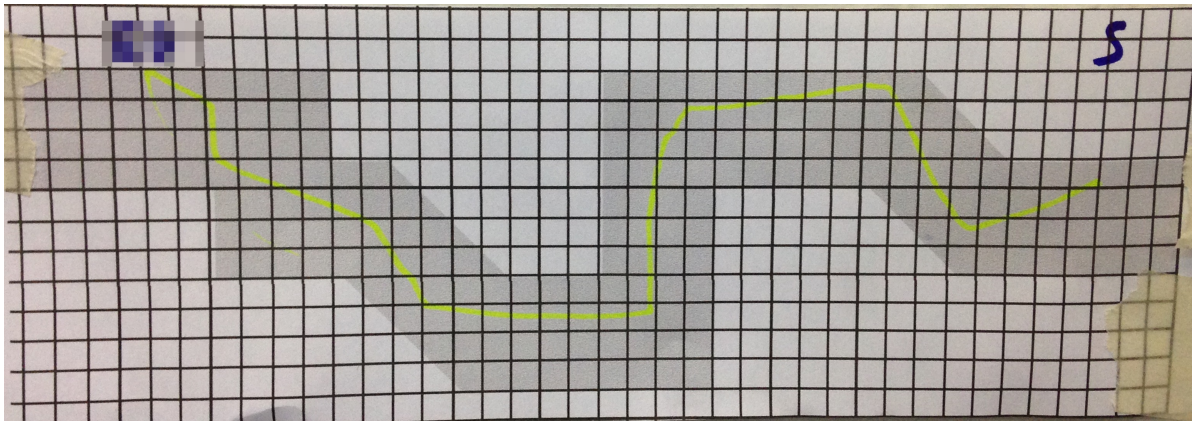
A.2 Testes da Categoria 5



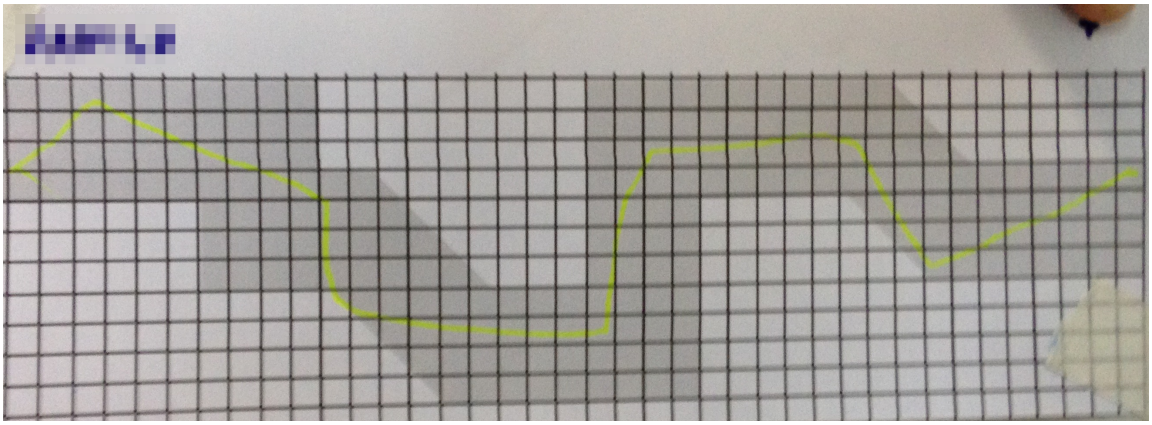


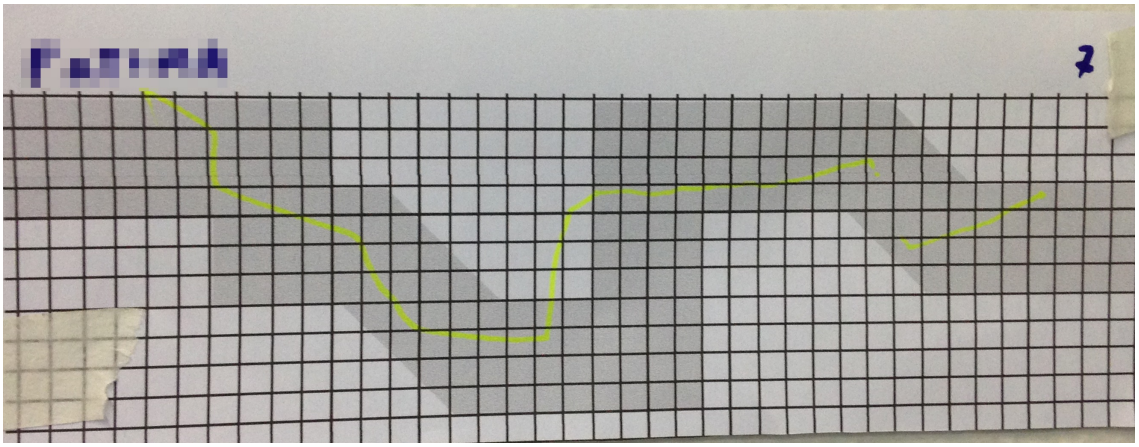


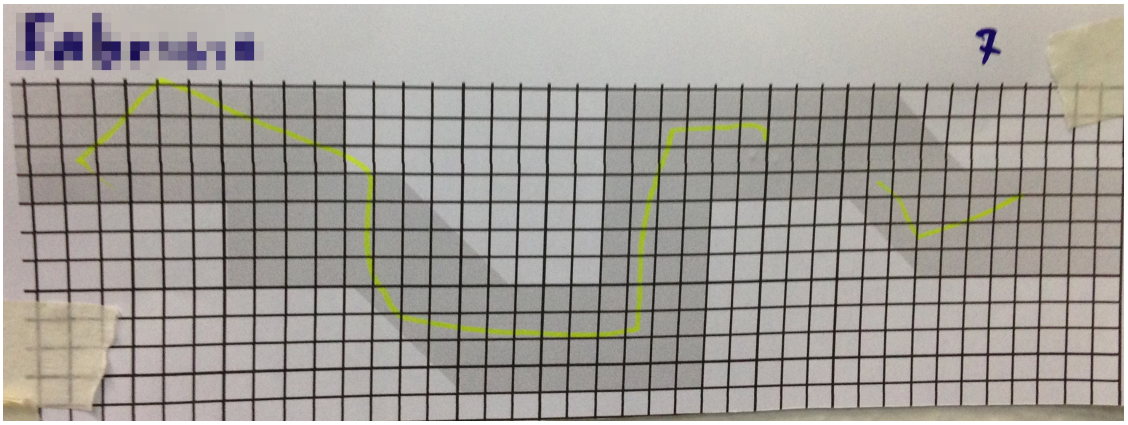


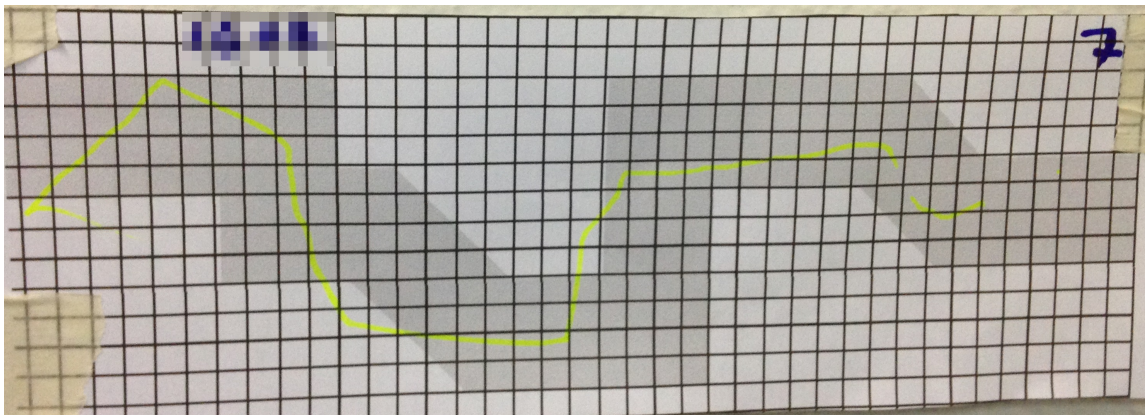


A.3 Testes da Categoria 7

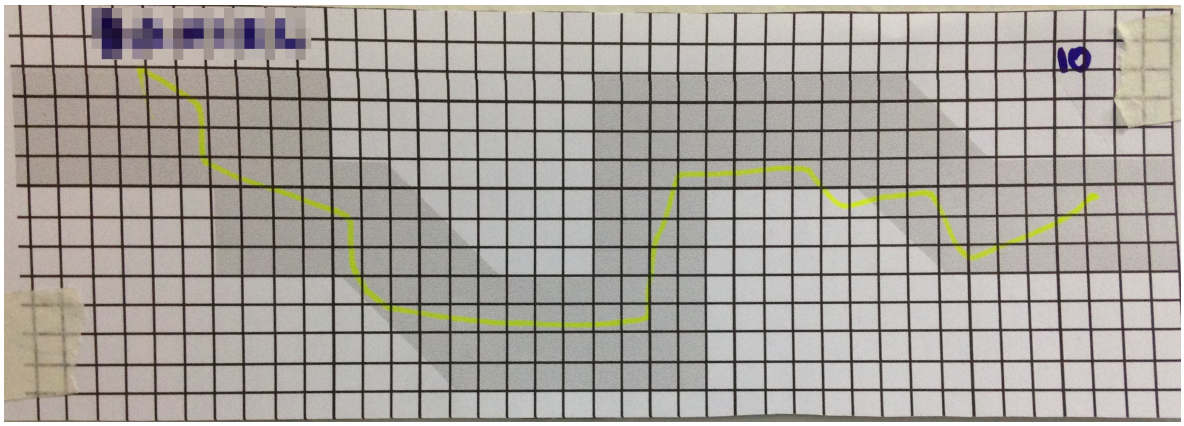


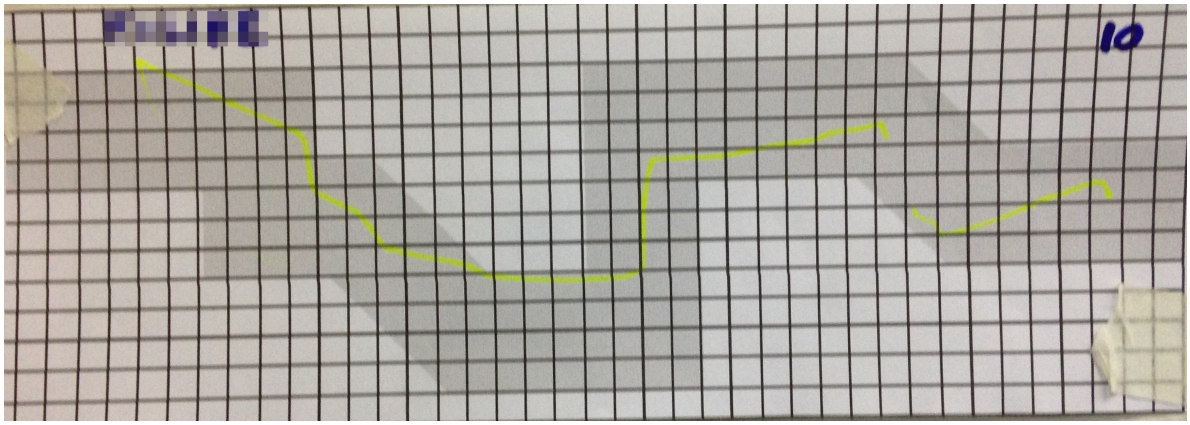


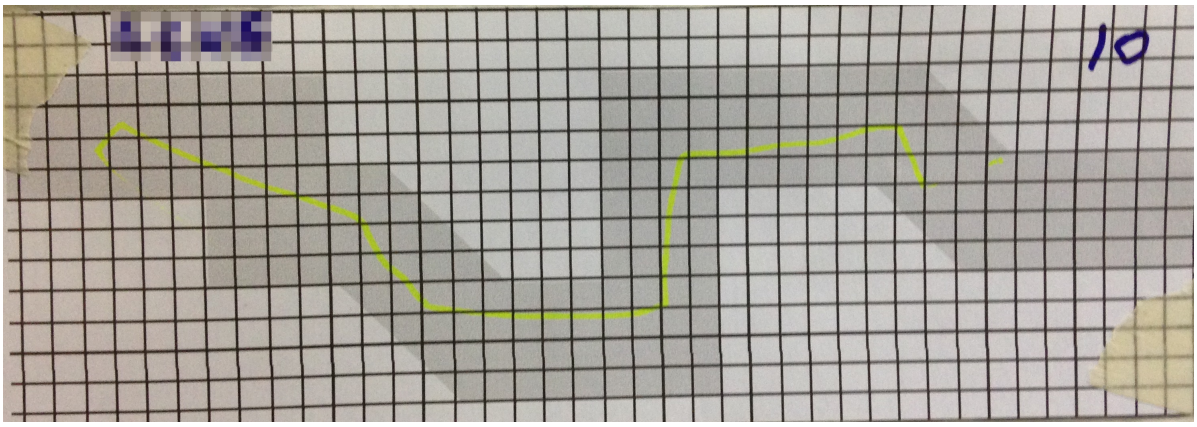


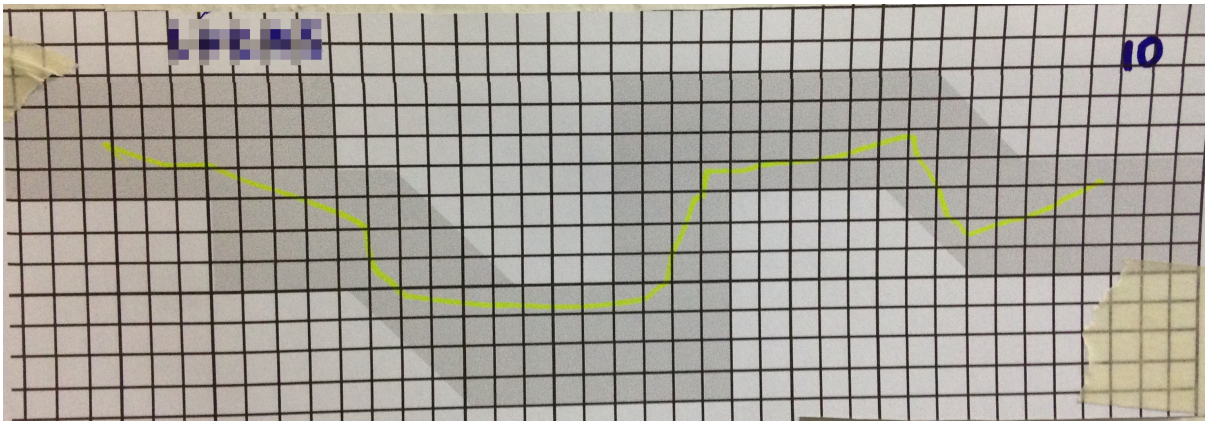


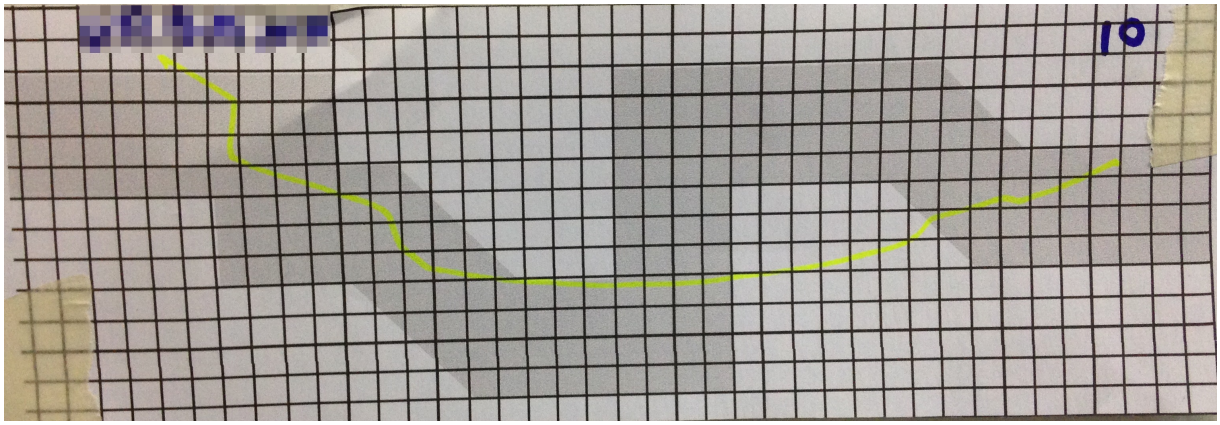
A.4 Testes da Categoria 10




10


10


10


10