UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAFAEL GARCIA

Generalização do Comportamento de Robôs usando Regressão de Manifolds

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. João Luiz Dihl Comba
Co-orientador: Prof. Dr. Bruno Castro da Silva

Porto Alegre
2016

RESUMO

Algoritmos de aprendizado de máquina podem ser muito custosos computacionalmente, em particular aqueles utilizados em tarefas na área de robótica, especialmente quando um número muito grande de tarefas de treinamento similares devem ser efetuadas. Encontrar formas de generalizar o comportamento de um robô é desejável pois permite que ele seja capaz de utilizar o conhecimento prévio de tarefas anteriores no aprendizado de novas tarefas. A solução encontrada por algoritmos de aprendizado de máquina a um vetor de N parâmetros ajustáveis, usados para definir a execução da tarefa. Em problemas reais, o valor de N pode ser muito alto, o que dificulta a análise destes conjuntos de dados. Este trabalho utiliza vários métodos de visualização científica para analizar tais dados de treinamento multidimensionais e estudar suas propriedades, tais como a presença de sub-regiões disjuntas no espaço de alta dimensão, a fim de guiar o desenvolvimento de um algoritmo capaz de promover a generalização do comportamento do robô a partir de poucas amostras de treinamento. Em particular, nós apresentamos um algoritmo que busca identificar o número de manifolds disjuntos formados pelos dados de treinamento e demonstramos como utilizar um método de clusterização baseado na otimização de matrizes esparsas para encontrar tais manifolds e permitir que o comportamento deles seja aprendido de maneira mais eficiente através de técnicas de regressões não-paramétricas.

Palavras-chave: Visualização e análise de dados multidimensionais. identificação de subespaços. clusterização de manifolds. regressões não-paramétricas. aprendizado de maquina. robótica.

Generalizing Robots Behavior using Manifolds Regression

ABSTRACT

Machine learning algorithms can be computationally costly, particularly those used in robotics tasks and specially when a great number of similar training tasks must be performed. Find ways to generalize the robot behavior is desirable because it allows the utilization of previous knowledge taken from already trained tasks to learn acceptable behaviors for new tasks. Solutions of machine learning problems are usually modeled as vectors of N adjustable parameters, used to define tasks executions. In real world problems, the value of N can be very big, making it difficult to analyse datasets formed by collections of these solutions. This work uses several methods of scientific visualization to analyse these multidimensional training datasets and to study their properties, such as the presence of disjoint subregions in the high dimensional space, in order to guide the development of an algorithm capable to promote the generalization of the robot's behavior from few training samples. Particularly, we present an algorithm that seeks to identify the number of disjoint manifolds formed by the training dataset and show how to use a clustering method based on sparse matrices optimization to find these manifolds and to learn their behavior through non-parametric regressions techniques.

Keywords: multidimensional data visualization and analysis, subspace finding, manifold clustering, non-parametric regression, machine learning, robotics.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DMP	Dynamic Movement Primitives
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
PCA	Principal Components Analysis
ISOMAP	Isometric Mapping
LLE	Locally Linear Embedding
t-SNE	t-Distributed Stochastic Neighbor Embedding
SMCE	Sparse Manifold Clustering and Embedding

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de algoritmos de aprendizado de máquina e a sua aplicação em problemas específicos de aprendizado ou otimização é um processo complexo que geralmente exige que o usuário utilize diversas formas de análise para que seja possível identificar erros, anomalias ou resultados não-ótimos nas soluções encontradas, de forma que essas informações possam ser usadas para aperfeiçoar os resultados do algoritmo. Para que esta análise seja possível, técnicas de visualização de dados são imprensidíveis, pois elas permitem uma melhor compreensão do comportamento das soluções encontradas, permitindo o reconhecimento de padrões, áreas de interesse e inconstâncias, entre outras informações relevantes, tornando-se uma ferramenta essencial para que o usuário possa fazer alterações visando o aperfeiçoamento do algoritmo em questão, como ajuste de meta-parâmetros e modificações nos dados de treinamento, caso existam.

Neste trabalho, é introduzido um projeto prévio que busca fazer o aprendendizado de movimentos que um braço mecânico deve realizar para concluir uma determinada tarefa, que no caso consiste em alcançar um determinado ponto alvo sobre uma circunferência em um plano, onde cada solução é modelada como um conjunto de seis parâmetros, denominado política. Em seguida, são listadas os métodos de análise exploratória realizados no conjunto destas soluções, especialmente por meio de técnicas de visualização de dados. Por se tratar de um conjunto de dados multidimensional, onde cada política pode ser vista como um ponto no espaço hexadimensional, torna-se necessário a utilização de técnicas que facilitem a análise para esse tipo de dado, como métodos de redução de dimensionalidade. Além disso, são apresentadas algumas conclusões tiradas dessa análise e soluções para alguns dos problemas encontrados.

A medida que os resultados desta análise foram aparecendo, o foco do trabalho evoluiu para desenvolver um método de generalização do comportamento do robô através de habilidades parametrizadas [Silva, Konidaris and Barto 2012], fazendo com que seja possível, a partir de um conjunto das política previamente aprendidas para um determinado número de tarefas, prever os parâmetros de novas tarefas sem ter que treiná-las através do modelo ou simulador do robô. Essa generalização é realizada através da clusterização dos manifolds existentes no conjunto de políticas, com o objetivo de identificar agrupamentos que representem regiões contínuas sobre o manifold, de forma que tais agrupamentos possam ser modelados através de técnicas de regressão, para que não seja necessário realizar o aprendizado completo para todas as posições alvo desejadas, o que

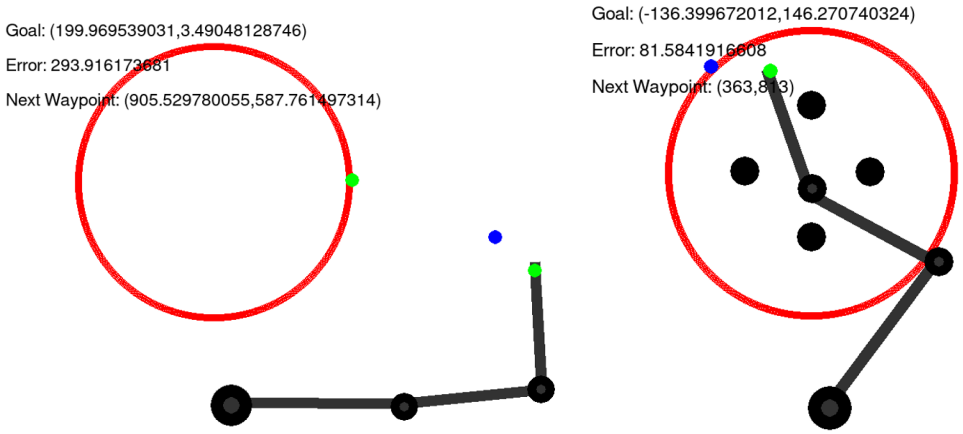
seria computacionalmente muito custoso, mesmo quando um simulador ou modelo está disponível, o que se tornaria muito pior, até impossível na prática, caso o treinamento tivesse que aconter em um robô físico, devido ao desgaste causado no aparelho. O trabalho é então concluído com a apresentação do algoritmo desenvolvido e os métodos de clusterização e regressão utilizados. O método de generalização foi desenvolvido com o objetivo de não ser restrito a esta aplicação, podendo ser adaptado no futuro à diferentes técnicas de aprendizado.

2 O PROBLEMA DE MOVIMENTAÇÃO DE BRAÇOS ROBÓTICOS

Uma das formas mais utilizadas para fazer a modelagem de trajetórias é o método DMP (*Dynamic Movement Primitives*) [Schaal], onde cada modelo de trajetória é representado de forma genérica por uma sequência de parâmetros de uma equação diferencial não-linear que, em conjunto com um ponto de inicio e de fim, definem a trajetória a ser percorrida pelo robô para completar determinada tarefa. O objetivo da DMP é permitir uma representação compacta, com poucos parâmetros, de trajetórias realizadas em algum espaço arbitrário de ações. Tais trajetórias podem estar modeladas de diversas formas, como por exemplo: sequência de ângulos das rotações realizadas pelas juntas de um braço robótico, trajetória de torques de uma junta do robô, sequência de posições *xy* percorridas, entre outras. Dentro do contexto deste trabalho, uma tarefa é uma posição alvo à ser alcançada pelo braço mecânico e um conjunto de seis parâmetros, denominado política, define uma sequência de rotações, em ângulos, das dobras do braço mecânico que o levam até o alvo.

A Figura 2.1 mostra a interface do simulador usado para treinar e executar os movimentos do robô. Na imagem da esquerda, encontra-se o braço mecânico formado por uma base rotativa e outros dois pontos intermediários de rotação. O objetivo do braço mecânico é executar um movimento que faça a sua mão, representada pelo ponto verde na ponta do braço, chegar o mais próximo possível do ponto alvo dentro da circunferência, também representado por um círculo verde. O ponto azul é um ponto de controle, chamado *waypoint*, que serve para guiar o trajetória do robô até o ponto alvo. Baseando-se em dois parâmetros, a política e o ponto alvo, a DMP calcula uma sequência de pontos de controle que, posteriormente, serão utilizados no calculo da trajetória efetivamente per-

Figura 2.1: Simulador do Braço Mecânico - Configurações com e sem Obstáculos



corrida, usando um controlador Proporcional-Integral (PI). As legendas no canto superior esquerdo indicam a posição x, y do ponto alvo (*Goal*), a distância atual do braço para o ponto alvo (*Error*) e a posição x, y do próximo *waypoint*. A imagem da direita mostra o mesmo simulador, porém com a adição de quatro obstáculos dentro da circunferência (os quatro círculos pretos formando os vértices de um losango). A presença de obstáculos pode dificultar o treinamento, já que irá restringir o número de movimentos possíveis que levam o robô à uma determinada posição. Neste trabalho, qualquer mensão à uma configuração com obstáculos estará se referindo a esta configuração em forma de losango.

Baseando-se em dois parâmetros, a política e o ponto alvo, a DMP calcula um conjunto de pontos de controle (*waypoints*), que servirão para guiar o movimento do braço mecânico, agindo como se fossem posições alvo intermediárias. A movimentação gera, a seguir, uma sequência de posições de fato percorridas, o que forma a trajetória realizada pelo braço mecânico até a posição alvo.

O treinamento do robô consiste em encontrar uma política, os parâmetros de uma DMP, a qual codifique uma trajetória capaz de resolver com sucesso uma determinada tarefa, com o menor erro de performance possível, calculado pela distância, em centímetros, entre o ponto alvo e a mão do robô ao final do movimento. Isso é feito através de um método de otimização denominado CMA-ES (*Covariance Matrix Adaptation Evolution Strategy*), que promove um algoritmo evolutivo onde, a cada iteração, novos indivíduos são gerados de maneira estocástica baseados em variações de indivíduos pais previamente existentes. Os indivíduos considerados mais eficientes por uma função de custo f tornam-se então pais na próxima iteração [Auger and Hansen 2012].

Normalmente, o treinamento de uma tarefa deve ser inicializado com uma política aleatória como semente que, durante a otimização, deve convergir para uma solução ótima local. Contudo, baseando-se na suposição de que tarefas parecidas exigirão políticas parecidas, é possível acelerar o treinamento inicializando-o com a solução encontrada para um ponto alvo próximo, caso exista e possa ser facilmente identificada. Ao início deste trabalho, o treinamento era realizado para todos os 360 pontos relativos aos ângulos inteiros ao longo da circunferência. A tarefa do ângulo 0 era então inicializada com uma semente aleatória enquanto todos os outros ângulos θ eram inicializados com a política encontrada para o ângulo $\theta - 1$, a menos que o erro de performance da tarefa $\theta - 1$ tenha sido mais alto que um limiar $l = 40$, escolhido empiricamente. Neste caso, o ângulo θ também era inicializado com uma semente aleatória. O procedimento é descrito no Algoritmo 2.1.

Algorithm 2.1 Treinamento de tarefas ao longo de uma circunferência

```
function TREINAMENTO DO ROBO
    S ← semente aleatória
    for θ = 0 to 359 do
        POLITICA, ERRO ← Aplicar CMA-ES para o ângulo θ com semente S
        if ERRO > 40 then
            S ← semente aleatória
        else
            S ← POLITICA
```

3 VISUALIZAÇÃO E ANÁLISE DOS DADOS DE TREINAMENTO

O uso de ferramentas de visualização é essencial na análise de dados, permitindo que um analista encontre e compreenda características, padrões e comportamentos presentes em um conjunto de dados que são, muitas vezes, subjetivas e impossíveis de serem identificadas exclusivamente por outros meios, como análises estatísticas ou mineração de dados. É importante notar que a visualização não está restrita apenas à apresentação de resultados ou à explicação de um conjunto de dados final, mas também possui um papel importante como uma ferramenta exploratória de apoio e validação durante o desenvolvimento de um projeto de aprendizado de máquina, como também de outras áreas.

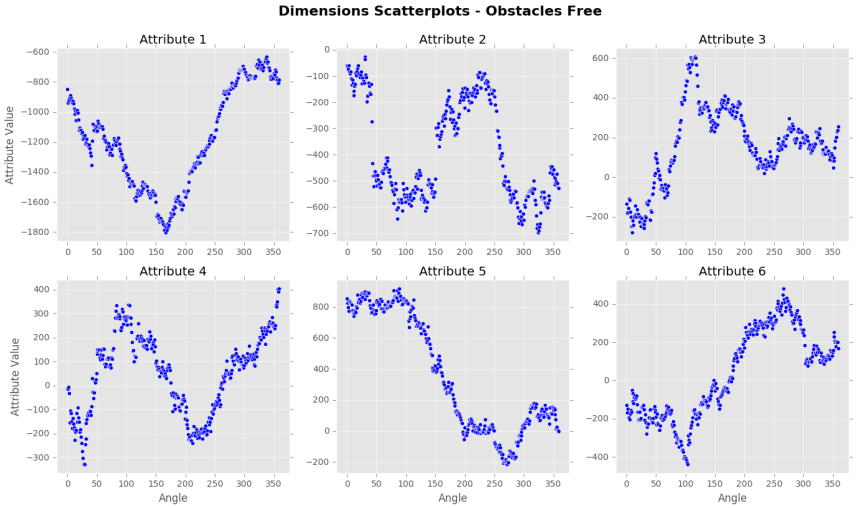
O procedimento de treinamento do robô gera uma quantidade considerável de dados que pode ser analisado. O primeiro deles é o conjunto de 360 políticas hexadimensionais usadas para gerar o movimento de cada tarefa, cada uma delas com um erro associado ao seu uso, durante a resolução de uma tarefa em particular. A execução de uma tarefa pelo robô também gera outros dois conjuntos de dados: um conjunto de 20 pontos de controle (chamados de *waypoints*) que são usados para guiar o braço até o ponto alvo e o conjunto de posições x, y efetivamente percorridas pela trajetória do braço. Este último conjunto não tem um tamanho definido, de forma que trajetórias mais complexas passarão por um número maior de pontos que uma trajetória mais simples.

3.1 Conjuntos de Treinamento de Políticas de Alta-Dimensionalidade

O primeiro passo da análise foi visualizar o comportamento dos seis parâmetros ao longo da circunferência. Para isso, foram gerados dois conjuntos de treinamento usando o procedimento mencionado no capítulo anterior. Um destes conjuntos foi gerado com a configuração sem obstáculos e o outro foi gerado com a configuração com obstáculos. As coletas de ambos os conjuntos servem para que seja possível procurar padrões que ajudem a identificar que características parecem estar presentes para que seja possível projetar uma solução mais eficiente de generalização dos movimentos, permitindo a compreensão de quando e como ocorrem descontinuidades no conjunto de dados, se eles podem ser vistos como uma ou mais superfícies disjuntas, se são lineares ou apresentam curvaturas, qual a dimensionalidade intrínseca, entre outras características relevantes.

Mais precisamente, cada conjunto de treinamento gerado é formado por $N = 360$ pontos $D = \pi_0, ..., \pi_{359}$, onde $\pi \in \mathbb{R}^6$ é um vetor $[\phi_0, ..., \phi_5]$ formado pelos seis parâmetros

Figura 3.1: Conjunto de Dados sem Obstáculos - Atributos da DMP em Função do Ângulo



da política treinada para a resolução da tarefa i, ou seja, o movimento necessário para que o robô alcance a posição do ângulo i na circunferência. Os ângulos são calculados em sentido anti-horário, onde o ângulo 0 está sempre posicionado no ponto verde na imagem da esquerda na Figura 2.1.

A Figura 3.1 mostra como os seis parâmetros da DMP variam em função da tarefa realizada, ou seja conforme o ângulo onde o alvo está localizado varia, para um treinamento realizado sem obstáculos. As políticas variam de maneira bem suave ao longo da circunferência, validando a suposição de que tarefas parecidas permitem soluções próximas. Essa constatação é importante pois, saber que a curva formada pelo treinamento possui uma continuidade como essa permite buscar formas de aprender o formato desta curva através de técnicas de regressão, de forma que não seja necessário executar todo o processo de aprendizagem de um novo movimento através da otimização da DMP.

O conjunto de dados do treinamento com obstáculos, por outro lado, não apresenta uma regularidade tão grande quanto o anterior. Como é possível observar na Figura 3.2, as políticas resultantes do treinamento com obstáculos resultaram em três regiões bastante distintas. Duas delas apresentam a mesma continuidade vista no conjunto do treinamento sem obstáculos, com as políticas variando suavemente ao longo dos ângulos. Porém, a terceira região apresenta um comportamento completamente irregular, sem qualquer relação aparente entre os pontos, provavelmente causado pela inicialização das DMPs com sementes aleatórias. A utilização de uma semente aleatória pode ser necessária em algum momento pois, devido a existência de obstáculos, a solução que teve boa performance para um ângulo θ pode ser bem insatisfatória para o ângulo $\theta + 1$, fazendo com que não

Figura 3.2: Treinamentos com Obstáculos - Atributos da DMP em Função do Ângulo

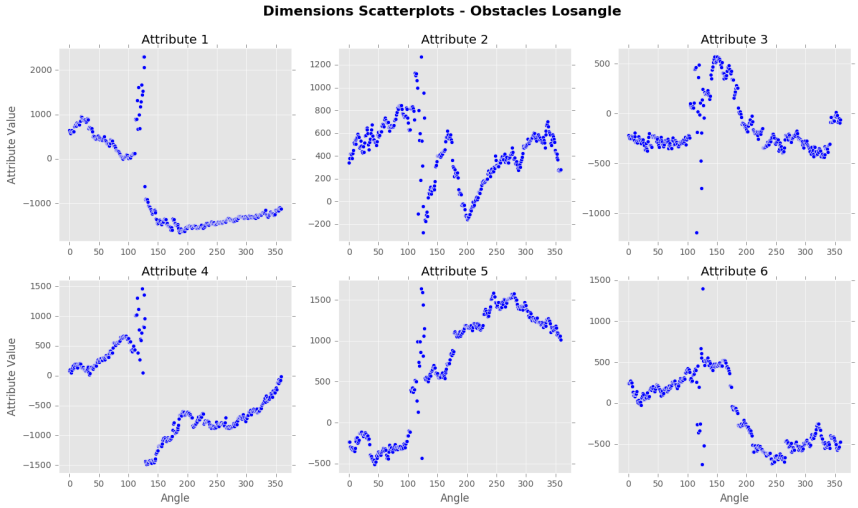
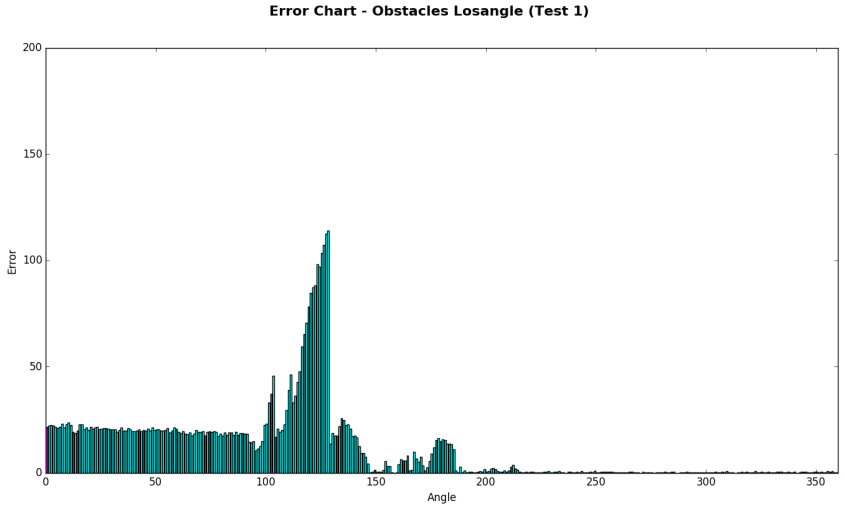


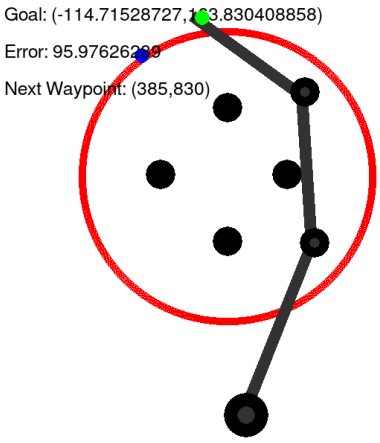
Figura 3.3: Treinamentos com Obstáculos - Erro de Performance (em centímetros)



seja um bom parâmetro de inicialização. Contudo, isso não explica o porquê de uma semente aleatória estar sendo usada repetidamente durante todo o intervalo. Um provável motivo para isso é que esta é uma região intrinsecamente difícil de ser aprendida e, mesmo as novas soluções, feitas a partir de sementes aleatórias, não são satisfatórias. A Figura 3.3 comprova isso, mostrando que esta região teve um erro de performance bastante alto em todas as tarefas, o que forçou as inicializações com semente aleatória repetidamente. A Figura 3.4 deixa mais claro o que está causando a alta taxa de erro de performance, o robô tenta alcançar a posição alvo através de um movimento que o deixa trancado em um obstáculo.

O fato do treinamento do braço mecânico resultar em um má performance mesmo quando inicializado com uma semente aleatória e percorrido num espaço de busca maior

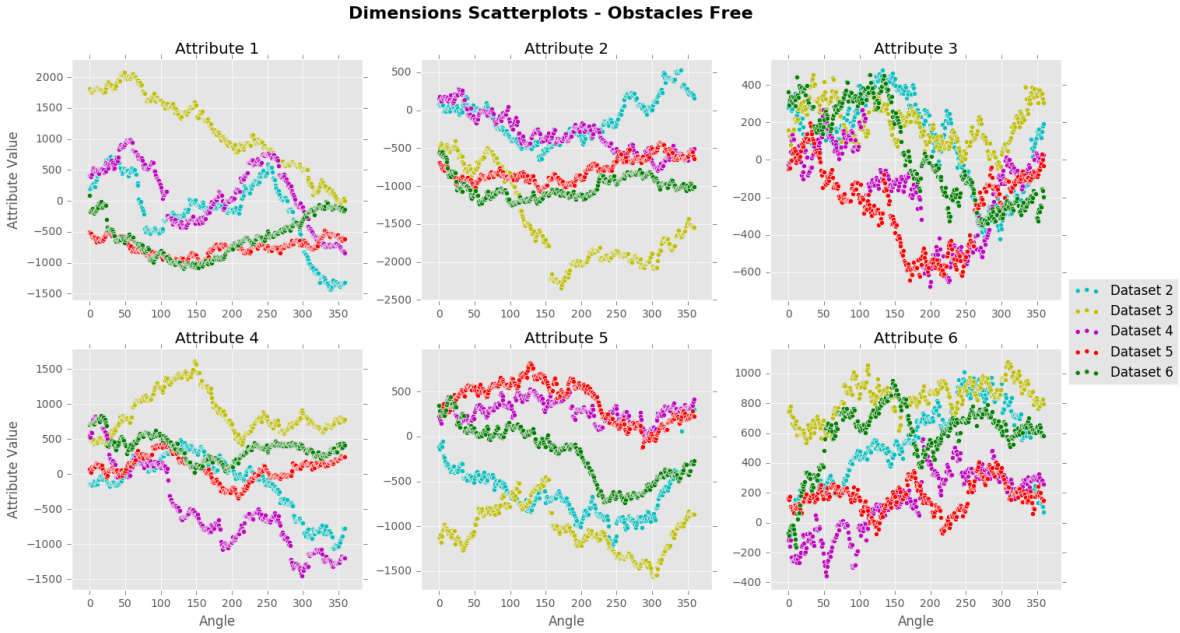
Figura 3.4: Treinamentos com Obstáculos - Comportamento Errôneo do Robô



é preocupante não apenas por causa da qualidade da execução da tarefa em si, até porque esta pode ser intrinsecamente difícil de ser realizada, mas também porque regiões completamente descontínuas como essa inviabilisariam, ou ao menos prejudicariam, a utilização de uma regressão para generalizar o comportamento do robô. Tendo isto em vista, foi necessário fazer uma modificação no procedimento de treinamento das tarefas pelo simulador. Ao invés de se basear no erro de performance do ângulo anterior para decidir se utiliza uma semente aleatória ou não, o treinamento é sempre realizado com a solução da tarefa anterior como semente e, caso resulte em um erro de performance p_s acima do limiar de tolerância, o treinamento é realizado novamente com uma semente aleatória, resultando em uma performance p_a. Contudo, o resultado do segundo treinamento só é utilizado como solução final se $p_a \leq p_s - \varepsilon$, onde ε é um limiar que define se a solução encontrada pela semente aleatória é suficientemente melhor que a solução encontrada com a política da tarefa anterior como semente, de forma que a estrutura formada pelo conjunto de parâmetros no espaço hexadimensional só perca a descontinuidade se isso resultar em um ganho na qualidade do movimento realizado. É importante notar que realizar dois treinamentos para um mesmo ângulo pode aumentar ainda mais o tempo total para treinar todas as tarefas desejadas. Porém, espera-se que, com a generalização do comportamento através de regressões, esse tempo adicional seja compensado.

Utilizando esta nova abordagem, foram gerados cinco novos conjuntos de treinamento com a configuração sem obstáculos e cinco outros com a configuração com obstáculos. Os gráficos de dispersão dos seus atributos se encontram nas Figuras 3.5 e 3.6. Nenhum dos novos conjuntos apresentou regiões tão irregulares como a presente na primeira versão do treinamento, apesar de ainda haver descontinuidades, o que não é possível de mudar, dada a existência de obstáculos, a qual implica a necessidade de movimentos

Figura 3.5: Novos conjuntos treinados com a configuração sem obstáculos



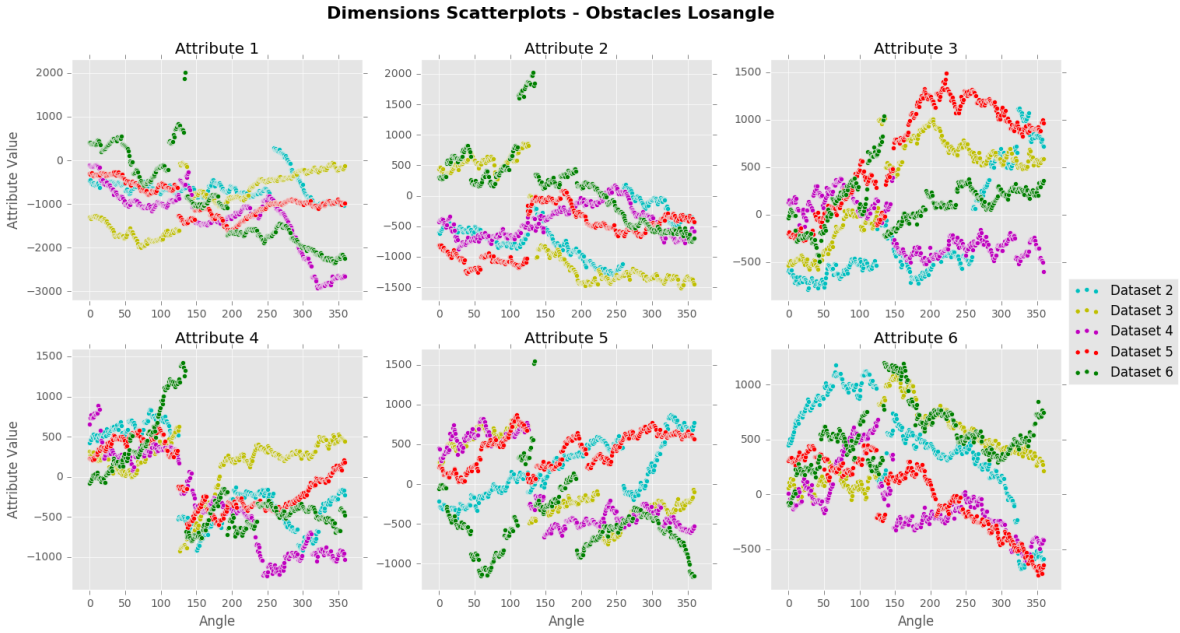
qualitativamente diferentes (e, portanto, de parâmetros da política DMP e regiões diferentes no espaço hexadimensional dos parâmetros). Além disso, o gráfico permite constatar que as soluções encontradas pelo treinamento do simulador são bastante distintas nos diferentes conjuntos, indicando que o resultado final do treinamento é bastante sensível à semente aleatória utilizada.

3.2 Mensurando Performance de Políticas

As características encontradas nos novos conjuntos treinados demonstram que o resultado final do treinamento é fortemente dependente da semente aleatória utilizada para treinar a primeira tarefa (ângulo alvo) de cada sequência. Isso pode indicar que existem várias soluções igualmente boas no espaço de busca ou que o treinamento está caíndo em um grande número de mínimos locais.

A Figura 3.7 mostra o gráfico de barras com os erros de performance (o quão distante o braço ficou do ponto alvo ao final do movimento para cada tarefa) de todos os conjuntos treinados com configuração sem obstáculos. Como era esperado, já que a inexistência de obstáculos facilita a movimentação do braço mecânico, em nenhum dos conjuntos ocorreu erros muito elevados de performance. Porém, é possível observar que os erros geralmente são maiores na metade superior da circunferência (ângulos 0^o à 180^o),

Figura 3.6: Novos conjuntos treinados com a configuração com obstáculos



indicando que essa é uma região levemente mais difícil de se treinar, o que é coerente, pois exige que o braço seja esticado para alcançar o alvo, o que reduz a quantidade de movimentos efetivos.

O erro de performance é mais crítico nos conjuntos das configurações com obstáculos, como pode ser visto na Figura 3.8. Em algumas regiões, em especial no intervalo entre 100º e 150º, os obstáculos fazem com que se torne complicado encontrar boas soluções para as novas tarefas.

O algoritmo de treinamento parece ser inclinado à encontrar uma solução próxima a da semente com qual ele foi inicializado, seja ela aleatória ou resultante de um treinamento anterior, com exceção de algumas regiões críticas que exigem uma mudança muito significativa do movimento. Sendo assim, se torna importante descobrir qual a chance de se encontrar uma solução boa dada uma semente aleatória, já que uma solução não-ótima seria propagada para as tarefas seguintes. Para isso, foi produzido um experimento onde foram conduzidos 100 treinamentos para um mesmo ângulo alvo θ com sementes completamente aleatórias. Formalmente, foi produzido um conjunto de $N = 100$ políticas $[\pi_1, ..., \pi_{100}]$, onde $\pi \in \mathbb{R}^6$ é um vetor $[\phi_0, ..., \phi_5]$ formado pelos seis parâmetros de uma política treinada para a resolução de uma mesma tarefa, representada por um ângulo alvo θ.

A Figura 3.9 mostra o resultado para treinamento com obstáculos do ângulo $\theta = 0$. No canto inferior esquerdo da figura, encontra-se a projeção linear, através de PCA,

Figura 3.7: Erros de Performance do Robô - Treinamentos sem Obstáculos (em centímetros)

Figura 3.8: Erros de Performance do Robô - Treinamentos com Obstáculos (em centímetros)

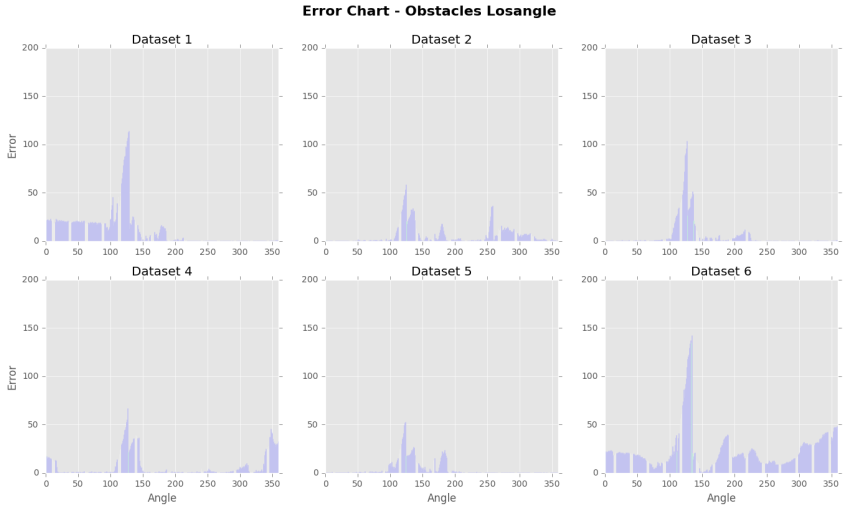
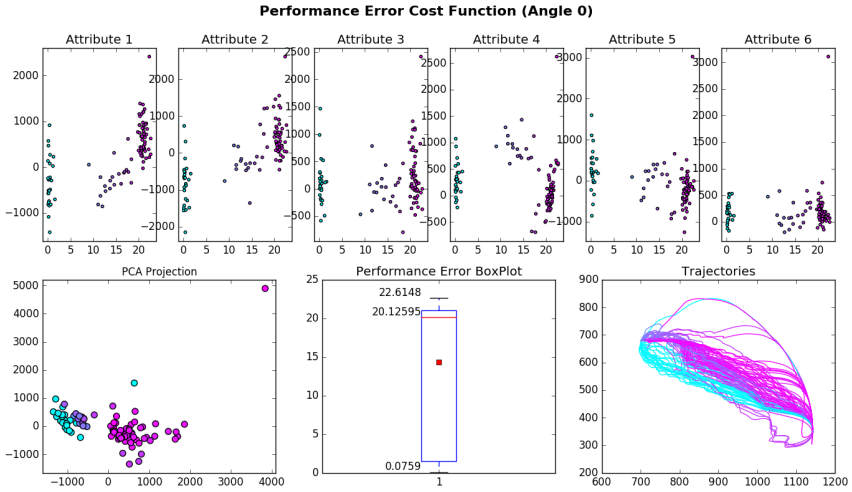


Figura 3.9: Função de Custo do Treinamento com Obstáculos para o Ângulo 0º



das 100 políticas (parâmetros hexadimensionais), onde cores mais próximas do ciano representam um erro baixo e cores mais próximas do rosa representam um erro alto. Os seis gráficos de dispersão na parte superior correlacionam o erro de treinamento (eixo x) e o valor de cada um dos seis atributos (eixo y). Por sua vez, no plot central inferior da imagem encontra-se o *boxplot* da taxa de erro dos 100 treinamentos realizados. Já no canto inferior direito, encontram-se os plots das trajetórias resultantes de cada política calculada. Novamente, as cores representam o erro de performance da trajetória.

O *boxplot* dos treinamentos aleatórios para $\theta = 0$ deixa claro que a maior parte dos treinamentos resultaram em mínimos locais, já que o erro mediano ficou em 20.12, muito próximo do erro máximo, que foi de 22.61, e longe do erro mínimo de 0.07. Ao se observar a projeção, é possível notar que, salvo alguns *outliers*, as políticas se concentraram em duas aglomerações, uma maior com uma taxa de erro alta e uma menor com uma taxa de erro mais baixa, indicando que existem duas regiões para as quais o treinamento tende a convergir, sendo uma delas bem mais eficiente que a outra. Isso fica mais claro ao se analisar as trajetórias, sendo possível observar que as políticas com boa performance seguiram um mesmo padrão de trajetória. Nos gráficos de dispersão de cada dimensão, é possível identificar as políticas com baixo erro possuem diferenças significativas apenas nos atributos 1 e 2, quando comparadas as políticas que resultaram em um erro mais alto. Isso indica que esses atributos são mais sensíveis a variação na semente e podem estar causando a existência destes mínimos locais.

O mesmo treinamento foi realizado para os ângulos $\theta = 120, 180, 270, 335$, com os gráficos resultantes encontrando-se, respectivamente, nas Figuras 3.10, 3.11, 3.12 e

Figura 3.10: Função de Custo do Treinamento com Obstáculos para o Ângulo 120º

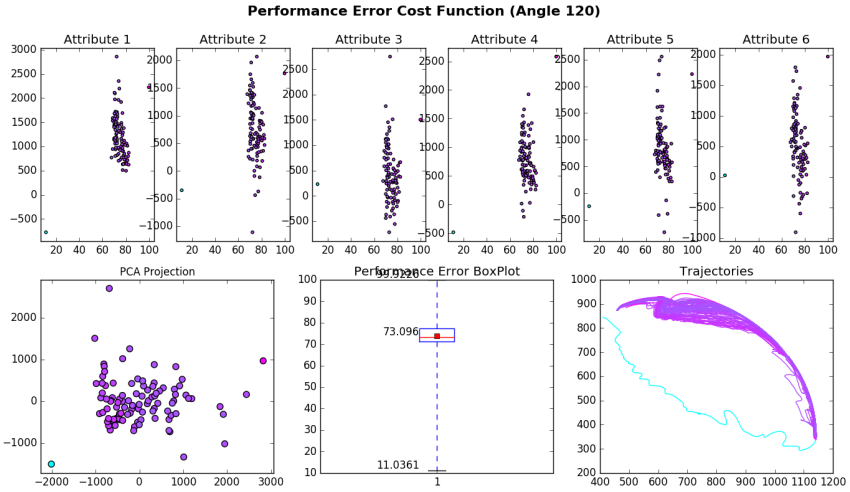
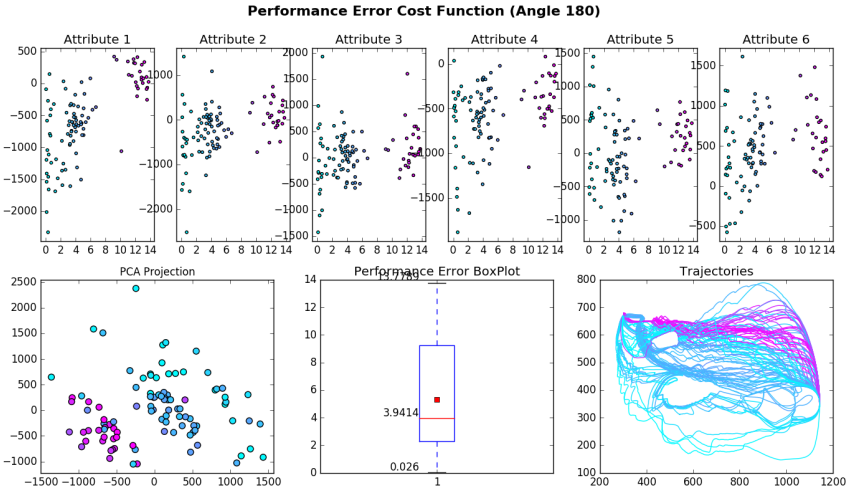


Figura 3.11: Função de Custo do Treinamento com Obstáculos para o Ângulo 180º



3.13. Tais testes validam algumas hipóteses já mencionadas, como a dificuldade que o algoritmo de treinamento possui em encontrar uma política boa para ângulos próximos a 120º, já que, dos 100 treinamentos realizados, apenas um chegou a um erro razoavelmente baixo, de 11.03, enquanto o erro mediano foi de 73.09, evidenciando que a configuração de obstáculos utilizada faz com que tal região fique muito difícil de ser treinada, notando-se também que a sua trajetória escolhida foi bastante diferente das que resultaram em erros altos. De maneira oposta, nos outros três ângulos testados, nota-se que os erros foram, em sua maioria, muito próximos de zeros, deixando claro que a região inferior da circunferência consegue ser alcançada mais facilmente pelo braço mecânico, já que uma gama maior de padrões de trajetória podem ser eficientes para estas tarefas.

Para corrigir estes resultados, o parâmetro σ do CMA-ES foi modificado, de forma

Figura 3.12: Função de Custo do Treinamento com Obstáculos para o Ângulo 270º

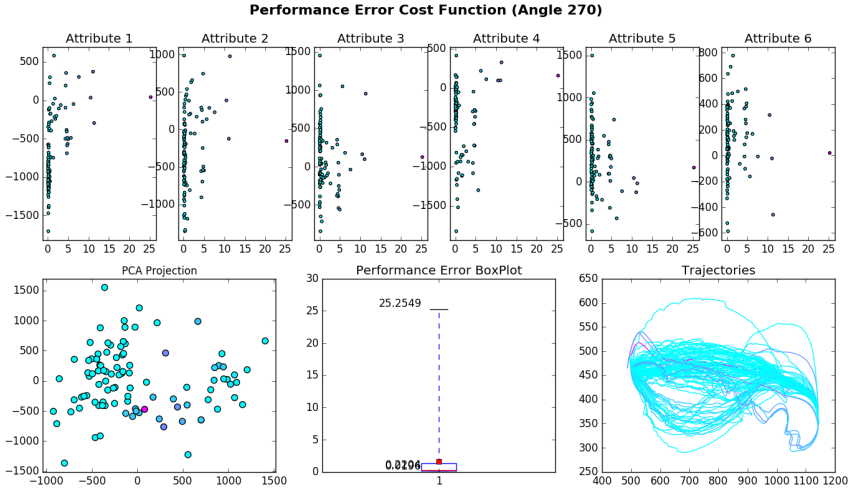


Figura 3.13: Função de Custo do Treinamento com Obstáculos para o Ângulo 335º

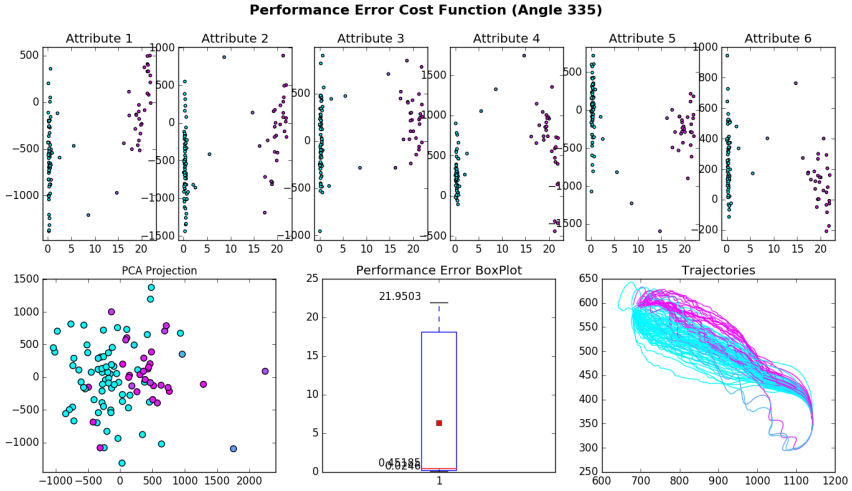
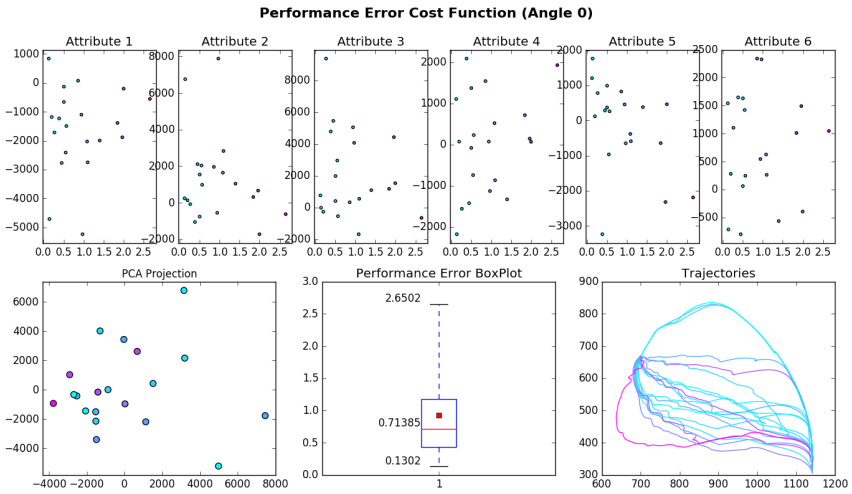


Figura 3.14: Função de Custo do Treinamento com Obstáculos para o Ângulo 0º (Após Modificação)

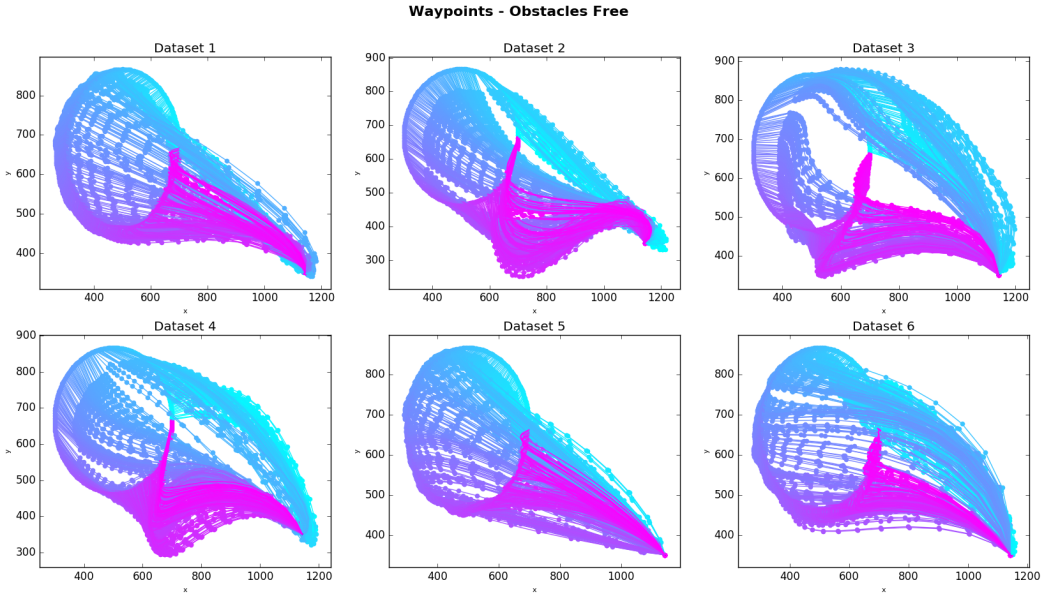


que a variância permitida das novas soluções filhas geradas a cada iteração fosse maior e, assim, tivesse uma maior chance de encontrar os melhores mínimos locais. A Figura 3.14 mostra que de fato, houve uma melhora enorme para o ângulo 0º, onde agora praticamente todas as políticas encontradas tem um erro bem próximo de zero.

3.3 Visualização de Trajetórias Codificadas por Políticas

Conforme discutido no capítulo 2, as política são codificações compactas de trajetórias. Dada uma política e um ponto alvo, a DMP é capaz de calcular a trajetória à ser efetuada através da amostragem de *waypoints* que serão então utilizados como guias no percurso da trajetória. A Figura 3.15 mostra os pontos de controle gerados pelos conjuntos de dados da configuração sem obstáculos. Todos eles começam em uma mesma posição inicial do braço e seguem uma curva até sua respectiva posição alvo na circunferência. Ao se comparar as curvas obtidas pelos pontos de controle com a taxa de erro mostrado na Figura 3.7, é possível notar que a alta taxa de erro que alguns dos conjuntos de dados apresentam nas posições da metade superior da circunferência ocorre justamente quando a curva percorrida pelos pontos de controle tentam alcançá-las por cima, esticando demais o braço. O comportamento oposto (curvas que tentam alcançar tais posições ”por dentro” da circunferência) também pode ser observado no comportamento dos conjuntos de dados 1 e 5. Esse comportamento fica mais explícito na Figura 3.16, que mostra as trajetórias de fato percorridas pelos conjuntos de dados sem obstáculos. No geral, elas

Figura 3.15: Pontos de Controle de cada Tarefa - Treinamento sem Obstáculos



seguem coerentemente os pontos de controle calculado, sem que a curva realize um caminho muito divergente daquele amostrado pelos pontos de controle. Porém, nas regiões em que o braço tenta alcançar a parte superior da circunferência se aproximando por cima dela, a trajetória sempre acaba antes de chegar ao ponto alvo.

As Figuras 3.17 e 3.18 mostram que a presença de obstáculos modifica drasticamente a curva de pontos de controle calculada, que em alguns casos se torna muito mais complexa, inclusive ficando longa demais para que o braço se quer se aproxime dos seus pontos de controle, como ocorre nos conjuntos 1 e 6, o que pode ocorrer quando um mínimo local não-ótimo força o algoritmo de otimização à seguir em uma direção improdutiva, onde aumentar a distancia dos *waypoints* servirá apenas para diminuir o erro de maneira pouco relevantes.

3.4 Redução de Dimensionalidade de Políticas

Uma das dificuldades em visualizar os dados das políticas é o número de dimensões que ele possui, impedindo uma clara compreensão da estrutura formada por elas no espaço hexadimensional. Técnicas tradicionais de visualização de dados multidimensionais, como matrizes de *scatterplot* e cordenadas paralelas priorizam a análise de cada dimensão individualmente, sendo apropriados quando deseja-se analisar correlações entre dimensões e encontrar certos padrões entre elas, porém eles não consegue dar ao usuário

Figura 3.16: Trajetórias realizadas por cada Tarefa - Treinamento sem Obstáculos

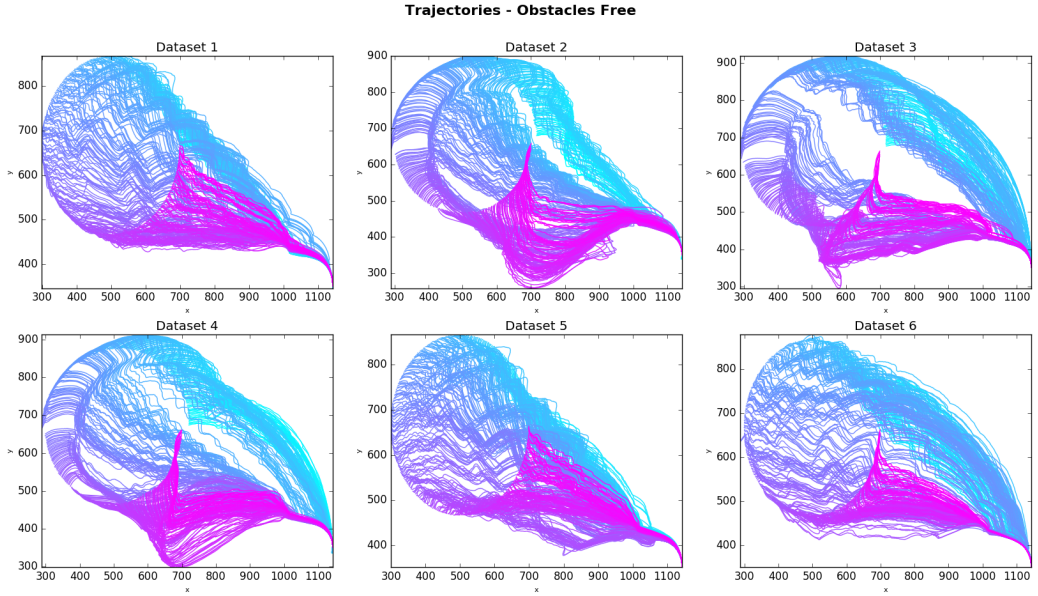


Figura 3.17: Pontos de Controle de cada Tarefa - Treinamento com Obstáculos

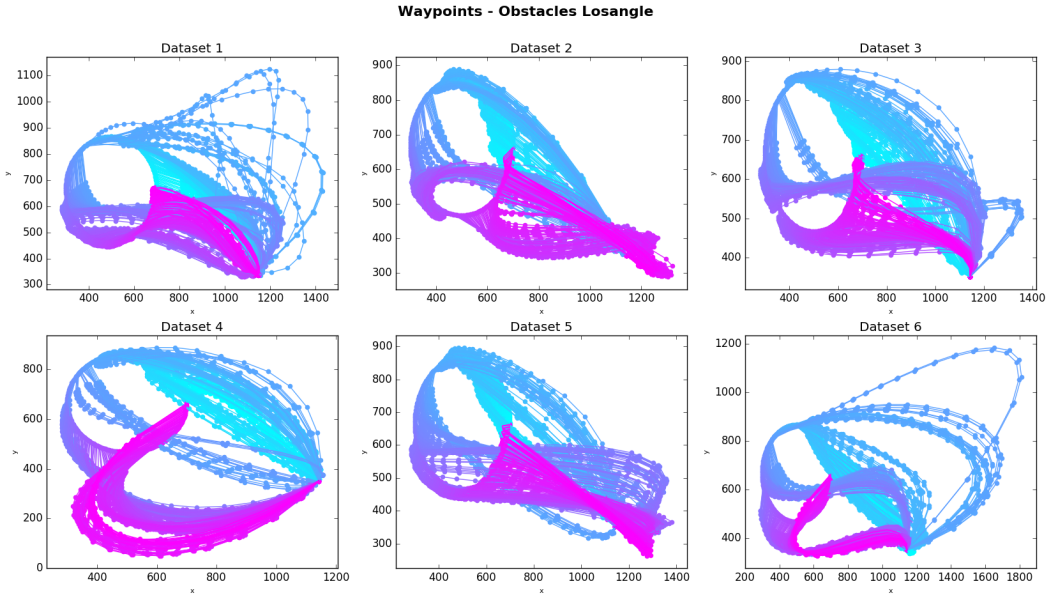
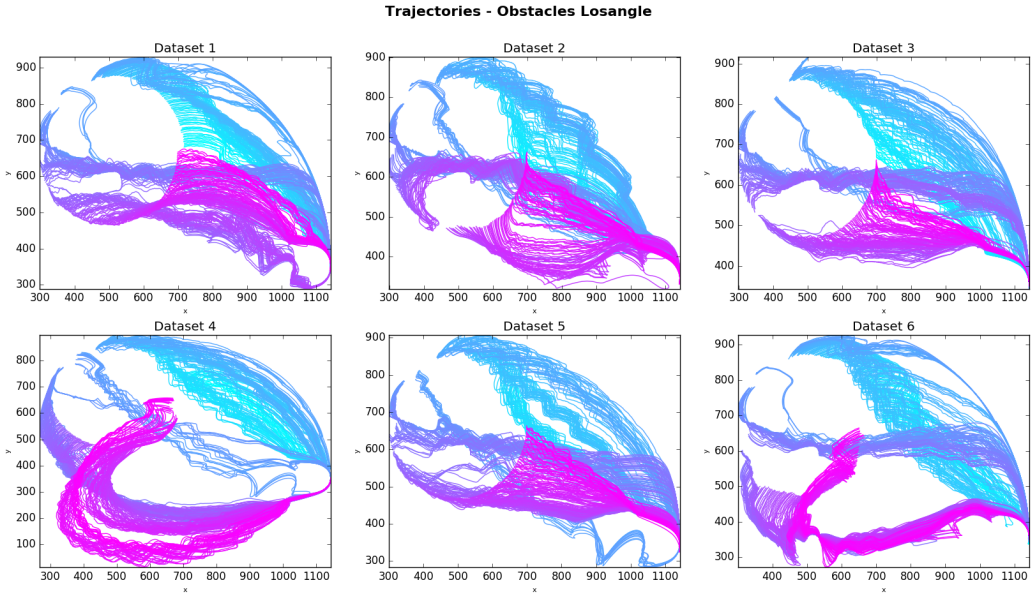


Figura 3.18: Trajetórias realizadas por cada Tarefa - Treinamento com Obstáculos



uma visão espacial de como os dados estão dispostos no espaço de alta dimensão. Conseguir uma compreensão perfeita dessas caracteristicas só seria possível se fizermos um plot N-dimensional do conjunto de dados, o que, devido as limitações da percepção humana, não é possível para $N \geq 4$. Para permitir que um analista possa ter essa visão em baixa dimensão de um conjunto de dados multidimensional, é necessário aplicar nele alguma técnica de redução de dimensionalidade para que possa ser feita uma projeção dos dados ao espaço bidimensional ou tridimensional.

Geralmente, mesmo que um conjunto de dados esteja contido dentro de um espaço d-dimensional, sua estrutura intrínseca pode ter uma dimensionalidade menor. Essa estrutura é conhecida como *manifold*. De maneira geral, um *manifold* m-dimensional é uma estrutura espacial que se comporta localmente como um espaço m-dimensional mas está globalmente embutida em um espaço d-dimensional, onde $d > m$. Exemplos disso são retas e circunferências, que são estruturas unidimensionais independentemente do espaço em que se encontram e esferas, que apesar de ocuparem um espaço tridimensional, são localmente bidimensionais. Algoritmos de redução de dimensionalidade visam reconhecer o formato do *manifold* formado pelo conjunto de dados e reconstruí-los num espaço de menor dimensão da maneira mais fiél possível, preservando as distâncias locais e/ou globais entre os pontos.

Os algoritmos de redução de dimensionalidade estão divididos em dois grupos principais: lineares e não-lineares. Como o nome indica, as técnicas do primeiro grupo

fazem a projeção através de uma transformação linear dos dados, sendo ideais quando eles formam estruturas lineares no espaço de alta dimensão, o que, em muitas aplicações de robótica, não ocorre, além de muitas vezes não ser possível saber de antemão se o conjunto de dados é linear ou não. As técnicas não lineares, por outro lado, buscam projeções que se adaptem às superfícies não-lineares do conjunto de dados, normalmente através de generalizações de técnicas lineares. Geralmente, as técnicas de projeção são formadas em três passos, onde elas divergem apenas em certos aspectos específicos: uma busca de vizinhos mais próximos, o cálculo da afinidade entre os pontos e, por fim, a solução de um *eigenproblem* para projetar o conjunto de dados no espaço de baixa dimensão. [Izenman 2012]

3.4.1 PCA

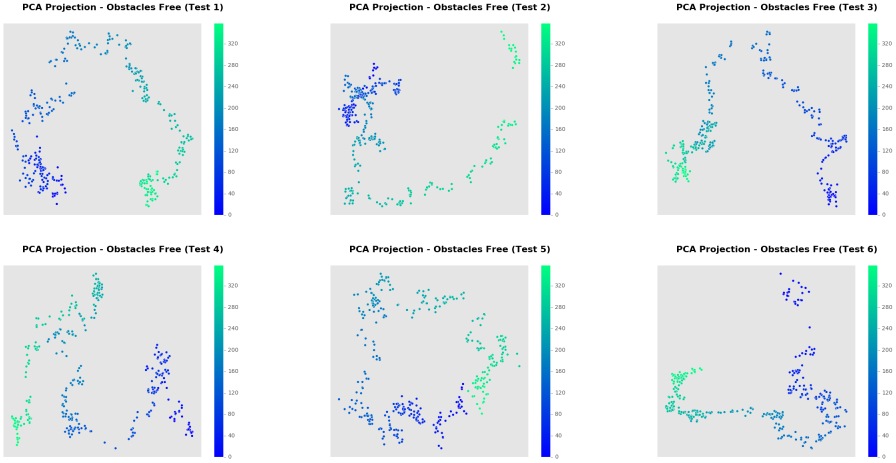
A Análise dos Componentes Principais (PCA) [Shlens 2005] é uma técnica que visa, através de uma transformação linear, decompor um conjunto de elementos N-dimensionais em $M \leq N$ eixos ortogonais alinhados às estruturas de maior variância presentes nas observações originais. Para o contexto de redução de dimensionalidade e visualização, as componentes de interesse são apenas as duas de maior variância, que são usadas como a projeção do conjunto de dados para o espaço bidimensional. O Algoritmo 3.1 ilustra os passos necessários para executar o PCA.

Algorithm 3.1 Método PCA

function PCA
$C \leftarrow$ matriz de entrada $N \times D$
$m \leftarrow$ vetor com a média C em cada uma das D dimensões$C$
$C' \leftarrow C - m$
$CovMat \leftarrow$ matriz de covariancia de C'
$w \leftarrow$ autovalores de CovMat
$v \leftarrow$ autovetores de CovMat
$M \leftarrow$ matriz $D \times 2$ onde as colunas são os autovetores dos dois maiores autovalores de w
$P \leftarrow C \times M^T$
return P

Intuitivamente, o que o PCA faz é criar uma matriz de transformação linear que, após a tranformação, preserva as duas componentes com maior variância do conjunto de dados, que são, em tese, as mais representativas dos dados. Contudo, quando o conjunto de dados é intrinsecamente não-linear, é provável que detalhes importantes da estrutura dos dados possam ser perdidos, já que podem haver um número maior de componen-

Figura 3.19: PCA - Conjuntos de Dados (Treinamento sem Obstáculos)



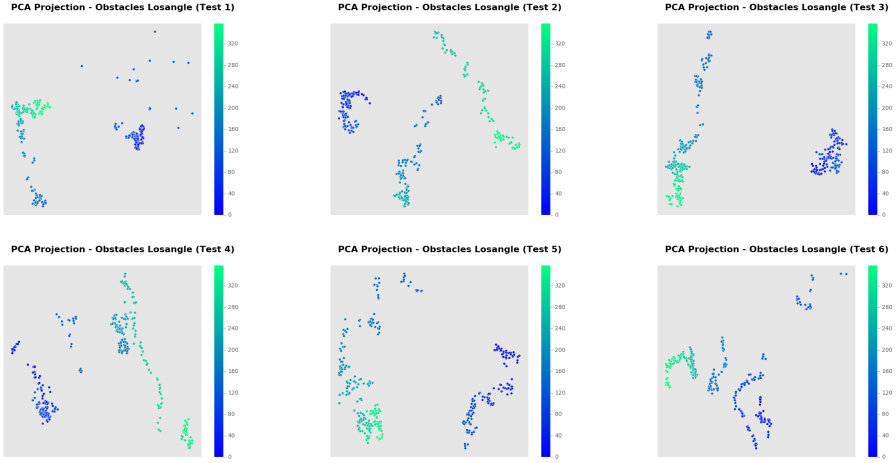
tes com variância significativa e com isso, a projeção linear pode sobrepor regiões não-lineares na alta dimensão.

A Figura 3.19 mostra as projeções dos conjuntos dos treinamentos sem obstáculos utilizando PCA. Nela, nota-se que as pontas do conjunto de dados, ou seja, as políticas designadas aos ângulos alvo 0^o e 359^o tendem a se aproximar, formando estruturas pseudo-circulares, que podem indicar que o treinamento das políticas, ao retornar para uma posição-alvo próxima da primeira posição treinada, acaba se reaproximando das políticas iniciais. Contudo, esse comportamento não ocorre nas projeções dos conjuntos do treinamento com obstáculos, mostrados na Figura 3.20. Nestas, o conjunto acaba sempre formando duas ou três estruturas pouco relacionadas, motivadas pela escolha de sementes aleatórias e consequente descontinuidades no conjunto de políticas. A existência de duas ou mais componentes desconexas nos dados prejudica a visualização através da redução de dimensionalidade, especialmente no caso de projeções lineares. Isso ocorre pois as regiões de máxima variancia de um componente podem não ser as mesmas da máxima variância de outro, fazendo com que ao menos um deles seja mal projetado e informações importantes sejam perdidas.

3.4.2 ISOMAP

Uma das técnicas mais clássicas de projeção não-linear, o ISOMAP (Isometric Mapping) [Tenenbaum" 2000] é baseado na mesma intuição dos algoritmos de projeção lineares, mas com uma diferença: os autovetores usados na projeção são calculados com base em uma matriz M contendo as distâncias geodésicas ao longo do *manifold* entre cada

Figura 3.20: PCA - Conjuntos de Dados (Treinamento com Obstáculos)



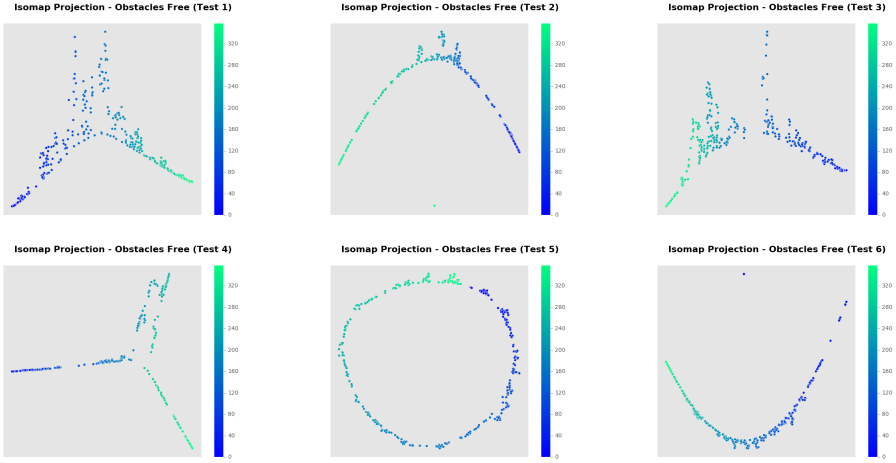
par de elementos do datataset, de forma que elas sejam preservadas, e não as distâncias euclidianas, após a projeção. Para calcular essa distância, cria-se um grafo onde cada elemento é representado por um nodo que possui arestas apenas para os k vizinhos mais próximos. Cada aresta possui um peso igual a distância euclidiana entre os dois pontos. Por serem vizinhos, essa distância é uma boa aproximação da distância entre os pontos ao longo da superfície do *manifold*. As $N \times N$ distâncias geodésicas são calculadas a partir de uma busca de caminho mínimo para cada nodo do grafo, que pode ser computada através de algoritmos como Dijkstra ou Floyd-Warshall. A matriz resultante é usada para fazer a decomposição dos autovetores em um conjunto de dados de baixa dimensionalidade. O Algoritmo 3.2 resume os passos para se calcular a projeção ISOMAP de um conjunto.

Algorithm 3.2 Método ISOMAP

function ISOMAP
- $C \leftarrow$ matriz de entrada $N \times D$
- $G \leftarrow$ grafo de k vizinhos mais próximos de C
- $M \leftarrow$ matriz de caminhos mais curtos em G
- $M \leftarrow M^2$
- $B \leftarrow -\frac{1}{2}JMJ$ onde $J = \mathbb{I} - \frac{1}{N}\mathbb{1}\mathbb{1}^T$
- $w \leftarrow$ autovalores de CovMat
- $v \leftarrow$ autovetores de CovMat
- $E_m \leftarrow$ matriz $D \times 2$ onde as colunas são os autovetores dos dois maiores autovalores de w
- $P \leftarrow E_m \Lambda_m^{\frac{1}{2}}$ onde Λ_m é a matriz diagonal dos m maiores autovalores em w
- **return** P

O ISOMAP apresenta algumas limitações, principamente por ser sensível ao ruído, pois elementos que deveriam estar na vizinhança de um ponto x podem acabar não entrando na vizinhança devido ao acrescimo na distância para x que o ruído pode causar,

Figura 3.21: ISOMAP - Conjuntos de Dados (Treinamento sem Obstáculos)



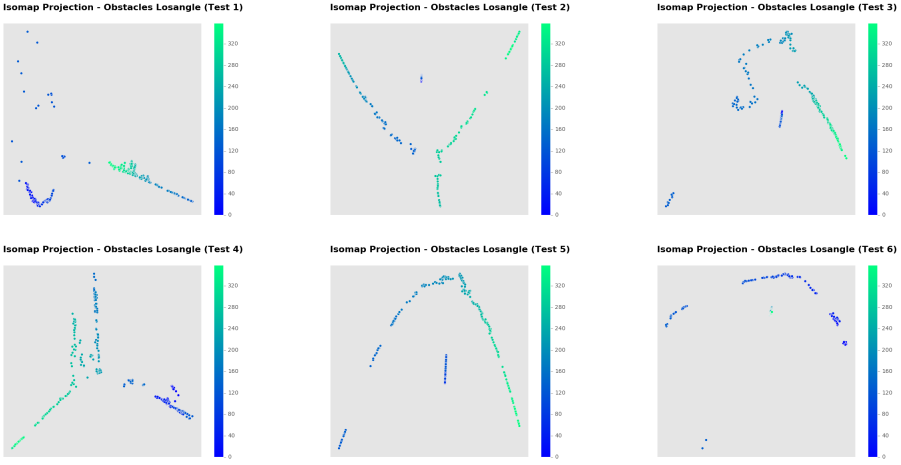
afetando as distâncias geodésicas calculadas e, consequentemente, a projeção realizada. Por levar em consideração a distância geodésica entre todos os pontos, o ISOMAP é considerado um método global, o que faz com que ele seja computacionalmente mais custoso que métodos locais, como o LLE (*Locally-Linear Embedding*) [Izenman 2012].

As Figuras 3.21 e 3.22 mostram as projeções por ISOMAP de ambos os conjuntos de dados. As projeções dos conjuntos de dados do treinamento com obstáculos apontaram a existência de diferentes *manifolds* disjuntos na alta-dimensão, indicando que tarefas semelhantes podem exigir movimentos qualitativamente diferentes, refletindo no uso de parametrizações bem diferentes da DMP correspondente. Já os conjuntos de dados sem obstáculos apresentaram curvas bastante próximas das criadas pelo PCA, porém um pouco mais abertas e com alguns outliers. Isso ocorreu pois o uso da distância geodésica faz as ”pontas” do *manifold* se distanciarem mais na projeção. Contudo, o conjunto de dados 5 demonstrou um comportamento oposto, em que a projeção recuperou uma circunferência quase perfeita, provavelmente pois o processo de treinamento do simulador conseguiu preservar o caráter circular do treinamento, fazendo com que as tarefas dos ângulos acima de 0^o tivessem políticas bem parecidas com as treinadas para os ângulos abaixo de 360^o. Os outliers muito provavelmente ocorreram pois o valor escolhido para $k = 20$ era insuficiente para alguns pontos.

3.4.3 LLE

O LLE (*Locally-Linear Embedding*) [Roweis and Saul 2000] é um método local de aprendizado de *manifolds* que visa preservar, na baixa dimensão, as mesmas distâncias

Figura 3.22: ISOMAP - Conjuntos de Dados (Treinamento com Obstáculos)



locais (dentro de uma vizinhança) que existem no espaço de alta dimensionalidade. Ele pode ser visto como a aplicação de uma série de PCAs locais ao longo da superfície do conjunto de dados, baseando-se na hipótese de que, dentro de uma vizinhança qualquer suficientemente pequena, o *manifold* sempre terá uma aparência localmente linear. O LLE também inicia seu procedimento através da escolha dos k vizinhos mais próximos de cada ponto. Após isso, ele computa os pesos W_{ij} necessários para reconstruir cada ponto x_i através de uma combinação linear dos seus k vizinhos $x_j|j = 1,...,k$. Por fim, ele computa as posições de baixa dimensionalidade y_i de cada ponto do conjunto de dados através dos menores autovetores de W. Os passos do algoritmo se encontram em 3.3.

Algorithm 3.3 Método LLE

function LLE
$C \leftarrow$ matriz de entrada $N \times D$
$G \leftarrow$ grafo de k vizinhos mais próximos de C
$W \leftarrow$ matriz $N \times N$ de pesos para reconstruir cada ponto a partir dos seus vizinhos
$P \leftarrow N$ vetores melhor reconstruídos pelos pesos em W
return P

Métodos locais como o LLE são efetivos quando o *manifold* está uniformemente amostrado, porém não apresentam boa performance ao serem aplicados em superfícies fechadas, como por exemplo, uma esfera. As Figuras 3.23 e 3.24 mostram as projeções dos conjuntos de dados de ambas configurações utilizando LLE. Novamente, as projeções dos conjuntos do treinamento sem obstáculos resultaram em uma visualização que melhor preserva a estrutura dos *manifolds* e permite um entendimento mais claro dos padrões no espaço de alta dimensão. Por outro lado, os conjuntos do treinamento com obstáculos, em que frequentemente a visualização de um cluster acaba sendo priorizada enquanto

Figura 3.23: LLE - Conjunto de Dados (Treinamento sem Obstáculos)

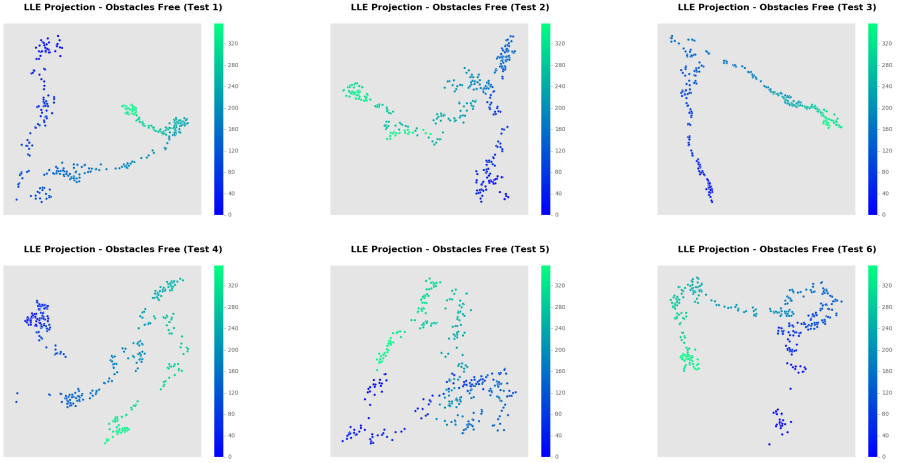
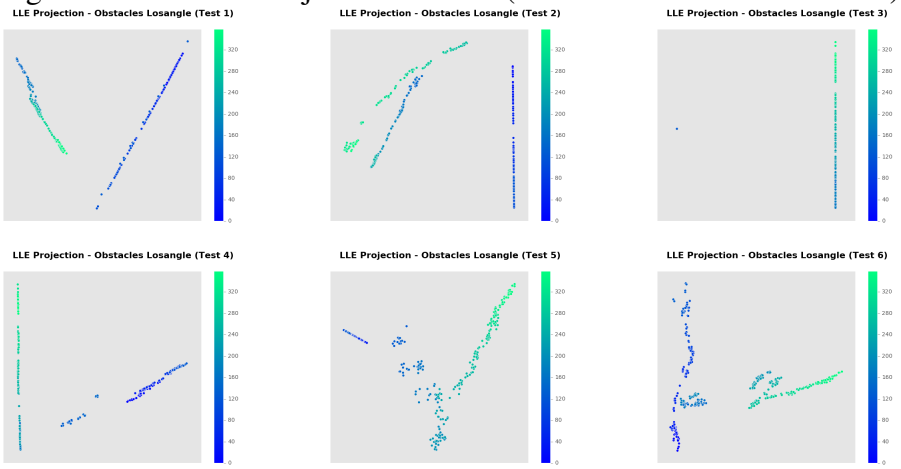


Figura 3.24: LLE - Conjunto de Dados (Treinamento com Obstáculos)

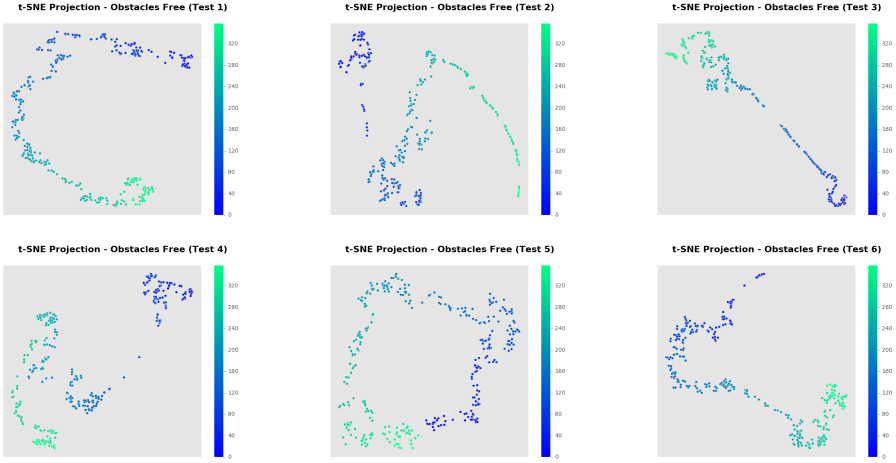


o outro é achatado a tamanhos ínfimos, como no conjunto de dados 3, em que vários elementos foram mapeados para a mesma posição no lado esquerdo da projeção. Outra característica do LLE foi 'linearizar' muitas das superfícies encontradas, o que é causado pelo seu comportamento local.

3.4.4 t-SNE

O *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [Maaten and Hinton 2008] é um algoritmo de redução de dimensionalidade desenvolvido especialmente para a utilização em visualização de dados, sendo ideal para projetar dados em duas ou três dimensões. O método visa fazer a projeção de um jeito que o tamanho da vizinhança afete o quão próximo o ponto ficará de seus vizinhos na baixa dimensão. Ou seja, elementos

Figura 3.25: t-SNE - Conjunto de Dados (Treinamento sem Obstáculos)



fortemente agrupados no alta dimensionalidade serão forçados a continuar próximos na baixa dimensão enquanto elementos com maior dissimilaridade serão afastados.

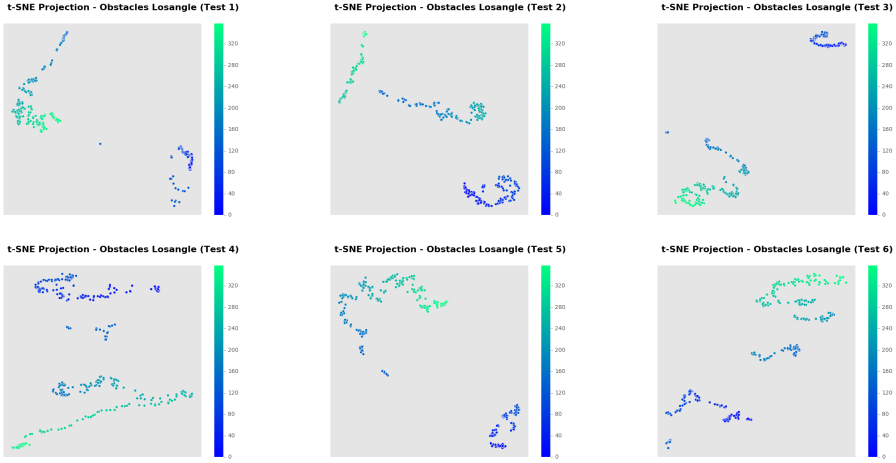
A primeira etapa do t-SNE é construir uma distribuição probabilística onde elementos muito similares tenho uma probabilidade mais alta de serem escolhidos. Em seguida, uma distribuição equivalente é feita no espaço de baixa dimensão e a diferença entre ambas é minimizada, de forma a encontrar uma projeção ótima para o conjunto de dados.

As Figuras 3.25 e 3.26 mostram as projeções de ambos os conjuntos utilizando o t-SNE. O t-SNE foi o método de projeção que resultou nos melhores resultados para os treinamentos com obstáculos, sendo possível identificar com mais facilidade o formato das componentes disjuntas e o número de elementos em cada uma delas. Nos conjuntos treinados sem obstáculos, seus resultados foram parecidos com os das outras projeções, porém com uma tendência maior a ”linearizar” os manifolds, como pode ser observado no conjunto de dados sem obstáculos 3.

3.5 Propriedades dos Manifolds de Soluções do Braço Robótico

Como foi possível identificar em praticamente todas as projeções, os conjuntos de dados das configurações sem obstáculos estão contidos em um *manifold* unidimensional não-linear embutido no espaço hexadimensional, enquanto os treinamentos com obstáculos apresentam múltiplos *manifolds* unidimensionais. Isso é coerente se for levado em consideração que, ao longo das tarefas (posições alvo), só existe um grau de liberdade real a ser modificado: o ângulo da circunferência. Outra constatação interessante foram

Figura 3.26: t-SNE - Conjunto de Dados (Treinamento com Obstáculos)



as curvas que ao chegar próxima ao grau 360 da circunferência, especialmente nos dados dos treinamentos sem obstáculos, voltam a se aproximar espacialmente (mesmo que não completamente) das tarefas treinadas para os ângulos logo acima de 0. Isso nos diz que o treinamento está conseguindo, além de criar um *manifold* contínuo e unidimensional, aprender, de algum jeito, a característica circular do treinamento. Apesar de coerente que as políticas eficazes para tais tarefas realmente possam ser parecidas, a forma como o treinamento é feito não indica que isso poderia ocorrer, já que, mesmo que a semente usada para treinar um novo ângulo seja sempre a política do ângulo anterior, esta já terá mudado completamente até chegar no final da circunferência. Uma outra observação importante é que mesmo havendo quatro obstáculos na configuração dos treinamentos, em alguns conjuntos só foi possível identificar uma ou duas regiões de grande descontinuidade. Isso indica que, dependendo da posição em que o obstáculo se encontra, os parâmetros não precisam ser modificados drasticamente para que ele seja desviado.

Sendo possível afirmar que os conjuntos possuem continuidade locais suaves, porém podendo apresentar quebras, causadas pelo mudança de *manifold*, o próximo passo deste trabalho se torna pensar em um jeito de generalizar o comportamento do robô de forma que não seja necessário repetir o treinamento exaustivamente para novas tarefas. Uma abordagem bastante utilizada para isso é fazer a regressão do conjunto de dados, encontrando uma curva que explique a distribuição dos dados ao longo das tarefas e possa prever novas saídas dada uma nova entrada não-vista.

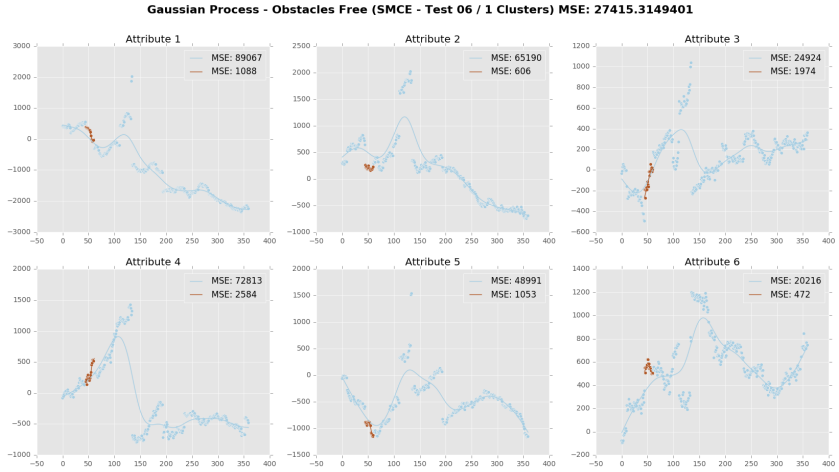
4 UM MÉTODO PARA GENERALIZAÇÃO DE COMPORTAMENTOS

A medida que o número de tarefas à serem treinadas aumenta, fica cada vez mais impraticável fazer o aprendizado de todas através do algoritmo de otimização de DMPs, já que o tempo necessário para encontrar boas soluções num espaço de busca hexadimensional se torna muito elevado. Tendo isto em vista, se faz necessária a busca por métodos que permitam a generalização dos comportamentos do robô. Isto é, métodos que permitam que soluções adequadas para novas tarefas sejam encontradas baseando-se em soluções já conhecidas para tarefas similares, mas sem que seja necessário fazer o aprendizado através do simulador.

Uma maneira tradicional de se fazer esse tipo de generalização é através de algoritmos de regressão. Estes algoritmos recebem como entrada um conjunto de treinamento consistindo de pares com um objeto de entrada e um valor de saída e buscam, baseando-se no conhecimento retirado destes dados, encontrar uma função que modele o comportamento apresentado por eles. No caso do problema de aprendizado de tarefas do braço mecânico, pode-se pensar no objeto de entrada como sendo o ângulo a ser alcançado e o valor de saída como sendo um dos atributos da política necessária para deslocar o braço até a posição alvo. Logicamente, isso necessitará que seis regressões sejam realizadas, uma para cada atributo. Caso o *manifold* formado pelo conjunto de dados fosse linear e de dimensão intrínseca menor que seis, seria possível diminuir o número de regressões à serem feitas, utilizando uma técnica de projeção linear, como o PCA, para que os dados sejam passados para a baixa dimensionalidade e fazendo a regressão apenas nesta baixa dimensionalidade. Assim, para descobrir uma política adequada para um dado ângulo seriam necessários dois passos: encontrar os valores na regressão da projeção correspondentes àquele ângulo e fazer a transformação linear pseudo-inversa da aplicada pelo PCA, de forma a recuperar o valor na alta dimensão. Infelizmente, essa abordagem não é possível quando se trata de *manifolds* não-lineares, já que, ao fazer a redução de um conjunto de dados não-linear, é bastante provável que algumas características do conjunto de dados fossem perdidas.

Além da necessidade de se fazer uma regressão para cada dimensão, existe outro problema a ser contornado: a existência de *manifolds* disjuntos. Uma das premissas da regressão é que o conjunto de dados de treinamento é contínuo ao longo de todo o seu domínio, de forma que a curva criada pela regressão permita representar os dados o mais fielmente possível. Porém, isso não é o que ocorre nos conjuntos apresentados,

Figura 4.1: Regressão em conjuntos com múltiplos *manifolds*



especialmente nos relativos ao treinamento com obstáculos, onde os valores de entrada da regressão são contínuos, mas a função à ser modelada poderia apresentar descontinuidades. Neste caso, a regressão pode não ser efetiva, já que ela tentará modelar dados provenientes de estruturas diferentes em uma mesma curva, fazendo com que as mesmas se unam e a regressão acabe não sendo fiél a nenhuma das duas. A Figura 4.1 exemplifica uma ocasião em que isso pode ocorrer.

Para contornar este problema, será necessário usar um método de clusterização dos *manifolds* presentes no conjunto de dados, de forma a identificar as regiões disjuntas presentes no conjunto de dados, para fazer uma regressão em cada uma delas. Métodos tradicionais de clusterização, como o *K-Means* podem funcionar em conjunto de dados com *manifolds* bem separados e quando os dados de cada agrupamento tem estrutura esférica ou elíptica, mas em conjuntos em que eles se encontram próximos no espaço multidimensional ou que até mesmo haja alguma intersecção, tais métodos não terão a mesma qualidade. Para resolver este problema, é utilizado um método de clusterização de *manifolds*, denominado SMCE, que, através de otimização de matrizes esparsas, consegue identificar eficientemente os diferentes *manifolds* presentes nos dados.

O método de treinamento proposto se constitui de três partes: Na primeira, um conjunto menor de tarefas será treinado através da otimização de DMP. Em seguida, as políticas resultantes destes treinamentos serão clusterizadas, para que seja possível identificar os seus *manifolds* e, no último passo, são feitas regressões de forma a encontrar curvas que modelem cada um dos atributos de cada um dos *manifolds* encontrados. Ao fazer a predição de uma política para uma nova tarefa, é necessário primeiro identificar o(s) *manifold(s)* que podem conter o seu respectivo ângulo de entrada e criar uma nova

política a partir das previsões feitas pelas regressões. Caso o ângulo possa pertencer a dois ou mais *manifolds*, o que pode ocorrer no caso de sobreposição de *manifolds* (onde um novo *manifold* começa em um ângulo de entrada anterior ao último ângulo do *manifold* prescedente) ou quando a tarefa se encontra num espaço entre *manifolds* disjuntos, em que não é possível afirmar previamente qual deles é o correto. Quando uma dessas condições ocorre, a regressão escolhida é aquela que resulta na política com o menor erro de performance, calculado através da execução da tarefa no simulador. O algoritmo de treinamento com generalização do comportamento é sumarizado em 4.1.

Algorithm 4.1 Treinamento de tarefas ao longo de uma circunferência com generalização do comportamento

```
function TREINAMENTO COM GENERALIZAÇÃO DO COMPORTAMENTO
    X ← Subconjunto de políticas treinadas pela otimização da DMP  2.1
    C ← Conjunto de clusters dos manifolds de X
    for Cada cluster C_i ∈ C do
        for Cada dimensão j ∈ C_i do
            R_ij ← Modelo de regressão para a dimensão j de C_i
```

Nas próximas sessões serão explicados em mais detalhes o funcionamento do algoritmo de clusterização, como foram modeladas as regressões e como funciona o método de amostragem das tarefas que serão treinadas no simulador.

4.1 Identificação e Clusterização de Manifolds Não-Lineares

A clusterização é um problema de aprendizado não-supervisionado que tem como objetivo agrupar automaticamente dados não-rotulados em grupos de acordo com alguma medida arbitrária de similaridade, de forma que membros de um mesmo grupo (*cluster*) sejam mais similares entre si do que com elementos pertencentes à outros *clusters*. É importante ter em mente que o conceito de *cluster* é bem vago, já que a qualidade e corretude de uma clusterização, mesmo que se use a mesma métrica e a mesma entrada, pode depender de outros fatores definidos pelo usuário, como quantidade ou tamanho dos *clusters* encontrados.

Na literatura, há uma imensa gama de algoritmos voltados à clusterização de dados, cada qual mais vantajoso para determinadas situações e menos para outras. Tais algoritmos podem ser classificados em diversos grupos, de acordo com suas características. Nas clusterizações particionais, por exemplo, cada elemento é designado à um único *cluster*, de forma que não haja sobreposição de *clusters*. Clusterizações hierárqui-

cas, contudo, permitem a existência de *sub-clusters*, ou seja, grupos menores contidos dentro de *clusters* principais, enquanto nas clusterizações sobrepostas permitem que um elemento pertença a mais de um *cluster*, sem que precise existir uma relação hierárquica entre eles. Nas clusterizações probabilísticas, por outro lado, cada elemento pertence a qualquer *cluster* com uma probabilidade associada [Xu and Tian 2015].

O objetivo buscado ao se fazer a clusterização do conjunto de políticas associadas aos movimentos dos robôs é separar os *manifolds* presentes na estrutura de alta dimensão dos dados em regiões contínuas, de forma que seja possível fazer regressões mais precisas. Portanto, apesar da distância euclidiana ser uma métrica importante a ser considerada, ela não é a única propriedade necessária nos *clusters* a serem encontrados, já que *manifolds* distintos podem estar próximos no espaço multidimensional. Para isso, foi utilizada uma técnica desenvolvida por Ehsan Elhamifar e René Vidal, chamada Sparse Manifold Clustering and Embedding (SMCE) [Elhamifar and Vidal. 2011], capaz de fazer a clusterização dos *manifolds* e a projeção deles para a baixa-dimensão separadamente.

4.1.1 O Método SMCE

Os algoritmos de aprendizado de *manifolds* tradicionais, como PCA e ISOMAP, são baseados na premissa de que os dados pertencem à uma única superfície, o que na prática nem sempre é verdade, já que conjuntos de dados podem possuir descontinuidades ou são formados por elementos pertencentes a diferentes *manifolds*. Nos conjuntos do braço mecânico, por exemplo, múltiplos *manifolds* podem surgir a partir da existência de obstáculos ou da necessidade de, em algum momento, reiniciar a semente com valores aleatórios. Para que seja possível trabalhar eficientemente com esse tipo de dado, é necessário primeiro identificar quantos *manifolds* ele possui, quais pontos pertencem a cada um deles e qual as suas dimensões intrínsecas. [Vidal 2011]

Na literatura existe uma grande gama de algoritmos de clusterização de subespaços lineares ou afins. Contudo, uma vez que é possível ver um *manifold* como uma coleção de subespaços localmente lineares, a clusterização linear pode resultar em um número muito alto de subespaços, exigindo um número excessivo de regressões, tornando interessante então a utilização de uma técnica de clusterização que permita encontrar os diferentes *manifolds* não-lineares.

O SMCE é um algoritmo que visa servir tanto para a clusterização quanto para a projeção de dados contidos em múltiplos *manifolds*. Assim como os métodos tradicionais

de redução de dimensionalidade, o SMCE busca criar um grafo de vizinhança e utilizar as proximidades locais entre os pontos para modelar a projeção. Contudo, isso não é suficiente para clusterização quando se assume que podem existir múltiplos *manifolds* no conjunto de dados, já que pontos pertencentes a *manifolds* distintos podem estar na vizinhança um do outro. O SMCE ataca este problema através da seleção de subespaços afins dentro da vizinhança de cada ponto, de forma que apenas os vizinhos que pertençam ao mesmo subespaço sejam selecionados na hora da clusterização. Isso é feito através da resolução de um problema de otimização de matrizes esparsas.

O funcionamento do SMCE pode ser resumido em duas etapas: a criação de uma matriz de afinidade entre os pontos do conjunto de dados e a clusterização espectral do conjunto de dados utilizando essa matriz como entrada. Idealmente, pontos dentro do mesmo *manifold* devem ter afinidade alta enquanto pontos em diferentes *manifolds* não, de forma a permitir que a clusterização espectral consiga os separar. Contudo, encontrar uma métrica para identificar isso não é trivial. A abordagem adotada pelo SMCE busca, através da resolução de um problema de otimização de matrizes esparsas, encontrar os vizinhos de cada ponto x_i do conjunto de dados que também se encontram no mesmo subespaço de x_i.

Dado um conjunto de entrada $X \in \mathbb{R}^{N \times D}$, o primeiro passo da construção da matriz de afinidade A entre os N elementos de X é, para cada elemento $x_i \in X$, selecionar os k vizinhos mais próximos, criando um novo conjunto V_i. O elemento x_i é então subtraído de V_i para que a vizinhança fique centrada em zero. Após isso, cada elemento $V_{ij} \in V_i$ é dividido pela sua distância para a origem. Isso cria um viés ao problema de otimização, de forma que os coeficientes relativos a elementos mais próximos de x_i tenham tendência a ganhar pesos não-zero, enquanto elementos mais distantes tenham tendência a receber peso zero. O problema de minimização apresentado na Equação 4.1 é então solucionado pelo SMCE para encontrar um vetor esparso de pesos c_i que sirva de afinidade para o elemento x_i. A matriz Q_i é uma matriz diagonal em que os elementos representam cada um dos pontos em V_i, onde o valor atribuído a cada elemento é proporcional à sua distância para x_i, de forma que pesos maiores em Q_i favoreçam pesos menores no vetor c_i, penalizando pontos que se distanciem muito de x_i. Uma métrica eficiente para calcular a diagonal principal de Q_i é $\frac{\|x_j - x_i\|_2}{\sum_{t \neq i} \|x_t - x_i\|_2}$.

$$\min \lambda \|Q_i c_i\|_1 + \frac{1}{2}\|V_i c_i\|_2^2 \text{ sujeito a } \mathbb{1}^T c_i = 1 \tag{4.1}$$

A matriz de afinidade W é então construída de forma que o elemento W_{ij} receba o

peso calculado c_{ij} dividido pela distância entre os pontos x_i e x_j e normalizado de acordo com o somatório de c_i, para que a matriz de afinidade capture as distâncias locais entre vizinhos dentro de um mesmo *manifold*, além da informação de pertencimento ao mesmo subespaço. O cálculo de cada elemento da matriz W é especificado na Equação 4.2.

$$W_{ij} = \frac{\frac{c_{ij}}{\|x_j - x_i\|_2}}{\sum_{t \neq i} \frac{c_{it}}{\|x_j - x_i\|_2}} \tag{4.2}$$

Por fim, W é utilizada como matriz de afinidade em um algoritmo de clusterização espectral, recuperando a segmentação dos elementos de acordo com os subespaços aos quais eles pertencem. O algoritmo de clusterização espectral funciona através da aplicação de uma redução de dimensionalidade da matriz de afinidade seguida por uma clusterização por K-Means dentro do espaço de baixa dimensionalidade. Mais especificamente, o que o método de clusterização espectral faz é calcular a matriz L do grafo laplaciano normalizado de W e pegar os K autovetores correspondentes aos K maiores autovalores de L, onde K é o número de *clusters* (subespaços) desejado pelo usuário. A matriz formada pelos K autovetores selecionados é então usado como entrada para o K-Means. Como o grafo laplaciano é normalizado, inferir o valor adequado para K é relativamente simples, pois, caso os *manifolds* sejam disjuntos, basta encontrar o número de autovetores (respeitando a ordem definida pelos autovalores) que só possuam elementos 0s ou 1s [Luxburg 2007]. Outro método possível para estimar o valor ideal de K é o método do cotovelo, que busca identificar o valor mais alto de K que ainda diminua significativamente a variância do erro médio quadrático de cada *cluster*. Como o método do cotovelo foi desenvolvido visando a utilização no K-Means, ele deve ser usado com a matriz de autovetores calculada dentro da clusterização espectral e não com os dados originais do conjunto de dados. Contudo, valores maiores de K do que os estimatidos pelos métodos apresentados podem ser interessantes caso deseje-se fazer regressões mais simples, já que o *manifold* pode ter curvas não-lineares bastante complexas.

As Figuras 4.2, 4.3, 4.4, 4.5 e 4.6 mostram alguns exemplos de clusterização feitas com o SMCE, todos usando configuração com obstáculos. É possível notar que o algoritmo se comportou de maneira satisfatória para os dados do treinamento com obstáculos, conseguindo encontrar os pontos de descontinuidade e separando cada *manifold* em seu próprio *cluster*. A Figura 4.7 mostra um exemplo de clusterização com o valor de K extrapolado, que permitirão regressões mais simples ao custo de mais regressões necessárias.

Algorithm 4.2 SMCE

function SMCE

$X \leftarrow$ matriz de entrada $N \times D$

for `i = 0, ..., N-1` **do**

$V \leftarrow k$ `vizinhos mais próxios de` X_i

$V \leftarrow V - X_i$

$Dist \leftarrow$ `distância de cada ponto em` V `para` X_i

$V \leftarrow V /$ `Dist`

$C_i \leftarrow$ `pesos do problema de otimização em 4.1`

for `cada elemento j pertencente a V` **do**

$W_{ij} = (C_{ij}/Dist)/\sum_{t \in V}(C_{it}/Dist)$

$W \leftarrow$ `valor absoluto de cada elemento de W`

$rotulos \leftarrow$ `clusterização espectral em W`

Figura 4.2: SMCE com o Conjunto de Dados 2 da Configuração com Obstáculos

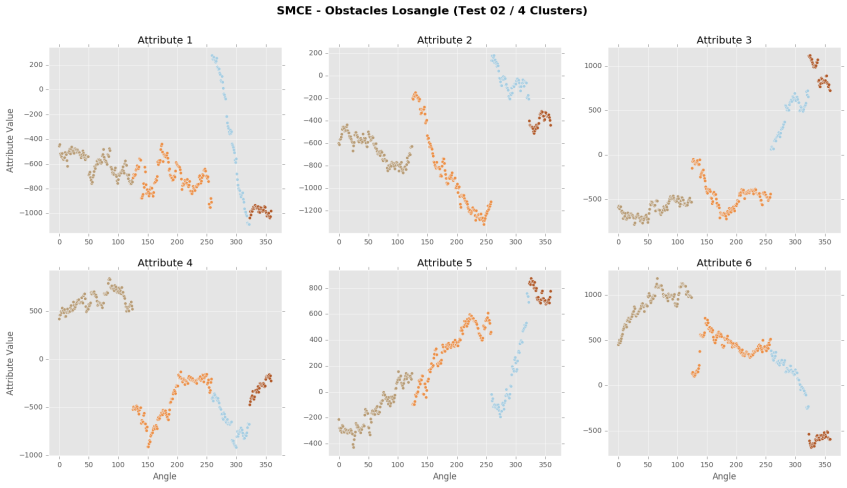


Figura 4.3: SMCE com o Conjunto de Dados 3 da Configuração com Obstáculos

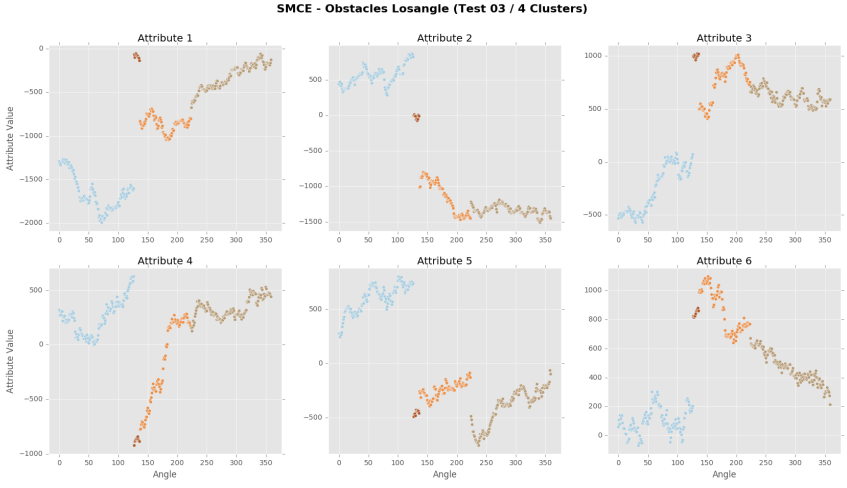


Figura 4.4: SMCE com o Conjunto de Dados 4 da Configuração com Obstáculos

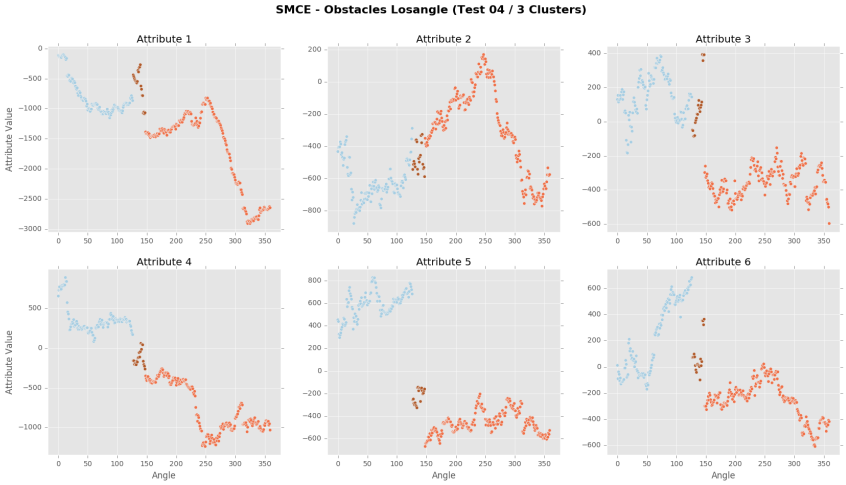


Figura 4.5: SMCE com o Conjunto de Dados 5 da Configuração com Obstáculos

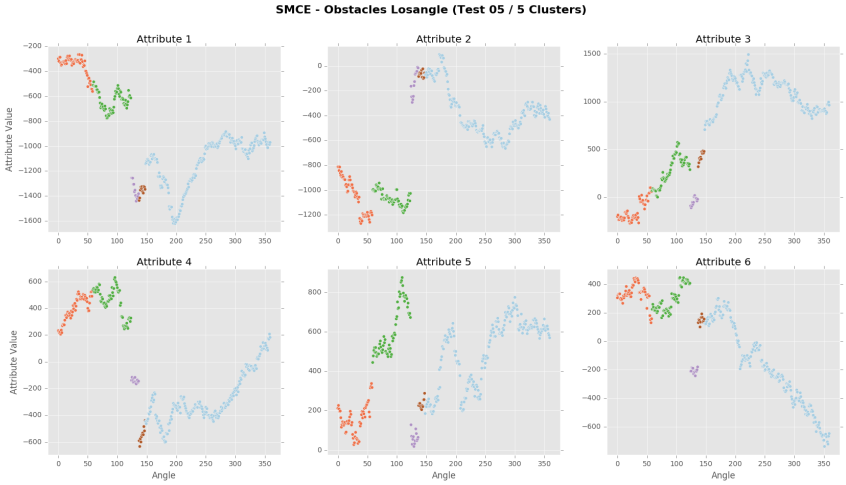


Figura 4.6: SMCE com o Conjunto de Dados 6 da Configuração com Obstáculos

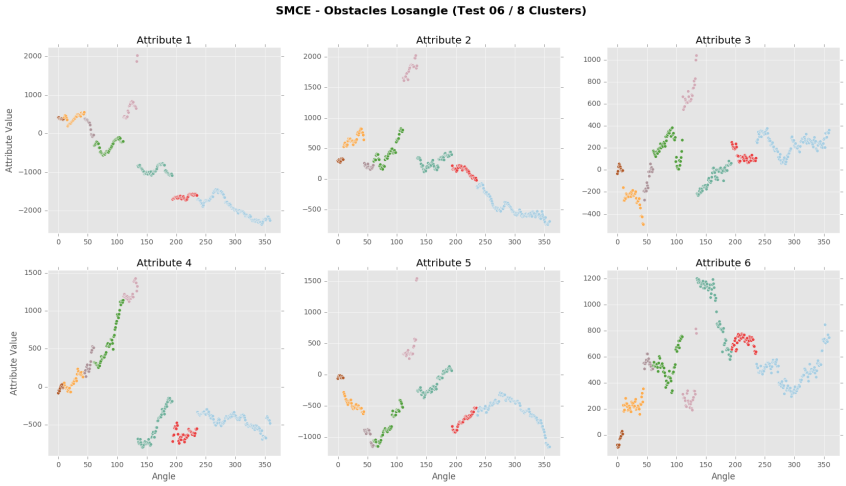
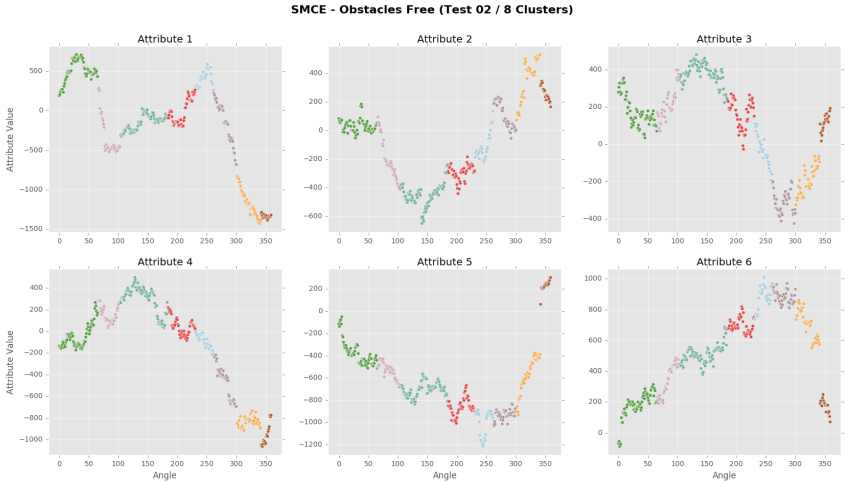


Figura 4.7: SMCE com o Conjunto de Dados 2 da Configuração sem Obstáculos



4.2 Regressão em Manifolds Não-Lineares

Após a clusterização do conjunto de políticas em diferentes *manifolds*, o próximo passo para que seja possível generalizar o comportamento do robô para novas tarefas é fazer regressões separadas para cada um dos *manifolds* encontrados. Isso é necessário pois há regiões do espaço de tarefas (ângulos alvo sobre a circunferência) que possuem soluções em partes disjuntas do espaço de política, e portanto cada um desses intervalos de tarefas deve ter seu próprio modelo de regressão. Ou seja, devem ser modeladas regressões especializadas em tarefas que exijam comportamento semelhante. A predição de uma política para uma nova posição alvo se dará em duas etapas: identificar o *manifold* mais apropriado para aquela tarefa (em casos de sobreposição ou região fronteiriça dos *manifolds*, pode haver mais de uma) e aplicar a predição no modelo de regressão escolhido.

Algoritmos de regressão são métodos de aprendizado supervisionado, capaz de inferir o funcionamento de uma função a partir de exemplos de treinamento cedidos pelo usuário. Um exemplo de treinamento consiste de um par formado por um objeto de entrada D-dimensional e um valor numérico de saída pertencentes à um domínio contínuo. O algoritmo busca, então, generalizar o comportamento encontrado nos exemplos de treinamento para novas entradas, inferindo valores de saída adequados. Os métodos de regressão se diferenciam dos métodos de classificação pois estes últimos trabalham com saídas categóricas, onde para cada objeto de entrada é atribuído um rótulo discreto de saída.

A escolha de um algoritmo de regressão adequado é essencial para o bom funcio-

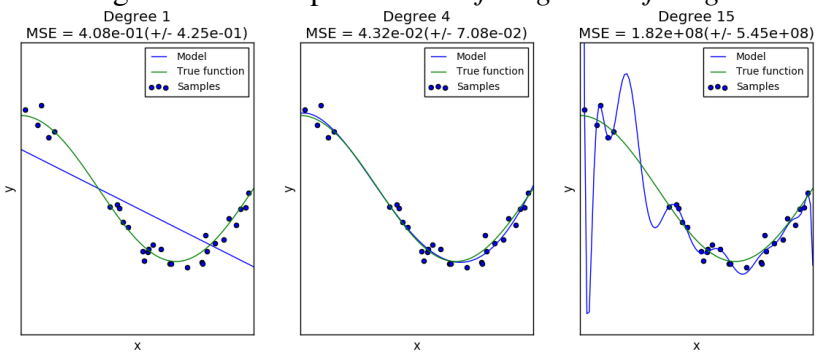
namento de uma regressão. Existe na literatura uma vasta gama de métodos de regressão, cada um com suas qualidades e deficiências, sendo necessário escolher métodos que sejam mais eficientes para o conjunto de dados que está sendo trabalhado, tarefa nem sempre trivial. Os métodos de regressão estão divididos em dois grupos principais: paramétricos e não-paramétricos. O primeiro utiliza o conjunto de treinamento para computar um número fixo de parâmetros que serão usados posteriormente para fazer a predição de novos dados de entrada. Os algoritmos não-paramétricos, por outro lado, utilizam diretamente os dados de treinamento, junto com o conjunto de hiper-parâmetros, para prever saídas para novas entradas. Algoritmos não-paramétricos são recomendáveis em casos de curvas muito complexas, pois não exigem que o projetista escolha tipos e número de *features*, já que o próprio algoritmo já se modela em torno dos dados. Regressão Linear, LASSO e *Ridge Regression* são alguns exemplos de regressões paramétricas, enquanto Regressão do Processo Gaussiano e *Kernel Regression* são alguns algoritmos não-paramétricos.

4.2.1 *Overfitting* e Underfitting

Existem dois grandes tipos de problemas que uma regressão feita incorretamente pode acarretar, *underfitting* e *overfitting*. *Underfitting* ocorre quando o modelo de regressão utilizado é muito simples ou rígido e o domínio dos dados é muito complexo, fazendo com que a regressão não seja capaz de aprender todas as características do comportamento da função. Um exemplo clássico de *underfitting* pode ser visto na Figura 4.8 (primeira imagem), que mostra uma tentativa de fazer a regressão de um conjunto de treinamento sinoidal utilizando uma regressão puramente linear. O *overfitting* (terceira imagem), por outro lado, ocorre quando o modelo de regressão é exageradamente complexo, fazendo com que a predição fique muito sensível a possíveis ruídos e outliers no conjunto de treinamento e gerando resultados errôneos para novas entradas não-vistas. A imagem central da Figura exemplifica uma regressão onde não ocorre nem *overfitting* e nem *underfitting*.

Caso o conjunto de treinamento tenha sido adequadamente gerado, o *underfitting* possui fácil identificação e solução, já que basta modificar o modelo de regressão de forma que ele se adapte melhor as estruturas presentes no domínio. Porém, o mesmo não é verdade quando se trata de *overfitting*. Uma caracteristica do *overfitting* é que o modelo de regressão se adapta de maneira exagerada com relação as amostras de treinamento, fazendo previsões muito boas apenas para as amostras usadas no treinamento, mas não trazendo boa performance na predição de novas amostras. Porém, nem sempre é possível

Figura 4.8: Exemplo de *Underfitting* e *Overfitting*



Fonte: Scikit-Learn [Scikit Learn - Underfitting vs. Overfitting]

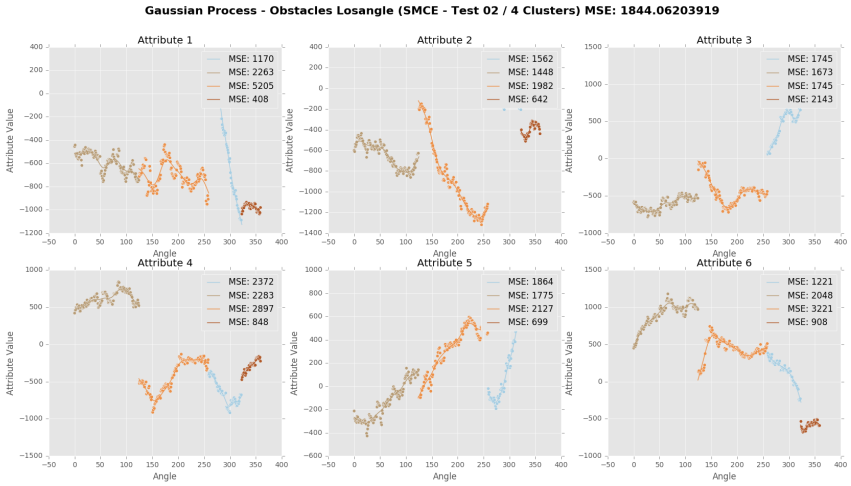
identificar se isso ocorre porque o conjunto de treinamento é bem comportado (com estruturas bem definidas e com pouco ruído e outliers) ou se é porque o treinamento está com *overfitting*. Para solucionar isso, é necessário que a eficência do modelo de regressão seja testado com amostras de testes distintas das usadas no conjunto de treinamento, de forma que os erros causados por *overfitting* possam ser revelados. Porém, em alguns casos a quantidade de amostras com saída conhecida pode ser escassa e dividí-las entre conjunto de treinamento e conjunto de testes pode fazer com que o número de amostras em cada um dos conjuntos fique muito baixo, comprementendo a capacidade de se fazer uma boa regressão.

Uma forma bastante comum de fazer o teste da regressão quando não se possui uma quantidade abundante de dados é a validação cruzada. Na validação cruzada, o conjunto de dados é dividido em k subconjuntos de tamanho similar. Para cada subconjunto k_i, o aprendizado é executado com um conjunto de treinamento formado pelos outros $k-1$ subconjuntos e o subconjunto k_i é usado para teste. A performance da regressão é medida então pela média, normalmente através do erro mínimo quadrado, das performances de cada um dos k aprendizados.

4.2.2 Regressão do Processo Gaussiano

Para este trabalho, foi utilizado o método não-paramétrico de regressão do processo gaussiano [Rasmussen and Williams 2005], implementado na biblioteca do Scikit Learn [Scikit Learn]. A regressão de um processo gaussiano assume que os dados de treinamento estão dispostos estocasticamente através de uma distribuição gaussiana e podem ser descrito pela sua média e variância. A regressão busca então generalizar esse

Figura 4.9: Processo Gaussiano com SMCE com o Conjunto de Dados 2 da Configuração com Obstáculos



conceito com o objetivo de encontrar uma função média e uma matriz de covariância que explique os valores no conjunto de treinamento. A regressão do processo gaussiano foi escolhida por ser um algoritmo não-paramétrico já que não é possível conhecer de antemão qual é a forma dos *manifolds* a serem modelados, sendo difícil de reconstruir as features adequadas necessárias em um método paramétrico.

Os hiper-parâmetros do modelo de regressão foram definidos através de uma busca exaustiva ao longo de um subconjunto de valores, como por exemplo, o coeficiente de regularização o parâmetro de autocorrelação, responsável por definir uma função de *kernel* que determina a similaridade entre os elementos de entrada da regressão. Os modelos foram testados através de uma validação cruzada em que o conjunto de treinamento é dividido em 10 subconjuntos e, a cada rodada, um deles é escolhido como conjunto de teste. O modelo de regressão escolhido foi o que resultou no menor erro médio quadrático entre os modelos de regressão testados na busca exaustiva. As Figuras 4.9, 4.10, 4.11, 4.12, 4.13 e 4.14 mostram como ficaram as regressões realizadas nos *manifolds* clusterizados no capítulo anterior. Para cada um dos *manifolds* clusterizados, foi feita uma regressão por atributo (totalizando seis) usando o número total de elementos pertencentes ao *manifold* na validação cruzada de cada regressão.

O teste do desempenho da regressão feito através da regressão é útil para escolher a configuração de hiper-parâmetros mais eficiente, porém para saber se as saídas geradas pela regressão são realmente boas para modelar trajetórias eficientes do robô, é necessário rodá-las no simulador. A Figura 4.15 mostra o erro de performance dos conjuntos de dados originais, treinados no simulador, enquanto a Figura 4.16 mostra o erro

Figura 4.10: Processo Gaussiano com SMCE com o Conjunto de Dados 3 da Configuração com Obstáculos

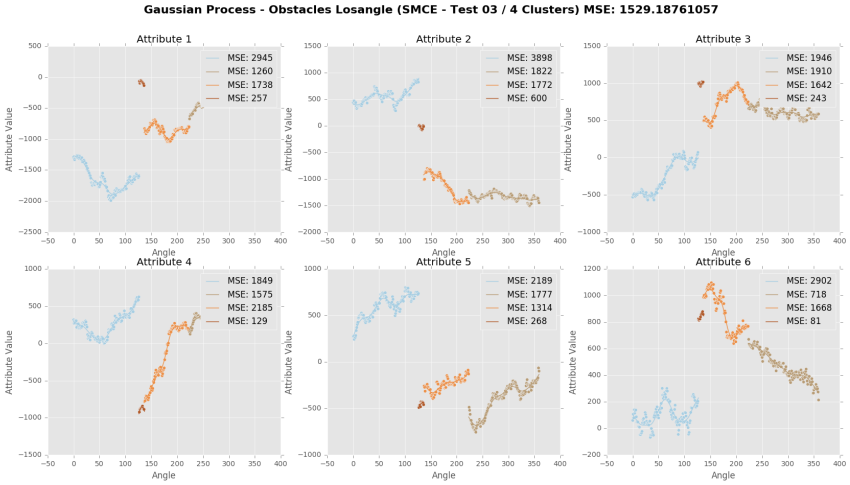


Figura 4.11: Processo Gaussiano com SMCE com o Conjunto de Dados 4 da Configuração com Obstáculos

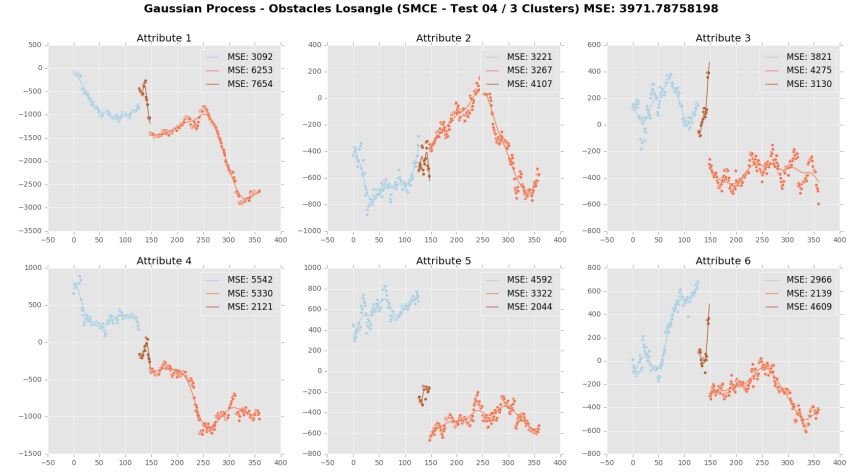


Figura 4.12: Processo Gaussiano com SMCE com o Conjunto de Dados 5 da Configuração com Obstáculos

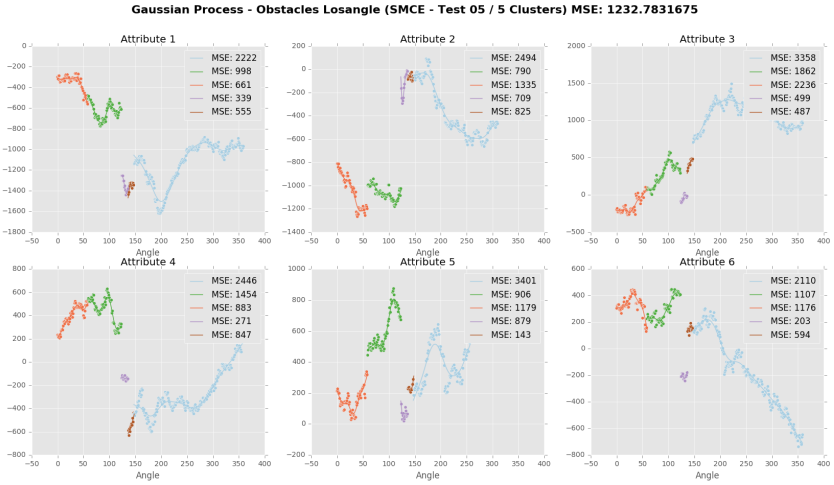


Figura 4.13: Processo Gaussiano com SMCE com o Conjunto de Dados 6 da Configuração com Obstáculos

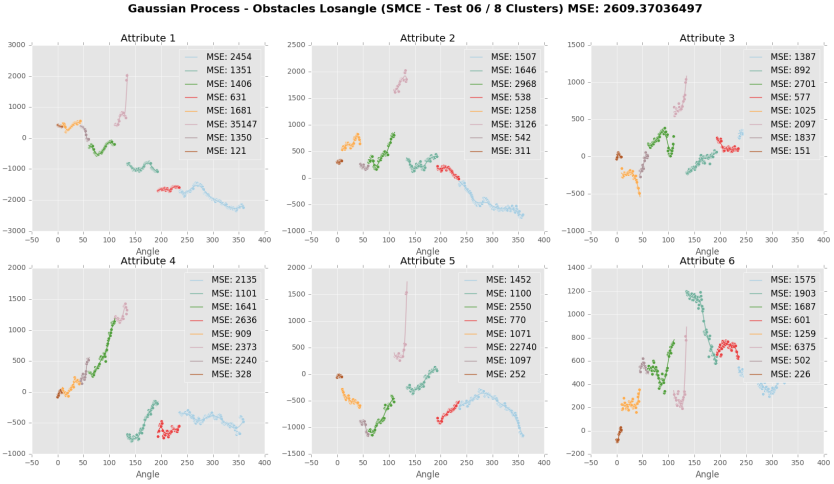


Figura 4.14: Processo Gaussiano com SMCE com o Conjunto de Dados 2 da Configuração sem Obstáculos

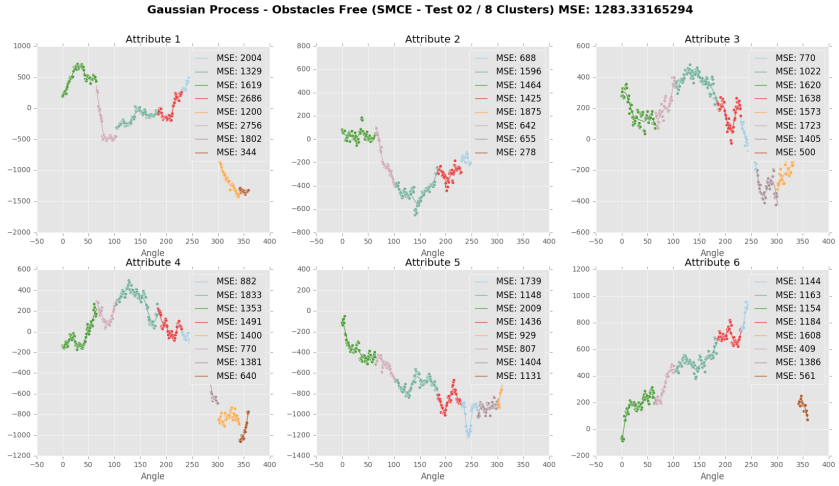
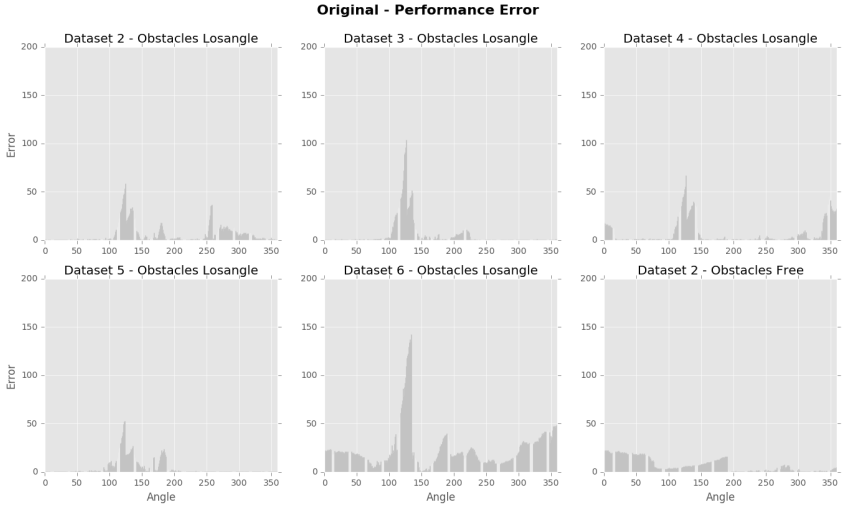


Figura 4.15: Erro de Performance dos Conjuntos de Dados Originais (em centímetros)



de performance dos conjuntos criados pelas regressões. Embora os erros de performance dos movimentos treinados através da regressão tenham sido mais altos que os erros de performance dos movimentos originais, eles não possuem uma diferença tão drástica na maior parte das tarefas, com exceção de alguns picos no conjunto 2 e 4 da configuração com obstáculos que, dada a baixa quantidade de ocorrências, poderiam ser resolvidos retreinando-os através do simulador, usando as predições feitas pela regressão como semente inicial do treinamento e executando-o, transformando tais soluções não-ótimas em soluções satisfatórias. Sendo assim, é possível dizer que os resultados das regressões são plenamente aceitáveis como uma forma de aproximação do comportamento ideal do robô, conseguido através do treinamento com o simulador.

Figura 4.16: Erro de Performance dos Conjuntos de Dados Previstos pelas Regressões (em centímetros)

4.3 Amostragem das Políticas usadas no Treinamento da Regressão

Naturalmente, o objetivo por traz da realização da regressão é que não seja necessário coletar um número tão grande de pontos a partir da otimização da DMP, já que essa é uma atividade custosa. Portanto, para que essa abordagem seja utilizada no treinamento com novas configurações de obstáculos, é necessário fazer alterações para que nem todos as tarefas sejam treinadas pelo simulador. Para isso, é importante fazer uma estimativa de quantos pontos são necessários em um conjunto de treinamento para que a curva treinada pela regressão não sofra com uma alta variância.

A qualidade do conjunto de treinamento usado na regressão é de extrema importância para que se consiga previsões de boa qualidade. Se o conjunto de treinamento não possui exemplos suficientes para que sejam reveladas todas as características da função (como mínimos e máximos locais em regiões de alta variância), o algoritmo não conseguirá reconhecer tais padrões e não os reproduzirá para novas entradas, deteriorando a performance. Um segundo perigo é ter um conjunto de treinamento que tenha um número excessivo de amostras redundantes de uma mesma região do domínio, que podem fazer com que novas predições tenham um viés muito alto em direção a tal comportamento, que pode ser inadequado para outras entradas.

4.3.1 Análise da Variância da Regressão

Uma das principais questões quando se trabalha com regressões é o balanço entre viés e variância. Supondo que um mesmo algoritmo de regressão seja executado repetidamente com vários conjuntos de treinamento (do mesmo problema) de igual tamanho e qualidade, porém diferentes entre si, é dito que o algoritmo de regressão têm um viés muito forte para uma determinada entrada x caso ele sistematicamente preveja uma saída incorreta para x, independentemente do conjunto de treinamento utilizado. Um alto viés ao longo do domínio do problema faz com que a predição falhe em identificar comportamentos relevantes da função à ser aprendida. Em outras palavras, causa underfitting. Paralelamente, quando, para uma determinada entrada x, a saída varia muito de acordo com o conjunto de treinamento utilizado, pode-se dizer que a regressão possui uma variância muito alta. A ocorrência de uma alta variância ao longo do dominínio indica que o modelo de regressão está muito sensível as variações no conjunto de treinamento. Essa variação pode indicar tanto que o tamanho do conjunto de treinamento utilizado é insuficiente para a complexidade do domínio, e então diferentes características estão sendo aprendidas a partir de cada conjunto de treinamento, como também pode indicar que o modelo está apresentando um overfitting muito elevado, que está moldando de maneira exagerada o resultado da regressão para que o erro de treinamento seja minimizado, e com isso, aprendendo ruído.

Idealmente, um modelo de regressão perfeito minimizaria tanto a variância quanto o viés do treinamento, ao mesmo tempo prevendo com precisão os valores pertencentes ao conjunto de treinamento quanto generalizando corretamente o comportamento da função para dados de entrada desconhecidos. Infelizmente isso não é possível na prática, fazendo com que a diminuiçao da variância do treinamento tenha como efeito colateral o aumento do viés e vice-versa. Isso faz com que seja importante balancear a variância e o viés de um treinamento, de forma que seja possível encontrar a melhor configuração para ele.

Assumindo que o conjunto de teste seja composto por um conjunto de entrada x associado à um conjunto de saída y e que tal conjunto seja composto por elementos de uma função ruidosa $y = f(x) + \varepsilon$ e que o objetivo da regressão é formular uma função aproximada de $f(x)$ chamada $g(x)$, a eficiência da regressão geralmente é medida através de uma comparação entre os valores de saída previstos $g(x)$ e originais y para um dado conjunto de testes. Uma das métricas mais utilizadas para isso é o erro quadrático médio ($MSE = E((y - g(x))^2)$). O erro de um modelo de regressão pode ser dividido em três

fontes: viés, variância e ruído. Enquanto o ruído causa um erro irredutível que não pode ser precisamente medido, é possível decompor o *MSE* de forma que se identifique o erro causado por um alto viés e o erro causado por uma alta variância. As equações em 4.3, 4.4 e 4.5 mostram a decomposição do cálculo do erro nas três componentes mencionadas. Nelas, E é definida como a esperança de uma variável.

$$E[(y-g(x))^2] = Bias[g(x)]^2 + Variance[g(x)] + \sigma^2 \tag{4.3}$$

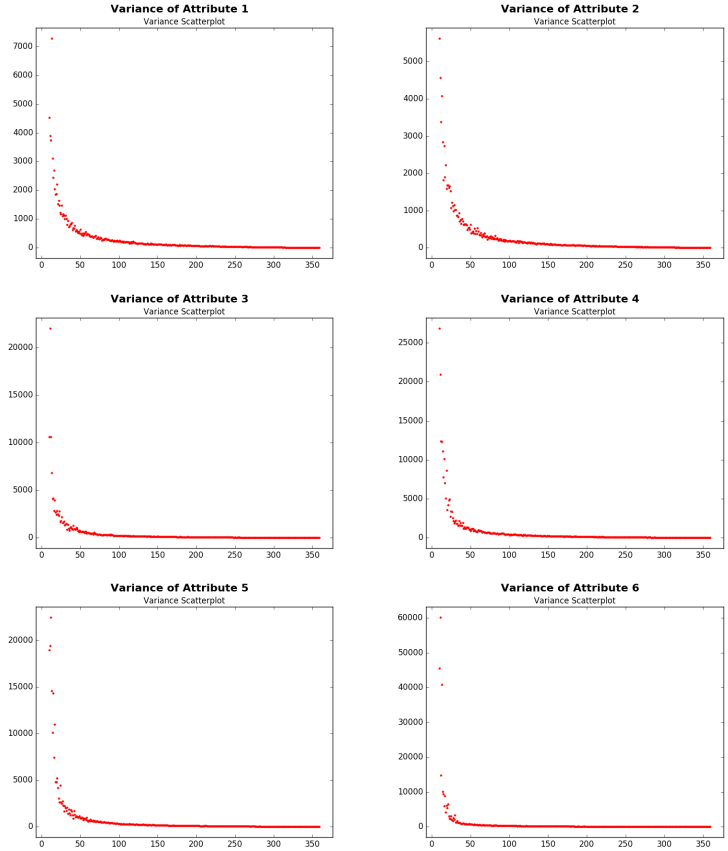
$$Bias[g(x)] = E[g(x) - f(x)] \tag{4.4}$$

$$Variance[g(x)] = E[g(x)^2] - E[g(x)]^2 \tag{4.5}$$

Como mostrado na Equação 4.4, o viés, para uma determinada entrada x, é medido na decomposição do erro como o valor médio da diferença entre o valor previsto $g(x)$ e o valor real $f(x)$ dentre as diferentes regressões. Valores distantes de zero indicam que, independentemente do conjunto de treinamento utilizado, a média das predições está se afastando consideravelmente (e para uma mesma direção) do valor ideal. Na prática, não é possível conhecer o valor real de $f(x)$, então é necessário usar y no seu lugar.

Para estimar o número de amostras necessárias para que se tenha uma baixa variância, a partir de um conjunto de dados de um treinamento sem obstáculos com políticas para todos os 360 ângulos alvo, foi calculada a variância que uma regressão teria caso fossem usadas $S = 10, ..., 360$ amostras. Para cada valor de S, foram retiradas do conjunto de treinamento 100 conjuntos de S amostras e foi realizada uma regressão para cada um destes conjuntos, calculando a variância média das regressões para cada valor de S. A Figura 4.17 mostra o gráfico da variância para cada atributo. Ao se fazer a análise do gráfico, foi possível notar que a variância se estabiliza para todos os ângulos com um número de amostras em torno de 80 e 90 amostras. Este é um limite superior, pois na prática, uma boa performance pode ser alcançada com menos amostras e, de fato, foram feitos com 45 amostras que trouxeram resultados satisfatórios. Contudo, o fato do limite superior ser alto indica que a superfície de políticas realmente é complexa e que para que a regressão não seja muito sensível ao conjunto específico de exemplos, a amostragem das tarefas precisa ter uma resolução alta.

Figura 4.17: Variância em função do número de amostras



4.3.2 Amostragem das Políticas Treinadas pela Otimização da DMP

Baseando-se nisso, decidiu-se por treinar, usando o simulador, 1 a cada 4 ângulos, totalizando 90 pontos alvo ao longo da circunferência treinados pela otimização de DMP. O procedimento de treinamento seguiu-se como descrito em 4.1, onde se usa como semente a política do treinamento anterior, que agora é o ângulo $\theta - 4$, refazendo o treinamento com uma semente aleatória caso ele não seja satisfatório. Coletados os 90 pontos, eles são clusterizados através do SMCE e cada *cluster* é usado como entrada de uma regressão por Gaussian Process, de forma a obter uma estimativa dos valores das políticas para as tarefas não treinadas com o simulador.

A Figura 4.18 mostra um conjunto colhido através desse processo de treinamento, com os dados já clusterizados pelo SMCE, enquanto a Figura 4.19 mostra o erro de performance deste conjunto. As Figuras 4.20 e 4.21 mostram a regressão e o erro de performance das políticas previstas pela regressão. Como foram cortadas um quarto do treinamento por otimização de DMPs e o tempo necessário para fazer a clusterização e regressões é despresível perto do tempo necessário para treinar uma DMP, é possível

Figura 4.18: Conjunto de Dados Treinados pelo Simulador com 90 Pontos

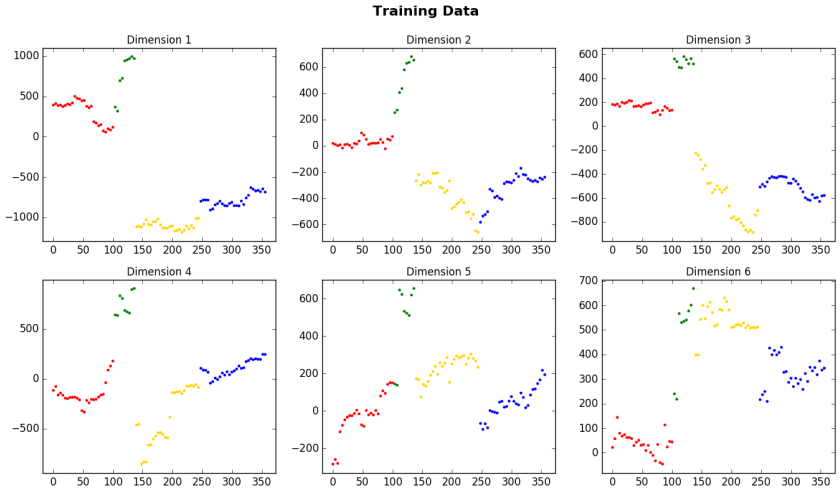
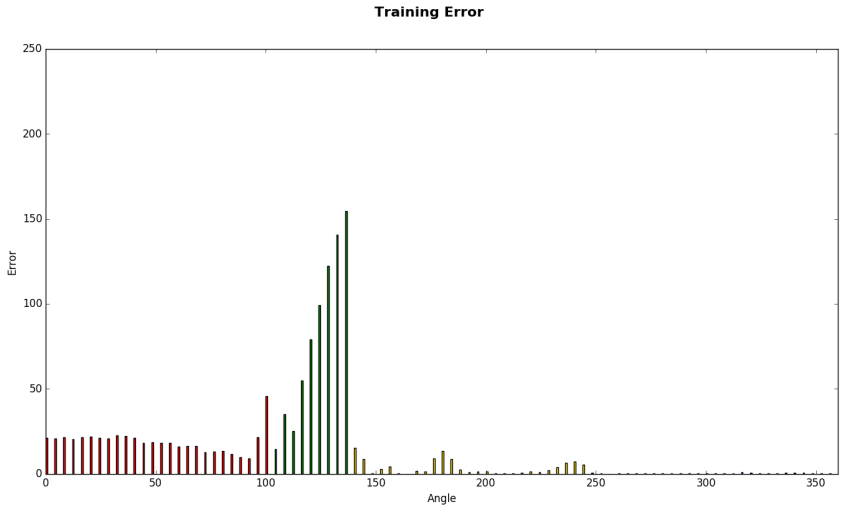


Figura 4.19: Erro de Performance do Conjunto com 90 Pontos (em centímetros)



afirmar que o novo método de generalização gasta apenas um quarto do tempo necessário originalmente. O número médio de *manifolds* encontrados também não mudou ao se utilizar menos pontos, ficando entre 4 e 5 *manifolds* encontrados por conjunto de dados.

4.4 Novas Configurações de Obstáculos

Como validação de que o método proposto funciona para novos treinamentos, o aprendizado das 360 tarefas foi executado para três novas configurações de obstáculos, com um crescente nível de dificuldade no aprendizado, devido ao posicionamento e quantidade dos obstáculos criados. Os três aprendizados foram realizados com uma

Figura 4.20: Regressão Realizada a Partir dos 90 Pontos

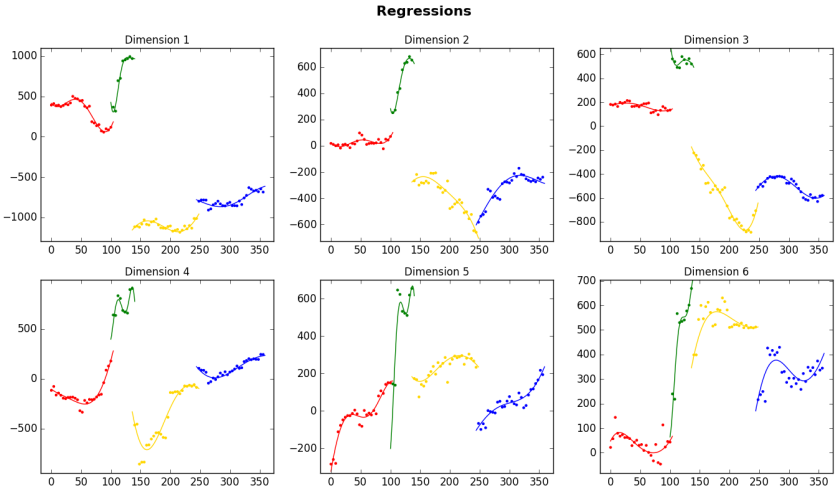


Figura 4.21: Erro de Performance das Previsões das Regressões (em centímetros)

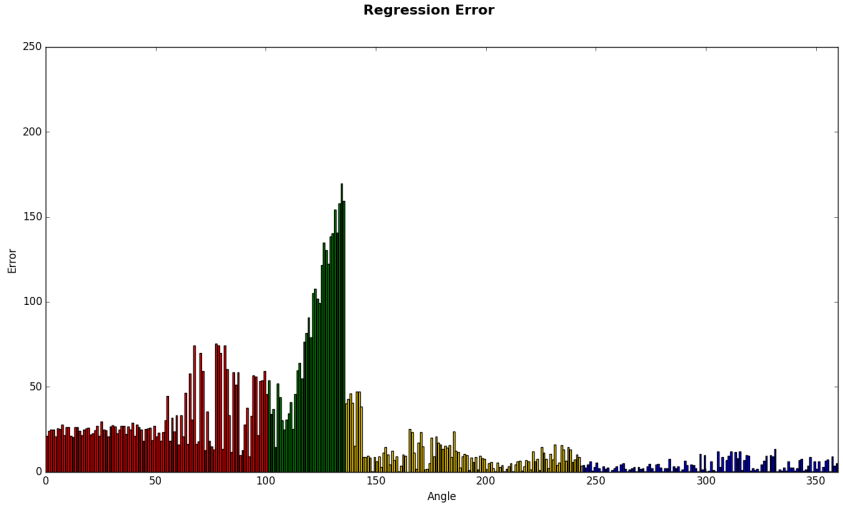


Figura 4.22: Primeiro Teste - Configuração dos Obstáculos

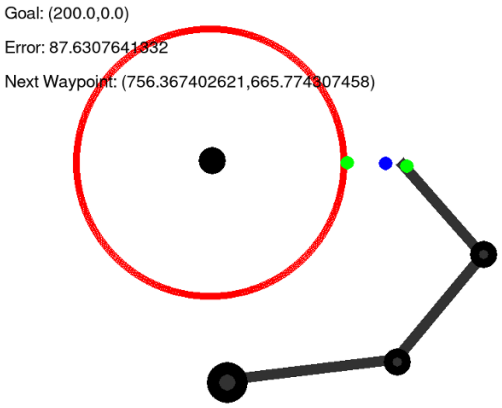
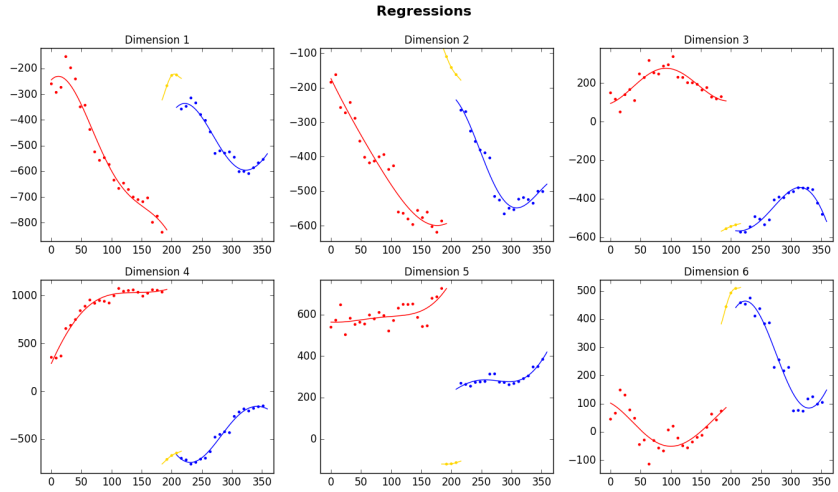


Figura 4.23: Primeiro Teste - Amostragem, Clusterização e Regressão



amostragem total de 45 políticas treinadas pela otimização da DMP.

A primeira configuração de obstáculos testada é composta por um único obstáculo no meio da circunferência, como mostrado na Figura 4.22. O SMCE encontrou três *manifolds* ao longo do conjunto de dados. A Figura 4.23 mostra o gráfico do conjunto de políticas amostradas já com a clusterização e a curva de regressão treinada pelo processo gaussiano. Na Figura 4.24 é possível comparar o erro de performance das amostras com o erro de performance de todas as 360 tarefas após feitas as regressões. É possível perceber que a diferença entre a performance das amostras e das previsões é baixa, indicando que o algoritmo de generalização teve resultados satisfatórios.

A segunda configuração de obstáculos treinada foi formada por quatro obstáculos dispostos dentro do círculo formando os vértices de um quadrado. Tal configuração se mostrou mais difícil de ser aprendida pela otimização da DMP, resultando na ocorrência de um grande número de *manifolds* e de erros de performance altos em algumas regiões.

Figura 4.24: Primeiro Teste - Comparação do Erro de Performance (em centímetros)

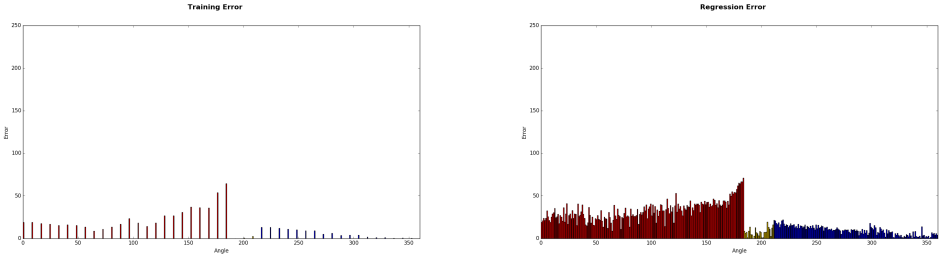
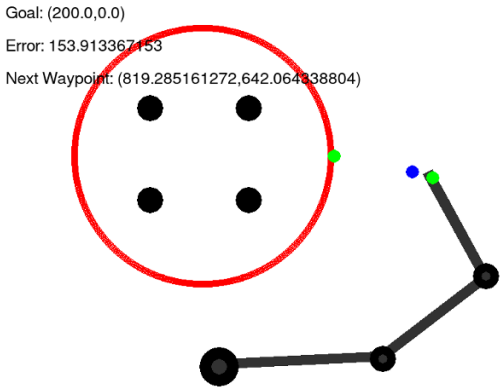


Figura 4.25: Segundo Teste - Configuração dos Obstáculos



Contudo, o método de generalização se comportou bem, apresentando performances condizentes com os erros da amostragem, apesar de possuir picos ocasionais. As Figuras 4.25, 4.26 e 4.27 mostram, respectivamente, a configuração dos obstáculos, amostragem das políticas e performance das amostras e das predições.

A terceira, e mais complexa, configuração de obstáculos treinada consiste em oito obstáculos dispostos em uma circunferência interna à circunferência alvo. Como pode ser visto nas Figuras 4.28, 4.29 e 4.30, a generalização foi eficaz na maior parte das tarefas. Contudo, essa configuração apresentou uma piora considerável nas performances das predições, quando comparadas com as performances das amostras, em algumas regiões, especialente no *cluster* vermelho. Estas provavelmente são regiões difíceis de serem treinadas e não aceitam uma variação muito alta nos parâmetros para que tenha bons resultados, fazendo com que a política treinada pela regressão possa não ter uma performance tão satisfatória.

Figura 4.26: Segundo Teste - Amostragem, Clusterização e Regressão (em centímetros)

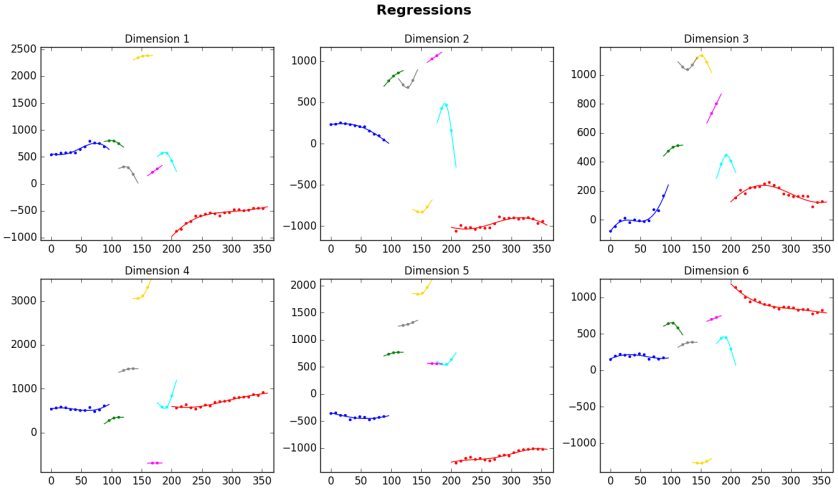


Figura 4.27: Segundo Teste - Comparação do Erro de Performance

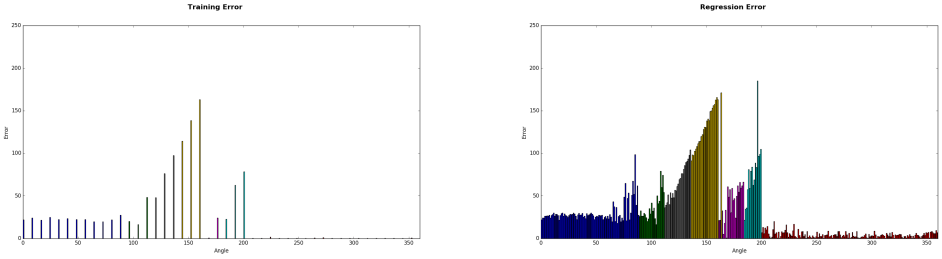


Figura 4.28: Terceiro Teste - Configuração dos Obstáculos

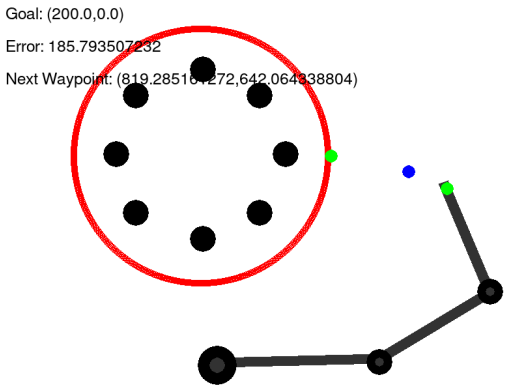


Figura 4.29: Terceiro Teste - Amostragem, Clusterização e Regressão

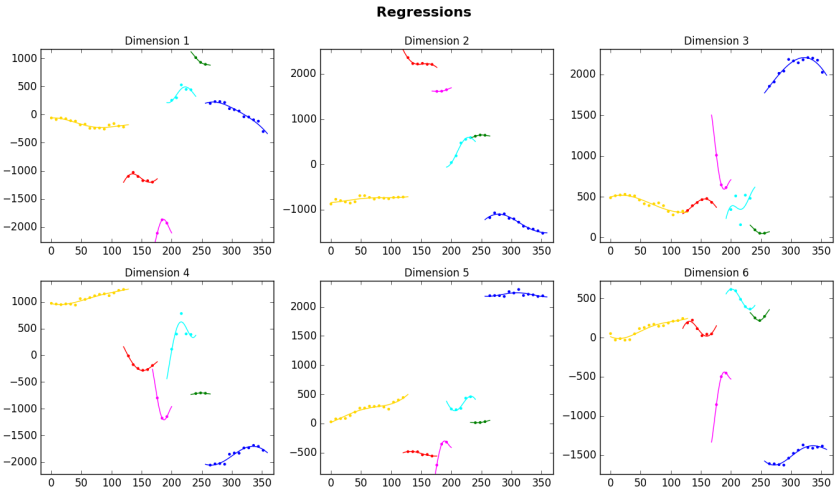
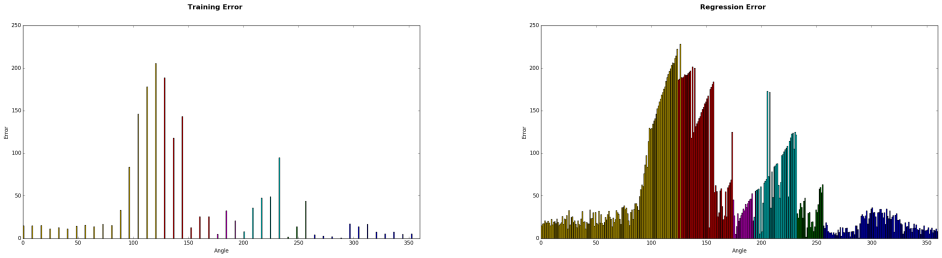


Figura 4.30: Terceiro Teste - Comparação do Erro de Performance (em centímetros)



5 CONCLUSÃO

Ao longo deste trabalho, a análise e visualização de dados se mostrou de grande importância para aumentar a compreensão do comportamento e das características presentes em conjuntos de dados multidimensionais. Consequentemente tornando-se uma ferramenta eficiente em ajudar a escolher as propriedades necessárias em um algoritmo de aprendizado de máquina para que ele consiga lidar com dados deste tipo. Através dela, foi possível identificar formas de generalizar o comportamento do aprendizado, diminuindo o custo computacional do aprendizado sem prejudicar drasticamente a performance do mesmo. A análise também permitiu encontrar relações importantes como a existência de mínimos locais não-ótimos sempre que os movimentos ótimos estão muito restritos à uma pequena variação da configuração das políticas de treinamento.

O algoritmo de generalização dos movimentos inspirado pelos resultados desta análise também se mostrou eficaz, conseguindo, na maior parte dos casos, prever soluções com performance próximas das que seriam geradas pela otimização da DMP, apenas trocando um pouco da precisão na performance por uma grande diminuição no tempo de treinamento, o que dado o custo computacional exigido para o treinamento completo de cada tarefa, é um ganho bem alto.

Os experimentos feitos neste trabalho indicaram várias características importantes que métodos de aprendizado de máquina, especialmente os relacionados à generalização de comportamentos, precisam levar em consideração, como a capacidade de identificação de manifolds e subespaços existentes nos conjuntos de treinamento, além de ser possível identificar e lidar com propriedades do formato dessas estruturas como linearidade e dimensão intrínseca.

Como trabalho futuro, existem diversas modificações que podem ser feitas para aperfeiçoar o método de treinamento. Uma delas seria extender o algoritmo de generalização para que ele possa prever posições alvo em qualquer ponto do espaço bidimensional, e não apenas ao longo de uma circunferência. O maior desafio para que isso seja possível seria coletar as amostras de forma que desse pra garantir continuidade em ambas as direções do plano, fazendo com que o conjunto de treinamento para a regressão fique contínuo. Isso exigiria uma nova estratégia para escolher a semente de inicialização da otimização da DMP, já que escolher um vizinho horizontal pode acarretar em descontinuidades no eixo vertical e vice-versa. Outra modificação a ser pensada seria como diminuir o número de parâmetros necessários para definir uma política, de forma que ela fique

compatível com o grau de liberdade real das tarefas.

Uma característica do treinamento que ficou clara pelas análises é que, muitas vezes, a política treinada para um ponto alvo pode ser usada diretamente para um ponto alvo vizinho, sem que haja mudança significativa na performance. Levando isso em consideração, seria possível fazer uma análise de qual é o ”alcance” de cada política, de forma que aconteça o reuso de políticas sempre que possível. Por fim, fazer modificações na base do algoritmo de treinamento, para que ele evite cair em mínimos locais de baixa performance também seria um possível trabalho para o futuro.

REFERÊNCIAS

AUGER, A.; HANSEN, N. Tutorial cma-es: Evolution strategies and covariance matrix adaptation. In: **Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2012. (GECCO '12), p. 827–848. ISBN 978-1-4503-1178-6. Available from Internet: <http://doi.acm.org/10.1145/2330784.2330919>.

ELHAMIFAR, E.; VIDAL., R. Sparse manifold clustering and embedding. In: **Neural Information Processing and Systems**. [S.l.: s.n.], 2011.

IZENMAN, A. J. Introduction to manifold learning. **Wiley Interdisciplinary Reviews: Computational Statistics**, John Wiley and Sons, Inc., v. 4, n. 5, p. 439–446, 2012. ISSN 1939-0068. Available from Internet: <http://dx.doi.org/10.1002/wics.1222>.

LUXBURG, U. von. A tutorial on spectral clustering. **CoRR**, abs/0711.0189, 2007. Available from Internet: <http://arxiv.org/abs/0711.0189>.

MAATEN, L. van der; HINTON, G. E. Visualizing high-dimensional data using t-sne. **Journal of Machine Learning Research**, v. 9, p. 2579–2605, 2008.

RASMUSSEN, C. E.; WILLIAMS, C. K. I. **Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)**. [S.l.]: The MIT Press, 2005. ISBN 026218253X.

ROWEIS, S. T.; SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. **SCIENCE**, v. 290, p. 2323–2326, 2000.

SCHAAL, S. **Dynamic Movement Primitives–A Framework for Motor Control in Humans and Humanoid Robotics**.

SCIKIT Learn. <http://scikit-learn.org/stable/index.html>. Accessed: 2016-11-30.

SCIKIT Learn - Underfitting vs. Overfitting. <http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html>. Accessed: 2016-11-30.

SHLENS, J. A tutorial on principal component analysis. In: **Systems Neurobiology Laboratory, Salk Institute for Biological Studies**. [S.l.: s.n.], 2005.

SILVA, B. C. D.; KONIDARIS, G.; BARTO, A. G. Learning parameterized skills. In: **in: Proceedings of International Conference of Machine Learning**. [S.l.: s.n.], 2012.

TENENBAUM", J. **A Global Geometric Framework for Nonlinear Dimensionality Reduction**. 2000.

VIDAL, R. Subspace clustering. **IEEE Signal Processing Magazine**, v. 28, n. 2, p. 52–68, March 2011. ISSN 1053-5888.

XU, D.; TIAN, Y. A comprehensive survey of clustering algorithms. **Annals of Data Science**, v. 2, n. 2, p. 165–193, 2015. ISSN 2198-5812. Available from Internet: <http://dx.doi.org/10.1007/s40745-015-0040-1>.