REDUÇÃO DE CUSTO NA GERAÇÃO DE PRIMITIVAS DE MOVIMENTO ATRAVÉS DE OTIMIZAÇÃO POR ELIMINAÇÃO

Hiparco Lins Vieira* , Valdir Grassi Jr* , Maíra Martins da Silva*

*Escola de Engenharia de São Carlos - Universidade de São Paulo
Av. Trabalhador São-carlense, 400
São Carlos, São Paulo, Brasil

Emails: hiparcolins@usp.br, vgrassi@usp.br, mairams@sc.usp.br

Abstract— This paper describes a method used to reduce the computacional cost of the motion primitives generation for autonomous robots. To accomplish this task, an elimination optimization method was applied in the primitives generation, together with some heuristics. In the method proposed, only the primitives inclined to produce low cost motions are generated. By comparing it with the traditional and probabilistic method for primitives generation, the obtained results showed, in most of the cases, reduced computational cost without compromising solution's quality.

Keywords— Motion planning, motion primitives, optimization, autonomous robots, primitives generation

Resumo— Este trabalho descreve um método utilizado para gerar primitivas de movimento para robôs autônomos com menor custo computacional em relação ao método tradicional e ao método de geração de primitivas aleatórias. Para isto, foi empregado um método de otimização por eliminação na geração das primitivas em conjunto com algumas heurísticas. No método apresentado, são geradas apenas primitivas de movimento que aparentam ser mais promissoras na obtenção de uma solução de baixo custo sem gerar colisão com obstáculos. Tal método apresentou significante redução do custo computacional mantendo o baixo custo da solução obtida em relação aos métodos comparados.

Palavras-chave— Planejamento de movimento, primitivas de movimento, otimização, robôs autônomos, geração de primitivas

1 Introdução

Em muitas aplicações na robótica, faz-se necessário realizar o planejamento de trajetórias que levem em consideração as restrições cinemáticas e dinâmicas do robô. Dentre as várias abordagens existentes para tais aplicações, estão os métodos baseados em amostragem, tais como malhas de estado (*State Lattice*)(Pivtoraiko and Kelly, 2011) e árvores de crescimento rápido (*Rapidly-exploring Random Trees* - RRTs)(LaValle, 1998).

Para a geração de trajetórias onde as restrições de movimento do robô sejam respeitadas, os algoritmos RRT e Malha de Estados utilizam as chamadas primitivas de movimento. Tais primitivas são geradas a partir da amostragem do espaço de entradas do robô, a fim de gerar um conjunto de estados, calculados através das equações cinemáticas e dinâmicas do robô. Cada uma dessas conexões representa uma trajetória alcançável pelo robô.

As primitivas de movimento tem sido utilizadas por algoritmos de planejamento de trajetória em diversos segmentos da robótica, dentre os quais destacam-se em robôs exploradores (Pivtoraiko et al., 2009), robôs terrestres (Frazzoli et al., 2001), robôs humanóides (Kris Hauser and claude Latombe, 2008), robôs aéreos (C. L. Bottasso and Savini, 2008), manipuladores robóticos (Cohen et al., 2010) e veículos autônomos (Likhachev and Ferguson, 2009), (Magalhães et al., 2013).

Contudo, na discretização do espaço de configuração do robô, mesmo que tal discretização seja realizada com uma alta taxa, o robô perde parte de sua mobilidade (Pivtoraiko et al., 2009). Para aplicações com vários graus de liberdade, o problema torna-se ainda maior. Caso sejam geradas várias primitivas, a fim de dar mais mobilidade ao robô, aumenta-se drasticamente a complexidade computacional do problema, em virtude da necessidade de geração de um maior número de primitivas, análises de colisão e de custo. Caso sejam geradas poucas primitivas, diminui-se ainda mais a mobilidade do robô. Desta forma, embora o custo computacional do problema torne-se menor, a perda de mobilidade no robô pode ser tamanha que, mesmo havendo uma solução para o problema, o algoritmo não consiga encontrá-la.

Em (Lavalle and Konkimalla, 2001) e (LaValle, 2003), foram mostrados métodos numéricos baseados em simulação e interpolação para a geração de primitivas com menor custo computacional em relação aos métodos tradicionais de interpolação. A otimização é realizada nas entradas e juntamente é avaliado se as primitivas geradas resultam em colisão.

A contribuição do presente trabalho consiste em demonstrar um novo método para geração de primitivas de movimento, com custo computacional inferior ao método tradicional e ao método aleatório de geração de primitivas, mantendo-se a qualidade da solução obtida. O método que está sendo proposto apresenta algumas similaridades com os métodos numéricos propostos em (Lavalle

and Konkimalla, 2001) e (LaValle, 2003). A primeira similaridade está no fato de se utilizar a interpolação para gerar novas primitivas. A segunda consiste na avaliação das colisões com obstáculos durante a geração das primitivas.

O método apresentado no presente trabalho, contudo, não utiliza a interpolação baseada na subdivisão completa do baricentro. Ao invés disso, utiliza-se um método de otimização por eliminação (Rao, 2009) e heurísticas na geração das primitivas de movimento. O método de otimização por eliminação é comumente utilizado quando a solução ótima para um problema está dentro de um intervalo conhecido, realizando-se uma hipótese inicial para a solução e deslocando tal hipótese um certo passo em uma direção favorável à obtenção da solução ótima. Durante este processo, os intervalos menos favoráveis à obtenção de uma solução ótima vão sendo eliminados do intervalo de busca.

Para mostrar a eficiência do método proposto, é realizada uma comparação do método proposto com dois métodos de geração de primitivas: o método tradicional (onde as primitivas são geradas pela simples combinação das entradas fisicamente possíveis do sistema) e o método de geração aleatória (onde as primitivas são geradas à partir de entradas aleatórias).

Na Seção 2, é realizada a descrição do problema. Na Seção 3 é realizada uma breve explanação sobre os métodos tradicional e aleatório de geração de primitivas de movimento. Na Seção 4, é apresentado o algoritmo proposto e na Seção 5, são apresentados os resultados do trabalho.

2 Descrição do problema

Considera-se um robô com equações de estado definidas por:

$$\dot{x} = f(x, u) \quad (1)$$

onde $x \in \Re^m$ é o vetor de estados de dimensão m do sistema, $u \in \Re^k$ é o vetor de entradas de dimensão k e $f(.)$ é uma função que representa as equações cinemáticas e dinâmicas do robô e relaciona o vetor x e o vetor u com o vetor $\dot{x}$.

Insere-se, então, o robô em uma posição inicial p_i em um ambiente com obstáculos estáticos circulares gerados aleatoriamente. O robô tem como objetivo gerar um grupo de primitivas de movimento e, dentre estas, retornar aquela primitiva e a respectiva entrada de controle que esteja livre de colisão e possua o menor custo de navegação em relação à uma configuração final p_f desejada. O custo de cada primitiva é calculado através da seguinte equação:

$$C = g(x, p_f) \quad (2)$$

onde C é o custo e $g(.)$ é a função que relaciona o custo C à x e p_f.

3 Métodos tradicional e aleatório de geração de primitivas de movimento

Para a geração de primitivas no método tradicional, combina-se um conjunto de valores fisicamente possíveis que cada componente do vetor u pode assumir dado o estado atual x do robô. Em seguida, utiliza-se cada combinação das entradas, juntamente com o estado x do robô, para calcular, através de uma integração, os estados do robô após um determinado intervalo de tempo $\triangle t$ utilizando-se a Equação 1. Dentre as primitivas obtidas, avalia-se aquela de menor custo de navegação em relação à p_f e que não gere colisão com obstáculos. A primitiva de menor custo e livre de colisão é retornada pelo algoritmo.

Na geração de primitivas de forma aleatória, gera-se para cada componente do vetor u de entrada valores aleatórios fisicamente alcançáveis pelo robô dado o estado atual do mesmo. Assim como no método tradicional, as primitivas são calculadas utilizando-se as combinações das entradas geradas e o estado x do robô, utilizando-se a Equação 1. Dentre as primitivas geradas, aquela de menor custo de navegação em relação à p_f e livre de colisões é retornada pelo algoritmo.

O número de primitivas geradas em cada algoritmo está diretamente relacionada com a quantidade de valores discretos de cada componente do vetor u, visto que as primitivas são geradas através da combinação de tais valores. No presente trabalho, ambos algoritmos gerarão o mesmo número de primitivas de movimento.

4 Algoritmo proposto para geração de primitivas de movimento

No Algoritmo 1 é possível ver o pseudocódigo do algoritmo proposto para a geração de primitivas. Tal algoritmo gera, para cada uma das k componentes do vetor u e a partir do intervalo $I_k = [u_{k,min} \quad u_{k,max}]$ de valores de entrada fisicamente possíveis de serem aplicados em um intervalo de tempo $\triangle t$, um vetor com três valores u_{k1}, u_{k2} e u_{k3}, onde $u_{k1} = u_{k,min}$, $u_{k2} = \frac{u_{k,min}+u_{k,max}}{2}$ e $u_{k3} = u_{k,max}$. Tal tarefa é executada pela função $InterpolaEntradas$ (linha 4).

Em seguida, é realizada uma combinação das k componentes do vetor u, a fim de gerar as primitivas de movimento. Como cada componente possui três valores diferentes, são geradas 3^k combinações. Tal tarefa é executada pelo função $CombValores$ (linha 5).

Posteriormente (linhas 7-11), as primitivas de movimento são calculadas pela função $RK4$, através de integração numérica, e os respectivos custos de navegação em relação a p_f são calculados

pela função *Custo*. Avalia-se, então, se as primitivas geradas causam colisão, através da função *ChecaColisao*.

Inicia-se, então, o cálculo da influência de cada u_{kx} na qualidade das 3^{k-1} primitivas que ajudou a gerar (linhas 13-16). Tal qualidade é avaliada quanto ao número de colisões e ao custo de cada primitiva gerada. Assim, para cada componente do vetor u, são calculadas as influências nos custos ($ICusto$) e nas colisões ($IColisao$). O primeiro valor é a soma dos custos das 3^{k-1} primitivas, calculado pelo função $InfCusto$. O segundo valor é o número de primitivas, dentre as 3^{k-1} primitivas, que não geram colisão com obstáculos, calculado pela função $InfColisao$.

Depois, inicia-se uma avaliação das variáveis $ICusto$ e $IColisao$ (linhas 19-23) utilizando a função *Condicao*, com o intuito de eliminar os valores u_{k1} ou u_{k3} que pior contribuam para a obtenção da primitiva de menor custo e livre de colisões.

Para avaliar qual dos dois valores u_{k1} e u_{k3} é mais promissor, avalia-se os valores de $ICusto$ e $IColisao$ de cada um. Analisando-se os valores u_{k1} e u_{k3}, escolhe-se como mais promissor aquele que possuir menor $ICusto$ e possuir juntamente com u_{k2} pelo menos uma primitiva gerada livre de colisões. O valor menos promissor é eliminado. Por exemplo, se u_{k1} se mostrar mais promissor na obtenção de uma melhor solução que u_{k3}, então $Intervalo = [u_{k1} \quad u_{k2}]$ e $Flag = 1$.

Se ambos possuírem o mesmo $ICusto$, o mais promissor é determinado como aquele que possuir, juntamente com u_{k2}, maior número de primitivas livres de colisão. Se ainda assim houver empate, escolhe-se aleatoriamente entre u_{k1} e u_{k3} qual será eliminado. Aquele que, juntamente com u_{k2}, estiver com todas as primitivas em colisão é eliminado e, caso ambos sejam eliminados, significa que para tal entrada u_k, nenhum dos três valores u_{kx} é capaz de produzir uma primitiva livre de colisão. Em tal caso, $Intervalo = NULL$ e $Flag = 0$, encerrando a execução dos laços das linhas 3 e 19, além de fazer o algoritmo retornar a ausência de solução (linhas 19-31).

Havendo solução para todos u_k, caso $Fim >= 1$ e $Flag == 1$, executa-se o laço iniciado na linha 3 até que este seja encerrado, reduzindo-se assim, a cada iteração, o intervalo de busca e melhorando-se a qualidade da solução.

Tal avaliação é realizada pela função $MelhorPrim$ (linha 28), onde são avaliadas as primitivas geradas que não tiveram nenhum de seus valores u_{kx} eliminados. Aquela primitiva de menor custo dentre estas últimas e que não estiver em colisão é selecionada pela função, que retorna para a função principal, a primitiva e as respectivas entradas de controle. Tais dados são, então, retornados pelo algoritmo.

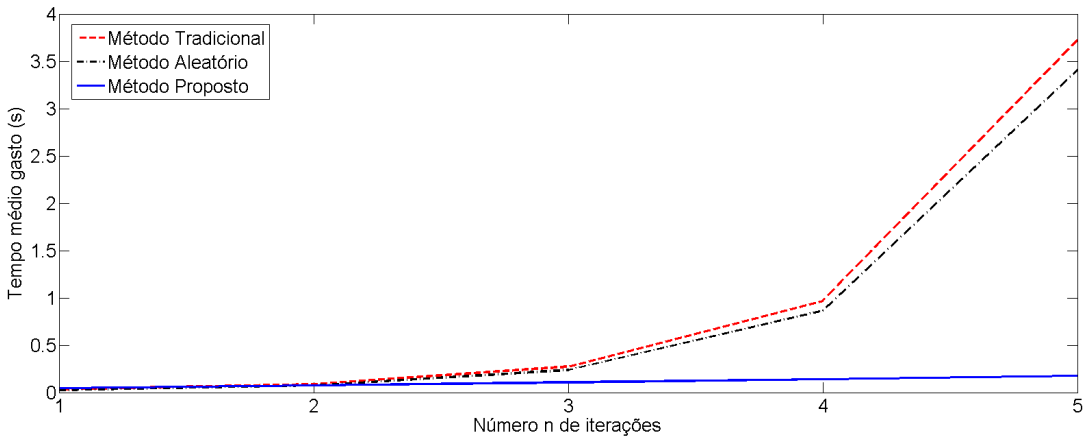
Algoritmo 1: Geração de Primitivas por Eliminação

1: $Fim \leftarrow n$
2: $Flag \leftarrow 1$
3: **enquanto** $(Fim >= 1)$ e $(Flag == 1)$ **faça**
4: $\quad Interp \leftarrow InterpolaEntradas(Intervalo)$
5: $\quad Comb \leftarrow CombValores(Interp)$
6:
7: $\quad$ **para** $i \leftarrow 1$ **até** 3^k **faça**
8: $\quad\quad Prim(i) \leftarrow RK4(Comb(i), \triangle t, p_i)$
9: $\quad\quad CostPrim(i) \leftarrow Custo(Prim(i), p_f)$
10: $\quad\quad Colisao(i) \leftarrow ChecaColisao(Prim(i))$
11: $\quad$ **fim para**
12:
13: $\quad$ **para** $i \leftarrow 1$ **até** k **faça**
14: $\quad\quad ICusto(i) \leftarrow InfCusto(CostPrim, i)$
15: $\quad\quad IColisao(i) \leftarrow InfColisao(Colisao, i)$
16: $\quad$ **fim para**
17:
18: $\quad a \leftarrow 1$
19: $\quad$ **enquanto** $((Flag == 1)$ ou $(a \leq k))$ **faça**
20: $\quad\quad [Intervalo(a) \quad Flag] \leftarrow Condicao(ICusto(a), IColisao(a))$
21: $\quad\quad a \leftarrow a + 1$
22: $\quad$ **fim enquanto**
23:
24: $\quad Fim = Fim - 1$
25: **fim enquanto**
26:
27: **se** $(Flag \neq 0)$ **então**
28: $\quad$ **devolve** $MelhorPrim(Intervalo)$
29: **senão**
30: $\quad$ **devolve** $NULL$
31: **fim se**

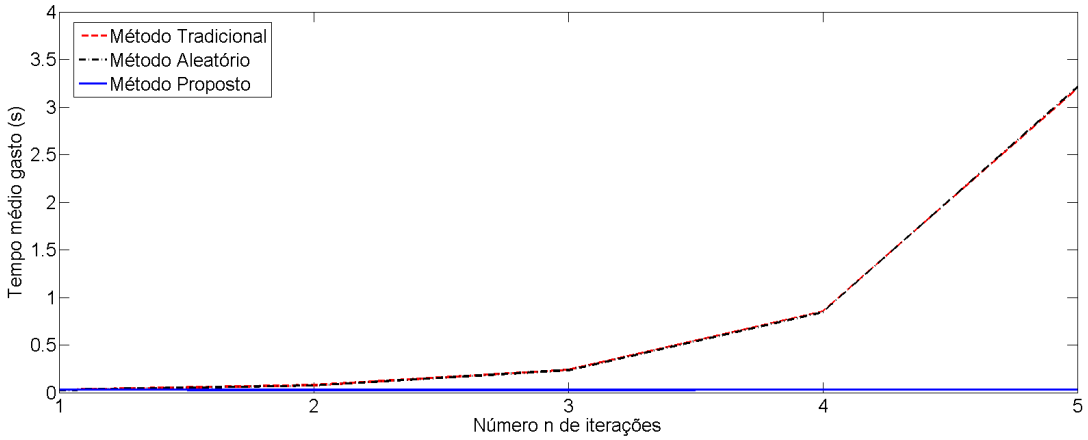
5 Resultados

As simulações apresentadas foram realizadas utilizando o software Matlab®, em um computador com processador Core i7, 3.40GHz e 8Gb de memória RAM.

O modelo cinemático-dinâmico utilizado para o robô teve como base o trabalho de (Monet, 2003). As entradas do sistema são a velocidade v e o ângulo de esterçamento ϕ, ambos com variação linear no tempo e mesmo número de amostras. Os estados do sistema são definidos por $X = [x \quad y \quad \theta]'$, onde (x, y) é a posição do robô e θ é a orientação do mesmo. Para realizar a integração das equações cinemáticas e dinâmicas do robô, utilizou-se o método de Runge-Kutta de quarta ordem, como mostrado em (Pepy et al., 2006). O custo de cada primitiva gerada é avaliado em relação à distância da configuração gerada a partir da primitiva e da configuração final x_f desejada para o robô.



(a)



(b)

Figura 1: Gráfico do aumento do tempo computacional em função do número de iterações n realizadas, comparando-se o método tradicional, proposto e aleatório. (a) Gráfico para estados onde todas as primitivas geradas estão livres de colisão. (b) Gráfico para estados onde todas as primitivas geradas causam colisão.

5.1 Avaliação do custo computacional

Na Figura 1(a), pode-se avaliar o custo computacional médio versus o número de iterações n do algoritmo proposto, comparando-se o algoritmo proposto, o algoritmo aleatório e o algoritmo tradicional de geração de primitivas e, na Tabela 1, são mostrados os desvios padrão. Em tal simulação, nenhuma das primitivas geradas pelos três algoritmos geravam colisão com obstáculos, visando-se avaliar o custo dos algoritmos para tais situações. Cada um dos três algoritmos foi executado 100 vezes. Na simulação, foi obtido o tempo médio da geração de primitivas por cada método a partir de um estado inicial. Para o cálculo da função custo, foi estipulada uma configuração p_f aleatória de destino.

Inicialmente, para a divisão de cada entrada em 3 amostras ($n = 1$), o algoritmo proposto mostrou-se aproximadamente 1,25 e 1,66 vezes mais custoso que os métodos tradicional e aleatório, respectivamente. Quando o número de amostras por entrada passa a ser igual a 5 ($n = 2$), o algoritmo proposto já se mostra menos custoso que o algoritmo tradicional e aproximadamente 2% mais custoso que o método aleatório.

Tabela 1: Desvios padrão em ms dos métodos tradicional, aleatório e proposto de geração de primitivas, obtidos a partir da simulação mostrada na Figura 1(a).

Método	n = 1	n = 2	n = 3	n = 4	n = 5
Tradicional	3,3	1,3	2,0	5,8	32,8
Aleatório	0,3	0,7	1,8	2,5	5,3
Proposto	1,2	1,0	1,1	2,6	1,6

Avaliando-se 33 amostras ($n = 5$) por entrada, percebe-se que os algoritmos tradicional e aleatório apresentaram-se aproximadamente 21,3 e 19,5 vezes mais custosos que o algoritmo pro-

posto, respectivamente. Isto nos mostra que, considerando-se a variação da variável n de 1 à 5, enquanto o número de amostras por entrada foi multiplicada por 11, o custo computacional do método tradicional e do aleatório aumentaram, respectivamente, aproximadamente 104,6 e 127.3 vezes. Enquanto isso, o algoritmo proposto aumentou o seu custo computacional em aproximadamente 3.95 vezes.

Ao analisar-se o aumento do custo computacional em função de n, percebe-se que o algoritmo tradicional e o aleatório necessitam de um custo computacional crescente aparentemente exponencialmente em relação à n, enquanto o método proposto mostra uma variação aparentemente constante.

Na Figura 1(b), realiza-se uma simulação para estados iniciais onde todas as primitivas de movimento geradas resultassem em colisão. Tal simulação visa avaliar os custos computacionais dos casos em que o robô realizará a expansão de primitivas e nenhuma delas estará livre de colisões. Na tabela 2, são mostrados os desvios padrão referentes à simulação. Os algoritmos tradicional e aleatório necessitam sempre checar todas as primitivas de movimento quanto ao custo e colisão, enquanto o algoritmo proposto checa apenas as primitivas geradas na primeira iteração.

Tabela 2: Desvios padrão em ms dos métodos tradicional, aleatório e proposto de geração de primitivas, obtidos a partir da simulação mostrada na Figura 1(b).

Método	n = 1	n = 2	n = 3	n = 4	n = 5
Tradicional	3,6	3,5	1,5	84,2	29,7
Aleatório	0,4	0,6	1,1	2,5	4,1
Proposto	0,78	0,33	0,47	0,38	0,38

Pode-se notar pelo gráfico que o custo computacional mantém-se praticamente constante no algoritmo proposto. Isso ocorre visto que o algoritmo proposto realiza apenas a sua primeira iteração e, ao constatar que todas as primitivas geradas geram colisão, considera que não há possibilidade de locomoção, retornando a ausência de uma solução sem necessitar executar as demais iterações. Neste caso, quando $n = 5$, os algoritmos tradicional e aleatório são, respectivamente, aproximadamente 110.65 e 111.03 vezes mais custosos computacionalmente que o algoritmo proposto.

Ou seja, em estados onde não haja possibilidade de movimentos sem gerar colisão, para $n = 5$ e $k = 2$, enquanto os algoritmos tradicional e aleatório gerarão todas as 1089 primitivas de movimento, calculam os custos de cada uma e verificam que todas estão colidindo com obstáculos, o algoritmo tradicional gera apenas 9 primitivas e verifica que já para as 3 amostras da primeira entrada, não há solução livre de colisão, retornando, assim, a ausência de uma primitiva livre de colisão para o problema.

5.2 Análise dos custos de navegação

A fim de comparar as soluções obtidas (primitivas retornadas) pelo algoritmo proposto em relação ao algoritmo tradicional, foram gerados 100 ambientes aleatoriamente, cada um com 3 obstáculos em formato circular. Para cada ambiente foram gerados aleatoriamente 100 estados iniciais e 100 estados finais para o robô, independente de ambos estarem ou não em colisão com obstáculos. Em seguida, para cada par de estado inicial e final gerado, eram expandidas as primitivas de movimento a partir do estado inicial em direção ao final e obtidas as primitivas de movimento e os custos de navegação provenientes da utilização de cada um dos dois métodos. Para fins computacionais, definiu-se o número de iterações como sendo $n = 2$, o que gera 5 amostras para cada entrada do método tradicional.

Os resultados obtidos podem ser vistos na Tabela 3. Dos 10.000 casos de expansões de primitivas realizadas por cada método, 2.019 casos não apresentavam solução através de nenhum dos dois métodos. Em 7.981 casos, havia solução para pelo menos um dos métodos. Em 54 casos, o método tradicional encontrou uma solução, enquanto o método proposto não encontrou nenhuma, o que equivale a aproximadamente 0,68% dos casos com solução. Em 7.217 casos, ambos algoritmos obtiveram a mesma solução, o que equivale a aproximadamente 90,43% dos casos com solução, contudo, o método proposto apresentou custo computacional menor que o método tradicional. Em 710 casos, ambos algoritmos encontraram solução para o problema, porém, soluções diferentes. Isto equivale a aproximadamente 8,90% dos casos com solução.

Nos casos em que ambos os métodos divergiram na solução, verificou-se que a média das razões entre o custo de navegação obtido através do método tradicional com o obtido pelo método proposto foi de aproximadamente 99.95%, com desvio padrão de aproximadamente 0.3%.

O mesmo teste foi realizado novamente, contudo, comparando-se o algoritmo proposto e o algoritmo aleatório. Dos 10.000 casos, 2.044 casos não apresentavam solução através de nenhum dos dois métodos. Em 7 casos, apenas o algoritmo proposto encontrou uma solução. Em 33 casos, apenas o algoritmo aleatório encontrou uma solução. Nos 7.916 casos restantes, ambos algoritmos encontraram uma solução, sendo que a média das razões entre o custo de navegação obtido pelo método proposto e aqueles obtidos pelo método aleatório foi de aproximadamente 99,75%, com desvio padrão de aproximadamente 0,47%.

Tabela 3: Comparativo do método tradicional *versus* método proposto das soluções obtidas.

Método tradicional x Método proposto			
Casos analisados nas simulações	Número de simulações	Porcentagem em relação ao total de simulações	Porcentagem em relação aos casos com solução
Casos onde ambos métodos não encontraram solução	2.019	20,19	-
Casos onde apenas o método tradicional encontrou uma solução	54	0,54	0,68
Casos onde ambos métodos encontraram a mesma solução	7.217	72,17	90,43
Casos onde Ambos métodos encontraram soluções diferentes	710	7,1	8,9

6 Conclusões

No presente trabalho foi proposto um algoritmo de geração de primitivas de movimento baseado no método de otimização por eliminação. Tal algoritmo foi comparado com os métodos tradicional e aleatório de geração de primitivas de movimento. Foi mostrado que o algoritmo proposto consegue gerar soluções com custo computacional menor que os métodos tradicional e aleatório. Além disso, o método proposto apresentou, na maioria dos casos, soluções com o mesmo custo que o método tradicional. Futuramente, pretende-se realizar um estudo do aumento da complexidade computacional do algoritmo proposto quando o número de entradas aumenta e aplicar tal método em algoritmos de planejamento de trajetória.

Agradecimentos

Os autores agradecem ao apoio financeiro fornecido pela CAPES e pela FAPESP (processos 2012/03088-3 e 2012/20824-5).

Referências

C. L. Bottasso, D. L. and Savini, B. (2008). Path planning for autonomous vehicles by trajectory smoothing using motion primitives, *IEEE Transactions on Control Systems Technology* **16**(6): 1152–1168.

Cohen, B., Chitta, S. and Likhachev, M. (2010). Search-based planning for manipulation with motion primitives, *IEEE International Conference on Robotics and Automation*, pp. 2902–2908.

Frazzoli, E., Dahleh, M. A. and Feron, E. (2001). Real-time motion planning for agile autonomous vehicles, *Proceedings of the American Control Conference*, Vol. 1, pp. 43–49 vol.1.

Kris Hauser, Tim Bretl, K. H. and claude Latombe, J. (2008). Using motion primitives in probabilistic sample-based planning for humanoid robots, *in* S. Akella, N. Amato, W. Huang and B. Mishra (eds), *Algorithmic Foundation of Robotics VII*, Vol. 47 of *Springer Tracts in Advanced Robotics*.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning.

LaValle, S. M. (2003). From dynamic programming to rrts: Algorithmic design of feasible trajectories, *in* A. Bicchi, D. Prattichizzo and H. Christensen (eds), *Control Problems in Robotics*, Vol. 4 of *Springer Tracts in Advanced Robotics*.

Lavalle, S. M. and Konkimalla, P. (2001). Algorithms for computing numerical optimal feedback motion strategies, **20**(9): 729–752.

Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles, *The International Journal of Robotics Research* **28**(8): 933–945.

Magalhães, A. C., Prado, M., Grassi Jr, V. and Wolf, D. F. (2013). Autonomous vehicle navigation in semi-structured urban environment, *Proceedings of the 2013 IFAC Intelligent Autonomous Vehicle Symposium*.

Monet, E. N. (2003). *Dynamic modeling and control of a car-like robot*, Master thesis, Faculty of the Virginia Polytechnic Institute and State University.

Pepy, R., Lambert, A. and Mounier, H. (2006). Path planning using a dynamic vehicle model, *Proceedings of the Information and Communication Technologies*, Vol. 1, pp. 781–786.

Pivtoraiko, M. and Kelly, A. (2011). Kinodynamic motion planning with state lattice motion primitives, *International Conference on Intelligent Robots and Systems*, pp. 2172–2179.

Pivtoraiko, M., Nesnas, I. A. D. and Kelly, A. (2009). Autonomous robot navigation using advanced motion primitives, *Proceedings of the IEEE Aerospace conference*, pp. 1 –7.

Rao, S. S. (2009). *Engineering Optimization: Theory and Practice*, John Wiley and Sons Inc.