ITVsim - Manual do Usuário

Sumário

1 Introdução

ITVsim é um simulador de estratégias cooperativas para múltiplos rôbos aéreos desenvolvido para o projeto *Desenvolvimento de Veículos Autônomos Cooperativos para Mapeamento e Mineração*. O principal objetivo do sistema é disponibilizar um ambiente de desenvolvimento prático com foco em interações simples entre os robôs e o ambiente, de modo que o usuário apenas se preocupe com os comportamentos de alto nível contidos na estratégia de cooperação. Para tanto, os modelos físicos e a comunicação entre os robôs são extremamente simplificados e não condizem com as restrições existentes em sistemas reais. Dessa maneira, o uso do sistema é indicado durante as primeiras etapas do projeto, quando a estratégia de cooperação está em processo de definição, podendo ser testada com um número arbitrário de robôs e comparada com outras estratégias candidatas. Para simulações complexas, é recomendado o uso do *Robot Operating System* (http://www.ros.org/) em conjunto com o simulador Gazebo (http://gazebosim.org/).

O simulador *ITVsim* foi desenvolvido no ambiente *MATLAB R2013a* devido às facilidades envolvidas na programação de controladores por parte

do usuário. Além disso, como os modelos físicos e de comunicação são simplificados, o sistema não requer uma capacidade elevada de processamento, o que justifica a utilização de uma linguagem de programação de alto nível.

2 Interface

Na Figura 1 é apresentada a interface principal do *ITVsim*, que é dividida em cinco subinterfaces: visualizador, mapa de anomalias, configurações do terreno, da simulação e do renderizador.

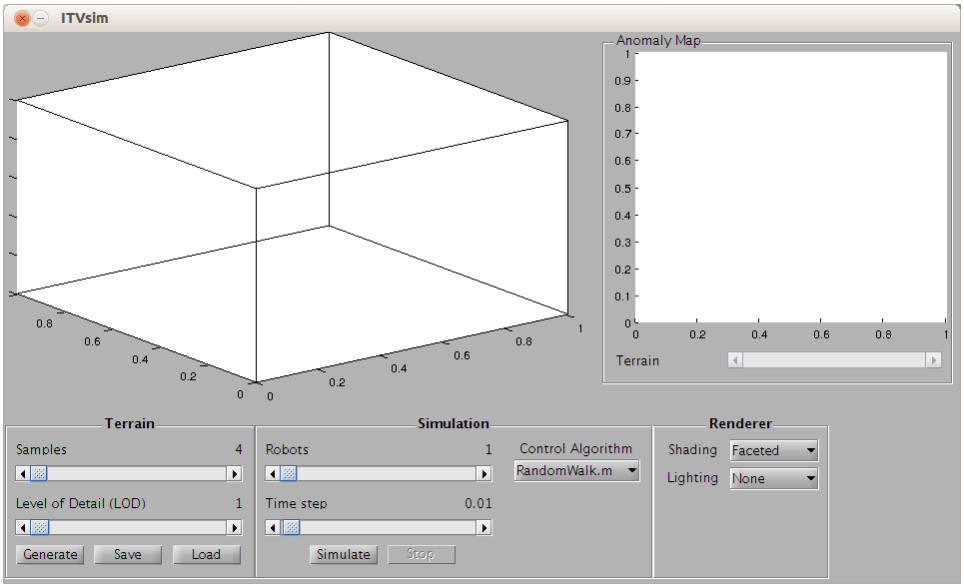


Figura 1: Interface principal do *ITVsim*.

2.1 Visualizador

O visualizador representa a simulação graficamente para o usuário, como mostrado na Figura 2. Existem três elementos gráficos contidos na simulação: o terreno, o mapa de anomalias e os robôs. O terreno é caracterizado por uma grade regular de vértices cujas alturas seguem a topografia de um modelo analítico interno do simulador, mas um arquivo externo com dados reais também pode ser utilizado. O mapa de anomalias encontra-se abaixo do terreno e representa as anomalias magnéticas contidas no mesmo através de um gradiente ciano–magenta. Os robôs são representados como círculos vermelhos. Todos os elementos gráficos são renderiados através de uma projeção ortográfica.

Em termos de interação, o usuário pode rotacionar a cena clicando com o botão esquerdo do mouse e o arrastando em qualquer direção. Além disso,

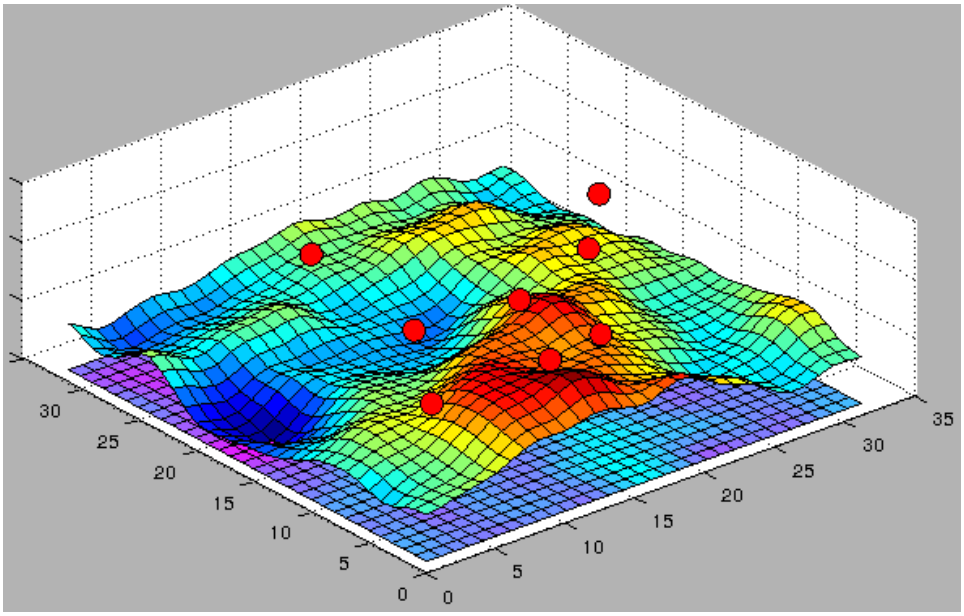


Figura 2: Elementos gráficos do Visualizador.

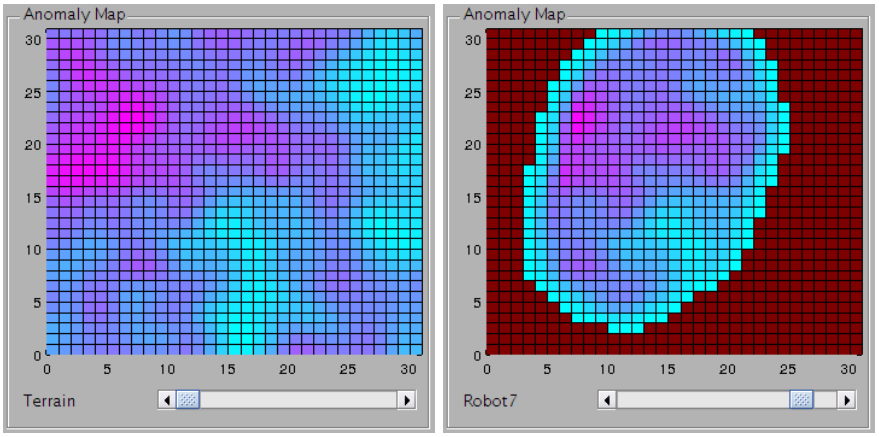
o botão direito permite acessar um menu que contém diversas opções de visualização, tais como:

- **Reset to Original View**: retorna o visualizador às configurações iniciais do simulador.
- **Goto X-Y view**: rotaciona a cena para visualização no plano X-Y.
- **Goto X-Z view**: rotaciona a cena para visualização no plano X-Z.
- **Goto Y-Z view**: rotaciona a cena para visualização no plano Y-Z.
- **Plot box rotate**: desabilita o modo de rotação com renderização contínua (recomendado para computadores com baixo processamento).
- **Continuous rotate**: habilita o modo de rotação com renderização contínua.
- **Stretch-to-fill axes**: Estende os eixos X-Y para cobrir todo o espaço livre do visualizador.
- **Fixed aspect ratio axes**: Mantém a mesma proporção nos eixos X-Y do visualizador.

2.2 Mapa de Anomalias

O mapa de anomalias é criado através de um modelo analítico interno do simulador e **não representa um modelo real de anomalias magnéticas**. Como o foco do *ITVsim* é o algoritmo de cooperação, métricas como a razão entre a área coberta e o tempo se sobrepõem à veracidade dos dados coletados. Assim, o objetivo do mapa de anomalias é apenas prover uma visualização da cobertura relacionada com a estratégia desenvolvida.

Além da representação contida no visualizador (Seção 2.1), o *ITVsim* provê a o mapa de anomalias em visão aérea, tanto globalmente quanto localmente, isto é, em relação ao conhecimento de cada robô (Figura 3). Esses mapas podem ser acessados individualmente através da barra de rolagem vertical localizada logo abaixo da representação gráfica dos mesmos. Com relação aos mapas locais, as células em vermelho escuro (Figura 3(b)) representam informações desconhecidas pelo robô.



(a) Mapa global. (b) Mapa local.

Figura 3: Visualizações aéreas do mapa de anomalias global e local.

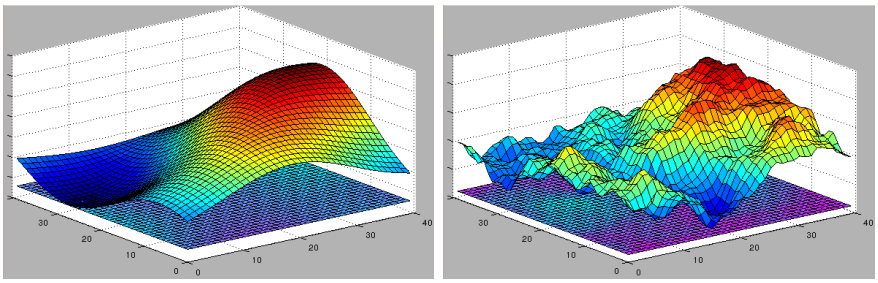
A leitura do mapa local é feita em cada iteração através um cone com ápice na posição do robô. Desse modo, quanto maior a altura do robô em relação ao terreno, maior será o seu campo de visão. Novamente, esse é apenas um modelo simplório de sensoriamento que não possui fundamentos em sensores reais.

2.3 Configurações do Terreno

Existem duas opções de configurações do terreno: **samples** e **level of detail**. Ambas devem ser usadas quando o usuário desejar criar um terreno baseado no algoritmo interno do simulador. Para carregar um arquivo externo, utilize o botão **Load**.

- **Samples**: refere-se ao número de amostras n da grade de vértices. O simulador irá criar uma grade com n^2 vértices. Quanto menor o número de amostas, menos detalhes podem ser modelados no terreno.
- **Level of Detail (LOD)**: define o nível de detalhes do terreno. Como o modelo interno de topografia é baseado em uma soma fractal de sinais ruidosos, mais detalhes de alta frequência são adicionados ao terreno quanto maior for o valor do LOD. Em outras palavras, um valor baixo irá gerar um terreno suave e com poucos detalhes, enquanto um valor alto irá gerar o oposto.

Para gerar um novo terreno, selecione as configurações desejadas para os parâmetros acima e clique em seguida no botão **Generate**. O terreno será calculado e logo após renderizado no visualizador. O usuário pode salvar os dados gerados em um arquivo externo ao clicar sobre o botão **Save**. Também é possível carregar um mapa previamente salvo através do botão **Load**.



(a) $LOD = 1$ (b) $LOD = 4$

Figura 4: Mapas gerados pelo algoritmo interno do simulador com 40 amostras (samples) e diferentes valores para o nível de detalhe (LOD).

2.4 Configurações da Simulação

Três configurações podem ser alteradas na simulação: o algoritmo de controle (**Control Algorithm**), o número de robôs (**Robots**) e o passo de integração (**Time step**).

- **Control algorithm**: define qual arquivo externo será utilizado para controlar os robôs. Esse arquivo encontra-se no diretório **algorithms**, dentro do diretório raíz do *ITVsim*.
- **Robots**: o número de robôs da simulação.
- **Time step**: o tempo de integração Δt da simulação.

Para rodar a simulação, selecione um controlador da lista disponível, o número de robôs e o tempo de integração. Logo após, clique no botão

Simulate. Se um terreno já existir, os robôs serão inicializados sobre ele. Caso contrário, um novo terreno será automaticamente criado de acordo com a sua configuração atual (Seção 2.3). Durante a simulação, todas as configurações serão travadas nos seus respectivos valores, somente podendo ser alteradas após o término da mesma. O botão **Stop** pode ser utilizado para terminar a simulação em qualquer momento.

2.5 Configurações do Renderizador

As configurações do renderizador efetuam apenas mudanças estéticas no visualizador. Existem duas opções: **Shading** e **Lighting**, cada qual pode assumir três diferentes valores. Tais configurações podem ser combinadas par a par, totalizando 15 diferentes variedades estéticas. Algumas dessas são mostradas na Figura 5.

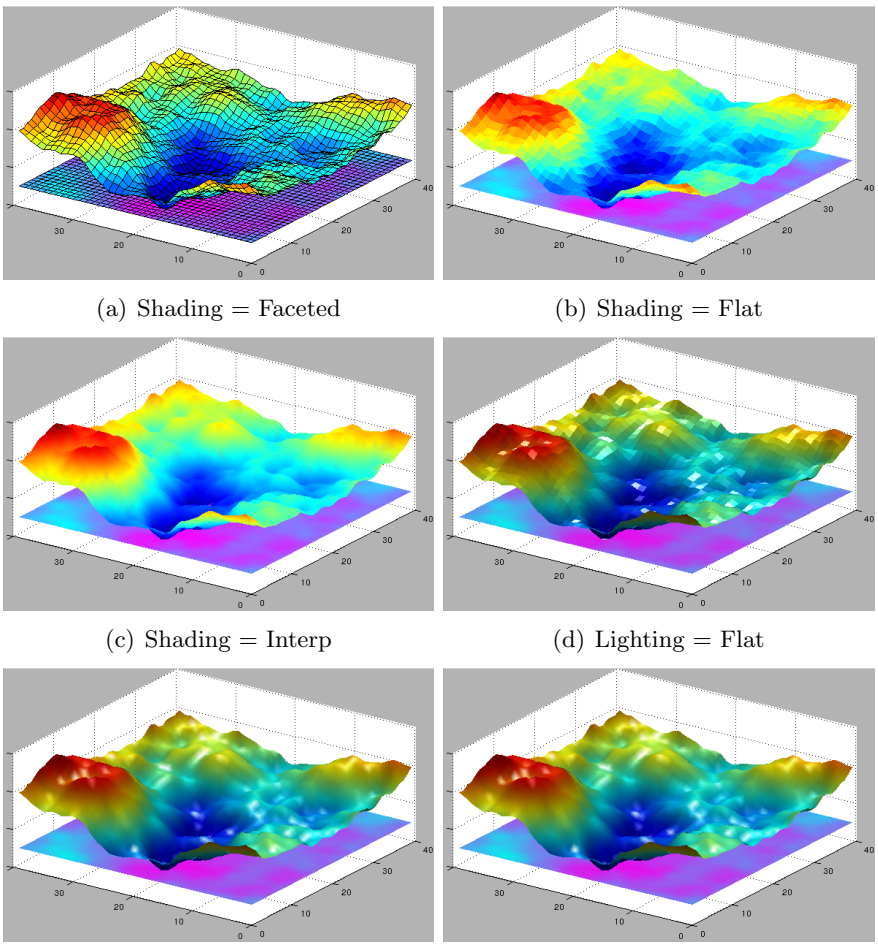
- **Shading**: controla a interpolação das cores na grade de vértices.
  - **Flat**: colore cada face do terreno com uma única cor.
  - **Faceted**: desenha linhas pretas conectando cada vértice.
  - **Interp**: interpola linearmente as cores dos vértices em cada face.
- **Lighting**: controla a iluminação local da cena.
  - **None**: desliga a iluminação.
  - **Flat**: calcula a iluminação por face.
  - **Gouraud**: calcula a iluminação por vértice.
  - **Phong**: calcula a iluminação por pixel.

3 Desenvolvimento de Estratégias

Para carregar uma nova estratégia no simulador basta criar uma nova classe no diretório **algorithms** dentro do diretório raiz do *ITVsim*. Ao iniciar o simulador, os arquivos referentes a classes do *MATLAB* serão carregados e disponibilizados na caixa **Control algorithm**. Para facilitar o desenvolvimento, dois exemplos básicos estão inclusos: **RandomWalk.m** e **Segregation.m**. No primeiro, os robôs navegam aleatoriamente, enquanto no segundo eles são divididos de maneira autônoma em três grupos contendo o mesmo número de robôs. Em ambos exemplos os robôs tentam se manter em uma altura constante com relação ao solo.

3.1 Classe Strategy

Para definir uma estratégia é necessário estender a classe **strategy**, através da qual o simulador efetua chamadas de alto nível sem se preocupar com a



(a) Shading = Faceted

(b) Shading = Flat

(c) Shading = Interp

(d) Lighting = Flat

(e) Lighting = Gouraud

(f) Lighting = Phong

Figura 5: Diferentes configurações de interpolação de cores e modelos de iluminação do renderizador.

implementação concreta. Em outras palavras, a classe provê uma interface abstrata comum para todas as estratégias.

O construtor da classe recebe como entrada do simulador a grade de vértices que representa o terreno. Portanto, qualquer nova estratégia deve possuir esses mesmos parâmetros em seu construtor, sendo obrigação da classe filha chamar o construtor de sua classe pai. Por exemplo, o código mais básico para inicializar essa hierarquia é apresentado logo abaixo, no qual a classe **NovaEstrategia** é definida. Seu construtor recebe três matrizes bidimensionais contendo as informações do terreno. Tais parâmetros seguem a especificação do comando **surf** do *MATLAB*.

```
1 classdef NovaEstrategia < strategy
2     methods
3         function this = NovaEstrategia(x, y, z)
4             this@strategy(x, y, z);
5         end
6 end
```

Na implementação de uma estratégia, os métodos concretos mais importantes são **initialize** e **control**. No primeiro, o desenvolvedor pode especificar as posições e velocidades iniciais de cada robô, além de escolher entre um modelo cinemático ou dinâmico de controle. No segundo, o desenvolvedor tem acesso às posições e velocidades de cada robô e deve determinar quais serão as respectivas entradas de controle a partir dessas informações. A assinatura do primeiro método é dada por

```
1 function [p, v, dynamics] = initialize(this, robots, limits)
```

no qual os valores de retorno **P** e **V** são matrizes contendo as posições e velocidades iniciais de cada robô e **dynamics** é uma variável booleana que define se as entradas de controle devem ser interpretadas como acelerações ou velocidades, de acordo com o valor **true** e **false**, respectivamente. Com relação aos parâmetros de entrada, **this** é uma referência para uma instância da classe (o padrão de orientação a objetos em *MATLAB* requer essa declaração), **robots** é o número de robôs e **limits** é um arranjo de duas posições contendo o valor máximo do eixo X e Y.

A posição e a velocidade dos robôs é descrita por uma matrix $n \times 3$, sendo n o número de robôs. Mais especificamente, seja $\mathbf{p}_i = [x_i, y_i, z_i]^T$ o vetor de posição do robô i. Dessa maneira, a matriz de posições conjunta **P** que deve ser retornado pelo método **initialize** é dado por

$$\mathbf{P} = \begin{pmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_n^T \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{pmatrix}, \tag{1}$$

e o mesmo padrão é seguido para as velocidades, isto é, cada coordenada em

uma coluna da matriz. Essa descrição permite o uso da função **bsxfun** do *MATLAB*, que pode efetuar o cálculo de distâncias entre todos os robôs com poucas linhas de código (veja o exemplo no arquivo **Segregation.m**). Se um vetor vazio for retornado para as posições, o simulador irá as inicializar com valores aleatórios distribuídos uniformemente dentro das dimensões do terreno. No caso das velocidades, elas serão inicializadas com a matriz nula.

O método **control** possui a seguinte assinatura:

```
1 function input = control(this, p, v)
```

onde **P** e **V** são as matrizes de posição e velocidade já discutidas e a variável de retorno **input** é uma matrix $n \times 3$ com as entradas de controle dos robôs. A especificação dessa matriz segue a mesma descrição das posições e velocidades, isto é, em cada linha são dadas as entradas de controle de um robô específico e cada coluna é reservada para uma dimensão específica.

3.2 Exemplo Mínimo

Abaixo segue o código fonte mínimo de uma estratégia que mantém os robôs parados no solo. Ambas as matrizes de posições e velocidades são inicializadas com valores vazios, o que delega a responsabilidade de posicionar os robôs no terreno para o simulador. Também, um modelo de controle dinâmico é selecionado, isto é, os valores de retorno do método **control** serão interpretados como acelerações. Esse método apenas cria uma matriz nula como entrada de controle e a retorna para o simulador.

```
1  classdef NovaEstrategia < strategy
2      properties (SetAccess = private)
3          robots
4      end
5
6      methods
7          function this = NovaEstrategia(x, y, z)
8              this@strategy(x, y, z);
9          end
10
11         function [p, v, dynamics] = initialize(this, r, wl)
12             p = []; v = [];
13             dynamics = true;
14             this.robots = r;
15         end
16
17         function input = control(this, p, v)
18             input = zeros(this.robots, 3);
19         end
20     end
21 end
```