ALGORÍTMO INTELIGENTE (AI) PARA ROBÔ AUTÔNOMO EM PLATAFORMA SIMULADA

Kauê C. Souza, André K. Barreto, Frederico F. Pinto, João G. Morais, Thales P. Cruz,
Equipe 3k.

***Resumo*- Este trabalho tem como objetivo o desenvolvimento e aprimoramento de um algoritmo capaz de guiar um robô simulado na plataforma *Cospace* a fim de obter o maior número de pontos possíveis, independentemente da arena apresentada. Três métodos diferentes foram utilizados para atingir este fim.**

Palavras-chave
CoSpace. Inteligência artificial. Algoritmo. Decisões. Autonomia. Mapeamento. 3k.

1 Introdução

Com o anúncio da edição deste ano da *Latin American Robotics Competition (*LARC*),* a qual acontecerá aqui no Brasil (Uberlândia-MG), surge a oportunidade de criação de uma equipe competidora na categoria *Cospace*.

Nela, diante de uma arena com ação física similar à realidade e que contem diversas paredes, buracos e armadilhas aleatórias, um robô virtual conta com uma bússola, quatro sensores de ultrassom e seis sensores de cor para se guiar além de um motor em cada uma de suas duas rodas para se locomover. Seu objetivo final é coletar objetos pontuantes e finalizar a partida de oito minutos com um placar maior que seu oponente.A localização destes objetos não é divulgada para o robô, então cabe à sua inteligência artificial definir qual a melhor estratégia para pontuar e evitar armadilhas.

Com isso em mente, faz-se necessário o desenvolvimento de um bom algoritmo capaz de fazer decisões por si só, ou seja, adaptável ao ambiente. Esta é a chave para a vitória.

Paralelamente, o ramo da Inteligência Artificial está desde a década de 1950 apresentando ao mundo uma nova visão computacional. Nomes como Stuart Russell e Peter Norvig têm quebrado o a visão passada do que é de fato uma máquina, dando a elas graus de liberdade impressionantes. Em especial, o método de desenvolvimento de inteligência artificial genético, o qual simula gerações de "neurônios" (códigos) para que a "seleção natural" aja sobre eles (os melhores códigos avançarão de geração e os piores serão descartados). Este método pode ser aplicado no projeto para a LARC 2015.

A extensa bibliografia sobre algoritmos de geração genética servirá de forte suporte para o desenvolvimento do código e, ainda mais, para a capacitação dos membros envolvidos no projeto.

2 Desenvolvimento passado

O algoritmo desenvolvido para a competição de 2014, o qual foi capaz de finalizar o evento em terceiro lugar, será utilizado como padrão de comparação para medir a eficácia deste projeto.

Pelo fato do código anterior não ter sido criado da maneira pretendida neste artigo, ele não pode ser incorporado no projeto atual em si. Isso ocorre pela chance da seleção de neurônios tornar-se parcial e não adaptável, já que o código de 2014 foi

Figura 1 - Souza(em azul) na LARC 2014 competindo com o algoritmo em questão. Disponível em: <http://cospacerobot.org/about/cospace-worldwide>

utilizado em um mapa específico diferente do mapa deste ano e contava com a aleatoriedade do surgimento das peças para ser eficaz. Isto resultava em desempenhos muito discrepantes.

3 Algoritmo por inteligência artificial

3.1 Estruturação

A seleção de códigos é vital para a seleção dos melhores. O fato de cada partida durar oito minutos dificultaria intensamente os testes e, por causa disso, foi utilizado um método de engenharia reversa para extrair da plataforma os parâmetros a serem posteriormente modificados com o fim de encurtar este tempo sem distorção dos resultados.

```
int mapa[X][Y];

void draw(int x, int y, int lado, char r, char g, char b)   {
    for(int i=x;i<x+lado;i++)   {
        for(int j=y;j<y+lado;++j)   {
            mapa[i][j] = (int) ((int)r<<16|(int)g<<8|(int)b);
        }
    }
}

void draw(reta guia)    {
    int v=guia.tga*guia.init+guia.offset;
    int init_y = ceil(v);
    for(int i=guia.init;i<guia.end;++i) {
        mapa[i][init_y] = 1;
        v = v + guia.tga;
        init_y = ceil(v);
    }
}
```

Figura 2- Parte do código da plataforma modificada para acelerar os testes. Autoria própria.

Apesar de ser impossível replicar a plataforma como um todo, especialmente a parte gráfica e física dela, uma simplificação com *debug* em blocos de nota já seria suficiente para analisar qualquer erro no processo automatizado subsequente.

3.2 Neurônios

Com a plataforma em mãos, entra-se na etapa mais importante do projeto: o desenvolvimento de côdigos-neurônio.

Nestes, deve-se decidir quais parâmetros serão variados nos testes, a interdependência destes parâmetros, limites de variação, formação da aleatoriedade e, por fim, a personalização da trifurcação dos tipos de neurônio: o sensitivo, (responsável pela leitura e interpretação) o motor,

```
#ifndef NEURONIO
#define NEURONIO

#include <vector>

#define MAX_QT_POWER 25
#define K_STR 10

class Neuron;
enum tipo_neuronio  {
    SENSITIVO, MOTOR, CONECTOR
};

struct connection   {
    double k_strength[K_STR];
    Neuron * c_neuronio;
};

struct connection_to_make   {
    double k_strength[K_STR];
    char identifier[16];
};
```

Figura 3- Inicio do código dos neurônios. Autoria própria.

(que dá valores para a rotação dos motores do robô), e o conector (capaz de realizar as conexões entre neurônios para que haja coesão entre estes e, ultimamente, a seleção possa ocorrer).

3.3 Seleção de neurônios

Similarmente ao desenvolvimento dos neurônios, a seleção destes é feita por meio da definição de parâmetros relevantes para a performance.

Fazer um desenvolvimento totalmente imparcial é além do escopo deste projeto, já que o time não possui tempo, poder de processamento ou um código robusto o suficiente para isso. Portanto, foi pelo próprio julgamento e também pela intuição humana as decisões foram tomadas.

Pela observação dos impactos que seleções diferentes tinham, foi definida que ao invés de priorizar o montante final de pontos do robô em uma partida, a seleção do robô que realizava a maior quantia de pontos médios por minuto era a que trazia melhores resultados. Claramente, inúmeros outros parâmetros como em quantas armadilhas o robô caia, quantas chances de depósito eram passadas, quanto tempo era perdido com obstáculos e velocidade de deslocamento foram levados em conta. As relações e

```
class Neuron    {
    private:
        double charge;
        double vcharge[MAX_QT_POWER];
        float threshold;
        char name[256];
    public:
        std::vector<connection> conexoes;
        char identifier[16];
        tipo_neuronio tipo;

        // se tipo == SENSITIVo, charge_id deve ser diferente de 0.
        // todos os neuronios motores tem action_id.
        int charge_id;
        int action_id;

        void set_charge();

        Neuron();
        Neuron(char * a, tipo_neuronio tipo, float threshold);
        Neuron(const char*);
        ~Neuron();

        void output_to_file(const char*);
        void output_to_file(const char*, const char*);
        std::vector<connection_to_make> load_from_file(const char * arquivo);

        void make_connection(Neuron&, const double *);
};

void make_connection(Neuron&, Neuron&);
#endif /* NEURONIO */
```

Figura 4- Parte do código de seleção neuronal. Autoria própria.

valores utilizados podem ser observados no código final.

4 Desenvolvimento do algoritmo convencional

4.1 Montagem pela plataforma

Diretamente pela plataforma foram criadas as funções básicas e pouco avançadas do robô: movimento, curvas, detecção de cores, detecção de obstáculos, teleporte etc.

Figura 5- Exemplo de função. Autoria própria.

4.2 Otimização pelo código fonte

Devido à alta limitação de customização do código pela plataforma, foi necessária a modificação manual do código gerado pela plataforma. Além disso, a estética dos arquivos .cs gerados era extremamente precária, o que propiciava o aparecimento de *bugs* e dificultava uma melhor coesão e otimizações.

Com isso em mente, tiramos algumas horas para destrinchar os imensos *else if*s e finalmente concluir o algorítmo básico, não inteligente, que serviria de backup e de base para testes de implementação de novas ideias.

```
}
else if(CSRight_R>=235 && CSRight_R<=255 && CSRight_G>=235
{
    Duration = 2;
    CurAction =5;
}
else if(CSLeft_R>=235 && CSLeft_R<=255 && CSLeft_G>=235 &&
{
    Duration = 2;
    CurAction =6;
}
else if(CSRight_R>=31 && CSRight_R<=71 && CSRight_G>=31 &&
{
    Duration = 9;
    CurAction =7;
}
else if(CSRight_R>=180 && CSRight_R<=229 && CSRight_G>=0 &
{
    Duration = 49;
    CurAction =8;
}
else if(CSLeft_R>=180 && CSLeft_R<=229 && CSLeft_G>=0 && C
{
    Duration = 49;
    CurAction =9;
}
else if(CSRight_R>=0 && CSRight_R<=60 && CSRight_G>=0 && C
{
    Duration = 49;
```

Figura 6 - Exemplo de sequência bruta no código, posteriormente corrigida. Autoria própria.

5 Desenvolvimento de algoritmo mapeador

5.1 Considerações iniciais

A terceira ideia de código é baseada no mapeamento fixo da arena para posterior tomada de decisões do robô. O curto tempo por rodada, a disposição desfavorável dos sensores angulados, os erros físicos embutidos na plataforma e a alta aleatoriedade de formação de peças fizeram com que este algoritmo não trouxesse bons resultados.

5.2 Definição espacial

O robô inicia o round realizando um giro de 360 graus e armazenando as distâncias dadas pelos sensores de ultrassom. Estas distâncias sâo associadas ao ângulo em relação à bússula e, após cálculos trigonométricos complexos, um *array* de *bytes* que representa a arena no momento começa a receber valores indicadores do que há em cada área mapeada. O processo é o mesmo para outros elementos do jogo (áreas especiais, zonas de perigo e objetos, por exemplo) e o algoritmo reseta após o teleporte. O debug foi feito por um arquivo .txt que nos permitia vsualizar a percepção do robô graficamente.

```
public static void mapear() {
    int x, y;
    for(int i=0;i<US_Front;++i) {
        x = TMX + (int) (Math.Round(i*Math.Sin(Compass*Math.PI/180)) + 20*Math
        y = TMY + (int) (Math.Round(i*Math.Cos(Compass*Math.PI/180)) + 20*Math
        mapa[x,y] = CHAO;
    }
    for(int i=0;i<US_Back;++i) {
        x = TMX - (int) Math.Round(i*Math.Sin(Compass*Math.PI/180));
        y = TMY - (int) Math.Round(i*Math.Cos(Compass*Math.PI/180));
        mapa[x,y] = CHAO;
    }
    for(int i=0;i<US_Left;++i) {
        x = TMX + (int) Math.Round(i*Math.Sin((Compass+45)*Math.PI/180));
        y = TMY + (int) Math.Round(i*Math.Cos((Compass+45)*Math.PI/180));
        mapa[x,y] = CHAO;
    }
    for(int i=0;i<US_Right;++i) {
        x = TMX + (int) Math.Round(i*Math.Sin((Compass-45)*Math.PI/180));
        y = TMY + (int) Math.Round(i*Math.Cos((Compass-45)*Math.PI/180));
        mapa[x,y] = CHAO;
    }
```

Figura 7 - Parte do código de mapeamento. Autoria própria.

5.3 Tomada de decisões

Até o mapeamento do local de depósito, o robô anda sempre em direção à área que foi menos explorada do mapa. Após isso, o robô é instruído a contornar a zona especial e continuar dentro dela coletar cinco peças, posteriormente indo em direção à zona de depósito. Caso o robô permaneça dez segundos sem coletar peças na zona especial, ele se dirige aos cantos do mapa com menos obstáculos. Caso haja mais que uma zona especial, o robô priorizará a que estiver mais próxima do depósito.

6 Testes comparativos finais

Por fim, já que a competição só será realizada em outubro de 2015, foi realizado um teste definitivo comparando os algoritmos de 2014 e o principal deste projeto (obtido por métodos de inteligência artificial). A alta aleatoriedade de geração de peças rosa influi na obtenção de um resultado pontual e preciso. Porém, mesmo observando uma única partida, já foi clara a vantagem motora, reativa e, finalmente, pontual do algoritmo deste ano. O placar foi de 1520 a 860 a favor do algoritmo em questão.

7 Conclusão

Por meio do desenvolvimento da inteligência do robô virtual, teve-se como visão a auto-capacitação e, ultimamente, o avanço da computação para que, por fim, tenhamos um futuro melhor. Com resultados claros e quantitativos, o este projeto foi capaz de entregar um algoritmo não só funcional, mas também eficiente e robusto. O aprendizado adquirido no processo de desenvolvimento foi realmente valioso e, apesar do evento ainda não ter acontecido, graças à comparação com o algoritmo previamente usado na LARC 2014 este projeto provou sua eficiência e possivelmente cumprirá com o resto de seus objetivos iniciais.

8 Referências

8.1 Bibliografia

NORVIG, Peter C. *Artificial Inteligence*: the next frontier. 1. ed. HarperCollins: Elsevier, 1972.

RUSSEL, Stuart. *Intelligence:* artificial intelligence. 7. ed. Prentice-Hall: Citeseer, 1995.