Relatório do Projeto de Robô Quadrúpede com Aprendizado Não Supervisionado

Marques, Miguel Nakajima, RA: 210433

Projeto da disciplina de Introdução à Computação Natural (IA013)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
CEP 13083-970 – Campinas, SP, Brasil

miguelnakajima@gmail.com

Abstract – With miniaturization and lower costs of semiconductor and electronics, the use of microprocessors and artificial intelligence in task-driven robotics is increasing every day. In this scenario the autonomous agents must be as resilient as possible in order to decrease the need for maintenance due to their large numbers and be able to complete the tasks assigned to them even in extreme conditions. In order to increase the resilience of autonomous robots, the following paper presents a solution that allows the agents to optimize a decision making system considering the assets available and the impact of the activation of each one in fulfilling the task assigned. A classifier system paired with a genetic algorithm is modeled in order to make the agent able to move forward using its available limbs and the degrees of freedom for each one. The solution was coded in C and implemented in an Arduino Mega 2560. The variables measured from the environment were the distance of the robot to an obstacle and the current position of each limb. The results show that the robot is capable of teaching itself a sequence of movements that decrease its distance to the obstacle ahead.

Keywords – Artificial Intelligence, Classifier system, Genetic Algorithm, Arduino, Robotics, Machine Learning.

1. Introdução

Com a miniaturização e o barateamento dos semicondutores, é cada vez mais comum encontrarmos sistemas eletrônicos microprocessados capazes de cumprir tarefas de forma autônoma, desde a mais simples (como varrer o chão de uma casa desviando de obstáculos) até a mais complexa (como explorar um planeta distante). A proliferação de tais sistemas cria o problema do aumento da necessidade de manutenção, que normalmente envolve um agente humano e seu deslocamento até o local de atuação do sistema eletrônico ou o transporte do sistema até um local apropriado para o seu reparo.

Para mitigar essa situação, sistemas capazes de operar em situações extremas, com parte de seus elementos comprometidos ou inoperantes, são interessantes no âmbito de atividades de alta prioridade ou de difícil acesso para a execução da manutenção.

Basta imaginar o transtorno causado caso uma sonda espacial explorando o planeta Marte necessite de intervenção humana para executar manutenção em seu sistema de deslocamento. É desejável que a sonda em questão seja capaz de se deslocar pelo ambiente mesmo que um ou mais de seus múltiplos motores deixe de operar adequadamente.

Com isso em mente o presente trabalho tem como objetivo a elaboração de um sistema robótico computacional capaz de descobrir de forma autônoma e não supervisionada uma sequência de acionamentos de seus atuadores (posteriormente referenciados como "patas") que seja capaz de deslocar o robô para frente.

É desejável que o sistema seja flexível o suficiente para ser capaz de cumprir esse objetivo independente do número de atuadores em funcionamento e da capacidade dos mesmos, prevendo a aplicação do sistema em situações onde algumas das peças mecânicas do robô estejam parcialmente comprometidas ou até completamente inoperantes.

Ao final do projeto almeja-se obter um sistema de baixo custo (<R$200,00), plataforma aberta (Arduino), com alto grau de replicabilidade e que demonstre os conhecimentos obtidos na disciplina IA013 Introdução à Computação Natural.

Os detalhes de cada elemento do sistema são apresentados no tópico 2.

Os resultados obtidos são apresentados no tópico 3.

As conclusões e considerações para futuras pesquisas são apresentadas no tópico 4.

O código fonte parcial em linguagem C é apresentado no Anexo A. O código fonte completo assim como a versão mais atual deste

documento podem ser acessados em https://github.com/oarcanjomiguel/walking-brain
Recomenda-se a leitura do presente trabalho em conjunto com Boccato [1].

2. Proposta

Para atingir o objetivo descrito no item anterior é proposto o uso de um sistema classificador como o utilizado em Vargas [2] aliado a um algoritmo genético para seleção evolução das regras do sistema classificador. Esses sistemas deverão ser inteiramente implementados em uma placa Arduino Mega 2560 que controlará quatro servo motores do modelo Tower Pro SG90 e um sensor de distância HC-SR04 que medirá o deslocamento a cada aplicação de regra. Cada um desses elementos é descrito em mais detalhes nos subtópicos a seguir.

2.1 Sistema classificador

O sistema classificador será composto por um conjunto de regras contendo os seguintes parâmetros:

- Antecedente: A posição atual dos servo motores, lida pela biblioteca *Servo.h* do Arduino através da função *Servo.read()*;
- Consequente: A posição futura dos servo motores, a ser atualizada usando a função *Servo.write*(X_{t+1}) da biblioteca *Servo.h* do Arduino;
- Energia: representa a força da regra associada e é modificada conforme os leilões vão acontecendo. Declarada como tipo *float*;
- Especificidade: representa a quantidade (em porcentagem) de elementos do antecedente que não possuem o símbolo de # (*don't care*). Declarada como tipo *float*;

Para as partes antecedente e consequente, cada posição representa um servo motor, sendo que do ponto de vista do sistema classificador cada servo motor possui somente duas posições: 0 e 1 que representam ângulos nos servo motores de 70^o e 110^o respectivamente. O valor de posição 90^o é usado somente como uma posição inicial de repouso, não sendo utilizado no sistema classificador. As três posições são mostradas a seguir.

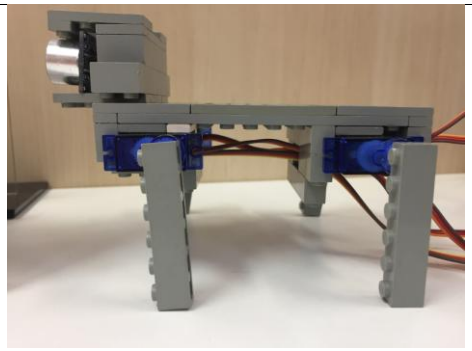

Figura 1. Posição 90^o(repouso)

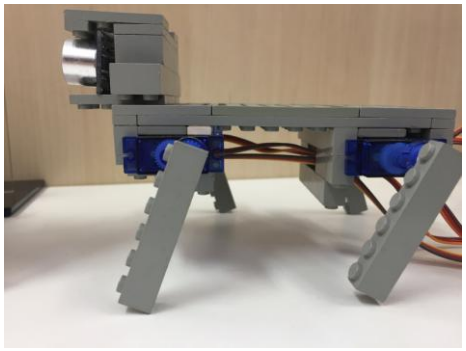

Figura 2. Posição 70^o

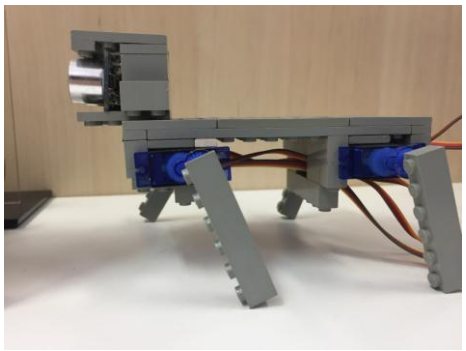

Figura 3. Posição 110^o

Da forma como foi parametrizado, o algoritmo permite que mais posições sejam adicionadas às possíveis posições dos servo motores e que mais servo motores sejam adicionados ao hardware, bastando que os parâmetros presentes no cabeçalho do programa sejam adequados.
A energia inicial foi parametrizada em 100.00 e declarada como tipo *float*.
Os parâmetros referentes ao cálculo do bid_t e do $ebid_t$ foram escolhidos de acordo com Vargas[2] e refinados para os valores mostrados na Tabela 1:

Parâmetro	Valor
k_0	0.1
k1	0.1
k2	0.0833
SPow	3
Meia vida	100
Bid_tax	0.0030

Tabela 1. Parâmetros do sistema classificador

A população inicial do sistema classificador (número de regras iniciais) foi inicialmente escolhida como 50 e posteriormente refinada para 20 regras. Os estados dos genes iniciais são aleatórios com média 0 e variância 1.
A recompensa adicionada à energia caso a regra aplicada cumpra o objetivo (deslocar-se para frente) foi determinada em +4.0. Já a punição no caso contrário é de -3.0.
O sistema classificador foi implementado como uma máquina de estados com 11 estados, sendo eles apresentados na tabela a seguir:

Estado	Descrição
AGUARDA	Aguarda comando de início (estado de repouso)
INICIALIZA	Inicializa os parâmetros e a população do sistema classificador
ESTABILIZA	Aguarda a estabilização da leitura do sensor de distância
MEDE_ANTES	Mede a distância até o obstáculo antes do leilão e da aplicação da regra e lê as posições atuais dos servo motores
APLICA_REGRA	Executa leilão e aplica parte consequente da regra vencedora aos servo motores
MEDE_DEPOIS	Aguarda término do posicionamento dos servo motores e lê a distância final até o obstáculo
COBRA_TAXAS	Determina se a regra deve ser recompensada ou punida de acordo com as distâncias lidas anteriormente e cobra todas as taxas do leilão (incluindo a recompensa)
RECOMPENSA	Se atingiu o número de iterações determinadas no parâmetro "Meia vida", vai para o estado CROSSOVER, caso contrário, incrementa esse número e vai para o estado ESTABILIZA
CROSSOVER	Executa o crossover entre as regras selecionadas e insere os indivíduos resultantes na população
MUTACAO	Seleciona genes dentro dos indivíduos resultantes do crossover e muda o estado deles
SUBSTITUI	Seleciona o número de indivíduos igual à população inicial (por elitismo ou torneio). Caso não tenha atingido o número máximo de gerações, vai para o estado ESTABILIZA, caso contrário vai para o estado AGUARDA

Tabela 2. Estados de máquina do sistema classificador

Todo o sistema classificador está contido no arquivo Sistema_Calssificador.ino no repositório indicado no item 1 deste trabalho.

2.2 Algoritmo genético

O algoritmo genético é executado nos estados CROSSOVER, MUTACAO e SUBSTITUI da máquina de estados do sistema classificador e consiste dos operadores de crossover, mutação e seleção de um AG convencional.
São utilizados os seguintes parâmetros:

Parâmetro	Valor
Taxa de mutação	0.3 (30%)
Número de duplas de filhos gerados a cada crossover	1 (cada conjunto de pais gera uma dupla de filhos)
Número de pais por crossover	2
Número máximo de gerações	10
Quantidade de competidores por torneio (seleção)	2

Tabela 3. Parâmetros do Algoritmo Genético

Os operadores do algoritmo genético são detalhados a seguir.

2.2.1 Crossover

Para o operador de crossover são selecionados os 4 indivíduos com maior energia, divididos em 2 pares.
Para cada par é selecionado um gene entre os 8 possíveis (4 genes são os antecedentes e 4 são os consequentes do sistema classificador) onde será feita a quebra do genoma. São então gerados dois descendentes dos indivíduos originais, sendo que um dos descendentes contém o genoma do primeiro pai até o ponto de crossover e o genoma do segundo pai a partir do

ponto de crossover. A ordem dos pais é invertida para o segundo indivíduo e o processo todo é repetido para o segundo par de pais selecionados.

2.2.1 Mutação

Para o operador de mutação são selecionados 30% dos genes dos indivíduos gerados por crossover e seus valores são ressorteados entre os possíveis valores para aquele gene.
Caso o valor ressorteado seja idêntico ao valor atual do gene, o sorteio é realizado novamente.

2.3 Servo Motores

Os servo motores do modelo Tower Pro SG90 foram escolhidos devido à facilidade de conexão com a placa Arduino.
Os pinos dos motores foram conectados diretamente à placa, sendo a alimentação (5V) também feita através dos pinos do Arduino. Os pinos de comando foram conectados a pinos de PWM da placa.
A IDE do Arduino em sua versão 1.6.13 conta com a biblioteca *Servo.h* que faz todo interfaceamento entre a placa e os servo motores. Os principais comandos utilizados são:

- Servo Servos[SERVO_MAX]: comando utilizado para criar os objetos dos servo motores dentro do código. O parâmetro SERVO_MAX é o número de servo motores conectados ao Arduino (no caso do projeto, esse número é igual a 4);
- Servos[i].attach(PINO_SERVO): comando utilizado para conectar o objeto criado no comando anterior ao pino da placa ao qual o motor está fisicamente conectado(PINO_SERVO);
- Servos[i].write(X): comanda o i-ésimo servo motor a ir para a posição X, medida em graus e deve estar entre 0 e 180;
- X = Servos[i].read(): armazena na variável X a posição atual do servo motor, em graus.

2.4 Sensor de distância

O sensor ultrassônico modelo HC-SR04 foi escolhido pela facilidade de conexão com a placa Arduino.
Os pinos de *trigger* e *echo* do sensor foram conectados a pinos de *i/o* digitais do Arduino e os pinos de alimentação foram conectados à fonte de alimentação de 5V.
O sensor funciona enviando um sinal digital no pino *echo* proporcional à distância percorrida pelo sinal de ultrassom, que é emitido pelo sensor ao receber um sinal digital de 10 microssegundos na porta *trigger*. O período medido do sinal da porta *echo* em microssegundos deve ser então convertido usando-se a Eq. (1).

$$f(x_{CM}) = \frac{período}{2 * 29{,}1} \qquad (1)$$

Onde *período* é o período do sinal medido em *echo* e $f(x_{CM})$ é a distância entre o sensor e o obstáculo medida em centímetros.
Para medir o período do sinal em *echo* é utilizada a função *pulseIn* com os parâmetros: pino do Arduino onde está conectado o pino *echo* do sensor, estado a ser medido entre *high* ou *low* e tempo de timeout da leitura do período.

2.5 Arduino

O Arduino foi escolhido por ser uma plataforma aberta, programável em linguagem C, com amplo suporte comunitário na Internet e por conter um relógio de tempo real (RTC, da expressão *Real Time Clock*) imprescindível para aplicações que dependam de tempo no mundo real.
O modelo Mega 2560 foi escolhido pela capacidade de memória necessária para armazenar e manusear as variáveis envolvidas no sistema classificador, algoritmo genético, servo motor e sensor.
Considerando somente o sistema classificador, contando os 8 genes (definidos como tipo *char*), as energias(*float*) e a especificidade(*float*) para 100 indivíduos, temos 1.600 bytes ocupados na RAM pelo "cérebro". Considerando as variáveis auxiliares e demais módulos do sistema, concluiu-se que o Arduino Uno (versão com menos memória e mais barato) não atenderia às necessidades do projeto.

3. Resultados

Primeiro vamos analisar o estado da tabela do sistema classificador depois de algumas iterações.

O sistema classificador é codificado da seguinte maneira:
antecedente : consequente | energia | especificidade

```
 anteced:conse|     energia    |  especificidade

00)110#:1100 | 049.8248558044 | Spec=0.7500000000
01)0#11:1100 | 047.4064254760 | Spec=0.7500000000
02)10#1:0011 | 055.0849456787 | Spec=0.7500000000
03)1###:0101 | 045.3093948364 | Spec=0.2500000000
04)#11#:1001 | 030.8557643890 | Spec=0.5000000000
05)111#:1000 | 026.5318050384 | Spec=0.7500000000
06)1111:0111 | 049.8268623352 | Spec=1.0000000000
07)0000:0010 | 049.8265724182 | Spec=1.0000000000
08)1011:1100 | 049.8266487121 | Spec=1.0000000000
09)#01#:0101 | 021.5331668853 | Spec=0.5000000000
10)1##1:1010 | 046.5724487304 | Spec=0.5000000000
11)#100:1001 | 047.3775482177 | Spec=0.7500000000
12)#0#0:0110 | 019.8173294067 | Spec=0.5000000000
13)00#1:1101 | 049.7985076904 | Spec=0.7500000000
14)0011:0101 | 071.0506057739 | Spec=1.0000000000
15)1#00:1110 | 041.5906867980 | Spec=0.7500000000
16)01##:0010 | 063.1211013793 | Spec=0.5000000000
17)##10:0100 | 020.5444183349 | Spec=0.5000000000
18)0100:1010 | 080.0138244628 | Spec=1.0000000000
19)#1#1:1110 | 077.3308868408 | Spec=0.5000000000
```

Figura 4. Estado do sistema classificador na iteração 1

```
00)#1#1:1110 | 065.2126083374 | Spec=0.5000000000
01)#1#1:1110 | 038.1486816406 | Spec=0.5000000000
02)1##1:1010 | 002.7116405963 | Spec=0.5000000000
03)10#1:0011 | 021.9146518707 | Spec=0.7500000000
04)#0#1:0100 | 000.0000000000 | Spec=0.0000000000
05)0011:0101 | 019.4625911712 | Spec=1.0000000000
06)00#1:1101 | 052.0470657348 | Spec=0.7500000000
07)1011:1100 | 014.3426389694 | Spec=1.0000000000
08)1011:1100 | 017.7427120208 | Spec=1.0000000000
09)00#1:1101 | 012.7669000625 | Spec=0.7500000000
10)0011:0101 | 024.3639259338 | Spec=1.0000000000
11)0011:0101 | 017.6087245941 | Spec=1.0000000000
12)1011:1100 | 012.3562135696 | Spec=1.0000000000
13)0011:0101 | 017.8955364227 | Spec=1.0000000000
14)1011:1100 | 013.9312162399 | Spec=1.0000000000
15)0011:0101 | 022.4450492858 | Spec=1.0000000000
16)1011:1100 | 010.5383605957 | Spec=1.0000000000
17)0011:0101 | 058.9776344299 | Spec=1.0000000000
18)0011:0101 | 017.6087245941 | Spec=1.0000000000
19)1011:1100 | 014.8579072952 | Spec=1.0000000000
```

Figura 5. Estado do sistema classificador na iteração 3

```
00)00#1:1101 | 008.7838258743 | Spec=0.7500000000
01)00#1:1101 | 006.4144792556 | Spec=0.7500000000
02)#1#1:1110 | 006.0592837333 | Spec=0.5000000000
03)00#1:1101 | 010.2312116622 | Spec=0.7500000000
04)00#1:1101 | 011.6796283721 | Spec=0.7500000000
05)00#1:1101 | 029.9311332702 | Spec=0.7500000000
06)00#1:1101 | 009.2707347869 | Spec=0.7500000000
07)00#1:1101 | 009.3829660415 | Spec=0.7500000000
08)00#1:1101 | 009.6513204574 | Spec=0.7500000000
09)00#1:1101 | 006.4144792556 | Spec=0.7500000000
10)00#1:1101 | 012.2322635650 | Spec=0.7500000000
11)00#1:1101 | 010.2312116622 | Spec=0.7500000000
12)00#1:1101 | 006.4144792556 | Spec=0.7500000000
13)00#1:1101 | 008.7838258743 | Spec=0.7500000000
14)00#1:1101 | 012.4185218811 | Spec=0.7500000000
15)#1#1:1110 | 062.8933448791 | Spec=0.5000000000
16)00#1:1101 | 006.4144792556 | Spec=0.7500000000
17)00#1:1101 | 009.3829660415 | Spec=0.7500000000
18)0011:0101 | 009.2113037109 | Spec=1.0000000000
19)00#1:1101 | 089.1935348510 | Spec=0.7500000000
```

Figura 6. Estado do sistema classificador na iteração 6

Cada iteração representa 100 leilões executados. Nota-se que em 600 iterações (tempo de execução de 33 minutos e 27 segundos) já emerge um padrão na ativação dos servo motores.

Um primeiro padrão notável que emerge é a presença do símbolo de don't care no antecedente do servo motor 3, como destacado no figura 7. Esse padrão identifica corretamente que a pata do servo motor 3 é levemente mais curta que as demais, não influenciando no deslocamento do robô.

```
00#1:1101
```

Figura 7. Destaque de um padrão emergente que identifica uma característica física

Outro padrão é a presença de duas regras dominantes, a regra 05 e a regra 15, sendo elas complementares.

O deslocamento do robô ao final de uma geração completa vai de aproximadamente 7 cm na iteração 1 até 15 cm na iteração 6.

Esse documento acompanha um vídeo que demonstra em tempo acelerado a execução da iteração 3 dos resultados aqui apresentados.

4. Conclusões

Considerando o problema de aprendizagem não supervisionada apresentado no item 1 deste documento, pode-se verificar que o sistema classificador proposto têm capacidade de encontrar uma configuração que o desloca para frente, sendo assim um candidato a compor o aprendizado de um sistema robótico variável. Sem que seja informado para o sistema ele foi capaz de identificar uma característica do hardware (a pata 3 mais curta).

Ainda há muito espaço para melhora, sendo que a manutenção da variabilidade genética e a exploração do espaço de possibilidades após muitas iterações podem ser linhas a serem seguidas em desenvolvimentos futuros.

Referências

[1] Levy Boccato. Notas de aula da disciplina IA013. http://www.dca.fee.unicamp.br/~lboccato/ia013_2s2017.html (acessado em 30/11/2017)

[2] Patrícia A. Vargas. Sistemas classificadores para redução de perdas em redes de distribuição de energia elétrica, 2000. http://www.dca.fee.unicamp.br/~vonzuben/research/pvargas_mest.html (acessado em 30/11/2017)