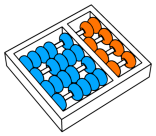


Aprendendo sobre o Espaço Atencional no CST - The Cognitive Systems Toolkit

C. Regattieri *E. Colombini*

Relatório Técnico - IC-PFG-18-31
Projeto Final de Graduação
2018 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Aprendendo sobre o Espaço Atencional no CST

-

The Cognitive System Toolkit

Carolina Regattieri de Biase Nicolini Delgado[1], Esther Colombini[2]

[1] NeuralMind, carolina.regattieri@neuralmind.ai

[2] Instituto de Computação Universidade Estadual de Campinas (UNICAMP), Caixa Postal 6176

13083-970 Campinas-SP, Brasil

esther@ic.unicamp.br

Resumo. O objetivo deste projeto de final de curso foi explorar os módulos de atenção baseados no CONAIM (Conscious Attention-Based Integrated Model) recentemente incorporados ao CST (Cognitive Systems Tool-kit). O CST é um framework geral para a construção de arquiteturas cognitivas que permite a utilização e integração de diversas tecnologias. Nosso propósito com este estudo é validar os processos atencionais definidos pelo CONAIM que foram implementados no CST na forma de módulos. Para isso, um conjunto de experimentos utilizando aprendizado de reforço foram definidos e implementados em um simulador robótico de alta fidelidade, e assim foi possibilitada a validação dos processos atencionais.

Palavras-Chave: Aprendizado por reforço, QLearning, CST, Modelo Atencional, CONAIM

1. Introdução

Ao longo da evolução da humanidade, e principalmente no mundo moderno, a mente foi vista como um artefato de imensa complexidade e importância em nossas vidas. Indagar sobre seu funcionamento trás a tona questões filosóficas que talvez nunca seremos capazes de sanar, mas diversas teorias foram e estão sendo criadas a fim de embasar, compreender e reproduzir como esta máquina super potente que está sempre presente conosco funciona.

A filosofia foi a pioneira na tentativa de explicar como o conhecimento e o comportamento humano se organizavam e o que era exatamente a cognição. Mais recentemente, cientistas voltados para áreas tecnológicas estão explorando extensivamente o assunto e a robótica foi uma das áreas que mais avançou em estruturar inteligência em agentes, com objetivos de executar as tarefas das mais simples até as mais complexas.

A fim de reproduzir em seres artificiais a competência humana de aprender, planejar, decidir e entender certos aspectos do ambiente, foi preciso ao longo do tempo desenvolver modelos teóricos que mapeiam como o cérebro realiza tais processos. Neste contexto, muitos modelos para diferentes capacidades inteligentes vieram a tona: aprendizado, atenção, tomada de decisão, memória, percepção, geração de comportamento, etc.

Ao associar um modelo cognitivo a sistemas computacionais, temos as chamadas arquiteturas cognitivas, que normalmente juntam tanto aspectos teóricos quanto frameworks de software para que possam ser utilizadas em diferentes aplicações.

Como arquitetura cognitiva utilizamos neste projeto o Cognitive System Toolkit [1], [2]. O CST possui vários componentes implementados, sendo necessária sua adaptação para a aplicação em questão, e este trabalho utiliza somente os componentes de memória, atenção e aprendizado para realizar experimentos.

Como modelo para embasar nossa implementação, escolhemos o Modelo Atencional CONAIM [3]. Ser apto a focar seu ciclo cognitivo em partes diferentes do ambiente ao longo do tempo é algo muito corriqueiro para o ser humano, algo que nem percebemos no nosso dia a dia, mas para robôs é um aspecto essencial. A quantidade de sensores e dados que chegam até eles

tende a ficar cada vez maior, e por isso é necessário um meio de decidir quais destes dados e medidas são mais importantes naquele momento.

Módulos atencionais em robôs possuem diversas aplicações, especialmente quando relacionadas a alguma inteligência. A capacidade de focar a atenção no objeto mais rápido de uma cena, por exemplo, pode compor um robô que tem como função perseguir carros que andam em alta velocidade nas estradas. Além disso, agentes que são capazes de absorver detalhes diferentes da cena ao longo do tempo são muito mais adaptáveis, já que esta é uma das maiores dificuldades da robótica fora de ambientes simulados: cenas que se alteram muito e não seguem regras pré-definidas.

Para demonstrar a usabilidade da atenção em agentes, vamos utilizar o output do nosso modelo atencional como entrada para um componente de aprendizado que deverá aprender a realizar um conjunto de tarefas via interação com o ambiente.

2. Objetivo

O objetivo deste Trabalho Final de Graduação é realizar um conjunto de experimentos para validar a implementação do módulo atencional do CST baseado no CONAIM e verificar a possibilidade de que um robô aprenda comportamentos inteligentes sobre o espaço atencional ao invés do espaço de features. Para tal, serão utilizadas features obtidas a partir da execução do ciclo atencional do CONAIM no CST a fim de que o robô aprenda a realizar tarefas pré-definidas.

Mais especificamente objetivamos:

1. Estudar o modelo atencional proposto pelo CONAIM
2. Estudar a implementação de 1. No CST
3. Construir features a partir dos dados sensórios
4. Combinar as features para construção dos mapas atencionais e de saliência
5. Validar individualmente as features assim como a composição das mesmas
6. Implementar o módulo de aprendizado por reforço no CST sobre o espaço atencional
7. Executar experimentos para aprendizado de tarefas

3. Materiais

Todos os experimentos foram realizados utilizando o simulador de alta finalidade V-REP. Para que fosse possível executar o código do CST no simulador, foi necessário configurar sua remoteAPI do cliente [4], isto é, além de adicionar os arquivos especiais da API na pasta do projeto, foi preciso adicionar as bibliotecas no caminho do Java: *-Djava.library.path=*.

Um robô do tipo Pioneer P3DX serviu como agente atencional capaz de percorrer o ambiente e aprender como chegar ao seu objetivo através de features atencionais. O robô estava equipado de 16 sensores sonares, 8 deles nos 180º da parte dianteira, e os outros 8 nos 180º da parte traseira, além de um sensor laser posicionado acima de sua lataria, no centro. O sensor de laser era capaz de obter 180 leituras a cada ciclo, referente aos dados da parte frontal do robô.

3.1 CST - The Cognitive System Toolkit

O CST possui diversos módulos implementados, mas existem dois conceitos primordiais na sua arquitetura: codelets e memory objects, conforme apresentado na Figura 1.

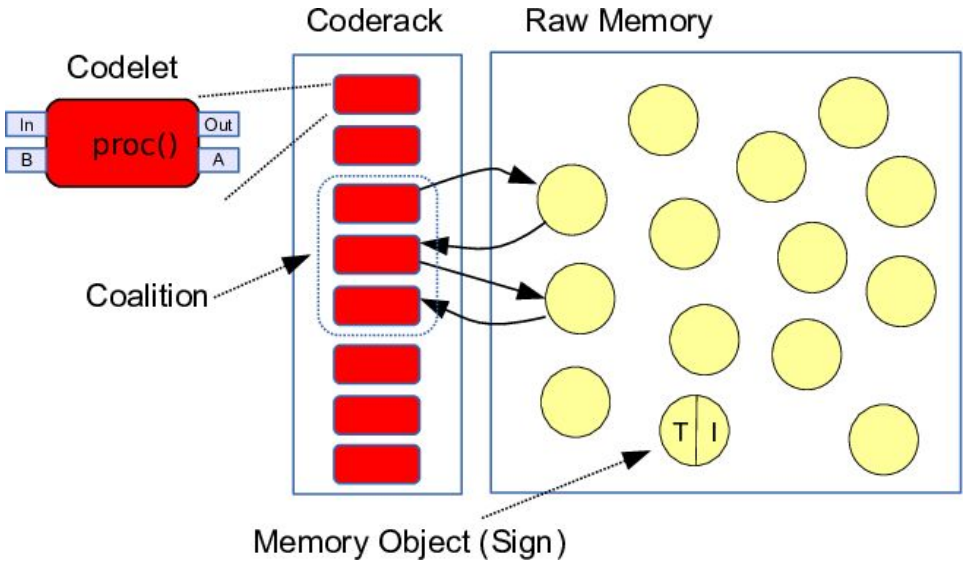


Figura (1) Representação esquemática dos codelets do CST. Referência: [5]

Codelets são pequeno componentes responsáveis por pequenas partes de processamento do sistema cognitivo. Eles podem ser por exemplo, divididos em codelet de percepção, codelet atencional, codelet de aprendizado, etc, variando com a aplicação. Cada codelet possui seu próprio input e output, que são memory objects, e estão agrupados em um Coderack. A cada ciclo atencional, todos os codelets de um codeRack serão executados, recebendo seus memory objects de entrada, fazendo seu devido processamento, e escrevendo em seus memory objects de saída.

3.2 O modelo atencional

Como proposto por [6], o modelo atencional utilizado neste projeto foi o definido pelo CONAIM. Existem diversos tipos de modelagem que visam explicar como funciona a atenção humana, mas este modelo o faz separando o processo atencional em dois componentes: bottom-up e top-down. O componente bottom-up se responsabiliza por processar os estímulos do ambiente tais como eles vem, dando importância para aqueles que mais se sobressaem. No top-down, existe um estímulo externo, uma tarefa ou objetivo, que guia a atenção do agente. Para este projeto, iremos abordar somente o componente bottom-up. A Figura 2 ilustra o funcionamento do sistema.

A cada ciclo atencional, espera-se que inúmeros dados de diversos sensores sejam captados e partir dos mesmos são construídos mapas de características (ou features). Para condensar tais dados, é preciso gerar um mapa de features combinadas, que deixe todas as features com a mesma dimensão em um só mapa. Da perspectiva bottom-up, o mapa de features serve como um indício do que deverá ser percebido no ambiente. Ao decidir qual das regiões do mapa de saliência é a vencedora, a partir da estratégia Winner Takes it All (ou seja, a feature mais sobressaltada vence), o vencedor é adicionado a uma lista de vencedores e seu ciclo atencional se inicia. Durante certo período de tempo, a região vencedora será realçada pelo mapa atencional, e depois entrará em um período inibitório. Durante este período, esta região fica como negativa no mapa atencional, para que outras possam ser atendidas pelo agente. Este processo é chamado de Inibição de Retorno (Inhibition Of Return - IOR). O mapa de saliência é

resultado da multiplicação do mapa atencional com o mapa de features combinadas, e será utilizado para uma super representação do ambiente percebido.

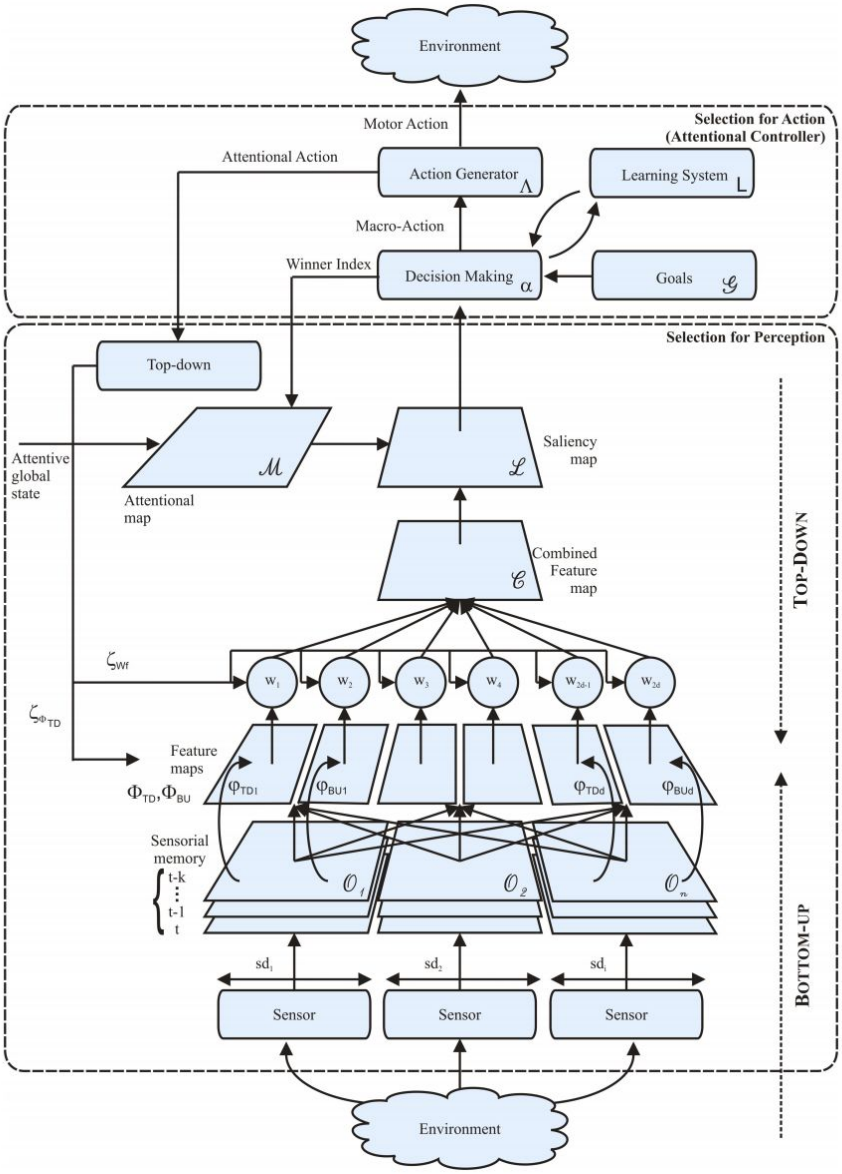


Figura (2) Ilustra o funcionamento do modelo atencional CONAIM. Referência: [6]

3.3 Implementação do modelo atencional no CST

Como mencionado previamente, o código base para o desenvolvimento dos experimentos foi desenvolvido a partir do Módulo Atencional do CST [7] e do próprio CST na branch atencional [8].

A implementação do modelo atencional proposto no CST segue a estrutura apresentada na Figura 3.

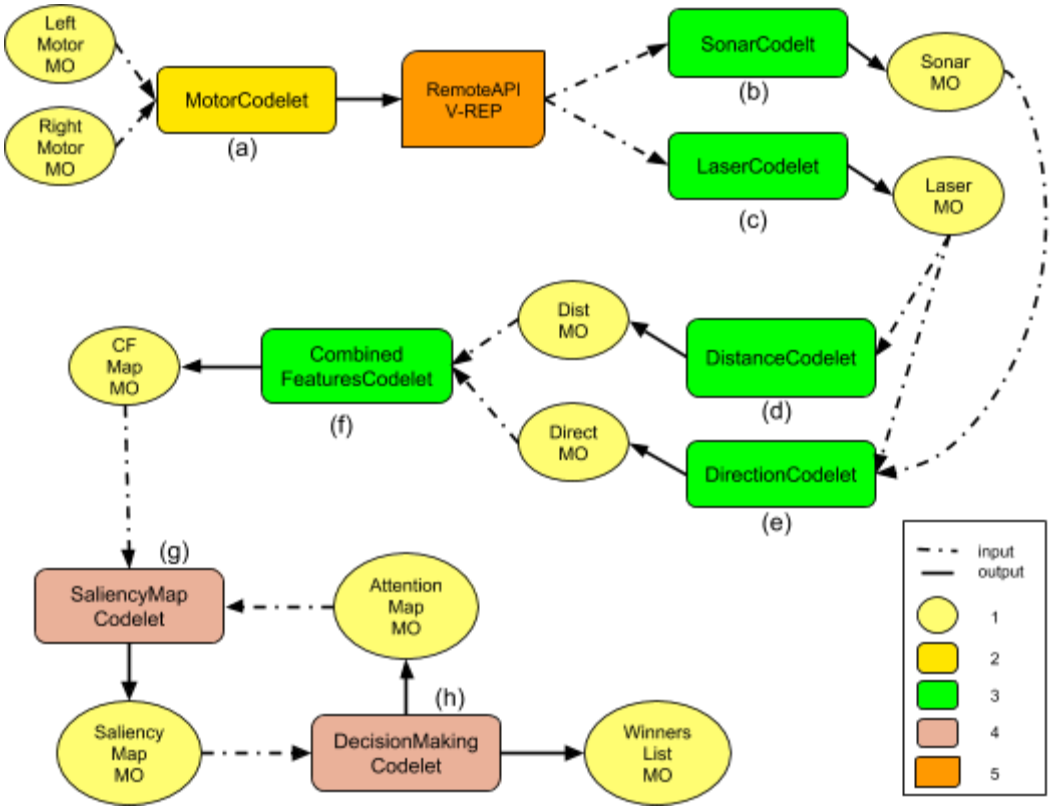


Figura (3). (1) Representa Memory Objects do CST. (2), (3) (4) Representa Codelets Motores, Sensoriais e Atencionais, respectivamente e (5) Representa as classes de comunicação externa com a remoteAPI do V-REP, que acessa o robô.

Os codelets mais primitivos são representados pelos MotorCodelet (a), SonarCodelet (b) e LaserCodelet (c). Estes componentes estão diretamente ligados com a remoteAPI do simulador V-REP e fazem a comunicação direta com o Pioneer, ou seja, eles interagem diretamente com o ambiente de estudo.

Pode-se perceber, portanto, que os dados brutos de distância vem diretamente dos Memory Objects do SonarCodelet e do LaserCodelet e são, então, utilizados para computar os valores de distância e direção do robô, através dos codelets DirectionCodelet (d) e DistanceCodelet (e). A distância é um cálculo direto dos valores computados pelo laser (as medidas são apenas normalizadas), enquanto que a direção é obtida através de uma combinação entre os valores de laser e sonar, e sua dimensão final é igual à dimensão do sonar: oito, referentes aos sonares dianteiros.

Estes valores são alocados em seus respectivos Memory Objects e passados para o codelet CombinedFeatureCodelet (f), cuja função é combinar estas duas features em um único espaço dimensional (que será novamente de tamanho oito, por conta dos sonares dianteiros). Assim o Memory Object que representa o output do codelet (f) são as features que escolhemos para caracterizar o ambiente, combinadas. Estas features foram distância (computada pelo laser) e direção (computada como um conjunto de laser e sonar).

Estes passos anteriores foram feitos somente para a aquisição de features, e o mapa final de features combinadas será passado como entrada para o SaliencyMapCodelet (g). A finalidade deste codelet é somente multiplicar o mapa atencional (que é inicializado com valores unitários, simulando um ambiente sem vencedores), pelo mapa de combined features.

A saída deste codelets, que é o mapa de saliências, é passada como entrada para o último codelet: DecisionMakingCodelet (h), que implementa em si o modelo atencional proposto. A partir do mapa de saliência, a função principal deste codelet irá calcular a feature vencedora daquele instante de tempo v_t, adicioná-la à lista de vencedores e verificar as curvas de atenção para todos os vencedores da lista, para assim poder atualizar o mapa atencional, realimentando o ciclo.

Todos codelets ficam contidos em um CodeRank, e após o início do ciclo atencional, a função principal de todos eles roda em Threads paralelas, alimentando seus devidos Memory Objects de output.

4. Modelo Proposto

A fim de realizar experimentos inteligentes no módulo atencional, o primeiro passo foi verificar se seu funcionamento básico estava correto. Esta validação refere-se ao Experimento 1.

Após esta análise, nossa proposta foi implementar um codelet inteligente, que utilizasse algum algoritmo de Aprendizado por Reforço [9] para realizar tarefas. O CST já possuía uma implementação de QLearning [10] interna, entretanto foi necessário modelar como as saídas do módulo atencional poderiam alimentar o algoritmo.

4.1 QLearning

QLearning é um algoritmo de Aprendizado por Reforço que capacita agentes a aprender a agir de forma ótima em ambientes dinâmicos. Algoritmos de aprendizado de reforço tem por objetivo aprender a realizar uma tarefa pré-determinada maximizando o valor total de recompensas que são obtidas realizando certas ações e consequentemente atingindo determinados estados.

Um agente em um ambiente tem como conjuntos de ações A, por exemplo, e pode encontrar-se em um conjunto de estados S no ambiente. Ao realizar uma ação $a \in A$ que leva de um estado a outro $s_1 \rightarrow s_2$ sendo $s_1, s_2 \in S$, é recebida uma recompensa r numérica. Assim, ao final de inúmeras iterações, o agente vai ser capaz de deduzir o melhor conjunto de ações que leva ao estado final (que completa a tarefa) com o valor ótimo de recompensas.

Para realizar o processo de aprender qual é o conjunto de ações ótimos para o problema, o QLeaning mantém uma tabela de dimensões $S \, x \, A$ onde cada elemento x_{ij} representa a um valor relacionado ao estado s_i inicial a ação a_j que foi executada para sair daquele estado. A cada iteração este valor é atualizado seguindo a Equação 1.

$$Q(s,a)= Q(s,a) + \alpha * (r(s) + \gamma * \max_{a'}(s',a') - Q(s,a))$$

Eq. 1

Onde α é a taxa de aprendizado e γ é a taxa de desconto temporal. Após a alteração na tabela, a próxima ação ótima é selecionada, escolhendo-se dentre duas opções: a ação cujo valor correspondente ao estado corrente é máximo, ou uma ação aleatória. A possibilidade de escolher entre estas alternativas permite o agente a explorar o conhecimento que já adquiriu sobre o ambiente e também a explorar novas partes do ambiente ainda desconhecidas.

Com este mecanismo, é possível aprender a melhor política de seleção de ações para qualquer tarefa.

5. Implementação

5.1 LearnerCodelet

Um novo codelet, chamado de LearnerCodelet, foi implementado para utilizar a classe QLearning e reunir os componentes de memória necessários para fazer o agente atencional aprender. Na Figura 4 está o esquema final de codelets e classes após a adição de nossas modificações.

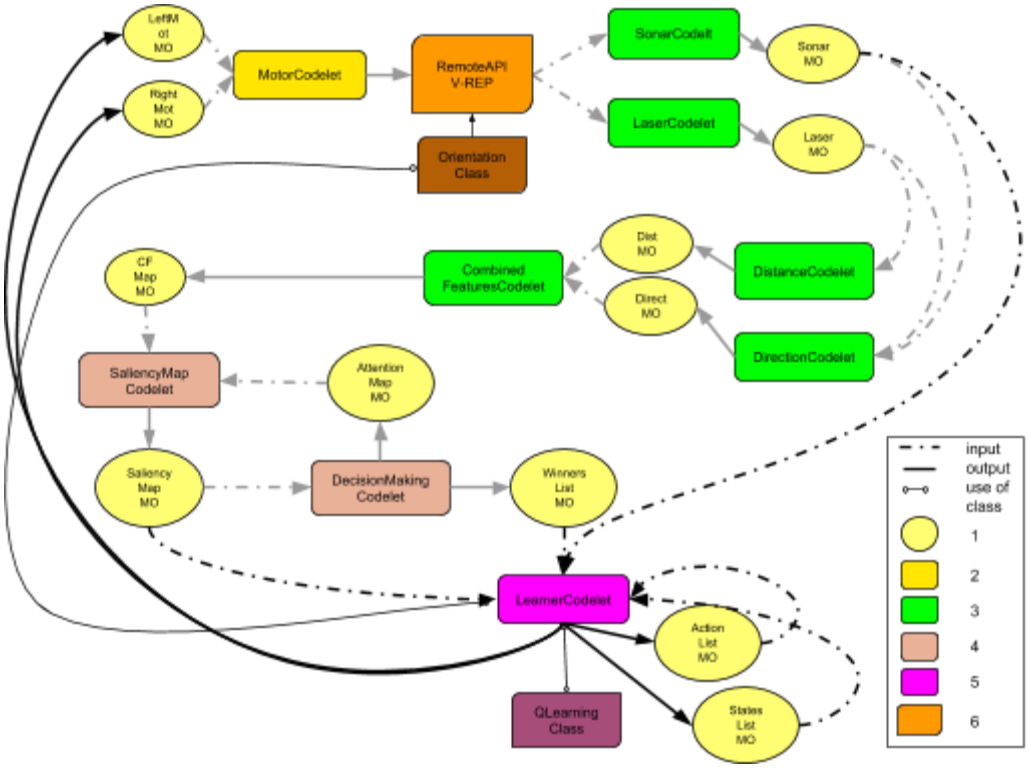


Figura (4).: (1) Representa Memory Objects do CST. (2), (3), (4) e (5) Representa Codelets Motores, Sensoriais e Atencionais e de Aprendizado, respectivamente e (6) Representa as classes de comunicação externa com a remoteAPI do V-REP, que acessa o robô.

Para alimentar o algoritmo de aprendizado de reforço escolhido, foi necessário definir algumas diretivas: **ações**, **estados** e **recompensas**.

A lista de ações está diretamente relacionada à tarefa que o agente deve aprender a realizar. Por exemplo, se essa tarefa for seguir o vencedor encontrado pelo módulo atencional, um possível conjunto de ações pode ser o listado seguindo a Equação 2.

$$A = \{ a_1, a_2, a_3 \}$$

$$a_1 \rightarrow Andar\ até\ o\ v_t$$
$$a_2 \rightarrow Girar\ em\ direção\ ao\ v_t$$
$$a_3 \rightarrow Fazer\ nada$$

$$v_t \rightarrow \text{Índice da feature vencedora do instante t, onde indice} \in \{ [0,7] \}$$

Eq. 2

As ações a_1 *e* a_2 traduzem-se em ações para o motor, assim os objetos de memória referentes à velocidade dos motores deveriam ser saídas do LearnerCodelet. Quando a ação selecionada pelo algoritmo era a_1, a mesma velocidade era passada para estes dois objetos.

A fim de decidir para qual lado o agente deveria rodar quando ação a_2 era selecionada, foi necessário também ter acesso à orientação do Pioneer no V-REP, e da orientação do sonar referente à feature de v_t.

Obtidos estes valores, foi calculada a diferença entre eles e uma velocidade fixa foi setada na roda correta. A figura 5 apresenta o esquema lógico correspondente.

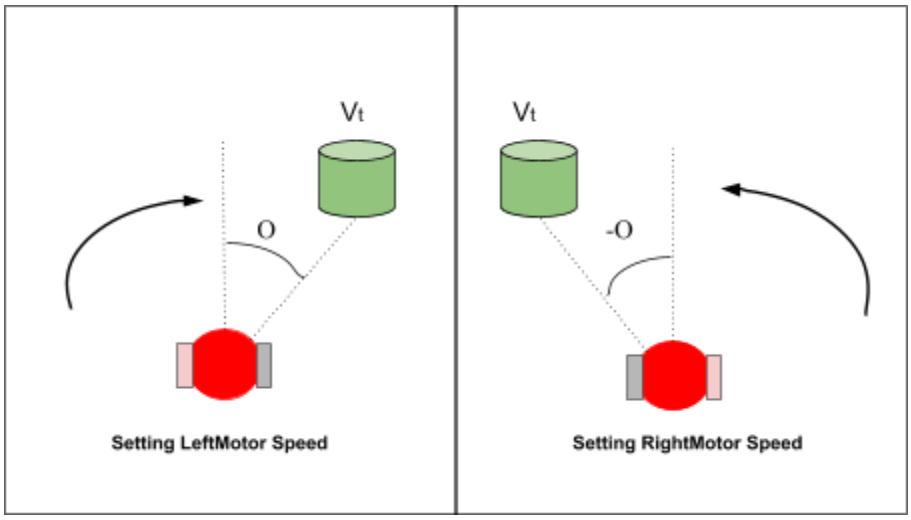


Figura (5) Esquema que ilustra a lógica de decisão de giro do robô

Para adquirir a orientação do agente via V-REP, foi criada uma nova classe OrientationClass, que conseguia capturar estes valores no simulador.

Os possíveis estados estão diretamente ligados ao mapa de saliências S. O mapa possui 8 dimensões possíveis, e portanto foi necessário mapear estes valores para um só, que representasse o estado naquele instante de tempo: $S_t \Rightarrow E_t$.

Como indicado na Figura 6, para realizar o procedimento de mapeamento, primeiramente é preciso normalizar cada um dos valores de S_t entre 0 e 1. Para calcular o valor normalizado, utiliza-se a fórmula apresentada na Equação 3.

$$n_i = \frac{s_i - min(S_t)}{max\,(S_t) - min\,(S_t)} \qquad \text{Eq 3}$$

Para transformar estes 8 valores em um só, é preciso escolher uma base numérica para servir de multiplicador. Em nossos experimentos escolhemos a base 5. Assim, cada valor n_i é discretizado em 5 intervalos possíveis, dando origem ao mapa intermediário N_t. Cada valor desse mapa é multiplicado por seu respectivo multiplicador da base 5, e quando somados eles dão origem ao estado E_t.

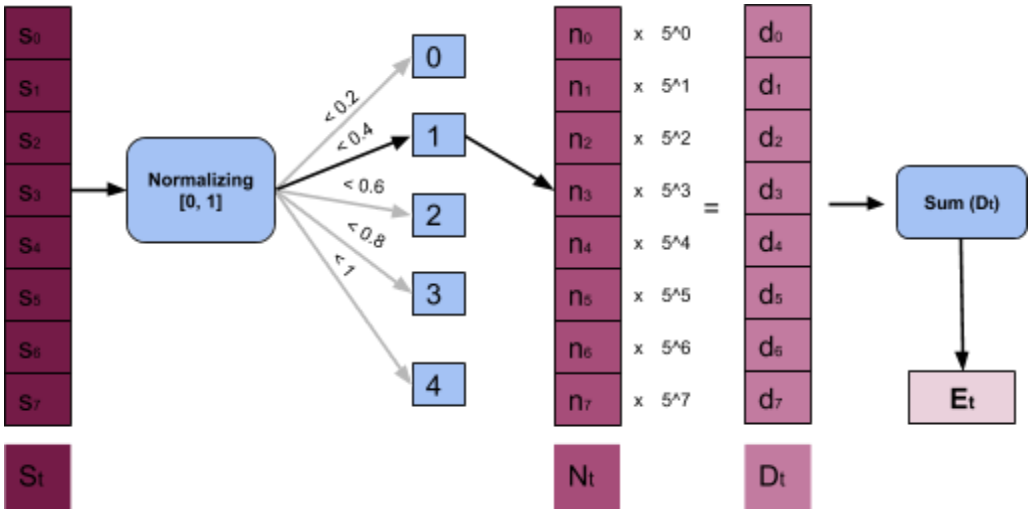


Figura (6). Esquema que ilustra o processo de discretização do mapa de saliência em um único estado

Portanto, dado um mapa de saliências, conseguimos inferir seu respectivo estado. Os estados, dada a escolha de base 5, variam entre 0 e 5^8 e esta é uma das dimensões da tabela do algoritmo de QLearnig. A outra dimensão é a mesma dimensão das Ações, ou seja, 3.

Para realizar o aprendizado, ou seja, atualizar a tabela em cada iteração, o algoritmo precisa ainda da recompensa que aquele estado ocasiona. A recompensa varia muito dependendo do objetivo do experimento. Um exemplo seria a Equação 4.

$$r_t = \{ \quad 10.0, \ se\ v_t\ foi\ atingido\ em\ t\ , \\ -10.0, \ se\ outro\ objeto\ foi\ atingido\ em\ t\ , \\ 0, \ caso\ contrário\ \} \qquad \text{Eq. 4}$$

A averiguação para saber se o agente atingiu qualquer objeto é feita através dos valores brutos dos sonares, já que eles possuem a mesma dimensão de todos os mapas atencionais e da lista de vencedores. Foi definido um valor de threshold para determinar se uma leitura do sensor era próxima demais e significava uma batida naquela posição.

No início de cada iteração do LearnerCodelet, primeiramente eram adquiridos o estado e ação do instante anterior e_{t-1} *e* a_{t-1}. Com estas variáveis em mãos, e calculando-se a recompensa r_t que o instante atual proporciona, a tabela de aprendizado era atualizada na posição e_{t-1} *e* a_{t-1} com o valor r_t.

Após a atualização, a próxima ação era selecionada através da função getAction() passando-se e_t como estado. A velocidade definida através da ação era passada para os objetos de memória dos motores, e o agente se atualizava e se locomovia no ambiente, reiniciando o ciclo.

6. Desenvolvimento dos Experimentos

Cada experimento possuía um objetivo específico mas os que utilizaram aprendizado para validação dos módulo tinham uma estrutura comum. Um processo de aprendizado por reforço pode ser dividido em duas partes: aprendizado e exploração. Para aprender, é necessário fazer uma série de rodadas de simulações, sorteando aleatoriamente a posição inicial do agente e definindo uma condição de parada para ressetar a rodada e iniciar uma nova simulação,

mantendo-se a tabela Q. Foi estipulado um número de 500 ações por rodada como número máximo.

Além disso, alguns parâmetros do algoritmo de QLearning foram comuns entre os experimentos de aprendizado, sendo:

$$\textit{Action Grid} = 0.1$$
$$\alpha = 0.5$$
$$\gamma = 0.9$$
$$\beta = 0.95$$

O valor de *Action Grid* se refere a probabilidade de o algoritmo escolher a melhor ação ao invés de uma ação randômica. Escolher somente a melhor ação em todas as iterações significa que o agente não será capaz de explorar a tabela inteira. O valor *Alpha* se refere ao learning rate, *Gamma* ao discount factor e *Beta* a probabilidade da escolha randômica de ação selecionar a não anterior ao invés de uma ação verdadeiramente randômica.

Ambos os experimentos de aprendizado utilizaram a cena ilustrada pela Figura 17 como ambiente.

6.1 Experimento de Validação Básica dos Módulos Atencionais

Como um primeiro experimento de validação básica, propusemos uma simulação simples, com o robô pioneer parado em um ambiente estático com um único objeto cruzando seu campo de visão.

Este experimento teve como objetivo testar o funcionamento dos módulos atencionais do framework, e verificar se eles estavam funcionando como o proposto. É de extrema importância que todos os codelets estejam funcionando de forma correta, já que se houver algum erro de implementação neste estágio, é impossível desenvolver qualquer tipo de feature adicional no modelo.

Os codelets mais básicos de sensores (sonar e laser) devem coletar as informações do ambiente e passá-las para os codelets de features (distância, direção e o combined features). O codelet que monta o mapa de saliência deve estar multiplicando corretamente o mapa de features com o mapa atencional. Por fim, o codelet DecisionMaking, o mais complexo, deve montar corretamente o próximo mapa atencional e definir o vencedor daquele timestamp, baseado no mapa de saliência e nos ciclos inibitórios e excitatórios do modelo Bottom Up.

Para que fosse possível rodar a iteração completa do ciclo atencional, foi necessário realizar alterações no codelet DecisionMaking. Primeiramente, de acordo com o modelo teórico, uma feature só é adicionada à lista de vencedores caso ela seja a vencedora do mapa de saliência e ela não estava previamente na lista. Esta última condição se traduz em: a feature só pode voltar a entrar na lista de vencedores caso ela já tenha passado por seu ciclo excitatório, inibitório, e ela tenha saído da curva atencional. No código anterior, isso não estava ocorrendo, já que para o ambiente estático do experimento, a cada iteração a mesma feature era adicionada novamente como vencedora. Esse detalhe foi então corrigido. Outros aspectos pequenos do código foram alterados: constantes da curva gaussiana e ampliação correta da curva nas features que não foram vencedoras.

Para validar este experimento, foram gerados gráficos de todos os mapas gerados por cada um dos codelets mencionados.

6.2 Experimento de Utilização do Codelet de Aprendizado

O segundo experimento visava em realmente testar e validar o modelo atencional utilizando sua saída (o mapa de saliências) como entrada para o algoritmo de aprendizado. O objetivo era fazer o robô aprender a explorar ao máximo o ambiente, utilizando somente o mapa de saliência.

Definimos que uma simulação iria se iniciar novamente quando o robô se chocasse com qualquer objeto ou um número de 500 ações fossem completadas. O sistema de recompensa foi definido segundo a Equação 5.

$$r_t = \{ \quad 1, \textit{ se nenhum objeto foi atingido em } t \ \} \qquad \text{Eq 5}$$

Assim, quanto mais ações fossem realizadas sem colisão, mais alta seria a recompensa acumulada. Caso o agente se chocasse com algum objeto antes do fim das 500 ações, sua recompensa acumulada seria menor.

Com estes dados de recompensa acumulada por rodada, foi possível plotar um gráfico que mostra a curva de aprendizado do robô.

Para validar a política de escolhas que o agente aprendeu, rodamos algumas vezes a simulação, dessa vez em modo de exploração, somente escolhendo a próxima ação baseado nos valores da tabela Q, sem atualizá-la com recompensas.

6.3 Experimento de Tentativa de Melhoria do Sistema de Aprendizado

Como último experimento, foram feitas algumas alterações em pequenos parâmetros visando melhorar a política aprendida no Experimento 6.2.

Primeiramente, o conjunto de ações foi modificado para atender melhor ao objetivo: ao invés do conjunto de ações citado na seção anterior, modificamos a configuração para:

$$A = \{ a_1, a_2, a_3 \}$$

$$a_1 \rightarrow \textit{Andar até o } v_t$$
$$a_2 \rightarrow \textit{Girar em direção oposta ao } v_t$$
$$a_3 \rightarrow \textit{Fazer nada}$$

$$v_t \rightarrow \textit{Indice da feature vencedora do instante } t, \textit{ onde indice } \in \{ [0, 7] \}$$

Assim, a informação de quem é o vencedor ajuda o agente a não se colidir com ele, pois ele pode girar em direção oposta.

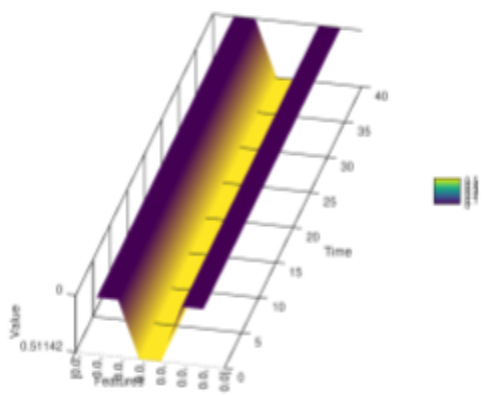
Além disso, aumentamos o número de rodadas para 100 no total, deixando a recompensa acumulada máxima em 50.000, ao invés de 25.000 como na seção anterior.

Os mesmos gráficos foram gerados para comparação com o Experimento 6.2.

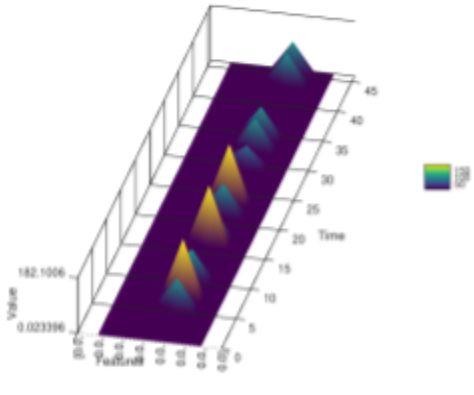
7. Resultados

7.1 Resultados Experimento 6.1

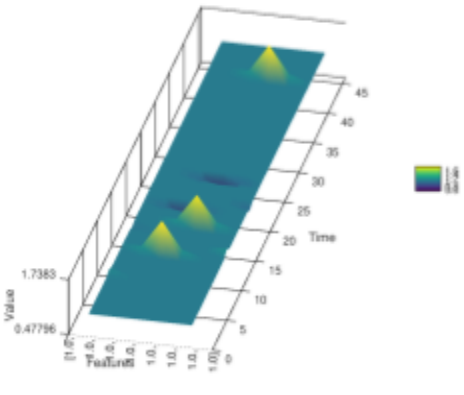
Neste experimento foram realizados testes para avaliação dos codelets de Sensores, CombinedFeature (Distância + Direção), de Saliência e Atencional. A Figura 7 mostra as saídas de cada mapa gerado com relação ao tempo. Eles foram coletados em uma cena simples e estática (seu printscreen está na Figura 8).



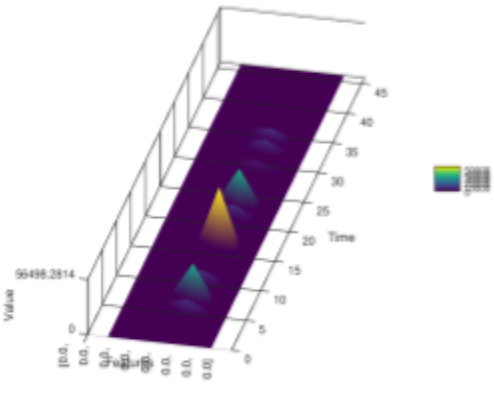
Sonar Data



CombinedFeature Map



Attentional Map



Saliency Map

Figura (7). Outputs dos codelets de Sonar, CombinedFeature, Attentional e Saliency com relação ao tempo

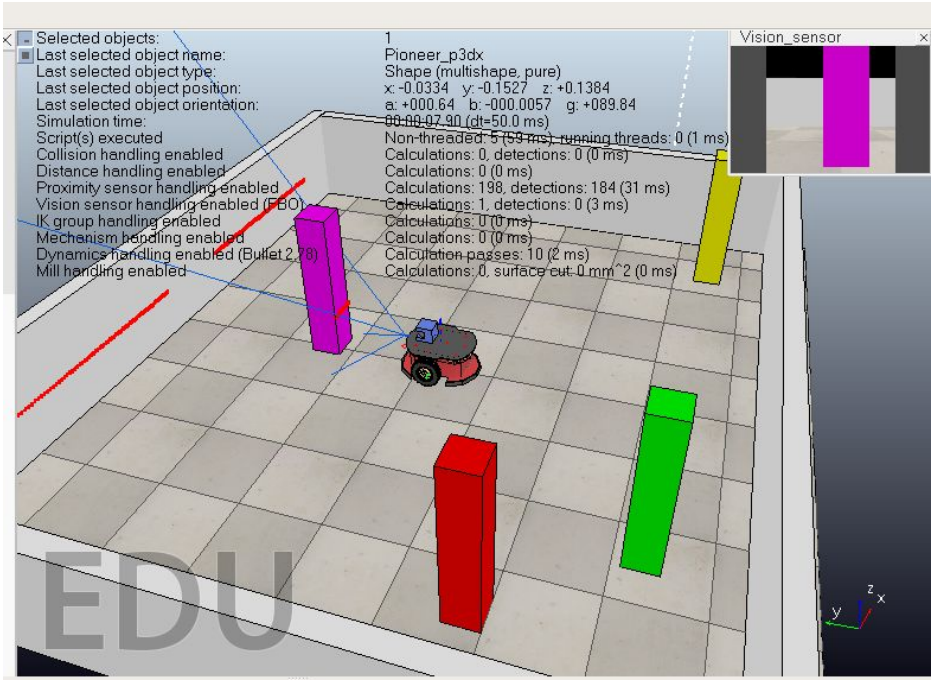


Figura (8) Cena estática utilizada no experimento 6.1

7.2 Resultados Experimento 6.2

Este experimento tem como objetivo validar o aprendizado utilizando QLearning realizando 50 rodadas, cada uma com 500 ações no máximo ou até colidir. Toda ação gerava uma recompensa de 1, e o agente poderia girar na direção da feature vencedora.

Foram plotados a curva de aprendizado (reforço acumulado com relação ao tempo) na Figura 9. As Figuras 10, 11 e 12 apresentam trajetos realizados pelo robô utilizando a política aprendida a partir de poses iniciais distintas.

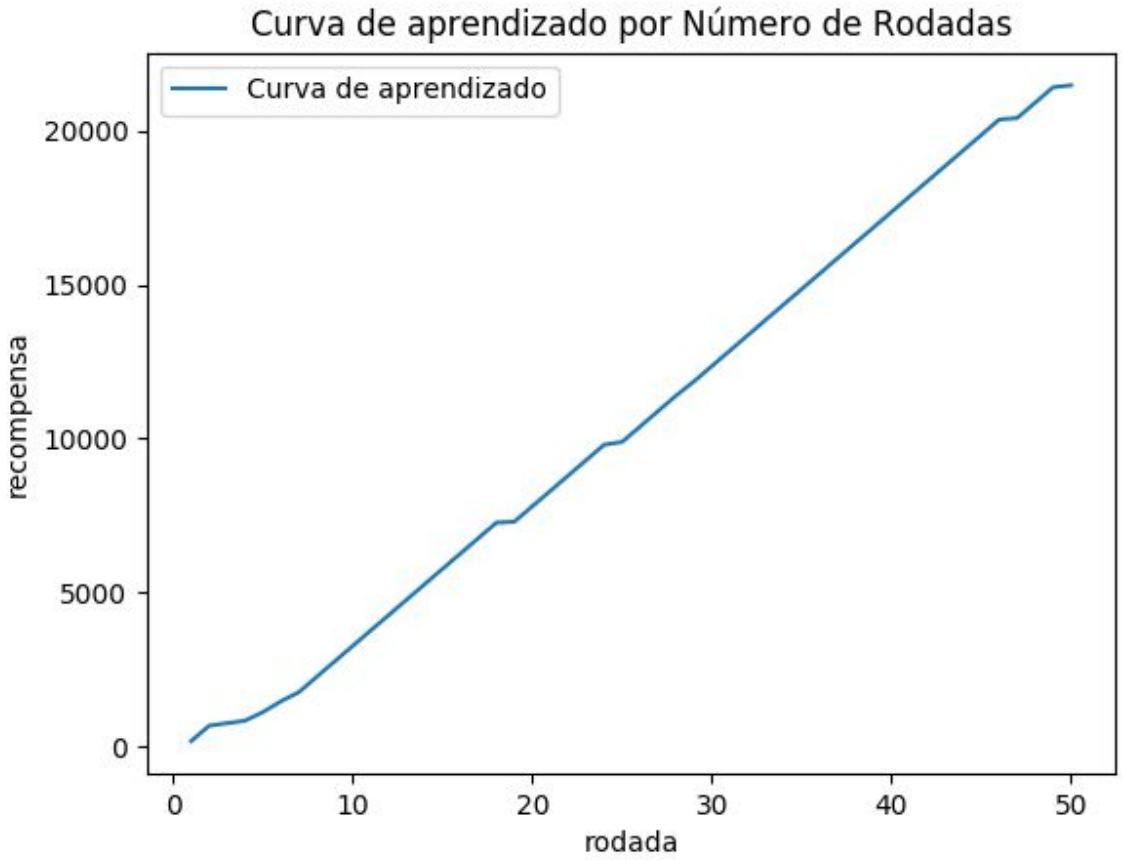
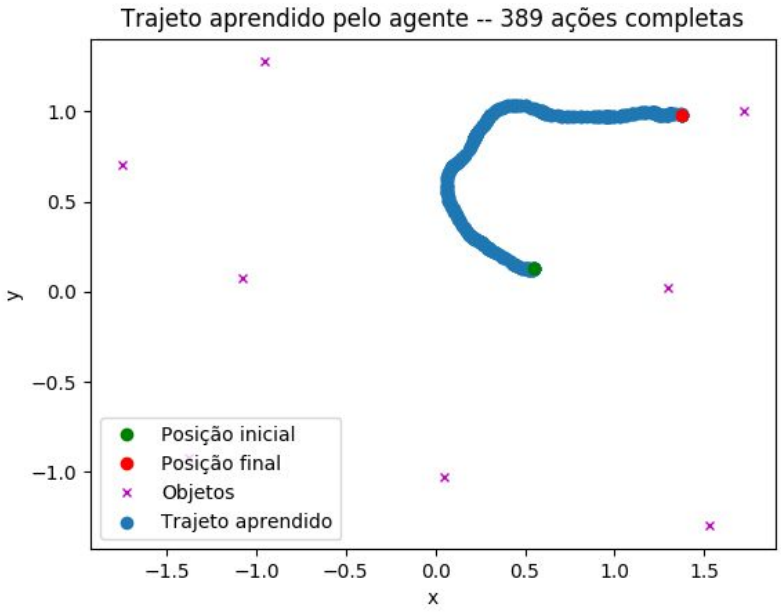
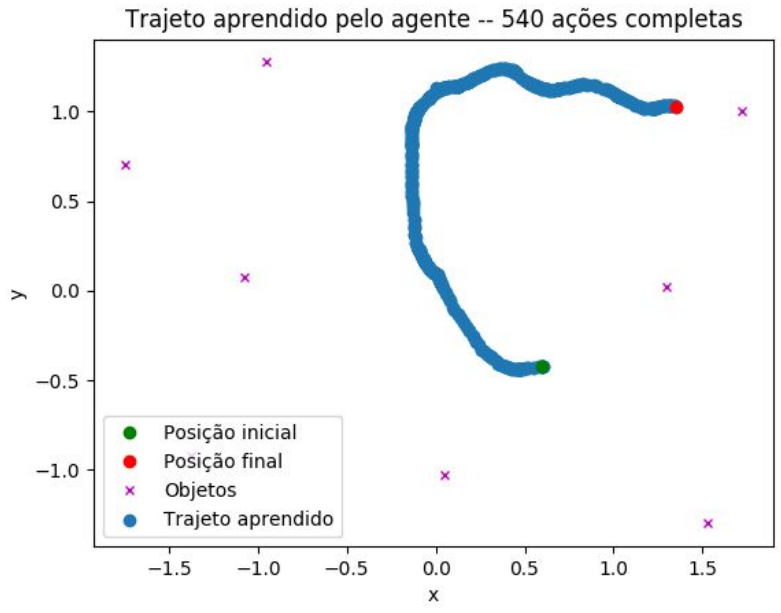


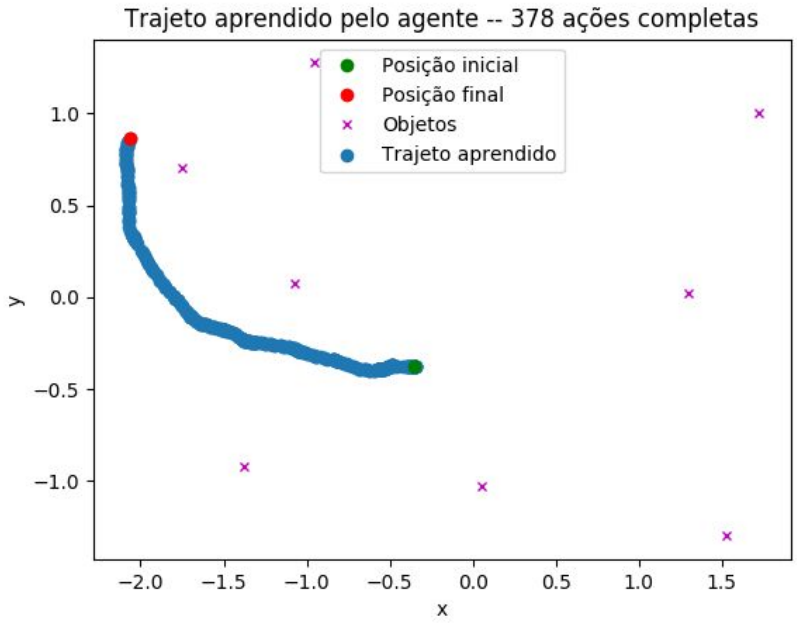
Figura (9) Curva de aprendizado no experimento 6.2



Figuras (10) Trajeto aprendido no experimento 6.2



Figuras (11) Trajeto aprendido no experimento 6.2



Figuras (12) Trajeto aprendido no experimento 6.2

7.3 Resultados Experimento 6.3

Neste experimento, o algoritmo de aprendizado utilizando QLearning foi executado por realizando 100 rodadas, cada uma com 500 ações no máximo ou até colidir. Toda ação gerava uma recompensa de 1, e o agente poderia girar somente na direção oposta a da feature vencedora.

Foram plotados a curva de aprendizado (reforço acumulado com relação ao tempo) na Figura 13 e trajetos realizados pelo robô utilizando somente a política aprendida nas Figuras 14, 15 e 16.

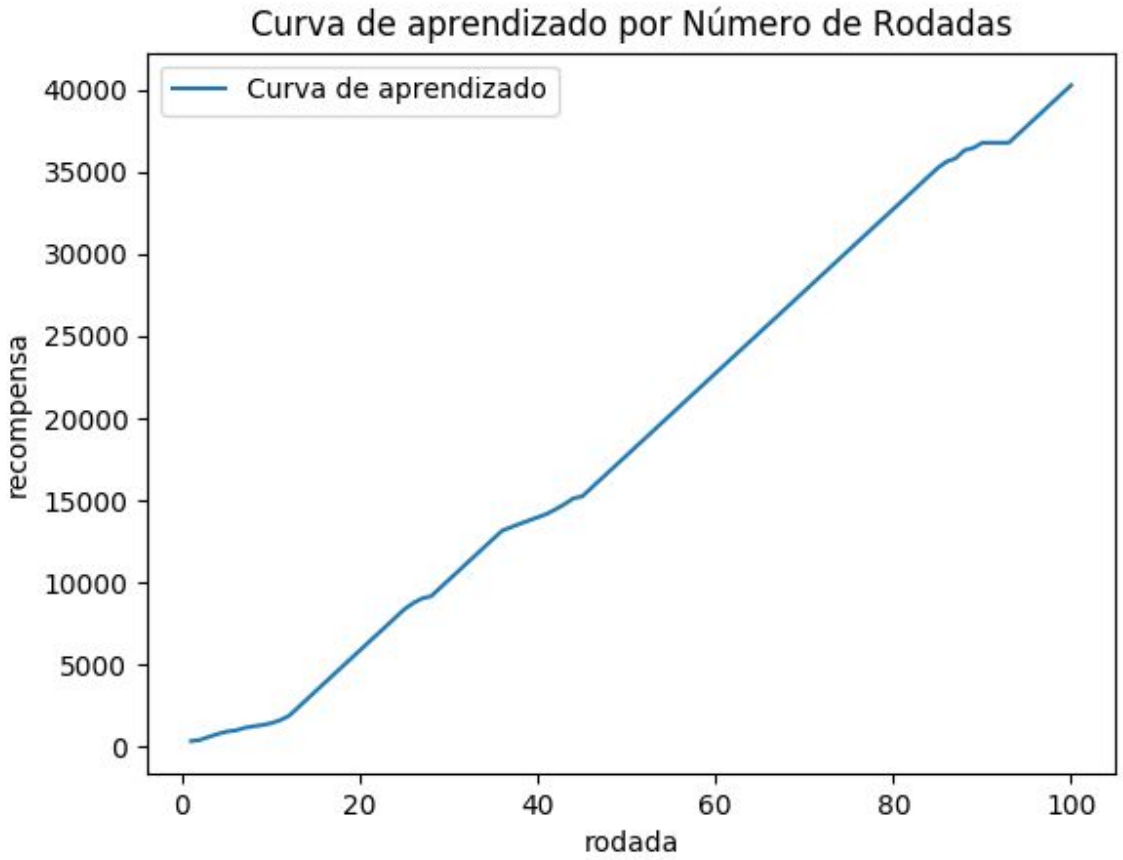
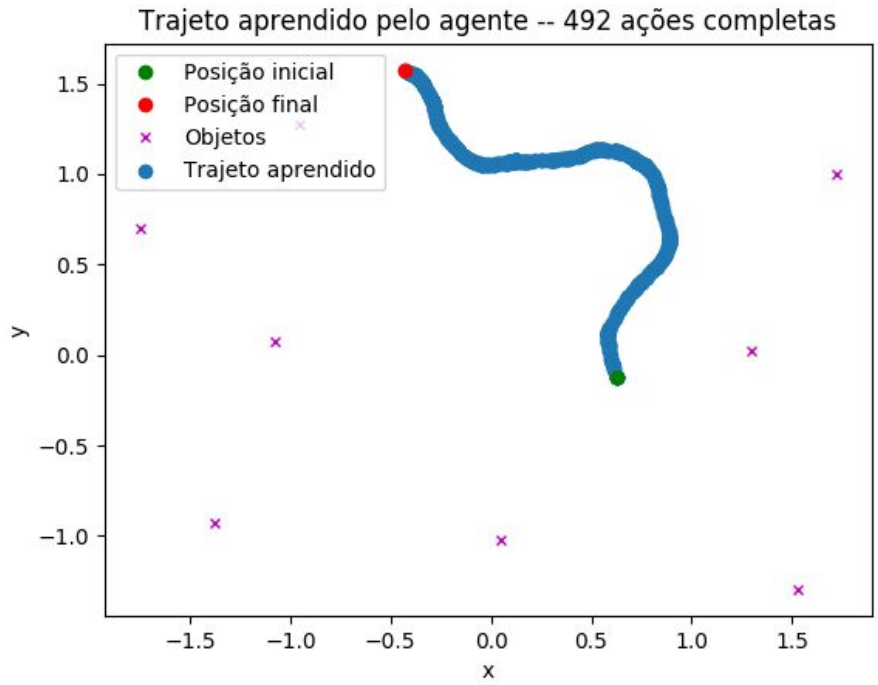
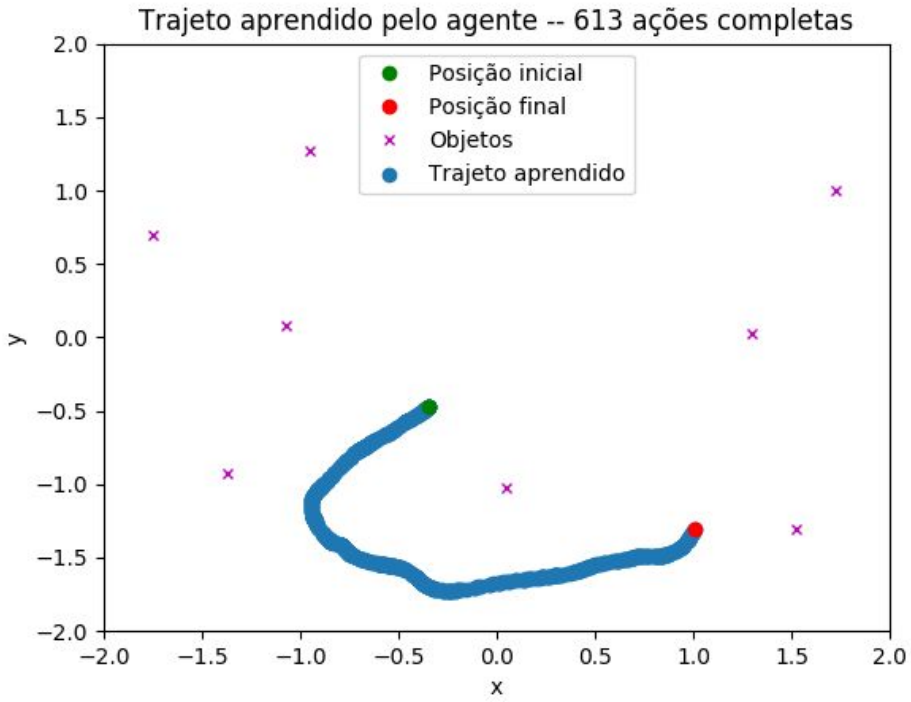


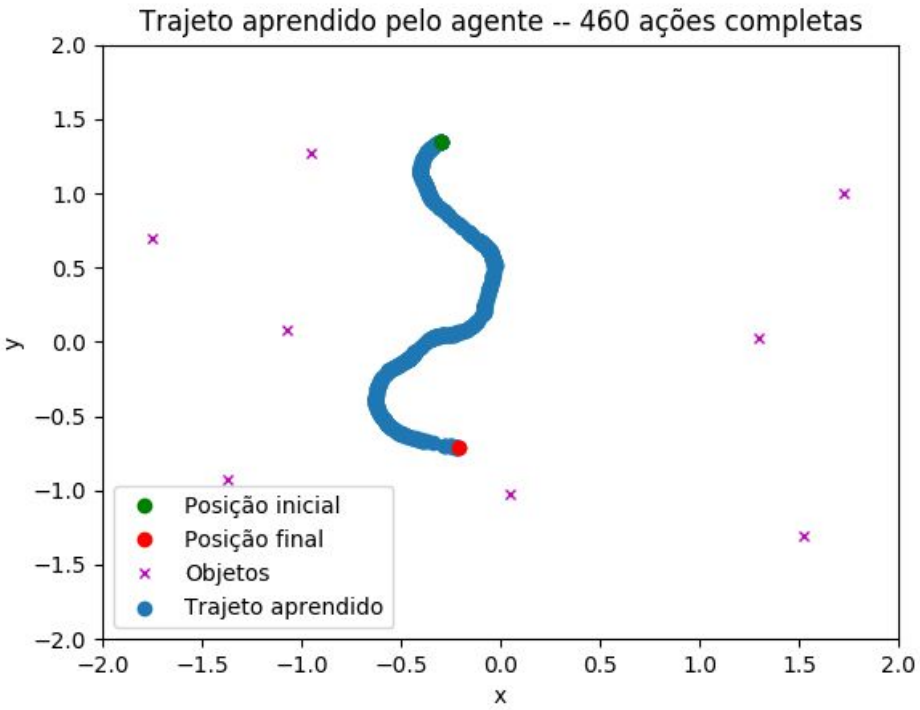
Figura (13) Curva de aprendizado no experimento 6.3



Figuras (14) Trajeto aprendido no experimento 6.3



Figuras (15) Trajeto aprendido no experimento 6.3



Figuras (16) Trajeto aprendido no experimento 6.3

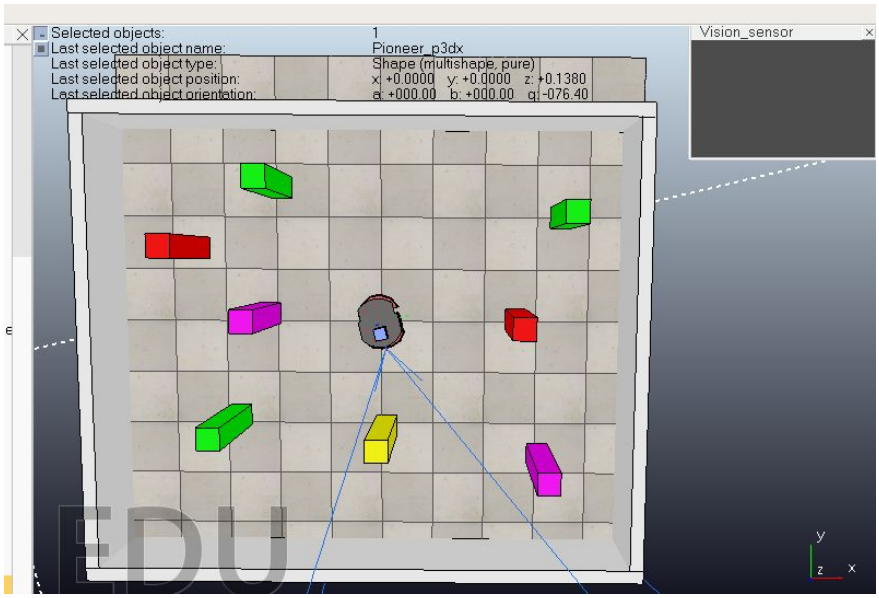


Figura (17) Cena utilizada pelos experimentos 6.2 e 6.3.

4. Conclusões

A partir dos experimentos é possível concluir que utilizar o mapa atencional, mais especificamente o de saliências, como entrada para um algoritmo de aprendizado de reforço produz resultados bastante positivos. Um dos pontos negativos é que, por exemplo, ao identificar pontos que não possuem saliências (como paredes), ele acaba indo diretamente de encontro a elas. Isto pode indicar que o mapa de saliência pode ser usado como fruto do aprendizado, mas podendo ser necessário realizar observações sobre os mapas de características originais para decisões mais complexas.

Comparando-se os experimentos 2 e 3 é possível notar que o conjunto de ações implica diretamente na qualidade do aprendizado, e que ter um conjunto que se relaciona perfeitamente com o objetivo em questão é fundamental.

Finalmente, concluímos que o trabalho de verificação, validação e utilização do modelo atencional utilizando o CST conforme proposto inicialmente foram realizados com sucesso. Todos os códigos implementados podem ser encontrados no github em [7] na branch “experiments”.

9. Trabalhos Futuros

Para melhorar a implementação do modelo atencional CONAIM no CST e ampliar suas possíveis aplicações, pode ser adicionado o curso TOP-DOWN do modelo, aumentando assim a gama de problemas que podem ser resolvidos, e tornando o modelo ainda mais real.

Mesmo sem implementar esta parte mais complexa, poderia ser feito um novo experimento com o objetivo de seguir o objeto mais rápido da cena, por exemplo. Seria necessário somente aumentar o peso da feature de direção no CombinedFeatureMap e mudar as recompensas para incentivar o agente a bater no vencedor.

Referências

[1] Raizer, K.; Paraense, A.L.O.; Gudwin, R. R. - A Cognitive Architecture with Incremental Levels of Machine Consciousness Inspired by Cognitive Neuroscience, International Journal of Machine Consciousness, Vol. 4, No. 2 (2012) 335-352, World Scientific Publishing Company,

[2] Castro, E.C.; Gudwin, R.R. - A Scene-Based Episodic Memory System for a Simulated Autonomous Creature, International Journal of Synthetic Emotions, 4(1), 32-64, January-June 2013, IGI Global.

[3] A.S. Simões. CONAIM: A Conscious Attention-Based Integrated Model for Human-Like Robots. Disponível em: http://ieeexplore.ieee.org/document/7383269/.

[4] Remote API de JAVA. Disponível em: http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsJava.htm

[5] CST - The Cognitive Systems Toolkit. Disponível em: http://cst.fee.unicamp.br.

[6] E.L. Colombini. An Attentional Model for Intelligent Robotics Agents. http://www.bdita.bibl.ita.br/tesesdigitais/lista_resumo.php?num_tese=66528

[7] GitHub Módulo Atencional do CST. Disponível em: https://github.com/leandrones/CST_AttMod_App

[8] GitHub CST - The Cognitive Systems Toolkit. Disponível em: https://github.com/CST-Group/cst

[9] SUTTON, R. S. Temporal Credit Assignment in Reinforcement Learning. University of Massachusetts, The MIT Press, 1984.

[10] Watkins, C.J.C.H. (1989). Learning from delayed rewards. PhD Thesis, University of Cambridge, England. Werbos, P.J. (1977).