Robotic Platform with independent vision and navigation systems

Tunai Porto Marques[1] Leonardo Nogueira Matos[1] Hendrik Macedo[1] Danilo Henrique[1]
tuna597@hotmail.com *lnmatos@ufs.br* *hendrik@ufs.br* *danilohfm@dcomp.ufs.br*

[1] *Universidade Federal de Sergipe - Departamento de computação, 49100-000 - São Cristóvão, SE*

The rescue of people or items of interest can be dangerous, since it is often done in hostile places. This paper proposes the creation of a robotic platform which is able to recognize specific objects and move independently, so that it can be used to rescue specific artifacts. The computer vision system of the platform cooperates with its navigation system, in order to make decisions regarding the environment and what the robot sees. In addition, the implementation of the vision system is made in parallel by multiple processors to improve its performance.

Keywords - Computer vision, Path Planning, Parallel Programming, Robot Rescue.

I. INTRODUÇÃO

Os principais objetivos da robótica são simular a locomoção, a percepção e a inteligência humana. Especificamente, de acordo com [1], um robô precisa ter: capacidade de locomoção (aparato para se movimentar no ambiente), manipulação, percepção própria (velocidade e posição) e do ambiente (visão computacional), além de inteligência. A plataforma robótica se torna uma ferramenta programável e de alta confiabilidade para uso genérico quando possui estas características. Buscar uma área ou aplicação que não possa ser (ou já tenha sido) automatizada por uma plataforma robótica vem se tornando cada vez mais difícil. Exemplos de áreas que utilizam a robótica extensivamente são: medicina, indústria automotiva, pesquisas espaciais, exploração petrolífera, entre outras. Em décadas passadas a robótica possuía um foco quase exclusivamente industrial. Hoje, a robótica busca resolver desafios do cotidiano na chamada human-centered and life-like robotics [1]. Esta nova geração de robôs deve oferecer suporte em serviços, entretenimento, educação, convivendo em segurança e independência com seres humanos.

Duas importantes características que o robô deve possuir para se tornar funcional são a capacidade de simular a visão humana e de se localizar e locomover. Desta forma, ele é capaz de se situar e se deslocar em um ambiente qualquer, além de ser capaz de reconhecer objetos de seu interesse.

Se os robôs são capazes de reconhecer padrões e interpretar seu significado, eles podem criar mapas de ambientes, reconhecer artefatos de interesse, tomar decisões e executar ações, exclusivamente baseados na visão computacional. Entretanto, os algoritmos utilizados na visão computacional devem lidar de forma eficiente com uma grande quantidade de dados a serem processados. Aplicações como [4] se propõem a criar uma máscara capaz de reconhecer padrões, tornando possível o casamento com objetos desejados. Estas máscaras utilizam inúmeras de imagens de entrada e consomem semanas de processamento [6] para serem criados eficientemente. Seja na construção de máscaras como em [4] ou mesmo na leitura dos dados de um dispositivo de captura, a quantidade de dados que devem ser processados é usualmente grande. Desta forma, políticas de processamento eficiente de dados se tornam atraentes

Os robôs precisam adotar um sistema de coordenadas para se locomoverem. Com ele, os robôs devem ser capazes de saber onde estão e para onde desejam ir. Sensores devem ser capazes de informar ao robô sua distância à quaisquer obstruções, para que eles possam determinar sua posição no espaço de coordenadas adotado. A técnica de localização MCL (Monte Carlo Localization) [15] foi a escolhida para ser utilizada neste trabalho.

A união dos sistemas de visão computacional e navegação conferem ao robô uma perspectiva inédita do ambiente[2], impossível de ser obtida usando tais funcionalidades independentemente. É importante que a interpretação dos dados obtidos por um sistema (visão) leve em consideração os dados do outro (localização).

A plataforma robótica desenvolvida neste projeto tomou como base os desafios propostos pela competição de robôs, Robocup - categoria de resgate [3]. Esta modalidade da competição possui um importante valor social, pois os robôs de resgate podem auxiliar ou mesmo substituir os agentes humanos que realizam a tarefa de resgatar pessoas ou objetos em ambientes que envolvem riscos. Como esta tarefa envolve um número grande de variáveis em um ambiente hostil, o objetivo deste trabalho é oferecer uma solução simplificada para dois problemas envolvidos nesta modalidade: a locomoção do robô e a capacidade de encontrar um objeto de interesse (a ser resgatado). Assim, o robô criado deve ser capaz de se locomover e localizar o objeto de interesse. Outras funcionalidades importantes, como a forma de resgate utilizada (braços mecânicos, plataformas de elevação) não fazem parte do escopo deste projeto.

II. REVISÃO DA LITERATURA RELACIONADA

A. Visão Computacional: OpenCV

O sistema responsável pela visão do robô recebe uma grande quantidade de dados que precisam ser processados (processamento de imagens) antes de serem interpretados (visão computacional). O OpenCV [5] é uma biblioteca livre que possui mais de 500 funções voltadas para o processamento de imagens. Além de funções de processamento de imagens, ela também oferece implementações de algoritmos e métodos relacionados a visão computacional[4][10].

B. Visão Computacional: Detector SURF

O detector SURF (Speed Up Robust Features), primeiramente apresentado em 2006 [9] e revisado em 2008 [10] por Herbert Bay, tem como objetivo fazer uma extração

rápida dos pontos de interesse de uma imagem. Com estes pontos, ele cria descritores que serão casados com uma imagem de entrada. Ao realizar o casamento dos descritores com uma imagem, ele identifica a presença da imagem geradora de tais pontos-chave. Em [10] é citado que o detector SURF possui boa repetibilidade, pois é capaz de reconhecer os mesmos descritores em diferentes condições de visualização: variações geométricas, fotográficas, ou mesmo ruídos. Quanto maior a dimensão do descritor, mais precisa porém custosa computacionalmente é a sua comparação com uma imagem. Assim, o compromisso do SURF é oferecer um detector que determine descritores de rápida computação, sem sacrificar sua taxa de acerto. A grande contribuição do SURF é o tempo reduzido que ele consome para encontrar os pontos de interesse de uma imagem.

Para tal, uma Matriz Hessiana é aplicada em pequenas frações da imagem. Para cada uma delas, o determinante da matriz é calculado. Em seguida, o método busca os determinantes de maior valor na imagem. Dado um ponto x = (x,y) em uma imagem I, a matriz Hessiana [17] $\mathcal{H}(x, \delta)$ deste ponto é dada por:

$$\mathcal{H}(x, \delta) = \begin{bmatrix} Lxx(x, \delta) & Lxy(x, \delta) \\ Lxy(x, \delta) & Lyy(x, \delta) \end{bmatrix} \quad (1)$$

Onde $Lxx(x, \delta)$ é a convolução da derivada Gaussiana de segunda ordem com a imagem I no ponto x, similarmente ao que acontece em $Lxy(x, \delta)$ e $Lyy(x, \delta)$. O determinante desta matriz pode ser obtido pelo produto dos autovalores. Entretanto, o SURF faz uma aproximação do cálculo do determinante de tal matriz utilizando a equação (2), onde w é um peso arbitrariamente atribuído (0,9), e Dxx, Dyy e Dxy são aproximações da derivada segunda Gaussiana:

$$\det(Happrox) = DxxDyy - (wDxy)^2 \quad (2)$$

Para acelerar o cálculo do determinante, Dxx, Dyy e Dxy são calculadas usando imagens integrais, o que economiza ainda mais tempo de processamento. A imagem integral de um ponto x=(x,y) é dada por:

$$I\Sigma = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j) \quad (3)$$

A imagem integral representa a soma dos valores de intensidade de todos os pontos na imagem com localização menor que (ou igual) a (x,y). Sua computação é rápida pois independente do seu tamanho são feitos apenas 4 acessos à memória e 3 operações.

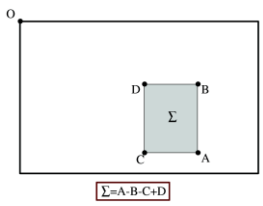

Figura 1. Cálculo da imagem integral em uma área retangular.

A imagem abaixo mostra os pontos-chave que foram extraídos usando o método "Fast-Hessian", descrito anteriormente:

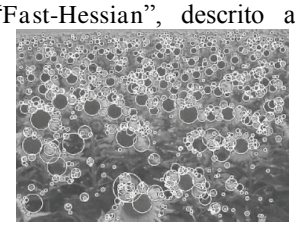

Figura 2. Pontos-chave extraídos pelo SURF de um campo de flores (retirado de [10]).

O detector do SURF é invariante a escala e a rotação. Para oferecer a primeira característica, ele faz uma estimativa da representação dos pontos-chave em diferentes escalas. Para ser invariante a rotação, ele identificada inicialmente a orientação dos pontos de interesse. Primeiro é calculada a resposta, nas direções x e y, do ponto de interesse e um pulso quadrado (wavelet de Haar) numa vizinhança de raio 6s, onde s é a escala na qual o ponto de interesse foi detectado. Com a resposta nas direções x e y, é determinada qual a inclinação horizontal e vertical dos pontos-chave. O pulso (filtro) usado para calcular as respostas da imagem nos eixos x e y é o seguinte:

Figura 3. Filtro para computar as respostas em x (esquerda) e y (direita) (retirado de [10]).

A diferença na extração de características usando o SURF e algoritmos como o de Viola e Jones [4] é que ao invés de construir classificadores baseados em uma grande base de dados, o SURF extrai pontos-chave de uma única imagem. Os classificadores criados em Viola e Jones possuem capacidade de distinção maior, pois utilizam várias imagens em sua construção. Por outro lado, os pontos-chave do SURF são extraídos quase que instantaneamente, em contraste com o longo tempo de processamento de Viola e Jones, visto que o SURF usa apenas uma imagem como base de dados. O OpenCV oferece um método chamado cvExtractSURF, que implementa o SURF.

C. Paralelização: OpenMP

As técnicas de visão computacional usadas no projeto fazem o processamento de um (ou de um grupo de) pixel(s), cujos resultados são independentes entre outros da mesma imagem. Além disso, experimentos usando o algoritmo de Viola e Jones [4] para uma base de dados de 5000 imagens positivas e 3500 imagens negativas [7] tomaram cerca de três dias para criar um classificador com 20 níveis na cascata de rejeição[4] (utilizando uma CPU Pentium 4). As técnicas de paralelização se mostraram uma interessante opção, pois como os resultados podiam ser obtidos paralelamente, havia um grande potencial de aceleração. Estudos com CUDA [11] (baseado na placa de vídeo) e OpenMP (baseado na CPU)

foram feitos para realizar a programação dos algoritmos de visão computacional em paralelo. Como nos experimentos foi usada a paralelização com o OpenMP, apenas ele será descrito.

O OpenMP [12] é uma API para programação multi-processo que utiliza memória compartilhada e é disponível para várias plataformas. A API oferece diretivas de compilador, rotinas de biblioteca e variáveis de ambiente que vão determinar como será feita a paralelização. Ela suporta nested parallelism (laços paralelos entrelaçados), além de oferecer suporte a número dinâmico de threads (o número de processos paralelos criados pode ser alterado em tempo de execução).

Com o OpenMP é possível criar uma thread (execução de um processo) principal que é executada enquanto a parte sequencial do código é visitada. A API divide a execução do código em um time de threads quando encontra uma região paralela, como mostra a figura 4:

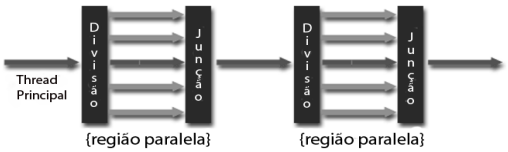

Figura 4. Divisão das threads em tempo de execução.

Dois tipos de paralelização são feitos: do/for, que divide o trabalho de um laço, distribuindo o cálculo das iterações entre os threads criados (data parallelism) e sections, que quebra o trabalho em partes discretas (ex: funções independentes). A paralelização em aplicações de visão computacional é feita dividindo laços que fazem cálculos repetitivos em uma região de pixels. Estes cálculos podem ser paralelizados utilizando um esquema do/for no OpenMP.

Para paralelizar um laço simples, por exemplo, basta inserir a seguinte diretiva de compilador fornecida pelo OpenMP antes do laço:

```
#pragma omp parallel
```

D. Navegação: Descrição do ambiente e planejamento do percurso

A navegação envolve a localização e o planejamento de rota do robô. Para introduzir as tecnologias utilizadas neste contexto, é indispensável descrever o ambiente usado nos testes. O mapa do ambiente de testes usado é ilustrado na figura 5. Foi escolhida uma representação computacional de grid [13] para modelá-lo. O mapa foi dividido em pequenas unidades – células - e então representado por um grafo. Os vértices e arestas do grafo correspondente foram construídos como segue: primeiro, o ambiente é aproximado por um grid de células de mesmo tamanho (20 cm). Há duas categorias de células: ocupadas e livres. Cada célula livre é representada por um vértice no grafo. Em seguida, um tipo de vizinhança é atribuído a cada célula, de forma que todas elas possuem arestas que as liguem com células vizinhas livres. O tipo de vizinhança adotado é apresentado a seguir:

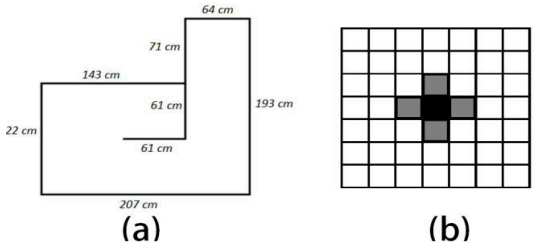

Figura 5. (a) Dimensões do mapa de teste (b) A vizinhança de Von Neumann (células cinzas compõem a vizinhança),

O planejamento do percurso do robô é feito analisando o grafo criado. Considerando um ponto de largada e um ponto de chegada como dois vértices do grafo, o algoritmo de busca em largura (BFS) [14] foi utilizado para determinar o caminho a ser percorrido. Considerando um grafo G = (V,E), onde V representa os vértices, E representa as arestas do grafo e são dados dois vértices s e f, representando vértices de largada e chegada, o algoritmo de busca em largura observa os vértices que s pode alcançar, e então determina o melhor caminho de s até f (se este existir) [14]. A aproximação do mapa usando grafos é mostrada abaixo:

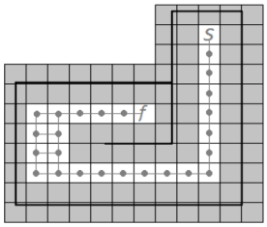

Figura 6. Aproximação em grid do mapa do ambiente.

E. Navegação: Localização

O problema da localização pode ser modelado probabilisticamente como um Filtro de Bayer.

Matematicamente, o que estamos tentando prever é a posição $x = [x, y, \theta]^t$ em um tempo k:

$$p(x_k|z^k) \quad (4)$$

onde $z^k = \{z_k, i = 1 \dots k\}$ e z_k e x_k representam, respectivamente, a posição e o tempo do robô num tempo k. Esta função de densidade de probabilidade pode ser calculada recursivamente em duas fases pelas medidas, z_k, e ações u_k, ao longo do tempo. A primeira fase é chamada fase de predição onde a ação feita pelo robô é levada em consideração. Por conta disso, ela também é conhecida como modelo de movimento. Nesta fase, (4) é derivada em:

$$p(x_k|z^{k-1}) = \int p(x_k|x_{k-1}, u_{k-1}) p(x_k|z^{k-1}) dx_{k-1} \quad (5)$$

A segunda fase é chamada de fase de atualização, onde as medições feitas são levadas em consideração derivando (5) em (6). Também conhecida como modelo de medição:

$$p(x_k|z^k) = \frac{p(z_k|x_k)p(x_k|z^{k-1})}{p(z_k|z^{k-1})} \quad (6)$$

Usando $\alpha = p(z_k|z^{k-1})$ como um fator de normalização, obtemos:

$$p(x_k|z^k) = \alpha p(z_k|x_k)p(x_k|z^{k-1}) \quad (7)$$

Daí em diante, só é necessário definir P(xo). Para implementação do Filtro de Bayer, foi usado o algoritmo de localização Monte Carlo (MCL) [15]. Esse algoritmo discretiza a função de probabilidade em um conjunto contendo M partículas, $X_t = \{x_t^{[1]}, x_t^{[2]}, x_t^{[3]}, \dots x_t^{[m]}\}$, onde cada uma delas tem um peso de importância. Considere o peso de importância da enésima partícula, num tempo t, como sendo $w_t^{[i]}$. Assim, nós temos que a soma das importâncias é dada por:

$$\sum_{i=1}^{M} (\alpha w_t^{[i]}) = 1 \quad (8)$$

No início do algoritmo todas as partículas estão na posição de início corretas. Ao mesmo tempo, os pesos de importância delas são $\alpha w_t^{[i]} = M^{-1}$, que define P(xo).

As partículas se movem de acordo com o modelo de movimento e seus pesos são atualizados de acordo com o modelo de medição. No final da fase de atualização, um processo chave chamado '*importance resample'* é executado para selecionar as partículas cuja probabilidade de representação correta da posição corrente do robô é maior. O desempenho do algoritmo é diretamente influenciado pelo número de partículas que vão ser usadas para tentar determinar a posição do robô. No estudo de caso utilizado como referência (e também nos experimentos) foram usadas 1500 partículas.

III. RESULTADOS E DISCUSSÃO

O robô utilizado nos experimentos foi o Mindstorms NXT, da empresa Lego. Ele possui sensores de luz, som e ultrassom e um bloco programável para controlá-los. Nos experimentos, dois servomotores envoltos em rodas de plástico foram acoplados no robô. O sensor ultrassônico que o NXT possui envia sinais sonoros com um frequência de aproximadamente 40 KHz que são recebidos, após colidirem com obstáculos, por um microfone. O sensor tem uma área de alcance de 0 a 255 cm com precisão de aproximadamente 3 cm. Para programar o robô foi utilizado o firmware fornecido pela Lego, LeJOS NXT 0.85, que é baseado em Java. O Lego NXT é mostrado na figura 7.

Figura 7. Robô Lego NXT com servomotores e sensores.

Para criar o sistema de visão do robô foi usado inicialmente o algoritmo de Viola e Jones. Entretanto, ele tomou dias de processamento e não apresentou, como resultado, um bom filtro de classificação. Assim, a técnica de reconhecimento que passou a ser usada foi exclusivamente o SURF, que ofereceu melhores resultados em um tempo extremamente reduzido (o SURF usa apenas uma imagem para extração das características). Para criar um filtro classificador usando as características obtidas com o SURF, capaz de reconhecer o objeto de interesse tendo como base uma imagem de 324x223 pixels, foram gastos apenas 300 ms.

A biblioteca OpenCV possui um exemplo em que o SURF é usado para extrair os pontos-chave de duas imagens, que em seguida são comparados para que se saiba se uma ocorre na outra (se o "objeto de interesse" está presente na outra imagem). Uma modificação foi feita nesta implementação, de forma que ao invés de extrair as características de apenas um par de imagens, o programa pudesse extrair de todas as imagens (frames) que recebesse com o passar do tempo. Este incremento no código original tornou possível usar o SURF para verificar a existência de um objeto (da foto dela, na verdade) em um vídeo.

O tempo que nos interessa medir e tentar diminuir é o consumido para a extração das características de uma determinada imagem, pois esta é a ação de maior custo computacional do SURF e possibilita o reconhecimento do objeto de interesse no ambiente. As características do objeto de interesse são extraídas uma vez, e em seguida são comparadas com aquelas de cada frame (imagem) recebido do robô, para determinar se o objeto de interesse está ou não no campo de visão deste. Diversos testes para determinar este tempo foram feitos utilizando uma plataforma com o processador Intel Core i5 M450 de 2,4 Ghz, uma imagem de 324x223 pixels e execução sequencial. Aproximadamente 290 ms eram consumidos para realizar a extração dos pontos-chave de cada imagem. Executando o SURF paralelamente (usando o OpenMP), a média de tempo para extração das características desta mesma imagem foi reduzida em aproximadamente 20% (caiu para 240 ms). Esta redução no tempo possui grande importância, uma vez que as características serão extraídas a cada quadro novo que é recebido. Este processo é o mais custoso do sistema robótico, pois ele é repetido várias vezes a cada segundo. Desta forma, a capacidade de executá-lo em paralelo caracteriza um grande êxito. A tabela 1 apresenta os ganhos, em termos de tempo, observados quando foi utilizado o OpenMP para execução do SURF.

TABELA 1. GANHOS COM O USO DO OPENMP

N° de imagens	**Tempo de extração (execução sequncial)**	**Tempo de extração (execução paralela)**	**Redução do tempo (%)**
1	290,128ms	242,330ms	19,72%
10	287,098ms	220,461ms	30,26%
50	284,819ms	202,204ms	40,85%
100	284,696ms	219,079	29,95%

Como os testes foram realizados com a mesma imagem e cada frame do vídeo é composto de imagens distintas (não há imagens iguais), o ganho mais importante é aquele para uma

imagem (20%), pois cada frame caracteriza uma imagem individual.

Os passos paralelizados e que compõem a execução do sistema de visão computacional são apresentados no algoritmo que segue:

(1) Extrair as características (pontos-chave) da imagem que contém o objeto de interesse (uma vez) usando o SURF.
(2) Extrair as características (pontos-chave) do frame corrente (recebido pela câmera acoplada ao robô) usando o SURF.
(3) Realizar o casamento das características do objeto com as características do frame, afim de saber se o objeto de interesse encontra-se no frame corrente.

O sistema de visão computacional do robô deve ser capaz de reconhecer as imagens apresentadas na figura 8. Na primeira, que representa uma seta para a esquerda (figura 8 à esquerda) foram detectados 273 pontos-chave pelo SURF. Na segunda, representando uma seta para a direita (figura 8 à direita), foram detectados 671 pontos-chave pelo SURF. As duas imagens possuem 436x337 pixels.

Figura 8. Imagens utilizadas como objetos de interesse nos experimentos com a plataforma robótica.

O processo envolvido na detecção de um objeto, como apresentado no anteriormente, é ilustrado na figura 9.

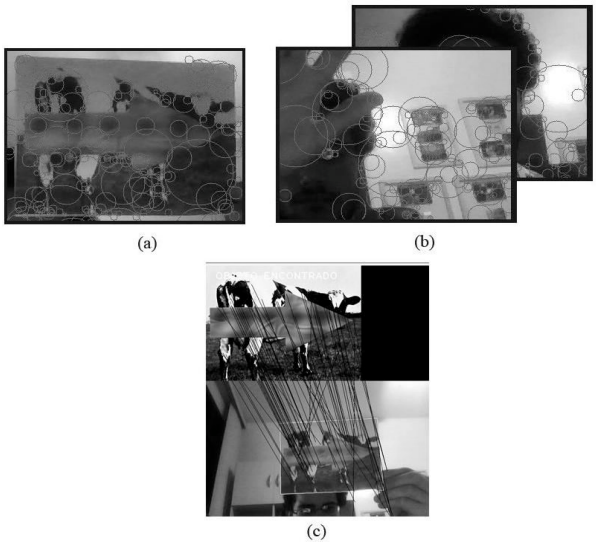

Figura 9. (a) Extração das características do objeto de interesse. (b) Extração das características de cada frame. (c) Casamento das características

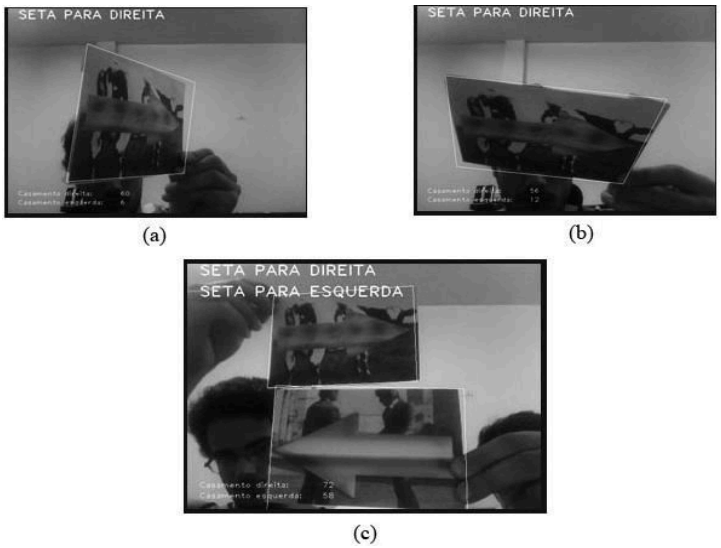

Figura 10. (a) e (b) Reconhecimento com variação de escala e orientação. (c) Reconhecimento de dois objetos de interesse simultaneamente.

A captura do vídeo no robô foi feita usando uma câmera de vídeo sem fio, de forma que a mobilidade dele não fosse afetada. A câmera envia dados numa determinada frequência para um receptor de frequência ajustável, que por sua vez se comunica com o computador por USB. Assim, o robô precisa apenas adquirir e enviar os dados para a central de processamento e esperar por instruções sobre as ações que deve tomar em seguida. As instruções sobre próximos movimentos são enviadas pelo computador para o robô por Bluetooth.

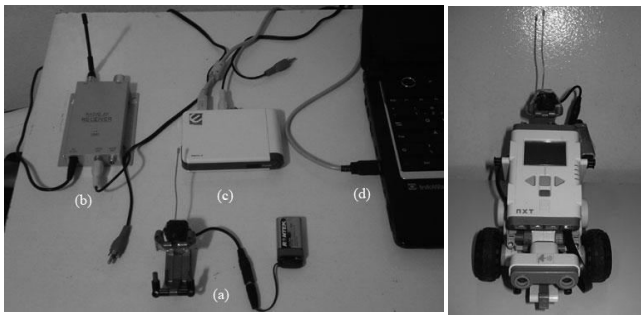

Figura 11. Esquerda: (a) Câmera de vídeo sem fio. (b) Receptor de frequência ajustável. (c) Placa de captura. (d) Comunicação USB com o computador. Direita: Plataforma robótica com câmera sem fio acoplada.

O ambiente usado para realização dos testes foi o mesmo descrito na seção de navegação. Os primeiros testes feitos usando o labirinto consistiram apenas em observar o comportamento do robô na travessia do labirinto inteiro.

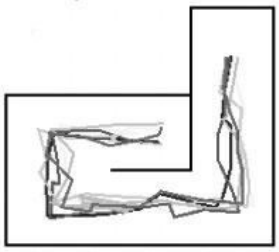

Figura 12. Caminhos percorridos pelo robô NXT na travessia do labirinto.

Em seguida foram fixados objetos de interesse (impressões das imagens da figura 8) nos pontos A e B do labirinto, como representado na figura 18.

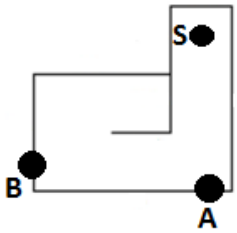

Figura 13. Pontos (A e B) onde as fotografias dos objetos de interesse foram posicionadas e ponto (S) representando a posição inicial do robô.

Os testes subsequentes se basearam na seguinte rotina: o robô é colocado em uma posição inicial (ponto S da figura 13) e realiza os cálculos para determinar sua posição. Em seguida, ele realiza a correção de sua posição e então, antes de executar o próximo movimento, verifica se o objeto de interesse está presente no seu campo de visão. Se estiver presente, ele emite um som (beep) e gira em torno de seu eixo indefinidamente.

Inicialmente os dois objetos estavam visíveis para o robô, de forma que ele devia encontrar primeiramente o objeto no ponto A e então parar, sem chegar a avistar o objeto no ponto B. Para esta configuração foram feitos 10 testes, nos quais: em 1 o robô não detectou o objeto em A, mas o objeto em B; em 1 ele não detectou nenhum dos objetos; em 1 ele detectou erroneamente um objeto de interesse que não existia; e em 7 ele avistou o objeto em A (condição de sucesso). Ou seja, para esta configuração o robô teve uma taxa de sucesso de 70%. Em seguida, o objeto no ponto A foi retirado, de forma que o robô devia percorrer uma maior distância no labirinto para achar o objeto no ponto B. Dos 10 testes feitos com esta configuração: em 2 o robô avistou objetos de interesse que não existiam; em 2 ele não detectou nenhum objeto de interesse; e em 6 ele obteve sucesso, detectando o objeto no ponto B. Desta forma, nesta configuração o robô teve uma taxa de acerto de 60%.

O número de frames em que o objeto de interesse deve ser detectado para que o robô o leve em consideração foi configurado para 2. Ou seja, uma detecção instantânea (1 frame), que muitas vezes não representa o objeto de interesse, passou a ser desconsiderada. Como o resgate propriamente dito não é objetivo deste projeto, os testes se limitaram em validar a fusão dos sistemas de visão (com execução em paralelo) e navegação.

IV. Conclusão

A plataforma robótica desenvolvida neste projeto foi capaz de se localizar e identificar a presença de um objeto de interesse no seu campo de visão de forma independente, além de executar o sistema de visão paralelamente. Ela se mostrou eficaz na navegação e no reconhecimento, atividades essenciais na modalidade de resgate. Trabalhos futuros se concentrarão no refino do uso do OpenMP, utilização de outras câmeras pelo robô (ex: câmera de celulares com Android) e na criação de sistemas de resgate propriamente ditos, para que o robô, além de navegar e localizar os objetos, possa movê-los.

V. Referências bibliográficas

[1] Siciliano, B.; Oussama, K. (Eds.). Handbook of Robotics. Springer-Verlag Berlin Heidelberg. 2008.

[2] NASA. Machine Vision for Robotics. 2008.Disponível em : www.nasa.gov/centers/ames/research/technology-onepagers/machine-vision.html

[3] Ruiz-del-Solar, J.; Chown, E.; Ploeger; Paul, G. (Eds.). RoboCup 2010: Robot Soccer World Cup XIV. 2011. Springer-Verlag.

[4] Viola, P.; Jones, M. Rapid Object Detection using a Boosted Cascade of Simple Features.2001. Em Conference on Computer Vision and Pattern Recognition

[5] Bradski, G.; Kaehler A. Learning OpenCV. 2008. O'Reilly Media.

[6] Seo, N.; Rapid Object Detection With a Cascade of Boosted Classifiers Based on Haar-like Features . 2008. Disponível em: www.note.sonots.com/SciSoftware/haartraining.html

[7] Kuranov, A.; Lienhart, R.; Pisarevsky, V. An Empirical Analysis of Boosting Algorithms for Rapid Objects With an Extended Set of Haar-like Features. 2002. Em Intel Technical Report.

[8] Schapire R. Theoretical Machine Learning. March 2008.

[9] Bay H. ; Tuytelaars T.; Gool L. V. SURF: Speed Up Robust Features. 2006. Em European Conference on Computer Vision, Graz, Austria.

[10] Bay H. ; Ess A. Tuytelaars T.; Gool L. V. Speed Up Robust Features (SURF) . 2008. Em Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346--359

[11] Sanders, J.; Kandrot, E. Cuda by example: An introduction to General-Purpose GPU programming.Addison-Wesley.2010.

[12] Chapman B.; Jost G.; Pas v. v. R. Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press. 2007.

[13] Crous, C. B., Autonomous robot path Planning (Master's Thesis). 2009.

[14] Cormen, T. H., C.E. Leirson, R. L. e Stein C., Introduction to Algorithms. MIT Press. 2001.

[15] Thrun, S., Fox D., Burgard, F. Robust monte Carlo Localization for mobile robots. Artifficial Inteligence. 2000.

[16] Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Em Journal of Systems and Software, 2005.

[17] Massago, S. Matriz Hessiana e Aplicações. Dezembro de 2010.