Atividade de Laboratório 9

Objetivos

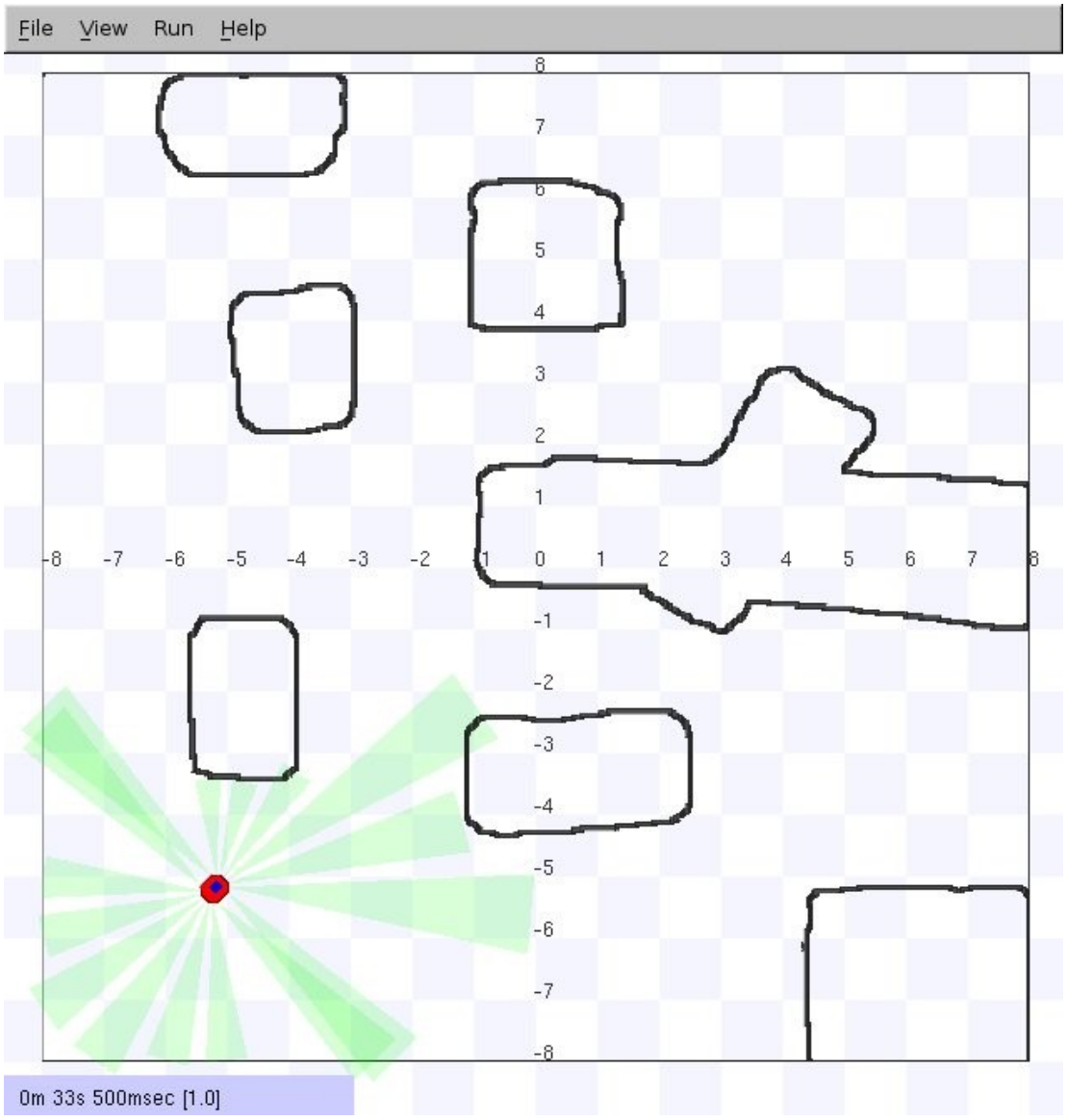
O objetivo desta atividade de laboratório é exercitar os conceitos de programação em linguagem de montagem do ARM, desenvolvendo um código de controle para o Uóli, um robô simulado.

Descrição

Uóli é um robô famoso que foi criado para nos ajudar a limpar o planeta. Infelizmente, em uma de suas aventuras, Uóli sofreu avarias sérias ao tentar impedir que um holodetector se fechasse e parou de funcionar.

Após o incidente, sua amiga Íva reconstruiu Uóli a partir de peças usadas de outros robôs, entretanto, seu programa foi apagado durante a reconstrução e, atualmente, apenas o *hardware* de Uóli foi recuperado. Com o intuito de ajudar Íva, construíremos parte do *software* que controlará Uóli.

Neste laboratório vamos utilizar uma versão modificada do simulador ARM das aulas anteriores que trabalha em conjunto com o *Player* (http://playerstage.sourceforge.net/index.php?src=player). *Player* é um simulador que modela a interação de um robô com o ambiente ao seu redor. O simulador permite que um código de usuário possa ser desenvolvido para controlar os periféricos do robô. Para habilitar a comunicação com o seu programa, o *Player* fornece uma interface via *sockets*.



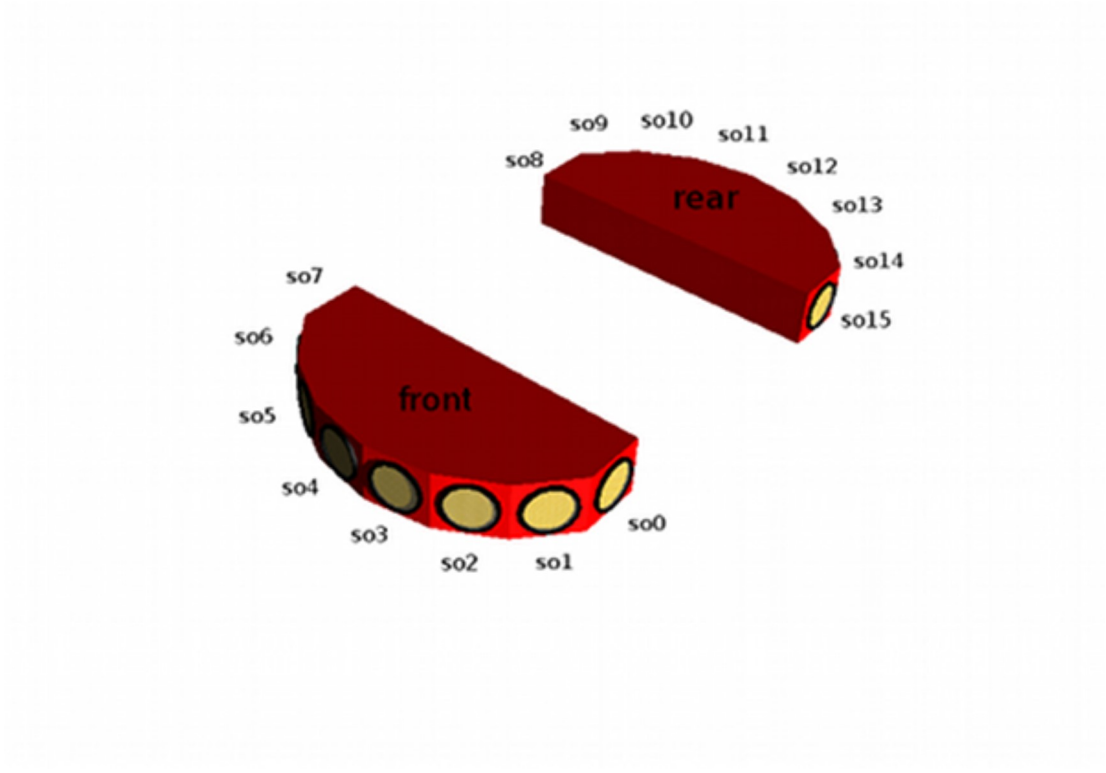
Para este laboratório, você pode supor que toda a parte de configuração do sistema, incluindo os periféricos, já foi

realizada pelo código básico do sistema e cabe a você apenas desenvolver um código em linguagem de montagem ARM para controlar o Uóli.

Informações e Detalhes

No nosso exemplo, o robô Uóli apresenta 3 periféricos: Dois motores e um conjunto de sonares. Você vai precisar definir uma velocidade para cada motor. Para verificar a proximidade de objetos/obstáculos, você deve ler alguns dos sonares, que são 16 ao todo. Um breve detalhamento sobre este robô pode ser encontrado no seguinte arquivo (anexos/Pioneer3DX-P3DX-RevA.pdf).

Na figura abaixo você pode ver como os sonares estão dispostos no robô. Por exemplo, se você quiser obter a distância de um objeto à frente do robô, você vai consultar os sonares 3 e 4. Para realizar a leitura de um sonar, existe a chamada de sistema (*syscall*) `read_sonar`, identificada pelo número **125**. Essa *syscall* recebe em `r0` o valor que identifica qual sonar você deseja ler e após a sua chamada o resultado será retornado também em `r0`. Somente os 4 *bits* menos significativos de `r0` serão consultados para indexar o sonar e o resultado vai ocupar os 12 *bits* menos significativos em `r0`.



O robô possui dois motores, um para cada roda lateral, que aqui vamos chamá-los de *motor0* e *motor1*. O *motor0* encontra-se localizado entre o sonar 7 e o sonar 8, enquanto o *motor1* está entre o sonar 0 e o sonar 15. Uma *syscall* de número **124** foi criada para controlar os motores, chamada **write_motors**. Para especificar as velocidades, você deve carregar em `r0` a velocidade do *motor0* e em `r1` a do *motor1*. Para configurar as velocidades, apenas os 6 *bits* menos significativos são considerados.

Neste laboratório você deverá implementar um código de usuário que ajusta a direção do robô para evitar colisões. Você pode ler os sonares frontais (3 e 4) do robô para detectar obstáculos e controlar seus motores para redirecionar o robô de modo que ele evite os obstáculos. Seu robô deve seguir uma trajetória retilínea após desviar dos obstáculos.

Exemplo de código

O código de exemplo (motors.s (anexos/motors.s)) implementa uma lógica simples de controle. Ele lê as distâncias retornadas pelos sonares 3 e 4 (frontais), se elas forem menores que um certo limiar, ele para o robô, senão, coloca-o pra andar para frente.

Informações importantes/Dicas

- Se você definiu uma velocidade para o robô, durante a leitura dos sonares e processamento da informação, o robô vai continuar em movimento.
- Caso os sensores do robô não estejam aparecendo no simulador gráfico, troque o gerenciador de janelas para o Cinnamon.

Outras Dicas

- Assista ao filme "WALL-E". É muito legal!

Simulação

Configuração de variáveis de ambiente:

```
source /home/specg12-1/mc404/simulador/set_path_player.sh
```

Montagem:

```
arm-eabi-as motors.s -o motors.o
```

Ligação:

```
arm-eabi-ld motors.o -o motors  -Ttext=0x77802000
```

Geração da imagem do cartão SD:

```
mksd.sh --so /home/specg12-1/mc404/simulador/simulador_player/bin/knrl --user motors
```

Antes de iniciar o simulador, você precisa abrir uma sessão do Player *EM OUTRO TERMINAL*:

```
player /home/specg12-1/mc404/simulador/simulador_player/worlds_mc404/simple.cfg
```

Simular:

```
armsim_player --rom=/home/specg12-1/mc404/simulador/simulador_player/bin/dumboot.bin --sd=disk.img
```

Requisitos de entrega

Endereço da atividade no sistema SuSy: https://susy.ic.unicamp.br:9999/mc404abef/09ab (https://susy.ic.unicamp.br:9999/mc404abef/09ab) ou https://susy.ic.unicamp.br:9999/mc404abef/09ef (https://susy.ic.unicamp.br:9999/mc404abef/09ef).

- Apenas deve ser entregue um arquivo denominado raXXXXXX.s, com seu código em linguagem de montagem (substitua XXXXXX por seu RA de 6 dígitos).