

Mecatrónica – 2013/2014

ROS - Detector de metais

Projecto

Luís Rocha

2010127532

José Medeiros

2010129934

Introdução

Neste trabalho foi incorporado um sensor inductivo ao robô Roomba para a detecção de minas terrestres através do uso de ROS. Para o controlo do robô foi necessário fazer uso de um API[6] que permitisse a comunicação com o mesmo e a plataforma ROS. Com este foi possível controlar os movimentos do robô através do teclado usando um terminal.

Quando na presença de metais e usando os dados devolvidos pelo sensor - proximidade e frequência - é possível determinar se o robô se encontra perante uma mina. Com base neste princípio é possível detectar a presença de minas.



(a) Robot Roomba



Bobine LDC1000 Interface USB - μ MSP430

(b) Componentes do LDC100EVM

Figura 1: Roomba e LDC100EVM com os componentes

Indíce

1	Lista de acrónimos	4
2	Material disponível	4
3	Descrição do problema	5
4	Conhecimentos necessários	5
5	Solução proposta	6
5.1	Hardware	6
5.1.1	Introdução ao sensor	7
5.1.2	Arquitectura	8
5.1.3	Implementação	9
5.2	Software	11
5.2.1	Como executar	11
5.2.2	Estrutura do Código	12
5.2.3	Utilização do ROS	12
5.2.4	Comunicação com o sensor	13
5.2.5	Protocolo VNC	13
6	Resultados	14
7	Conclusões	15
8	Anexos	16
8.1	Código C++	16
8.2	Código Python	20
8.3	Código Matlab	21

Indice de figuras

1	Roomba e LDC100EVM com os componentes	1
2	Placa breadboard + Sensor	6
3	Estrutura do robô Roomba com o sensor LDC100EVM	6
4	Sensor LDC1000EVM	7
5	Esquemático do sensor	7
6	Imagen real do robô	8
7	LCD1000EVM com indicação das perfurações	9
8	LCR Meter 186	10
9	Registros do sensor LDC1000EVM	10
10	Interligação dos diferentes programas	12
11	Protocolo VNC	14
12	Teste para detecção de minas num labirinto	14

1 Lista de acrónimos

ROS

Robot Operation System - Sistema Operativo para Robos

LRC Meter

Aparelho que mede indutância, capacidade e resistência de um componente[3]

PCB

Printed Circuit Board - Placa de Circuito Impresso

API

Application Programming Interface - Interface de Programação de Aplicações

USB

Universal Serial Bus

VNC

Virtual Network Computing

2 Material disponível

Computadores

Notebook

Netbook

Componentes/Aparelhos Electrónicos

Bobina de 1.71 mH

Condensador de 3.09 nF

LRC Meter - Modelo 186

Robô Detector de Minas

Robô Roomba 555

Detector de Metais - LDC1000EVM

API/Wrapper para comunicação com o Roomba

Acessórios

Teclado Wireless

USB Hub com quatro portas

3 Descrição do problema

Pretende-se detectar minas terrestres através do uso de um robô Roomba e de um sensor indutivo LCD1000EVM capaz de medir a proximidade e a frequência associada á presença de um metal nas imediações. Este sensor é composto por uma bobine em PCB, um LDC1000 que faz uso da bobine e um microcontrolador - MSP430, tal como se pode observar na figura 1b.

Para ser possível a concretização deste projecto é necessário implementar uma interface de comunicação entre o computador e o sensor para a aquisição de dados. Esta interface vai ser baseada num protocolo de comunicação que terá que ser o mesmo que é usado pelo microcontrolador. Será portanto necessário entender bem o protocolo visto que toda a base deste projecto se baseia neste sensor.

Após a interligação entre o sensor e o computador através de uma porta USB é necessário combinar o robô Roomba com o sensor utilizando a plataforma ROS¹. Para isso é obrigatório fazer uso de transformadas ($tf's$) para referenciar o sensor ao eixo/referencial em relação ao referencial principal do robô. Mais á frente no relatório explicaremos melhor o conceito e a maneira como interligamos as transformações na nossa implementação. Este passo é importantíssimo para que se possa calcular as posições onde o sensor detecta minas.

4 Conhecimentos necessários

Após analisarmos os requisitos deste projecto chegámos á conclusão de que era necessário familiarizarmos com a plataforma ROS. Sendo assim seguimos os tutoriais existentes no website WikiRos[1]. Após concluidos os tutoriais, ficamos aptos a usar ROS usando C++ como linguagem de programação.

Os seguintes conceitos/utilidades foram usados no nosso projecto, a sua explicação breve apresenta-se abaixo:

Tf

Estruturação dos referenciais do robô assim como as transformações necessárias entre os diferentes componentes² do robô para que se possa saber as posições de qualquer elemento num dado instante.

Marker

Estrutura de dados que permite usar as bibliotecas do *rviz* para marcar minas no mapa. A posição das minas é obtida pela transformação da *tf* do sensor em relação ao referencial base do robô.

ArrayMarker

Guardamos os markers (minas) que detectamos num array de markers. Para tal é necessário fazer *push_back* de um marker para o array quando é detectada a presença de uma mina.

Serial Port

Protocolo de comunicação usado para comunicar com o sensor através de uma porta USB.

¹Distribuição Hydro

²Sensores, plataformas, rodas, entre outros

5 Solução proposta

5.1 Hardware

Como podemos ver na figura 2, foi colocado numa breadboard o condensador e foram feitas as ligações necessárias de modo a facilitar a mudança de bobinas e para garantir que o circuito está em contacto com os componentes. Foi ainda soldado ao sensor dois fifos conectores para tornar mais fácil o seu uso.

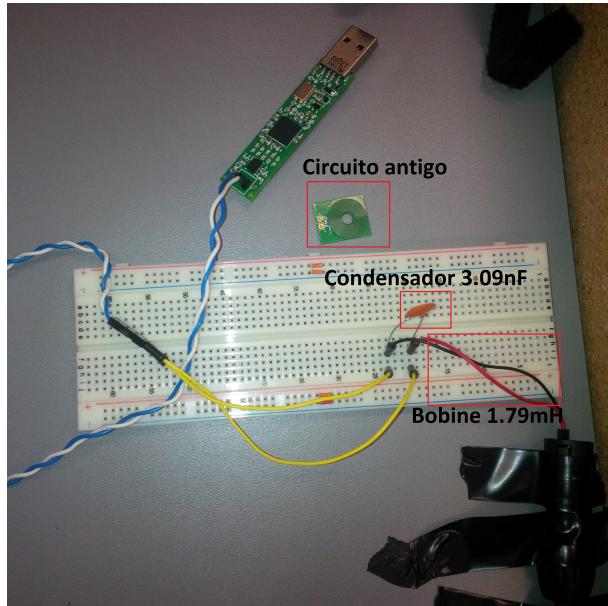


Figura 2: Placa breadboard + Sensor

Em termos de ligações é mais facil de compreender na figura 3, onde é possível ver a ligação dos componentes ao Roomba e ao sensor. O computador vai comunicar com o Roomba e receber dados do sensor. Através da plataforma ROS é feito o controlo do Roomba por teclado e feita a interligação dos dados enviados pelo sensor e o robô.

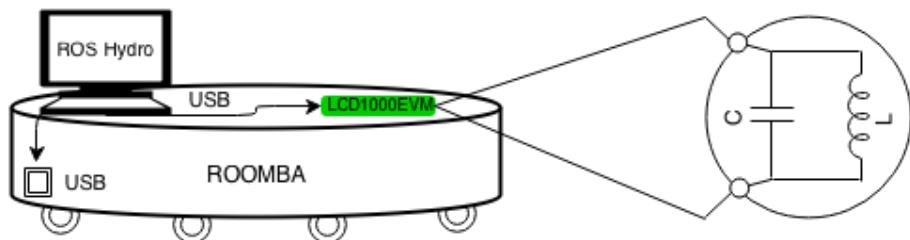


Figura 3: Estrutura do robô Roomba com o sensor LDC100EVM

5.1.1 Introdução ao sensor

O LDC100EVM é um sensor indutivo que permite medir a presença de metais retornando duas variáveis: posição do metal e a frequência de oscilação. É um circuito integrado que contém um microcontrolador MSP430 que é usado para fazer a interface entre o LDC (sensor) e um computador. Este integrado vêm com uma bobine que pode ser mudada se se partir a PCB na zona a tracejado e colocando ai uma bobine á nossa escolha. É possivel ver na figura 4 a zona a tracejado:



Figura 4: Sensor LDC100EVM

Para dimensionar a bobine é necessário satisfazer duas condições para que o sensor possa funcionar na gama ideal:

1. A frequência de ressonância deve estar compreendida entre 5 kHz e 5 MHz
2. A resistência de ressonância³ deve estar compreendida entre 798Ω e $3.93\text{ M}\Omega$

Para o cálculo da frequência e a respectiva resistência usa-se as seguintes fórmulas:

$$f_{\text{ressonância}} = \frac{1}{2\pi\sqrt{LC}} \quad (1)$$

$$R_p = \frac{1}{R_s} \frac{L}{C} \quad (2)$$

Abaixo encontra-se representado o esquemático do circuito que dá origem ás equações (1) e (2) acima descritas:

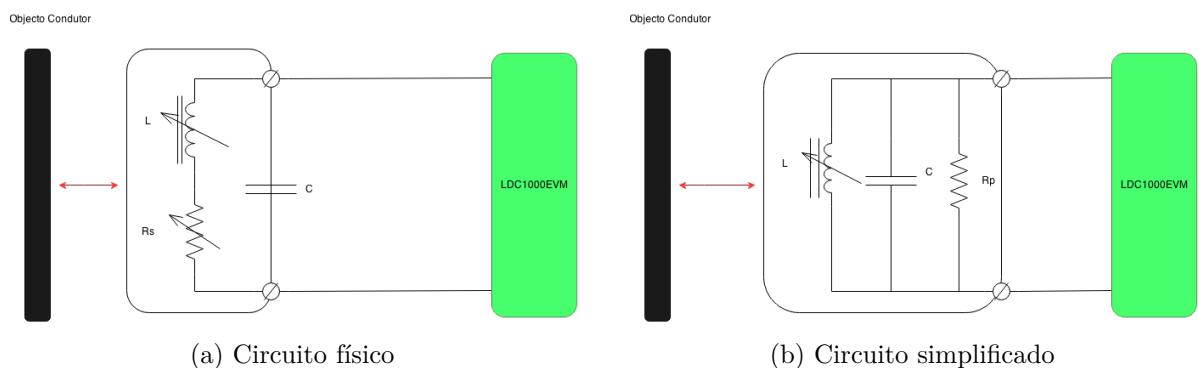


Figura 5: Esquemático do sensor

É importante referir que o circuito presente na figura 5a é o circuito real do sensor e o da figura 5b é uma simplificação que em termos prácticos permite ao sensor variar a sua gama de funcionamento afinando o valor minimo e máximo de R_p . Este processo faz com que o sensor possa discriminar diferentes tipos de metal se bem afinado e projectado.

³Simboliza as perdas associadas ás correntes de Eddy

5.1.2 Arquitectura

Em termos de hardware para este trabalho foi utilizado um computador que vai controlar o Roomba e o sensor LCD1000EVM. No sensor foi ainda substituída a bobine por uma bobine e um condensador de valores por nós pretendidos.

O robô Roomba encontra-se conectado por usb a um computador que o controla através do teclado. Este computador encontra-se ainda ligado a um sensor LCD1000EVM que por sua vez tem ligado a si uma bobine e um condensador por nós projectados e adaptados.

Na figura 6 é visível o robô já adaptado à nova bobine. Foi necessário criar uma estrutura que suportasse a bobine mas que mantivesse a estabilidade do robô, para tal é necessário o uso de contra-pesos como é visível.



Figura 6: Imagem real do robô

Na figura 6 é visível o robô já adaptado à nova bobine. Foi necessário criar uma estrutura que suportasse a bobine mas que mantivesse a estabilidade do robô, para tal é necessário o uso de contra-pesos como é visível.

A estrutura que suporta a bobine é facilmente removível estando apenas presa por abraçadeiras. Esta estrutura garante tanto a estabilidade do robô como da bobine. A bobine encontra-se ligada em paralelo ao condensador e ao sensor numa breadboard algo que pode ser visto na figura 2.

5.1.3 Implementação

Como foi anteriormente explicado, o circuito pode ser alterado e dimensionado para se trabalhar na gama de valores desejada. A seguir iremos explicar os passos necessários para implementar o novo circuito e adaptá-lo ao sensor.

1. O primeiro passo será escolher simplesmente a nova bobine por nós pretendida e escolher um condensador que permita cumprir as condições impostas à frequência de ressonância(1) e à resistência de ressonância(2).
2. O próximo passo será com o auxílio de um “LCR meter”, colocando as pontas deste medidor em paralelo com o paralelo do condensador e da bobine que escolhemos têmos que anotar o valor da resistência R_p lido no medidor, na ausência de qualquer objecto magnético.
3. Neste passo temos que colocar um objecto magnético o mais próximo possível da bobine e anotar o novo valor de R_p .
4. É necessário anotar também a frequência de varrimento que se encontra definida no medidor.
5. No passo seguinte é necessário actualizar os registos do sensor para que se adapte ao novo circuito. Para modificar os registos é necessário conectar o sensor a um computador coonduoresm windows e utilizando a gui application fornecida pela texas instruments na secção de alteração dos registos teremos que modificar três registos. O valor do registo R_{pMAX} deve ser alterado para o dobro do valor anotado na ausência de objectos c, como os valores disponíveis são discretos deve ser escolhido o mais próximo deste valor por excesso. O valor do registo R_{pMIN} deve ser alterado para metade do valor anotado na presença de um objecto condutor, como os valores disponíveis são discretos deve ser escolhido o mais próximo deste valor por defeito. O registo da freqüência deve ser alterado para no máximo 80% do valor da frequência de varrimento.
6. Por fim é necessário partir o sensor onde diz “coil perforation” na figura 7 e soldar o paralelo da bobine e do condensador no sítio onde diz “conections for custom LC tank” na figura 7.



Figura 7: LCD1000EVM com indicação das perfurações

Com o auxílio do medidor LCR meter figura 8 conseguimos adaptar o sensor à nova bobine. Para cumprir o primeiro passo referido anteriormente, primeiro medimos a indutância da nova bobine e de seguida escolhemos um condensador e medimos a sua capacitância.



Figura 8: LCR Meter 186

Medimos o condensador e a bobine utilizando o aparelho da figura 8 obtendo 3.09 nF e 1.71 mH . De seguida, utilizando (1) calculámos a nova frequência de ressonância e obtivemos:

$$f_{\text{ressonância}} = \frac{1}{2\pi\sqrt{1.7098 \times 10^{-3} \times 3.0863 \times 10^{-9}}} = 69.283 \text{ kHz} \quad (3)$$

Como podemos concluir, a a frequência calculada está dentro da gama pretendida.

De seguida anotámos o valor de R_p na ausência e na presença de objectos condutores obtendo os seguintes valores respectivamente 18.29 kΩ e 8.79 kΩ . Para a obtençā destes valores foi usada uma frequência de varrimento de valor 66.667 kHz .

Garantimos assim que também a resistência de ressonância se encontra dentro da gama de valores do referidos anteriormente. Por fim, fizémos a alteração dos registos do sensor para se adaptar aos novos valores, pelo que foram modificados os seguintes registos:

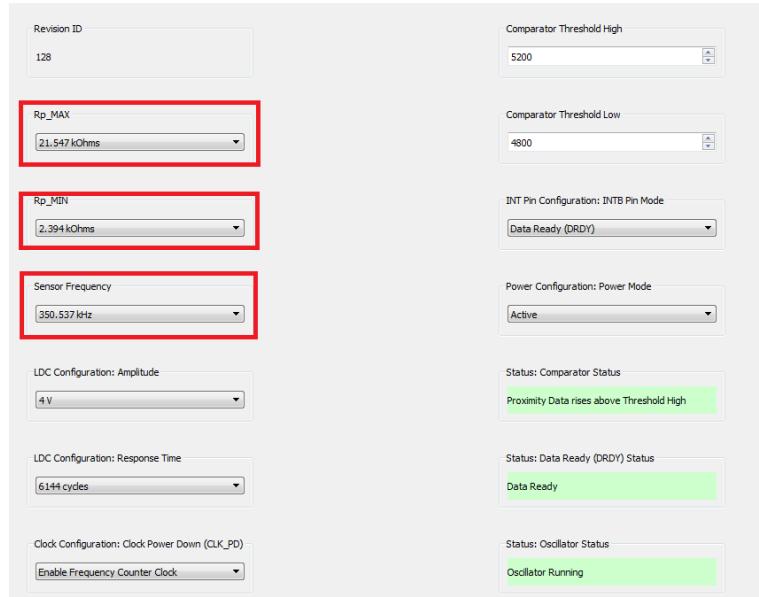


Figura 9: Registos do sensor LDC1000EVM

- $R_{P_{MAX}}$ foi alterado para 38.785 kΩ , valor que foi aproximado por excesso como era exigido.

2. $R_{P\text{MIN}}$ foi alterado $4.309 \text{ k}\Omega$, valor que foi aproximado por defeito como era exigido.
3. Sensor frequency foi alterado para 52.231 kHz , valor que foi aproximado para o valor mais próximo de 80% da frequência de varrimento, por defeito.

Os registos que foram alterados encontram-se a vermelho na figura 9. Com todos os passos cumpridos o novo sensor ficou pronto a ser utilizado.

5.2 Software

5.2.1 Como executar

Em termos de aplicação ROS, elaboramos dois tipos de programas. O primeiro contém um nó que lê valores do sensor e publica esses dados usando mensagens e um outro nó que recebe esses dados e imprime para o ecrã. O segundo usa um ó que publica os dados por mensagens e um outro nó recebe esses dados e analisa se se trata de uma mina e em caso afirmativo marca-a num mapa desenhando um cubo de pequenas dimensões na posição em que detectou a mina. Abaixo encontram-se os passos necessários para os executar:

Primeiro Codigo:

1. roscore
2. rosrun sensor le_sensor threshold(int)
3. rosrun sensor recebe_dados

Segundo Codigo:

1. roscore
2. roslaunch roomba_bringup roomba_555.launch
3. roslaunch sensor sensor.launch
4. rosrun sensor le_sensor threshold (int)
5. rosrun sensor points_sensor
6. roslaunch roomba_teleop keyboard_teleop.launch
7. rosrun rviz rviz

No parametro Treshold(int) é preciso colocar um inteiro que funcionará como comparador. Quanto á configuração do rviz é necessário carregar o modelo do robot, as $tf's$, os markers e o array de markers. Para poder executar correctamente o programa é necessário fazer sudo chmod 777 /dev/ttyUSB0 e sudo chmod 777 /dev/ttyACM0 para garantir que conseguimos abrir a porta série do Roomba e do sensor.

5.2.2 Estrutura do Código

Como é possível observar na figura 10, o código principal que é executado interliga-se entre os diferentes programas e o Roomba da seguinte maneira:

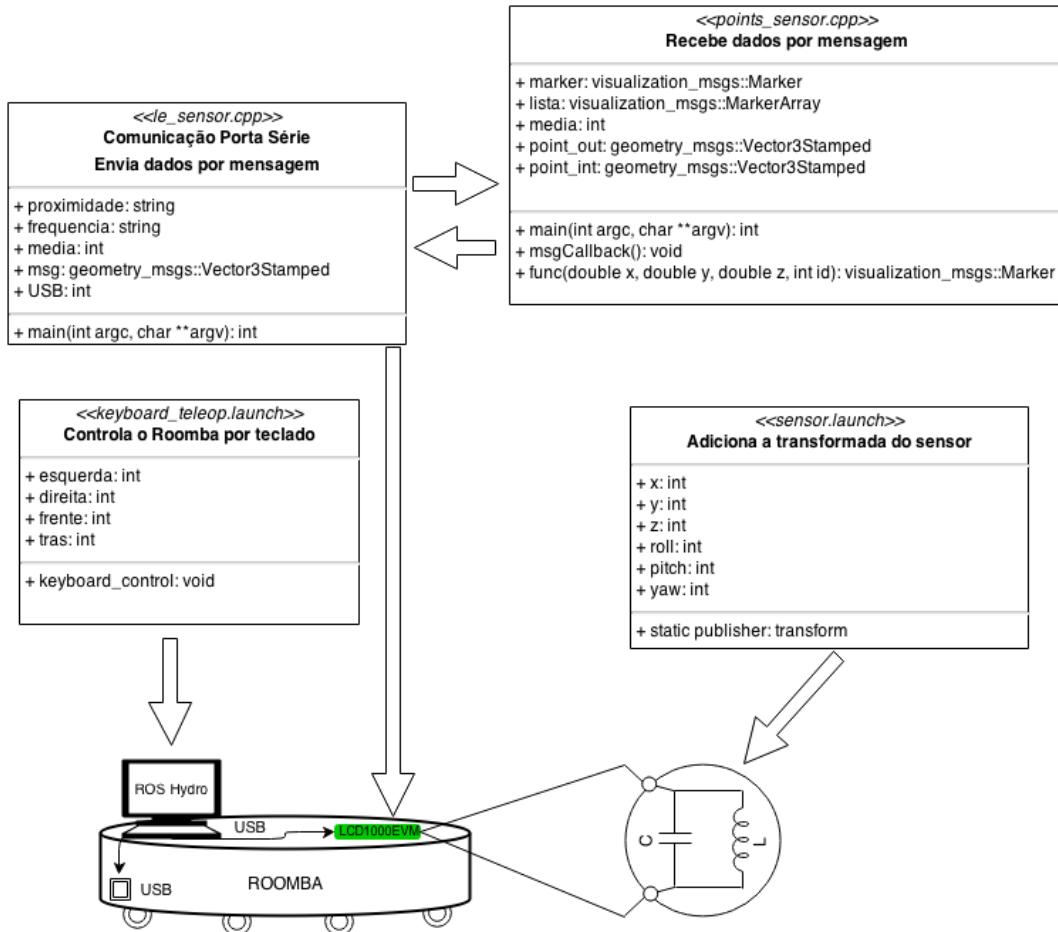


Figura 10: Interligação dos diferentes programas

Em termos prácticos o ficheiro **le_sensor.cpp** comunica co o sensor por porta série e publica os dados numa variavel predefinida no ROS, mensagens do tipo `geometry_msgs::Vector3Stamped`. O ficheiro **point_sensor.cpp** sempre que é publicado alguma mensagem com o frame_id sensor é recebida calcula-se a posição do sensor no mapa virtual e verifica-se se está na presença de uma mina. O ficheiro launch **sensor.launch** é uma transformação do sensor em relacão é transformada odom (odométria). Como o sensor não apresenta nem possibilidade de translação nem rotação podemos simplesmente fazer um static transform publisher[2]. Para comandar o Roomba é usado o ficheiro launch **keyboard_teleop.launch**, que contém a velocidade linear e a velocidade de rotação default. Este launcher executa um executável que lê do terminal as setas do teclado e sendo assim é possivel comandar/deslocar o robo.

5.2.3 Utilização do ROS

Utilizámos o Robot Operating System (ROS) como plataforma de comunicação entre os dois periféricos por nós utilizados e o computador. Com o ROS é possível implementar bibliotecas de

cliente ROS como roscpp, rospy ou roslib. Por outras palavras significa que facilmente se consegue adaptar programas criados em outras línguas (de entre C++, Python e LISP). No nosso caso em específico adaptámos o programa de leitura do sensor em cpp para a plataforma ROS. Numa segunda parte através do API disponibilizado pelo Gonçalo Cabrita[5] e pelo Bruno Gouveia[4] conseguimos controlar o Roomba pelo teclado do computador. Através da leitura dos encoders e do controlo dos motores foi possível controlar a velocidade angular e linear do robô.

Utilizando os nós do ROS foi possível assim controlar o robô e utilizá-lo para a deteção e marcação nas minas. De uma forma mais detalhada foi usada como referência a transformada "ODOM" que nos fornece a posição inicial em que o robô se encontra, com o nó entre esta transformada e o base link do Roomba sabe-se sempre a posição relativa à posição inicial. De seguida criando o nó da transformada sensor(criada por nós) é assim possível, com a informação devolvida pelo sensor, saber se o robô se encontra na presença de uma mina. Caso seja verificada a presença de uma mina são enviadas as coordenadas (x,y,z) para uma função "markers" que irá marcar no mapa as minas em relação a "ODOM", ou seja em relação à posição inicial do robô.

5.2.4 Comunicação com o sensor

A primeira parte deste trabalho foi concluir que o sensor utiliza porta série como protocolo de comunicação quando comunica com o computador. Como tal, para facilitar o uso futuro deste sensor criámos código para as leituras do sensor em três plataformas diferentes de programação. Adaptámos por isso o nosso código às plataformas C++, Matlab e Python⁴. Estes três códigos encontram-se mais a baixo na tabela(Fazer referencia à tabela).

Analisando as partes mais importantes do código, o primeiro passo é definir a porta série pela qual estamos a comunicar, no nosso caso em particular foi a porta "COM9" em *Windows* e "/dev/ttyACM0" em *Linux*. De seguida definimos o tipo de controlo de fluxo, que neste caso é do tipo "software flow control". É necessário definir a frequência de comunicação, Baud Rate, que foi definida por nós a 9600. O próximo passo é definir a paridade e o número de bits da comunicação, neste caso não existe paridade e são utilizados 8bits na comunicação. Tal como na maioria dos dispositivos electrónicos na comunicação por porta série apenas é necessário de 1 "stop bit", foi ainda definido o tempo de espera para leitura máximo("Timeout") a 5 segundos. Definimos o modo de leitura contínua para funcionar de forma assíncrona. Por fim ao escrever na porta série o comando 0x33(hexadecimal) ou 3(ASCII), iniciamos a stream de dados.

5.2.5 Protocolo VNC

Este protocolo baseia-se no conceito de partilha remota de um computador. Para isso existe um computador que partilha o monitor e, se o utilizador permitir, os controlos do mesmo (servidor) e um ou mais computadores (clientes) que se conectam ao servidor e são capazes de realizar as acções permitidas pelo mesmo .

Como a base do VNC foi construída/projectada sobre o protocolo TCP/IP, este é capaz de funcionar via internet. É usado em aulas interactivas onde um professor permite a visualização do seu monitor e os alunos conectam-se podendo assim visualizar os passos do professor.

Nota: Rápido e eficaz numa rede local com boa largura de banda.

⁴O código fonte encontra-se em anexo



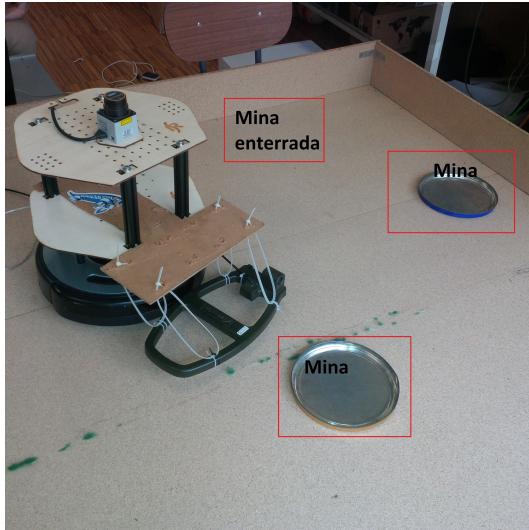
Figura 11: Protocolo VNC

6 Resultados

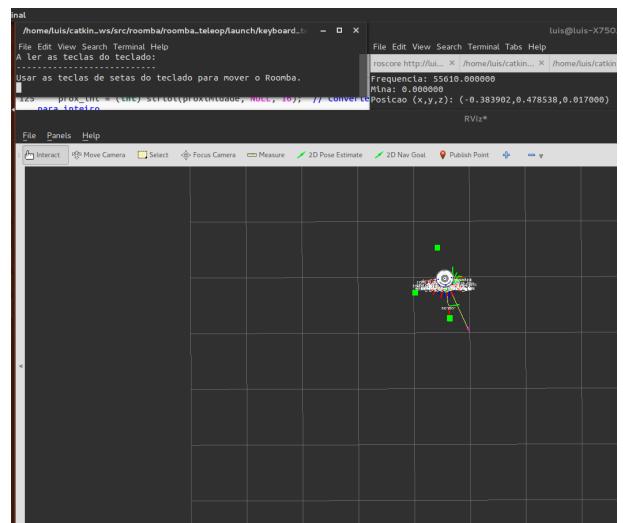
Realizamos diversos testes de modo a colocar à prova o robô em diversas situações. Acabámos por perceber que o robô detecta minas mesmo quando estas se encontram debaixo do solo.

Por fim acabámos por testar o robô numa superfície em que garantimos que não existem quaisquer condutor. Fizemos isto porque, no sítio em que nos encontramos a trabalhar a instalação eléctrica foi toda feita pelo solo e como consequência o robô detecta minas em diversos sítios algo que é lógico dado que os campos magnéticos gerados pelos cabos eléctricos vão interagir com o campo magnético do nosso sensor. Na figura 12a temos identificadas três minas, duas visíveis e uma enterrada e temos o robô na sua posição inicial.

A figura 12 foi tirada de um ângulo contrário à direção inicial do robô sendo que o mapa virtual obtido é o o mapa real



(a) Mapa real



(b) Mapa virtual

Figura 12: Teste para detecção de minas num labirinto

7 Conclusões

Com este projecto foi possível adaptar o robô ROOMBA para a deteção de minas com o auxílio do sensor LCD1000EVM. Numa primeira parte do projecto foi criada a interface para que o sensor pudesse ser utilizado em Linux. Com o auxílio da plataforma ROS (Robot Operatyng System) foi possível fazer a comunicação dos três periféricos (computador, robô e sensor) esta plataforma permitiu ainda criar o controlo por teclado da velocidade e direcção do robô. Por fim foi possível adaptar o sensor para uma bobine por nós desejada.

Este projecto permitiu perceber a potencialidade tanto da plataforma ROS como do sensor LCD1000EVM. A grande vantagem da plataforma ROS é que no futuro muito facilmente qualquer pessoa poderá adaptar este sensor a outros robôs ou outros sistemas robóticos, isto deve-se à vantagem do sistema de transformadas em que o ROS trabalha. O sensor LCD1000EVM tem a grande vantagem de com alguns pequenos passos se poder modificar e adaptar a qualquer bobine que o utilizador pretenda, criando-se assim um sensor à medida do que o utilizador precisa.

Por fim foi conseguido utilizar o robô para a detecção de minas mesmo quando estas se encontram enterradas.

Referências

- [1] Robot operating system. <http://wiki.ros.org/>, 2007. Hydro Edition.
- [2] Ros static transform. http://wiki.ros.org/tf#static_transform_publisher, 2013.
- [3] Lrc meter, 2014. http://en.wikipedia.org/wiki/LCR_meter.
- [4] Gouveia Bruno. frames id. https://github.com/ras-sight/hratc2014_entry_template, 2014.
- [5] Cabrita Gonçalo. cereal port. https://github.com/goncabrita/cereal_port, 2013.
- [6] Cabrita Gonçalo. Roomba api. <http://wiki.ros.org/Robots/Roomba>, 2013.

8 Anexos

8.1 Código C++

```
// Cout
#include <iostream>

// Open
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

// Write e Sleep
#include <unistd.h>

// Perror
#include <stdio.h>
#include <errno.h>

// Memset
#include <string.h>

// Strtol
#include <stdlib.h>

/*
? --> Menu
3 --> Start
4 --> End

*/
using namespace std;

void leDados( int USB );
void mostraMenu( int USB );
void PiscaLeds( int USB );

int main()
{
```

```

int USB;

const char *portname = "/dev/ttyACM0";

USB = open (portname , O_RDWR | O_NOCTTY);

if (USB < 0)
{
    cout << "Erro " << errno << " - Abrir Porta Serie: " << portname << endl;
}
else
{

cout << "leDados(USB) - Proximidade,Frequencia:" << endl;
leDados(USB);

//cout << "\n\nmostraMenu(USB):" << endl;
//mostraMenu(USB);

//cout << "PiscaLeds(USB):" << endl;
//PiscaLeds(USB);

}

write(USB,"4",1);
close(USB);

return (0);
}

void mostraMenu( int USB)
{
    int n;
    char buf[21];

    memset(buf,0,sizeof(buf));

    write(USB,"?",1);

    for( int i = 0; i < 12; i++)

```

```

{
    n = read(USB,buf , sizeof(buf));

    buf[n] = '\0';
    printf ("\n%#s",buf);
}

void leDados( int USB)
{
    int n;
    char buf[16];
    char proximidade[4];
    char frequencia[4];

    int prox_int;
    int freq_int;

    memset(buf,0 , sizeof(buf));

    write(USB,"3" ,1);
    read(USB,buf , sizeof(buf)); // a primeira stream e "stream start"

    for( int i = 0; i < 10; i++)
    {

        n = read(USB,buf , sizeof(buf));
        buf[n] = '\0';

        proximidade[0] = buf[2];
        proximidade[1] = buf[3];
        proximidade[2] = buf[4];
        proximidade[3] = buf[5];
        proximidade[4] = '\0';

        frequencia[0] = buf[9];
        frequencia[1] = buf[10];
        frequencia[2] = buf[11];
        frequencia[3] = buf[12];
        frequencia[4] = '\0';

        prox_int = (int) strtol(proximidade , NULL, 16); // Converte hexadecimal para
        freq_int = (int) strtol(frequencia , NULL, 16);
    }
}

```

```

    printf ("\nProximidade: %d Frequencia: %d", prox_int, freq_int);

    printf ("\nValores: %s", buf);

}

void PiscaLeds(int USB)
{
    int n;
    char buf[16];

    memset(buf, 0, sizeof(buf));

    for (int i = 0; i < 8; i++)
    {
        write(USB, "7", 1);
        sleep(1);
    }
}

```

8.2 Código Python

```
# sudo chmod 0777 /dev/ttyACM0
# phyton main.py
# sudo chmod 0777 /dev/ttyACM0 && clear && python main.py

import serial
import string
import matplotlib.pyplot as plt

fig=plt.figure()
plt.axis([0,10000,0,12000])

i=0
x=list()
y=list()

plt.ion()
plt.show()

USB = serial.Serial("/dev/ttyACM0",9600)

USB.write("3")

hex_digits = set(string.hexdigits)

while True:
    if USB.inWaiting():
        leitura = USB.read(14)
        proximidade = leitura[2:6]
        frequencia = leitura[9:12]

        if all(c in hex_digits for c in proximidade): # verifica se é hexadecimal
            prox_int = int(proximidade,16)
            freq_int = int(frequencia,16)

            print("Proximidade: {} Frequencia: {}".format(prox_int,freq_int))

            x.append(i)
            y.append(prox_int)
            plt.scatter(x,y)
            i+=200
            plt.draw()

USB.close()
```

8.3 Código Matlab

```
%% Open serial port
clc

%fclose(instrfind)

sport = serial('COM9'); % assigns the object s to serial port
set(sport, 'RecordDetail', 'verbose')
set(sport, 'InputBufferSize', 64*4096); % number of bytes in input buffer
set(sport, 'OutputBufferSize', 16*4096); % number of bytes in output buffer
set(sport, 'FlowControl', 'software');
set(sport, 'BaudRate', 9600);
set(sport, 'Parity', 'none');
set(sport, 'DataBits', 8);
set(sport, 'StopBit', 1);
set(sport, 'Timeout', 5);
set(sport, 'ReadAsyncMode', 'continuous');

fopen(sport); %opens the serial port

fwrite(sport, '08271000') % 10kHz samples frequency
fwrite(sport, '3')

while true
fscanf(sport)

end

%% Close serial port
fclose(sport);
delete(sport);
clear sport
```