Como criar um rádio-controlador para um carro, com os aparelhos Nokia N97 ou Nokia 6260 slide

Instruções

Vários projetos tem se caracterizado um rádio-controlador (RC) do carro que foi controlado com um telefone celular em vez de um controlador de rádio tradicional. Comumente, a comunicação entre o carro e o aparelho celular foi feito usando uma conexão Bluetooth, e os comandos enviados para o carro foram baseados em um sensor do acelerador no dispositivo móvel. A peça de rádio no carro foi substituída com um módulo Bluetooth feito especificamente para receberem os comandos do dispositivo móvel.

Este documento apresenta duas soluções diferentes para controlar um carro RC com um dispositivo móvel. O primeiro usa o dispositivo Nokia N97, que suporta o padrão Mobile Sensor API (JSR-256). O segundo usa o dispositivoNokia 6260 slide, em que uma tecla joystick é usada para controlar o carro. Em ambos os casos, o carro é controlado através de um pequeno celular e uma aplicação Java ™, que cria e envia as mensagens para controlar o carro.

O principal objetivo deste documento é descrever a parte do software do projeto, como os MIDlets controladores que foram criados, testados e utilizados. Os leitores devem ter pelo menos um conhecimento básico de programação Java e como criar MIDlets.

O projeto MiniController

Este projeto foi iniciado no final de Março de 2009, cerca de dois meses antes da manifestação prevista na primeira vez que o evento Nokia Developer Summit aconteceu. O evento foi nos dias 28 e 29 de maio, em Mônaco. A intenção era ter um demo stand para demonstrar o controle do carro de RC com um dispositivo móvel e também para compartilhar as últimas informações sobre S60 e Symbian Series 40 e recursos relacionados.

O projeto consistiu-se na construção do carro, e escrever, testar e ajustar o controlador de MIDlets. O tempo total gasto foi de aproximadamente uma semana, mas a obra foi distribuída ao longo de dois meses. Alguns documentos bem escritos sobre projetos anteriores (como <broken link to Technical_Background_of_ShakerRacer.pdf >) foram muito úteis na criação deste projeto.

Sobre o carro

O carro utilizado foi um modelo-padrão feito por Tamiya: o RC MINI Cooper S 2006. Nenhuma alteração no carro foi feita, exceto o módulo de rádio foi substituído com o módulo Bluetooth suplementar.

O módulo Bluetooth foi ordenado a partir de Alta Áustria University of Applied Sciences, do Departamento de Computação Móvel (CM) .

Figura 1: Construindo o carro

(C) Copyright Nokia 2013. All rights reserved.

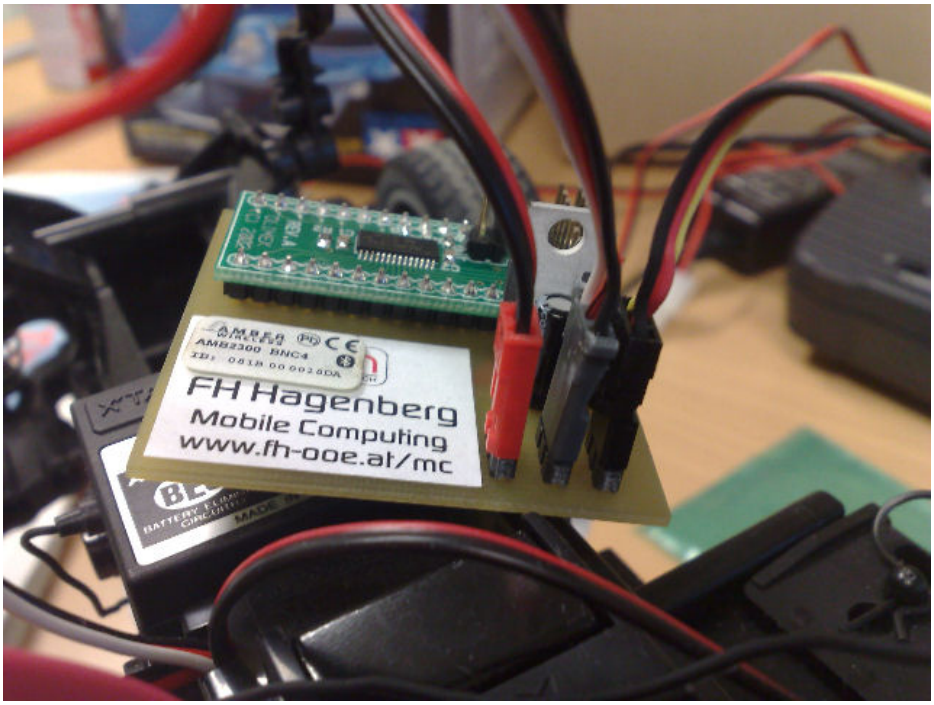

Figura 2: O módulo Bluetooth ligado ao carro

Sobre o controlador de MIDlets

Existem dois MIDlets separados para controlar o carro, para cada um dos (dois) telefones móveis usados. Esta parte pretendia escrever os mesmos códigos para os dispositivos, mas por causa das diferenças de hardware se justificava a escrita de dois MIDlets separados. O dispositivo Nokia N97 tem uma Mobile Sensor API (JSR-256), um ecrã táctil e uma nHD (640 x 360 pixels) o tamanho da tela, enquanto o dispositivo Nokia 6260 slide tem suporte a JSR-256, sem touchscreen, e um ecrã QVGA . O dispositivo Nokia 6260 slide tem um joystick sensível[5], que é ideal para o uso do controlador.

Em ambos os MIDlets, as mensagens são enviadas para o carro e são geradas com base na entrada do usuário, e elas são enviadas para o carro para conexão Bluetooth.

Figura 3: Dispositivos Nokia N97 e Nokia 6260

MiniController MIDlet para o dispositivo Nokia N97

A questão mais importante no projeto foi demonstrar o novo (Sensor de API móvel) Mobile Sensor API suportado em dispositivos Nokia. O Nokia N97 foi o primeiro a ter esta API como um recurso interno. Há realmente quatro sensores suportados pela API no dispositivo Nokia N97: acelerômetro, sensor de nível de carga da bateria, sensor de status de carregamento, e sensor de nível de sinal.

O MIDlet MiniController usa o acelerômetro para controlar a posição do dispositivo, e com base nas informações que cria as mensagens controladoras, que são então enviados para o carro.

O MIDlet consiste em seis classes:

Class	**Main purpose**
MiniController.java	Classe main do MIDlet.
LogCanvas.java	Canvas para mostrar a saída, por exemplo, a descoberta de dispositivo Bluetooth e serviços de descoberta.
SensorCanvas.java	Tela Canvas real para mostrar os valores dos sensores, criando a ligação Bluetooth, criando as mensagens, e enviando as mensagens para o carro. Esta tela também possui alguns botões de ajuste para a fixação dos deslocamentos para controlar os valores.
DeviceDiscoverer.java	Executa a instrução do dispositivo, conforme especificado na API Bluetooth (JSR-82). Nota: Atualmente não utilizada, porque esta só precisa ser feita uma vez.
ServiceDiscoverer.java	Executa a busca de serviços, conforme especificado na API Bluetooth (JSR-82). Nota: Atualmente não utilizada, porque esta só precisa ser feita uma vez.
SettingsForm.java	Um formulário para inserir diferentes tipos de configurações. Atualmente, esse recurso não é utilizado.

Tabela 1: Classe MiniController MIDlet

Principais características da MIDlet

Copyright Nokia 2013. All rights reserved.

No início do desenvolvimento do trabalho, o MIDlet foi usado para procurar os dispositivos Bluetooth, bem como os serviços em si. Durante o desenvolvimento da aplicação tornou-se claro que o a procura do dispositivo não seria necessário ser feita novamente, uma vez que a URL para o serviço foi corretamente encontrada. Isso também torna nosso `LogCanvas` completamente inútil. No entanto, esses recursos foram salvos porque eles são necessários se o módulo de Bluetooth é substituído por um novo (ou alguém mais está usando o MIDlet para o seu teste).

No aparelho Nokia N97, é possível obter os valores acelerômetro, que indicam a posição do telefone móvel. Com base nas informações recolhidas a partir de projetos semelhantes, controlando o carro inclinando o dispositivo móvel é muito intuitivo. Além disso, os dados recebidos do sensor é bastante precisa, e é recebido com rapidez suficiente para controlar a trabalhar muito bem.

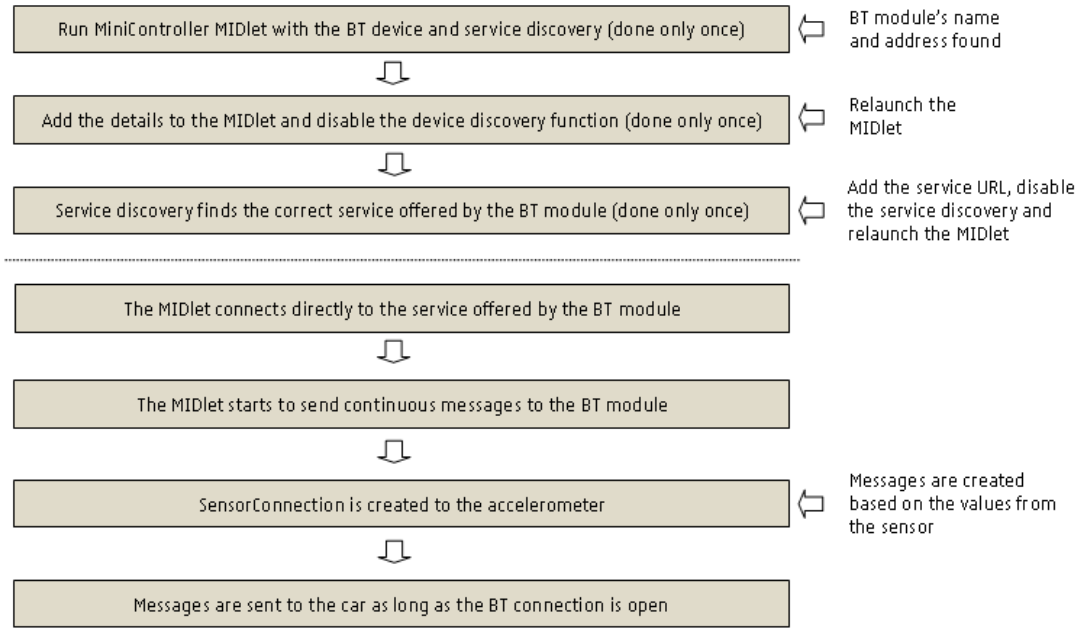

Figure 4: Simplificador do fluxo de processo de MiniController MIDlet

Usando a API de Sensor de movimento

A API Mobile Sensor (JSR-256) é uma API opcional, que oferece uma forma unificada de gerenciamento de sensores conectados a dispositivos móveis, e acesso fácil aos dados do sensor. O primeiro dispositivo da Nokia para apoiar essa API é o Nokia N97. Atualmente (abril 2009), há um SDK Nokia disponível que suporta esta API: o Nokia N97 SDK 0.5.

Usando a API isso torna possível informações sobre os sensores apoiada pela implementação Java Mobile. O dispositivo Nokia N97 mostra informações sobre dois acelerômetros diferentes, um sensor pode retornar valores inteiros e retornar valores de um outro caso. Neste caso, os valores inteiros foram usados. O método abaixo mostra como a escolha do sensor é feita. O método `findSensors()` `SensorInfo` e os objetos de todos os sensores do tipo «aceleração». Em seguida, digite o sensor de dados e verifique o sensor com o tipo de dados TYPE_INT.

Geralmente, se o sensor mede diferentes propriedades ou dimensões, simultaneamente, esses valores são considerados canais independentes. Os tipos de dados são realmente características dos canais. Assim, no caso do acelerômetro, o sensor está retornando três valores, um de cada canal (axis_x, axis_y e axis_z). No nosso caso, queremos usar o acelerômetro, que retorna valores inteiros (o tipo de canal de dados é TYPE_INT).

```
private SensorConnection openSensor() {
 infos = SensorManager.findSensors("acceleration", null);
 if (infos.length==0) return null;
 int datatypes[] = new int[infos.length];
 int i = 0;
 String sensor_url = "";
 while (!sensor_found) {
  datatypes[i] = infos[i].getChannelInfos()[0].getDataType();
  if (datatypes[i] == 2) { //ChannelType.TYPE_INT = 2
   sensor_url = infos[i].getUrl();
   sensor_found = true;
  }
  else i++;
 }
 try {
  return (SensorConnection)Connector.open(sensor_url);
 }catch (IOException ioe) {
 ioe.printStackTrace();
  return null;
 }
}
```

Exemplo 1: O método openSensor() da classe SensorCanvas, é usado para encontrar o sensor correto e para a criação do SensorConnection.

Quando o sensor correto é encontrado e a ligação é estabelecida, `DataListener` está configurado para escutar os valores dos sensores. Observe o BUFFER_SIZE como um parâmetro para o método `setDataListener()` . Neste caso, um tamanho de 4 foi utilizado. O registo `DataListener` recebe os dados dentro da sequencia `DataReceived()`. A notificação é enviada quando o número de valores recolhidos dos dados é igual a um conjunto de tamanho do buffer. O tamanho do buffer foi baseado em testes, a ideia era a de limitar a quantidade de dados a serem enviados para o módulo Bluetooth. O buffer de valores também permitiu facilmente calcular uma média dos valores de reserva, que foram então utilizados como valores reais para os controladores.

O `DataListener` de estado é controlado através de uma variável booleana IsStopped. Quando essa variável recebe o valor 'true', o `DataListener` é removido e a conexão `SensorConnection` está fechada.

(C) Copyright Nokia 2013. All rights reserved.

```
private synchronized void initSensor() {
 sensor = openSensor();
 if (sensor == null) return;
 try {
  sensor.setDataListener(this, BUFFER_SIZE);
  while(!isStopped){
   try {
    wait();
   } catch(InterruptedException ie){}
  }
  sensor.removeDataListener();
 }catch (IllegalMonitorStateException imse) {
  imse.printStackTrace();
 }catch (IllegalArgumentException iae) {
  iae.printStackTrace();
 }
 try {
  sensor.close();
 } catch(IOException ioe){
  ioe.printStackTrace();
 }
 if (isStopped) {
  sensor = null;
 }
}
```

Exemplo 2: O método initSensor() da classe SensorCanvas, que chama o método openSensor() para abrir a conexão SensorConnection, define o DataListener, e é responsável por fechar a conexão.

Quando todos os preparativos forem feitos, `DataReceived()` está vindo continuamente. Os dados recebidos do `DataReceived()` consiste na `SensorConnection` , onde os dados são provenientes de dados reais como uma matriz de objetos de dados e um valor booleano, que nos diz, se alguns dados são perdidos. No nosso caso, apenas os dados reais são usados.

Um valor médio para cada direção é calculado e, em seguida, as mensagens são criadas.

```
/**
  * Notificação de dados do sensor recebidas.
  * @ Param sensor - SensorConnection, a origem dos dados recebidos
  * @ Param dados - os dados recebidos do sensor
  * @ Param isDataLost - true se alguns dados foram perdidos
 */
public void dataReceived(SensorConnection sensor, Data[] data, boolean isDataLost) {
 int[] directions = new int[3];
 for (int i = 0; i < data.length; i++) {
  int values[] = data[i].getIntValues();
  int temp = 0;
  for (int j = 0; j < values.length; j++) {
   temp = temp + values[j];
  }
  temp = temp / BUFFER_SIZE;
  directions[i] = temp;
 }
 x_int = directions[0];
 y_int = directions[1];
 z_int = directions[2];
 createMessageData();
 repaint();
}
```

Exemplo 3: O método DataReceived() da classe SensorCanvas, que é chamado quando o número de valores recolhidos de dados é igual a um conjunto de tamanho do buffer (BUFFER_SIZE)

Usando a API Bluetooth

Atualmente, o MIDlet cria a conexão `StreamConnection` diretamente para o serviço disponível no módulo de Bluetooth do carro. O dispositivo correto foi pesquisado usando o método padrão `discoveryAgent.startInquiry()` :

```
try {
 LocalDevice localDevice = LocalDevice.getLocalDevice();
 DiscoveryAgent agent = localDevice.getDiscoveryAgent();
 agent.startInquiry(DiscoveryAgent.GIAC, this);

catch(BluetoothStateException bse) {
 bse.printStackTrace();
}
```

Example 4: Code demonstrating how the device inquiry can be done

The device names and addresses were printed out in `deviceDiscovered()` method. The `RemoteDevice` found was added to a `Vector` (remoteDevices) for later use.

```
public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
 String address = btDevice.getBluetoothAddress();
 String name = "";
 try {
  name = btDevice.getFriendlyName(false);
 } catch (IOException ex) {
  ex.printStackTrace();
 }
 midlet.logCanvas.addString(name + "(" + address + ")");
 if (address.equals("0018DA00081B")) {
  midlet.logCanvas.addString("0018DA00081B found!");
  agent.cancelInquiry(this);
 }
 remoteDevices.addElement(btDevice);
}
```

Exemplo 5: O método deviceDiscovered() da classe DeviceDiscoverer, que é chamado quando um dispositivo é encontrado durante um inquérito.

Da mesma forma, o método `LocalDevice.searchServices()` foi usado para encontrar os serviços. A URL correta foi encontrada usando a

(C) Copyright Nokia 2013. All rights reserved.

`ServiceDiscoverer` (classe de execução `DiscoveryListener`) e seu método `serviceSearchCompleted()` . Este método é chamado quando o serviço de busca está concluído. Neste método, o método da classe MiniController 'searchCompleted()' é chamado, com a classe `ServiceRecord` e seu parâmetro. O método `searchCompleted()` passa pelo array `ServiceRecord` , que neste caso consiste em apenas um serviço. A URL é encontrada usando método `ServiceRecord.getConnectionURL()` :

```
for (int i = 0; i < servRecord.length; i++) {
 url = servRecord[i].getConnectionURL(ServiceRecord.AUTHENTICATE_NOENCRYPT, false);
 logCanvas.addString("URL: " + url);
}
```

Exemplo 6: Código demonstrando como a URL do serviço é encontrada

Nota: Como mencionado anteriormente, o DeviceDiscoverer e a classe ServiceDiscoverer não são utilizadas na aplicação MiniController. Elas podem ser usadas para procurar dispositivos e serviços simplesmente removendo as '//' a partir do início da linhas comentadas em dois locais:

1. Na classe LogCanvas, método paint(), primeira linha:
   //midlet.startDeviceDiscovery(); //NÃO É NECESSÁRIO, se o serviço da BT URL é conhecido.
2. Na classe MiniController, método inquiryCompleted(), linhas:
   //serviceDiscoverer = new ServiceDiscoverer(this); //Não é necessária se
   //serviceDiscoverer.startServiceSearch(devices[index]); // a URL é reconhecida

Também na classe MiniController, a seguinte variável precisa ser mudada quando o nome correto do dispositivo e serviço de URL foram encontrados:
protected String device = "BNC4-081B000018DA";
protected String url = "btspp://0018DA00081B:1;authenticate=true;encrypt=false;master=false";

Criando as mensagens

Depois que a conexão foi criada, o MIDlet começa a enviar mensagens para o carro. Há mensagens separadas para aceleração e direção, e elas são enviados continuamente em um segmento separado. Entre cada envio, um pequeno atraso de 50 ms é realizado.

Os detalhes das mensagens são explicados na Figura 11 em [1]. Em resumo, a mensagem é constituída por seis bytes, como mostrado abaixo:

- Preâmbulo: 1 byte, sempre 0xFF, usado para identificar o início de um frame. DeviceID, MessageID e DataLength não deve ser 0xFF.
- DeviceID: 1 byte, identifica o aparelho, neste caso, 3 (direção) ou 4 (aceleração).
- MessageID: 1 byte, identifica o tipo de mensagem que um dispositivo envia ou recebe, ou 0 para o recebimento do carro ou 1 para enviar para o carro.
- DataLength: 1 byte, indica quantos bytes de dados seguirão (0 - 255).
- Dados: 0 - 255 bytes.
- Checksum: 1 byte, calculado através da realização de um modulo de soma 256 sobre todo o pacote (preâmbulo, DeviceID, MessageID, DataLength, e dados).

Existe um recurso conhecido na linguagem de programação Java, que seus Bytes são assinados valores de -128 a 127. Tal como acima demonstrado, o preâmbulo deve ser 0xFF! Bytes. Após alguns testes com matrizes de bytes e inteiro, parecia que a utilização do método `DataOutputStream.write (int b)` funciona muito bem. Conforme consta no MIDP 2.0 javadoc do método "escreve o byte especificado (e baixa oito bits do argumento b) e o fluxo de saída subjacentes", e o parâmetro b é o byte a ser escrito, mesmo que seja dado como um inteiro .

Uma mensagem (conjunto de seis inteiros) é criada apenas em um método, conforme mostrado abaixo:

```
private int[] createIntMessage(int device, int data) {
    int[] message = new int[6];
    message[0] = preamble; // Preâmbulo, considerando sempre 0xFF
    message[1] = device; // deviceID, direção (=4) ou aceleração (=3)
    message[2] = 1;// messageID, mensagem type: 0 = recebida, 1 = enviada
    message[3] = 1;    // tamanho do dado, neste caso sempre 1
    message[4] = data; // dado atual
    int checksum = (int)((message[0] + message[1] + message[2] + message[3] + message[4])%256);
    message[5] = checksum;     // checksum
    return message;
}
```

Exemplo 7: O método createIntMessage() da classe SensorCanvas, o que gera a mensagem de seis inteiros

O método `createIntMessage ()` é chamado no `createMessageData ()`, que controla e ajusta a aceleração e direção dos valores para estar dentro do limite 'correto'. Para este projeto foi escolhida de que existe uma certa área em torno do zero, onde apenas os valores zero são enviados para o carro. Se os valores estiverem fora desta "zona zero", eles são tratados adequadamente. Também é verificada no `createMessageData ()` se o carro está se movendo para a frente ou para trás.

A Figura 4 mostra o princípio da geração de valor. Observe que os valores origem na verdade não são zero. Foi esclarecido, ajustando os valores de deslocamento com teclas touchscreen, quais são os "valores zero", isto é, quando os pneus não estão indo para a frente ou para trás e quando eles estão apontando exatamente para a frente. Estes valores zero são enviados como 'zero', quando o dispositivo é controlado, por exemplo, encontrando-se no ecrã da tabela para cima.

(C) Copyright Nokia 2013. All rights reserved.

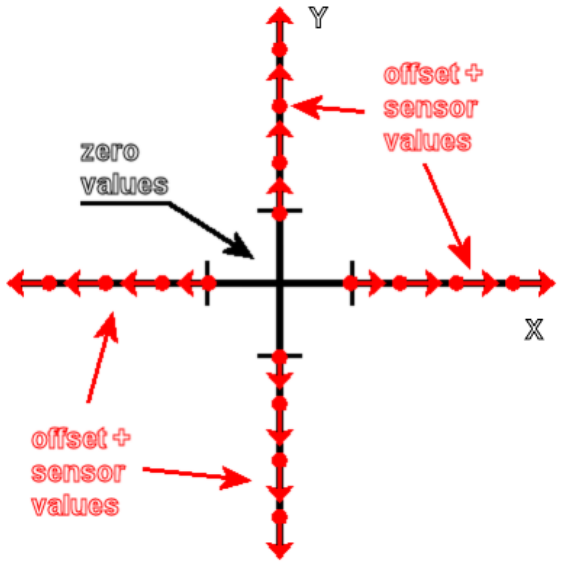

Figure 5: The principle of generating the accelerating and steering values

Abaixo está o código do método `createMessageData()` , que é usado para a criação de mensagens de dados. As variáveis acc_offset e dir_offset são os valores zero, quando o carro não está se movendo e os pneus são dirigidos exatamente em frente. O valor do offset é definir os limites de área, onde a mensagem zero é enviada. Velocidade para o retrocesso está definida para ser constante.

```
/**
  * Cria os dados da mensagem, tanto para acelerar e direção. Se z_int
  * Ou y_int (do sensor) está perto do valor "zero" (-offset <->
  * Offset), "Mensagem de zero" é criada. Caso contrário, o valor é baseado em
  * Combinação de deslocamento e o valor do sensor.
 */
private void createMessageData() {
 // Código para usar os valores x para acelerar
 if (x_int < 0) forward = true;
 else forward = false;
 if (x_int > -(2*offset) && x_int < (2*offset)) {
  if (!forward && back) {
   accelerating = acc_offset + 15;
  }
  else accelerating = acc_offset;
 }
 else {
  if (forward) accelerating = acc_offset + x_int/3; // Limited speed
  else accelerating = acc_offset + 15; // Para trás inclinando
  if (accelerating > 120) accelerating = 120;
  if (accelerating < -120) accelerating = -120;
 }
 if (y_int > -offset && y_int < offset) {
  steering = dir_offset;
 }
 else {
  steering = dir_offset + y_int;
  if (steering > 120) steering = 120;
  if (steering < -120) steering = -120;
 }
 dir_int_message = createIntMessage(STEERING, steering);
 acc_int_message = createIntMessage(ACCELERATOR, accelerating);
}
```

Exemplo 8: O método createMessageData() na classe SensorCanvas, que cria os dados da mensagem e chama o createIntMessage() para realmente criar as mensagens.

Há planos futuros para melhorar a velocidade e o controlo da direção. Atualmente, parece que a tentativa de ir para a frente e dirigir ao mesmo tempo não funciona muito bem. Também seria bom ter um melhor controle de ajustar a velocidade máxima. Uma ideia é usar a melhoria de valores não-linear para a direção (e talvez para a aceleração), que poderiam substituir a solução atual do uso de deslocamentos. (Valores não-lineares também foram usados no projeto ShakerRacer [1].)

(C) Copyright Nokia 2013. All rights reserved.

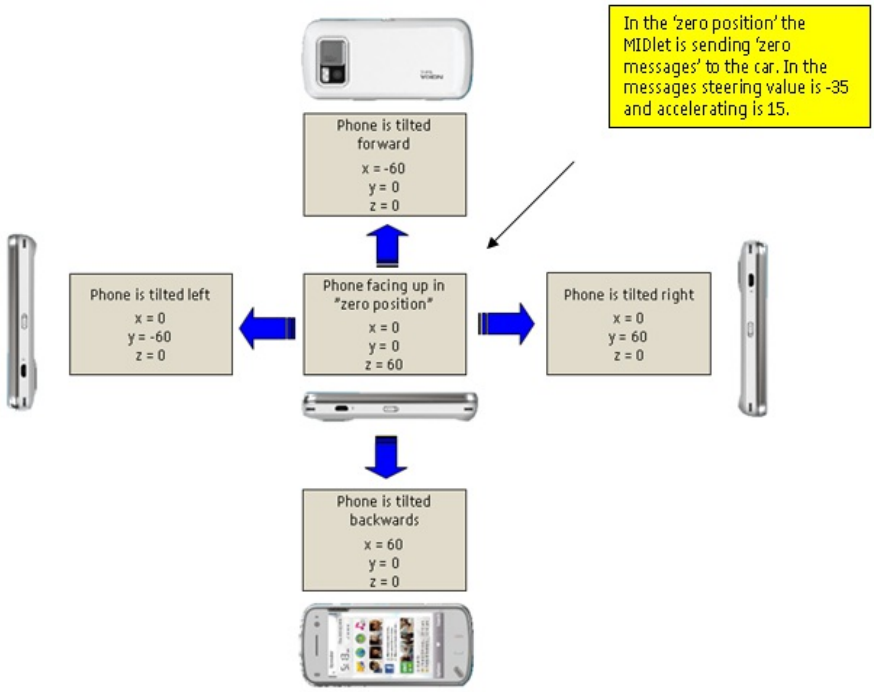

Figura 6: Aproximadamente o valor máximo do sensor de acelerômetro, quando o dispositivo Nokia N97 é inclinado

A Figura 6 mostra os valores máximos aproximados do sensor quando o aparelho Nokia N97 é inclinado. Note que os valores são diferentes o tempo todo, e os valores não são exatamente os mostrados na figura. Ao testar e ajustar os valores de zero a mensagem de que não foi encontrado quando o carro estava em movimento e os pneus estavam apontando diretamente para a frente, o valor de direcção deve ser cerca de -35 e acelerando o valor deve ser cerca de 15. Não se sabe por que esses valores estão tão longe de zero. Os valores máximos semelhantes para o carro variam caso a caso e só pode ser determinada através de testes. Para este efeito, existem quatro botões na tela MIDlet MiniController. Os deslocamentos de orientação e aceleração podem ser ajustados usando os botões.

Quando os valores zero são encontrados, é hora de procurar adequados valores máximos de condução e acelerando para frente e para trás.

A tela principal MIDlet (`SensorCanvas`) é mostrada abaixo. A exibição da tela consiste no seguinte:

- Os valores do sensor real (int x, int y, int valores z);
- O volante, o acelerar e os valores enviados para o carro;
- Os valores de deslocamento para a direção e aceleração;
- Indicador de vermelho/amarelo mostrando graficamente o sensor de movimento;
- Touchscreen botões para iniciar e parar (= criar a conexão BT e fechá-lo);
- Touchscreen botões para ajustar os deslocamentos (Acc, Acc + Dir-Dir +);
- Touchscreen botão para sair do MIDlet.

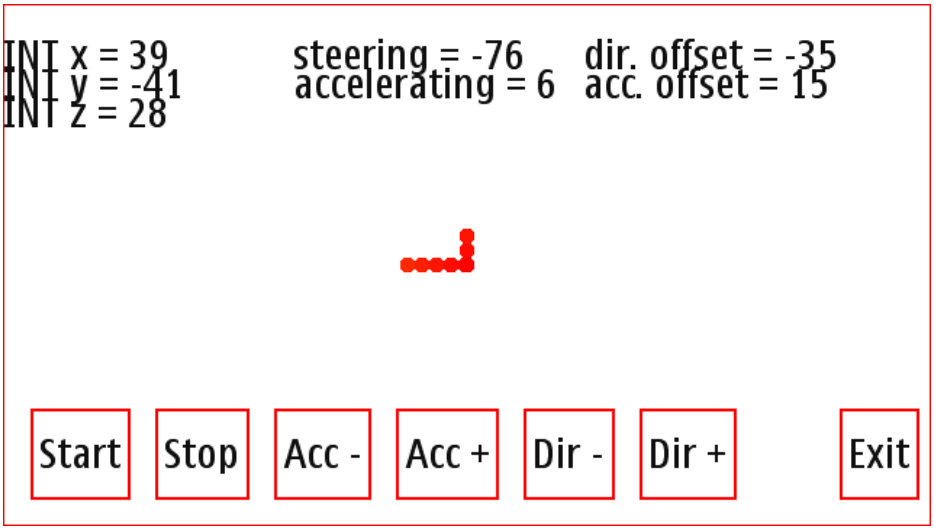

Figure 7: SensorCanvas screenshot

Na `SensorCanvas` (classe que implementa a interface Runnable) as mensagens são escritas para o fluxo de saída:

```
public void run() {
 while (!isStopped) {
  if (acc_int_message == null) acc_int_message =
          createIntMessage(ACCELERATOR, acc_offset);
  if (dir_int_message == null) dir_int_message =
          createIntMessage(STEERING, dir_offset);
  try {
   for (int i = 0; i < acc_int_message.length; i++) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
    dataOutputStream.write(acc_int_message[i]);
   }
   dataOutputStream.flush();
   for (int i = 0; i < dir_int_message.length; i++) {
    dataOutputStream.write(dir_int_message[i]);
   }
   dataOutputStream.flush();
  } catch (IOException ioe) {
   ioe.printStackTrace();
  }
  try {
   Thread.sleep(DELAY); // Atraso de 50 ms
  } catch (InterruptedException ie){}
 }
}
```

Exemplo 9: O método run() na classe SensorCanvas, que se encarrega de enviar as mensagens para controlar o carro

S40Controller MIDlet para o Nokia 6260 slide

Outra forma interessante de controlar o carro de RC é a utilização do dispositivo Nokia 6260 slide, porque ele tem uma tecla-joystick sensível. É possível obter os valores x e y, pressionando as bordas da tecla, e mesmo de tal forma que o valor depende do poder de pressão. Obtendo os valores é muito fácil simplesmente importar uma classe adicional da API Nokia UI: `com.nokia.mid.ui.JoystickEventListener` . Note que essa classe é um recurso opcional no Nokia UI API, e dependendo das capacidades do dispositivo pode haver implementações que não suportam eventos joystick. Atualmente (maio 2009), apenas o aparelho Nokia 6260 slide oferece suporte a esse recurso. O `System.getProperty("com.nokia.mid.ui.joystick_event")` vai retornar "true" se este recurso está disponível no dispositivo.

A classe `JoystickEventListener` tem apenas um método, `joystickEvent(int x, y)` , que é chamado quando a tecla Joystick envia um evento. Os eventos principais são em relação à sua posição inicial (0, 0), e os valores mínimo e máximo de x e y são -127 e 127. Como exemplo, suponha que o evento (5, 5) foi recebido. Significa que o joystick esta na diagonal esquerda e para cima. Agora, se o evento (-25, 25) é enviado, significa que o joystick está ainda no mesmo sentido de (-5, 5), no entanto, mais força foi aplicada.

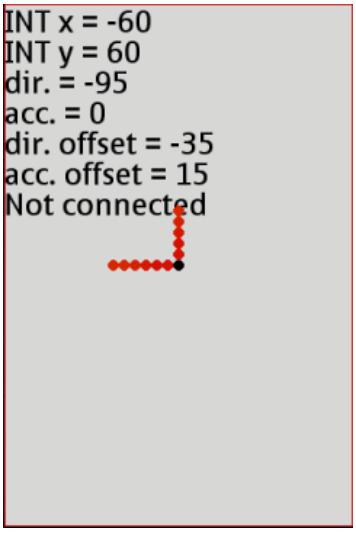

Figura 8: A tela do MIDlet S40Controller no SDK S40 6ª Edição

O S40Controller parte do MIDlet um monte de código com a MIDlet MiniController. É realmente muito menor, pois usando o `JoystickEventListener` foi tão simples. Ele também tem Bluetooth similares relacionados com as classes, que não são normalmente utilizados.

As partes mais importantes do código são listados abaixo:

```
    public void joystickEvent(int x, int y) {
        x_int = x;
        y_int = y;
        createMessageData();
        repaint();
    }
```

Exemplo 10: O método joystickEvent() na classe ControlCanvas, que lida com o joystick eventos-chave

```
    private void createMessageData() {
        if (y_int > -offset && y_int < offset) {
            accelerating = acc_offset;
        }
        else {
            accelerating = acc_offset - y_int/4;
            if (accelerating > 35) accelerating = 35;
            if (accelerating < 0) accelerating = 0;
        }
        if (x_int > -offset && x_int < offset) {
            steering = dir_offset;
        }
        else {
            steering = dir_offset + x_int;
            if (steering > 30) steering = 30;
            if (steering < -95) steering = -95;
        }
        dir_int_message = createIntMessage(STEERING, steering);
        acc_int_message = createIntMessage(ACCELERATOR, accelerating);
    }
```


(C) Copyright Nokia 2013. All rights reserved.

Exemplo 11: O método createMessageData() da classe ControlCanvas, que cria as mensagens do volante e do acelerador.

O método `createMessageData()` é tradado de maneira semelhante como no MIDlet MiniController. Há um certo deslocamento, quando 'nenhuma mensagens' são enviadas para o carro e os valores fora dessa área são realmente tratados como valores do controlador. Os valores máximos a serem enviados para o carro tem sido procurados por testes. Eles variam em combinações de hardware de outros (automóveis, outros telefones móveis, e outros módulos BT), o teste tão completo é sempre obrigatório. As variáveis acc_offset e dir_offset são os valores zero, quando o carro não está se movendo e os pneus estão dirigidos exatamente para a frente.

Os planos futuros para o projeto

O projeto MiniController foi realizado em um período de tempo limitado. Havia algumas questões que não foram abordadas no momento em que podem ser úteis e interessantes para dar uma olhada. Por exemplo, no projeto ShakerRacer, direção não-linear foi implementada para facilitar a movimentação diretamente para a frente. Um tipo similar de aceleração não-linear pode ser boa para este projeto.

Atualmente, o MIDlet MiniController simplesmente uma certa área em torno de zero, quando apenas os valores "zero" são enviados para o carro como mensagem de volante e acelerar.

Deve também ser possível receber mensagens a partir do carro-valores de rotações do motor por minuto e velocidade da corrente, pelo menos. Recebendo mensagens do carro exige alguns Bluetooth adicionais relacionados ao código, mas caso contrário deve ser fácil de implementar.

Outra ideia interessante seria para gravar as mensagens enviadas e tentar redrive a mesma rota novamente sem ninguém controlar o carro. Além disso, deverá ser possível utilizar a rede sem fios em vez de Bluetooth para o alongamento da distância máxima entre o telefone móvel e controlar o carro.

Contatos com o criador do projeto

O projeto foi realizado principalmente por uma pessoa, Jarmo Lahtinen, engenheiro-chefe da Nokia Devices R & D da unidade em organizações S60 Java. Seu trabalho concentra-se em diferentes tipos de projetos relacionados com ferramentas de MIDlet de desenvolvimento, documentação de desenvolvimento e cooperação com outras organizações dentro da Nokia, por exemplo, do Fórum Nokia.

Para enviar comentários, ou perguntas sobre o projeto, use o seguinte endereço de e-mail: jarmlaht@ovi.com. Você também poderá utilizar o endereço de e-mail do tradutor deste artigo para maiores informações: Maicon Herverton

Referências

- [1] ShakerRacer: Drive Your Car with Your Phone (<broken link to> Technical_Background_of_ShakerRacer.pdf)
- [2] JSR 256: Mobile Sensor API Specification (http://www.jcp.org/en/jsr/detail?id=256)
- [3] How To Get Information about Sensors in Java ME (How to get information about sensors in Java ME)
- [4] Video of Using Mobile Sensor API for Controlling RC Car in Nokia Developer Summit 2009 in Monaco (http://share.ovi.com/media/jarmlaht.public/jarmlaht.10005)
- [5] Developing Java ME Applications with Force-Sensitive Joystick and Force-Sensitive Joystick API

Veja também

- MiniController_project_1_0.pdf This wiki article as downloadable pdf file
- MiniController.zip The MiniController MIDlet as NetBeans project (For Nokia N97 and other S60 5th Ed. devices supporting JSR-256)
- S40Controller.zip The S40Controller MIDlet as NetBeans project (For Nokia 6260 Slide)
- Mobile Sensor API (JSR-256) javadoc documentation and RI Binary
- How to get information about sensors in Java ME
- How to get accelerator sensor values in Java ME
- How to use sensors in Java ME
- Mobile Sensor API (JSR-256) beta add-on for Nokia 5800 XpressMusic
- Nokia N97 SDK 0.5

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.