Interface Homem-máquina Android para Sistemas Embarcados com Recursos Restritos

Silotti, Natyelle Oliveira[1]; Iaione, Fábio[1]; Minakawa, Riccieli Kendy Zan[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul
79070-900 – Campo Grande – MS – Brasil

fabio.iaione@ufms.br,{naty.silotti,riccieli.minakawa}@gmail.com

Abstract. *Embedded systems (ESs) are present in many types of equipment, which need to have low energy consumption and or low cost. This results in limited human-machine interfaces (HMI), constructed by LEDs and switches, and no internet connection. Given this limitation, the goal of this work was to develop a configurable HMI for Android smartphones. The developed HMI (protocol + Android application) allows users to interact with any ES through the smartphone screen and a Bluetooth connection. The HMI can interact with different types of equipment since the components displayed on the smartphone are defined by the ES.*

Resumo. *Os sistemas embarcados (SEs) estão presentes em uma infinidade de equipamentos, sendo que muitos necessitam apresentar baixo consumo de energia e/ou baixo custo. Isso leva à interfaces homem-máquina (IHM) muito limitadas, compostas por LEDs e teclas, e sem conexão com a internet. Dada essa limitação, o objetivo desse trabalho foi desenvolver uma IHM para smartphones Android configurável. A IHM desenvolvida (protocolo + aplicativo Android) permite ao usuário interagir com qualquer SE por meio da tela do smartphone e uma conexão Bluetooth. Dado que os componentes visualizados no smartphone são definidos pelo próprio SE, por meio de comandos via Bluetooth, a IHM pode ser utilizada para interagir com diferentes equipamentos.*

1. Introdução

Os avanços tecnológicos no mercado de smartphones tem mudado a forma de interagir com a tecnologia. Hoje em dia um smartphone pode perfeitamente substituir o uso de um computador para realizar grande parte de tarefas comuns, como por exemplo: navegar na internet, responder e-mails, organizar compromissos, jogar on-line, pagar contas, acessar redes sociais, e outras. Além de possuir outros recursos próprios, como por exemplo, os serviços de localização, câmera fotográfica embutida e comandos de voz para realizar pesquisas na internet, a experiência do usuário com o smartphone inclui uma grande variedade de entretenimento, informação e outras tarefas específicas que podem facilitar o dia a dia [Feijó et al 2013].

Desde o primeiro smartphone lançado em 1993 [Sager 2012], o número desses dispositivos cresceu, chegando em 2015 à 80% de usuários em países desenvolvidos [Poushter 2016]. O BlackBerry foi o primeiro smartphone com acesso à internet e câmera, que surgiu no final da década de 1990. Só no ano de 2007 apareceram os primeiros

smartphones para o mercado de "massa", com o lançamento do iOS da Apple. No ano seguinte, o primeiro celular utilizando o sistema operacional Android foi lançado. Presente em smartphones e *tablets* de diferentes marcas, o Android é o sistema operacional mais utilizado no mundo [[Tristan 2013], sendo encontrado atualmente em 2,8 bilhões de dispositivos móveis, o que corresponde a aproximadamente 75 % do mercado [Curry 2021]. Sua loja de aplicativos conta com mais de 2,9 milhões de aplicativos, que juntos somam 108 bilhões de downloads [Curry 2021]. Atualmente, todos os smartphones possuem capacidade de comunicação por meio do protocolo *Bluetooth*. O *Bluetooth* é um protocolo de comunicação sem fios, caracterizado pelo baixo consumo de energia e curto alcance (*Personal Area Network - PAN*), tendo uma versão específica para baixo consumo (*Bluetooth Low Energy - BLE*), com o propósito de eliminar as conexões físicas entre dispositivos de diferentes fabricantes. A grande vantagem dessa tecnologia sobre outros protocolos sem fios é a facilidade de conexão [Tosi et al 2017].

Segundo levantamento feito pela Intel, cerca de 80% dos microprocessadores fabricados no mundo são utilizados em sistemas embarcados [Barbiero and Hexsel 2006]. Os sistemas embarcados possuem aplicações em diversos segmentos, tais como, eletrônica de consumo, automação, robótica, comunicação, segurança, medicina, e outros, influenciando o modo de vida, trabalho e diversão das pessoas. Em geral, esses sistemas possuem em comum algumas restrições, como por exemplo, consumo de energia reduzido, utilização mínima de memória, pequenas dimensões e baixo custo [Borges 2011]. Nas últimas décadas, com os avanços na miniaturização dos dispositivos microprocessadores/microcontroladores, redução do consumo de energia (*ultra low power devices*), e advento da colheita de energia (*energy harvesting*) [Selvan and Ali 2016], surgiu a possibilidade de "embutir" sistemas embarcados nos mais diferentes tipos de objetos. Cabe observar que dada as grandes restrições de consumo de energia, custos e dimensões nesses sistemas, normalmente é inviável a utilização de um *display* de cristal líquido, teclado e mesmo conexão com a internet. Além disso, mesmo que as restrições permitam uma conexão com a internet, esta muitas vezes não é imprescindível para o funcionamento do sistema embarcado, trazendo vulnerabilidades desnecessárias e podendo causar perdas financeiras, como reportado recentemente por [BBC 2021] e [Telford et al 2021]. Em muitas aplicações, o sistema embarcado não precisa coletar e transmitir dados continuamente, necessitando ser configurado uma única vez, no momento de sua instalação, o que não justifica uma conexão com a internet. Exemplos de sistemas embarcados desse tipo são controladores para automação: de irrigação em jardins, de tratamento de água de piscinas, de controle de temperatura, e outros, que após sua configuração inicial automatizam tarefas repetitivas, sem nenhuma necessidade de enviarem dados para algum servidor e de serem reconfigurados remotamente [Rain Bird 2020] [Bn-Link 2020] [Full Gauge 2020].

Existem algumas ferramentas para implementação de IHMs com sistemas embarcados, como por exemplo, o Megunolink [MegunoLink 2021], o MakerPlot [MakerPlot 2021], Cayenne [myDevices 2021], TagoIO [Tago 2021], e outras. Entretanto, essas plataformas (Tabela 1) ou são para computadores desktops, ou exigem que o sistema embarcado esteja conectado à internet, ou estão restritas à plataforma Arduino, ou exigem o pagamento de taxas mensais, ou exigem que a IHM seja desenvolvida nos "dois lados", firmware do sistema embarcado e aplicativo *web* de configuração da plataforma.

Dado o exposto, o objetivo desse trabalho foi desenvolver um protocolo e um apli-

Tabela 1. Sistemas semelhantes ao desenvolvido. Os gratuitos são para uso não comercial.

	Dispositivo da IHM	Conexão	SE	Custo U$	Construção da IHM
MegunoLink	desktop (apl Windows)	USB, XBee e internet	Arduino	39 /licença	apl. editor
MakerPlot	desktop (apl Windows)	USB e internet	qualquer	59 /licença	apl editor ou comando SE
Cayenne	smartphone (apl) desktop (browser)	internet	Raspberry Arduino	Gratuito	editor online
TagoIO	smartphone (apl) desktop (browser)	internet, BLE e Zigbee	qualquer	Gratuito 49/mês 199/mês	editor online

cativo para Android que permitam criar uma IHM específica, para ser usada com qualquer sistema embarcado, independente da aplicação e do microprocessador/microcontrolador utilizado. Como a IHM é configurada pelo próprio sistema embarcado via *Bluetooth*, só os componentes necessários para o sistema embarcado conectado estarão visíveis, proporcionando ao usuário IHMs diferentes, específicas para cada aplicação diferente.

2. Metodologia

Os componentes implementados na IHM desenvolvida para Android, chamada de Mercúrio, são apresentados nessa seção, por meio de três subseções. Na primeira é apresentado o processo de inicialização, na segunda, os componentes de entrada da IHM, e na terceira, os componentes de saída da IHM. Nessas seções são descritas as funcionalidades, os comandos de configuração correspondentes, e exemplos desses comandos. O *layout* do Mercúrio foi inicialmente desenvolvido com o auxílio da ferramenta Moqups (https://app.moqups.com).

2.1. Inicialização

Ao iniciar o Mercúrio, caso o módulo *Bluetooth* do aparelho celular esteja desativado, uma mensagem solicitando que ele seja ativado é exibida no aplicativo. Para conectar-se a um dispositivo, deve-se localizar o ícone *Bluetooth* na página inicial do aplicativo, e clicar no botão para iniciar a busca por dispositivos próximos. Logo após, uma lista contendo os dispositivos disponíveis é exibida

Após o dispositivo ter sido selecionado, uma caixa de diálogo indicará se foi possível estabelecer conexão com o mesmo. Uma vez conectado, o Mercúrio envia ao sistema embarcado a mensagem `CONFIG#OK` que indica que a conexão foi realizada com sucesso. Assim, o sistema embarcado sabe que pode enviar as mensagens de configuração para o Mercúrio, indicando quais componentes da interface gráfica ele utilizará.

Após as mensagens de configuração serem recebidas pelo Mercúrio, o "fragmento" que contém a lista de dispositivos próximos é substituído pelo "fragmento" que contém os componentes da interface, conforme configurado pelo sistema embarcado. Todos os componentes são carregados e apresentados em um único "fragmento" de tela.

2.2. Componentes de Entrada da IHM

Os componentes de entrada da IHM correspondem aos componentes que funcionam como entrada de dados para o sistema embarcado, ou seja, o usuário deve realizar alguma interação com o Mercúrio, no smartphone, para que o sistema embarcado receba esses dados e realize alguma ação.

A Tabela 2 mostra os comandos de configuração para cada um dos sete componentes de entrada implementados. Os comandos de configuração são enviados pelo sistema embarcado logo após o processo de inicialização descrito anteriormente. Dessa forma, a IHM Mercúrio mostra na tela do smartphone apenas os componentes definidos pelos comandos de configuração, recebidos do sistema embarcado. Cabe observar que o caractere "#" tem a finalidade de separar as diferentes seções contidas nos comandos de configuração e mensagens. O caractere "," tem a finalidade de separar diferentes *labels*, opções, ou valores, dentro de uma seção de alguns comandos. Já o caractere ";" indica o final de todos os comandos e mensagens.

Tabela 2. Comandos de configuração, com exemplos, para os componentes de entrada de dados da IHM Mercúrio.

Componente de entrada	Comando de configuração Sistema Embarcado $\rightarrow$ Mercúrio
Chave liga/desliga	SWITCH#<$quantity$>#<$label1, label2, ..$>#<$option1, option2, ..$>; Ex.: SWITCH#2#RED LED,Timer#OFF,ON,MIN,SEC;
Entrada de texto	TEXT#<$label$>; Ex.: TEXT#Blue LED Control;
Caixa de seleção	COMBOBOX#<$label$>#<$number\ items$>#<$item1, item2, ..$>; Ex.: COMBOBOX#LED RGB#3#RED,GREEN,PURPLE;
Entrada de números	TEXT2#<$title$>#<min, max>; Ex.: TEXT2#SERVOMOTOR2#0,180;
Entrada de horário	BUTTON#<$title$>; Ex.: BUTTON#SELECTHOUR;
Entrada de data	BUTTON1#<$title$>; Ex.: BUTTON1#SELECTDATE;
Barra de seleção	BAR2#<$title$>#<min, max>; Ex.: BAR2#SERVOMOTOR#0,180;

Quando o usuário interage com os componentes de entrada no smartphone, gerando eventos de atualização, mensagens são enviadas para o sistema embarcado com dados atualizados, indicando as escolhas/ações do usuário. A Tabela 3 mostra esses eventos e as respectivas mensagens.

Por exemplo, o componente chave liga/desliga permite criar até três *Switch Buttons* que podem ser utilizados como uma chave liga/desliga, ou para alternar uma unidade de medida entre duas opções possíveis (min ou seg), por exemplo. Para criar *Switch Buttons* no Mercúrio, o sistema embarcado deve enviar o seguinte comando de configuração: `SWITCH#<quantity>#<label1,label2,..>#<option1,option2,..>;` (Tabela 2). Onde *Switch* indica o tipo de componente (*Switch Button*) e $< quantity >$ indica a quantidade utilizada, podendo variar de um à três. $< label1, label2, .. >$ é uma lista

Tabela 3. Eventos de atualização e respectivas mensagens enviadas pela IHM Mercúrio ao Sistema Embarcado, com exemplos, para os componentes de entrada de dados.

Componente de entrada	Evento de atualização (ação na tela do smartphone)	Mensagem de atualização Mercúrio→ Sist. Emb.
Chave liga/desliga	Mudar posição da chave L/D.	SWITCH<$number$>#<$state$>; Ex.: SWITCH1#ON;
Entrada de texto	Digitar texto na caixa de texto e clicar no botão SEND.	TEXT#<$typed\ text$>; Ex.: TEXT#Turn LED Blue on;
Caixa de seleção	Selecionar um item da lista da caixa de seleção.	SELECTED#<$item$>; Ex.: SELECTED#GREEN;
Entrada de números	Digitar valor na caixa e clicar no botão OK.	UPDATE_VALUE#<$value$>; Ex.: UPDATE_VALUE#110;
Entrada de horário	Clicar botão <$title$>, selecionar hora no popup e clicar botão OK.	UPDATE_HOUR#<$hh:mm$>; Ex.: UPDATE_HOUR#16:11;
Entrada de data	Clicar botão <$title$>, selecionar data no popup e clicar botão OK.	UPDATE_DATA#<$dd.mm.aa$>; Ex.: UPDATE_DATA#14.02.22;
Barra de seleção	Mover o botão da barra de seleção para esquerda ou direita.	UPDATE_BAR2#<$value$>; Ex.: UPDATE_BAR2#123;

de rótulos separados por vírgula, com nomes intuitivos para cada *Switch Button* (Ex.: Lâmpada, Motor). $< option1, option2, .. >$ é uma lista de opções onde cada par corresponde aos dois rótulos que aparecerão ao lado esquerdo e direito dos *Switch Buttons*. Exemplo de comando para configurar dois *Switches Buttons*: `SWITCH#2#RED LED,Timer#OFF,ON,MIN,SEC;`. Ao alterar o estado de um *Switch Button*, o Mercúrio envia a seguinte mensagem: `SWITCH<number>#<state>;` (Tabela 3). Onde $< number >$ identifica o *Switch Button* que foi modificado na IHM Mercúrio. $< state >$ pode assumir o valor ON ou OFF, onde OFF indica que a chave está posicionada no lado esquerdo e ON no lado direito (Ex.: `SWITCH1#ON; e SWITCH2#OFF;`).

2.3. Componentes de Saída da IHM

Os componentes de saída da IHM correspondem aos componentes que funcionam como saída de dados para o sistema embarcado, ou seja, mostram para o usuário, na tela do smartphone, os dados enviados pelo sistema embarcado. A Tabela 4 mostra os comandos de configuração para cada um dos três componentes de saída implementados. Os comandos de configuração são enviados pelo sistema embarcado logo após o processo de inicialização descrito anteriormente. Dessa forma, a IHM Mercúrio mostra na tela do smartphone apenas os componentes definidos pelos comandos de configuração, recebidos do sistema embarcado.

Quando o usuário interage com os componentes de saída no smartphone, gerando eventos de atualização, mensagens são enviadas para o sistema embarcado solicitando dados atualizados, que são enviados pelo mesmo. A Tabela 5 mostra esses eventos e as respectivas mensagens.

Por exemplo, o componente de saída de texto (*Display Text*) é utilizado para exibir na IHM texto enviado pelo sistema embarcado. Pode ser utilizado para apresentar dados

Tabela 4. Comandos de configuração, com exemplos, para os componentes de saída de dados da IHM Mercúrio.

Componente de saída	Comando de configuração Sistema Embarcado → Mercúrio
Saída de texto	DISPLAY#<$label$>#<$text$>; Ex.: DISPLAY#Temperatura: #27 oC;
Barra de progresso	BAR#<$label$>#<min, max>; Ex.: BAR#SERVO MOTOR#0,255;
Gráfico	BUTTON2#<$title$>; Ex.: BUTTON2#LINEGRAPH;

Tabela 5. Eventos de atualização e respectivas mensagens enviadas pela IHM Mercúrio ao Sistema Embarcado, e mensagens de resposta, com exemplos, para os componentes de saída de dados.

Componente de saída	Evento de atualização (ação no smartphone)	Mens. de atualização Mercúrio→Sist. Emb.	Mensagem de atualização Sist. Emb. → Mercúrio
Saída de texto	Clicar no botão UPDATE próximo à caixa de texto.	UPDATE_DISPLAY;	UPDATE_DISPLAY#<$text$>; Ex.: UPDATE_DISPLAY#23 C;
Barra de progresso	Clicar no botão UPDATE próximo à barra de progresso.	UPDATE_BAR;	UPDATE_BAR#<$value$>; Ex.: UPDATE_BAR#114;
Gráfico	Clicar no botão <$title$>, que mostra o gráfico.	UPDATE_GRAFICO;	UPDATE_GRAFICO#<$value$>; enviada periodicamente para atualizar o gráfico.
Gráfico	Clicar no botão BACK, que fecha o gráfico.	STOP_GRAFICO;	Interrompe envio das mensagens UPDATE_GRAFICO#<$value$>;

de um sensor, ou ainda para monitorar por meio de textos curtos a execução do código do sistema embarcado, tendo assim sua funcionalidade semelhante ao Terminal Serial da IDE do Arduino. Para configurar sua utilização na IHM, deve-se utilizar o seguinte comado: `DISPLAY#<label>#<text>;` (Tabela 4). Onde DISPLAY indica que será utilizada uma *TextView* na IHM para exibir mensagens. Não existe uma limitação do número de caracteres, mas é recomendável que o texto enviado tenha no máximo 25 caracteres, por questões estéticas na visualização do texto na IHM. Exemplo do comando para configurar um *Display Text*: `DISPLAY#Temperatura: #27 oC;`. Para que o texto seja atualizado, deve-se clicar no botão ***UPDATE***, assim o aplicativo envia para o sistema embarcado a mensagem `UPDATE_DISPLAY` (Tabela 5), e este deve enviar de volta o novo texto para ser atualizado na tela da IHM Mercúrio, seguindo o formato: `UPDATE_DISPLAY#<text>`. Onde $< text >$ indica a nova mensagem que deve ser exibida na IHM. Exemplo do comando para atualizar o *Display Text*: `UPDATE_DISPLAY#23 oC;`.

Para desenvolver a IHM Mercúrio foi utilizado o Software Android Studio 2.2.2 [Android 2020], instalado em um notebook Dell Inspiron com sistema operacional Windows 10 64-bits, processador intel core i5 e 8,00 GB de RAM. Utilizou-se o compilador

compileSdkVersion 25 para compilar e gerar o *apk* do *app*. A versão mínima do Android em que o Mercúrio roda é a *Lollipop* (API 21). A versão em que o sistema foi desenvolvido é o Android 6.0. Além dessas duas versões, o aplicativo foi testado com as versões Android 7.0 e Android 11. Foram utilizados nos testes um *smartphone* Sony, modelo Xperia Z3+, e dois Samsung, modelos Galaxy E5 e Galaxy A50.

3. Resultados e Discussões

Para testar a IHM Mercúrio, um sistema embarcado experimental foi montado em uma matriz de contatos.

3.1. Circuito

O circuito do sistema embarcado montado pode ser observado na Figura 1, sendo constituído pelos seguintes componentes: Arduino Mega 2560, módulo *Bluetooth* HC-06 [Raajkumar 2021], três LEDs simples e um RGB, sensor LDR, potenciômetro, servomotor, além de resistores.

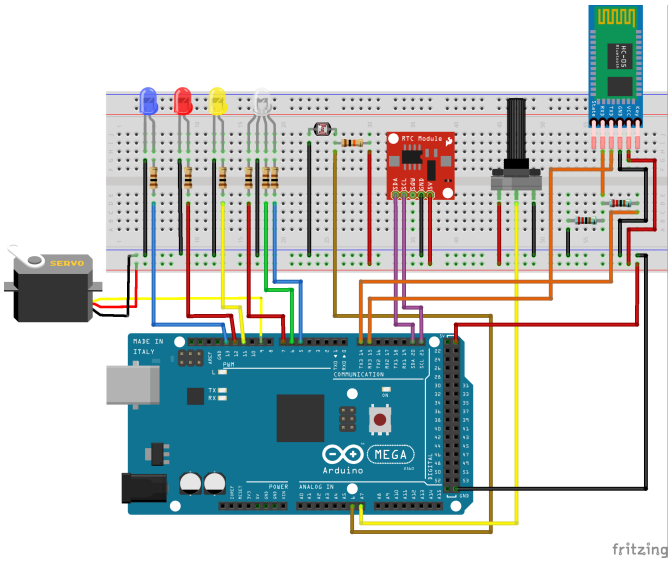

Figura 1. Circuito do sistema embarcado montado para testar a IHM Mercúrio. Elaborado por meio da ferramenta *Fritzing* (http://fritzing.org).

3.2. Firmware do Arduino

O firmware foi desenvolvido na IDE Arduino e sua parte inicial contem a declaração das variáveis globais e a função *setup()*, na qual realiza-se a configuração dos pinos de entrada e dos pinos de saída, bem como das portas seriais. Na porta *Serial3* (pinos RX3/TX3) está conectado o módulo *Bluetooth* (9600 bps) e na porta *Serial0* é realizada a comunicação com o Monitor Serial (9600 bps) da IDE Arduino, por meio da porta serial virtual-USB, para fins de testes e "depuração". Nota-se que o pino TX3 do Arduino Mega 2560 opera em 5V e o pino RX do módulo *Bluetooth* opera em 3,3V, portanto foi utilizado um divisor de tensão (680 ohms e 1,3 kohms).

No Quadro 1 é mostrado o código para receber uma mensagem enviada pela IHM Mercúrio, que recebe os caracteres até encontrar o caractere ";", e separa o comando (cmd) e o rótulo (tag) contidos na mensagem.

```
void loop() {
  char c;
  boolean proc_str = false;

  if(Serial3.available() > 0) {
   c = char(Serial3.read());
   data += c;
   if(c == ';') proc_str = true;
  }

  /*Processa a string recebida da IHM Mercúrio*/
  if (proc_str == true) {
    proc_str = false;
    Serial.print("String recebida: ");
    Serial.println(data);
    /*Separa comando e tag*/
    String cmd = data.substring(0, data.indexOf('#')) ;
    String tag = data.substring(data.indexOf('#') + 1, data.indexOf(";"));

    /*Caso o Mercúrio tenha se conectado ao sistema, o sistema recebe CONFIG#OK
    *confirmando a conexão e solicitando as mensagens de configuração
    Essa linha de código limpa possíveis lixos de cabeçalho ao conectar-se*/
    if (cmd.substring(data.indexOf("CONFIG")) == "CONFIG") {
      cmd = "CONFIG";
      }
    data = "";
```

Quadro 1. Leitura dos dados na porta *Serial3* e separação do comando e rótulo da mensagem recebida pelo Arduino

No Quadro 2, pode-se observar o envio dos comandos para a configuração da IHM no Mercúrio, após a recepção de uma mensagem `CONFIG#OK;`, que ocorre quando uma conexão é estabelecida entre o Mercúrio e o sistema embarcado. Com as mensagens de configuração enviadas para o Mercúrio, os componentes da IHM estarão disponíveis para o usuário, no caso do exemplo: duas chaves liga/desliga, uma caixa de entrada de texto, uma caixa de seleção, uma barra de progresso, uma barra de seleção, uma caixa de saída de texto, uma caixa de entrada de números, um seletor de horário, um seletor de data, e um gráfico.

```
 /*Verifica se é uma solicitação de configuração, caso seja,
  * o Arduino envia comandos de confiugração para o Mercúrio*/
 if (cmd == "CONFIG") {
   if (tag == "OK" ) {
     Serial3.print("SWITCH#2#RED LED,YELLOW RED#OFF,ON ,OFF,ON;");
     Serial3.print("TEXT#BLUE LED;");
     Serial3.print("COMBOBOX#LED RGB#4#OFF,RED,GREEN,BLUE;");
     Serial3.print("BAR#POTENTIOMETER #0,255;");
     Serial3.print("BAR2#SERVOMOTOR #0,180;");
     Serial3.print("DISPLAY#LUMINOSITY: #0%;");
     Serial3.print("TEXT2#SERVOMOTOR2#0,180;");
     Serial3.print("HORA#SELECTHOUR;");
     Serial3.print("DATA#SELECTDATE;");
     Serial3.print("GRAFICO#LINEGRAPH;");
     }
 }
```

Quadro 2. Código para enviar as mensagens de configuração à IHM Mercúrio.

O Quadro 3 mostra os códigos: para atualizar o estado dos dois LEDs (vermelho e amarelo), conforme a manipulação das chaves na IHM (esquerda); para atualizar o estado do LED azul, conforme o texto inserido na caixa de texto de entrada (centro); e para atualizar o estado do LED RGB, conforme o item escolhido na caixa de seleção da IHM (direita).

O Quadro 4 (esquerda) mostra o código para atualizar a barra de progresso, que ocorre quando o usuário pressiona o botão *UPDATE* e o Mercúrio envia a mensagem `UPDATE_BAR;` para o Arduino. Nesse firmware de teste, o Arduino responde com o valor lido em uma de suas entradas analógicas, na qual está conectado um potenciômetro. O comando `Serial3.print("BAR2#SERVOMOTOR #0,180;");` (Quadro 2) cria na IHM uma barra para seleção de um valor entre 0 e 180, que é enviado após a ação do usuário sobre a barra. O código para receber o valor e aplicá-lo ao servomotor pode ser observado no Quadro 4 (centro). O Quadro 4 (direita) mostra o código para atualizar o texto mostrado na IHM, que corresponde a leitura de intensidade luminosa (0 a 100%) por meio de um LDR.

```
else if (cmd == "SWITCH1") {
  if (tag == "ON" )
    digitalWrite(RED, HIGH);
  if (tag == "OFF")
    digitalWrite(RED, LOW);
}

else if (cmd == "SWITCH2") {
  if (tag == "ON" )
    digitalWrite(YELLOW, HIGH);
  if (tag == "OFF")
    digitalWrite(YELLOW, LOW);
}
```

```
else if (cmd == "TEXT") {
  if (tag == "Turn on")
    digitalWrite(BLUE, HIGH);
  else if (tag == "Turn off")
    digitalWrite(BLUE, LOW);
}
```

```
else if (cmd == "SELECTED") {
  if (tag == "RED")
    setColor(255, 0, 0);
  else if (tag == "OFF")
    setColor(0, 0, 0);
  else if (tag == "BLUE")
    setColor(0, 0, 255);
  else if (tag == "GREEN")
    setColor(255, 255, 0);
}
```

Quadro 3. Código para controlar os LEDs vermelho e amarelo por meio das chaves On/Off (esquerda), para controlar o LED azul por meio da caixa de entrada de texto (centro), e para controlar o LED RGB por meio da caixa de seleção (direita).

```
else if (cmd == "UPDATE_BAR") {
  int pos = analogRead(Pot);
  pos = map(pos, 0, 1023, 0, 255);
  String SetRange = "";
  SetRange.concat("UPDATE_BAR#");
  SetRange.concat(pos);
  SetRange.concat(";");
  Serial3.print(SetRange);
  Serial.print("String enviada: ");
  Serial.println(SetRange);
}
```

```
else if (cmd == "UPDATE_BAR2") {
  int pos = tag.toInt();
  myservo.write(pos);
```

```
else if (cmd == "UPDATE_DISPLAY") {
  int luz = analogRead(LDR);
  luz = map(luz, 0, 1023, 0, 100);
  float porc = 100 / luz ;
  String displayText = "";
  displayText.concat("UPDATE_DISPLAY#");
  displayText.concat(porc);
  displayText.concat("%;");
  Serial3.print(displayText);
  Serial.print("String enviada: ");
  Serial.println(displayText);
}
```

Quadro 4. Código para atualização da barra de progresso (esquerda), para tratamento do valor inserido na barra de seleção (centro), e para atualizar o texto visualizado na IHM (direita).

O comando `Serial3.print("TEXT2#SERVOMOTOR 2#0,180;");` (Quadro 2) cria uma caixa para a entrada de valores numéricos entre 0 e 180. Ao clicar na caixa, aparece um teclado numérico para entrada do valor, que é enviado para o Arduino após o botão OK ser pressionado. O código para receber o valor e aplicá-lo ao servomotor pode ser observado no Quadro 5.

```
else if (cmd == "UPDATE_VALUE") {
   int pos = tag.toInt();
   myservo.write(pos);
```

Quadro 5. Código para tratamento do valor inserido na caixa de entrada de números.

O comando `Serial3.print("BUTTON#SELECT HOUR;");` (Quadro 2) cria um botão para entrada de horário. Ao clicar no botão, o *pop-up* do relógio se abre, onde a hora deve ser seleciona e o botão OK pressionado para enviá-la ao Arduino. Da mesma forma, o comando `Serial3.print("BUTTON1#SELECTDATE;");` (Quadro 2) cria um botão para entrada de data. Ao clicar no botão, o *pop-up* do calendário se abre, onde a data deve ser seleciona e o botão OK pressionado para enviá-la ao Arduino. Os códigos para receber o horário e a data podem ser observados no Quadro 6.

O comando `Serial3.print("BUTTON2#LINE GRAPH;");` (Quadro 2) cria um gráfico na IHM para visualização de dados enviados pelo Arduino em tempo real. Ao pressionar o botão LINEGRAPH, o gráfico é visualizado na IHM e um comando UPDATE_GRAFICO é enviado para o Arduino, fazendo a variável x receber *true* e a função ler() enviar periodicamente as leituras da tensão gerada pelo potenciômetro conectado ao

```
 else if (cmd == "UPDATE_HORA"){
   Serial.print("Hora recebida: ");
   Serial.println(tag);
  }

else if (cmd == "UPDATE_DATA"){
   Serial.print("Data recebida: ");
   Serial.println(tag);
  }
```

Quadro 6. Código para tratamento dos dados inseridos nos componentes de entrada de horário e data.

conversor A/D do Arduino (Quadro 7). Quando o usuário pressiona o botão BACK na IHM, o gráfico desaparece e o comando STOP_GRAFICO é enviado ao Arduino, fazendo a variável x receber *false*, interrompendo as leituras periódicas.

```
   else if(cmd == "UPDATE_GRAFICO"){
      x=true;
      }

   else if(cmd == "STOP_GRAFICO"){
      x=false;
      }

   }
ler();
} //Fim da função loop()


void ler(){
  if (x == true){
      // faz a leitura da entrada analogica:
      int pos = analogRead(Pot);
      // mapeia o resultado da entrada analogica dentro do intervalo de 0 a 255:
      pos = map(pos, 0, 1023, 0, 255);
      // envia o valor lido:
      String SetRange = "";
      SetRange.concat("UPDATE_GRAFICO#");
      SetRange.concat(pos);
      SetRange.concat(";");
      Serial3.print(SetRange);
      Serial.print("String enviada: ");
      Serial.println(SetRange);
      delay(100);
     }
}
```

Quadro 7. Código para tratamento do comando GRAFICO, enviado pela IHM, e função ler(), responsável pelo envio periódico de dados para o gráfico.

Na Figura 2 é mostrada a IHM Mercúrio após algumas interações do usuário, sendo que todos os componentes da IHM funcionaram perfeitamente na interação com o sistema embarcado experimental.

4. Conclusão

Nesse trabalho foi desenvolvido um protocolo e um aplicativo Android que permitem criar uma IHM específica, compatível com qualquer sistema embarcado, independente da aplicação e do microcontrolador utilizado. A IHM é configurada pelo firmware do sistema embarcado via *Bluetooth*, assim, somente os componentes necessários para entrada e saída de dados estarão visíveis, proporcionando ao usuário IHMs diferentes, específicas para cada aplicação. A grande vantagem dessa IHM é a possibilidade de utilizá-la em aplicações diferentes sem a necessidade de alteração do código e recompilação do aplicativo Android, facilitando e acelerando o desenvolvimento. Além disso, permite adicionar à sistemas embarcados com restrições de energia, dimensões e custo, uma IHM "rica" em componentes de entrada e saída.

Nos testes realizados utilizou-se um sistema embarcado experimental, baseado em uma placa Arduino, e a IHM Mercúrio funcionou perfeitamente conforme o esperado.

Sugere-se como trabalhos futuros a modificação dos componentes "Saída de texto" e "Barra de progresso", para que possam ser atualizados periodicamente de forma

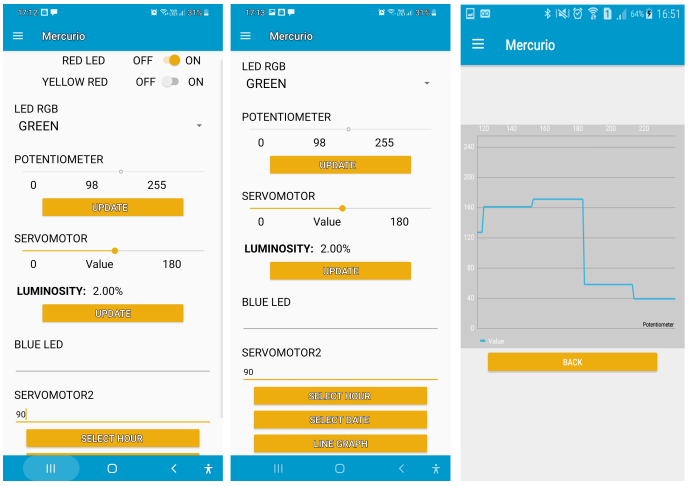

Figura 2. IHM Mercúrio configurada pelo firmware de teste, com a tela deslocada para cima (esquerda) e para baixo (centro), e o gráfico visualizado (direita).

automática, e a utilização da IHM Android em aplicações reais, interagindo com diferentes sistemas embarcados, para verificar suas limitações e implementar novos componentes de entrada e saída, que porventura sejam necessários.

Dado o exposto, percebe-se que esse trabalho trouxe uma contribuição útil ao desenvolvimento de IHMs para sistemas embarcados, principalmente aqueles com restrições no hardware.

Agradecimentos

O presente trabalho foi realizado com apoio da Universidade Federal de Mato Grosso do Sul – UFMS/MEC/Brasil, e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES/Brasil.

Referências

Android Studio 2.2.2. Disponível em: https://developer.android.com/studio/index.html?hl=pt-br

Barbiero, A. A.; Hexsel, R. A. Simulação de Sistemas Embarcados utilizando ArchC. VII Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2006). Ouro Preto - MG, 2006.

BBC News Brasil. O ataque de hackers ao maior oleoduto dos EUA que fez governo declarar estado de emergência. 10 maio 2021. Internacional. Disponível em: https://www.bbc.com/portuguese/internacional-57055618

Bn-Link. 7-Day Digital Dual Outlet Outdoor Timer - BND-60/U58S. Disponível em: http://www.bn-link.com. Acesso em: jun. 2020.

Borges, R. W. Aplicabilidade de Sistemas Operacionais de Tempo Real (RTOS) para sistemas embarcados de baixo custo e pequeno porte. Dissertação de Mestrado - Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

Curry, D. Android Statistics (2021). Disponível em: https://www.businessofapps.com/data/android-statistics/ Acesso em: jun. 2021.

Feijó, V. C.; Gonçalves, B. S.; Gomez, L. S. R. Heurística para Avaliação de Usabilidade em Interfaces de Aplicativos Smartphones: Utilidade, Produtividade e Imersão. Programa de Pós-Graduação em Design e Expressão Gráfica. Universidade Federal de Santa Catarina, Florianópolis, Brasil. 2013.

Full Gauge Controls. Termostato Digital TIC-17RGTi - Manual do Produto. Ver.09. Disponível em: https://www.fullgauge.com.br/produto-tic-17rgt-i. Acesso em: jun. 2020.

MakerPlot - Graphical Plotting, Data Acquisition and Control Software for Your Microcontroller. SelmaWare Solutions, LLC. Disponível em: http://makerplot.com/ Acesso em: out. 2021.

MegunoLink - What is MegunoLink? Disponível em: http://www.megunolink.com/introduction/what-is-megunolink/ Acesso em: out. 2021.

myDevices Inc. The world's first drag-and-drop IoT project builder. Disponível em: https://developers.mydevices.com/cayenne/features/ Acesso em: nov. 2021.

Poushter, J. Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies. Spring 2015 Global attitudes Survey, Pew Research Center. 2016.

Raajkumar, R. Interfacing the HC-06 Bluetooth module with Arduino. Disponível em https://create.arduino.cc/projecthub/RucksikaaR/interfacing-the-hc-06-bluetooth-module-with-arduino-f9c315 Acesso em: jun. 2021.

Rain Bird. Digital Hose-end Timer 1ZEHTMR - Instructions. Disponível em: https://www.rainbird.com/sites/default/files/media/documents/2018-03/man_1ZEHTMR-DigitalHoseEndTimer.pdf. Acesso em: mai. 2020.

Sager, I. Before IPhone and Android came Simon, the first smartphone. Bloomberg Businessweek. v. 29, 2012.

Selvan, K. V., Ali, M. S. M. Micro-scale energy harvesting devices: Review of methodological performances in the last decade, Renewable and Sustainable Energy Reviews, Volume 54, 2016, Pages 1035-1047.

TagoIO Platform. Disponível em: https://tago.io/. Acesso em: nov. 2021.

Telford, T., Englund, W., Laverty, R. Fuel shortages crop up in Southeast, gas prices climb after pipeline hack. The Washington Post, 11 may 2021. Business. Disponível em: https://www.washingtonpost.com/business/2021/05/11/gas-shortage-colonial-pipeline/

Tosi, J.; Taffoni, F.; Santacatterina, M.; Sannino, R.; Formica, D. Performance Evaluation of Bluetooth Low Energy: A Systematic Review. Sensors 2017, 17, 2898. https://doi.org/10.3390/s17122898

Tristan, L. How Much Do Average Apps Make?. Forbes. 10 Aug. 2013.