Capítulo 2 - Monografia

2.1. Desenvolvimento

Para facilitar o desenvolvimento do aplicativo, o sistema foi dividido em duas partes: Construção do *hardware* e elaboração do *software*, as quais são realizadas separadamente e, ao final, são integradas em um único projeto.

2.1.1. Construção do hardware

O controle físico do veiculo é realizado pelo microcontrolador PIC, porém os comandos são provenientes do celular. A maior dificuldade do projeto consiste em realizar a comunicação entre esses dois dispositivos. Para isso utiliza-se o modulo HC-05 que contém interfaces de comunicação *Bluetooth* e Serial. Desta forma, esse dispositivo trabalha como intermediário na troca de informações entre o PIC e o aparelho celular. Na Figura 20, pode ser visualizado este componente.

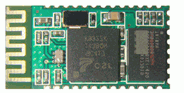


Figura 20 – Módulo Bluetooth HC-05

O primeiro passo é aprender como funciona o modulo*bluetooth*, para isso utiliza–se o manual fornecido pelo fabricante que lista os diversos comandos suportados.

Para que o módulo HC-05 funcione corretamente são necessárias algumas configurações prévias, tais como: modo mestre ou escravo, segurança, velocidade de comunicação, entre outros. Para efetuar essa configuração pode ser utilizado o *software* SSCOM32E, para transmitir ao módulo, via porta serial,uma sequência de instruções conhecida como comandos AT.Na Figura 21, é ilustrada a interface desse programa.

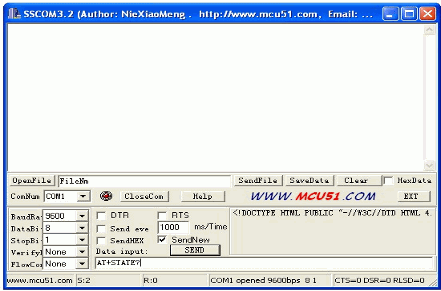


Figura 21 – Interface do programa SSCOM32E

Após uma análise do circuito elétrico apresentado no manual foi possível identificar os pinos de conexão e suas funcionalidades. Os pinos identificados foram:

Pino 1 Saída de comunicação serial

Pino 2 Entrada de comunicação serial

Pino 12 Alimentação positiva com 3,3 Volts

Pino 13,21 e 22 Alimentação negativa

Pino 31 LED indicador de status

Pino 32 LED indicador de pareamento

Pino 34 Configuração de modo de funcionamento.

Foram então, soldados fios nesses pinos e realizada a montagem na matriz de contatos, dessa forma, é possível verificar na prática o funcionamento de diversos comandos suportados pelo dispositivo. Essa montagem pode ser observada na Figura 22.

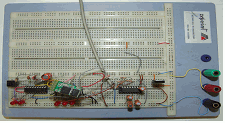


Figura 22 – Matriz de Contatos

Ao receber os comandos, o módulo HC-05 responde com um “OK”, para indicar que o comando foi aceito sem erros, ou a palavra ERROR(0) com código de erro, para indicar que houve falha. Durante a configuração, vários comandos são enviados, cada um com uma função específica, o primeiro é o AT para verificar se o módulo está funcionando corretamente.

Depois de fazer a verificação de funcionamento do módulo, é necessário configurá-lo para trabalhar no modo escravo ou mestre. No modo escravo, o dispositivo aguarda um pedido de conexão e, no modo mestre, o módulo procura por dispositivos para se conectar. O comando usado para verificar o modo de trabalho do HC-05 é AT+ROLE? e para escolher o modo em que o mesmo vai trabalhar é o AT+ROLE<Param>, no qual <parâmetro> pode ser 0 (escravo) e 1(mestre).

Uma funcionalidade interessante é a opção de provocar um “RESET” no módulo e restaurá-lo ao estado original, caso alguma configuração faça com que o módulo deixe de trabalhar.

De acordo com o manual, os passos iniciais para testar o funcionamento como mestre são:

1 – Conectar o pino 34 ao positivo da alimentação para que o dispositivo entre no “modo AT” e aceite os comandos pela serial.

2 – Ligar a alimentação do módulo interface de comunicação serial a um computador com um programa para comunicação serial, por exemplo, o SSCOM32E, configurado na velocidade de 38400 bps, oito *bits* de dados, um *StopBit*, sem paridade.

3 – Envia-se o comando “AT+ROLE=1\r\n”, caso tudo ocorra normalmente o módulo responde com “OK”.

4 – Desligar o módulo, conectar o pino 34 ao negativo e ligar novamente o módulo.

A partir desse momento, o módulo inicia a procura por dispositivos escravos para se conectar.

Com o dispositivo testado, é necessário mudar o seu modo de conexão, pois no projeto, o celular que se conectará ao módulo, então, este deve estar configurado como escravo e para isso, basta seguir os mesmos passos, mudando apenas o comando enviado no passo quatro para:

“AT+ROLE=0\r\n”

No módulo escravo, é muito importante configurar uma senha, para garantir que apenas dispositivos conhecidos, conseguirão conectar-se a ele. Para inserir essa funcionalidade, basta enviar o seguinte comando: “AT+PSWD=<Param>”. Onde o <param> entende-se como a senha a ser gravada.

À partir desse momento o módulo ficará visível a outros dispositivos *Bluetooth*, a espera de uma conexão.

Para realizar o próximo passo foi necessário desenvolver o esquema elétrico da placa que contem o microcontrolador, pois ele será responsável por interpretar os dados recebidos pela porta serial e executar a ação no veículo, como por exemplo, acionar a sirene e desligar a ignição. Essa placa substituiu a matriz de contatos.

Para poder efetuar o desenvolvimento do esquema elétrico, foi utilizado o software CircuitMaker 2000, e o resultado obtido pode ser visto na figura xxxx.

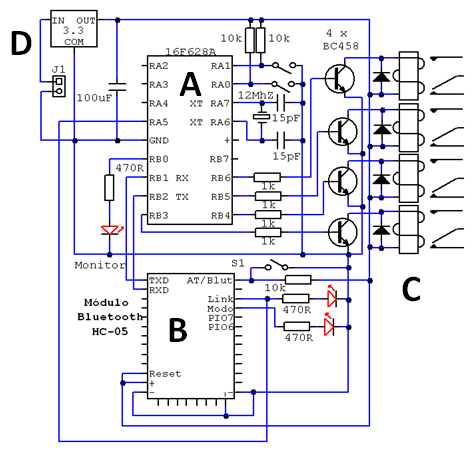


Figura xxxx – XXXX

No esquema elétrico pode ser visualizado o Microcontrolador PIC16F628A (A) responsável pelo controle físico do veículo. O módulo Bluetooth HC-05 (B) que serve como intermediário na comunicação entre o celular e o microcontrolador. Os relês (C) para acionamento de cargas externas de potência, já que as saídas do PIC apresentam apenas um sinal elétrico de baixa corrente. O regulador de tensão (D) para 3,3 volts, pois o PIC trabalha com tensões de 2 a 5 volts, mas o módulo HC-05 somente com 3,3 volts.

Após o esquema elétrico estar totalmente concluído, foi preciso à elaboração de um circuito impresso que representa fisicamente o módulo eletrônico, que será responsável em efetuar os comandos no veiculo.

O circuito impresso é a forma de representar onde serão fixados os componentes eletrônicos de forma a conectar corretamente os terminais de todos os componentes no menor espaço possível em uma placa de fenolite que é revestida em um de seus lados por uma fina camada de cobre.

O desenvolvimento do desenho é realizado com o uso do software xxxxx, a fim de proporcionar um acabamento mais compacto e profissional, após a elaboração do desenho conforme mostra a figura xxx foi preciso imprimi-lo em uma folha de papel fotográfico de baixa porosidade denominada glossypaperutilizando uma impressora a laser.

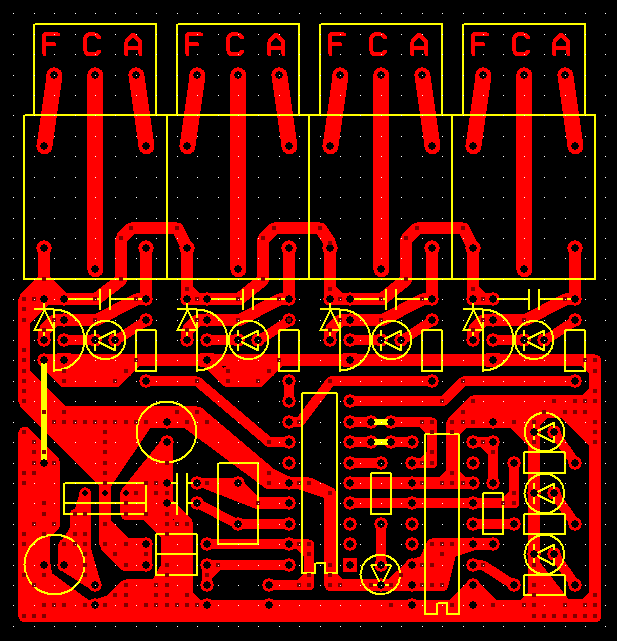


Figura xxx – XXXX

A técnica de corrosão consiste em transferir o desenho do circuito que esta na folha de papel fotográfico para uma placa de fenolite, para realizar esse procedimento foi utilizado um ferro de passar roupa comum.

Esse processo é muito parecido com a técnica de transfer para camisetas, pois o papel com a imagem do circuito é colocado em contato com face de cobre placa de fenolitee pressionando o ferro de passar roupas em alta temperatura sobre esse papel por aproximadamente uns cinco minutos para provocar a fixação da impressão sobre o cobre.

Depois de cinco minutos o papel será grudado na placa, e a mesma é então mergulhada em um recipiente com percloreto de ferro em torno de duas horas. Após duas horas de molho é preciso realizar a remoção total desse papel para que somente a imagem do circuito impresso permaneça da placa, para que o seu funcionamento seja perfeito evitando assim qualquer tipo de mal contanto entre os componentes e os trilhos de cobre, para realizar a remoção basta lavar a placa de fenolite com agua e esfregar a mesma por alguns minutos.

Os circuitos não são corroídos, pois a tinta da impressora protege a placa, com os processos realizados nota-se com clareza os caminhos do circuito e as posições de todos os componentes que serão soldados.

Após passar pelo processo de corrosão, é realizada a perfuração dos pontos de passagem dos terminais de cada componente eletrônicona placa, iniciando assim o processo de solda o resultado do módulo eletrônico totalmente construído com todos os componentes soldados na placa pode ser visto na figura xxxxx.

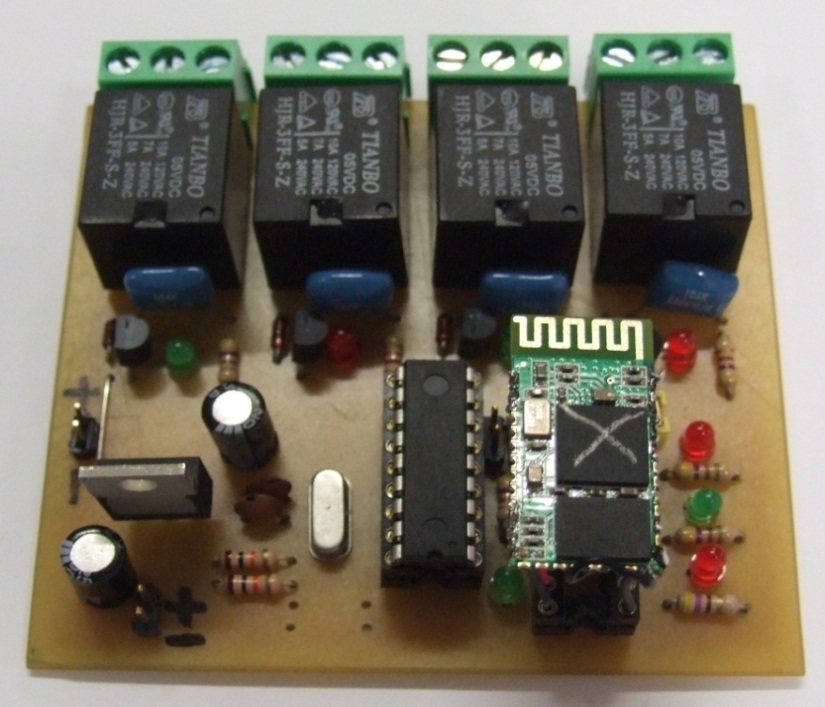


Figura xxxxx – XXXX

Com o módulo eletrônico construído e a troca de informações entre o módulo bluetooth e microcontrolador funcionando, o próximopassofoi desenvolvero código Assembly para tratar a informação recebida e executar as ações no veículo. Esse código é executado dentro do PIC de acordo com as informações que são transmitidas pelo celular.

Para desenvolver e testar o código em modo depuração foi utilizada a IDE MPLab e as rotinas desenvolvidas, que envolvem ações no carro foram:

1. Desligar ignição
2. Acionar sirene
3. Travar portas
4. Destravar portas
5. Piscar faróis para localizar o veículo estacionado

Ver com o Ivair se preciso colocar algo do código assembly.

2.1.2.Elaboração do Software

Para realizar todas as tarefas necessárias, como rastrear o veículo, receber os comandos por SMS, interpretá-los, enviar comandos para o microcontrolador, confirmar o desligamento, foi necessário descobrir dentro da API Android o que poderia ser usado e entender como utilizá-la.

Como a principal dificuldade do projeto é fazer a comunicação Bluetooth com o microcontrolador, esse foi o ponto escolhido para inicializar o desenvolvimento do software nessa etapa.

Basicamente, tudo o que é necessário para realizar comunicação via Bluetooth, encontra-se nas classes BluetoothAdapter, BluetoothDevice e BluetoothSocket.

A classe BluetoothAdapter representa o dispositivo Bluetooth local do aparelho celular e entre suas principais funções estão: verificar se o dispositivo está ativo, iniciar busca por outros dispositivos e verificar se ele ainda está buscando. Na figura 25 é possível ver um exemplo de código que obtém o dispositivo, verifica se ele está ativo e inicia a busca por outros dispositivos.

privateBluetoothAdaptermBluetoothAdapter;

...

// Obtém dispositivo Bluetooth default

mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Verifica se o dispositivo está ativado

if (mBluetoothAdapter.isEnabled()) {

// Se estiver ativado, inicia busca por outros dispositivos

mBluetoothAdapter.startDiscovery();

// Aguarda até terminarem as buscas

while (mBluetoothAdapter.isDiscovering()) {

try {

synchronized (this) {

this.wait(1000);

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Figura 25 – Obter, verificar se está ativo e buscar outros dispositivos.

Para obter os dispositivos encontrados, é necessário utilizar uma classe chamada BroadcastReceiver, que trabalha junto com a BluetoothAdapter. Quando o adapter inicia a busca, toda vez que um dispositivo é encontrado, o sistema envia uma chamada broadcast para as aplicações do celular que possuem o receiver configurado.

No caso desse projeto, ele obtém o dispositivo remoto encontrado e coloca em uma lista Java (ArrayList). Todo dispositivo encontrado, é representado pela classe BluetoothDevice, que possui entre as principais funções, os métodos para obter o nome e preparar uma conexão a ele.

Quando se encerra a busca por novos dispositivos, o sistema trata a lista de devices encontrados e verifica se algum deles é o microcontrolador. Se for encontrado, prepara a conexão.

Ao preparar uma conexão, é obtido uma instância da classe BluetoothSocket, que faz a conexão realmente. A partir desse momento, é possível enviar dados ao dispositivo utilizando a classe OutputStream, nativa do Java. Essa classe é provida pelo socket.

Na figura 26, é possível ver o tratamento e como é feito a conexão.

private final List<BluetoothDevice>lstDevices;

privateBluetoothDevicealarmeDevice;

privateBluetoothSocketbtSocket = null;

private OutputStream outStream = null;

…

// Percorre toda lista de devices encontrados

for (BluetoothDevice device : lstDevices) {

// Verifica se algum deles é o microcontrolador (pelo nome)

if (settings.getString("picName", "").equals(device.getName())) {

alarmeDevice = device;

break;

} else {

alarmeDevice = null;

}

}

…

// Verifica se encontrou o dispositivo

if (alarmeDevice != null) {

BluetoothSocket tmp = null;

try {

// Prepara a conexão

Method m =

alarmeDevice.getClass().getMethod(

"createRfcommSocket", new Class[] {int.class});

// Obtéminstância do BluetoothSocket

tmp = (BluetoothSocket) m.invoke(alarmeDevice, 1);

} catch (Exception e) {

e.printStackTrace();

}

btSocket = tmp;

try {

// Tenta conectar-se ao dispositivo

btSocket.connect();

outStream = btSocket.getOutputStream();

} catch (Exception e) {

e.printStackTrace();

}

}

Na figura 26, conexão com o dispositivo remoto.

Depois de concluídos todos esses passos, faltava apenas enviar dados e certificar que estavam chegando corretamente ao microcontrolador. Para isso, foi ligada a saída do módulo Bluetooth diretamente na porta serial de um computador. Com o programa XXXX era possível exibir o que estava sendo recebido pelo módulo e transmitido pela sua porta serial de saída.

Depois de todos os testes realizados, foi trocada a entrada serial do computador, pela entrada serial do PIC. Na figura 27, é possível verificar como enviar dados pelo Android utilizando a classe OutputStream. Um ponto de extrema importância, é que os dados enviados são ASCII.

privateOutputStreamoutStream = null;

…

try {

outStream.write(codigo);

} catch (IOException e) {

e.printStackTrace();

}

Na figura 26, envio de dados ao dispositivo remoto

Os testes dessa etapa foram todos realizados utilizando uma interface simples no celular que possuía um campo que era preenchido o valor que seria enviado e um botão que disparava a ação.

Com a conexão e troca de dados realizados e testados, o próximo passo realizado foi o de obtenção das coordenadas GPS. Para esse etapa, basicamente foram utilizadas uma classe e uma interface da API android, que chamam-seLocationManager e LocationListener, respectivamente.

Para obter as coordenadas toda vez que ela muda, é necessário criar uma implementação da interface listener e registrar uma requisição de atualização de coordenadas. Com isso, Android começa a buscar atualizações de coordenadas.

Para realizar o rastreamento é possível utilizar duas opções, uma que notifica quando a coordenadas mudam e outra por tempo. As duas opções são configuráveis pelo desenvolvedor. No caso da opção de ser notificado toda vez que as coordenadas mudam, ele define a distância mínima e no caso por tempo, define que a cada X milissegundos quer receber as coordenadas. No caso desse projeto, estão sendo utilizadas ambas as opções para maior segurança.

Na figura 27 é possível ver a implementação da interface e como faz para registrar requisições de atualização.

privateLocationManagerlocationManager;

…

// Obtéminstância do LocationManager

locationManager = (LocationManager) service.getSystemService(

Context.LOCATION\_SERVICE);

// Registra requisição de atualização de localidade

locationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, 10000, 0, gpsLocationListener,Looper.getMainLooper());

…

// Cria um LocationListener

private final LocationListenergpsLocationListener = new LocationListener() {

@Override

public void onStatusChanged(String provider, int status, Bundle extras) {

switch (status) {

caseLocationProvider.AVAILABLE:

break;

caseLocationProvider.OUT\_OF\_SERVICE:

break;

caseLocationProvider.TEMPORARILY\_UNAVAILABLE:

break;

}

}

…

// Toda vez que a localização muda, esse método é chamado

@Override

public void onLocationChanged(Location location) {

// Mostra as coordenadas obtidas na tela do celular

Toast.makeText(this,

String.valueOf(location.getLatitude())+String.valueOf(location.getLongitude()),

Toast.LENGTH\_LONG);

}

};

Na figura 27, obter coordenadas GPS

Os testes iniciais para verificar se as coordenadas estão sendo geradas de forma, foi a mesma estratégia do passo anterior, com a diferença de que a caixa de texto não era mais necessária. Neste caso, foi utilizado uma interface com apenas um botão, que ao ser clicado, começava todo procedimento de rastreamento.

Ao recuperar as coordenadas, o aplicativo mostrava na tela do celular os números referentes a longitude, e para verificar que ele estava mostrando corretamente, era necessário apenas inserir esses dados no Google maps para ver o resultado.

Com a garantia que o sistema estava rastreando corretamente, ainda faltava substituir para o celular enviar as coordenadas a um servidor ao invés de exibir na tela do celular, mas, só foi possível realizar esse passo após o servidor estar pronto. Esse tópico voltará a ser abordado após o terminoda construção do servidor.

O terceiro ponto considerado mais importante na construção do software, foi a interpretação de mensagens SMS, pois é dessa forma que se envia ordens para o microcontrolador realizar as ações necessárias.

Para que um aplicativo seja notificado pelo sistema operacional toda vez que uma nova mensagem SMS chega ao aparelho é necessário criar uma classe que estende a classe BroadcastReceiver e sobrescrever o método onReceive.

Alguns tratamentos para filtrar apenas mensagens SMS, e obter o corpo da mensagem e quem a enviou, foram realizados. Um exemplo desse código pode ser visto na figura 28.

public class SmsReceptor extends BroadcastReceiver {

static final String ACTION = "android.provider.Telephony.SMS\_RECEIVED";

@Override

public void onReceive(Context context, Intent intent) {

// Verifica se a ação recebida é de um SMS recebido

if(intent.getAction().equals(ACTION)) {

Bundle bundle= intent.getExtras();

if(bundle != null ){

Object [] pdus=(Object[]) bundle.get("pdus");

// Obtém a mensagem SMS

SmsMessage message = SmsMessage.createFromPdu((byte[]) pdus[0]);

// Verifica se não é e-mail

if (!message.isEmail()) {

// Verifica se mensagem recebi possui pelo menos 2 caracteres

if(message.getMessageBody().length() >=2 ) {

Intent intentAlarme = new Intent("ALARME\_ANDROID");

intentAlarme.putExtra("Message", message.getMessageBody());

intentAlarme.putExtra("Cellphone", message.getOriginatingAddress());

// Dispara serviço de tratamento de mensagens

context.startService(intentAlarme);

}

}

}

}

}

}

Figura 28 – BroadcastReceiver para ler mensagens SMS

Além de implementar essa classe, também é necessário realizar uma configuração no arquivo “AndroidManifest.xml”. Neste arquivo, encontram-se todas as configurações necessárias para a aplicação. Nesse caso, estamos registrando que a aplicação deve ser notificada quando uma mensagem chegar.

Juntamente com a leitura das mensagens SMS, foi necessário criar um protocolo para saber como seria interpretado a senha e a opção de menu. Para isso, foi definido que a mensagem conteria o primeiro caractere como a opção de menu e os subsequentes como a senha. Portanto, a mensagem deve ter no mínimo dois caracteres. Foi definido também os números do menu que seriam utilizados, e o resultado disso pode ser visto a seguir:

1. - Ativar o alarme
2. - Localizar o carro
3. - Travar as portas
4. - Obter posição atual
5. - Cancelar localizador
6. - Destravar portas
7. - Desativar o alarme

Portanto, supondo que a senha seja “SENHA DO ALARME” e o usuário deseja ativar o alarme, ele deveria enviar um SMS com a seguinte mensagem “1SENHA DO ALARME”.

Para realizar os testes desse passo, bastava enviar mensagens ao celular, e verificar que ele estava ativando a opção correta do microcontrolador e/ou estava realizando as buscas do GPS e exibindo as coordenadas na tela.

Logo em seguida, foi desenvolvido o mecanismo de envio de mensagem de confirmação. Esse mecanismo consistia em enviar uma mensagem para o mesmo celular que requisitou ação dizendo se foi possível executar com sucesso ou não.

Foi definido que haveria mensagem de confirmação apenas quando o celular tentasse se comunicar com o microcontrolador, pois, esses são as ações mais críticas, como ativar ou desativar o alarme, travar ou destravar a porta e ativar o localizador.

Basicamente, toda vez que chega uma ação pedindo uma das opções citadas anteriormente, o sistema tenta se comunicar com o microcontrolador, qualquer falha que aconteça, uma mensagem de erro é enviada. Se tudo ocorrer sem problemas, uma mensagem de sucesso.

Para enviar mensagens automáticas, a API do Android disponibiliza uma classe chamada SmsManager, que possui o método sendTextMessage. Esse método recebe entre os principais parâmetros, o corpo da mensagem e o número a que será enviado.

Após o sistema que rodará dentro do celular Android estar quase pronto, faltou apenas enviar as coordenadas GPS para o servidor, portanto necessitou-se o desenvolvimento deste.

A aplicação deveria ser capaz de receber as coordenas e armazená-las em um banco de dados, além de exibi-las no Google maps quando o usuário acessasse o site e digitasse seu ID e senha. Por se tratar de uma aplicação web simples, que utiliza banco de dados e Java, foi decidido usar o container web Tomcat, desenvolvido pela Apache Software Foundation.

Para o recebimento e armazenamento das coordenadas enviadas, foi utilizado Servlet simples, que quando era recebido uma requisição HTTP comum, obtinha os parametros das coordenadas, o ID do automóvel e a senha. Em seguida, armazenava-os em um banco de dados MySQL.

O MER (modelo entidade relacionamento) deste projeto é extremamente simples, pois é necessário apenas identificar o automóvel que enviou as coordenadas e armazená-las. Por isso, o sistema possui apenas duas tabelas, uma que guarda o automóvel e uma que guarda o histórico das coordenadas. Na figura 29 é possível ver o MER do sistema.

|  |
| --- |
| Adicionar imagem aqui – será adicionado em 26/11 |

Figura 29 – Modelo Entidade relacionamento

Para facilitar a exibição das ultimas coordenadas GPS recebidas pelo servidor web no Google maps, foi utilizado o framework JSF (Java Server Faces). Esse framework facilita a elaboração e construção de paginas Java web além de proporcionar facilidade na construção de aplicações no modelo MVC.

O resultado obtido, da tela inicial, em que o usuário digita seu ID e senha e em seguida obtém a última localização enviada do automóvel mostrada no Google maps pode ser visto nas figuras 30a e 30b, respectivamente.

|  |
| --- |
| Adicionar figura aqui – será adicionado em 26/10 |

Figura 29a – Tela de Login| Figura 30b – Google Maps

Para fechar o desenvolvimento do software, bastava alterar para ao invés de exibir na tela do celular as coordenadas obtidas, enviar para o servidor web. Como o servidor está esperando uma requisição HTTP comum com parâmetros, bastou implementar uma chamada simples.

A maior dificuldade nesse momento, foi descobrir como realizar essa chamada sem possuir um browser e de maneira automatizada (em que o aplicativo conseguisse fazer) de dentro de uma aplicação mobile.

As classes utilizadas para resolver esse problema foram as nativas da própria API Java, URL e HttpURLConnection.

Em conjunto, essas classes conseguem realizar uma requisição HTTP passando parâmetros, sem a necessidade de um browser. A figura 31 mostra como realizar uma requisição.

try {

URL url = new URL(urlString);

HttpURLConnection connection =

(HttpURLConnection) url.openConnection();

connection.setRequestProperty("Request-Method", "GET");

connection.setDoInput(true);

connection.setDoOutput(false);

connection.connect();

connection.getResponseMessage();

connection.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

Figura 30 – Requisição HTTP

Após o aplicativo que roda no celular e o servidor web estarem prontos, foi desenvolvido uma interface configurável para o celular, que servirá para definir vários dados que o sistema necessita e mudá-los sempre que necessário. Esses dados são:

1. Servidor – Representa o endereço do servidor que o aplicativo deverá utilizar para enviar as coordenadas GPS
2. ID Carro – cadastrado no sistema
3. Senha Carro – cadastrado no sistema
4. PIC – Qual nome está configurado o Bluetooth do microcontrolador para o celular encontrá-lo e conectar-se
5. Senha – Utilizada para verificar se o SMS recebido deve ser interpretado ou não

Após todo o software e o hardware estarem prontos e testados parcialmente (durante a integração das partes), foram realizados os testes finais, com todo o sistema funcionando.

Os testes simularam situações reais, onde eram enviados os comandos com opção a ser testada, e verificava-se se o resultado obtido era o esperado.

Entre os vários testes realizados, podem-se destacar situações de sucesso, como ativar e desativar o alarme, ativar desativar o localizador, obter posição atual do automóvel e travar e destravar as portas. Um teste de extrema importância realizado, foi o do sistema de rastreamento, que para isso, foi feito testes andando pelas ruas próximas a faculdade para verificar que as coordenadas estavam sendo enviadas para o servidor.

Foram também realizados inúmeros testes de falha, como por exemplo, desligar o microcontrolador e enviar os comandos para ativar ou desativar o alarme ou travar e destravar as portas. Nesse caso, não era possível realizar as ações, portanto, era esperado que uma mensagem de erro fosse enviada para o mesmo celular que a solicitou.

Teste de erros mais simples também foram realizados, como enviar uma opção de menu inválida, ou até mesmo enviar uma opção existente mas com a senha incorreta.

No acesso ao servidor, foram realizados além dos testes de sucesso, que consistia em digitar um ID e senha corretos, tentar acessar com esses dados inválidos ou realizar o acesso aos dados de um automóvel sem autorização.

Com todos os testes realizados, foi possível verificar que o alarme estava funcional.

Capítulo 3– Análises e Resultados

Com o hardware e o software finalizados, os objetivos definidos no inicio do projeto foram alcançados. Com isso, é possível instalar o alarme em qualquer automóvel e utilizar todas as funções propostas sem erros.

O alarme consiste em: um dispositivo físico conectado ao sistema elétrico do veículo, um software instalado no celular Android que fica escondido no automóvel e uma aplicação instalada em um servidor java web, para ser utilizado quando a funções de rastreamento pelo GPS é ativada.

Para desenvolver todo o conjunto e possuir um alarme funcional, até a finalização dessa monografia, foi gasto algo em torno de R$ xxx. Esses custos incluem desde os componentes eletrônicos até o celular equipado com o sistema operacional Android. A tabela de preço de todo material utilizado no alarme pode ser visto na tabela X.

Tabela X – Custo final do projeto.

|  |  |  |
| --- | --- | --- |
| Componente | Valor Unitário | Valor Total |
| PIC 16F628A (x1) | R$ 7,60 | R$ 7,60 |
| Módulo Bluetooth ??? (x1) |  |  |
| Regulador tensão |  |  |
| Transistores |  |  |
| Resistores |  |  |
| Led (x6) | R$ 0,15 | R$ 1,50 |
| Celular Android ZTE (x1) | R$ 250,00 |  |

Outro ponto de extrema importância, é que esse trabalho pode servir de referencia para trabalhos futuros, pois se trata de um projeto desenvolvido com a tecnologia Android, que é extremamente nova no mercado, portanto, possui poucas referências.

Como não foi encontrado nenhum trabalho correlato na universidade que utilizava comunicação bluetooth entre Android e microcontrolador, a principio, não era possível saber da compatibilidade entre eles, portanto não era possível saber se a comunicação iria funcionar como esperado.

Esse foi o ponto mais difícil do projeto, pois todas as outras atividades realizadas pelo alarme foram possíveis comprovar a viabilidade com trabalhos correlatos.

Capítulo 4–Considerações Finais

Esse projeto realiza todas as funções propostas sem erros, mas, as tecnologias utilizadas possuem falhas. Desde o inicio, o objetivo sempre foi utiliza-las e não resolver seus problemas.

Entre os problemas existentes nessas tecnologias, pode-se destacar:

1. Telefonia móvel indisponível: quando a rede móvel está indisponível seja por qual motivo for, como por exemplo, um sistema que bloqueia os sinais GSM ao redor ou por falha na rede, não é possível realizar ações no alarme. Esse não é um grande problema, pois os sistema atuais também enfrentam diversos problemas similares.
2. GPS em um local fechado: quando o automóvel está em um local fechado, como por exemplo um túnel, o GPS não consegue obter a localização exata, pois ele não consegue fazer a triangulação entre os satélites. Neste caso, ele obtém uma localização aproximada, pois obtém as coordenadas assistidas pela internet. Neste caso, não há o que fazer, pois até mesmo o melhor aparelho receptor de GPS disponível no mercado perde o sinal ao entrar em um local assim.

Este sistema pode ser melhorado em diversos aspectos e os principais são citados a seguir, pois podem ser feitos em trabalhos futuros.

Atualmente o sistema funciona apenas quando acionado pelo usuário, para ser um alarme completo, é necessário que ele também dispare avisos quando alguém tenta agir sobre o veiculo e o mesmo não está presente.

Diversas opções interessantes tangem esse assunto, como por exemplo, sensor de presença para realizar o disparo e aviso por SMS para um celular pré cadastrado com um aviso que o mesmo encontra-se disparado.

Um dos principais objetivos deste projeto era prover ao usuário total autonomia sobre seu sistema, deixando-o independente de centrais. Atualmente, a única maneira de consultar a posição do carro, é acessar o site.

Manter um servidor com esse site no ar pode ser extremamente complicado para alguns usuários, portanto, customizar a aplicação que roda no celular para ser configurável para ao invés de enviar as coordenadas ao site, responder com as coordenadas para o mesmo celular que solicitou a ação seria extremamente interessante, pois, bastaria o usuário digitar aquelas coordenadas no sistema de mapa que ele deseja. Isso eliminaria o problema de manter um servidor, se o usuário preferir.

Com essas funções implementadas, este sistema pode ser comercializado como um alarme completo e com grande potencial de mercado.