Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Engenharia de Computação
Oficina de Integração 3 (EEX23) – S71 – 2019/1

Relatório Técnico
STEVE - Sistema de Telemetria Veicular

Fernando A. Silva – fernandoargentino@outlook.com
Gabriel S. Silva – silvag.2015@alunos.utfpr.edu.br
Gustavo C. Libel – gustavolibel@alunos.utfpr.edu.br
Igor P. Latini – igorlatini@alunos.utfpr.edu.br
Mateus C. Hércules – mchercules@gmail.com

Julho de 2019

Resumo

Motivado pela ideia de facilitar o diagnóstico de problemas no automóvel e pela segurança veicular, esse documento apresenta o projeto STEVE. Atuando junto ao veículo, o sistema desenvolvido é capaz de coletar informações de desempenho e códigos de erro diretamente do carro, enviando os dados para um servidor web. Através de um aplicativo desenvolvido para *Smartphones Android* e *iOS*, é possível ter acesso a essas informações de modo claro e objetivo, com explicações para os problemas encontrados e possibilidade de acompanhar o desempenho do veículo ao longo do tempo. Para a parte de segurança, o usuário pode não só acompanhar a posição atual de seu automóvel através de um GPS instalado no sistema, facilitando sua localização em caso de furto, como também cortar o fluxo de combustível do veículo através do aplicativo, imobilizando o carro. Por fim, também é possível capturar uma foto do condutor a qualquer instante, para que o usuário do sistema possa saber quem está dirigindo no momento. Com STEVE, é possível conhecer melhor o carro e mantê-lo mais seguro. Mais informações estão disponíveis no link: https://projetosteve.wordpress.com/

1 Introdução

A utilidade dos automóveis é inegável, permitindo um grande decréscimo no tempo de deslocamento e adicionando conforto a viagem. Entretanto, trata-se de um sistema complexo e de custo elevado, podendo apresentar diversas falhas em seus muitos sistemas, além de atrair a atenção de ladrões.

Assim, os avisos e alertas de erros e falhas nos automóveis são, normalmente, motivos de grande estresse para os motoristas, principalmente pela extensa variedade de símbolos e pouca claridade em seus significados [1]. Juntamente, a segurança de automóveis é um problema conhecido no Brasil: no ano de 2015, foi atingida a marca de um roubo ou furto de veículo por minuto [2].

Motivado por esses problemas, o projeto STEVE é um sistema de telemetria e segurança veicular que integra três principais objetivos: informar sobre o desempenho do veículo, traduzir os indicadores de erro apresentados no painel, oferecendo sugestões de resolução, e promover a segurança do automóvel, tornando-o rastreável em caso de furto e permitindo sua imobilização remotamente. A visão geral do sistema pode ser vista no diagrama apresentado na Figura 1.

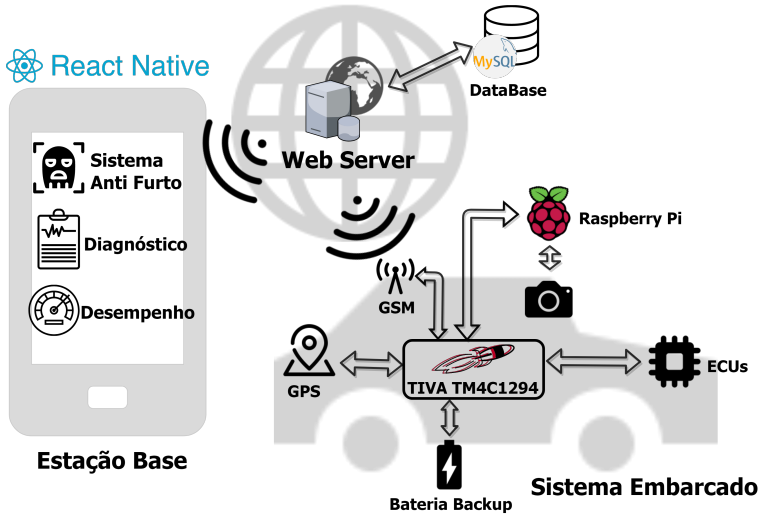


Figura 1: Diagrama Geral - STEVE.

O projeto divide-se em três partes principais: Sistema Embarcado, *Web Server* e Estação Base. O Sistema Embarcado tem como base a plataforma Tiva TM4C1294NCPDT [3], ficando localizado dentro do veículo, lendo os dados das *ECUs* (*Electronic Control Unit*) do automóvel, disponíveis no conector OBD2 [4]. As *ECUs* são responsáveis por controlar a parte elétrica do veículo, trocando mensagens entre si através de protocolos como CAN [5] ou K-Line [6], dependendo da marca e do ano do automóvel. Conectado à Tiva, há uma bateria reserva, um módulo GPS e um Raspberry Pi, sendo que esse último é responsável por capturar imagens a partir de uma câmera. Tanto o módulo GPS quanto a câmera são utilizados na função antifurto do projeto, em que é possível não só saber a localização atual do veículo através da estação base, como também pode-se imobilizá-lo remotamente. Para a transferência e recebimento de dados, há ainda um módulo GSM.

A comunicação entre o Sistema Embarcado e a Estação Base é feita através de requisições ao *Web Server*, que mantém um banco de dados com as informações recebidas do veículo. Dessa forma, a Estação Base também pode fazer requisições, podendo receber os dados e realizar os acionamentos.

A estação base trata-se de um aplicativo com três telas principais. Na primeira, o usuário pode ver os dados quantitativos de desempenho, como rotações por minuto, temperatura do líquido de arrefecimento do motor e posição do acelerador, com gráficos indicando a evolução dos dados ao longo do tempo. Em outra tela, há as informações de diagnóstico, com as devidas "traduções" dos erros apresentados, como [EXEMPLOS]. Por último, há a tela de antifurto, em que é possível visualizar a posição do veículo em tempo real e ativar a função de imobilização do automóvel. Esse bloqueio é feito através do desativamento da bomba de combustível do carro, de forma que a paralisação seja gradual, evitando acidentes. Juntamente à imobilização, há o ativamento dos faróis de luz alta, que começam a piscar.

1.1 Requisitos

Os requisitos foram divididos conforme as três partes principais do projeto. As Tabelas 1, 2 e 3 apresentam, respectivamente, os principais requisitos funcionais e não funcionais do sistema embarcado, do *Web Server* e do aplicativo.

Tabela 1: Requisitos funcionais e não funcionais - Sistema Embarcado

Requisitos Funcionais	
Requisito	**Descrição**
RF01	O sistema deverá coletar dados sobre o veículo[1]
RF02	O sistema deverá ser capaz de acionar os faróis do veículo
RF03	O sistema deverá ser capaz de cortar o fluxo de combustível do veículo
RF04	O sistema deverá ser capaz de tirar foto do motorista
RF05	O sistema deverá ser capaz de utilizar uma bateria reserva
RF06	O sistema deverá ser capaz de carregar uma bateria reserva
RF07	O sistema deverá enviar os dados de GPS a cada, pelo menos, 5 segundos
Requisitos Não Funcionais	
Requisito	**Descrição**
RNF01	O sistema deverá trocar informações com o aplicativo de forma segura (criptografada)
RNF02	O sistema deverá ter comunicação externa através de um módulo GSM
RNF03	O sistema deverá utilizar uma câmera Raspberry como câmera de segurança
RNF04	O sistema deverá usar um Raspberry Pi para controlar a câmera
RNF05	O sistema deverá usar uma Tiva TM4C1294 como base
RNF06	O sistema deverá usar um módulo GPS

[1] Os dados coletados são: RPM, velocidade, carga/torque no motor, pressão no combustível, nível de combustível, temperatura do líquido de refrigeração, posição do acelerador, composição do combustível, odômetro, temperatura do ar de entrada no motor, tempo de funcionamento do veículo, tensão na bateria e marcha atual. Entretanto, é importante destacar que nem todos os veículos disponibilizam todas essas informações. Assim, o conjunto de dados coletados pode ser reduzido dependendo da marca e do ano do automóvel.

Tabela 2: Requisitos funcionais e não funcionais - *Web Server*

Requisitos Funcionais	
Requisito	**Descrição**
RF01	O servidor deverá salvar as informações recebidas do *hardware* no banco de dados
RF02	O sistema deverá enviar para o banco de dados um diagnóstico sempre que o veículo for ligado ou desligado
Requisitos Não Funcionais	
Requisito	**Descrição**
RNF01	Cada parte do sistema deverá utilizar o mesmo serviço de *Web Service* para interagir com as outras
RNF02	O sistema deverá usar banco de dados *MySQL*

Tabela 3: Requisitos funcionais e não funcionais - Aplicativo

Requisitos Funcionais	
Requisito	**Descrição**
RF01	O aplicativo deverá informar ao usuário dados estatísticos e de desempenho do veículo
RF02	O aplicativo deverá informar ao usuário dados diagnósticos de problemas no veículo
RF03	O aplicativo deverá informar ao usuário a localização do veículo
RF04	O aplicativo deverá alertar o usuário sobre condições do veículo
RF05	O aplicativo deverá permitir que o usuário acione o sistema antifurto
RF06	O aplicativo deverá permitir ao usuário ver as fotos tiradas pela câmera de segurança
Requisitos Não Funcionais	
Requisito	**Descrição**
RNF01	O aplicativo deverá executar em um *Smartphone Android*
RNF02	O aplicativo deverá executar em um *Smartphone iOS*
RNF03	O aplicativo deverá ser desenvolvido usando *React Native*

2 Tecnologias

A seguir, encontram-se breves descrições das tecnologias utilizadas no projeto.

2.1 Tiva TM4C1294NCPDT

O TM4C1294NCPDT é um microcontrolador da *Texas Instruments* com processador ARM Cortex-M4F. Possuindo 256KB de SRAM e 1024KB de *flash*, a Tiva conta com quinze blocos físicos de GPIO, oito UARTs e uma interface I^2C. A Figura 2 apresenta o *Evaluation Kit* EK-TM4C1294, utilizado no projeto, que embarca esse microcontrolador e alguns periféricos.

Seu processador utiliza a arquitetura ARMv7E-M [7], sendo projetado para programação em C e uso de RTOS (*Real Time Operating System*) [8], além de possuir um controlador de interrupções fortemente acoplado ao núcleo, permitindo um tratamento eficiente das interrupções. Para facilitar a programação, a *Texas Instruments* disponibiliza a *TivaWare*, um conjunto de bibliotecas contemplando diversos aspectos, como por exemplo *drivers* para periféricos [9].

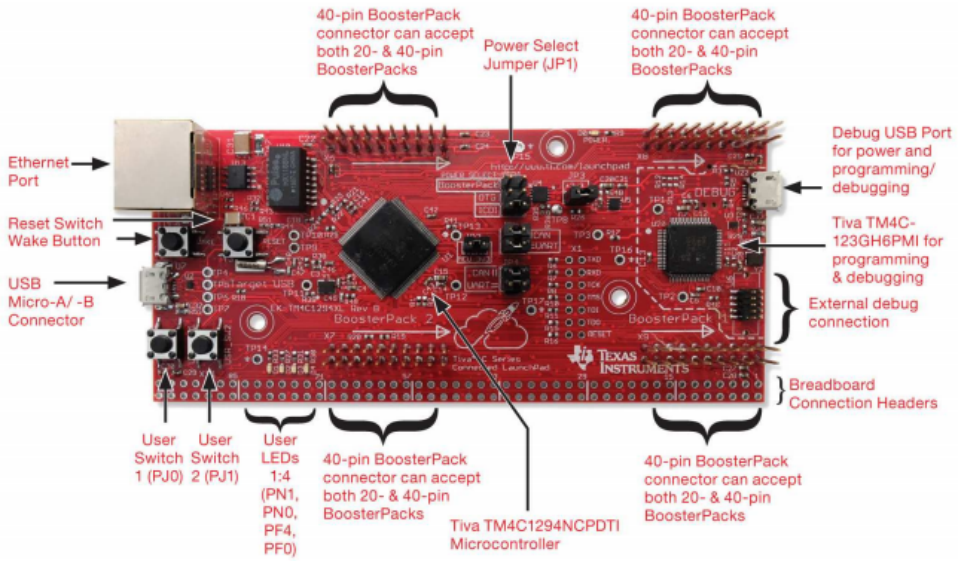


Figura 2: *Evaluation Kit* EK-TM4C1294. Fonte: [10]

2.2 Módulo GPS: Gy-neo6mv2

Através do módulo GPS Gy-neo6mv2 [11], é possível obter dados de posicionamento por satélite com uma taxa de atualização máxima de 5Hz, suficiente para o sistema antifurto de STEVE que necessita de apenas 1Hz. Opera com uma tensão entre 2.7V e 5V, consumindo 45mA de corrente em seu funcionamento normal. Para a comunicação, utiliza o padrão NMEA [12] e serial.

2.3 Módulo GSM: SIM800L

SIM800L [13] é um módulo GSM com suporte GPRS, possuindo *downlink* e *uplink* de 85,6kbps. GSM (*Global System for Mobile Communications*, ou "Sistema Global para Comunicações Móveis") é um sistema de comunicação sem fio utilizado principalmente por *smartphones* [14]. GPRS (*General Packet Radio Service*) é uma tecnologia que aumenta as taxas de transferência de dados nas redes GSM existentes [15].

Sobre suas especificações, o módulo opera com uma tensão entre 3,4V e 5V, com picos de corrente de 2A durante transmissões. É controlado e configurado através de comandos AT [16], usando comunicação serial.

2.4 Módulo Carregador de Bateria de Lítio: Controlador TP4056

Apesar de o projeto ser alimentado pela bateria do veículo, o módulo carregador de bateria [17] permite que haja uma bateria reserva, que alimente o sistema caso a principal não esteja disponível (removida, por exemplo). Trabalha com uma tensão de entrada de até 8V, com uma saída predefinida de 4.2V (com erro

de 1,5%), além de ter proteção contra sobrecarga e pinos indicadores de estado - carregando e carregado.

2.5 Protocolo CAN

CAN (Controller Area Network) é um protocolo de comunicação serial síncrono, cuja principal característica é o fato de não trabalhar com grandes blocos de dados para comunicação entre mestres e escravos, como acontece em outros protocolos. No CAN, as mensagens são trocadas na rede inteira, de forma que todos os módulos podem agir como mestres e escravos, e as mensagens são entregues para todos da rede, tornando os dados consistentes por todo o sistema.

2.6 Barramento K-Line

K-Line, em veículos, é um tipo de barramento e protocolo de comunicação que vem sendo substituído pelo CAN. É baseado em uma comunicação serial e assíncrona, através de apenas um fio, chamado de K-Line.

Trabalha com um sistema de apenas um mestre e múltiplos escravos, sendo esse mestre a unidade de controle para os escravos. A comunicação é feita de forma *half-duplex*, em que se pode transmitir e enviar dados, mas não ao mesmo tempo. Assim, funciona com um sistema de *request and response*, em que, quando há um pedido de algum dado ao barramento, essa informação é, então, fornecida.

2.7 Módulo CAN Bus: MCP2515

O MCP2515 [18] é um controlador que implementa a especificação CAN, na versão 2.0B. Operando com 5V e 5mA (normalmente), é capaz de transmitir e receber dados e frames (formatos padrão e estendido) a 1Mb/s, além de também possuir buffers, máscaras e filtros de aceitação para remoção de mensagens não desejadas, de forma a diminuir o *overhead* do microcontrolador ao qual está conectado. A interface com esse microcontrolador é através do padrão SPI [19]. Sua utilização é necessária principalmente em razão da Tiva não possuir um transceptor em sua interface CAN.

2.8 OBD2

OBD2 (ou OBDII) é um sistema de diagnóstico disponível nos veículos, cuja conexão consiste em um conector padronizado (a partir de 2010 no Brasil, 1996 na Europa e Estados Unidos). Através desse sistema, os diversos subsistemas do automóvel conseguem trocar mensagens e acessar os status uns dos outros, além de possibilitar alertar o usuário por meio de luzes no painel.

Pelo conector, tem-se acesso a mensagens de controle de cada subsistema, indicando qual é o estado deles no momento atual (*real time*). Para requisitar tais dados, são utilizados os PIDs (*Parameter IDs*) [20], cujas respostas vêm com

uma codificação específica. Além disso, o OBD2 provê uma série de códigos de diagnóstico de problemas (*Diagnostic Trouble Codes*: DTCs) [21], que permitem rápida identificação de mal funcionamentos no veículo. A Figura 3 apresenta um diagrama com a pinagem do conector OBD2.

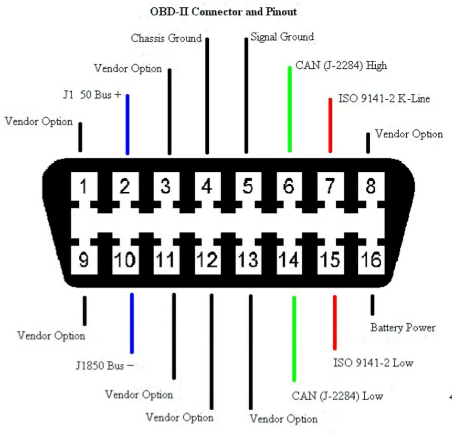


Figura 3: Pinagem e protocolos do OBD2. Adaptado de: [22]

2.9 Raspberry Pi e Raspberry Pi Camera Module v2

O Raspberry Pi 3 [23] conta com CPU de 1.2GHz (Broadcom BCM2837), 1GB de RAM, conector CSI para câmera e 40 GPIOs. Nessa placa, é possível executar diversas distribuições de Linux, como Ubuntu e Raspbian (variação do Debian). O módulo para câmera [24] possui um sensor Sony IMX219 de 8-megapixel, sendo necessário um cabo CSI conectando-a ao Raspberry Pi.

2.10 REST e MySQL

REST (*Representational State Transfer*) [25] é um modelo de arquitetura de software distribuído. Define e implementa um conjunto de restrições e princípios a ser utilizado na aplicação, como por exemplo, identificar o recurso sendo requisitado através de uma URI (*Uniform Resource Identifier*).

O *Web Service* é utilizado em conjunto com o MySQL [26], um sistema de gerenciamento de banco de dados da Oracle Corporation que utiliza a linguagem SQL como interface, tendo alto desempenho e estabilidade.

2.11 React Native

React Native [27] é um *framework* para desenvolvimento de aplicativos nas plataformas iOS e Android, utilizando a linguagem JavaScript. É baseado na biblioteca React, do Facebook, uma biblioteca JavaScript para desenvolvimento *Web*. Entretanto, o *framework* atua como uma ponte, possibilitando o desenvolvimento de aplicativos que sejam de fato nativos, ou seja, não pareçam uma

aplicação *Web* importada como aplicação móvel, o que permite a criação de interfaces responsivas para o usuário.

Para testes, há a ferramenta Expo [28], que possibilita emular o aplicativo no celular através de um servidor local. Já para a navegação entre telas foi escolhida a biblioteca React Navigation [29].

3 Desenvolvimento

3.1 Sistema Embarcado

Composto pela Tiva, pelo Raspberry Pi e pelos módulos citados na Seção 2, o sistema embarcado é o responsável não só pela captura e envio dos dados do automóvel, mas também pela atuação no veículo.

3.1.1 Projeto das Placas

Para o sistema embarcado, foram projetadas duas placas: alimentação e acionamentos. Elas foram projetadas usando o software gratuito KiCad, que permite a elaboração tanto do esquemático quanto do design.

A primeira placa foi a de alimentação do sistema. Ela tem a função de fornecer a tensão e a corrente necessária para todos os outros componentes do sistema embarcado. Para isso, utiliza a própria bateria do automóvel como entrada do circuito, usando de reguladores para alcançar as tensões desejadas. Além disso, conta com proteções contra picos de tensão através dos diodos TVS e também filtra sinais de alta frequência que possam aparecer. O esquemático dessa placa pode ser visto na Figura 4.

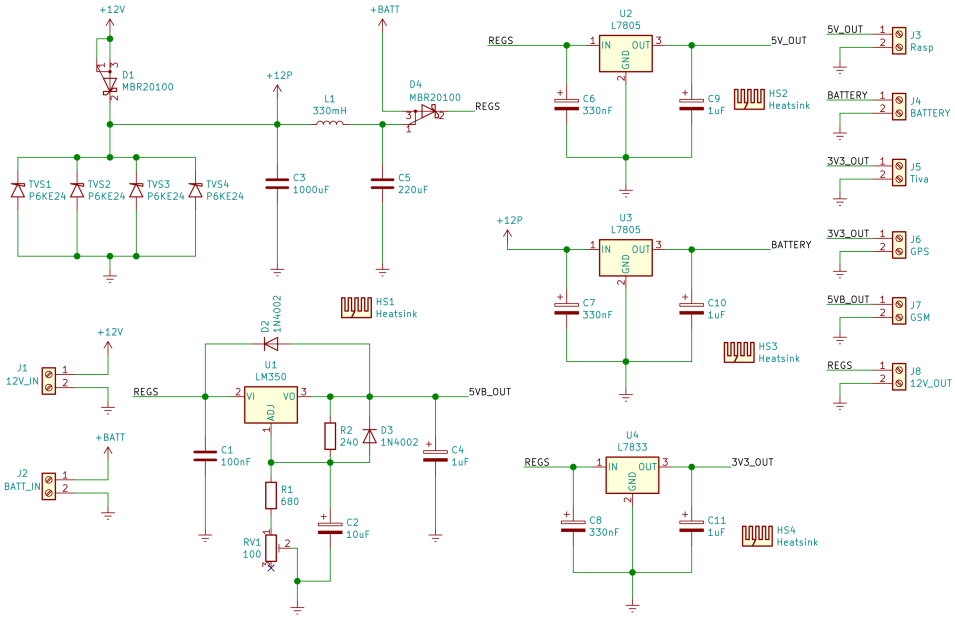


Figura 4: Esquemático da placa de alimentação

Em seguida, foi projetada a placa de acionamento. Seu esquemático pode ser visto na Figura 5. Sua função é atuar agregada ao modo antifurto do STEVE, permitindo que o usuário tenha certo controle sobre o veículo mesmo remotamente. Assim, é possível tanto acionar os faróis quanto desativar a bomba de combustível do veículo, fazendo-o parar gradativamente. Para realizar o acionamento dos faróis e das setas, a placa possui relés que foram colocados de forma a ficarem em paralelo com os relés de acionamento do próprio veículo. Dessa forma, é possível forçar suas ativações sem uma grande interferência no comportamento original do automóvel. Entretanto, para o caso específico da bomba de combustível, o relé deve ficar em série com o acionamento, uma vez que devemos forçar o corte.

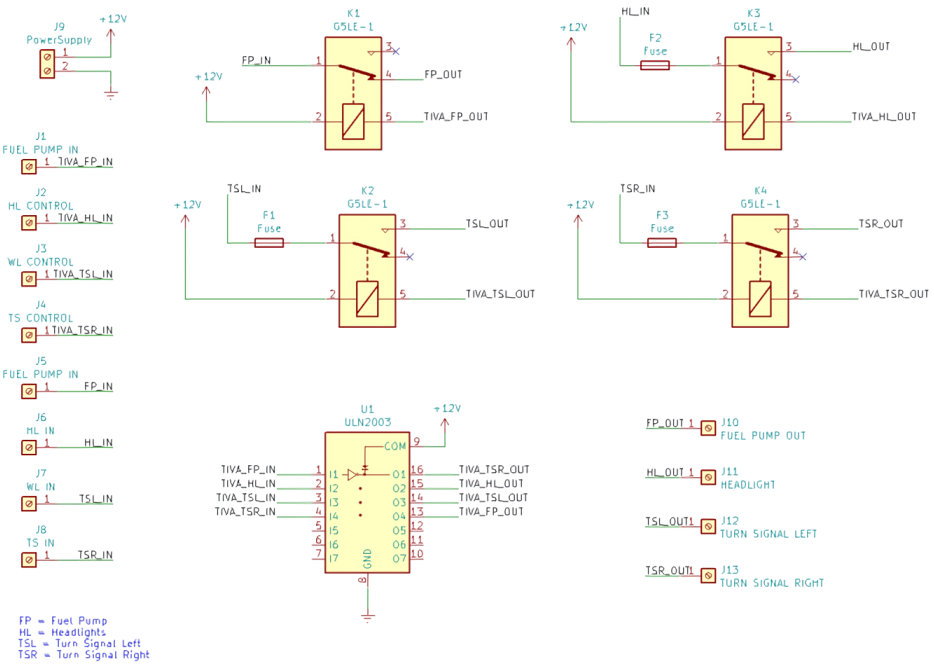


Figura 5: Esquemático da placa de acionamento

Para cortar o fluxo de combustível sem modificar tanto o automóvel, uma solução encontrada foi colocar o relé em série com o interruptor inercial, um dispositivo presente nos veículos com a função de cortar a alimentação da bomba de combustível em caso de colisões e acidentes, de modo a diminuir o risco de incêndio.

Em relação à interface entre o sistema embarcado e o veículo, os protocolos CAN e K-Line, explicados na Seção 2, foram abordados de maneiras diferentes. Para o CAN, foi utilizado diretamente o módulo CAN Bus conectado à Tiva. Já para o K-Line, foi usado um Arduino juntamente com um pequeno transceptor como ponte entre o automóvel e a Tiva, utilizando uma biblioteca disponível gratuitamente para fazer a leitura [30].

3.1.2 Captura de Imagem

Os componentes do sistema de captura de imagens pode ser visto na Figura 6, juntamente com o fluxo de informação.

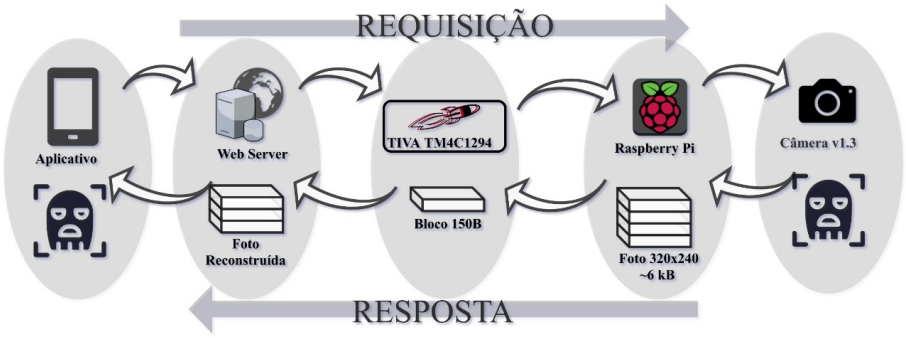


Figura 6: Esquemático - Sistema de Captura de Imagem.

Assim, o sistema de captura de imagem pode ser dividido em duas partes:

Requisição: inicialmente, o usuário faz uma requisição de foto pelo aplicativo. Então, o servidor envia um aviso de requisição de foto para a Tiva, que, por sua vez, o repassa para o Raspberry Pi via protocolo serial.

Resposta: Raspberry Pi captura uma imagem (320x240) em escala de cinza e recorta o rosto do motorista caso seja detectado sua face, utilizando um algoritmo chamado *Haar Cascade* [31]. São enviados blocos de 150B dessa imagem para a Tiva. Por sua vez, a Tiva envia esses blocos para o servidor, que então reconstrói a imagem e a envia para o aplicativo.

O envio foi separado em blocos em virtude da limitação de memória da Tiva e também para facilitar o envio pelo módulo GSM.

3.1.3 *Firmware*

O *firmware* foi desenvolvido baseado em *threads* e uma fila de mensagens, ferramentas disponíveis no RTOS. Há cinco threads principais:

***Thread* Telemetria:** é a responsável por ler os dados do veículo, seja ele CAN ou K-Line.

***Thread* do GPS:** trata principalmente de identificar os valores de interesse que indicam a posição espacial do módulo.

***Thread* da Imagem:** ao receber do servidor uma requisição de foto, a Tiva inicia a comunicação com o Raspberry, fazendo-o utilizar a câmera para tirar uma foto e começar a enviá-la em blocos para ela.

***Thread* de Acionamento:** ao receber o aviso do servidor de que o modo antifurto foi ativado, os pinos da Tiva conectados aos relés são alterados de forma a fazer com que cesse o fluxo de combustível para o veículo e os faróis de luz alta fiquem piscando.

***Thread* do GSM:** é a responsável por ler a fila de mensagens e realizar o envio através do módulo GSM.

A dinâmica de funcionamento ocorre através da comunicação entre a *thread* do GSM e as demais. Todo o recebimento de informação dos módulos é

feito através de interrupções, para tentar garantir que nenhuma mensagem será perdida.

3.2 *Web Server e Banco de Dados*

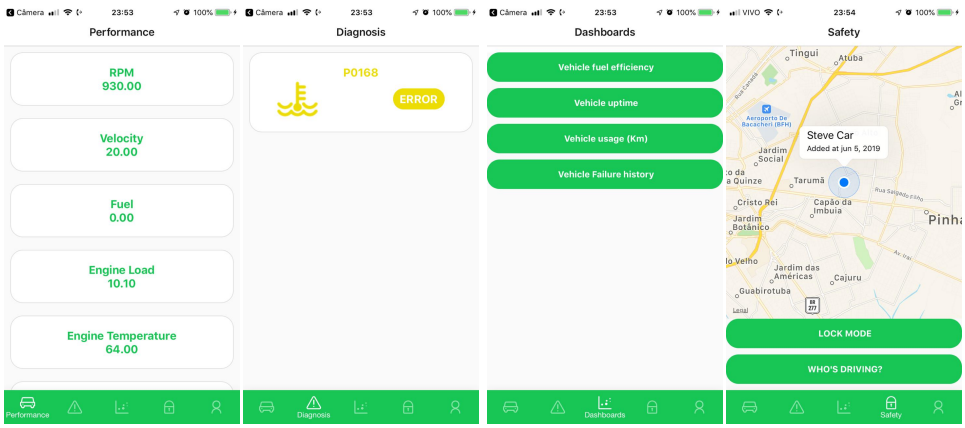
O *Web Server* tem o papel centralizador da comunicação no projeto. Dessa forma, todas as requisições passam por ele, tanto do aplicativo quanto do sistema embarcado, pois ele é o único responsável pelo interfaceamento com o banco de dados.

Para o armazenamento e recuperação das informações, é necessário que todos os dados fiquem disponíveis em um banco de dados. Esse banco é alimentado pelo sistema embarcado e recuperado pelo aplicativo, que, então, disponibiliza as informações recebidas.

Tanto o *Web Server* quanto o Banco de Dados estão hospedados no mesmo servidor, na empresa Solution Source.

3.3 Estação Base - Aplicativo

As telas do aplicativo foram organizadas de acordo com as principais funções do projeto. Com isso, foram projetadas 3 telas: performance, diagnósticos e segurança. Há, ainda, duas outras telas: uma para o histórico, em que é possível ver as mudanças nos dados ao longo do tempo, e outra para o usuário, em que é possível ver informações sobre o usuário e o veículo cadastrado. As telas podem ser vistas na Figura 7



(a) Tela de performance (b) Tela de diagnóstico (c) Tela de históricos (d) Tela de segurança

Figura 7: Telas principais do aplicativo

A tela de performance, responsável pela visualização das informações de desempenho em tempo real do veículo, foi desenhada de maneira simples, apenas como uma lista de atributos e seus valores, que são disponibilizados ao usuário. Por sua vez, a tela de diagnósticos também foi desenvolvida como uma lista,

contendo os erros que o veículo apresenta no momento. Já a tela de histórico foi projetada para ter botões que levam a outras telas, onde são disponibilizadas as informações que estão armazenadas no banco de dados.

Em relação à tela de segurança, ela foi projetada para conter a visualização do mapa da API Google Maps e dois botões que levam, respectivamente, para a ativação da funcionalidade de corte de fluxo de combustível (e acionamento dos faróis) e para uma tela que contém a lista de imagens tiradas do condutor e armazenadas no banco de dados.

A integração com a API Maps do Google Maps foi feita usando o módulo MapView contido na própria ferramenta Expo, mencionada na Subseção 2.11. Já a navegabilidade foi desenvolvida utilizando os componentes *Stack Navigator* e *Bottom Tab Navigator*, representadas respectivamente, pela barra superior e inferior de navegação disponibilizadas no aplicativo.

As informações da tela de performance são obtidas com título, valor e estado de visualização da variável em questão. A lista foi desenvolvida para ser renderizada dinamicamente, ou seja, o aplicativo só renderiza as informações que o carro consegue coletar. A tela de diagnósticos recebe o título, estado, código e uma mensagem de resolução de problema, apenas para os problemas que o carro esteja detectando no momento. As informações da tela de histórico são requisitadas de acordo com qual informação o usuário deseja, tal que esta requisição é feita quando ele clica em um dos botões da tela de histórico. As informações são renderizadas em gráficos e listas que mostram o histórico mensal das informações requisitadas.

4 Testes

Inicialmente, foram feitos diversos testes de forma separada, principalmente com os módulos citados na Seção 2.

Para as placas projetadas, primeiramente foram testadas suas conexões, para conferir se não havia nenhuma falha. Então, seu funcionamento foi testado logo em seguida.

Em relação ao *firmware*, esse foi testado utilizando principalmente um conversor USB-Serial, justamente por se tratar da comunicação entre diversos módulos que usam interface serial. A conexão entre o sistema embarcado e o servidor foi testada exaustivamente, observando as respostas do módulo GSM de acordo com as requisições feitas.

A parte da imagem foi testada em separado, com seus blocos sendo enviados diretamente ao servidor sem o uso do GSM, em uma rede local. Com isso, o servidor deveria poder reconstruir a imagem, para conferir se a lógica da divisão e da concatenação das partes estava funcionando corretamente.

O aplicativo foi testado junto ao servidor, fazendo requisições a ele observando as respostas. Ambos foram sendo reajustados conforme a necessidade.

Integrando todas as partes, os testes foram realizados em um Fiat Palio 2012,

que utiliza o protocolo K-Line. Nesse veículo, foram testados desde a coleta de dados até o acionamento do sistema antifurto.

5 Dificuldades

Em relação ao sistema embarcado, houve um problema ao se fabricar as placas inicialmente. Foram utilizados placas e percloreto de ferro antigos, o que comprometeu a qualidade delas. Trocando esses produtos, novas placas foram feitas. Além disso, houve pequenos erros em relação ao encapsulamento de alguns componentes durante o design das placas no KiCad, mas nada grave, que necessitasse refazê-las. Ainda sobre as placas, foi projetada uma terceira, de interface, que acabou não sendo utilizada, sendo substituída pelo módulo CAN Bus e pelo Arduino.

Sobre a imagem, ocorreram problemas com seu dimensionamento, pois as imagens geradas eram muito grandes para a Tiva e também seria problemáticas para o módulo GSM. Para corrigir, a imagem teve sua resolução diminuída e passou a ser enviada em blocos, além de ter sido utilizado um algoritmo para detecção de faces, para que, caso possível, apenas a parte do rosto do motorista na imagem seja enviada.

Para o aplicativo, a criptografia mostrou-se ser um empecilho. Sendo o React Native modular, quatro módulos para criptografia foram encontrados, mas não foi obtido, inicialmente, êxito na implementação de nenhum desses. De forma semelhante, os dados criptografados no servidor (usa linguagem Java) podiam ser facilmente descriptografados no sistema embarcado (usa linguagem C), mas o contrário não acontecia.

Mesmo com essas dificuldades, a principal foi a falta de experiência no planejamento de um projeto, o que resultou em erros de cronogramas e atividades atrasadas. Ainda sim, foi possível contornar todos os problemas e finalizar o projeto de forma satisfatória.

6 Conclusão

Com a finalização do projeto, STEVE conseguiu compreender todas as áreas pretendidas, servindo como um sistema confiável de telemetria e segurança veicular. O projeto finalizado pode ser visto na Figura 8.

(a) Projeto STEVE no gabinete (b) Projeto STEVE por dentro do gabinete

Figura 8: Projeto STEVE finalizado

O projeto mostrou-se, desde o início, desafiador, abrangendo diversas disciplinas cursadas anteriormente, como sistemas embarcados, banco de dados, criptografia, redes de computadores, sistemas distribuídos, entre outras. Dessa forma, o papel integrador da disciplina de Oficina de Integração 3 foi alcançado com êxito, aplicando os conhecimentos aprendidos durante o curso de Engenharia de Computação.

Além disso, o desenvolvimento do projeto foi uma oportunidade para o aprendizado de tecnologias das quais os integrantes do grupo não tinham muito conhecimento, como o React Native e o padrão de comunicação REST de servidores web.

Em relação ao cronograma, foi percebido que a folga de 30% no tempo previsto para o desenvolvimento das atividades é de grande importância, pois já contempla possíveis atrasos e dificuldades que possam vir a acontecer, sem afetar o restante do projeto diretamente. Entretanto, devido à falta de experiência no planejamento de grandes projetos por parte da equipe, houve uma superestimação no total de horas previsto para a realização do projeto. A Tabela 4 apresenta as horas previstas, a folga e as horas empregadas no desenvolvimento.

Por fim, a Tabela 5 apresenta a comparação entre os gastos previstos e os reais, levando em consideração que muitos dos componentes utilizados já estavam sob a posse da equipe previamente.

Tabela 4: Horas gastas pela equipe no desenvolvimento do projeto.

Atividades	**Horas Estimadas**	**Horas + Folga 30%**	**Horas Gastas**	**Gastas/ Estimadas**	**Gastas/ (Estimadas+Folga)**
Hardware	270	351	239	88,50%	68,00%
Servidor	107	139	123	115,00%	88,25%
Aplicativo	196	254	87	44,29%	34,12%
Documentação	185	241	121	65,41%	50,21%
Total	**757**	**985**	**569**	**75,17%**	**57,77%**

Tabela 5: Comparação entre o orçamento do projeto e o valor pago pela equipe.

Componentes	Quantidade	Preço Un.	Total Previsto	Total Gasto
Módulo CAN mcp2515	2	R$ 29,00	R$ 58,00	R$ 58,00
Módulo GSM	2	R$ 38,95	R$ 77,90	R$ 131,39
Módulo carregador bateria	2	R$ 7,90	R$ 15,80	
Tiva Tm4c1294	2	R$ 90,00	R$ 180,00	
Raspberry Pi Zero	2	R$ 112,00	R$ 224,00	
Relés	2	R$ 12,50	R$ 25,00	
Circuitos Diversos	1	R$ 100,00	R$ 100,00	R$ 113,77
Bateria Lítio 3v7 3000mAh	2	R$ 32,50	R$ 65,00	
Módulo GPS	2	R$ 55,45	R$ 110,90	R$ 61,99
Camera Raspberry Pi v1.3	2	R$ 57,90	R$ 115,80	
Arduino Pro Micro	2	R$ 26,00	R$ 52,00	
Total		R$ 562,20	R$ 1024,40	R$ 365,15

Referências

[1] Which Warning Lights Are Active On Your Dashboard? GOFAR. https://www.gofar.co/car-warning-lights/car-warning-light-symbols-and-indicators/.

[2] Thiago Amâncio. Folha de S. Paulo, 2017. https://www1.folha.uol.com.br/cotidiano/2017/10/1931061-brasil-tem-1-roubo-ou-furto-de-veiculo-a-cada-minuto-rio-lidera-o-ranking.shtml.

[3] Texas Instruments Tiva TM4C1294NCPDT Microcontroller Data Sheet. http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf.

[4] On-board diagnostics. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/On-board_diagnostics.

[5] Vitor Amadeu Souza. Introdução ao CAN - Cerne Tecnologia. https://www.academia.edu/33663024/Introdu%C3%A7%C3%A3o_ao_CAN_Vitor_Amadeu_Souza_Cerne_Tecnologia.

[6] K-line Communication Description. Volkswagen Group of America. https://obdclearinghouse.com/Files/viewFile?fileID=1380.

[7] ARM Cortex-M4 Processor Technical Reference Manual. http://infocenter.arm.com/help/topic/com.arm.doc.100166_0001_00_en/arm_cortexm4_processor_trm_100166_0001_00_en.pdf.

[8] CMSIS-RTOS2 Documentation. ARM Keil. https://www.keil.com/pack/doc/cmsis/RTOS2/html/index.html.

[9] Texas Instruments TivaWare Peripheral Driver Library User's Guide. http://www.ti.com/lit/ug/spmu298d/spmu298d.pdf.

[10] Texas Instruments Tiva C Series TM4C1294 Connected LaunchPad Evaluation Kit EK-TM4C1294XL User's Guide. http://www.ti.com/lit/ug/spmu365c/spmu365c.pdf.

[11] UART GPS NEO-6M User Manual. https://www.terraelectronica.ru/pdf/show?pdf_file=%2Fz%2FDatasheet%2FU%2FUART+GPS+NEO-6M+User+Manual.pdf.

[12] Joe Mehaffey, Jack Yeazel, Sam Penrod e Allory Deiss. gpsinformation.net. https://www.

gpsinformation.org/dale/nmea.htm.

[13] SIM Com - SIM800L Hardware Design. https://img.filipeflop.com/files/download/Datasheet_SIM800L.pdf.

[14] GSM. Wikipedia, the free encyclopedia. https://pt.wikipedia.org/wiki/GSM.

[15] GPRS. Wikipedia, the free encyclopedia. https://pt.wikipedia.org/wiki/General_Packet_Radio_Service.

[16] SIM Com - SIM800L Series AT Command Manual. https://www.elecrow.com/wiki/images/2/20/SIM800_Series_AT_Command_Manual_V1.09.pdf.

[17] Standalone Linear Li-Ion Battery Charger. NanJing Top Power ASIC Corp. https://img.filipeflop.com/files/download/Datasheet_TP4056.pdf.

[18] Microchip - MCP2515 Stand-Alone CAN Controller With SPI Interface. http://www.alldatasheet.com/datasheet-pdf/pdf/166906/MICROCHIP/MCP2515.html?

[19] SPI. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

[20] OBD-II PIDs. Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/OBD-II_PIDs.

[21] DTC (DIAGNOSTIC TROUBLE CODES) LIST. Launch UK. https://www.launchtech.co.uk/support/information/dtc-codes-list/.

[22] OBDII MasterPinOut. Motor Talk. https://www.motor-talk.de/bilder/obd-buchse-sicherung-g75261548/odbiimasterpinout-i208591642.html.

[23] Raspberry Pi Zero 3. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[24] Raspberry Pi Camera Module V2. https://www.raspberrypi.org/products/camera-module-v2/.

[25] Rodrigo Ferreira. REST: Princípios e boas práticas, 2017. Caelum. https://blog.caelum.com.br/rest-principios-e-boas-praticas/.

[26] MySQL. Oracle Corporation. https://dev.mysql.com/doc/.

[27] React Native. Facebook Inc. https://facebook.github.io/react-native/.

[28] Expo Documentation. https://docs.expo.io/versions/latest/.

[29] React Navigation Documentation. https://reactnavigation.org/en/.

[30] OBD9141. iwanders. https://github.com/iwanders/OBD9141.

[31] Face Detection using Haar Cascades. OpenCV: Open Source Computer Vision. https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html.