TELEMETRIA EM VÍDEO PARA CARROS DE COMPETIÇÃO DE ESPORTE A MOTOR

Cristiane Aparecida Lemos
Engenharia de Transportes e Logística
cristianelemos@yahoo.com

Michel Fernando Madeira
Engenharia de Transportes e Logística
michelmadeira@outlook.com

Resumo – As competições esportivas despertam o interesse e chamam a atenção cada vez mais. Em esportes envolvendo carros, é gerada muita curiosidade acerca das informações do veículo. Com o intuito de apresentar essas informações aos telespectadores, o uso da telemetria é indispensável. Este artigo apresenta a implementação de um código de programação em C++ que tem como saída as informações provenientes dos sensores do veículo, convertidas em animação gráfica, de modo a proporcionar detalhes acerca do comportamento do veículo nas transmissões e vídeos dos esportes a motor.

Palavras-chave: programação, C++, telemetria, automobilismo, animação coordenada por dados.

I. INTRODUÇÃO

A palavra “tele” significa remotamente e “metria” é o mesmo que medição. Assim telemetria é uma técnica utilizada para medir coisas a distância. A telemetria pode ser usada para leitura, coleta e armazenamento de dados, além de poder realizar comandos remotos por operadores ou desenvolvedores de sistemas.

Devido à grande aplicabilidade, nas últimas décadas, conquistou diversos setores. Atualmente, já é utilizada nas áreas de agricultura, internet das coisas (IoT), informática, metrologia, gestão de frotas, entre outras. Este artigo apresentará o uso da telemetria em outro âmbito: monitoramento de informações de carros de competição a partir de dados de injeção eletrônica programável.

Um uso muito semelhante é aplicado na Fórmula 1. Nestes há diversos sensores que medem velocidade, deslocamento, temperatura, pressão, aerodinâmica e carga mecânica. As informações são coletadas por um computador de bordo, transformadas em sinais de rádio e enviadas as antenas localizadas nos boxes. Desse modo, os engenheiros podem analisar os dados e projetar peças mais fortes e que atendam a necessidade do carro, tornando-o mais seguro e potente.

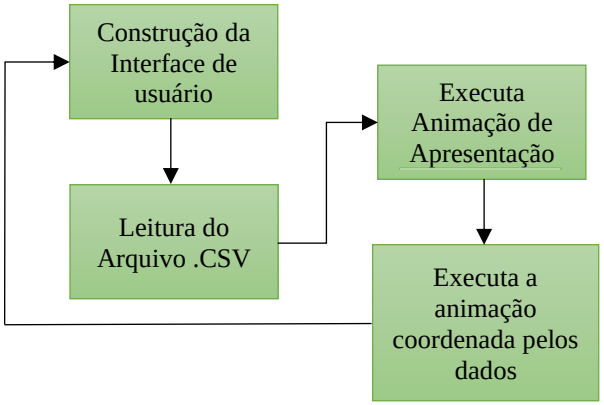
Figura 1 - Telemetria em vídeo nas transmissões de fórmula 1

Neste artigo é detalhada a implementação de um código de programação na linguagem C++ que coleta dados de tempo, RPM (rotações por minuto), MAP (Manifold Absolute Pressure), ou mais conhecida como a pressão do turbo e a posição do acelerador (TPS – Throttle Position Sensor) oriundas de logs de injeção eletrônica programável em formato .CSV (Comma Separated Values).

A IDE framework utilizada foi o Qt Creator, o qual permite o desenvolvimento de recursos e aplicativos para múltiplas plataformas, possibilitando programar em linguagens como C, C++, Java e Python.

Com o Qt Creator é viável criar programas utilizando a linguagem C++ em sintonia com elementos de programação visual, o que pode aumentar a velocidade de programação. Várias empresas utilizam o Qt Creator como, por exemplo, Skype e Google Earth. Além disso, o Qt Creator suporta diversos sistemas operacionais tal como iOS, Android, Windows e Linux.

II. FLUXO DE OPERAÇÃO



A. Construção da Interface de usuário

O construtor da classe *MainWindow*, criada automaticamente pelo Qt e que é pai de todos os elementos da interface, utiliza das classes *QPixmap* e *QIcon* para carregar a imagem de fundo e o ícone do programa respectivamente.

Essas duas imagens são arquivos do tipo PNG, geradas através de ilustrações vetoriais feitas no programa Adobe Illustrator.

Procurou-se modelar a interface de modo a facilitar o uso e tornar o programa intuitivo.

A função membro *setPixmap*, do elemento de interface *Label* é responsável por carregar o *pixmap* gerado. O tamanho deste elemento é configurado para coincidir com o tamanho da janela do programa, assim gerando a imagem de fundo.

Os elementos da interface são desligados de acordo com a fase de execução do programa, e para isso, são utilizadas funções membro do ponteiro *ui*, *setVisible(bool)*.

O tamanho da janela da interface é configurado para fixo para evitar que o usuário altere seu tamanho e desconfigure o design.

Então, quando o programa é aberto, aparecerá na interface a logo, uma caixa de diálogo, um elemento para selecionar o arquivo a ser executado e outro para rodar o programa (só liberado após a leitura do arquivo). No canto superior esquerdo, há a página de ajuda com informações gerais e um link para o *readme* do programa, onde é possível encontrar links para um vídeo exemplo, detalhes de implementação e o repositório do código no *GitHub*.

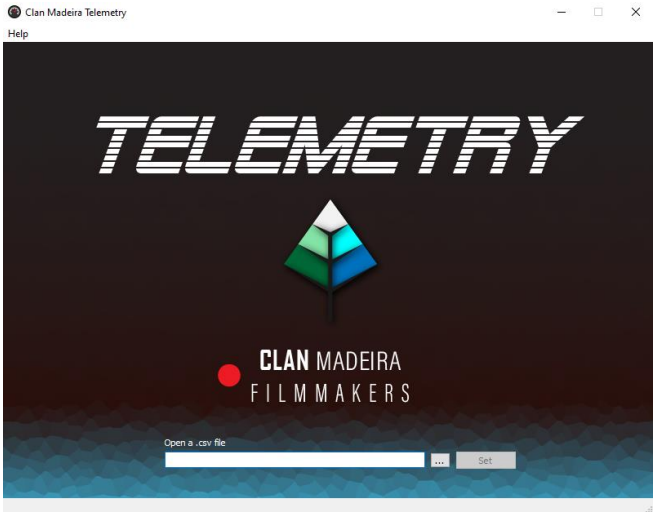


Figura 2 - Interface principal do programa

B. *Leitura do arquivo .CSV*

Os dados do carro são coletados pelos sensores do veículo e armazenados em um arquivo de extensão CSV *(comma separated values*). Estes arquivos podem ser obtidos através das aplicações de visualização de datalogs disponibilizados pelos fabricantes dos módulos de injeção. Dentro deste meio, em território nacional, se destaca a *FUELTECH* e para este trabalho, utilizou-se como referência, os exemplos de logs contidos no aplicativo *FT datalogger*.

A classe *CSV_read* possui os membros private *indexTIME, indexRPM, indexMAP e indexTPS*, todos do tipo inteiro utilizados para armazenar o valor do índice de coluna correspondente as séries de dados time, rpm, map e tps, no arquivo .CSV.

Como atributos *public* possui vetores do tipo *static*. Escolhidos deste tipo para que o gasto computacional em se ler o arquivo seja feito uma única vez, e os vetores armazenadores estejam disponíveis independentes de instanciação, uma vez que eles são utilizados em pelo menos 3 escopos diferentes do programa.

O construtor da classe *CSV_rea*d irá receber o caminho do arquivo usando a classe *QFileDialog* ou diretamente através do campo *LineEdit*.

```
"Linha de Tempo";"RPM";"Lambda";"MAP";"Pressão Combustível";
"0";"4530";"0.701774796195652";"-0.504064873964089";"3.75";"
"0.02";"4470";"0.681765879755435";"-0.551813924378453";"3.70
"0.04";"4380";"0.672106402853261";"-0.567730274516575";"3.67
"0.06";"4350";"0.672106402853261";"-0.480190348756906";"3.82
"0.08";"4260";"0.682455842391304";"-0.249403271754144";"3.89
"0.1";"4380";"0.690735394021739";"-0.0504488950276244";"4.05
"0.12";"4560";"0.710744310461957";"-0.00269984461325967";"3.
"0.14";"4830";"0.727993376358696";"0.00525833045580115";"4.0
"0.16";"5040";"0.741792629076087";"-0.0345325448895027";"4.0
"0.18";"5280";"0.759731657608696";"-0.169821521063536";"3.87
"0.2";"5490";"0.760421620244565";"-0.336943197513812";"3.798
"0.22";"5550";"0.748692255434783";"-0.472232173687845";"3.75
"0.24";"5520";"0.729373301630435";"-0.559772099447514";"3.66
"0.26";"5460";"0.710744310461957";"-0.615479324930939";"3.64
"0.28";"5370";"0.690735394021739";"-0.647312025207182";"3.61
"0.3";"5280";"0.681765879755435";"-0.671186550414365";"3.627
```

Figura 3 - Exemplo usado como referência para a construção da classe *CSV_read*

A primeira linha é lida e armazenada em uma string. É feita a retirada de todas as aspas duplas. Posteriormente, quando encontrado o caractere “;” através do método split das *QStrings*, é realizada a separação da linha em uma lista de *Strings*.

Em seguida, percorre-se a lista buscando o índice das variáveis de interesse (RPM, MAP e TPS), sem distinguir letras maiúsculas de minúsculas, e testando também outras possibilidades de nomenclatura para estas informações. Caso encontrado, os atributos da classe do tipo *isfound*, serão setadas para verdadeiro para cada série. Caso não será solicitado ao usuário mudar o nome das variáveis através do *FT datalogger*.

Isto deve ocorrer, visto que a estrutura de dados varia de carro para carro, por exemplo, alguns casos tinham a variável como “rpm” outros como “rotação”. Preparadores diferentes também nomeiam os sensores ao seu gosto.

No entanto, o código foi concebido para ser bastante versátil, operando com a maioria dos arquivos disponíveis, e quando não, orientando o usuário a adaptá-lo.

Não é necessário buscar o índice da variável tempo, dado que essa será sempre a primeira coluna do arquivo (*indexTime=0*).

Dispondo da lista de strings formada, utiliza-se o método *push_back*, para formar linha a linha, as variáveis de armazenamento da classe.

Nessa etapa, a variável “Time” é transformada em float e multiplicada por mil, tal multiplicação ocorre, pois a unidade de tempo utilizada pelo Qt é comumente milisegundos. O mesmo acontece para a variável “RPM”, porém essa é convertida para o tipo inteiro, as variáveis MAP e TPS são convertidas para float.

É gerado também texto a partir da variável TPS.

Inicialmente também planejou-se coletar a variável *two step*. Trata-se de um pico gerado no log quando o piloto aciona o botão correspondente no painel de controle da injeção. Esse pico é usado como referência para a leitura dos logs. O piloto pode acionar o *two step* quando o carro arrancou de fato, ou após completar a primeira volta, por exemplo.

No âmbito deste trabalho, este sinal seria utilizado para auxiliar na sincronização com o vídeo de pilotagem. No entanto esse recurso não foi implementado por motivos que serão discutidos na conclusão deste artigo.

C. *Classes RPM, MAP e TPS e GasPedal*

Nos arquivos *RPMGauge.h* e *boostgauge.h e gaspedal.h* encontram-se os protótipos das funções utilizadas para exibir os mostradores, o tacômetro para RPM e o relógio de pressão do turbo, além do pedal de acelerador. Cada um desses arquivos possui atributos com ponteiros para a classe *QGraphicsPixmapItem*, utilizada para processar imagens 2d. Nos arquivos *RPMGauge.cpp* e *boostgauge.cpp* e *gaspedal.cpp*. situa-se o desenvolvimento das funções.

Todas as classes gráficas operam da mesma maneira e tem basicamente o mesmo código.

A classe *QPixmap* carrega a imagem do mostrador, com endereço referente a pasta de instalação do programa, isso ocorre para que, caso o programa seja instalado em outro computador, não ocorra nenhum erro ao tentar localizar a imagem, visto que a estrutura de dados varia de computador para computador.

Em seguida, é criado um ponteiro para *QGraphicsItem* a partir do *pixmap* gerado anteriormente. A função *setTransformOriginPoint* é responsável por realocar o ponto de referência das transformações para pixels convenientes. Tal realocação é necessária, dado que, o Qt creator possui como ponto de referência o pixel esquerdo superior (0,0) como origem e desse modo não seria possível realizar a movimentação dos elementos corretamente.

setTransformationOriginPoint é a função encarregada de realocar o ponto de referência para o eixo de rotação do ponteiro por exemplo, e *setPos* posiciona o ponteiro no centro do relógio, assim é possível fazê-lo girar em torno do próprio eixo exibindo corretamente as informações coletadas.

A classe *gaspedal* destinada a exibir as informações do pedal seguem a mesma lógica, entretanto não há a rotação e sim escalonamento da imagem para realizar sua animação.

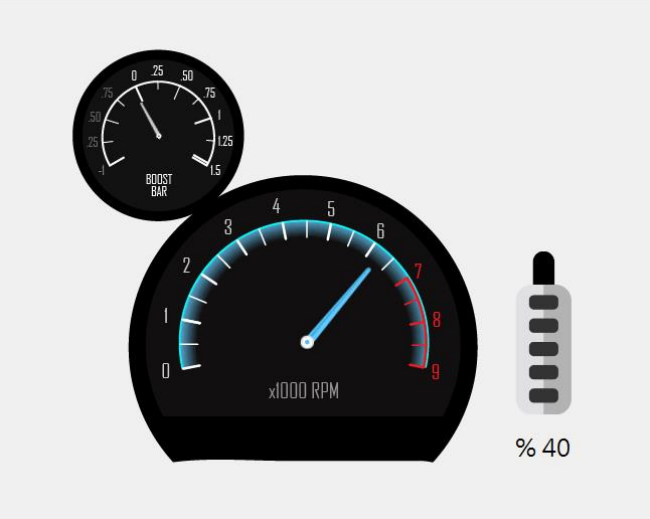


Figura 4 - O painel de mostradores

D. *Animação inicial*

Nesta parte da execução do programa o arquivo CSV já está lido e o botão "*Set*" é habilitado ao usuário.

É então executada a animação inicial como quando você liga um carro esportivo. Os ponteiros percorrem toda a extensão dos relógios demonstrando um setup inicial e que o sistema está pronto e operante.

Um timer que executa um *SLOT* é chamado, aproveitando-se do vetor Time gerado pela leitura do arquivo, realizando a rotação dos ponteiros.

As variáveis estáticas *animationStateRpm* e *animationStateBoost* são responsáveis por controlar a animação, e um laço de decisões também é responsável por identificar quando a animação acaba e habilita o botão para a próxima fase do programa.

E. *Animação Coordenada por dados*

Esta fase do programa, opera da mesma forma que a animação inicial. No entanto, o *Qtimer* é chamado tantas vezes quanto o número de linhas do arquivo.

Além disso, a rotação do ponteiro não é incrementada em intervalos fixos, mas encontrando-se a função matemática de primeiro grau que coordena a animação.

Para a rotação do tacômetro: Pontos p1(0,-47) e p2(9000,159) encontrando-se:

$$rot_{ponteiro\ rpm} = (0.02288xRPM) - 47$$

Para a rotação do ponteiro do relógio MAP: p1(-1,-65) e p2(1.5,175) encontrando-se:

$$rot_{ponteiro\ boost} = (96xMAP) + 31$$

Para o escalonamento do pedal de acelerador:

$$escala_{gaspedal} = (-0.002xTPS) + 0.6$$

É mostrado junto ao pedal do acelerador o texto da String TPS.

Ao final do arquivo, é oferecido a opção de reset para o usuário.

III. DISCUSSÃO

A. *Quanto as classes gráficas*

Apesar do Qt oferecer recursos de desenho em 2D, optou-se por apenas carregar imagens no formato PNG deixando o desenho para ambientes mais favoráveis a esse assunto. Adobe Illustrator no caso. Fazer qualquer coisa no Qt exige recompilação para mostrar o resultado e assim sendo, o programa é pouco prático para este tipo de tarefa.

Durante a modelagem inicial do programa, pensou-se em constituir uma classe mãe HUD, na qual todos os elementos dos mostradores fossem filhas e poderiam ser implementados, conceitos de herança e polimorfismo. No entanto, não se observou caminhos no qual esses conceitos pudessem ser utilizados acredita-se pela falta de experiência com programação orientada ao objeto.

O grupo tem consciência que as classes poderiam estar mais bem relacionadas e utilizando mais conceitos de orientação ao objeto, muito embora também reconhece que foram necessários muitos desses conceitos para se conseguir interpretar a documentação, e encontrar as soluções dentro do Qt.

B. *Quanto ao Controle da interface*

Durante a construção do código percebeu-se que existem maneiras mais eficientes para administrar a diagramação dos elementos da interface. Trata-se da classe *QState*, que é capaz de salvar estados e dessa forma seria uma solução mais elegante e tornaria o código mais legível também.

Para esta implementação de pequeno porte, ligar e desligar os elementos não foi trabalho tão extenso, mas em um programa com mais elementos e mais estados de execução o uso de *QState* torna-se essencial.

C. *Quanto as animações*

Também se descobriu durante o desenvolvimento que existem classes específicas para lidar com animações. Normalmente utilizar este tipo de implementação torna o código mais eficiente, legível e seguro.

Poderiam ser utilizados os recursos de Sinais e Slots conectados também, para identificar uma mudança na rotação do ponteiro uma vez que a classe *QGraphicsScene* identifica automaticamente mudanças em seus elementos e atualiza a exibição automaticamente.

O uso de variáveis estáticas também poderia ser dispensado com o uso de funções lambda. Fomos compelidos a implementar assim, porque SLOTS não podem receber argumentos em suas chamadas.

D. *Análises de código através da IDE*

Foram realizadas as análises: *Memcheck*, *Valgrind*, *Clang-Tidy*. Nenhuma delas, apresentou qualquer sinal de erro, ou indício de vazamento de memória

Com exceção da *CLang-Tidy*, indicando: "warning: Slots named on_foo_bar are error prone [clazy-connect-by-name]", indicando propensão a erros de alguns *SLOTS.*

Observou-se queda de performance na execução do programa depois de 30 segundos de execução.

IV. CONCLUSÃO

O programa implementado atende ao seu propósito de concepção para fins didáticos, mas infelizmente para um uso fora do contexto de sala de aula, carece de implementação do recurso de captura da animação, transformando-a em um vídeo de fundo transparente, que poderia ser incorporada aos vídeos de pilotagem.

O Qt não oferece meios práticos para se capturar uma animação exibida na *QGraphicView*. Existem classes que gravam vídeos, acessam a webcam mas nenhuma disponível para capturar elementos em 2d em vídeo.

Avaliou-se a também carregar o vídeo diretamente no programa, mas a ferramenta de captura ignora qualquer "overlay" que pudesse ser inserido ao vídeo.

Após extensa pesquisa, cogitou-se a possibilidade de se capturar imagens estáticas e juntá-las através do uso da biblioteca externa FFMEG. Contudo, tal implementação presumiu-se custosa diante das circunstâncias e prazos de entrega do trabalho.

Além disso, verificou-se que as animações não ficaram tão fluídas e que somando ao fato da queda de performance na execução do programa, fez a equipe concluir que a telemetria deva ser implementada em plataforma mais orientada a animações, para viabilizar amplamente o seu uso.

A equipe fez sondagem no software da mesma desenvolvedora Qt 3d Studio, o qual oferece nativamente uma timeline, eliminado a dimensão de dificuldade "tempo". Além disso, o Qt 3d oferece feedback de design em tempo real, sem a necessidade de compilação, ainda que estranhamente não contenha também, recursos de renderização para arquivos em formato de vídeo.

Dessa forma, valida-se absolutamente a experiência de incursão nas bibliotecas do Qt. Trabalhar com classes gráficas e a dimensão tempo, adicionaram camadas de dificuldade ao trabalho, e foi possível expandir os conceitos com relação a programação e a orientação ao objeto.

REFERÊNCIAS

[1] Telemetria**:** o que é e como funciona? **OP Services**. Disponível em: https://www.opservices.com.br/telemetria/ Acesso em: 19 nov. 2020

[2] Telemetria veicular**:** o que é? Entenda os tipos e como funciona. **Cobli Bolg**. Disponível em: https://www.cobli.co/blog/telemetria-veicular/ Acesso em: 19 nov. 2020

[3] Capítulo 1. Introdução O que é Qt (Software)? **Agostinho Brito Jr.** Disponível em: https://agostinhobritojr.github.io/tutorial/qt/cha.introducao.htmlAcesso em: 19 nov. 2020

[4] Entenda o que é o formato CSV e como importar e exportar esses arquivos. **Blog Rock Content**. Disponível em: https://rockcontent.com/br/blog/csv/ Acesso em: 25 nov. 2020