SISTEMA DE TELEMETRIA PARA KART

ALUNO: Diogo Holanda Dantas – RA: 2081726/2

Orientadora: Profª. M.C. Maria Marony Sousa Farias

Brasília - DF, dezembro de 2012

SISTEMA DE TELEMETRIA PARA KART

por

Diogo Holanda Dantas

Monografia apresentada à Banca Examinadora do curso de Engenharia de Computação da FATECS – Faculdade de Tecnologia e Ciências Sociais Aplicadas – Centro Universitário de Brasília como requisito parcial para obtenção do título de Engenheiro de Computação.

Brasília – DF, dezembro de 2012.

Banca Examinadora

_____

Prof. Luciano Henrique Duque

_____

Prof. Miguel Arcanjo Telles

_____

Prof. Fernando Chagas

AGRADECIMENTOS

Ao meu pai, pelo auxílio no desenvolvimento da solução nos produtos que são de seu conhecimento e na descrição dos mesmos.

À minha família pelo apoio e incentivo no desenvolvimento do projeto. À minha namorada pela compreensão e apoio.

Aos meus amigos que me auxiliaram com materiais e conhecimentos para o desenvolvimento e testes do projeto. Em especial ao José Carlos pelos ensinamentos do funcionamento dos sensores, ao Flávio Cardoso pelo compartilhamento de conhecimentos e materiais em comum em nossos projetos.

A todos os professores e mestres do curso de Engenharia de Computação que compartilharam seus conhecimentos. À minha orientadora Maria Marony pelo apoio ao projeto. Ao Francisco Javier, pelo incentivo e confiança no meu trabalho.

Ao Lucas Oliveira e ao Rodrigo Silva por me emprestarem o kart para testes e demonstração.

Por fim, a todos que incentivaram, apoiaram e contribuíram de alguma forma para a realização deste trabalho.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de telemetria para kart, que permite à equipe monitorar o desempenho de um piloto durante uma corrida, além de permitir uma análise comparativa posterior através de gráficos. A solução desenvolvida foi implementada utilizando os seguintes componentes eletrônicos: um transdutor LM35, um giroscópio LPY530AL, dois potenciômetros lineares de 100K, um sensor de velocidade e um microcontrolador atmel ATmega 328 para gerenciar os sensores. Os dados obtidos são enviados pelo microcontrolador, via WI-FI para o computador que está conectado na mesma rede.

Com os dados obtidos na pista sendo transmitidos para o computador em tempo real, a equipe pode orientar o piloto a melhorar sua próxima passagem. Esses dados são armazenados em um banco de dados Oracle 10g Express Edition para que seja gerada uma análise off-line.

O microcontrolador foi programado utilizando a linguagem de programação Arduino. O software utilizado para a criação de gráficos em tempo real foi o TIBCO RTView, para a criação de análises off-line foi o TIBCO Spotfire e para transformação de dados, inserção no banco de dados e publicação de mensagens foi o TIBCO BusinessWorks, todos possuem programação visual, utilizando drag-and-drop. As mensagens para o TIBCO RTView são enviadas através do TIBCO Rendezvous.

Palavras-chave:

Arduino; microcontrolador; giroscópio; Oracle; Wireless; Shield WI-FI 802.11b; Seeed Studio; LM35; Banco de Dados; RTView; BusinessWorks; Rendezvous; mensagem; Spotfire.

ABSTRACT

This study presents a development of a Telemetry System for Kart, which allows the team to monitor the driver's performance during the race and allows a graphical posterior comparative analysis. The developed solution was built using the following electronics components: a LM35 transducer, an LPY530AL gyroscope, two 100K linear potentiometers, a speed sensor and an Atmel ATmega 328 microcontroller to manage the sensors. The obtained data are sent from a microcontroller via Wi-Fi to computer connected at the same network.

The team can guide the driver to do the next lap better with the obtained data at trace being transmitted to a computer in real time. This data is stored into an Oracle 10g Express Edition database in order that generates an off-line analysis.

The microcontroller was programmed using Arduino programming language. The software used to create real time graphics was TIBCO RTView, to create offline analysis was TIBCO Spotfire and to transform data, insert into database and to public messages was TIBCO BusinessWorks. All of then uses visual programming with drag-and-drop system. The messages to TIBCO RTView are sent by TIBCO Rendezvous.

Key-words:

Arduino; microcontroller; gyroscope; Oracle; Wireless; Shield WI-FI 802.11b; Seeed Studio; LM35; Database; RTView; BusinessWorks; Rendezvous; message; Spotfire

SUMÁRIO

SISTEMA DE TELEMETRIA PARA KART i
AGRADECIMENTOS ii
RESUMO iii
ABSTRACT iv
LISTA DE TABELAS xiii
1- INTRODUÇÃO 14
1.1 MOTIVAÇÃO 14
1.2 PROBLEMA 15
1.3 OBJETIVOS 15
1.4 ESTRUTURA 16
2- REFERENCIAL TEÓRICO E TECNOLÓGICO 17
2.1 MOTOR 17
2.1.1 MOTOR DOIS TEMPOS 18
2.1.2 MOTOR QUATRO TEMPOS 19
2.2 TELEMETRIA 22
2.3 HARDWARE 25
2.3.1 SENSORES 25
2.3.1.1 SENSOR DE TEMPERATURA - LM35 25
2.3.1.2 SENSOR DE PROXIMIDADE DE EFEITO HALL 26
2.3.2 MICROCONTROLADOR 26
2.3.2.1 ATMEGA328 26
2.3.2.2 ARDUINO 27
2.3.3 WI-FI 31
2.3.3.1 WIFI SHIELD – WIFLY RN-171 31
2.3.4 POTENCIÔMETRO 32
2.3.5 GIROSCÓPIO 33
2.4 SOFTWARE 35
2.4.1 BANCO DE DADOS ORACLE 35
2.4.1.1 TABELAS 36
2.4.1.2 ÍNDICES 36
2.4.1.3 Structured Query Language (SQL) 36
2.4.1.4 PL/SQL e Java 37

2.4.2 TIBCO RTVIEW 38
2.4.3 TIBCO BUSINESSWORKS 41
2.4.4 TIBCO SPOTFIRE 43
2.4.5 TIBCO RENDEZVOUS 45
3- IMPLANTAÇÃO 47
3.1 HARDWARE 47
3.1.1 TRANSFORMAÇÃO DE DADOS LIDOS PELOS SENSORES 49
3.1.2 ENVIO DE DADOS PELA WI-FI 50
3.2 SOFTWARE 50
3.2.1 INTERFACE DE CADASTRO 51
3.2.2 BANCO DE DADOS 58
3.2.3 INTEGRAÇÃO 61
3.2.3.1 RECURSOS COMPATILHADOS 61
3.2.3.1.1 CONEXÕES 61
3.2.3.1.2 XSD – XML SCHEMA DEFINITION 62
3.2.3.1.3 DATAFORMAT 65
3.2.3.1.4 SHARED VARIABLE 66
3.2.3.2 INSERÇÃO NO BANCO DE DADOS 66
3.2.3.2.1 RECEBIMENTO DE DADOS CADASTRAIS 68
3.2.3.2.2 ARMAZENAMENTO DOS DADOS DO KART 72
3.2.3.3 RECEBIMENTO E TRANSFORMAÇÃO DE DADOS 73
3.2.3.4 INICIO E FIM DA TELEMETRIA 74
3.2.4 GRÁFICOS ONLINE 76
3.2.5 GRÁFICOS OFFLINE 82
4- RESULTADOS OBTIDOS 88
4.1 SISTEMA DE CADASTRO 88
4.2 INTEGRAÇÃO E PAINÉIS DE ANÁLISE 95
4.3 DADOS DOS SENSORES 104
4.4 INTEGRAÇÃO DO PROTÓTIPO COM O COMPUTADOR 105
4.5 INSTALAÇÃO NO KART 106
5- CONCLUSÕES 109
REFERÊNCIAS BIBLIOGRÁFICAS 111
ANEXO 1 – BIBLIOTECA WIFLY.H 115
ANEXO 2 – BIBLIOTECA HARDWARE.H 116

ANEXO 3 – BIBLIOTECA DEBUG.H 117
APÊNDICE A – WSDL CONCRETO UTILIZADO PARA REQUISIÇÕES SOAP 118
APÊNDICE B – WSDL ABSTRATO UTILIZADO NO WEBSERVICE 121
APÊNDICE C – PROGRAMA DO ARDUINO 122
APÊNDICE D – SQL UTILIZADO PELO SPOTFIRE 124
APÊNDICE E – PROGRAMA DE TESTES DO PROTÓTIPO 125
APÊNDICE F – CÓDIGO CORRIGIDO DO ARQUIVO WIFLY.CPP 127
APÊNDICE G – FLUXO DO PROJETO 132

LISTA DE FIGURAS

Figura 2.1 – Fases do pistão 17
Figura 2.2 – Primeiro tempo do ciclo de dois tempos 18
Figura 2.3 – Segundo tempo do ciclo de dois tempos 19
Figura 2.4 – Primeiro tempo - Admissão 20
Figura 2.5 – Segundo tempo - Compressão 20
Figura 2.6 – Terceiro tempo - Explosão 21
Figura 2.7 – Quarto tempo - Exaustão 21
Figura 2.8 – Míssil em testes enviando sinal de telemetria 22
Figura 2.9 – Barrichello na sala de telemetria da Ferrari 24
Figura 2.10 – Telemetria de uma volta em Mônaco 24
Figura 2.11 – LM35 25
Figura 2.12 – Sensor de Proximidade de Efeito Hall 26
Figura 2.13 – Arduino Duemilanove 27
Figura 2.14 – Interface de Desenvolvimento Integrado do Arduino 30
Figura 2.15 – Interface da Wifi Shield 32
Figura 2.16 – Potenciômetro Linear 33
Figura 2.17 – Diagrama de bloco do giroscópio LPY530AL 34
Figura 2.18 – Pinos do giroscópio LPY530AL 34
Figura 2.19 – Exemplo de Dashboard do RTView 40
Figura 2.20 – Exemplo de processo 42
Figura 2.21 – Exemplo de XSD 43
Figura 2.22 – Exemplo de análise no Spotfire 45
Figura 3.1 – Esquema das ligações do Arduino 48

Figura 3.2 – Diagrama de Blocos do Hardware 48
Figura 3.3 – Diagrama de Blocos do Software 51
Figura 3.4 – Código HTML da página home 52
Figura 3.5 – Design da página home 52
Figura 3.6 – Codificação da página kart 53
Figura 3.7 – Design da página kart 53
Figura 3.8 – Codificação da página piloto 55
Figura 3.9 - Design da página piloto 56
Figura 3.10 – Codificação da página corrida 57
Figura 3.11 – Design da página corrida 58
Figura 3.12 – Modelo Lógico do Banco de Dados 59
Figura 3.13 – Tabela Karts 59
Figura 3.14 – Tabela Pilotos 60
Figura 3.15 – Tabela Corridas 60
Figura 3.16 – Tabela Dados 60
Figura 3.17 – XSD dos dados de piloto 63
Figura 3.18 – XSD dos dados de erro 63
Figura 3.19 – XSD dos dados de cadastro de kart 64
Figura 3.20 – XSD dos dados de telemetria do kart 64
Figura 3.21 – XSD dos dados de cadastro de corrida 65
Figura 3.22 – XSD dos dados de resposta do *WebService* 65
Figura 3.23 – Estrutura do projeto de integração 67
Figura 3.24 – Criação da página de resposta 68
Figura 3.25 – Fluxo do processo HTTPReceiverProcess 69
Figura 3.26 – Fluxo do processo de erro 69
Figura 3.27 – Operações do WebService 70

Figura 3.28 – Processo de inserção de kart 70
Figura 3.29 – Processo de inserção de piloto 71
Figura 3.30 – Processo de inserção de corrida 71
Figura 3.31 – Processo de geração de código identificador de corrida 71
Figura 3.32 – Atividade de inserção de corrida 72
Figura 3.33 – Fluxo do processo InsertDB 73
Figura 3.34 – Código de inserção de dados coletados 73
Figura 3.35 – Processo de recebimento de dados por TCP 74
Figura 3.36 – Processo de início e término da telemetria 75
Figura 3.37 – Configuração do RV para o RTView 76
Figura 3.38 – Recebimento de dados do medidor e do gráfico de velocidade 77
Figura 3.39 – Recebimento de dados da escala e do gráfico de temperatura 78
Figura 3.40 – Recebimento de dados da escala e do gráfico de movimentação do volante 78
Figura 3.41 – Recebimento de dados da escala e do gráfico do acelerador 79
Figura 3.42 – Recebimento de dados da escala e o gráfico do freio 80
Figura 3.43 – Painel de análise online 81
Figura 3.44 – Tela inicial do programa de telemetria 82
Figura 3.45 – Colunas e Joins criados no TIBCO Spotfire 83
Figura 3.46 – Information Link criado no TIBCO Spotfire 83
Figura 3.47 – Página Inicial do Spotfire 84
Figura 3.48 – Página de análise dos dados de movimentação do volante 85
Figura 3.49 – Página de análise dos dados de posição dos pedais de acelerador e freio 85
Figura 3.50 – Página de análise dos dados de velocidade do kart 86
Figura 3.51 – Página de análise dos dados de temperatura do motor 87
Figura 4.1 – Dados informados na página de cadastro de piloto 89
Figura 4.2 – Processo recebendo os dados de piloto e invocando o WebService 89

Figura 4.3 – Dados do piloto sendo inseridos no banco de dados 90
Figura 4.4 – Dados do piloto presentes no banco de dados 90
Figura 4.5 – Dados informados na página de cadastro de karts 91
Figura 4.6 - Processo recebendo os dados de kart e invocando o WebService 91
Figura 4.7 - Dados do kart sendo inseridos no banco de dados 92
Figura 4.8 - Dados do kart presentes no banco de dados 92
Figura 4.9 - Dados informados na página de cadastro de corrida 93
Figura 4.10 - Processo recebendo os dados de corrida e invocando o WebService 94
Figura 4.11 - Dados da corrida sendo inseridos no banco de dados 94
Figura 4.12 - Dados da corrida presentes no banco de dados 94
Figura 4.13 – Processo de geração de massa de teste 95
Figura 4.14 – Iniciando a telemetria 95
Figura 4.15 – Recebimento da informação do início da telemetria e resposta com o nome da corrida 96
Figura 4.16 – Painel de gráficos chamado com as informações da corrida em andamento 97
Figura 4.17 – Dados recebidos e tratados no processo de integração 97
Figura 4.18 – Dados exibidos no painel de análise online 98
Figura 4.19 – Arquivo texto com os dados gravados 98
Figura 4.20 – Recebimento da informação do término da telemetria, resposta e gravação no banco 99
Figura 4.21 – Painel atualizado com o término da telemetria 100
Figura 4.22 – Processo de inserção de dados no banco 100
Figura 4.23 – Dados inseridos no banco 101
Figura 4.24 – Testes da análise offline 101
Figura 4.25 – Gráfico com informações da movimentação do volante 102
Figura 4.26 – Gráficos com informações da posição dos pedais de aceleração e frenagem 102
Figura 4.27 - Gráfico com informações da velocidade do kart 103

Figura 4.28 – Gráfico com informações da temperatura do motor 104

Figura 4.29 – Dados dos sensores exibidos via porta serial 104

Figura 4.30 – Conexão TCP estabelecida 105

Figura 4.31 – Dados recebidos do kart são exibidos no painel online 106

Figura 4.32 – Sensores instalados 107

Figura 4.33 – Dados do kart recebidos 107

Figura 4.34 – Dados do kart exibidos nos gráficos online 108

LISTA DE TABELAS

Tabela 2.1 – Características do Arduino 28

Tabela 2.2 – Especificações Wifi Shield 31

1- INTRODUÇÃO

1.1 MOTIVAÇÃO

A motivação para esse projeto é a elaboração de um protótipo acadêmico de coleta de informações, para análise online disponibilizada em um computador, dos seguintes parâmetros: velocidade, posição dos pedais de freio e acelerador, temperatura do motor e movimentação do volante. Com os dados obtidos online, obter uma maior durabilidade das peças e um ganho de desempenho dos pilotos, possibilitando a equipe instruir o piloto como melhorar suas voltas, instruí-lo a retornar para o Box para evitar uma quebra ou a mudar a pilotagem para preservar o equipamento.

Um kart possui uma estrutura simples, um *chassis* monoposto sem *cockpit*, quatro rodas e um motor com apenas um pistão (COMMISSION INTERNATIONALE DE KARTING, 2012), mas sua regulagem é pessoal e pode ser complexa. Uma configuração pode funcionar muito bem para um piloto, porém para outro pode não ter o mesmo desempenho e um aumento de desgaste de peças. Essas diferenças são ainda mais perceptíveis e influentes em categorias superiores como Fórmula 1, Fórmula Indy, entre outras. Com isso, é de extrema importância o uso de um sistema de telemetria tanto para conquistar uma vitória quanto para evitar quebras.

O kart é a categoria de escola para o automobilismo. Grandes nomes do automobilismo mundial, como Rubens Barrichello, Tony Kanaan, Michael Schumacher e outros, começaram suas carreiras nessa categoria e até hoje se divertem organizando mini campeonatos de kart durante suas férias. No kart, como em todas as outras categorias, uma corrida pode ser ganha no detalhe, no meio segundo de vantagem. Essa vantagem pode ser obtida de diversas maneiras e nas mais variadas situações como: um ajuste de *chassis* pode otimizar a velocidade na reta ou melhorar o contorno de uma curva, uma regulagem de admissão de combustível e ar pode aumentar a velocidade ou aumentar a economia para ganhar na estratégia de reabastecimento, uma limitação ideal do giro do motor para obter um melhor equilíbrio entre alta potência e baixo risco de quebra, a pressão ideal de pneus para um baixo desgaste e melhor desempenho. Todos esses ajustes são variáveis que dependem do estilo de pilotagem do piloto e essa informação também é obtida por meio da telemetria. Essas

informações são analisadas volta a volta e, com elas, é possível corrigir alguns pontos da pilotagem e fazer as regulagens necessárias no retorno do piloto ao Box.

O protótipo construído coleta dados, posição dos pedais, movimentação do volante, velocidade e temperatura do motor, e exibe gráficos online e off-line para ajustes e comparações entre dois ou mais pilotos.

1.2 PROBLEMA

A grande maioria dos sistemas de telemetria atuais exibem os dados apenas para o piloto e esses dados não contribuem o suficiente para um ajuste e melhoria da forma de pilotagem.

Informações como posição dos pedais e movimentação do volante não são informados pelos sistemas atuais e esses dados são muito importantes para a melhoria no desempenho do piloto. Com esses dados e a devida análise da equipe, o piloto pode ser informado das mudanças que deve fazer para ganhar tempo durante a volta.

1.3 OBJETIVOS

O projeto tem como finalidade obter e processar dados, para obter um aumento de durabilidade de peças e um ganho de desempenho dos pilotos.

São objetivos específicos do projeto:

- Capturar, exibir e armazenar as posições dos pedais de aceleração e freio;
- Capturar, exibir e armazenar a movimentação do volante;
- Capturar, exibir e armazenar a temperatura do motor;
- Capturar, exibir e armazenar a velocidade instantânea do kart;
- Construir um *dashboard* para exibir os gráficos online;
- Construir processo de transformação e persistência dos dados;
- Construir um *dashboard* para análise off-line.

1.4 ESTRUTURA

Este trabalho possui cinco capítulos, a introdução (Capítulo 1) e os outros assim dispostos:

- Capítulo 2 – Referencial Teórico e Tecnológico – abrange as principais definições e conceitos desse projeto, como motor, telemetria, sensores, microcontrolador, Wirelles, Banco de Dados, Linguagem C, softwares (RTView, BusinessWorks, Spotfire, Rendezvous);
- Capítulo 3 – Implantação – abrange o detalhamento do projeto, construção do protótipo, ferramentas utilizadas, códigos desenvolvidos, criação do banco de dados;
- Capítulo 4 – Resultados obtidos – descreve todos os testes e resultados obtidos;
- Capítulo 5 – Conclusões – descreve as conclusões do projeto e sugestões para projetos futuros.

2- REFERENCIAL TEÓRICO E TECNOLÓGICO

2.1 MOTOR

O motor é uma peça que converte o calor produzido pela combustão do carburante em energia mecânica. O carburante, que normalmente é uma mistura de gasolina e ar, é queimado no interior do cilindro. [MARTINS, 2006]

A estrutura de um motor deve suportar pressão e temperaturas elevadas, com isso deve ser extremamente rígida. É constituído basicamente por duas partes ligadas por meio de parafusos: o cabeçote, parte superior, e o bloco do motor, parte inferior, que normalmente são feitos em ferro fundido. [MARTINS, 2006]

O motor possui três estágios: os pontos mortos e o curso. Durante o movimento do pistão no interior do cilindro, ele atinge dois pontos extremos, o Ponto Morto Alto e o Ponto Morto Baixo. A distância entre esses ponto é chamada de Curso. Conforme ilustra a Figura 2.1. [RCMASTERS, 2012]

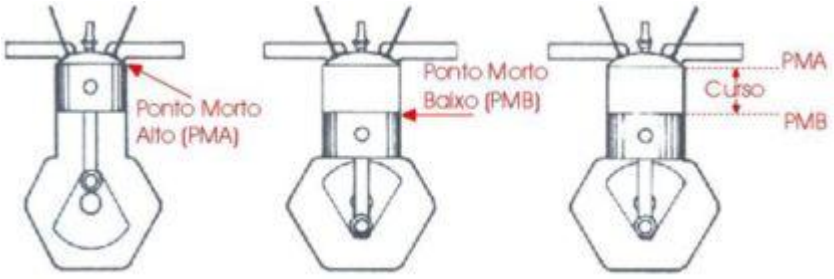


Figura 2.1 – Fases do pistão

[RADIOCONTROLADO, 2012]

O tempo do motor consiste no conjunto de fases que ocorrem durante o tempo em que o pistão percorre um curso. [RCMASTERS, 2012]

2.1.1 MOTOR DOIS TEMPOS

O motor que possui dois tempos no ciclo é simples. Possui poucas peças moveis. Comparado com motores quatro tempos, ele é mais potente e mais leve. Não possui as válvulas como no motor quatro tempos, o pistão funciona como válvula deslizante abrindo e fechando as janelas de admissão e escape. [MARTINS, 2006]

Primeiro tempo (estágio): Ao subir o pistão produz uma rarefação no cárter e a mistura no cilindro é comprimida. Quando o pistão chega ao ponto morto alto, bem próximo ao cabeçote, dá-se a ignição e a combustão da mistura, é nesse momento que ocorre a admissão da nova mistura no cárter, devido à subida do pistão, conforme ilustrado na Figura 2.2. [MARTINS, 2006]

Figura 2.2 – Primeiro tempo do ciclo de dois tempos

[RADIOCONTROLADO, 2012]

Segundo tempo (estágio): Com a expansão dos gases da combustão, o pistão é empurrado comprimindo a mistura do cárter. Ao chegar ao seu ponto mais baixo, comumente chamado de ponto morto baixo, o pistão abre a janela de exaustão permitindo a saída dos gases queimados durante a combustão. A seguir, a janela de transferência é aberta e assim, a mistura é comprimida no cárter empurra os gases queimados ao entrar no cilindro. Com isso, p ciclo é finalizado e depois repetido na mesma sequencia, conforme ilustrado na Figura 2.3. [MARTINS, 2006]

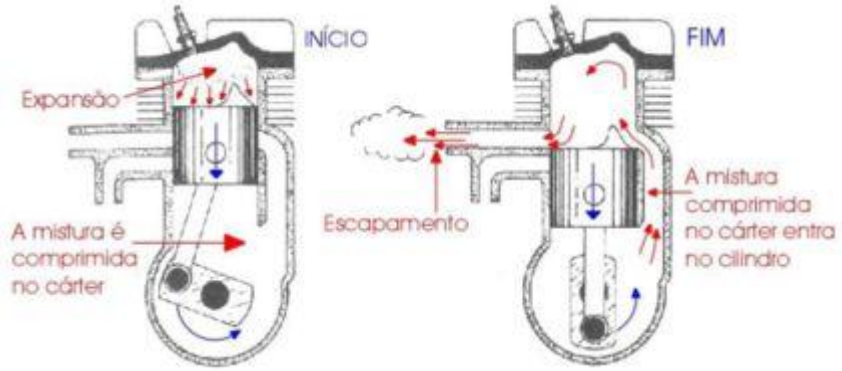


Figura 2.3 – Segundo tempo do ciclo de dois tempos

[RADIOCONTROLADO, 2012]

2.1.2 MOTOR QUATRO TEMPOS

A maior vantagem do motor quatro tempos é sua durabilidade, por isso é encontrado em quase todos os carros. São mais econômicos se comparados ao motor dois tempos, pois trabalham com uma baixa frequência. [MARTINS, 2006]

Pode-se dividir os quatro tempos do ciclo de um motor da seguinte maneira: [MARTINS, 2006]

- Tempo de Admissão ou Aspiração;
- Tempo de Compressão;
- Tempo de Explosão ou Combustão;
- Tempo de Escapamento ou Exaustão.

O tempo de admissão ou aspiração consiste no movimento do pistão do Ponto Morto Alto para o Ponto Morto Baixo com a válvula de admissão aberta. Nesse tempo, o motor aspira a mistura de ar e combustível para dentro do cilindro. Ao chegar ao Ponto Morto Baixo, a válvula de admissão é fechada prendendo a mistura no interior do cilindro. Conforme é mostrado na Figura 2.4. [RCMASTERS, 2012]

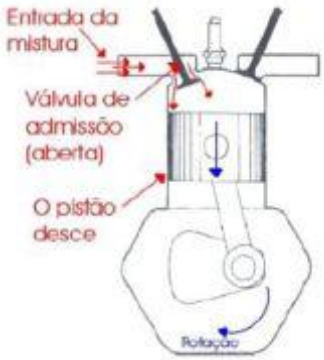


Figura 2.4 – Primeiro tempo - Admissão

[RADIOCONTROLADO, 2012]

O tempo de compressão consiste no movimento do pistão do Ponto Morto Baixo em direção ao Ponto Morto Alto com as válvulas fechadas promovendo a compressão da mistura no cilindro. Conforme mostrado na Figura 2.5. Essa fase pode parecer inútil, mas sem ela, a combustão produziria pouca potência e a energia se perderia na forma de calor. [RCMASTERS, 2012]

Figura 2.5 – Segundo tempo - Compressão

[RADIOCONTROLADO, 2012]

O tempo de explosão ou combustão consiste em três fases. Primeiro ocorre a ignição, onde a vela produz uma faísca, dando início a fase de combustão, onde a mistura é queimada. Então começa a última fase, chamada de expansão, quando o pistão parte do Ponto Morto Alto para o Ponto Morto Baixo por causa da forte pressão dos gases queimados que se expandem. A partir desse momento o motor consegue se manter funcionando sozinho, pois o

impulso é forte o suficiente para mantê-lo girando até a próxima combustão. Na Figura 2.6 é mostrado como funciona esse tempo. [RCMASTERS, 2012]

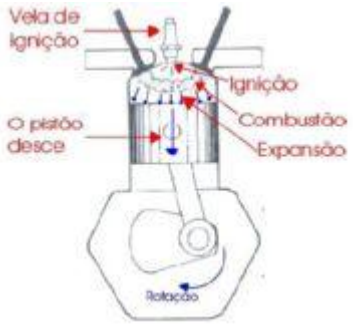


Figura 2.6 – Terceiro tempo - Explosão

[RCMASTERS, 2012]

O tempo de escapamento ou exaustão consiste no movimento do pistão do Ponto Morto Baixo para o Ponto Morto Alto com a válvula de escapamento aberta. Nesse momento, os gases queimados são expulsos do cilindro pelo pistão. Assim que o pistão chega ao Ponto Morto Alto a válvula de escapamento é fachada encerrando o ciclo do motor conforme é mostrado na Figura 2.7. [RCMASTERS, 2012]

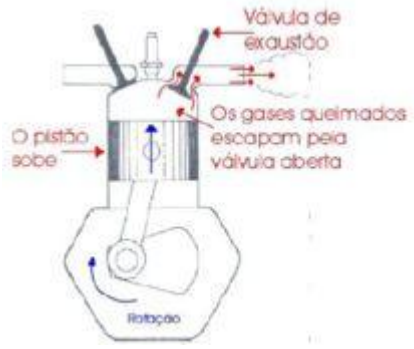


Figura 2.7 – Quarto tempo - Exaustão

[RADIOCONTROLADO, 2012]

2.2 TELEMETRIA

A palavra telemetria é resultado da união de duas palavras gregas. *Tele* que significa longe e *meter* que significa medir. Assim, telemetria significa realizar medições à distância ou remotamente. Como se trata de uma tecnologia onde sua maior vantagem é a troca de dados instantaneamente, na grande maioria dos casos utiliza-se comunicação sem fio [MATTOS, 2004].

A telemetria começou a ser utilizada devido à necessidade de medições em locais inacessíveis, como a temperatura interna de um forno, e hoje é utilizada em inúmeros casos, como monitoração da situação de um carro, medição de um míssil guiado, entre outros [MATTOS, 2004]. A Figura 2.8 ilustra um míssil em testes. O míssil contém a bordo um pacote de telemetria de baixa potência, que pode ser chamado de módulo de telemetria. Esse módulo contém equipamentos que medem os parâmetros físicos do dispositivo em teste, o circuito condicionador de sinal que prepara os sinais para transmissão e um rádio transmissor para enviar os dados para a base [MATTOS, 2004].

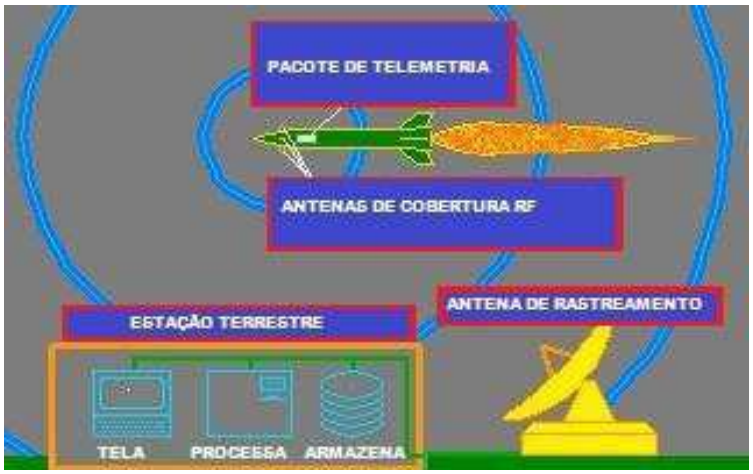


Figura 2.8 – Míssil em testes enviando sinal de telemetria

[MATTOS, 2004]

A telemetria é requerida quando uma medida tem que ser realizada em locais que é inacessível ao homem, esta é uma definição bem mais geral do que a original na qual expande o conceito da remoticidade [MATTOS, 2004].

Apesar de a telemetria ser em tempo real, nem todos os dados são analisados durante os testes, alguns dos dados são analisados apenas depois. Essa análise pós-operacional só é possível devido à gravação dos dados realizada durante os testes.

No setor automotivo e de logística as informações também podem ser transmitidas via GPRS juntamente com localização e outras informações de rastreamento [MATTOS, 2004]. Nesse projeto a transmissão de dados foi feita através da rede Wi-Fi.

A telemetria começou a ser usada pelo setor aéreo, principalmente na monitoração de mísseis, mas hoje vem se diversificando cada vez mais devido à redução dos equipamentos e sendo aplicada também na medicina e em animais [MATTOS, 2004].

Hoje, a telemetria é usada em larga escala em competições automobilísticas, os dados são utilizados pelos engenheiros da equipe para verificar o desgaste de peças, o desempenho do piloto, pontos que podem ser melhorados, onde o piloto está melhor ou pior que seus rivais, entre outras informações.

A Fórmula 1 se destaca das outras categorias por causa da alta tecnologia utilizada nela. O sistema de aquisição coleta muito parâmetros como temperatura e velocidade do motor, pressão de óleo, suspensões, rodas e elementos aerodinâmicos [MATTOS, 2004]. Esses dados são enviados para os *boxes* e são de extrema importância para que os engenheiros os analisem e consigam, por exemplo, prever uma possível quebra, instruir o piloto a fazer um ajuste (aerodinâmico, de freio, mistura de combustível) para melhorar o desempenho, detectar um pneu furado, detectar um vazamento de óleo.

A Figura 2.9 ilustra o piloto Rubens Barrichello na sala de telemetria da Ferrari. E na Figura 2.10 é mostrada a telemetria de uma volta no circuito de Mônaco.

Figura 2.9 – Barrichello na sala de telemetria da Ferrari

[MOTORSPORT, 2005]

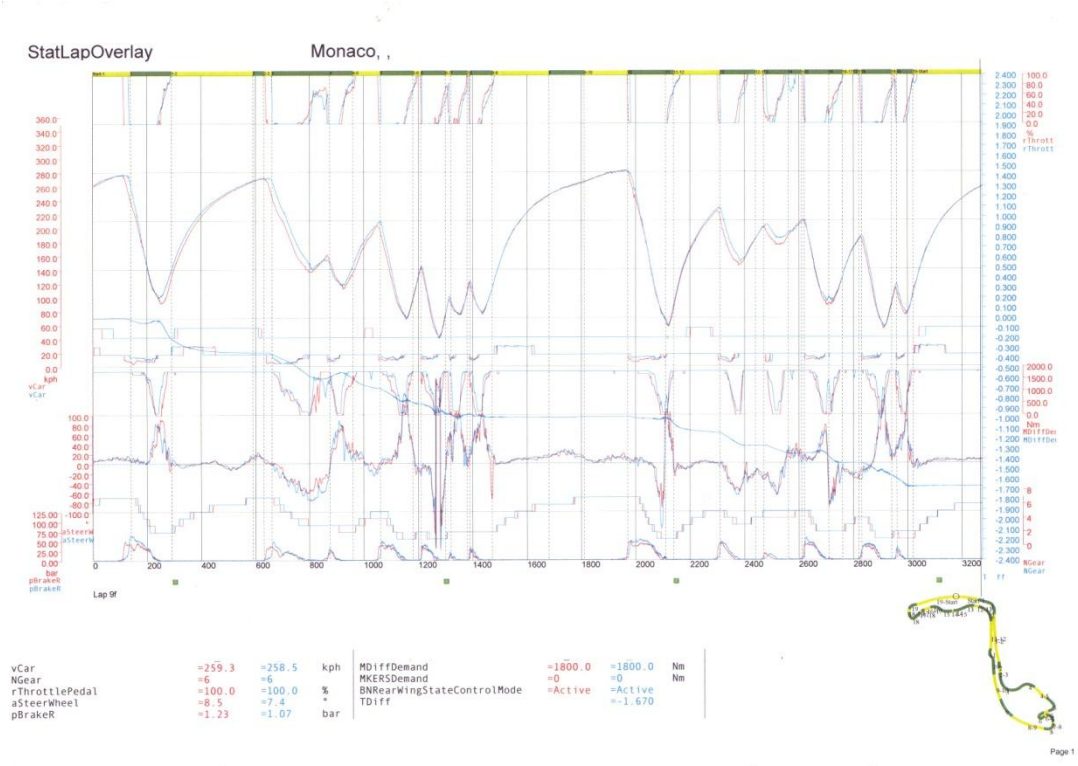


Figura 2.10 – Telemetria de uma volta em Mônaco

[SCARBSF1, 2011]

2.3 HARDWARE

2.3.1 SENSORES

A maioria dos sensores funciona convertendo um parâmetro físico, como a temperatura ou posição, em um sinal elétrico. [KILIAN, 2001]

Existem vários tipos de sensores, os que foram utilizados nesse projeto foram: sensor de temperatura e sensor de proximidade.

2.3.1.1 SENSOR DE TEMPERATURA - LM35

Sensores de temperatura possuem uma saída proporcional à temperatura em que está exposto. A maioria dos sensores de temperatura possui um coeficiente de temperatura positivo, a tensão sobe à medida que a temperatura sobe, porém, existem sensores de temperatura com coeficiente de temperatura negativo, a tensão diminui à medida que a temperatura sobe. [KILIAN, 2001]

O LM35 é um sensor de temperatura de circuito integrado e produz uma saída proporcional à temperatura em graus Celsius. O LM35 possui três terminais: Alimentação ($+V_s$), Terra (GND) e Saída (V_{out}), conforme é ilustrado na Figura 2.11. A tensão de saída do LM35 aumenta 1mV para cada 1ºC [KILIAN, 2001]. O LM35 possui um encapsulamento plástico TO-92 e um alcance de temperatura entre -60ºC e 150ºC [NATIONAL SEMICONDUCTOR, 2012]

Figura 2.11 – LM35

[NATIONAL SEMICONDUCTOR, 2012]

2.3.1.2 SENSOR DE PROXIMIDADE DE EFEITO HALL

O Efeito Hall é um efeito que ocorre causado pela propriedade especial de semicondutores em que produzem uma tensão na presença de um campo magnético. Esse feito foi descoberto por E. H. Hall em 1879. [KILIAN, 2001]

Um sensor de efeito Hall produz uma tensão quando o campo magnético aumenta. Isso é feito com a movimentação de um ímã ou modificando o sentido do campo magnético. [KILIAN, 2001]

Nesse projeto, foi utilizado um sensor de efeito Hall que coleta dados que, posteriormente, foram utilizados para o cálculo da velocidade. O sensor funciona conforme ilustrado na Figura 2.12.

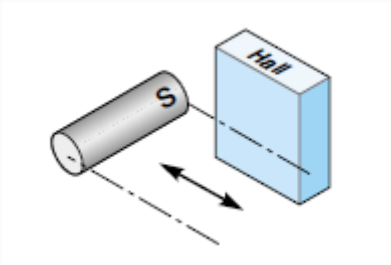


Figura 2.12 – Sensor de Proximidade de Efeito Hall

[KILIAN, 2001]

2.3.2 MICROCONTROLADOR

2.3.2.1 ATMEGA328

O ATmega328 é um microcontrolador CMOS 8-bit de baixa potência baseado em AVR melhorado com arquitetura RISC. Executa funções poderosas em um único ciclo de *clock*, alcançando aproximadamente 1 MIPS por MHz permitindo a otimização velocidade de processamento em função do consumo de energia. [ATMEL, 2012]

O núcleo AVR combina um rico conjunto de instruções com 32 registros de uso geral de trabalho. Todos estão diretamente ligados à unidade lógica e aritmética (ULA), permitindo

que dois registradores sejam acessados com uma instrução em um único ciclo de *clock*. [ATMEL, 2012]

O ATmega328 possui 32KBytes de memória Flash, com possibilidade de leitura e escrita simultânea, 1KBytes de EEPROM e 2KBytes de RAM. O tamanho do vetor de interrupção é de duas palavras ou vetores. [ATMEL, 2012]

2.3.2.2 ARDUINO

O Arduino Duemilanove é uma placa de microcontrolador baseada no ATmega328. Possui 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saída PWM – *Pulse-width Modulation*) e 6 entradas analógicas. Essas entradas podem ser facilmente configuradas e utilizadas para sensores. Além das entradas, o Arduino possui: um cristal oscilador de 16MHz, uma conexão USB, uma entrada para alimentação, um cabeçalho ICSP e um botão reset. Esses componentes do Arduino são mostrados na Figura 2.13 e suas características na Tabela 2.1. [ARDUINO DUEMILANOVE, 2012]

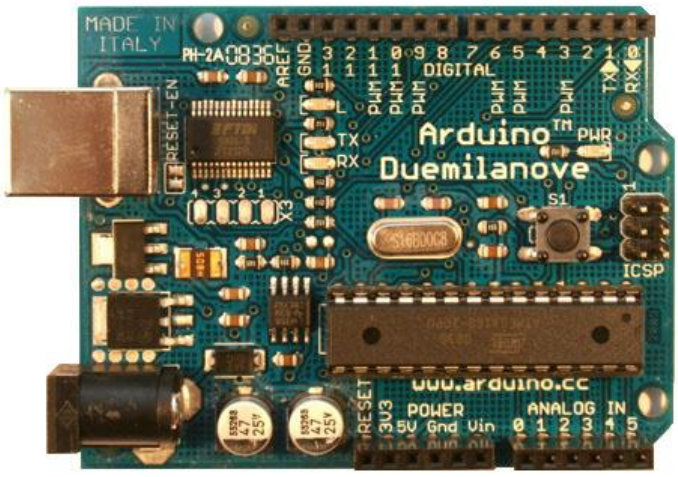


Figura 2.13 – Arduino Duemilanove

[ARDUINO DUEMILANOVE, 2012]

Tabela 2.1 – Características do Arduino

Microcontrolador	**ATmega328**
Voltagem operacional	5V
Voltagem de alimentação (recomendada)	7-12V
Voltagem de alimentação (limites)	6-20V
Pinos I/O digitais	14 (6 podem ser saídas PWM)
Pinos de entrada analógica	6
Corrente contínua por pino I/O	40 mA
Corrente contínua para o pino 3.3V	50 mA
Memória *flash*	32 KB (2KB usados para o *bootloader*)
SRAM	2 KB
EEPROM	1 KB
Velocidade de *clock*	16 MHz

[ARDUINO DUEMILANOVE, 2012]

O Arduino Duemilanove faz a conexão com um computador através de um cabo USB-AB, onde pode ser programado utilizando o software Arduino. O Arduino Duemilanove vem programado com um *bootloader*, que permite o envio de novos programas sem a utilização de um programador de hardware externo. [ARDUINO DUEMILANOVE, 2012]

O Arduino pode ser alimentado pela conexão USB ou uma fonte de alimentação externa, que pode ser uma fonte ou uma bateria. A fonte pode ser conectada com um *plug* de 2,1mm, com centro positivo, no conector de alimentação. A bateria pode ser conectada no mesmo *plug* ou conectada nos pinos Gnd (Terra) e Vin (entrada de voltagem) através de cabos. [ARDUINO DUEMILANOVE, 2012]

Os pinos de alimentação que o Arduino possui são:

- VIN – Entrada de alimentação para a placa quando uma fonte externa é utilizada. Pode-se fornecer alimentação por este pino ou, caso o conector esteja sendo utilizado, pode-se acessar a alimentação.

- 5V – Fonte de alimentação para o microcontrolador e para outros componentes da placa.
- 3V3 – Alimentação de 3,3 volts fornecida pelo chip FTDI.
- GND – Pino Terra.

Os 14 pinos digitais do Duemilanove podem ser utilizados como entrada ou saída. Eles operam com 5 volts. Alguns deles possuem funções especiais: [ARDUINO DUEMILANOVE, 2012]

- Serial: 0(RX) e 1(TX) – Usados para receber (RX) e transmitir (TX) dados seriais TTL.
- External Interrupts: 2 e 3 – Podem ser configurados para enviar uma interrupção por um valor baixo, uma elevação ou *falling edge* ou mudança de valor.
- PWM : 3, 5, 6, 9, 10 e 11 – Fornecem uma saída analógica PWM de 8-bit com a função analogWrite().
- SPI: 10(SS), 11(MOSI), 12(MISO), 13(SCK) – Pinos que suportam comunicação SPI.
- LED: 13 – Existe um LED já montado e conectado ao pino 13. O LED acende quando o valor é HIGH e apaga quando o valor é LOW.
- I^2C: 4(SDA) e 5(SCL) – Comunicação I^2C(TWI) usando a biblioteca Wire.
- AREF – Referência de voltagem para entradas analógicas.
- Reset – Reseta o microcontrolador ao receber o valor LOW.

Além dos pinos digitais, o Duemilanove possui 6 entradas analógicas com uma resolução de 10 bits, 1024 valores distintos. Por padrão, medem de 0 a 5 volts, é possível mudar o limite superior usando o pino AREF. [ARDUINO DUEMILANOVE, 2012]

O Arduino Duemilanove também possui um polifusível resetável para proteger a porta USB do computador contra curto-circuito. Em caso de sobrecarga, o fusível interrompe a conexão até que a sobrecarga seja removida.

O Arduino também possui uma interface de desenvolvimento integrado próprio, conforme é mostrado na Figura 2.14. A plataforma é composta por um escritor de código, um compilador e um comunicador USB serial, que é responsável pela comunicação entre o computador e a placa e pela gravação do programa.

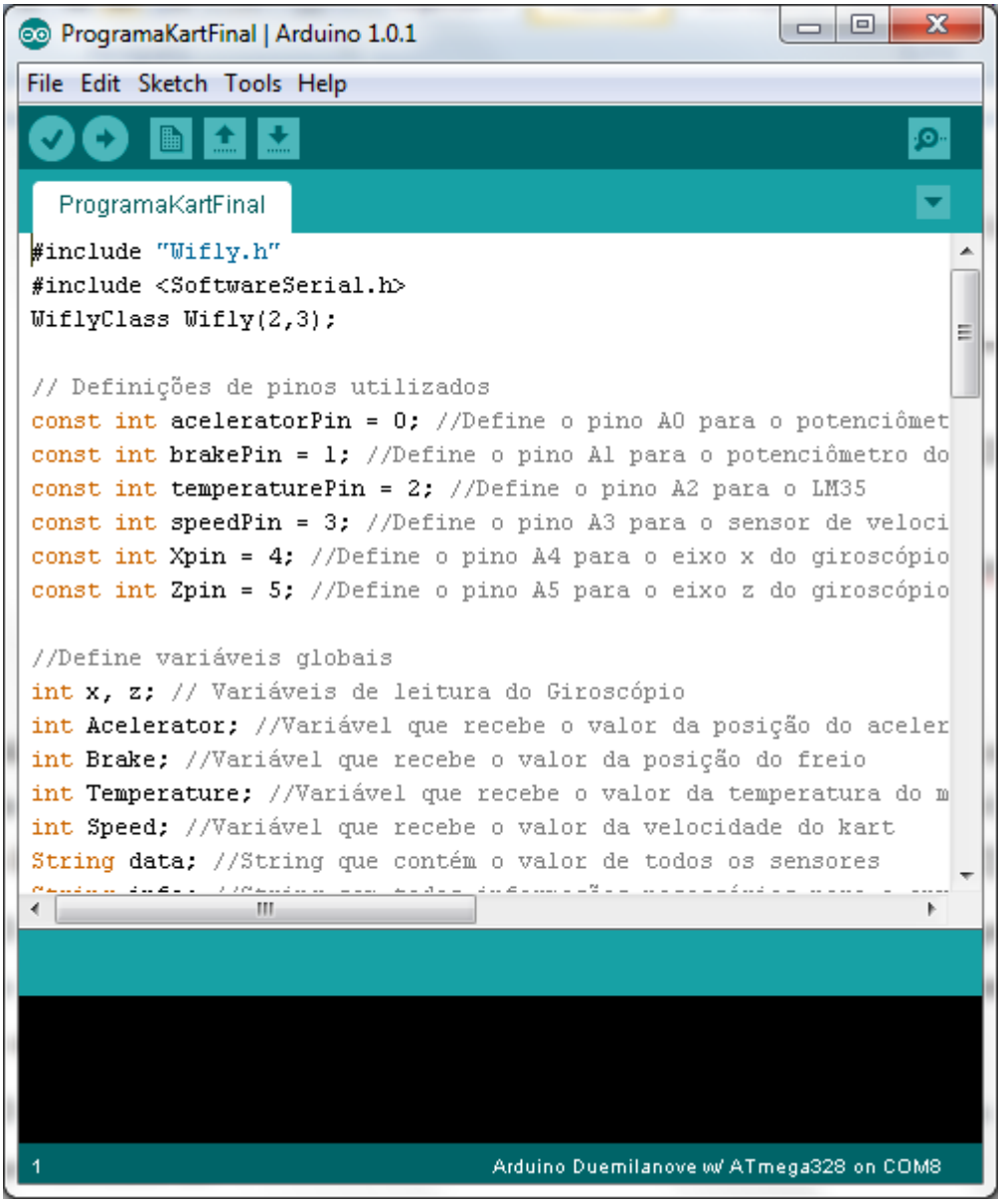


Figura 2.14 – Interface de Desenvolvimento Integrado do Arduino

A programação no Arduino é feita em uma linguagem baseada em C/C++. Possui uma estrutura básica com dois métodos: *setup()* e *loop()*. A função *setup()* é chamada quando o programa começa e é utilizada para definir os modos de entrada ou saída dos pinos, indicar bibliotecas, inicializar variáveis, etc. É executada apenas uma vez, quando o Arduino é iniciado ou resetado. A função *loop()* faz o que o nome indica, repete-se continuamente permitindo que o programa funcione dinamicamente. É responsável pela inicialização e declaração dos valores iniciais. [ARDUINO, 2012]

2.3.3 WI-FI

Redes Wi-Fi utilizam tecnologias de rádio chamadas IEEE 802.11a, 802.11b e 802.11g e fornecem uma conectividade sem fio rápida, confiável e segura. A rede Wi-Fi pode ser utilizada para conectar computadores uns aos outros, para conectar na internet e para conectar em redes com fio (Ethernet ou IEEE 802.3). [WI-FI ALLIANCE, 2012] A rede Wi-Fi opera nas bandas de rádio não licenciadas 2.4GHz nas tecnologias IEEE 802.11b e IEEE 802.11g e na frequência 5GHz na tecnologia 802.11a.[IEEE802]

2.3.3.1 WIFI SHIELD – WIFLY RN-171

A Wifi Shield foi desenvolvida pela empresa Seeed Studio e utiliza um módulo RN-171 para promover a conexão às redes sem fio. Utilizando a Wifi Shield, são necessários apenas dois pinos, sem contar os pinos de alimentação, para conectar o dispositivo em uma das redes sem fio 802.11b/g. Possui uma antena independente, aumentando o raio de ação e oferecendo sinais de transmissão mais fortes. Oferece suporte aos protocolos mais comuns de comunicação: TCP, UDP e FTP. [SEEED STUDIO, 2012]. Na Tabela 2.2 são mostradas as especificações da Wifi Shield e na Figura 2.15 é ilustrada sua interface.

Tabela 2.2 – Especificações Wifi Shield

Item	Min	Típico	Max	Unidade
Voltagem	3.3	5	5.5	VDC
Corrente	25	60	400	mA
Energia de Transmissão		0-10		dBm
Frequência		2402~2480		MHz
Taxa de rede		1-11 Mbps for 802.11b/6-54Mbps for 802.11g		
Dimensões		60x56x19		Mm
Peso		20±2		g

[SEEED STUDIO, 2012]

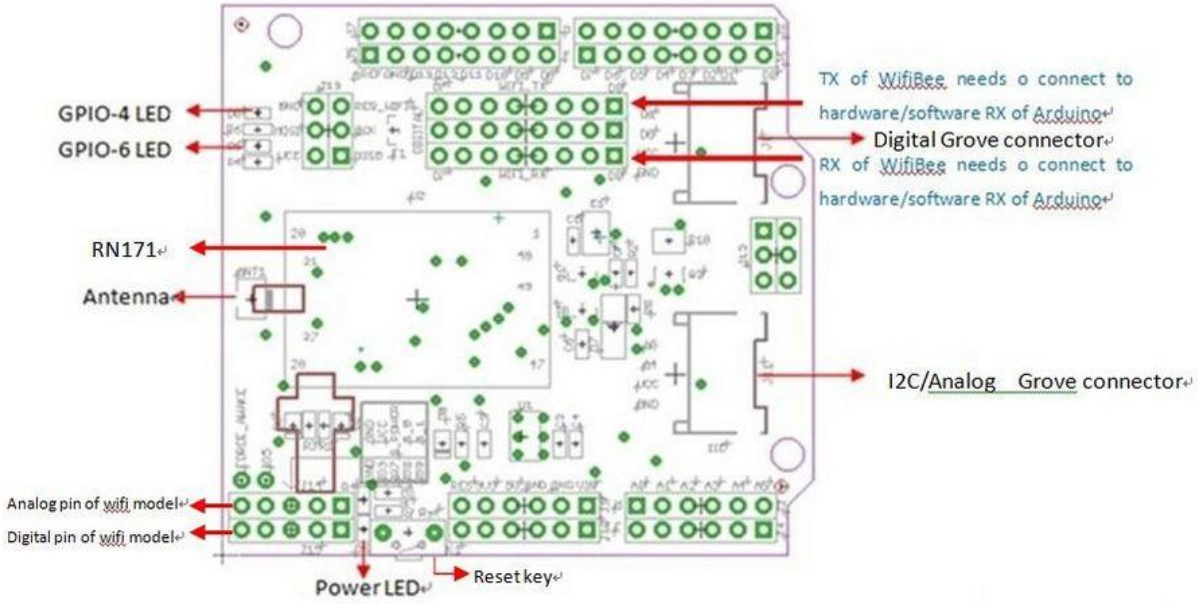


Figura 2.15 – Interface da Wifi Shield

[SEEED STUDIO, 2012]

O módulo RN-171 é um módulo de rede sem fio TCP/IP completo. O módulo incorpora uma rádio frequência de 2.4GHz, possui um processador SPARC de 32 bits, pilha TCP/IP, relógio em tempo real, acelerador de criptografia, gerenciamento de energia e interfaces para sensores analógicos. O módulo é pré-carregado com um firmware para facilitar a integração e minimizar o desenvolvimento da aplicação. Assim como a Wifi Shield, necessita de quatro pinos, dois de alimentação e dois de transmissão.

2.3.4 POTENCIÔMETRO

Potenciômetros são componentes que convertem deslocamentos rotativos ou lineares em uma tensão. Na verdade, um potenciômetro altera o valor de uma resistência, que é facilmente convertido em uma tensão, conforme é ilustrado na Figura 2.16 [KILIAN, 2001]

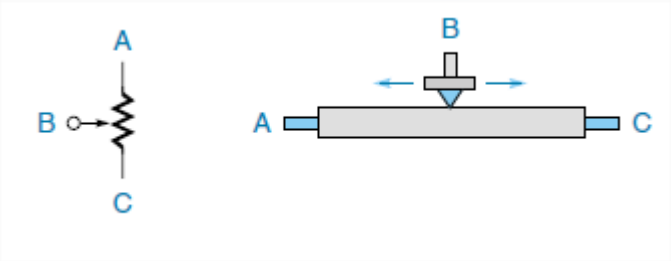


Figura 2.16 – Potenciômetro Linear

[KILIAN, 2001]

Nesse projeto foram utilizados dois potenciômetros de 100K para aferir a posição dos pedais de aceleração e freio do kart.

2.3.5 GIROSCÓPIO

O giroscópio utilizado no projeto foi o LPY530AL, que é um giroscópio de dois eixos capaz de medir a velocidade angular. Ele trabalha em temperaturas entre -40ºC e 85ºC e possui uma escala total de ± 300º/s. Esse giroscópio é uma combinação de um atuador e um acelerômetro integrados em uma única estrutura.

A Figura 2.17 ilustra o diagrama de bloco do giroscópio e a Figura 2.18 ilustra os pinos.

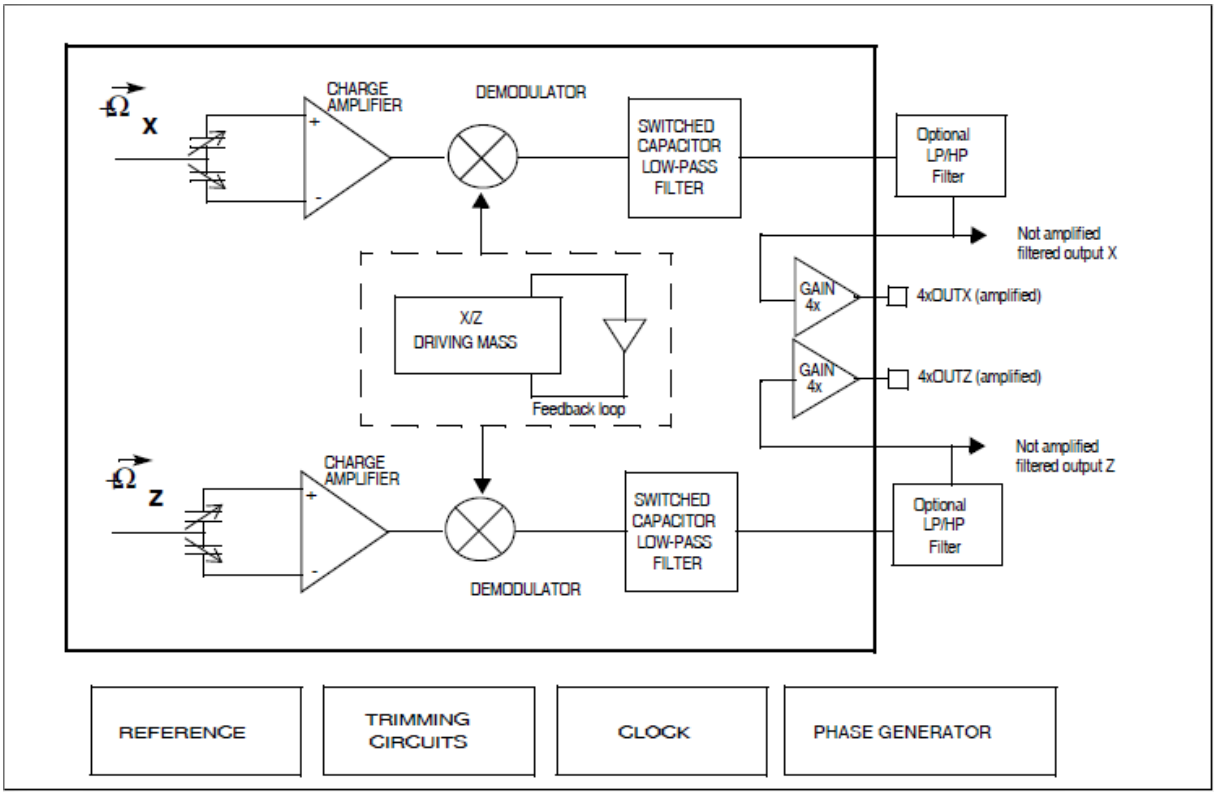


Figura 2.17 – Diagrama de bloco do giroscópio LPY530AL

[ST, 2012]

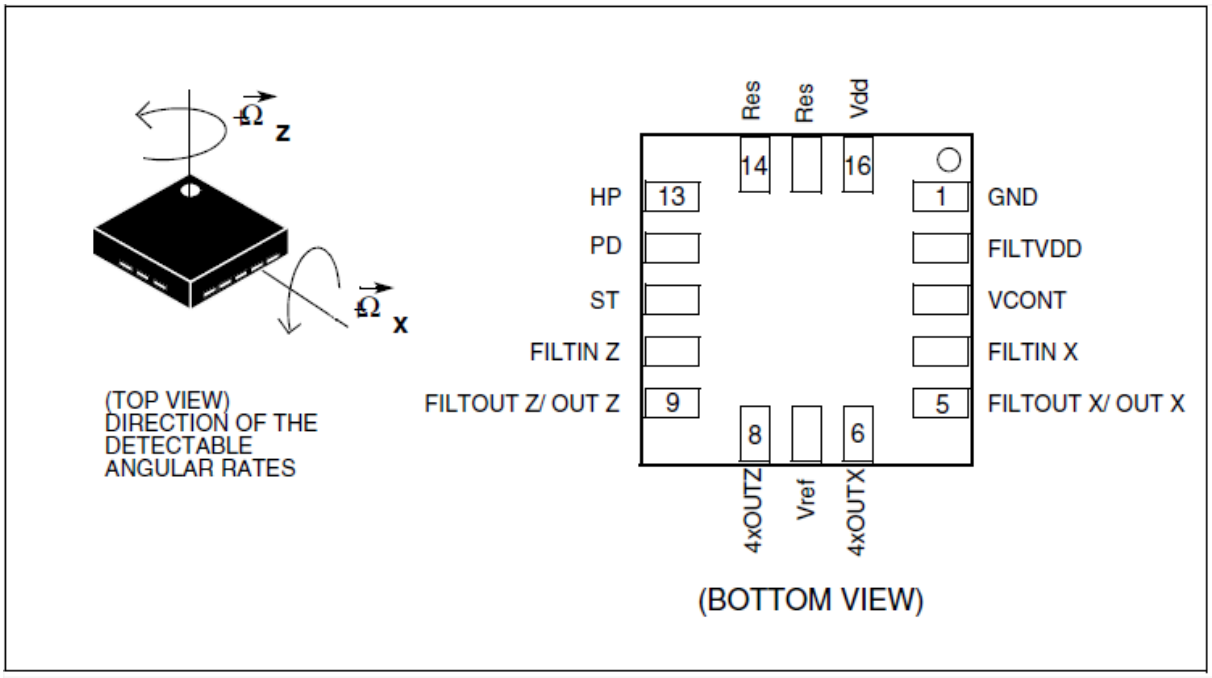


Figura 2.18 – Pinos do giroscópio LPY530AL

[ST, 2012]

2.4 SOFTWARE

2.4.1 BANCO DE DADOS ORACLE

O Oracle Database é um RDBMS (relational database management system) orientado a objeto (também chamado de ORDBMS), ou seja, um sistema de gerenciamento de banco de dados relacional de objetos. O modelo relacional é a base para um RDBMS. Essencialmente, um RDBMS insere o dado no banco de dados, guarda esse dado e retorna ele para que possa ser manipulado por alguma aplicação. O RDBMS é distinguido entre 2 tipos de operação: [ORACLE, 2012]

Operações lógicas – Neste caso, uma aplicação especifica qual conteúdo é exigido. Por exemplo, uma aplicação solicita o nome de um funcionário ou adiciona um registro do funcionário a uma tabela. [ORACLE, 2012]

Operações físicas – Neste caso, o RDBMS determina como deve ser feito, e leva a cabo a operação. Por exemplo, após uma aplicação consultar uma tabela, o banco de dados pode usar um índice para encontrar as linhas solicitadas, ler os dados na memória, e realizar muitas outras etapas antes de retornar um resultado para o usuário. O RDBMS armazena e recupera dados de modo que as operações físicas são transparentes para as aplicações do banco de dados. [ORACLE, 2012]

Uma característica de um RDBMS é a independência de armazenamento de dados físicos de estruturas de dados lógicos. No banco de dados Oracle, um esquema de banco de dados é uma coleção de estruturas de dados lógicos, ou objetos de esquema. Um esquema de banco de dados é propriedade de um usuário de banco de dados e tem o mesmo nome que o nome de usuário. [ORACLE, 2012]

Objetos de esquema são estruturas criadas pelo usuário que diretamente se referem aos dados no banco de dados. O banco de dados suporta muitos tipos de objetos de esquema, o mais importante dos quais são tabelas e índices. [ORACLE, 2012]

2.4.1.1 TABELAS

Uma tabela descreve uma entidade, por exemplo uma tabela de empregados. Você define uma tabela com um nome de tabela, como funcionários, e um conjunto de colunas. Em geral, você dá a cada coluna um nome, um tipo de dado, e um tamanho ao criar a tabela. [ORACLE, 2012]

Uma tabela possui um conjunto de linhas. A coluna identifica um atributo da entidade descrita pela tabela, enquanto que uma linha identifica uma instância da entidade. Por exemplo, os atributos da entidade funcionários correspondem a colunas de ID do empregado e último nome. Uma linha identifica um funcionário específico. [ORACLE, 2012]

Você pode, opcionalmente, especificar as regras de cada coluna de uma tabela. Essas regras são chamadas de *integrity constraints*. Um exemplo é uma *integrity constraints* NOT NULL. Isto força a coluna conter um valor em cada linha. [ORACLE, 2012]

2.4.1.2 ÍNDICES

Um índice é uma estrutura de dados opcional que você pode criar em uma ou mais colunas de uma tabela. Os índices podem aumentar o desempenho de recuperação de dados. Quando está processando um pedido, o banco de dados pode usar índices disponíveis para localizar as linhas solicitadas de forma eficiente. Índices são úteis quando os aplicativos frequentemente consultam uma linha específica ou intervalo de linhas. [ORACLE, 2012]

Os índices são lógica e fisicamente independente dos dados. Assim, você pode excluir e criar índices com nenhum efeito sobre as tabelas ou outros índices. Todas as aplicações continuam a funcionar depois de excluir um índice. [ORACLE, 2012]

2.4.1.3 Structured Query Language (SQL)

SQL é uma linguagem declarativa baseada em conjunto que fornece uma interface para um RDBMS, como para o Oracle Database. Em contraste com as linguagens procedurais, como C, que descrevem como as coisas devem ser feitas, SQL é não-procedural

e descreve o que deve ser feito. Usuários especificam o resultado que eles querem (por exemplo, os nomes dos funcionários atuais), e não como a extraí-lo. SQL é a linguagem padrão ANSI para bancos de dados relacionais. [ORACLE, 2012]

Todas as operações sobre os dados em um banco de dados Oracle são realizadas utilizando instruções SQL. Por exemplo, você usa o SQL para criar tabelas, consultar e modificar dados. Uma instrução SQL é uma seqüência de texto SQL, como o seguinte: [ORACLE, 2012]

```
SELECT primeiro_nome, ultimo_nome FROM funcionario;
```

Instruções SQL permitem que você execute as seguintes tarefas:

- Consulta de dados
- Inserir, atualizar e excluir linhas em uma tabela
- Criar, substituir, alterar e descartar objetos
- Controlar o acesso ao banco de dados e seus objetos
- Garantir consistência e integridade do banco de dados

2.4.1.4 PL/SQL e Java

PL/SQL é uma extensão processual ao Oracle SQL. PL/SQL é integrado com o banco de dados Oracle, permitindo que você use todas as instruções SQL Oracle Database, funções e tipos de dados. Você pode usar PL/SQL para controlar o fluxo de um programa SQL, usar variáveis e criar logs de tratamento de erros de procedimentos. [ORACLE, 2012]

Um dos principais benefícios do PL/SQL é a capacidade de armazenar a lógica da aplicação no próprio banco de dados. Um procedimento ou função é um objeto de esquema que consiste em um conjunto de instruções SQL e outras PL/SQL, agrupadas, armazenadas no banco de dados, e executada como uma unidade para resolver um problema específico ou para executar um conjunto de tarefas relacionadas. [ORACLE, 2012]

Banco de Dados Oracle também pode armazenar unidades de programas escritos em Java. Um procedimento armazenado em Java é um método Java publicada para o SQL e

armazenados na base de dados para o uso geral. Você pode chamar programas existentes em PL/SQL do Java e programas em Java do PL/SQL. [ORACLE, 2012]

2.4.2 TIBCO RTVIEW

O TIBCO® Enterprise RTView é uma aplicação leve de monitoramento e exibição que permite que os usuários otimizem seus investimentos TIBCO aumentando drasticamente a visibilidade de sua infraestrutura TIBCO. A aplicação fornece modelos pré-desenvolvidos de monitoramento TIBCO para as aplicações TIBCO Enterprise Message Service™, TIBCO Hawk® e TIBCO Rendezvous®. [TIBCO, 2012]

O Enterprise RTView é flexível e configurável. O Display Builder no Enterprise RTView possui palhetas tipo *point-and-click* de objetos de exibição tais como medidores, graduação, gráficos, escalas e campos de texto dinâmicos para a criação de *dashboards* e relatórios dinâmicos. Esses *dashboards* permitem o monitoramento em tempo real de informações customizadas, essenciais. Os usuários do TIBCO podem expor qualquer aplicação ativada para evento e elegantemente apresentar displays de dados em aplicações Java, *applets*, *thin clients* ou portais. [TIBCO, 2012]

A tecnologia de exibição dinâmica do Enterprise RTView fornece uma maneira intuitiva e eficiente para que os usuários interajam com os dados. O Enterprise RTView tem como base a tecnologia *thin-client* que permite que dados críticos sejam visualizados por meio de navegadores que executam pequenos *applets*, HTML *thin clients*, ou *portlets* compatíveis com JSR-168. Isto permite desempenho extremamente alto, custos de implementação zero e fácil distribuição das informações da empresa nos firewalls. [TIBCO, 2012]

- Fornece maior visibilidade e feedback em tempo real sobre o status da rede sem o custo de construir e manter interfaces customizadas.
- Apresenta e correlaciona a ampla variedade de dados que possam pelo *backbone*.
- Permite resposta imediata a condições excepcionais por meio de maior visibilidade e alertas configuráveis.
- Permite uma visão unificada de todas as atividades de gestão de rede ao integrar com modelos de gestão de rede tais como Tivoli e Unicenter e dispositivos SNMP.

• Simplifica a definição de bases de regras para monitoramento automatizado do sistema com editor GUI.

Principais Características

• Paletas tipo apontar e clicar para a criação de *dashboards* e relatórios customizados.

• Alertas e limiares customizáveis.

• Exibição dinâmica via navegador da web como aplicações em Java, *applets, thin clients* interativos ou portais empresariais.

• Capacidade de *drill down* para informações detalhadas do agente e microagente.

Arquivo de dados e análise de tendência. Plataforma RTView Business Information Delivery Platform. [TIBCO, 2012]

• Fornece informações de negócios visualmente

– Tabelas, gráficos, medidores, mostradores, rótulos, imagens, controles ou acrescente seus próprios.

• Integra fontes de dados distintas e diversas

– RDBMS, OLAP, JMS, TIBCO, XML e adaptadores customizados.

• Integra as tecnologias de implementação existentes

– Desktop Application, Web Browser rich client, Web Browser thin client

• Escalabilidade e confiabilidade de classe empresarial

– Componentes do lado do servidor otimizam a distribuição de informações para desktops

Na Figura 2.19 é mostrado um exemplo de *dashboard* criado no RTView:

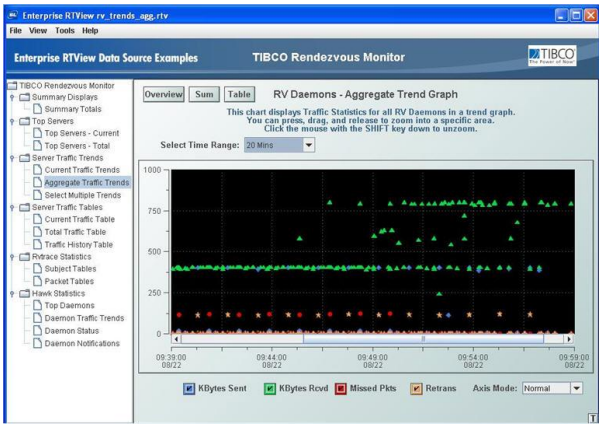


Figura 2.19 – Exemplo de Dashboard do RTView

[SL, 2012]

Recursos do Enterprise RTView: [TIBCO, 2012]

Modelo de aplicação

Desenvolve aplicações visualmente com controles de padrão industrial, objetos gráficos inclusos e Intelligent Parameterized Navigation (IPN).

Interoperabilidade e Flexibilidade

Integra aplicações com tecnologias empresariais existentes e aplicações na web. Construir ou integrar objetos customizados/terceiros.

Otimizado para desempenho

Objetos integrados com base em tecnologia SL-GMS em tempo real.

A tecnologia do servidor minimiza o uso de recursos da rede, enquanto gravação em *cache* sob demanda fornece resposta rápida para usuários. [TIBCO, 2012]

Mecanismos de segurança integrada com base em função limitam o acesso a informações e podem ser customizados. [TIBCO, 2012]

Usuário Pass-through

As credenciais pass-through do usuário se beneficiam dos sistemas de segurança de *back-end* existentes. [TIBCO, 2012]

O OS Independence é executado e/ou implementado em qualquer OS suportado por JRE, incluindo Windows, Linux, Solaris, Mac, PDA, etc. [TIBCO, 2012]

2.4.3 TIBCO BUSINESSWORKS

O TIBCO ActiveMatrix BusinessWorks (BW) é uma plataforma de integração escalável, extensível e fácil de usar que permite o desenvolvimento de projetos de integração. O BW usa a interface gráfica TIBCO Designer para definição de processos de negócio e para o BW executar as *engines* dos processos. [TIBCO, 2012]

2.4.3.1 MENSAGERIA

O BW é baseado em padrões de mensagens com histórico comprovado. Para que o projeto de integração funcione em tempo de execução, se faz necessário um sistema de mensageria (barramento de mensagens) para suportar, confiantemente, o elevado número de mensagens enviadas e recebidas. Os protocolos suportados são: TIBCO Rendezvous, JMS, HTTP. [TIBCO, 2012]

2.4.3.2 ADAPTADORES

As informações de negócio são distribuídas em várias aplicações distintas ou disponíveis em bancos de dados ou arquivos. Os adaptadores (Adapters) ajudam a disponibilizar essas informações para os processos “adaptando” as aplicações para o formato comum de sistema de mensagem. [TIBCO, 2012]

2.4.3.3 MODELAGEM DE PROCESSOS

Os processos descrevem o fluxo real dos dados dentro da empresa. Eles são feitos na interface TIBCO Designer e suas características incluem: Configuração de serviços adaptadores; Conjunto completo de atividades comuns(leitura de arquivos, escrita em arquivo, criação de arquivo, temporizadores, recepção e emissão de mensagens, envio e recebimento de e-mails, entre outros); Cria grupos de atividades; Suporta XPath; Atividade de

transformação de dados que vieram de atividades anteriores; Interface fácil de usar e fácil de encontrar e corrigir erros. [TIBCO, 2012]

A Figura 2.20 ilustra um processo desenvolvido no BW.

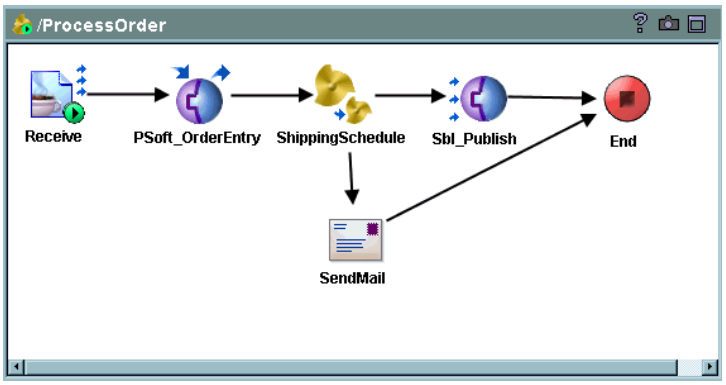


Figura 2.20 – Exemplo de processo

[TIBCO, 2012]

2.4.3.4 MAPEAMENTO DE DADOS E ESQUEMAS

Aplicações diferentes no negócio utilizam representação de dados diferentes. O TIBCO BusinessWorks permite ver e manipular os dados que estão chegando ou saindo de uma atividade através de esquemas XML, que são chamados de XSD (XML Schema Definition). Os XSD's definem os dados e formatos que devem estar presentes no XML, que são os dados de entrada e saída do BW, garantindo a integridade dos mesmos. [TIBCO, 2012]

Na Figura 2.21 é mostrado um exemplo de XSD.

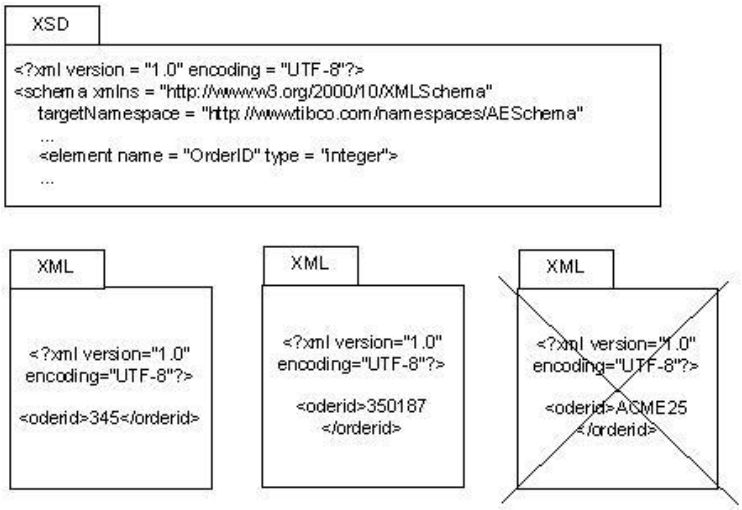


Figura 2.21 – Exemplo de XSD

[TIBCO, 2012]

2.4.4 TIBCO SPOTFIRE

O Spotfire Analytics Platform atende às necessidades de todos os usuários do seguimento de análise de negócios- a partir da descoberta de dados e análise *ad-hoc*, até relatórios interativos e *dashboards*, e também para o domínio de aplicações específicas, a análise orientada a eventos em tempo real e análise preditiva - tudo a partir de uma única arquitetura. [TIBCO, 2012]

O TIBCO Spotfire ® Enterprise Analytics oferece uma plataforma de próxima geração de BI radicalmente mais rápida e é muito mais adaptável para a indústria e os desafios específicos de negócios. Ao contrário do tradicional BI, o software Spotfire permite fazer e responder perguntas de forma praticamente irrestrita aos usuários da linha de frente - as pessoas que tomam decisões todos os dias - sem a necessidade de inúmeros novos relatórios ou consultas personalizadas de TI. Spotfire é interativo, com recursos visuais que permitem às pessoas ver facilmente as tendências, padrões, exceções e imprevistos nas relações de dados com uma velocidade e adaptabilidade sem precedentes. [TIBCO, 2012]

O TIBCO Spotfire Analytics oferece um ambiente incrivelmente rápido e flexível para a análise de dados críticos para ajudar a organização a tomar decisões melhores e mais inteligentes. Com uma poderosa funcionalidade de análise e modelagem preditiva e uma interface altamente visual e intuitiva, o Spotfire oferece aos profissionais de negócio e

técnicos a capacidade de explorar rapidamente os seus dados, ajudando a encontrar *insights* (conhecimento) chave para dar-lhes uma vantagem competitiva exclusiva. [TIBCO, 2012]

O Spotfire Analytics é uma plataforma única que abrange o espectro das análises *ad-hoc* para a construção rápida de aplicações analíticas de forma personalizada, a partir de análise de planilhas, banco de dados em tempo real, de dados orientados a eventos e de desktop baseados em dados de pesquisa amplamente distribuída na web através de painéis interativos. Incorporando funções estatísticas poderosas, o Spotfire é a plataforma analítica mais poderosa do mundo, dando aos indivíduos e às organizações uma vantagem de informação imediata sobre seus concorrentes. [TIBCO, 2012]

Principais Funcionalidades

- Análise exploração dos dados com visualização intuitiva e interativa
  - Gráficos de barra, *scatter plots*, mapas *treemaps*, *box plots*, mapas, etc.
  - Arquitetura de dados em memória promove uma incrível velocidade e flexibilidade.
- Acesso direto a dados corporativos e locais, incluindo planilhas eletrônicas.
  - SAP BW, SAP R/3, Oracle eBusiness Suite, Salesforce.com, bancos de dados ODBC, etc.
  - Excel, csv, e qualquer arquivo de dados flat.
- Criação e distribuição de aplicações analíticas para a web
  - Nenhuma codificação é necessária
- Criação de analises estatísticas sofisticadas
  - Estreita relação entre o Spotfire e pacotes estatísticos de mercado
- Análise de dados de baseados em eventos e em tempo real
  - Estreita integração com as ferramentas de infraestrutura da TIBCO

Na Figura 2.22 é mostrada a interface de construção de análises do TIBCO Spotfire.

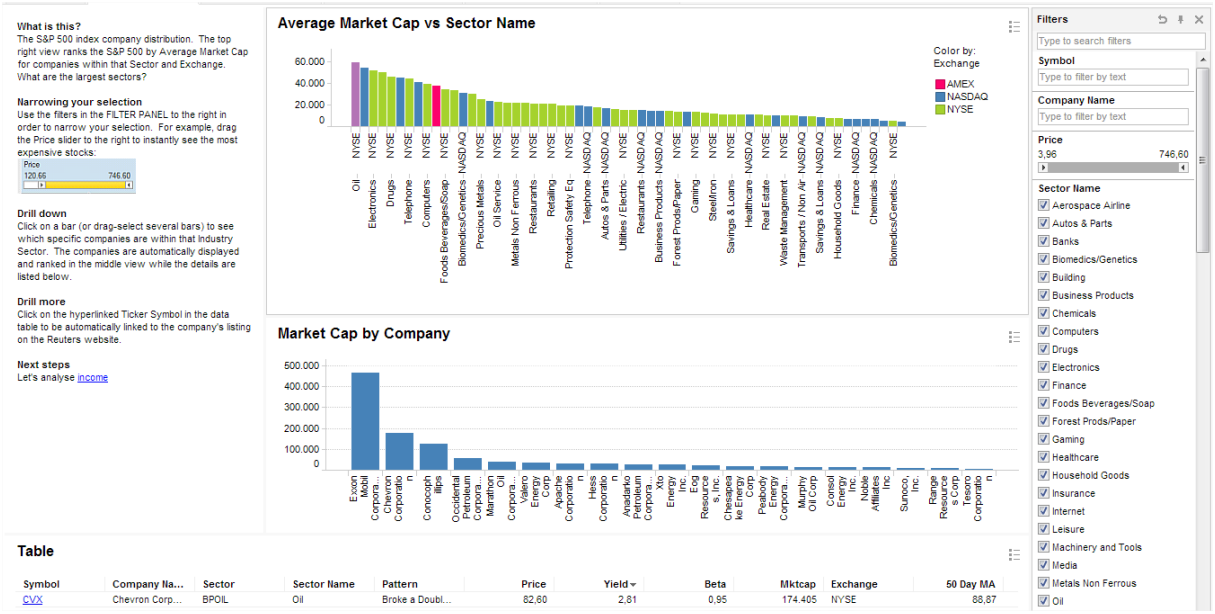


Figura 2.22 – Exemplo de análise no Spotfire

[TIBCO, 2012]

2.4.5 TIBCO RENDEZVOUS

O Rendezvous facilita a criação de aplicações distribuídas que trocam dados através da rede. O Rendezvous suporta várias plataformas de sofware e hardware, assim programas executando em muitos tipos de computadores diferentes em uma rede se comunicam sem problemas. [TIBCO, 2012]

Para programadores, o Rendezvous se divide em dois componentes principais: a API e o Daemon. Existem API's do Rendezvous para diversas linguagens, como C, Java, .NET, entre outras. O Rendezvous Daemon executa em cada computador participante na sua rede, todas as informações que trafegam pelos programas passam pelo Rendezvous Daemon quando essa informação entra e sai do computador hospedeiro. Programas que utilizam o Rendezvous como meio de comunicação são chamados programas Rendezvous. [TIBCO, 2012]

Os programas Rendezvous comunicam-se enviando mensagens. Cada mensagem possui um nome de assunto que, simultaneamente, especifica o destino da mensagem e o modo de entrega. [TIBCO, 2012]

Tanto em comunicação ponto a ponto com em *multicast,* as mensagens do Rendezvous são enviadas anonimamente. O remetente publica a mensagem endereçada a um assunto, em vez de programas ou computadores. O receptor subscreve a um assunto, onde receberá todas as mensagens enviadas para esse assunto, em vez de estabelecer comunicações únicas com o remetente. [TIBCO, 2012]

Antes de usar o Rendezvous, os programas devem abrir o ambiente Rendezvous. Abrindo o ambiente, os recursos globais cruciais são iniciados. Fechando o ambiente esses recursos são destruídos. Todo programa que efetuar comunicação pelo Rendezvous deve abrir um ambiente antes da comunicação e antes do término do programa, depois de concluída a comunicação, deve fechar o ambiente. Os comandos de abrir e fechar ambiente, nas linguagens que foram utilizadas nesse projeto, são: [TIBCO, 2012]

- C: abrir – tibrv_Open(); / fechar tibrv_Close();
- Java: abrir – Tibrv.open(); / fechar Tibrv.close();
- BW: possui programação visual e faz de forma transparente.

Os programas usam o objeto transporte para enviar e receber mensagens. O transporte é composto por três parâmetros: *service*, *network* e *daemon*. Em redes simples, as configurações padrões desses parâmetros são suficientes. [TIBCO, 2012]

Para enviar mensagens, são possíveis três opções: [TIBCO, 2012]

- Enviar uma mensagem. Comando C: tibrvTransport_Send(); Java: TibrvTransport.send();
- Enviar uma mensagem resposta a uma mensagem recebida. Comando C: tibrvTransport_SendReply(); Java: TibrvTransport.sendReply();
- Enviar uma mensagem e aguardar uma resposta. Comando C: tibrvTransport_SendRequest(); Java: TibrvTransport.sendRequest();

Para receber mensagens é necessário criar um evento de recebimento, conforme abaixo: [TIBCO, 2012]

- C: tibrvEvent_CreateListener();
- Java: TibrvListener();

3- IMPLANTAÇÃO

Nesse capítulo, é descrita a construção e programação do protótipo e dos softwares de análise. Para o funcionamento do protótipo foram implementados um banco de dados, uma interface de cadastro dos karts, pilotos e corridas, um programa para coleta de dados, *WebServices* para inserção no banco de dados, um programa para recebimento e transformação dos dados, um painel de análise *online* e um painel para análise *offline*.

Antes do início da coleta de dados do kart, é necessário fazer os cadastros de karts, pilotos e corridas. Com os cadastros efetuados, a telemetria pode ser iniciada. Os dados coletados pelos sensores são processados e enviados pelo Arduino. Esses dados são recebidos em um computador, onde eles são processados, salvos em um arquivo texto e exibidos no painel de análise *online*. Ao finalizar a telemetria, os dados do arquivo texto são salvos no banco de dados e ficam disponíveis para o painel de análise *offline*.

3.1 HARDWARE

O protótipo desenvolvido neste trabalho é formado por dispositivos eletrônicos que são capazes de capturar, processar e transmitir as informações importantes para a análise do equipamento, piloto ou corrida. Os dados de temperatura, movimentação do volante, posição dos pedais de aceleração e frenagem e velocidade são capturados pelos dispositivos. Esses dados necessitam de um processamento para transformar tensão em informações úteis e então serem transmitidos para um computador, onde são feitas as análises online e offline. As ligações dos sensores e da *WifiShield* são mostradas na Figura 3.1 e o diagrama de blocos do Hardware é mostrado na Figura 3.2.

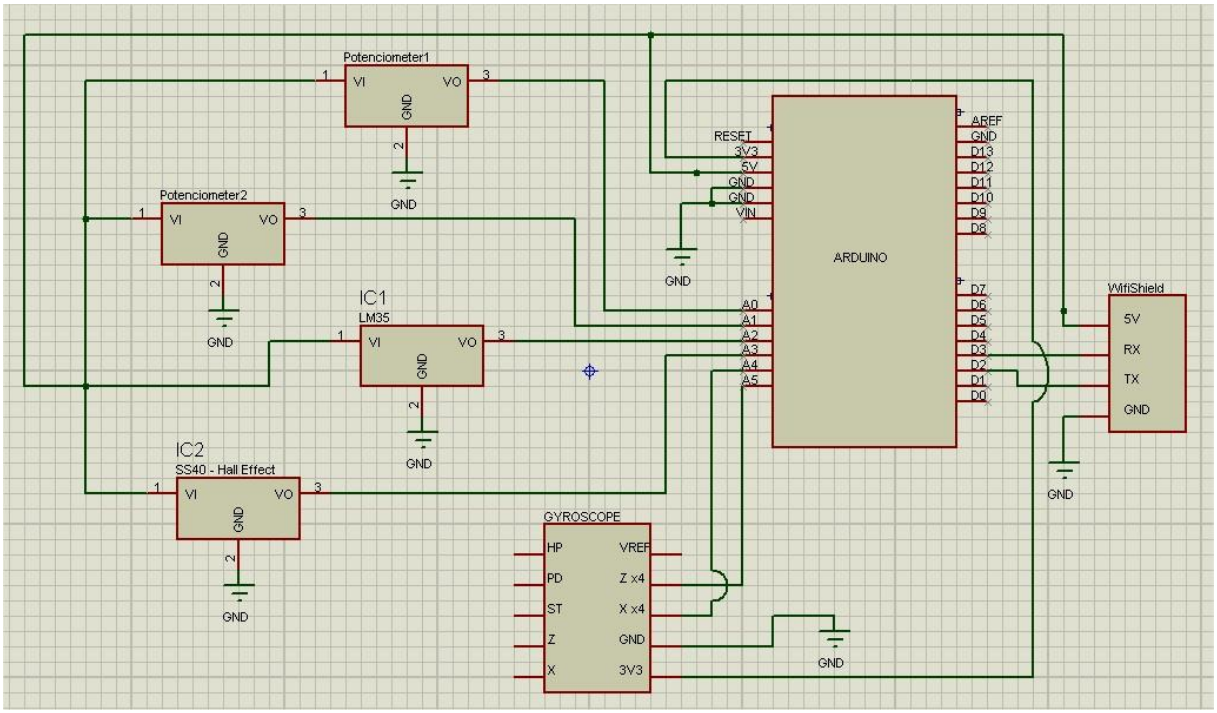


Figura 3.1 – Esquema das ligações do Arduino

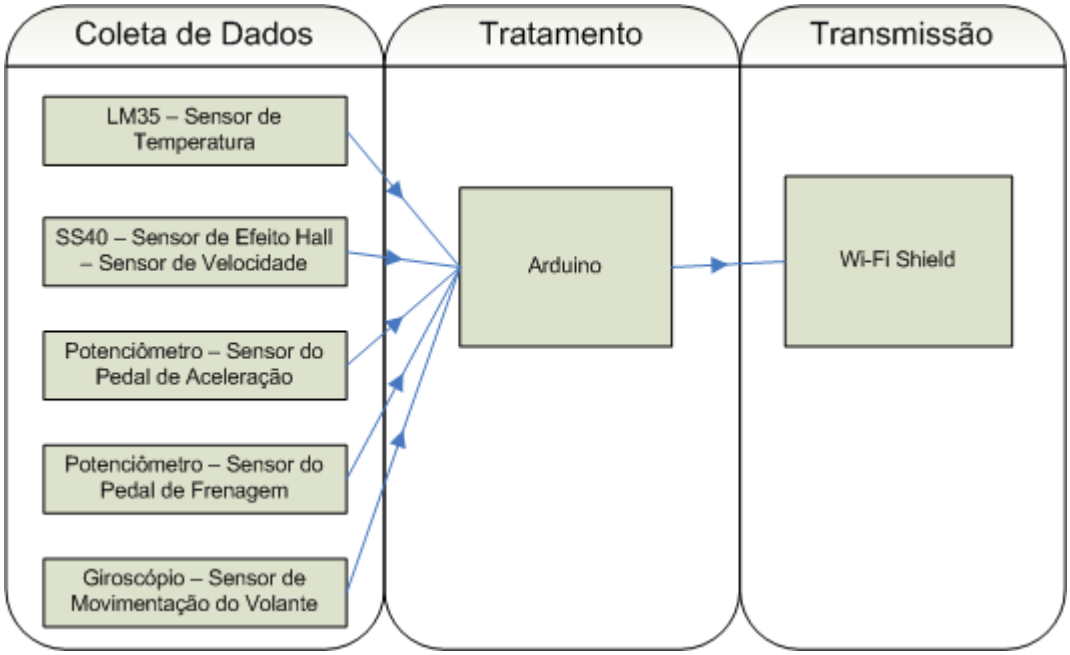


Figura 3.2 – Diagrama de Blocos do Hardware

No programa desenvolvido, primeiramente são declaradas as bibliotecas necessárias e são definidos os pinos que são utilizados e as variáveis que recebem os dados. Então, é iniciado o comando *setup()* onde é definida a velocidade de comunicação do Arduino em 115200 bits/segundo com a função “Serial.init(115200)” e configurada a comunicação WiFi,

que é detalhada no tópico 3.1.2. Depois é iniciado o comando *loop()* onde são recebidos os dados dos sensores e preparados para o envio pela WiFi. Para preparar os dados enviados, os dados coletados são agrupados em uma string através do comando "string.concat()" e a string é transformada em char através do comando "string.toCharArray(char,length);".

Por fim, foi criado um delay para diminuir a quantidade de dados enviados. O código do programa é apresentado no APÊNDICE C.

3.1.1 TRANSFORMAÇÃO DE DADOS LIDOS PELOS SENSORES

Os dados lidos pelos sensores precisaram ser tratados antes de serem atribuídos às variáveis, pois todos os valores recebidos dos sensores possuem uma resolução de 10 bits, variando de 0 a 1023.

Para medir a posição dos pedais de aceleração e frenagem variando de 0 a 100, foi necessário fazer uma divisão por 10,23 e do resultado foi atribuída a parte inteira do número às variáveis Acelerator e Brake, para posição de acelerador e freio, respectivamente.

Para converter o valor recebido pelo sensor de temperatura em graus Celsius foi utilizada a função "map(0, 1023, 0, 500)" e desse resultado, foi atribuída a parte inteira à variável Temperature. Outra forma de chegar à temperatura em ºC é a seguinte: a variação de 0 a 1023 é correspondente à variação de tensão do sensor de 0 a 5V e a cada 10ºC a tensão varia em 100mV, então a temperatura = (valor lido*(5/1023))*100.

A variável de Speed funciona como um indicador, quando o valor lido pelo sensor de efeito hall é superior à 1000, a variável recebe o valor 1, caso contrário, permanece com o valor 0. Essa informação é utilizada posteriormente na integração para calcular a velocidade.

A movimentação do volante é armazenada na variável z que é atribuída ao valor lido do eixo z do giroscópio menos 250. Esse valor foi escolhido pois, quando o giroscópio está em repouso, o valor lido no eixo z oscila próximo a 250, então esse valor é reduzido do valor lido para que quando não ocorra movimentação no volante, o valor da variável seja 0.

3.1.2 ENVIO DE DADOS PELA WI-FI

Para este projeto, foi necessário adicionar uma biblioteca para fazer a comunicação via WiFi, a *Wifly.h*, que é mostrada no ANEXO 1. A biblioteca utiliza o arquivo de código *Wifly*.cpp que sofreu uma alteração na função "`WiflyClass::init()`" para que os dados dos sensores funcionem corretamente, o código alterado é mostrado no APÊNDICE F. A Biblioteca *Wifly*.h também faz chamada de outras duas bibliotecas: *Hardware.h*, que é mostrada no ANEXO 2 e *debug.h*, que é mostrada no ANEXO 3.

É através da *Wifly.h* que o Arduino utiliza a Shield WiFi para enviar e receber dados. O comando "WiflyClass Wifly(int,int);" define quais o pinos que são utilizados pela *shield* para realizar o recebimento(RX) e transmissão(TX) dos dados. Através do comando "Wifly.init();" a shield é iniciada e configurada. No comando "Wifly.setConfig(SSID, password);", são passados os parâmetros para o Arduino conectar na rede WiFi, onde SSID é o nome da rede e password é a senha de conexão. Após receber os parâmetros o Arduino conecta na rede através do comando "Wifly.join(SSID);" e o comando "Wifly.checkAssociated();" verifica se o Arduino já está conectado na rede. Depois de conectado na rede, o Arduino fica em um *loop* até que conecte no serviço TCP com o comando "while(!Wifly.connect(ip,port));". O Arduino envia os dados por TCP através do comando "Wifly.writeToSocket(const char);".

3.2 SOFTWARE

O Software desenvolvido neste trabalho é composto por páginas *Web*, um processo de integração no BusinessWorks, painéis de análise *online*, no RTView, e *offline*, no Spotfire, e um Banco de Dados. A organização dos programas no projeto é ilustrada na Figura 3.3.

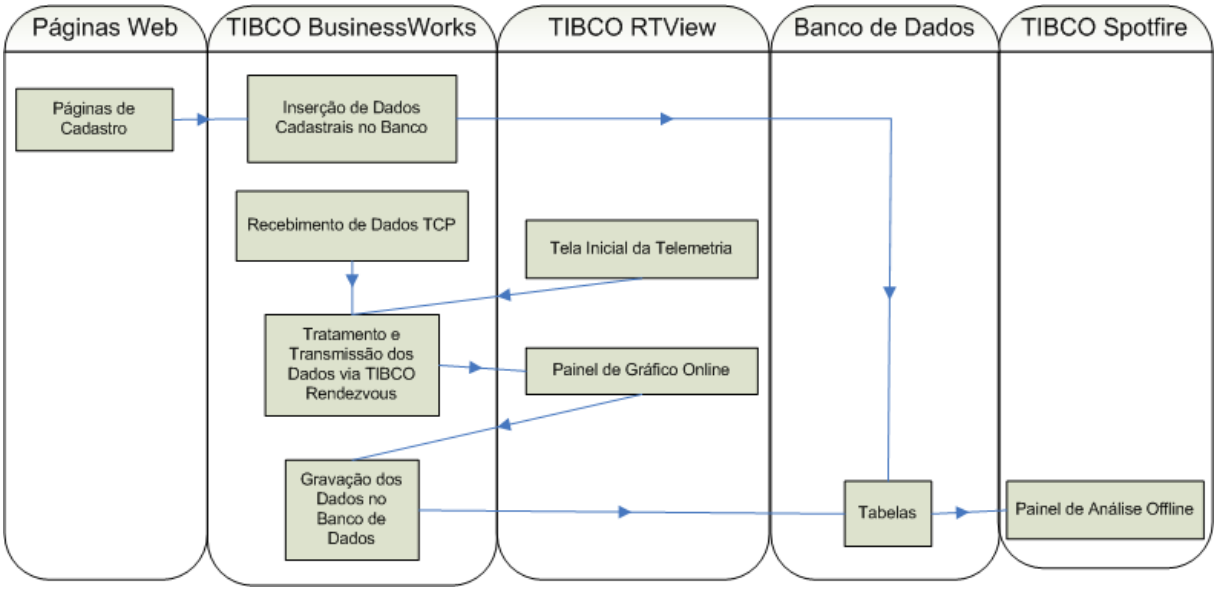


Figura 3.3 – Diagrama de Blocos do Software

3.2.1 INTERFACE DE CADASTRO

A interface de cadastros consiste em quatro páginas em HTML: home, kart, piloto e corrida. A página home é bastante simples, possui uma imagem, que aparece em todas as páginas, três botões que chamam as outras três. Na Figura 3.4 é ilustrada a codificação da página e na Figura 3.5 é mostrado o *design* da página.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta content="pt-br" http-equiv="Content-Language" />
<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
<title>Telemetria - Inicial</title>
</head>

<body>
<center>
<img alt="" height="169" src="V1.0/imagens/bandeira.jpg" width="237" /><br />
<label id="lblTitulo" style="font-family: 'Alpine 7558S'; font-size: 50px">Telemetria</label>
<br /><br /><br /><br />
<table>
    <tr>
        <form action="piloto.html" method="post">
        <input name="piloto" type="submit" value="Cadastrar Piloto" />
        </form>
    </tr>
    <tr>
        <form action="kart.html" method="post">
        <input name="kart" type="submit" value="Cadastrar Kart" />
        </form>
    </tr>
    <tr>
        <form action="corrida.html" method="post">
        <input name="corrida" type="submit" value="Cadastrar Corrida" />
        </form>
    </tr>
</table>
</center>
</body>
</html>
```

Figura 3.4 – Código HTML da página home

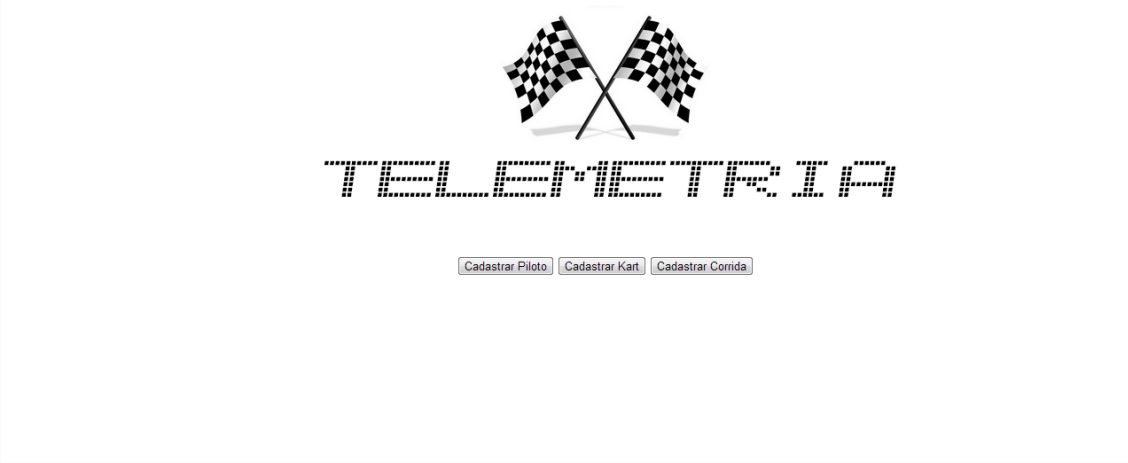


Figura 3.5 – Design da página home

A página kart é um formulário onde é informado o número do kart, o nome do kartódromo, o número do sensor que está instalado no kart, um *checkbutton* para a opção de cadastrar um novo kart, e um parâmetro que informa qual tipo de cadastro está sendo feito, nesse caso, cadastro de kart.

Os dados são enviados através do método post para a porta 2222. Na Figura 3.6 é ilustrada a codificação e na Figura 3.7 é ilustrado o *design*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
<title>Telemetria  Cadastro Kart</title>
</head>

<body>
<center>
<form action="http://localhost:2222" method="post">
<img alt="" height="169" src="imagens/bandeira.jpg" width="237" /><br />
<label id="lblTitulo" style="font-family: 'Alpine 7558S'; font-size: 50px">Telemetria</label><br /><br />
<label id="lblSubTitulo" style="font-family: 'Alpine 7558S'; font-size: 30px">Cadastro Kart</label><br /><br />
<table>
    <tr>
        <td>
            <label id="lblNumber">Número:</label>
        </td>
        <td>
            <input name="number" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblKarting">Kartódromo:</label>
        </td>
        <td>
            <input name="karting" type="text" />
        </td>
    </tr>
        <tr>
        <td>
            <label id="lblSensor">Número do Sensor:</label>
        </td>
        <td>
            <input name="sensor" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblNew">Cadastrar outro?</label>
        </td>
        <td>
            <input name="new" type="checkbox" value="yes" />
        </td>
    </tr>
</table>
<input name="type" type="hidden" value="kart" />
<br /><input name="Submit" type="submit" value="Enviar" />
</form>
</center>
</body>
</html>
```

Figura 3.6 – Codificação da página kart

Figura 3.7 – Design da página kart

A página piloto também é um formulário onde são informados os dados do piloto: nome, apelido, categoria, idade, sexo e peso. Existe também um *checkbutton* para a opção de cadastrar um novo piloto e um parâmetro que informa qual tipo de cadastro está sendo feito, nesse caso, cadastro de piloto.

Os dados são enviados através do método post para a porta 2222. Na Figura 3.8 é ilustrada a codificação e na Figura 3.9 é ilustrado o *design*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
<title>Telemetria  Cadastro Piloto</title>
</head>

<body>
<center>
<img alt="" height="169" src="imagens/bandeira.jpg" width="237" /><br />
<label id="lblTitulo" style="font-family: 'Alpine 7558S'; font-size: 50px">Telemetria</label><br /><br />
<label id="lblSubTitulo" style="font-family: 'Alpine 7558S'; font-size: 30px">Cadastro Piloto</label><br /><br />
<form action="http://localhost:2222" method="post">
<table>
    <tr>
        <td>
            <label id="lblName">Nome:</label>
        </td>
        <td>
            <input name="name" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblNickname">Apelido:</label>
        </td>
        <td>
            <input name="nickname" type="text" />
        </td>
                <td align="left">
        <label>(sem espaços)</label>
        </td>

    </tr>
    <tr>
        <td>
            <label id="lblCategory">Categoria:</label>
        </td>
        <td>
            <input name="category" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblAge">Idade:</label>
        </td>
        <td>
            <input name="age" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblGender">Sexo:</label>
        </td>
        <td>
            <input name="gender" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblWeight">Peso:</label>
        </td>
        <td>
            <input name="weight" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblNew">Cadastrar outro?</label>
        </td>
        <td>
            <input name="new" type="checkbox" value="yes" />
        </td>
    </tr>
</table>
<input name="type" type="hidden" value="piloto" />
<br /><input name="Submit" type="submit" value="Enviar" />
</form>
</center>
</body>

</html>
```

Figura 3.8 – Codificação da página piloto

Figura 3.9 - Design da página piloto

A página corrida também é um formulário onde são informados os dados da corrida: nome da corrida, apelidos dos pilotos, karts, data, kartódromo e um parâmetro que informa qual tipo de cadastro está sendo feito, nesse caso, cadastro de corrida.

Os dados são enviados através do método post para a porta 2222. Na Figura 3.10 é ilustrada a codificação e na Figura 3.11 é ilustrado o *design*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
<title>Telemetria  Cadastro Corrida</title>
</head>

<body>
<center>
<img alt="" height="169" src="imagens/bandeira.jpg" width="237" /><br />
<label id="lblTitulo" style="font-family: 'Alpine 7558S'; font-size: 50px">Telemetria</label><br /><br />
<label id="lblSubTitulo" style="font-family: 'Alpine 7558S'; font-size: 30px">Cadastro Corrida</label><br /><br />
<form action="http://localhost:2222" method="post">
<table>
    <tr>
        <td>
            <label id="lblName">Nome da Corrida:</label>
        </td>
        <td>
            <input name="name" type="text" />
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblRacers">Apelido Pilotos:</label>
        </td>
        <td>
            <input name="nickname" type="text" />
        </td>
        <td align="left">
        <label>(separados por vírgula)</label>
        </td>

    </tr>
    <tr>
        <td>
            <label id="lblkart">Karts:</label>
        </td>
        <td>
            <input name="kart" type="text" />
        </td>
        <td>
        <label>(mesma ordem dos pilotos e separados por vírgula)</label>
        </td>
    </tr>
    <tr>
        <td>
            <label id="lblDate">Data:</label>
        </td>
        <td>
            <input name="date" type="text"  />
        </td>
        <td align="left">
        <label>(formato: DD/MM/YYYY)</label>
        </td>

    </tr>
    <tr>
        <td>
            <label id="lblKarting">Kartódromo:</label>
        </td>
        <td>
            <input name="karting" type="text" />
        </td>
    </tr>

</table>
<br />
<input name="type" type="hidden" value="corrida" />
<input name="Submit" type="submit" value="Cadastrar" />
</form>
</center>
</body>

</html>
```

Figura 3.10 – Codificação da página corrida

Figura 3.11 – Design da página corrida

3.2.2 BANCO DE DADOS

Foi criado um banco de dados no Oracle Database 10g para este projeto. Ele é composto por quatro tabelas: Karts, Pilotos, Corridas e Dados. A tabela Karts é utilizada para armazenar os dados cadastrais dos karts. A tabela Pilotos para armazenar os dados cadastrais dos pilotos. A tabela Corridas para armazenar as corridas e a tabela Dados para armazenar os dados coletados no kart.

Os campos das tabelas, os tipos de dados e relacionamentos entre tabelas são ilustrados na Figura 3.12. A Figura 3.13 ilustra a tabela Karts, a Figura 3.14 ilustra a tabela Pilotos, a Figura 3.15 ilustra a tabela Corridas e a Figura 3.16 ilustra a tabela Dados, criadas no banco de dados.

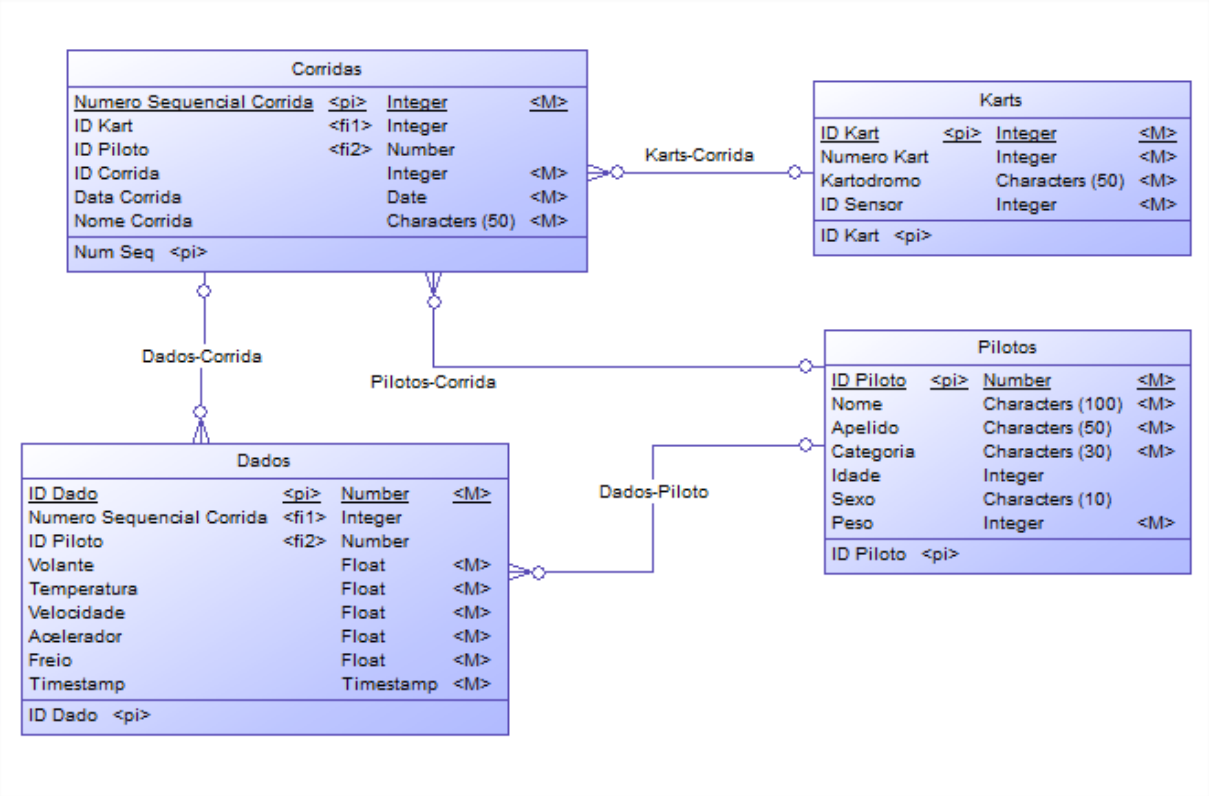


Figura 3.12 – Modelo Lógico do Banco de Dados

KARTS

Colunas | Dados | Restrições | Grants | Estatísticas | Triggers | Dependências | Detalhes | Partições | Índices | SQL

Ações...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID_KART	NUMBER(38,0)	No	(null)	1	(null)
NUM_KART	NUMBER(38,0)	No	(null)	2	(null)
KTDMO	VARCHAR2(50 BYTE)	No	(null)	3	(null)
ID_SENSOR	NUMBER(38,0)	No	(null)	4	(null)

Figura 3.13 – Tabela Karts

PILOTOS

Colunas | Dados | Restrições | Grants | Estatísticas | Triggers | Dependências | Detalhes | Partições | Índices | SQL

Ações...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID_PILOTO	NUMBER	No	(null)	1	(null)
NOME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
APELIDO	VARCHAR2(20 BYTE)	No	(null)	3	(null)
CATEGORIA	VARCHAR2(20 BYTE)	No	(null)	4	(null)
IDADE	NUMBER(38,0)	Yes	(null)	5	(null)
SEXO	VARCHAR2(10 BYTE)	Yes	(null)	6	(null)
PESO	NUMBER(38,0)	No	(null)	7	(null)

Figura 3.14 – Tabela Pilotos

CORRIDAS

Colunas | Dados | Restrições | Grants | Estatísticas | Triggers | Dependências | Detalhes | Partições | Índices | SQL

Ações...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
NUM_SQC_CORRIDA	NUMBER(38,0)	No	(null)	1	(null)
ID_KART	NUMBER(38,0)	Yes	(null)	2	(null)
ID_PILOTO	NUMBER	Yes	(null)	3	(null)
ID_CORRIDA	VARCHAR2(20 BYTE)	No	(null)	4	(null)
DT_CORRIDA	DATE	No	(null)	5	(null)
NM_CORRIDA	VARCHAR2(50 BYTE)	No	(null)	6	(null)

Figura 3.15 – Tabela Corridas

DADOS

Colunas | Dados | Restrições | Grants | Estatísticas | Triggers | Dependências | Detalhes | Partições | Índices | SQL

Ações...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID_DADO	NUMBER	No	(null)	1	(null)
NUM_SQC_CORRIDA	NUMBER(38,0)	Yes	(null)	2	(null)
ID_PILOTO	NUMBER	Yes	(null)	3	(null)
VOLANTE	FLOAT	No	(null)	4	(null)
TEMPERATURA	FLOAT	No	(null)	5	(null)
VELOCIDADE	FLOAT	No	(null)	6	(null)
ACELERADOR	FLOAT	No	(null)	7	(null)
FREIO	FLOAT	No	(null)	8	(null)
TS_DADO	TIMESTAMP(3)	No	(null)	9	(null)

Figura 3.16 – Tabela Dados

3.2.3 INTEGRAÇÃO

Para desenvolver a aplicação que recebe e transforma os dados, foi escolhida a ferramenta TIBCO BusinessWorks por causa da facilidade e velocidade de desenvolvimento.

A aplicação é responsável pela integração do projeto, as atividades executadas por ela são:

- Recebimento dos dados inseridos na página web e inserção no banco de dados;
- Recebimento dos dados enviados do kart por TCP, transformação desses dados, envio para o programa de análise online pelo TIBCO Rendezvous e armazenamento dos dados em um arquivo texto
- Recebimento de mensagem enviada pelo TIBCO RTView com a informação do início da coleta de dados e com isso, começa a armazenar os dados no arquivo texto
- Recebimento de mensagem enviada pelo TIBCO RTView com a informação da conclusão da corrida e com isso, armazenamento os dados do arquivo texto no banco de dados.

Para toda a aplicação de integração, são utilizadas algumas atividades compartilhadas, que são as conexões, esquemas de XML, *DataFormats* e Variáveis Compartilhadas (Shared Variable). Essas atividades são detalhadas no tópico seguinte.

3.2.3.1 RECURSOS COMPATILHADOS

3.2.3.1.1 CONEXÕES

Para que o TIBCO BusinessWorks se comunique com outras aplicações, é necessário criar atividades de conexões. Nessas atividades são informados os parâmetros de conexão que podem ser utilizados por todas as atividades que necessitem desses parâmetros. Neste projeto, foram configuradas as conexões:

- Conexão com o banco de dados, através da atividade JDBC Connection;

- Conexão com o TIBCO Rendezvous, através da atividade RV Transport;
- Conexão HTTP para receber dados da *web* e para o *WebService*, em ambos os casos foi utilizada a atividade HTTP Connection;
- Conexão TCP, para recebimento de dados do kart.

Para a conexão com o banco de dados, foi utilizado o JDBC da Oracle e informados os seguintes parâmetros necessários: URL do banco de dados, usuário e senha. A atividade criada para armazenar esses parâmetros recebeu o nome de JDBCConnection.

Para a conexão com o TIBCO Rendezvous, foram informados os seguintes parâmetros: Network, Service e Daemon. A atividade criada para armazenar esses parâmetros recebeu o nome de RVTransport.

Para as conexões HTTP e TCP, foram informados o *host* e a porta. A conexão HTTP de comunicação com as páginas *web* foi configurada através da atividade chamada HTTPConection, a conexão HTTP do *WebService* foi configurada através da atividade chamada HTTPSOAPConection e a conexão TCP foi configurada através da atividade chamada TCPKART.

3.2.3.1.2 XSD – XML SCHEMA DEFINITION

Foram criados os XSDs para padronizar a maneira em que os dados trafegam entre as atividades do TIBCO BusinessWorks. Neste projeto, foram criados seis esquemas:

- Driver: possui os campos de informações cadastrais do piloto, conforme é mostrado na Figura 3.17.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/DriverSchema.xsd"
      targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/DriverSchema.xsd"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
      <xs:element name="driver">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="Name" type="xs:string"/>
                        <xs:element name="Nickname" type="xs:string"/>
                        <xs:element name="Category" type="xs:string"/>
                        <xs:element name="Age" type="xs:string"/>
                        <xs:element name="Gender" type="xs:string"/>
                        <xs:element name="Weight" type="xs:string"/>
                        <xs:element name="New" type="xs:string" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Figura 3.17 – XSD dos dados de piloto

- Error: possui os campos necessários para identificar o erro ocorrido, conforme é mostrado na Figura 3.18.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/ErrorSchema.xsd"
      targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/ErrorSchema.xsd"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
      <xs:element name="Error">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="Msg" type="xs:string"/>
                        <xs:element name="MsgCode" type="xs:string"/>
                        <xs:element name="Process" type="xs:string"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Figura 3.18 – XSD dos dados de erro

- Kart: possui os campos de informações cadastrais do kart, conforme é mostrado na Figura 3.19.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/KartSchema.xsd"
      targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/KartSchema.xsd"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
     <xs:element name="kart">
          <xs:complexType>
               <xs:sequence>
                    <xs:element name="Number" type="xs:string"/>
                    <xs:element name="Karting" type="xs:string"/>
                    <xs:element name="Sensor" type="xs:string"/>
                    <xs:element name="New" type="xs:string" minOccurs="0"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

Figura 3.19 – XSD dos dados de cadastro de kart

- KartData: possui os campos dos dados que são enviados pelo protótipo, conforme é mostrado na Figura 3.20.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/KartDataSchema.xsd"
      targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/KartDataSchema.xsd"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
     <xs:element name="root">
          <xs:complexType>
               <xs:sequence>
                    <xs:element name="IdSensor" type="xs:int"/>
                    <xs:element name="SteeringWheel" type="xs:int"/>
                    <xs:element name="Acelerator" type="xs:int"/>
                    <xs:element name="Brake" type="xs:int"/>
                    <xs:element name="Temperature" type="xs:int"/>
                    <xs:element name="Speed" type="xs:int"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

Figura 3.20 – XSD dos dados de telemetria do kart

- Race: possui os campos de informações cadastrais do corrida, conforme é mostrado na Figura 3.21.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/RaceSchema.xsd"
      targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/RaceSchema.xsd"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
     <xs:element name="race">
          <xs:complexType>
               <xs:sequence>
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="Nickname" type="xs:string"/>
                    <xs:element name="Kart" type="xs:string"/>
                    <xs:element name="Date" type="xs:string"/>
                    <xs:element name="Karting" type="xs:string"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

Figura 3.21 – XSD dos dados de cadastro de corrida

- Response: possui o campo de resposta do *WebService*, conforme é mostrado na Figura 3.22.

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/ResponseSchema.xsd"
      targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/ResponseSchema.xsd"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
     <xs:element name="response">
          <xs:complexType>
               <xs:sequence>
                    <xs:element name="Status" type="xs:string"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

Figura 3.22 – XSD dos dados de resposta do *WebService*

3.2.3.1.3 DATAFORMAT

Os DataFormats são utilizados no TIBCO BusinessWorks para definir os campos de saída para as atividades Parse Data, que lê uma string e a divide nos campos definidos. Para que essa divisão seja feita, é informado no DataFormat qual o separador de colunas utilizado. Neste projeto foram utilizados quatro DataFormats:

- FormatData: define os campos da string lida no arquivo texto. Campos: IdSensor, SteeringWheel, Temperature, Speed, Acelerator, Brake, Timestamp.
- FormatDriver: define os campos da string que foi informada na página *web* corrida, no campo “Apelido Pilotos:”. Neste DataFormat existe apenas um campo repetitivo chamado driver, é repetido uma vez para cada piloto informado.
- FormatKart: define os campos da string que foi informada na página *web* corrida, no campo “Karts:”. Neste DataFormat existe apenas um campo repetitivo chamado kart, é repetido uma vez para cada kart informado.
- FormatKartData: define os campos da string enviada pelo Arduino. Os campos deste DataFormat referenciam o XSD KartData.

3.2.3.1.4 SHARED VARIABLE

As variáveis compartilhadas são utilizadas quando um dado resultante de um processo é necessário na próxima execução do processo ou na execução de outro processo. Esses valores são armazenados nessas variáveis e podem ser consultados e/ou alterados por qualquer processo que precise. Neste projeto foram utilizadas três variáveis globais:

- RaceID: Armazena o ID da corrida e o ID utilizado para nomear o arquivo texto.
- Speed: Armazena o *timestamp* da última vez que o valor da variável Speed foi 1. Essa informação é necessária para o cálculo da velocidade.
- StartStop: Armazena o estado da telemetria, se está iniciado ou parado.

3.2.3.2 INSERÇÃO NO BANCO DE DADOS

A inserção no banco de dados pode ser dividida em duas partes. A primeira parte consiste em um processo que recebe os dados da *web* e chamada um *WebService* com uma solicitação SOAP. O *WebService* chama um dos três processos de inserção no banco: inserção de karts, inserção de pilotos e inserção de corridas. A segunda parte consiste na inserção dos dados coletados do kart no banco de dados.

Toda a estrutura do projeto desenvolvido no TIBCO BusinessWorks é ilustrada na Figura 3.23.

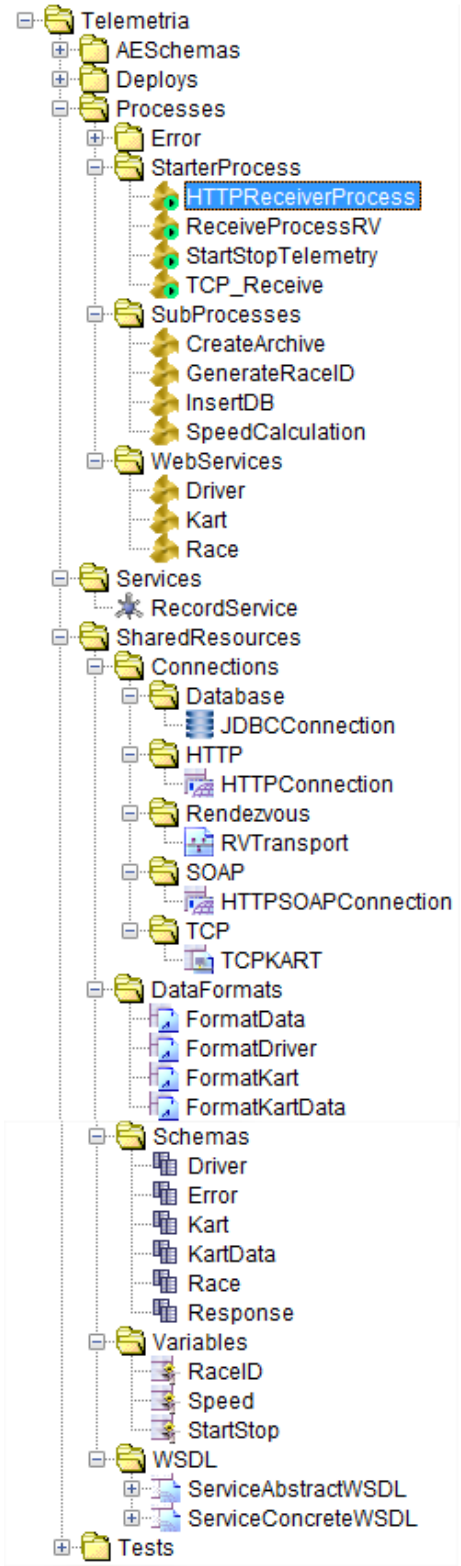


Figura 3.23 – Estrutura do projeto de integração

3.2.3.2.1 RECEBIMENTO DE DADOS CADASTRAIS

O processo, chamado de HTTPReceiverProcess, é iniciado com o recebimento do formulário enviado por uma das páginas *web*, a partir do parâmetro que informa o tipo de cadastro desejado, o processo decide a operação do *WebService* que será utilizada. O APÊNDICE A contém o WSDL utilizado para chamar o *WebService*. Depois que o *WebService* responde, o processo monta uma página *web* de resposta, que faz um redirecionamento dependendo das opções enviadas no formulário: caso seja um cadastro de kart e a opção “Cadastrar outro?” tenha sido escolhida, a página irá redirecionar para a página kart, caso a opção não tenha sido escolhida, a página irá redirecionar para a página home; caso seja um cadastro de piloto e a opção “Cadastrar outro?” tenha sido escolhida, a página irá redirecionar para a página piloto, caso a opção não tenha sido escolhida, a página irá redirecionar para a página home; caso seja um cadastro de corrida, a página irá redirecionar para a página home. A construção da página de resposta é mostrada na Figura 3.24.

```
concat(
'<!DOCTYPE html>',
'<html>',

'<head>',
'<meta content="text/html; charset=utf-8" http-equiv="Content-Type">',
'<meta http-equiv="refresh" content="5; url=http://localhost:8068/telemetry/Registration/',
if($HTTPReceiver/ProcessStarterOutput/parameters/new = 'yes') then $HTTPReceiver/ProcessStarterOutput/parameters/type else 'home',
'.html">',
'<title>Telemetria - Sucesso</title>',
'</head>',

'<body style="text-align: center">',
'<br>',
'<br>',
'<br>',
'<label id="lblTitulo" style="font-family:'," 'Alpine 7558S'",'; font-size: 50px">&tab;Telemetria</label>',
'<br>',
'<br>',
'<label id="lblMsg" style="font-family:'," 'Arcade'",'; font-size:30px"> Cadastro feito com Sucesso! </label>',
'<br>',
'<label id="lblMsg" style="font-family:'," 'Arial'",'; font-size:15px"> Você está sendo redirecionado.</label>',
'<br>',
'<label id="lblMsg" style="font-family:'," 'Arial'",'; font-size:15px"> Se estiver demorando,</label>',
'<a href="http://localhost:8068/telemetry/Registration/',
if($HTTPReceiver/ProcessStarterOutput/parameters/new = 'yes') then $HTTPReceiver/ProcessStarterOutput/parameters/type else 'home',
'.html">clique aqui.</a>',
'</body>',
'</html>')
```

Figura 3.24 – Criação da página de resposta

Foi criado um fluxo de exceção para o navegador Google Chrome, pois, ao enviar um formulário utilizando-o, ele envia uma segunda requisição sem nenhum parâmetro. Foi criado também um fluxo para captura de erros, que formata e escreve o erro em um arquivo texto.

O fluxo desse processo é ilustrado na Figura 3.25 e o fluxo de erro é ilustrado na Figura 3.26.

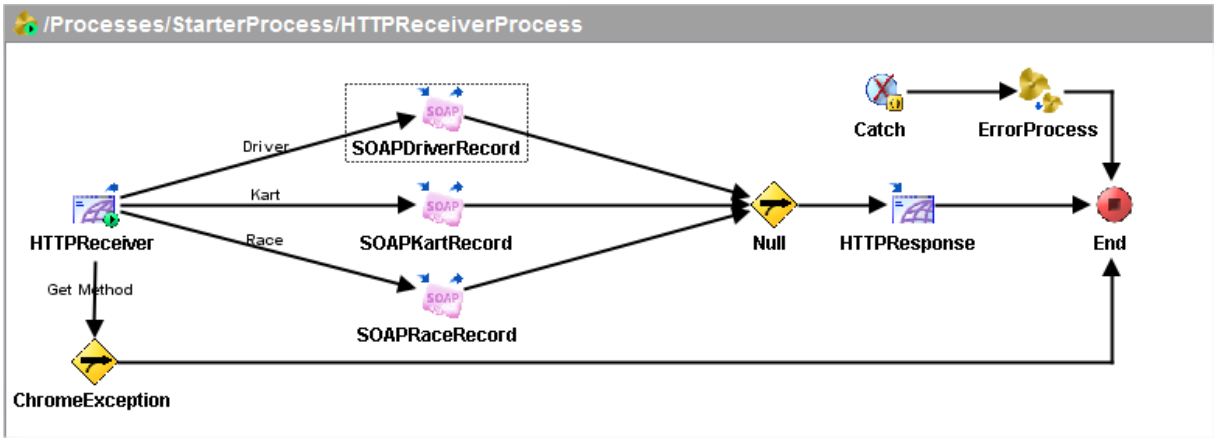


Figura 3.25 – Fluxo do processo HTTPReceiverProcess

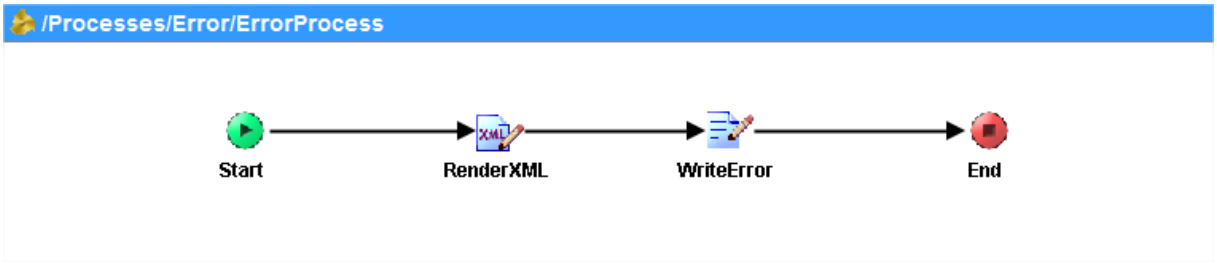


Figura 3.26 – Fluxo do processo de erro

O WebService foi criado com três operações, uma para cada tipo de cadastro. Cada operação chama um processo diferente conforme é mostrado na Figura 3.27. O APÊNDICE B mostra o WSDL utilizado pelo WebService.

O processo Kart é composto por uma atividade que é responsável pela inserção de informações de cadastro do kart, conforme é mostrado na Figura 3.28.

O processo Driver é composto por uma atividade que é responsável pela inserção de informações de cadastro do piloto, conforme é mostrado na Figura 3.29.

O processo Race é responsável pela inserção de informações de cadastro da corrida, conforme é mostrado na Figura 3.30. Ele é composto por uma chamada ao processo GenerateRaceID, que gera um identificador da corrida, conforme é mostrado na Figura 3.31. Depois são feitas três atividades de formatação de dados de kart e piloto: ExcludeSpaces (remove os espaços digitados no início e final da palavra, ParseKart (salva o número do kart

em um parâmetro que é repetitivo, é criado um parâmetro para cada número de kart) e ParseDriver (salva o apelido do piloto em um parâmetro que é repetitivo, é criado um parâmetro para cada apelido). Com os dados parametrizados, são executadas três atividades no banco: a primeira consulta o ID do piloto a partir do apelido, a segunda consulta o ID do kart a partir do número e kartódromo e a terceira insere os dados no banco de dados. O código de inserção é mostrado na Figura 3.32.

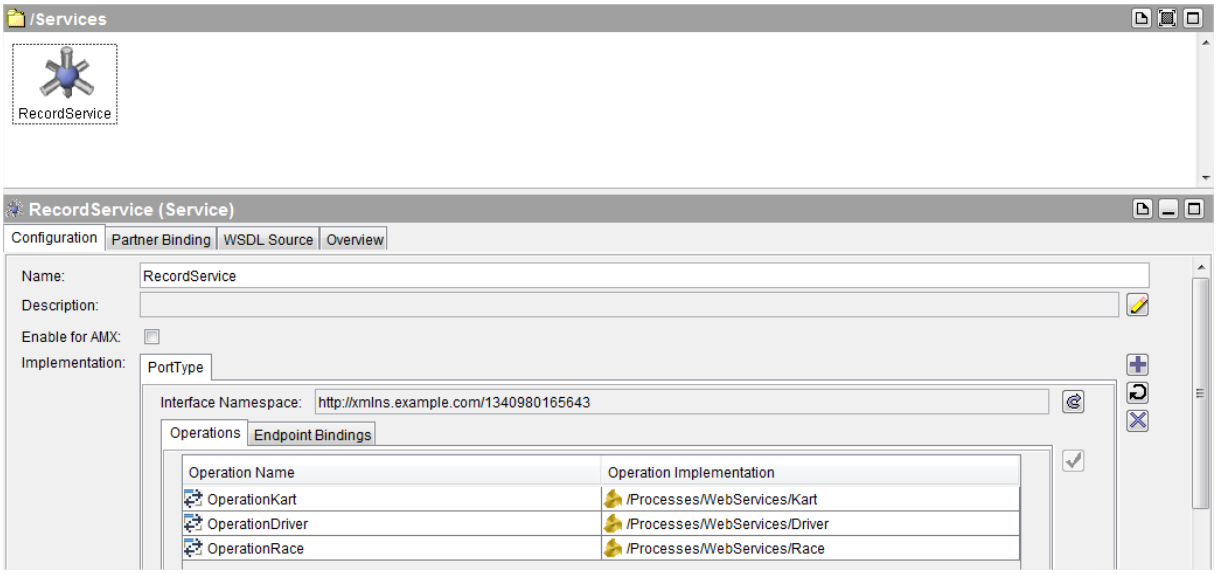


Figura 3.27 – Operações do WebService

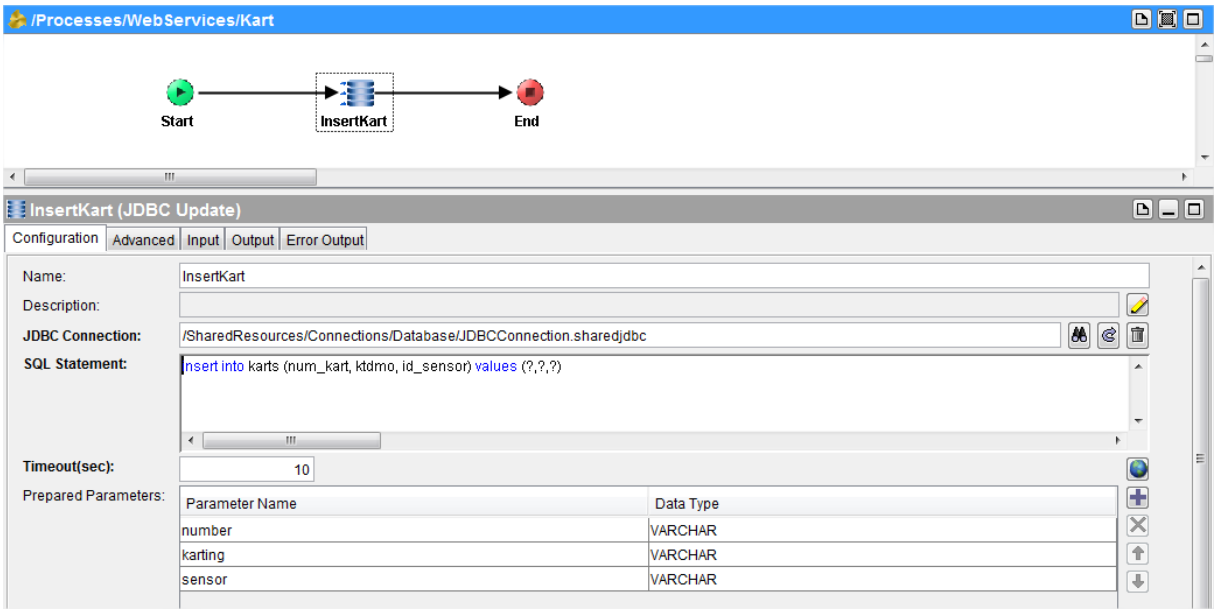


Figura 3.28 – Processo de inserção de kart

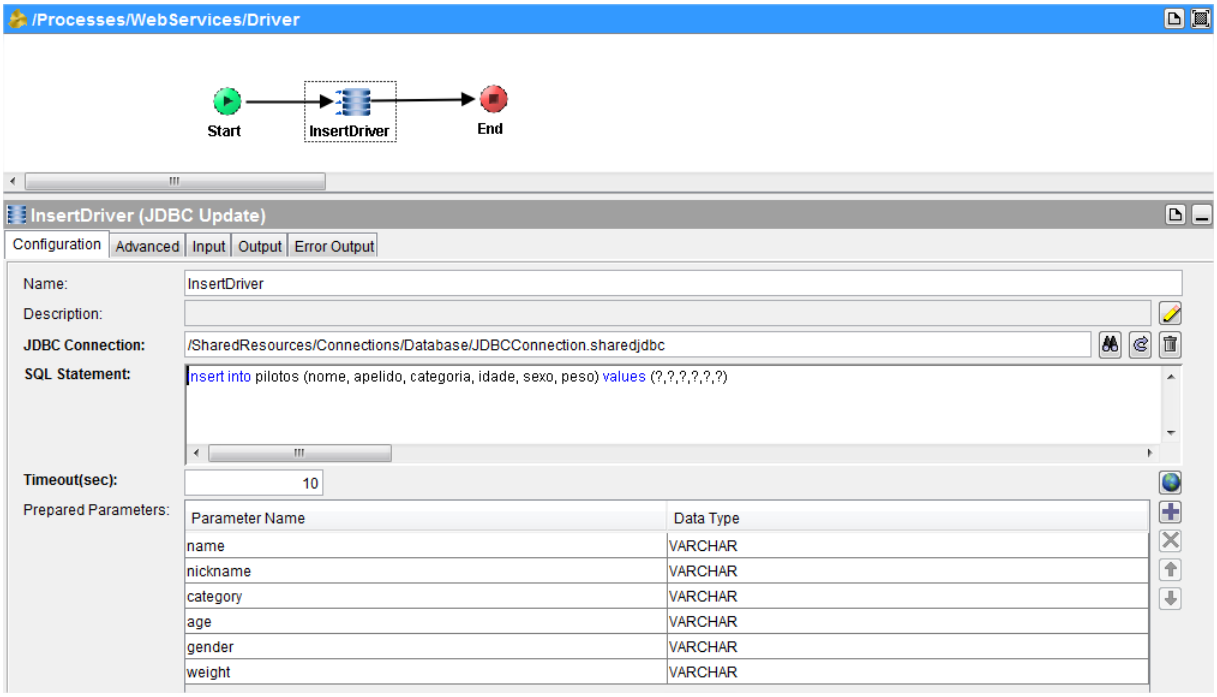


Figura 3.29 – Processo de inserção de piloto

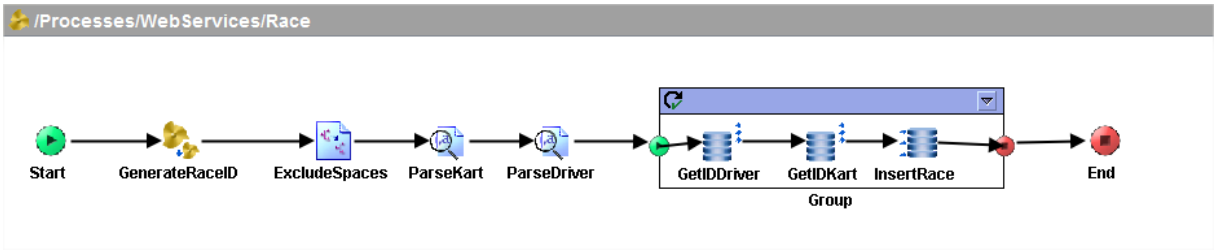


Figura 3.30 – Processo de inserção de corrida

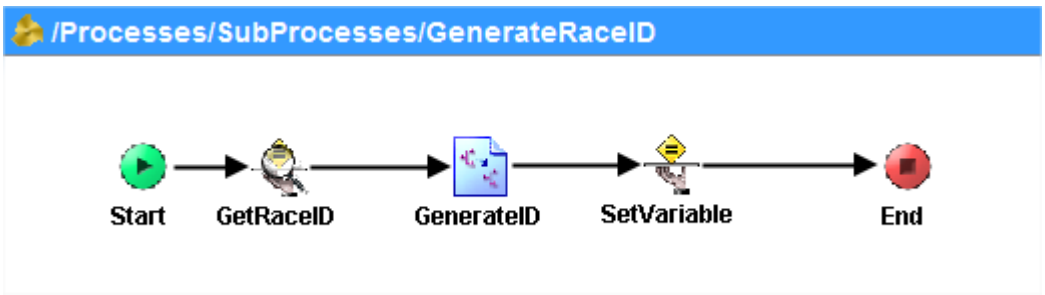


Figura 3.31 – Processo de geração de código identificador de corrida

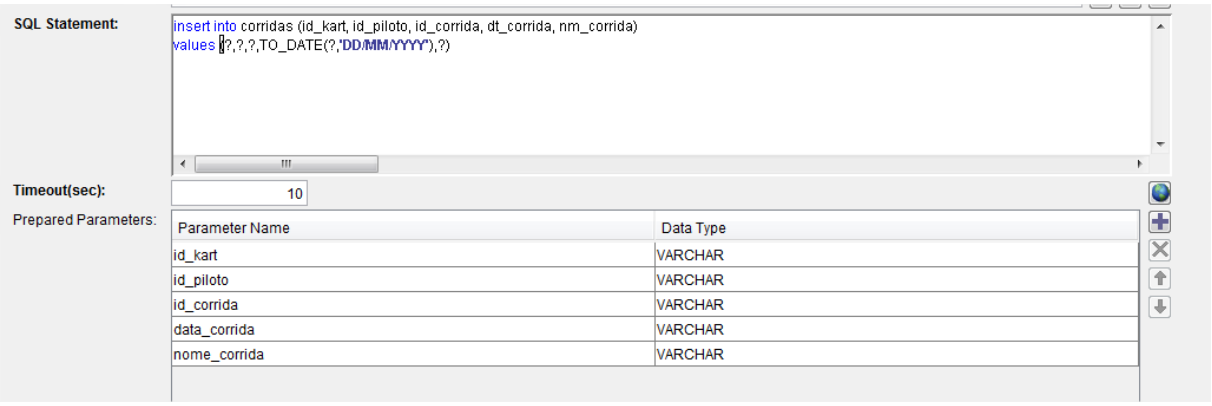


Figura 3.32 – Atividade de inserção de corrida

3.2.3.2.2 ARMAZENAMENTO DOS DADOS DO KART

O processo InsertDB é responsável pela inserção dos dados coletados durante a corrida no banco de dados, o fluxo do processo é ilustrado na Figura 3.33. Primeiramente, é executada uma atividade que coleta o identificador da corrida, com o identificador, é executada uma atividade para ler o arquivo texto que possui os dados coletados (é necessário o identificador da corrida, pois o nome do arquivo é formado com ele).

Após a leitura do arquivo, os dados são formatados e armazenados em variáveis repetitivas, é criado um grupo de variáveis para cada linha do arquivo. Então, são executadas três atividades de banco de dados dentro de um grupo de repetição, até que todas as linhas do arquivo tenham sido processadas.

A primeira atividade do banco é uma consulta ao ID do kart a partir do ID do sensor. A segunda atividade é uma consulta ao número sequencial da corrida e ID do piloto a partir do ID da corrida e o ID do kart. E a última atividade é a inserção dos dados coletados no banco. O código de inserção é ilustrado na Figura 3.34.

Foi criado um *catch* que faz com que o processo não faça nada caso o arquivo não exista.

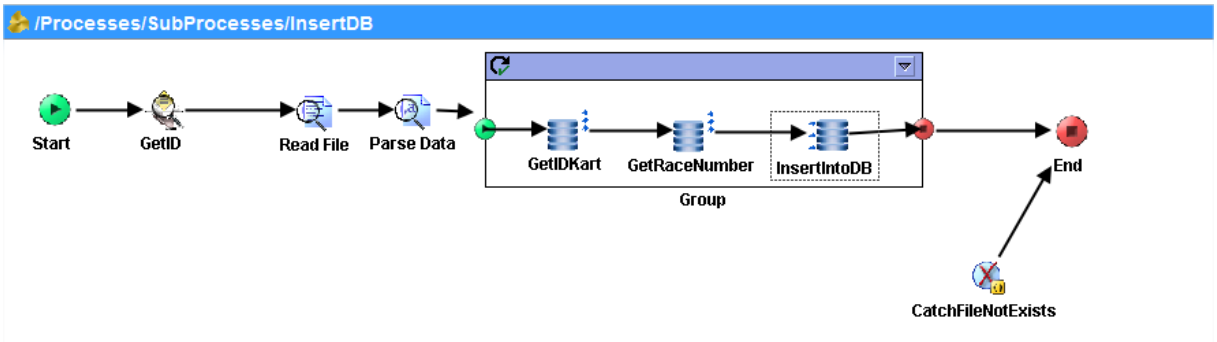


Figura 3.33 – Fluxo do processo InsertDB

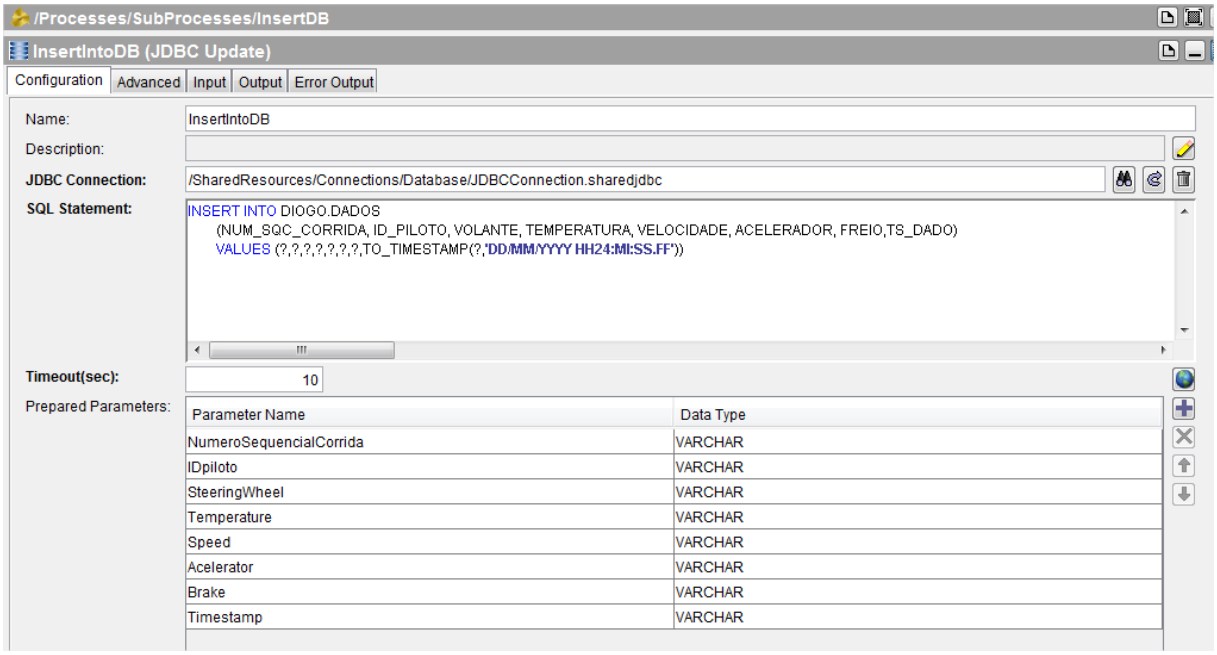


Figura 3.34 – Código de inserção de dados coletados

3.2.3.3 RECEBIMENTO E TRANSFORMAÇÃO DE DADOS

A transmissão de dados do kart para o computador ocorre por TCP e para que isso seja possível, foi criado um servidor TCP no TIBCO BusinessWorks. O processo que faz a função de servidor é chamado de TCP_Receive. Ele se inicia quando algum programa solicita a conexão com o servidor TCP.

A atividade que detecta a conexão TCP e inicia o processo é a TCP Receiver. Com a conexão estabelecida, inicia-se um *loop* infinito para recebimento dos dados. A primeira atividade, que é chamada de Read-Size, lê os primeiros quatro bytes da string, onde é informado o tamanho da string de dados enviada.

A próxima atividade lê a string utilizando a informação do tamanho capturada na atividade anterior. A string lida é compartilhada para o resto do processo no formato binário, então, foi criada uma atividade Map Data para converter a string de binário para caracteres novamente. A string convertida é quebrada e seus valores atribuídos à variáveis através da atividade ParseKartData.

A atividade GetStartIndicator verifica se a telemetria está iniciada. Se não estiver iniciada, o processo não faz mais nenhuma ação e volta ao início do *loop*. Se estiver iniciada, é chamado um processo para calcular a velocidade, o cálculo é feito utilizando a informação do timestamp atual e do timestamp salvo na última vez que a variável Speed foi definida como 1.Caso o valor da variável Speed seja 0, o processo chamado não faz nada.

Com todas as informações prontas, é enviada uma mensagem pelo TIBCO Rendezvous para o painel de análise online, onde os dados são exibidos.

Depois existem duas atividades para gravar esses dados em um arquivo. A atividade GetID busca o nome do arquivo para a atividade WriteFile encontrar o arquivo, caso já exista, ou criar o arquivo e escrever os dados. O fluxo desse processo é mostrado na Figura 3.35.

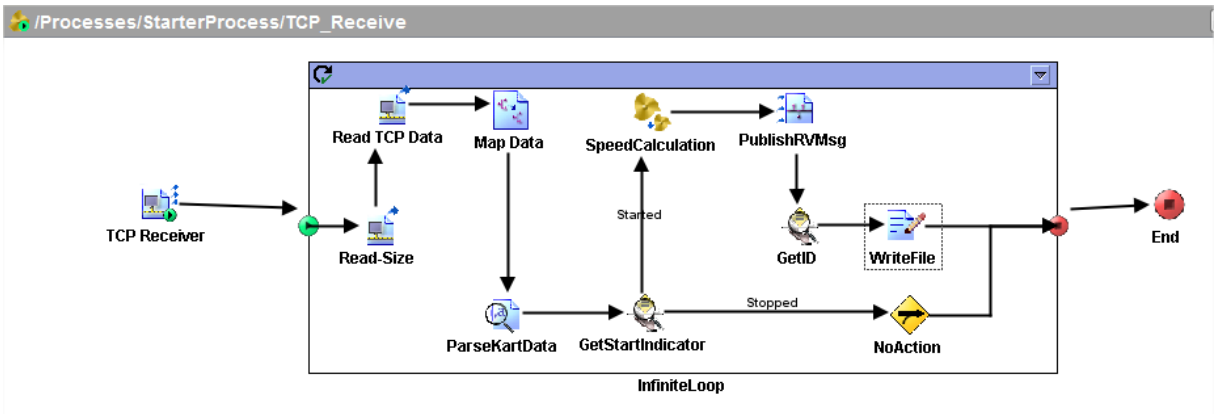


Figura 3.35 – Processo de recebimento de dados por TCP

3.2.3.4 INICIO E FIM DA TELEMETRIA

Foi criado um processo para definir quando a telemetria está iniciada e quando ela está finalizada. O processo chama-se StartStopTelemetry e é iniciado quando o TIBCO RTView envia uma mensagem para o BW. Na mensagem existe apenas um campo, chamado StartStop, do tipo boolean.

Caso o valor de StartStop seja ‘true’, então a telemetria é iniciada. O BW chama um processo para criar o arquivo onde, posteriormente os dados são gravados. Após isso, é verificada a existência de uma corrida cadastrada. Caso não exista, é enviada uma mensagem de resposta informando a inexistência de corrida cadastrada. Caso exista, é feita uma consulta no banco de dados para retornar o nome da corrida a partir do seu identificador, que é o nome do arquivo criado anteriormente. Após a consulta, a variável compartilhada StartStop é definida como ‘true’, é feita uma pausa de 1 segundo para estabilização e então, enviada uma mensagem de resposta com as seguintes informações: nome da corrida, estado da telemetria, identificador da cor.

Caso o valor de StartStop seja ‘false’, então a telemetria é finalizada. O BW define o valor da variável compatilhada StartStop como ‘false’, envia uma mensagem de resposta com os seguintes dados: estado da telemetria, identificador da cor e *Return*(valor é utilizado para habilitar o botão de retorno à página inicial no RTView). Depois de enviada a mensagem, o BW chama um processo para inserir os dados, que estavam no arquivo texto, no banco de dados.

O fluxo desse processo é mostrado na Figura 3.36.

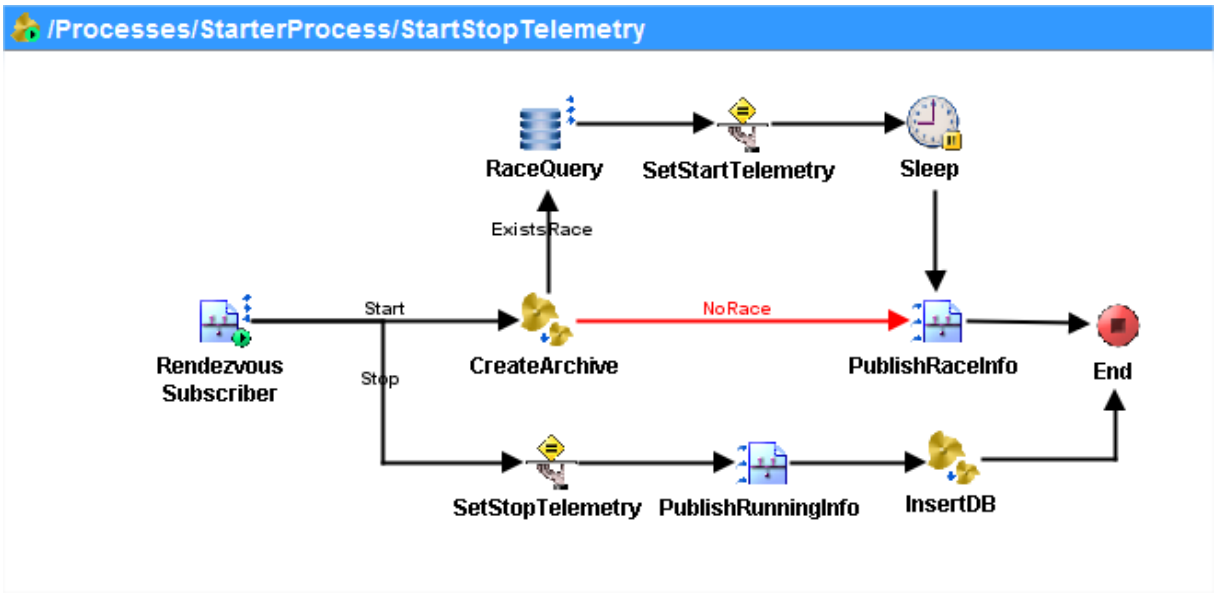


Figura 3.36 – Processo de início e término da telemetria

3.2.4 GRÁFICOS ONLINE

Para desenvolvimento do sistema de análise online, foi utilizado o programa TIBCO RTView. Esse programa foi escolhido pela facilidade de desenvolvimento de gráficos. Além dos gráficos, o sistema é responsável por chamar as páginas de cadastro e informar o início e término da telemetria.

Antes de começar a desenvolver a solução, foi necessário configurar o software para receber os dados do TIBCO Rendezvous na janela que é mostrada na Figura 3.37 com as configurações prontas.

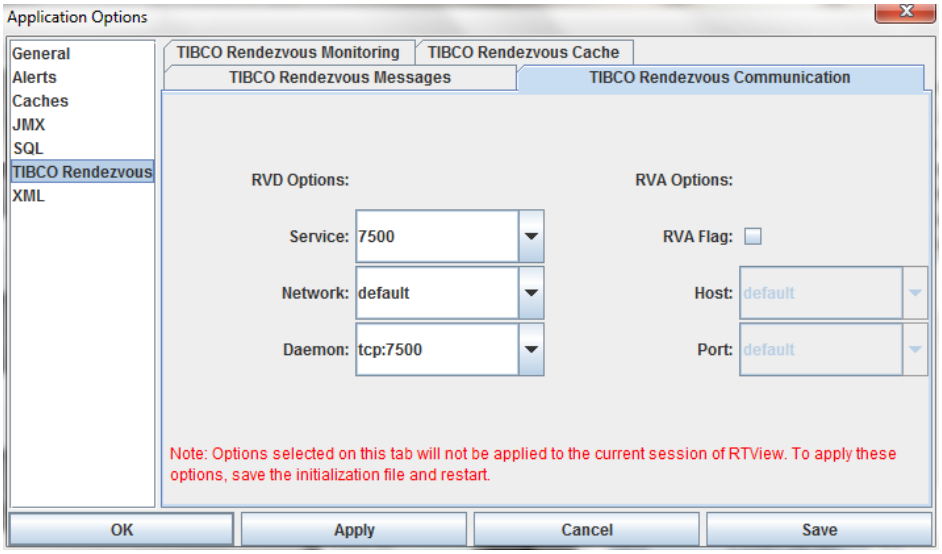


Figura 3.37 – Configuração do RV para o RTView

Com as configurações prontas, foi desenvolvido o painel “LiveInfoNew.rtv” com medidores, escalas e gráficos. Um medidor é utilizado para exibir a informação de velocidade, uma escala é utilizada para exibir a informação de temperatura, uma escala é utilizada para exibir a informação de movimentação do volante e duas escalas são utilizadas para exibir as informações de acelerador e freio. Também foram criados cinco gráficos, que exibem as informações anteriores em uma linha do tempo.

O medidor e o gráfico de velocidade foram configurados para receber os dados do campo “Speed” da mensagem do Rendezvous. Na Figura 3.38 é mostrada a configuração do medidor e do gráfico. Também foi configurado para o medidor exibir velocidades até 70

km/h, para que a escala seja verde de 0 a 60km/h, amarela entre 60 e 65 km/h e vermelha acima de 65 km/h.

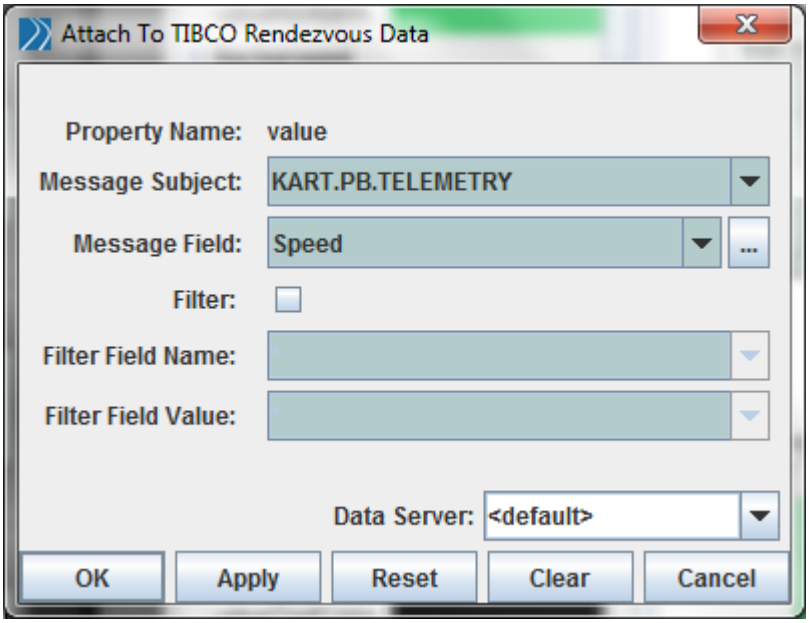


Figura 3.38 – Recebimento de dados do medidor e do gráfico de velocidade

A escala e o gráfico de temperatura foram configurados para receber os dados do campo "Temperature" da mensagem do Rendezvous. Na Figura 3.39 é mostrada a configuração da escala e do gráfico. Também foi configurado para a escala exibir temperaturas até 150ºC, para que a escala seja verde de 0 a 110ºC, amarela entre 110 e 130ºC e vermelha acima de 130ºC.

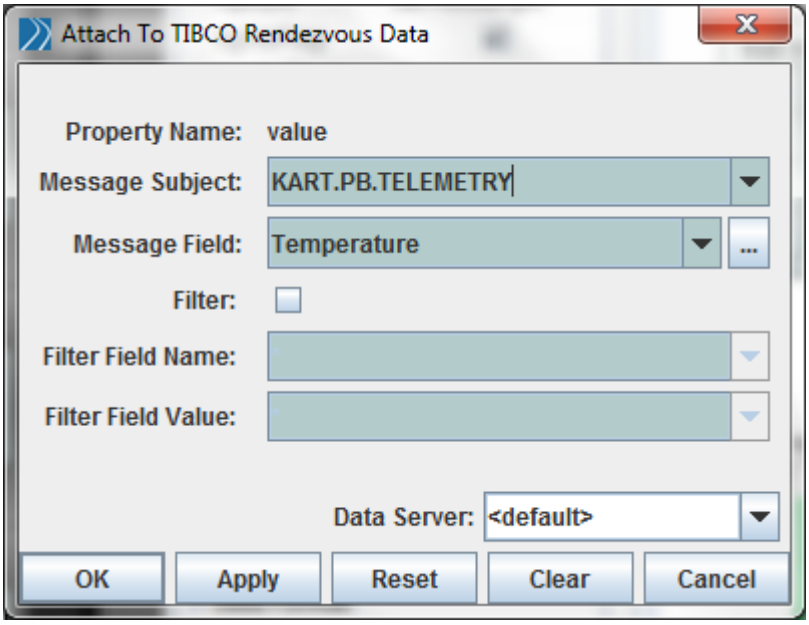


Figura 3.39 – Recebimento de dados da escala e do gráfico de temperatura

A escala e o gráfico de movimentação do volante foram configurados para receber os dados do campo "SteeringWheel" da mensagem do Rendezvous. Na Figura 3.40 é mostrada a configuração da escala e do gráfico. Também foi configurado para a escala exibir valores entre -500 e 500.

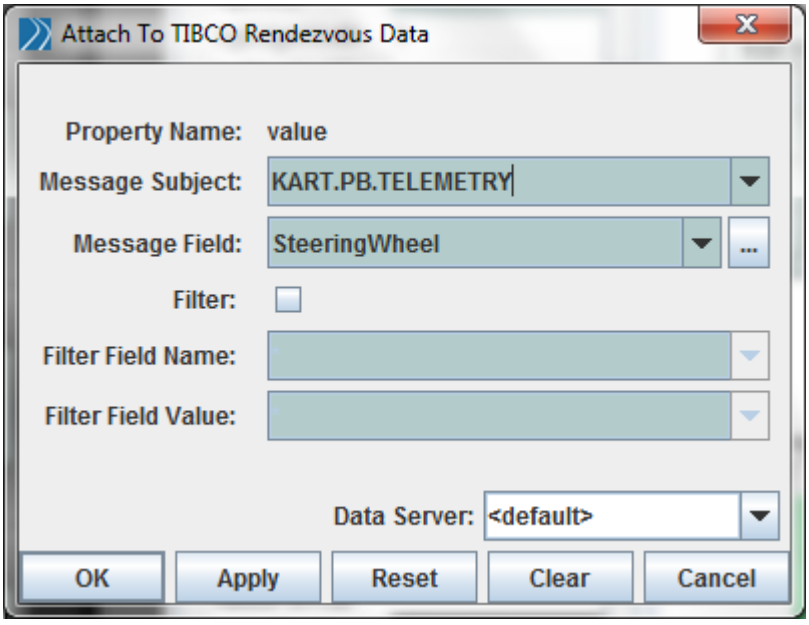


Figura 3.40 – Recebimento de dados da escala e do gráfico de movimentação do volante

O medidor e o gráfico do acelerador foram configurados para receber os dados do campo “Acelerator” da mensagem do Rendezvous. Na Figura 3.41 é mostrada a configuração da escala e do gráfico. Também foi configurado para a escala exibir valores de 0 a 100, para que a escala seja verde de 0 a 80, amarela entre 80 e 90 e vermelha acima de 90.

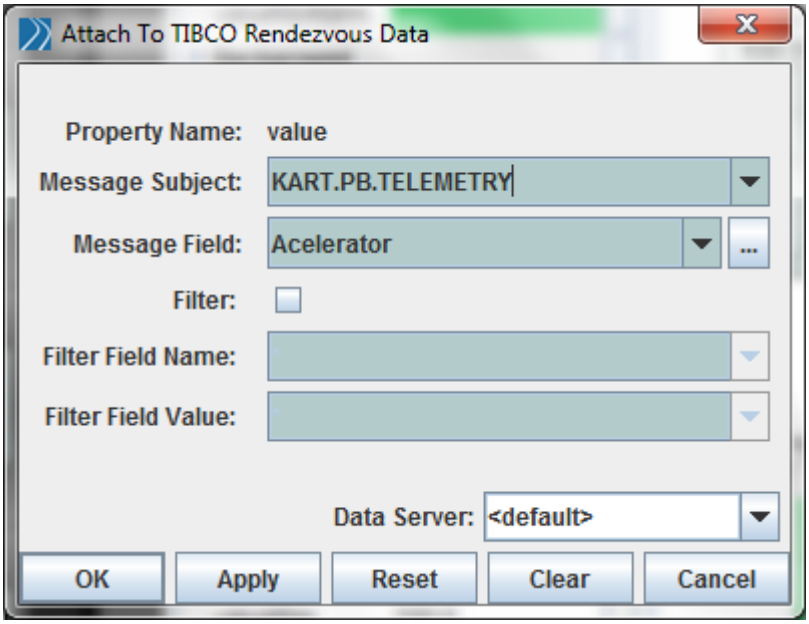


Figura 3.41 – Recebimento de dados da escala e do gráfico do acelerador

O medidor e o gráfico do freio foram configurados para receber os dados do campo “Brake” da mensagem do Rendezvous. Na Figura 3.42 é mostrada a configuração da escala e do gráfico. Também foi configurado para a escala exibir valores de 0 a 100, para que a escala seja verde de 0 a 80, amarela entre 80 e 90 e vermelha acima de 90.

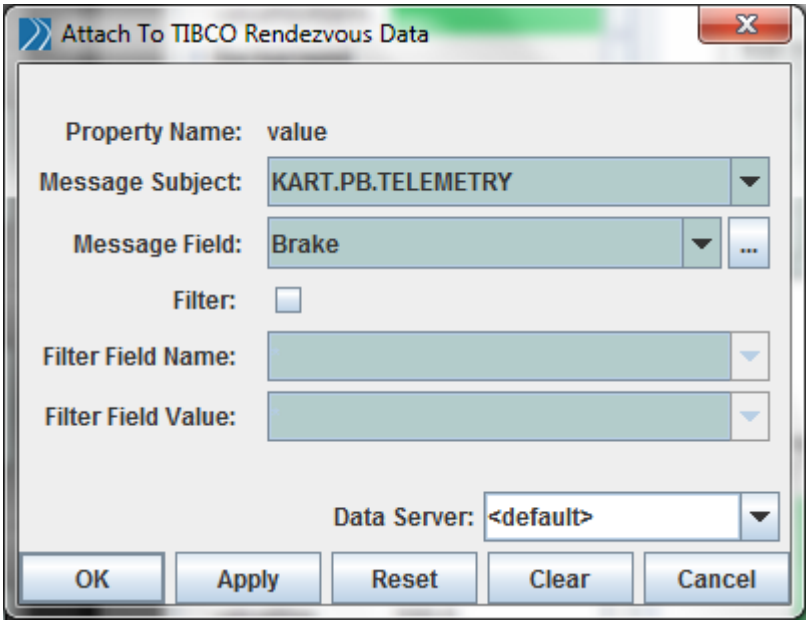


Figura 3.42 – Recebimento de dados da escala e o gráfico do freio

Foi criado um *label* na parte superior direita onde é informado o nome da corrida e um *label* na parte superior esquerda onde é informado se a telemetria está iniciada ou finalizada.

Foram criados três botões no painel. O botão “Start” envia uma mensagem para o TIBCO BusinessWorks informando que a telemetria está iniciada. O botão “Stop” envia uma mensagem para o TIBCO BusinessWorks informando que a telemetria está terminada. O TIBCO BusinessWorks responde essa mensagem com uma mensagem que faz com que o botão “Voltar a Tela Inicial” seja habilitado. Esse último botão, chama o painel “InitialScreen.rtv”.

O painel “LiveInfoNew” finalizado é mostrado na Figura 3.43.

Figura 3.43 – Painel de análise online

O painel “InitialScreen” consiste em dois botões. O botão “Efetuar Cadastros” abre o navegador na página inicial dos cadastros. A URL configurada foi “http://localhost:8068/telemetry/Registration/home.html”. O botão “Ir para Telemetria” chama o painel “LiveInfoNew”.

O painel “InitialScreen” é mostrado na Figura 3.44.

Figura 3.44 – Tela inicial do programa de telemetria

3.2.5 GRÁFICOS OFFLINE

Para desenvolvimento do sistema de análise online, foi utilizado o programa TIBCO Spotfire. O Spotfire utiliza o banco de dados para obter os dados para exibir nas análises.

Foram criadas colunas dentro do Spotfire relacionadas à colunas do banco de dados. Depois, foram criados os *joins* entre as tabelas. As colunas e os *joins* são ilustrados na Figura 3.45. Com as colunas e os *joins* criados, foi criado um "Information Link", que é uma das forma que o TIBCO Spotfire utiliza para fazer consultas no banco de dados. O "Information Link" criado é mostrado na Figura 3.46 e o SQL que foi gerado por ele é mostrado no APÊNDICE D.

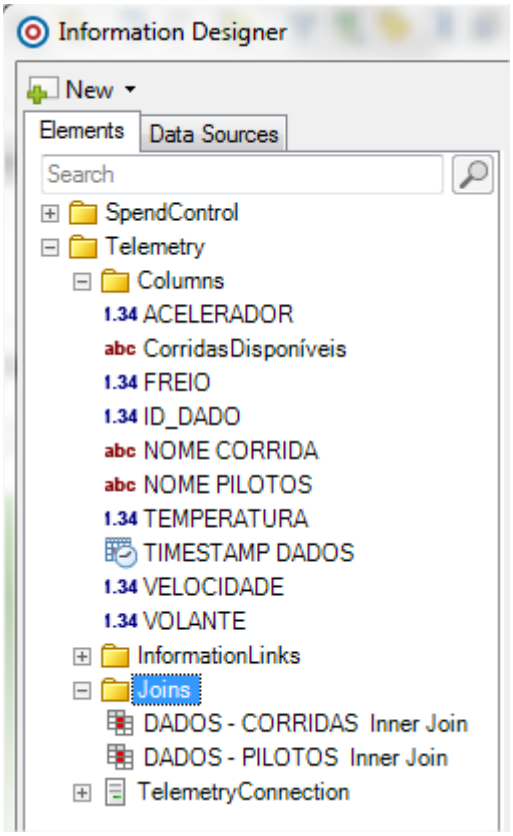


Figura 3.45 – Colunas e Joins criados no TIBCO Spotfire

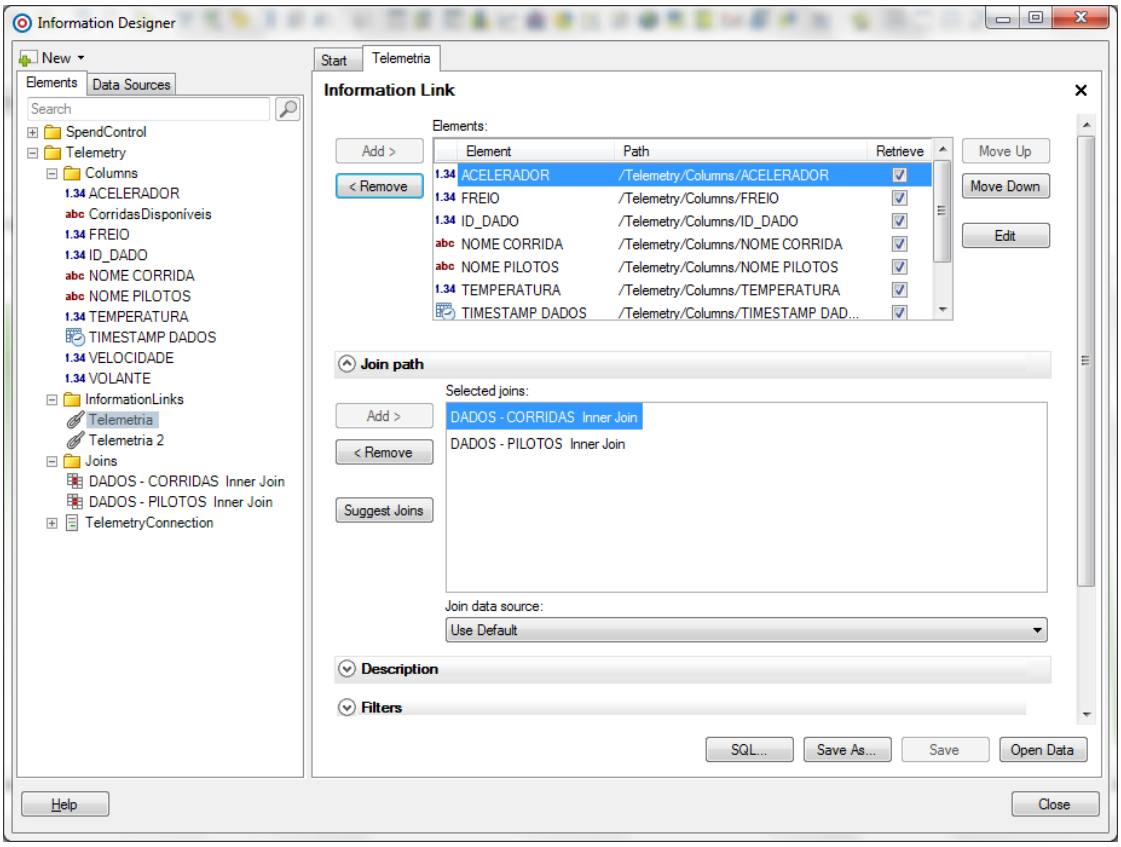


Figura 3.46 – Information Link criado no TIBCO Spotfire

Depois de criadas e feitas as configurações necessárias, foram desenvolvidas cinco páginas: Inicial, Volante, Acelerador e Freio, Velocidade e Temperatura.

A página Inicial possui apenas quatro botões: “Visualizar informações de Volante”, “Visualizar informações de Acelerador e Freio”, “Visualizar informações de Velocidade”, “Visualizar informações de Temperatura”. Cada um deles chama uma das páginas de análise. A página Inicial é mostrada na Figura 3.47.

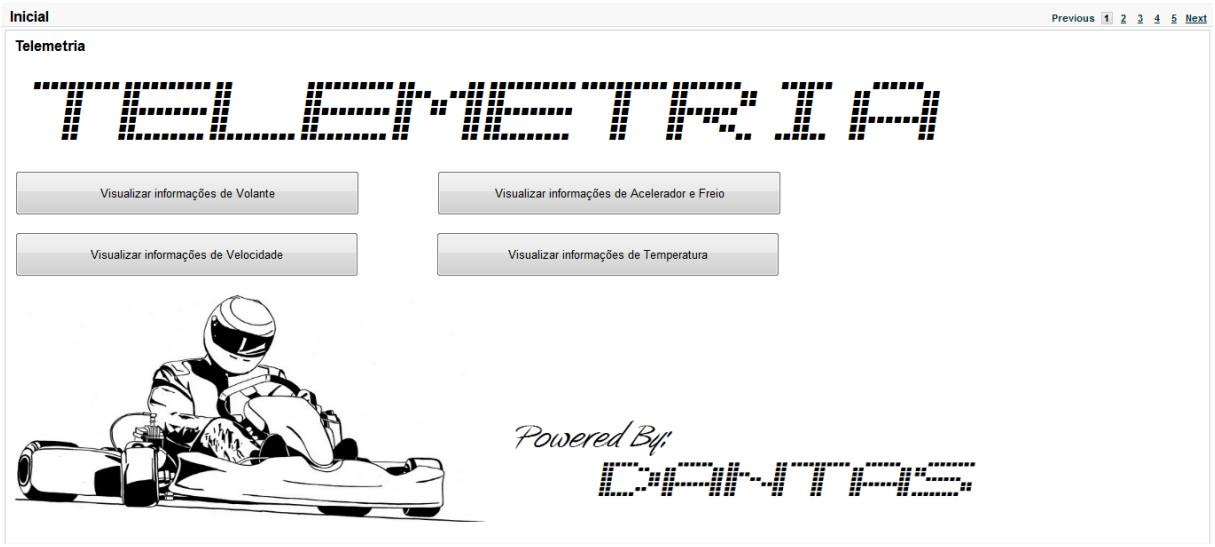


Figura 3.47 – Página Inicial do Spotfire

Na página Volante, foi criado um gráfico de linha, o eixo x foi associado à coluna “TIMESTAMP DADOS” e o eixo y foi associado à soma dos valores da coluna “VOLANTE”. Além do gráfico, foi criado um botão para retornar à página inicial e um filtro para escolher quais corridas devem ser mostradas no gráfico. A página é mostrada na Figura 3.48.

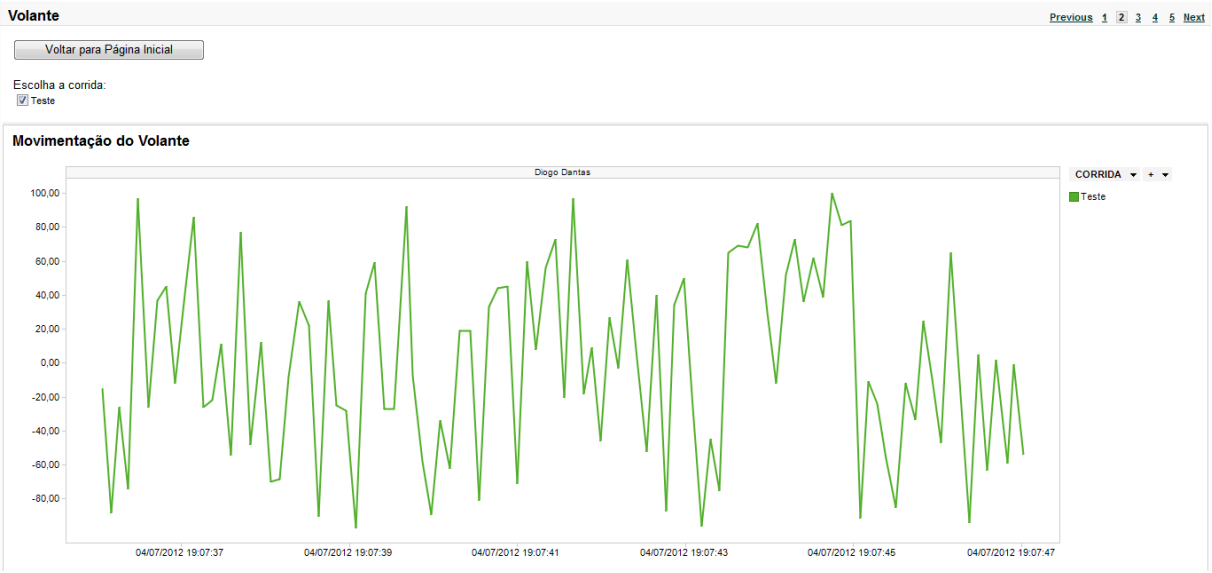


Figura 3.48 – Página de análise dos dados de movimentação do volante

Na página Acelerador e Freio, foram criados dois gráficos de linha, um para acelerador e outro para o freio. No de acelerador, o eixo x foi associado à coluna "TIMESTAMP DADOS" e o eixo y foi associado à soma dos valores da coluna "ACELERADOR". No de freio, o eixo foi associado à coluna "TIMESTAMP DADOS" e o eixo y foi associado à soma dos valores da coluna "FREIO". Além dos gráficos, foi criado um botão para retornar à página inicial e um filtro para escolher quais corridas devem ser mostradas no gráfico. A página é mostrada na Figura 3.49.

Figura 3.49 – Página de análise dos dados de posição dos pedais de acelerador e freio

Na página Velocidade, foi criado um gráfico de linha, o eixo x foi associado à coluna "TIMESTAMP DADOS" e o eixo y foi associado à soma dos valores da coluna "VELOCIDADE". Além do gráfico, foi criado um botão para retornar à página inicial e um filtro para escolher quais corridas devem ser mostradas no gráfico. A página é mostrada na Figura 3.50.

Figura 3.50 – Página de análise dos dados de velocidade do kart

Na página Temperatura, foi criado um gráfico de linha, o eixo x foi associado à coluna "TIMESTAMP DADOS" e o eixo y foi associado à soma dos valores da coluna "TEMPERATURA". Além do gráfico, foi criado um botão para retornar à página inicial e um filtro para escolher quais corridas devem ser mostradas no gráfico. A página é mostrada na Figura 3.51.

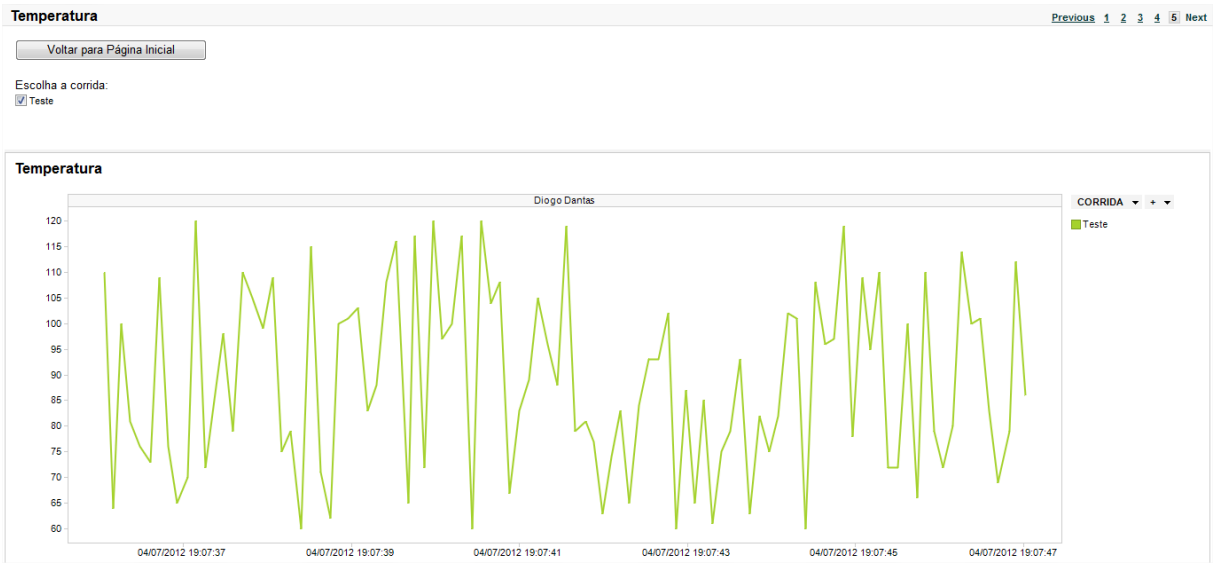


Figura 3.51 – Página de análise dos dados de temperatura do motor

4- RESULTADOS OBTIDOS

A análise dos resultados obtidos está dividida em cinco etapas. A primeira foi a realização dos testes do sistema de cadastro, que consiste nas páginas *web*, no *WebService* e nas tabelas do banco de dados. A segunda, o teste do sistema de integração e dos produtos de análise com dados de teste. A terceira, o teste da coleta de informações dos sensores e exibição na porta serial. A quarta, o teste do protótipo integrando com os softwares. A quinta, o protótipo acoplado no kart enviando os dados para o computador. O fluxo do projeto é mostrado no APÊNDICE G.

4.1 SISTEMA DE CADASTRO

Desenvolvido para facilitar o cadastro de karts, pilotos e corridas no banco de dados. Para testar o funcionamento do sistema, foram iniciados o processo do BW que recebe dados da *web* e o *WebService* e foram acessadas as páginas de cadastro.

Primeiro é acessada a página de cadastro de piloto onde são informados os dados, conforme Figura 4.1. Os dados são recebidos pelo processo do BW e é chamado o *WebService*, conforme Figura 4.2. O *WebService* executa o processo de inserção no banco de dados, conforme Figura 4.3. O banco de dados então é aberto e verifica-se se os dados foram inseridos com sucesso, conforme ilustrado na Figura 4.4.

Figura 4.1 – Dados informados na página de cadastro de piloto

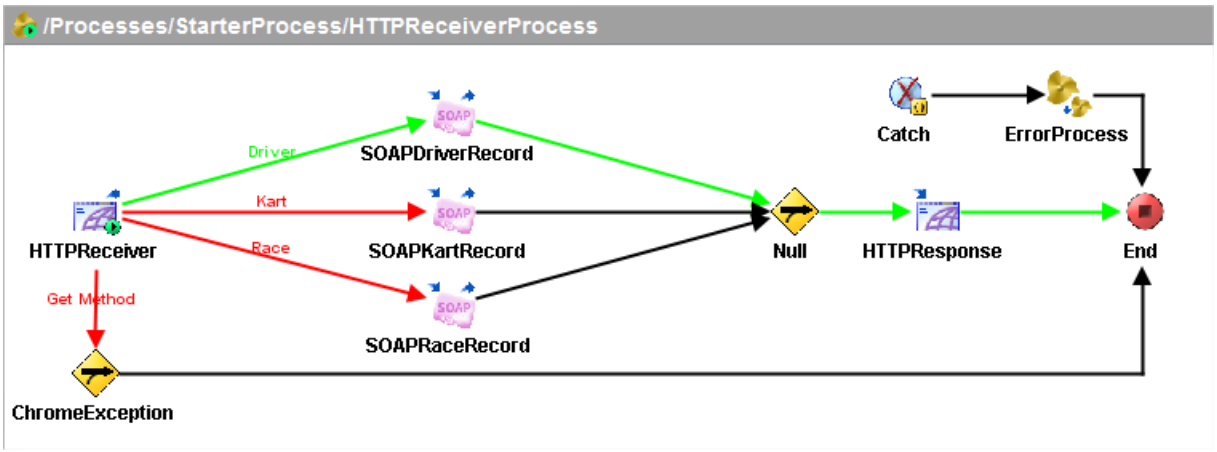


Figura 4.2 – Processo recebendo os dados de piloto e invocando o WebService

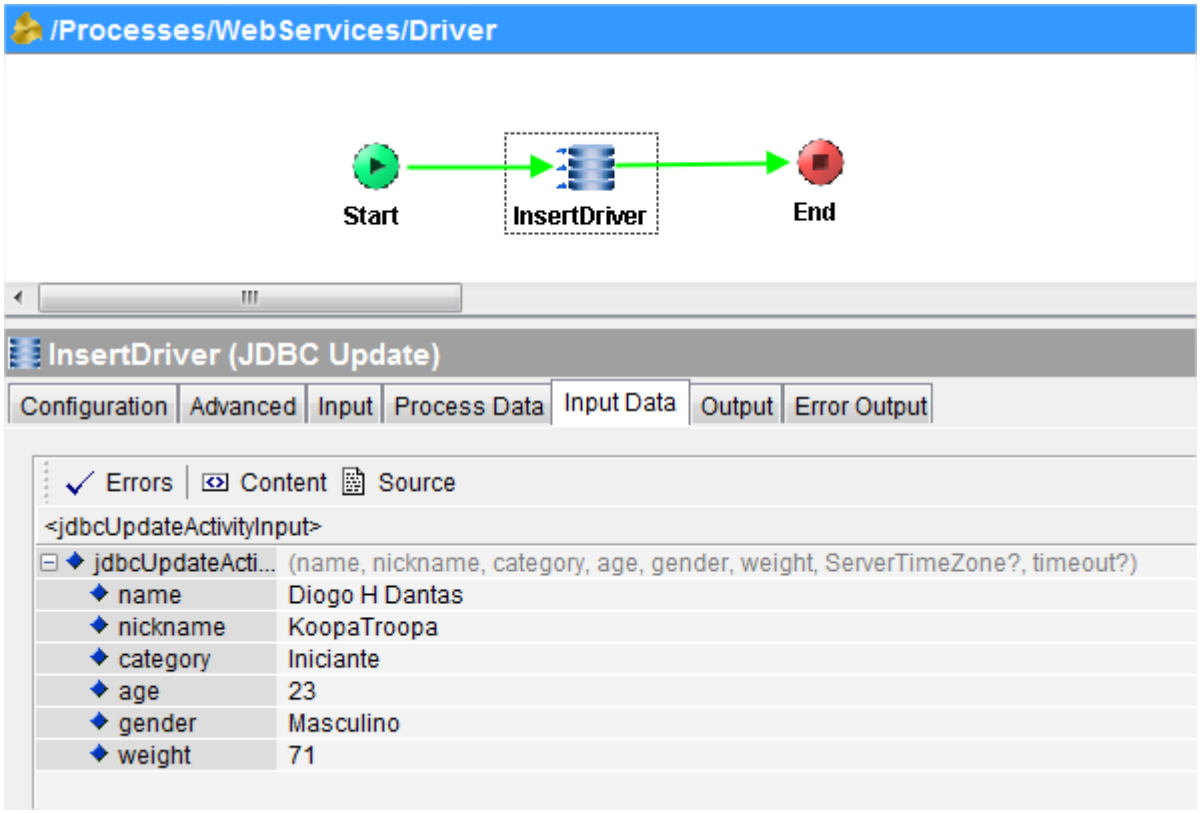
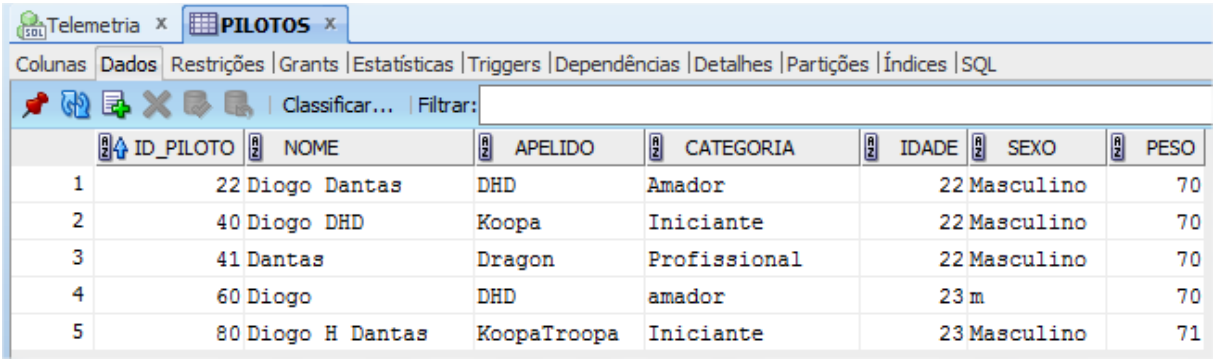


Figura 4.3 – Dados do piloto sendo inseridos no banco de dados



	ID_PILOTO	NOME	APELIDO	CATEGORIA	IDADE	SEXO	PESO
1	22	Diogo Dantas	DHD	Amador	22	Masculino	70
2	40	Diogo DHD	Koopa	Iniciante	22	Masculino	70
3	41	Dantas	Dragon	Profissional	22	Masculino	70
4	60	Diogo	DHD	amador	23	m	70
5	80	Diogo H Dantas	KoopaTroopa	Iniciante	23	Masculino	71

Figura 4.4 – Dados do piloto presentes no banco de dados

Depois é acessada a página de cadastro de karts onde são informados os dados, conforme Figura 4.5. Os dados são recebidos pelo processo do BW e é chamado o *WebService*, conforme Figura 4.6. O *WebService* executa o processo de inserção no banco de dados, conforme Figura 4.7. O banco de dados então é aberto e verifica-se se os dados foram inseridos com sucesso, ilustrado na Figura 4.8.

Figura 4.5 – Dados informados na página de cadastro de karts

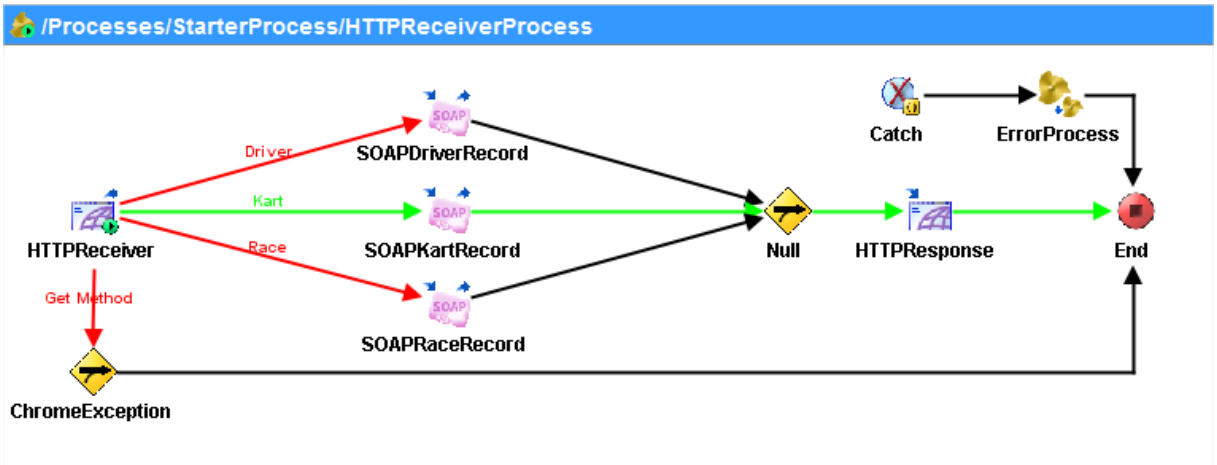


Figura 4.6 - Processo recebendo os dados de kart e invocando o WebService

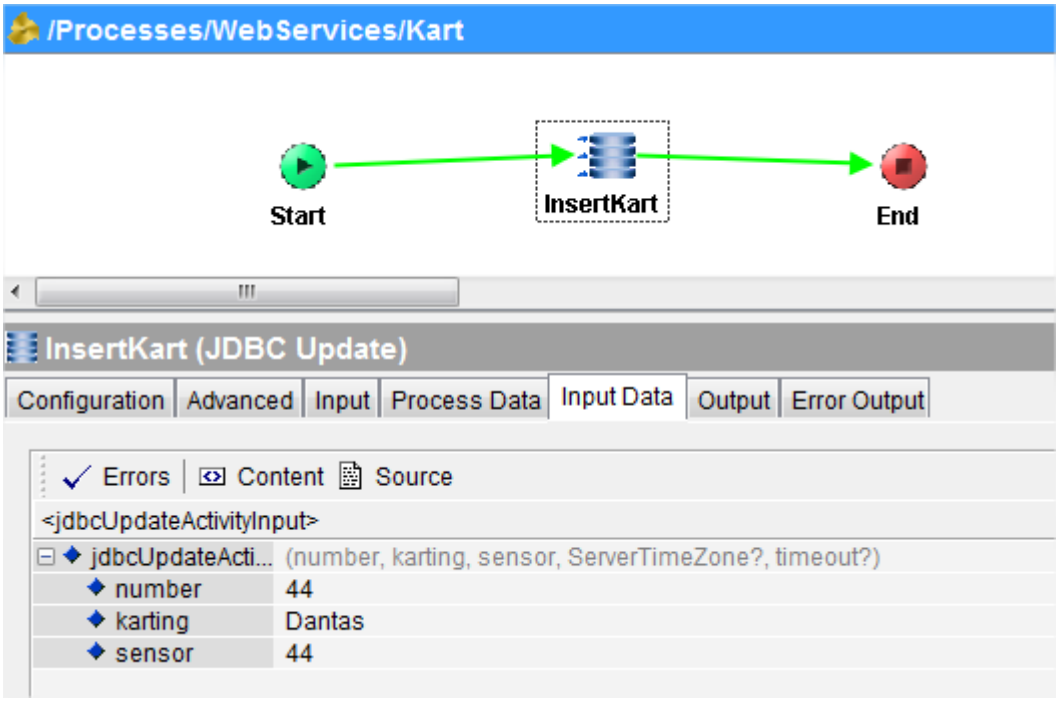
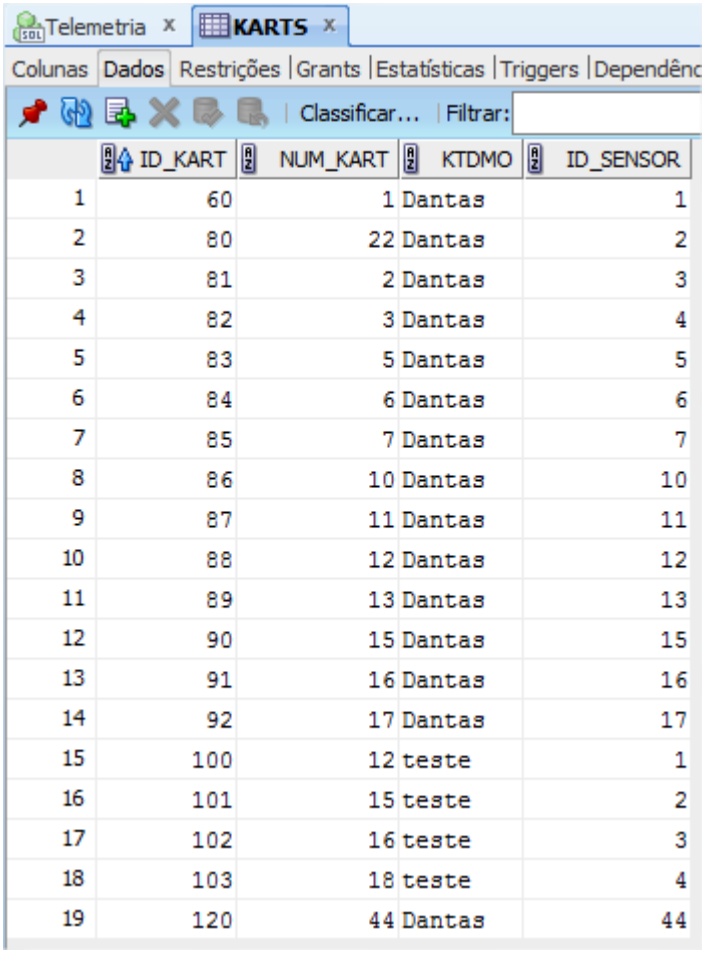


Figura 4.7 - Dados do kart sendo inseridos no banco de dados



	ID_KART	NUM_KART	KTDMO	ID_SENSOR
1	60	1	Dantas	1
2	80	22	Dantas	2
3	81	2	Dantas	3
4	82	3	Dantas	4
5	83	5	Dantas	5
6	84	6	Dantas	6
7	85	7	Dantas	7
8	86	10	Dantas	10
9	87	11	Dantas	11
10	88	12	Dantas	12
11	89	13	Dantas	13
12	90	15	Dantas	15
13	91	16	Dantas	16
14	92	17	Dantas	17
15	100	12	teste	1
16	101	15	teste	2
17	102	16	teste	3
18	103	18	teste	4
19	120	44	Dantas	44

Figura 4.8 - Dados do kart presentes no banco de dados

Por último é acessada a página de cadastro de corridas onde são informados os dados, conforme Figura 4.9. Os dados são recebidos pelo processo do BW e é chamado o *WebService*, conforme Figura 4.10. O *WebService* executa o processo de inserção no banco de dados, conforme Figura 4.11. O banco de dados então é aberto e verifica-se os dados inseridos com sucesso (uma linha para cada piloto), ilustrado na Figura 4.12.

Figura 4.9 - Dados informados na página de cadastro de corrida

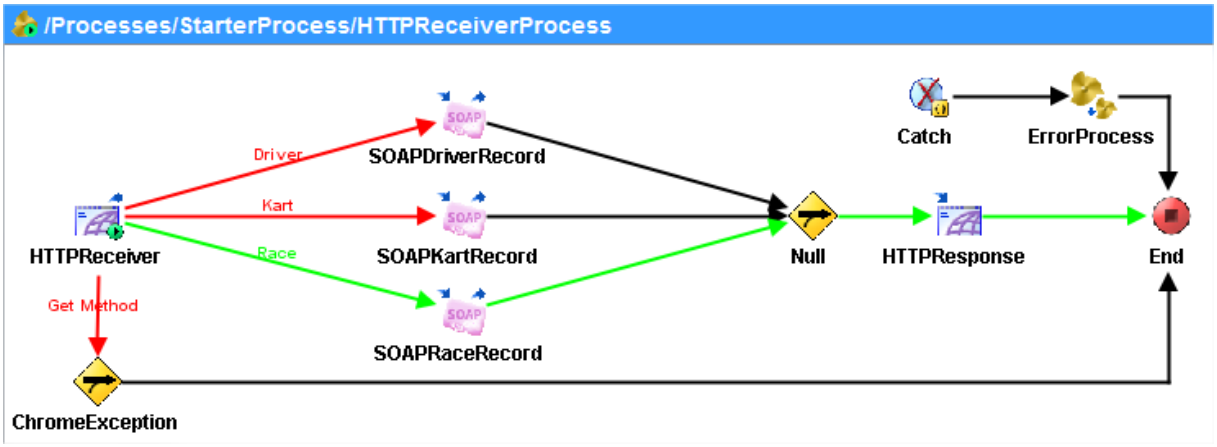


Figura 4.10 - Processo recebendo os dados de corrida e invocando o WebService

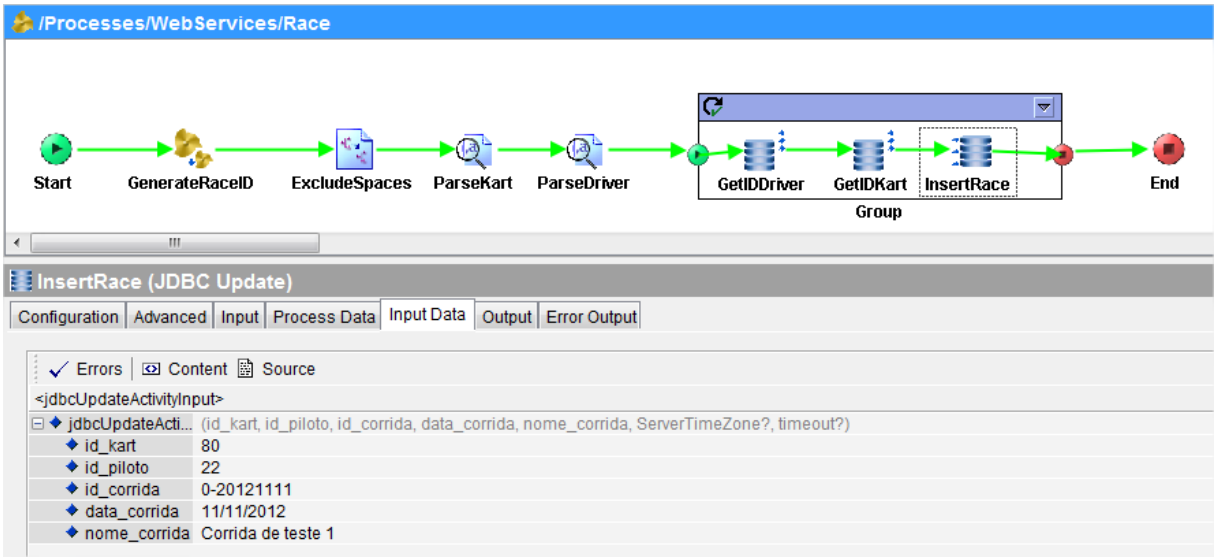
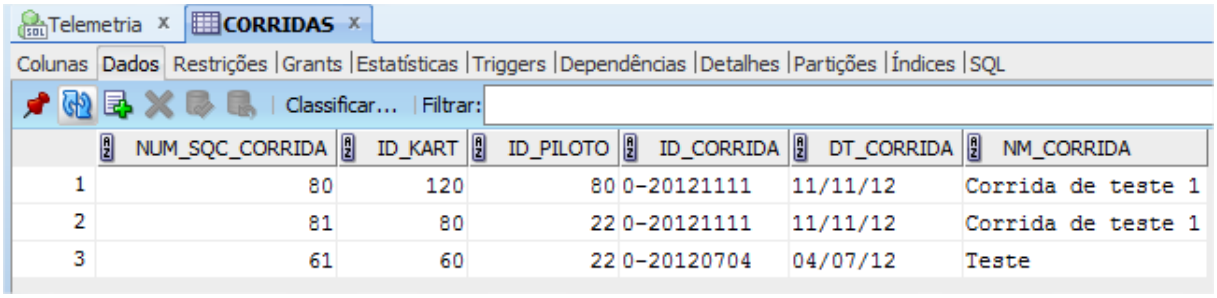


Figura 4.11 - Dados da corrida sendo inseridos no banco de dados



	NUM_SQC_CORRIDA	ID_KART	ID_PILOTO	ID_CORRIDA	DT_CORRIDA	NM_CORRIDA
1	80	120	80	0-20121111	11/11/12	Corrida de teste 1
2	81	80	22	0-20121111	11/11/12	Corrida de teste 1
3	61	60	22	0-20120704	04/07/12	Teste

Figura 4.12 - Dados da corrida presentes no banco de dados

4.2 INTEGRAÇÃO E PAINÉIS DE ANÁLISE

Os testes da integração e dos painéis foram realizados com dados gerados que simulam os dados enviados pelo protótipo. Esses dados gerados também são enviados por TCP. O processo de geração de massa de teste é mostrado na Figura 4.13.

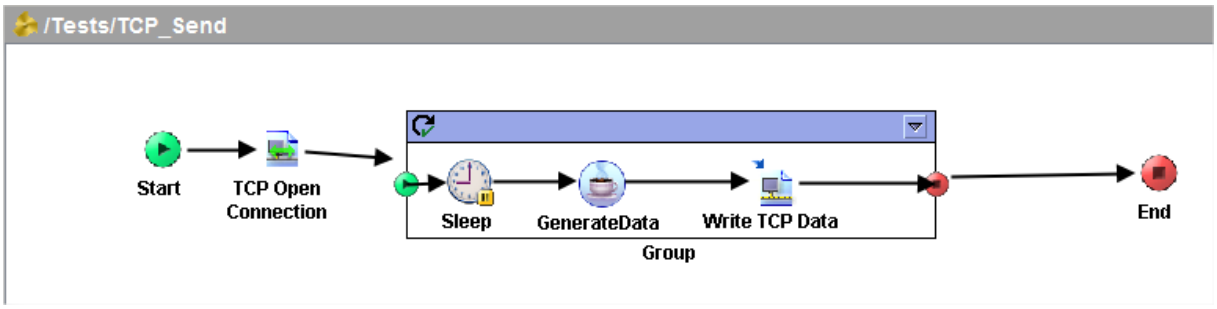


Figura 4.13 – Processo de geração de massa de teste

Os testes são iniciados acessando o painel “InitialScreen” clicando no botão “Ir para Telemetria”, ilustrado na Figura 4.14.

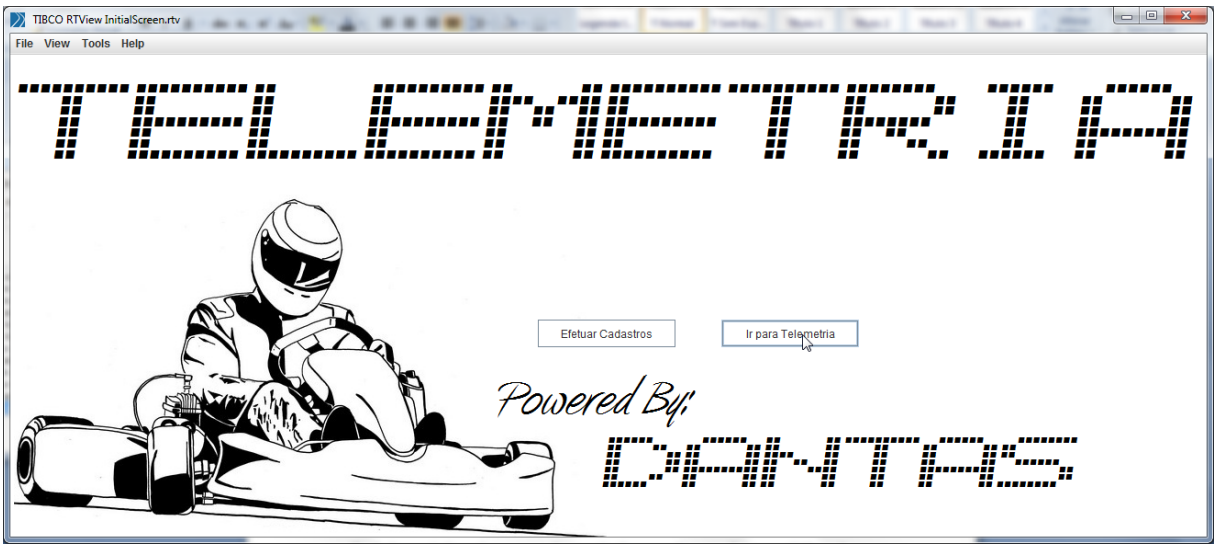


Figura 4.14 – Iniciando a telemetria

Quando o botão é acionado, são executadas duas ações. Primeiro, é enviada uma mensagem para o TIBCO BusinessWorks informando o início da telemetria e o BW envia uma resposta com o nome da corrida, conforme Figura 4.15. Segundo, é chamado o painel “LiveInfoNew” onde os dados são exibidos, conforme Figura 4.16.

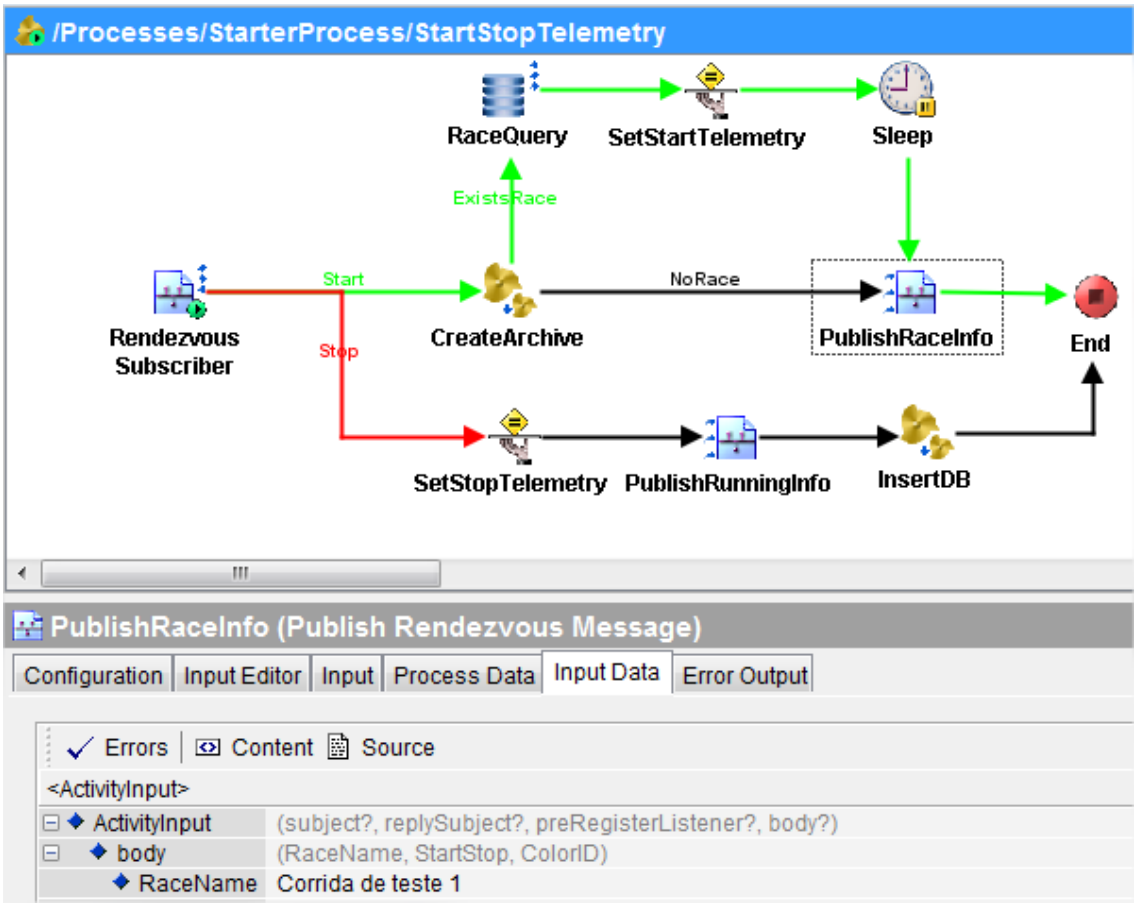


Figura 4.15 – Recebimento da informação do início da telemetria e resposta com o nome da corrida

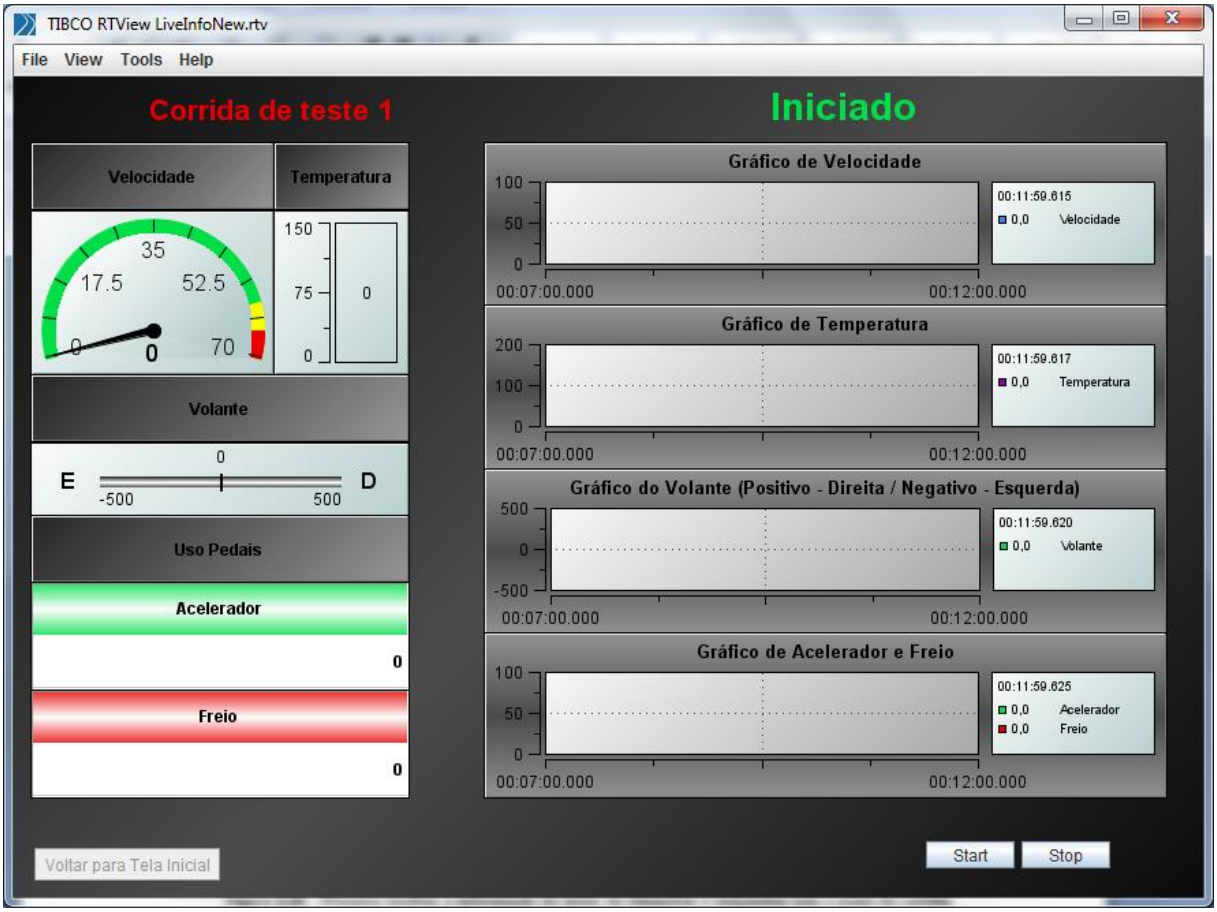


Figura 4.16 – Painel de gráficos chamado com as informações da corrida em andamento

Com o painel iniciado, é executado o processo de geração de massa de teste. Os dados são recebidos pelo processo TCP_Receive, tratados, enviados para o painel e gravados em um arquivo texto, conforme ilustrado na Figura 4.17. Esses dados são recebidos e exibidos no painel "LiveInfoNew", conforme Figura 4.18. O arquivo texto com os dados é ilustrado na Figura 4.19.

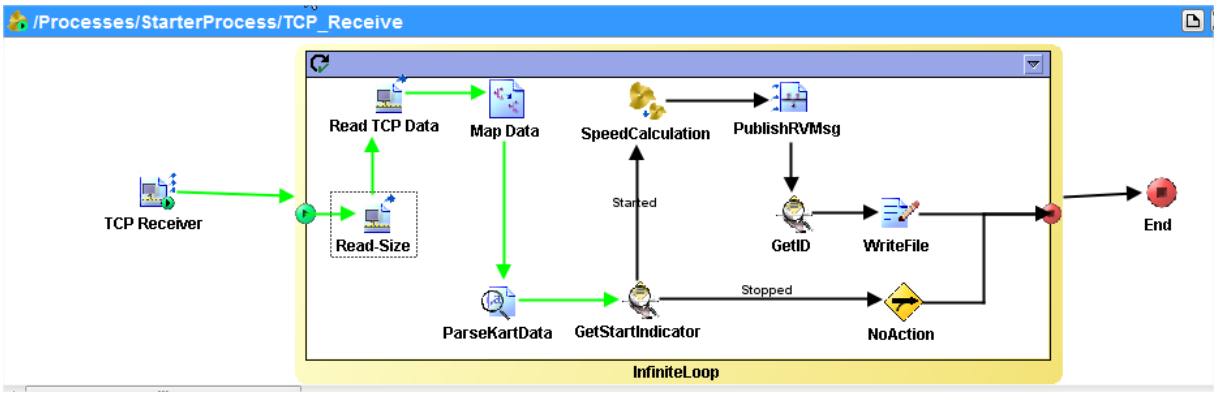


Figura 4.17 – Dados recebidos e tratados no processo de integração

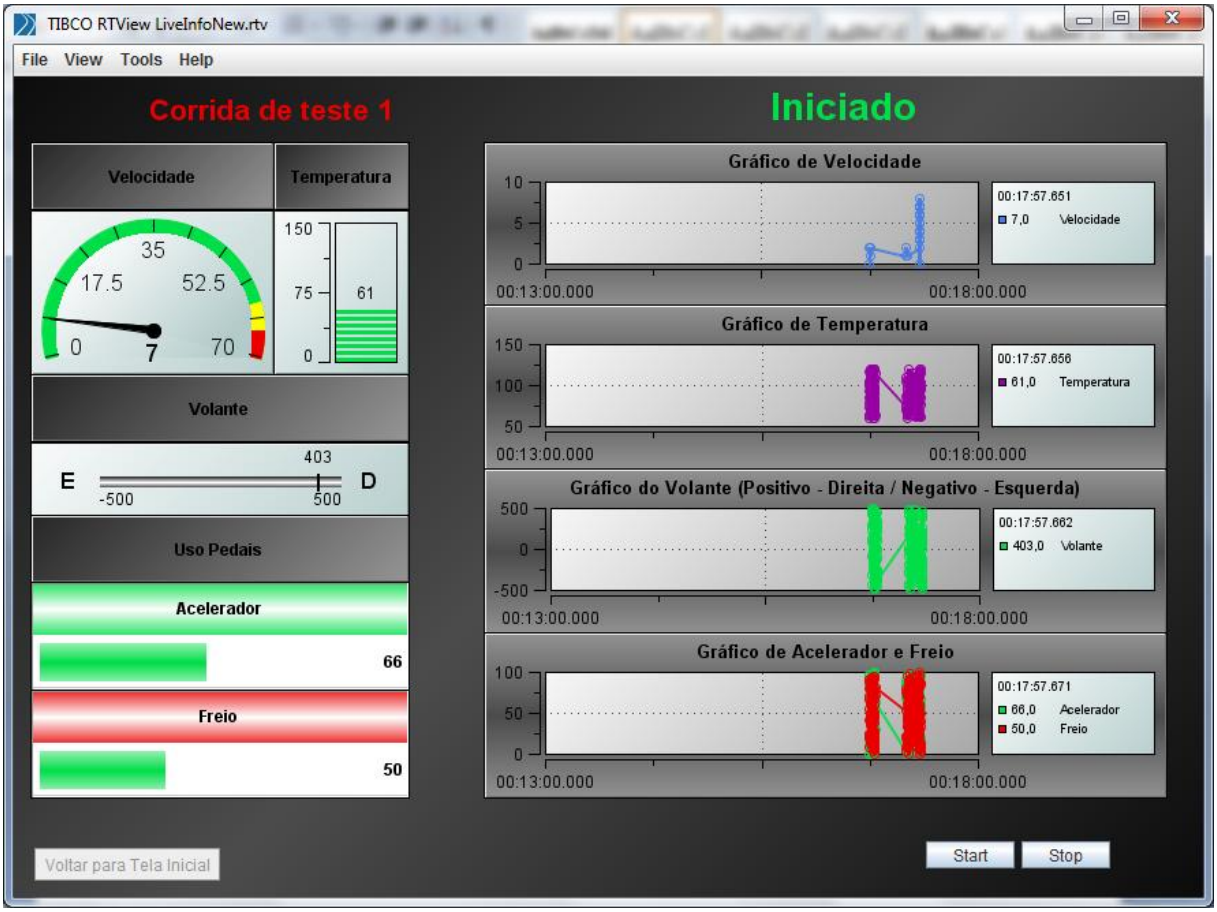


Figura 4.18 – Dados exibidos no painel de análise online

race-0-20121111.txt

```
1  IDSensor;Volante;Temperatura;Velocidade;Acelerador;Freio;Timestamp;
2  44;179;116;0;8;73;12/11/2012 00:37:28.826;
3  44;419;114;0;9;47;12/11/2012 00:37:28.867;
4  44;-210;116;0;93;7;12/11/2012 00:37:28.905;
5  44;-398;92;1;51;62;12/11/2012 00:37:28.951;
6  44;-150;95;0;79;47;12/11/2012 00:37:28.989;
7  44;-423;73;1;83;21;12/11/2012 00:37:29.034;
8  44;-457;60;0;59;40;12/11/2012 00:37:29.074;
9  44;370;87;1;59;19;12/11/2012 00:37:29.116;
10 44;169;82;2;24;63;12/11/2012 00:37:29.163;
11 44;-163;114;0;58;0;12/11/2012 00:37:29.200;
12 44;250;98;1;10;6;12/11/2012 00:37:29.246;
13 44;-279;113;2;96;19;12/11/2012 00:37:29.285;
14 44;331;112;2;70;16;12/11/2012 00:37:29.328;
15 44;-412;114;2;7;63;12/11/2012 00:37:29.373;
```

Figura 4.19 – Arquivo texto com os dados gravados

Após o envio dos dados, clica-se no botão “Stop”. Com isso, é enviada uma mensagem para o BW informando o término da telemetria. O BW recebe a mensagem e responde mudando o estado da telemetria para “Parado” e habilitando o botão “Voltar para a Tela Inicial”, depois lê o arquivo e insere os dados no banco de dados. O processo de recebimento de mensagem do BW é ilustrado na Figura 4.20, o painel atualizado é mostrado na Figura 4.21, o processo de inserção no banco é mostrado na Figura 4.22 e o banco de dados atualizado é mostrado na Figura 4.23.

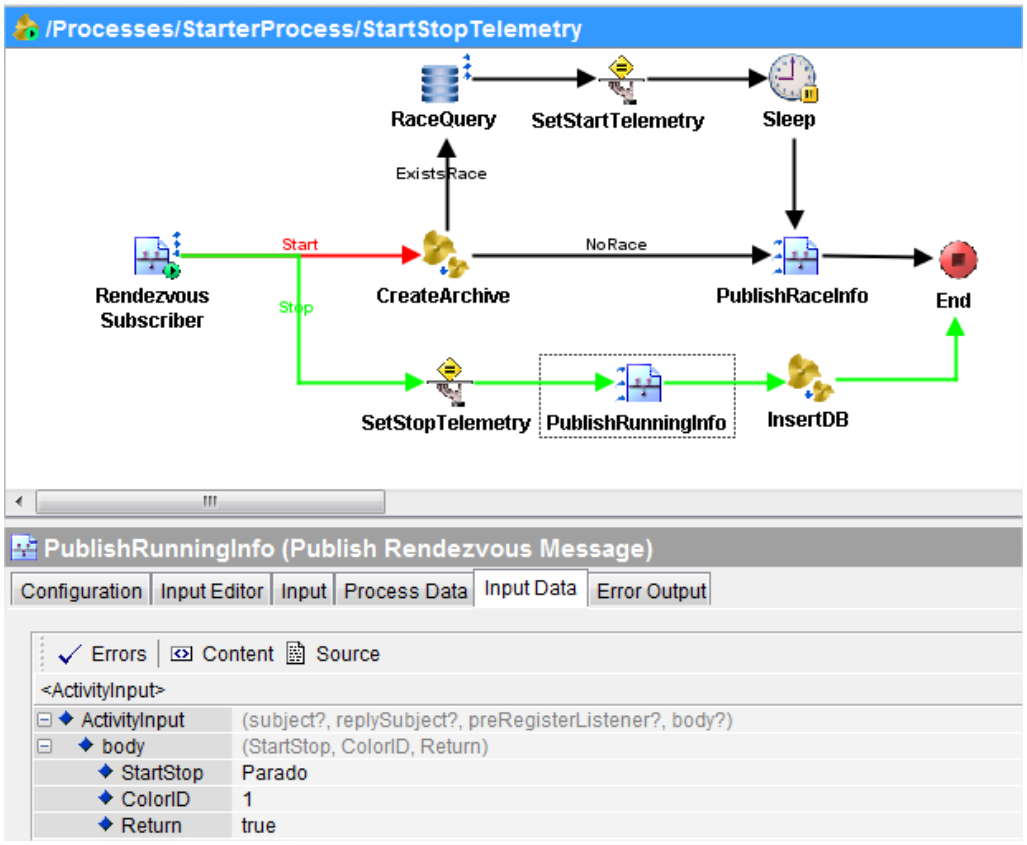


Figura 4.20 – Recebimento da informação do término da telemetria, resposta e gravação no banco

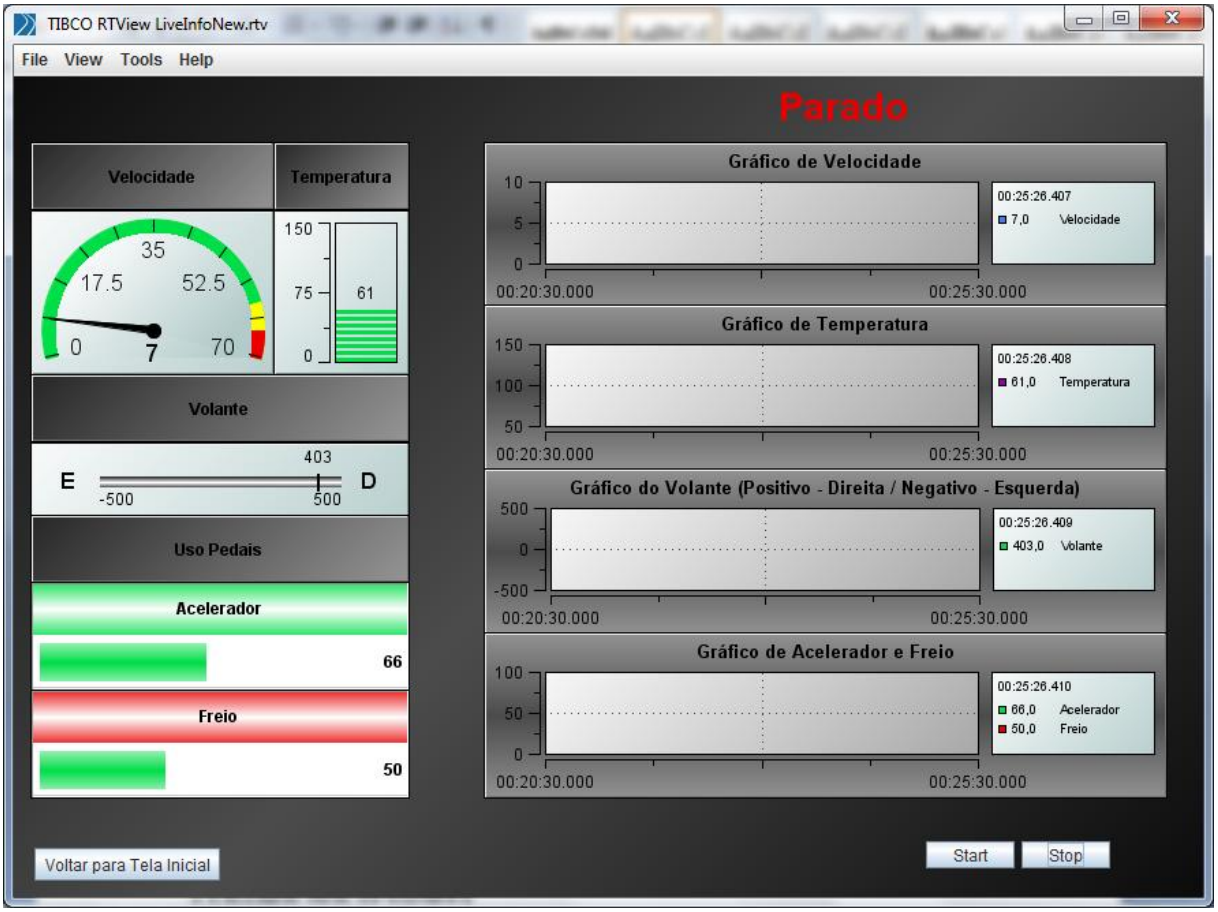


Figura 4.21 – Painel atualizado com o término da telemetria

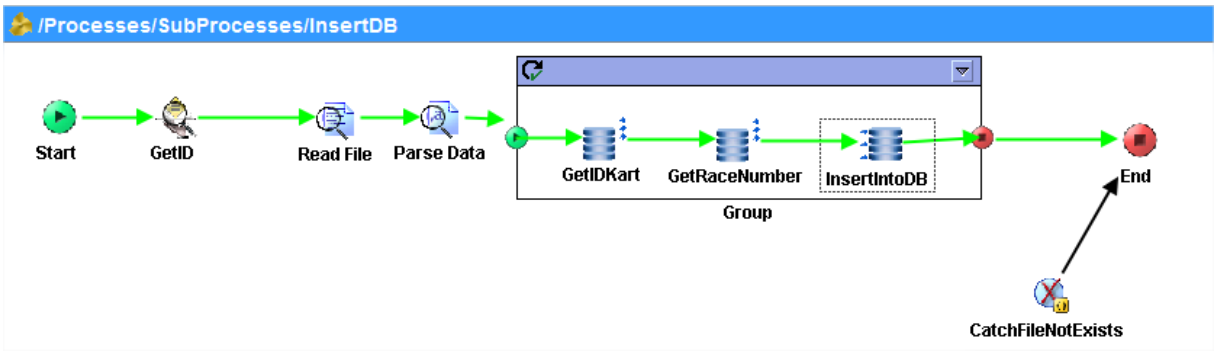


Figura 4.22 – Processo de inserção de dados no banco

Telemetria × DADOS ×

Colunas | Dados | Restrições | Grants | Estatísticas | Triggers | Dependências | Detalhes | Partições | Índices | SQL

Classificar... Filtrar: Ações...

	ID_DADO	NUM_SQC_CORRIDA	ID_PILOTO	VOLANTE	TEMPERATURA	VELOCIDADE	ACELERADOR	FREIO	TS_DADO
1	621	80	80	469	76	1	1	36	12/11/12 00:37:33,011000000
2	620	80	80	-418	65	0	58	37	12/11/12 00:37:32,972000000
3	619	80	80	426	96	2	14	1	12/11/12 00:37:32,915000000
4	618	80	80	-107	67	1	18	65	12/11/12 00:37:32,873000000
5	617	80	80	-83	85	0	67	46	12/11/12 00:37:32,829000000
6	616	80	80	-488	114	0	15	43	12/11/12 00:37:32,788000000
7	615	80	80	-164	108	0	37	5	12/11/12 00:37:32,744000000
8	614	80	80	362	92	0	76	28	12/11/12 00:37:32,701000000
9	613	80	80	380	107	2	48	41	12/11/12 00:37:32,671000000
10	612	80	80	-473	114	1	81	78	12/11/12 00:37:32,620000000
11	611	80	80	259	101	0	72	59	12/11/12 00:37:32,577000000
12	610	80	80	-39	109	0	85	81	12/11/12 00:37:32,534000000
13	609	80	80	158	99	3	95	99	12/11/12 00:37:32,494000000
14	608	80	80	310	83	2	85	13	12/11/12 00:37:32,467000000
15	607	80	80	-197	90	2	21	96	12/11/12 00:37:32,406000000
16	606	80	80	452	79	2	6	73	12/11/12 00:37:32,366000000
17	605	80	80	330	72	2	44	44	12/11/12 00:37:32,324000000
18	604	80	80	292	117	2	19	35	12/11/12 00:37:32,282000000
19	603	80	80	-105	114	1	34	78	12/11/12 00:37:32,241000000
20	602	80	80	225	63	0	56	2	12/11/12 00:37:32,194000000

Figura 4.23 – Dados inseridos no banco

Com os dados salvos no banco, foi então testada a solução de análise offline.

É aberta a análise no TIBCO Spotfire e, na página inicial, clicando no botão "Visualizar informações de Volante", conforme Figura 4.24.

Figura 4.24 – Testes da análise offline

Com o acionamento do botão, é aberta a página “Volante”, onde são exibidas as informações da movimentação do volante em um gráfico de linhas, conforme Figura 4.25.

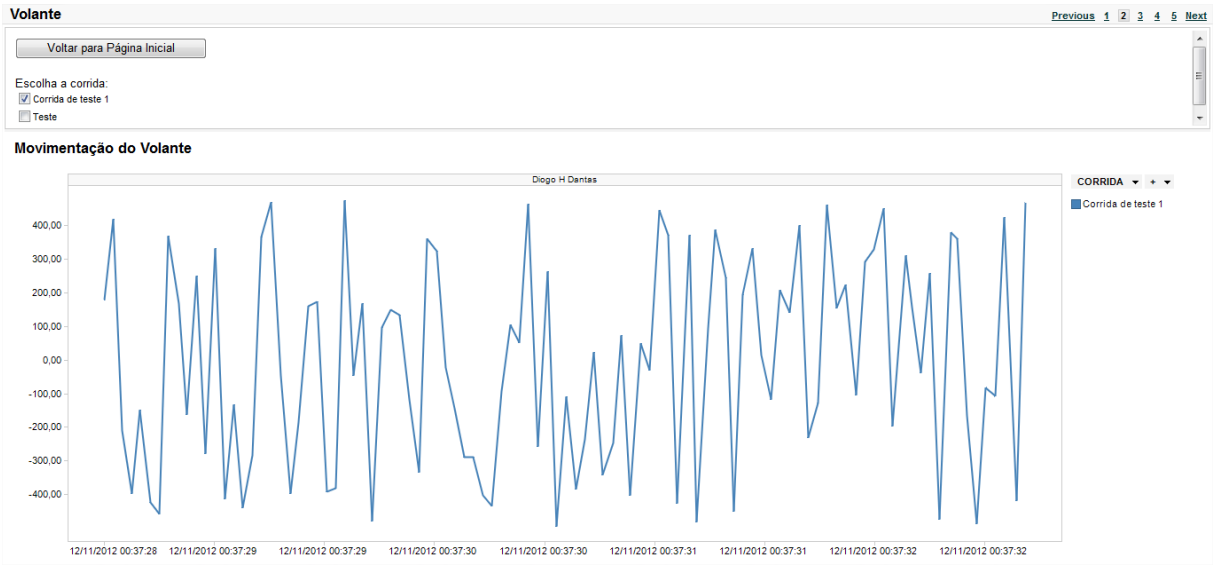


Figura 4.25 – Gráfico com informações da movimentação do volante

Depois se clica no botão “Voltar para Página Inicial” e, na página inicial, clica-se no botão “Visualizar informações de Acelerador e Freio”. Com isso, a análise é direcionada para a página “Acelerador e Freio”, onde são exibidas as informações da posição dos pedais de aceleração e frenagem em gráficos de linha, conforme a Figura 4.26.

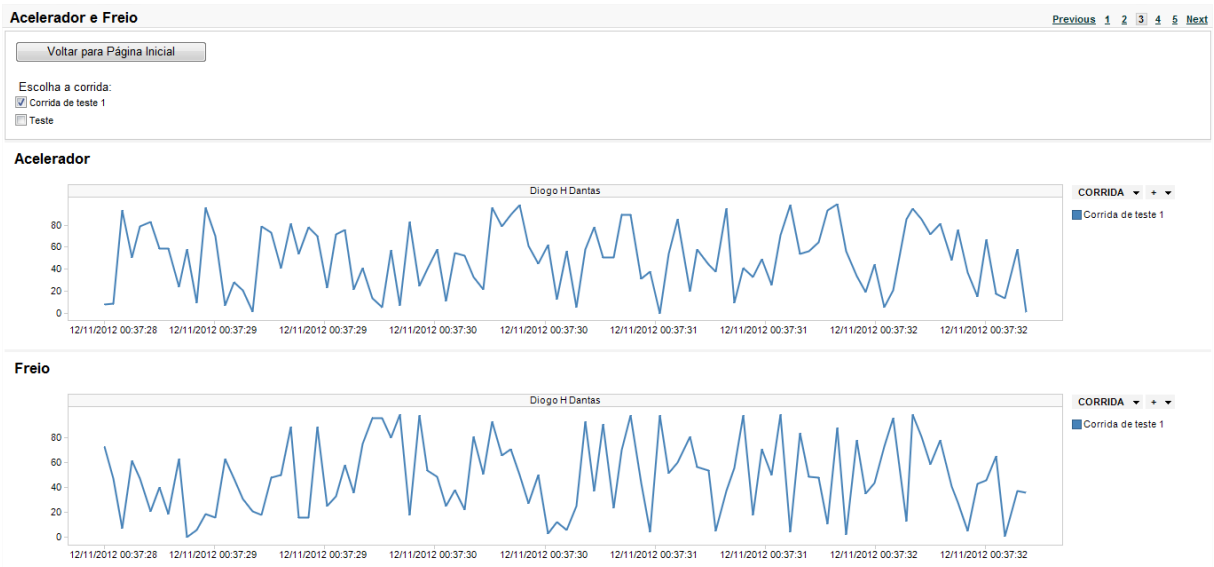


Figura 4.26 – Gráficos com informações da posição dos pedais de aceleração e frenagem

Então se clica no botão “Voltar para Página Inicial” e, na página inicial, clica-se no botão “Visualizar informações de Velocidade”. Com isso, a análise é direcionada para a página “Velocidade”, onde são exibidas as informações da velocidade do kart em gráficos de linha, conforme a Figura 4.27.

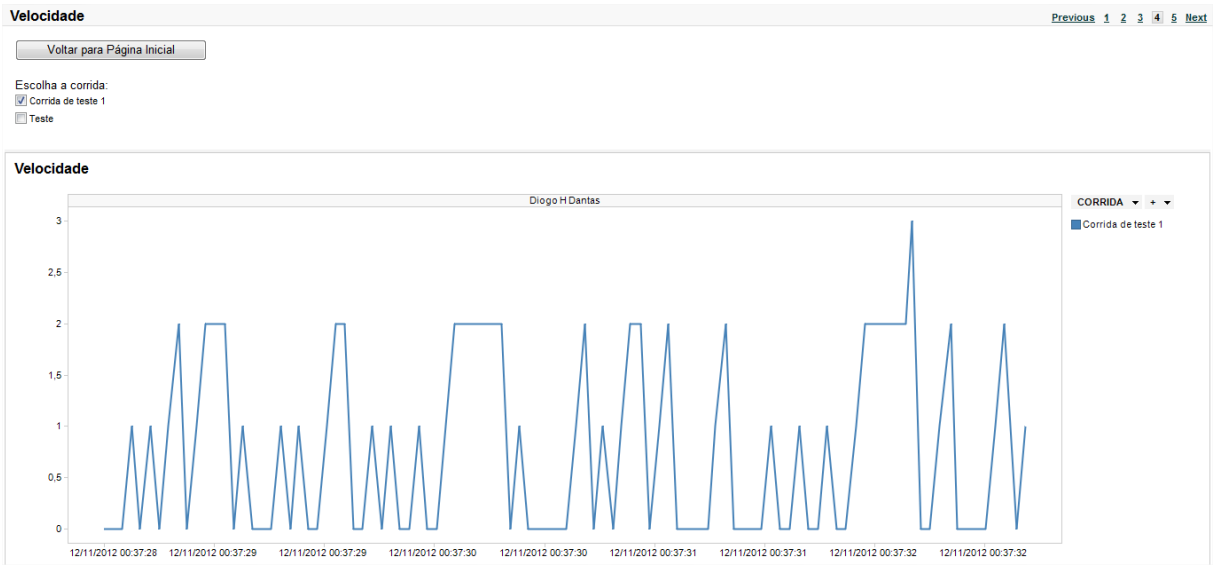


Figura 4.27 - Gráfico com informações da velocidade do kart

Por fim, clica-se no botão “Voltar para Página Inicial” e, na página inicial, clica-se no botão “Visualizar informações de Temperatura”. Com isso, a análise é direcionada para a página “Temperatura”, onde são exibidas as informações da temperatura do motor em gráficos de linha, conforme a Figura 4.28.

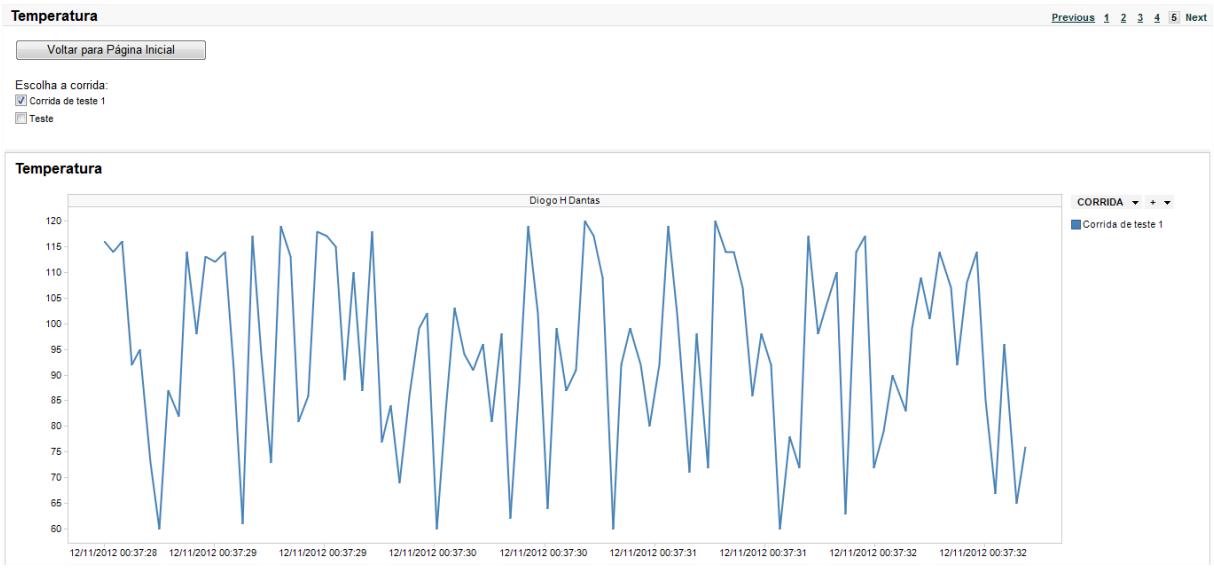


Figura 4.28 – Gráfico com informações da temperatura do motor

4.3 DADOS DOS SENSORES

Para verificar os dados enviados dos sensores para o microcontrolador, foi alterado o código do protótipo para enviar os dados por serial e o protótipo foi conectado no computador pela porta USB. O código é mostrado no APÊNDICDE E.

Os dados são verificados utilizando o Serial Monitor do programa do Arduino, conforme Figura 4.29.

```
ID Sensor
1
Volante
4
Acelerador
44
Freio
58
Temperatura
24
Velocidade
0
data:
;1;4;44;58;24;0;
info:
0016;1;4;44;58;24;0;
temp:
0016;1;4;44;58;24;0;
msg:
0016;1;4;44;58;24;0;
```

Figura 4.29 – Dados dos sensores exibidos via porta serial

4.4 INTEGRAÇÃO DO PROTÓTIPO COM O COMPUTADOR

Para verificar a total funcionalidade do protótipo antes de instalá-lo no kart, é iniciado o servidor TCP e o protótipo. A conexão com o servidor é estabelecida e os dados são enviados, conforme é mostrado na Figura 4.30. Os dados coletados pelos sensores e enviados pelo protótipo são recebidos, tratados no servidor TCP, e exibidos no gráfico online, conforme é mostrado na Figura 4.31.

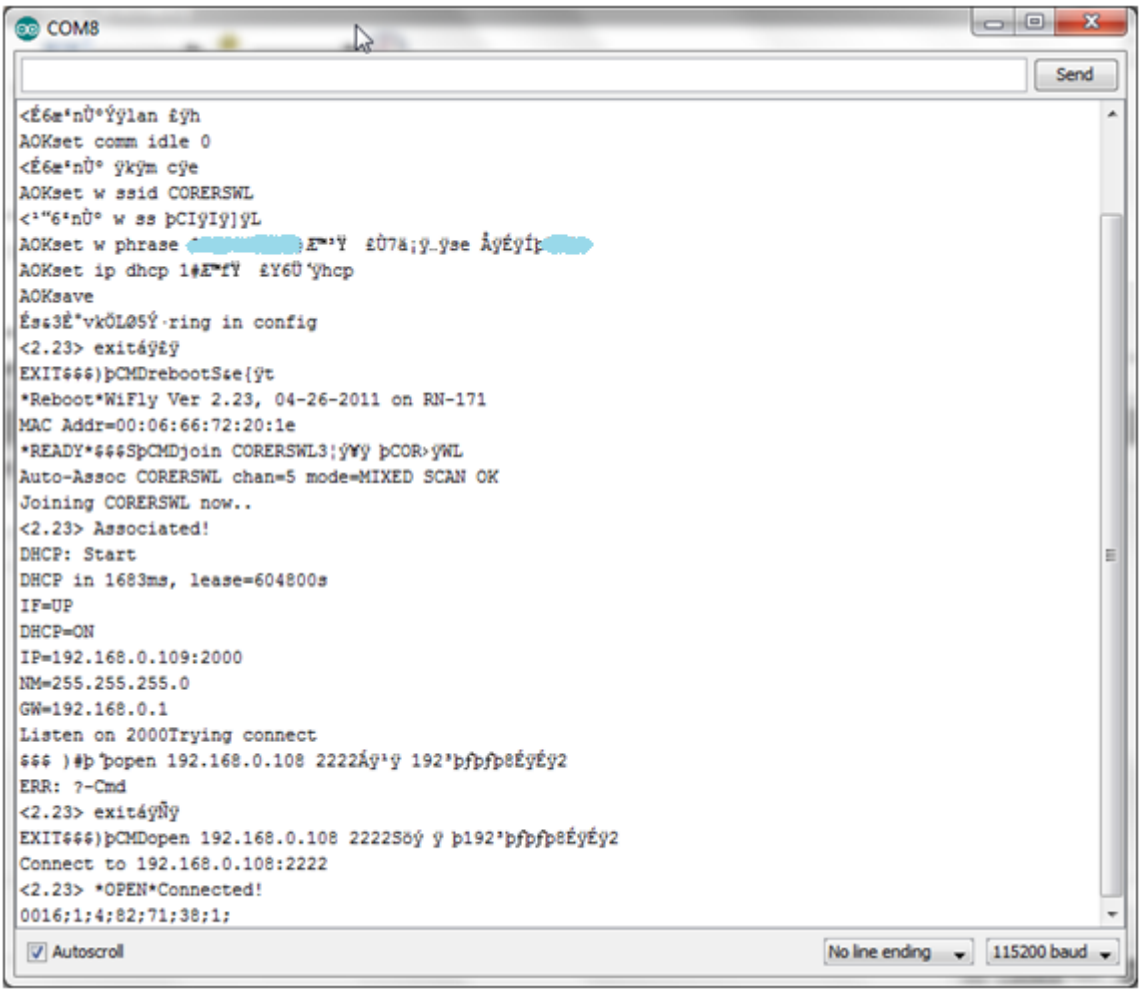


Figura 4.30 – Conexão TCP estabelecida

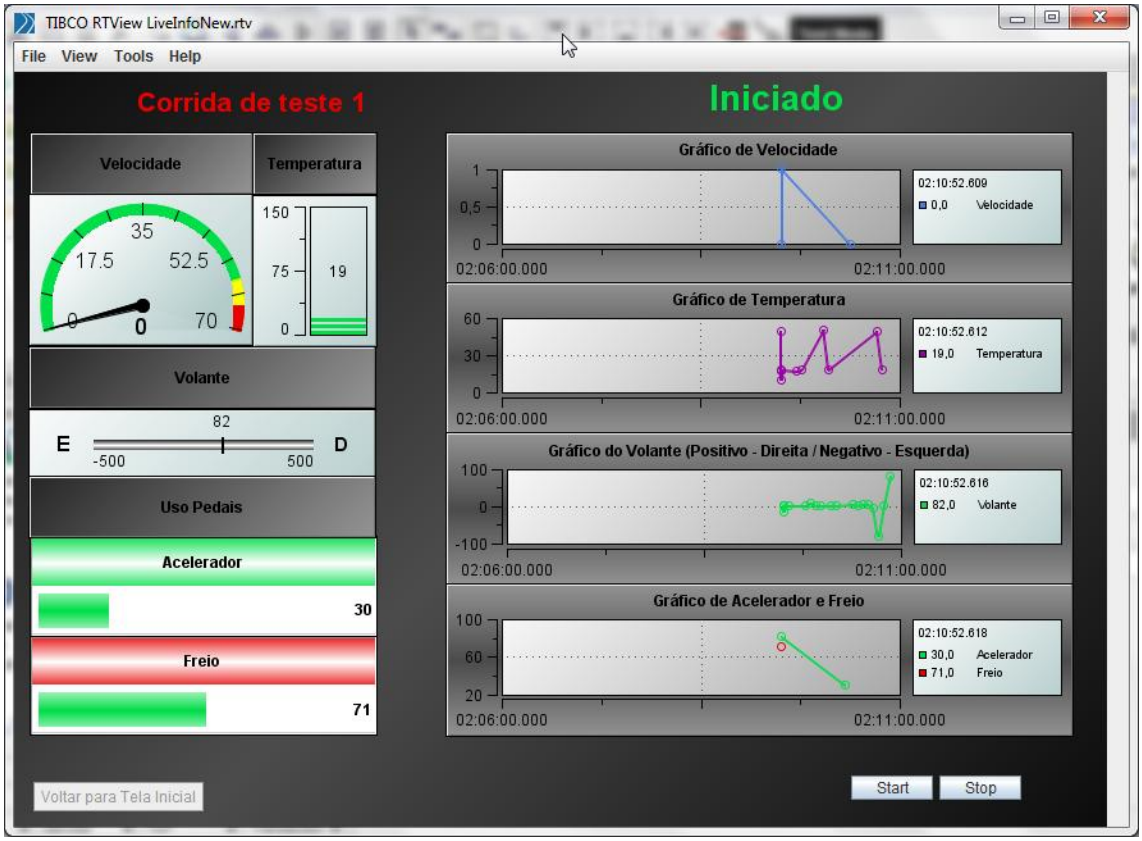


Figura 4.31 – Dados recebidos do kart são exibidos no painel online

4.5 INSTALAÇÃO NO KART

Com o protótipo funcionando corretamente, ele é instalado no kart, conforme Figura 4.32. Então verificado se as informações são transmitidas corretamente. A conexão estabelecida e os dados recebidos no servidor TCP, são mostrados na Figura 4.33. O painel “LiveInfoNew” também exibe os dados corretamente, conforme Figura 4.34.

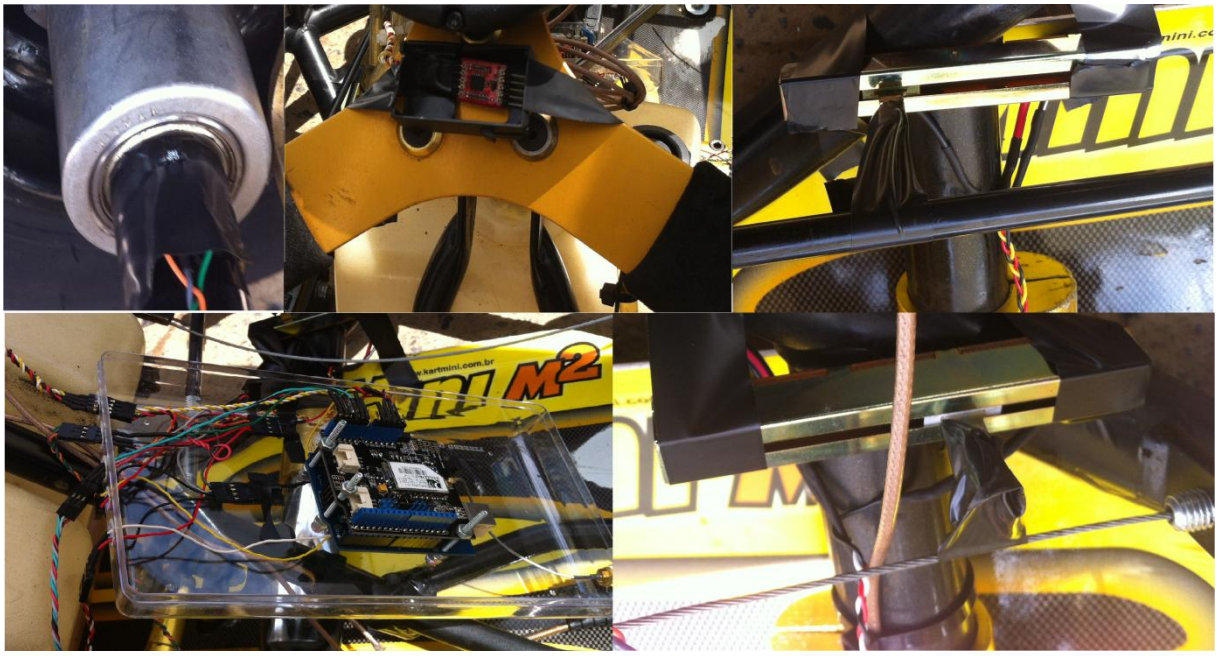


Figura 4.32 – Sensores instalados

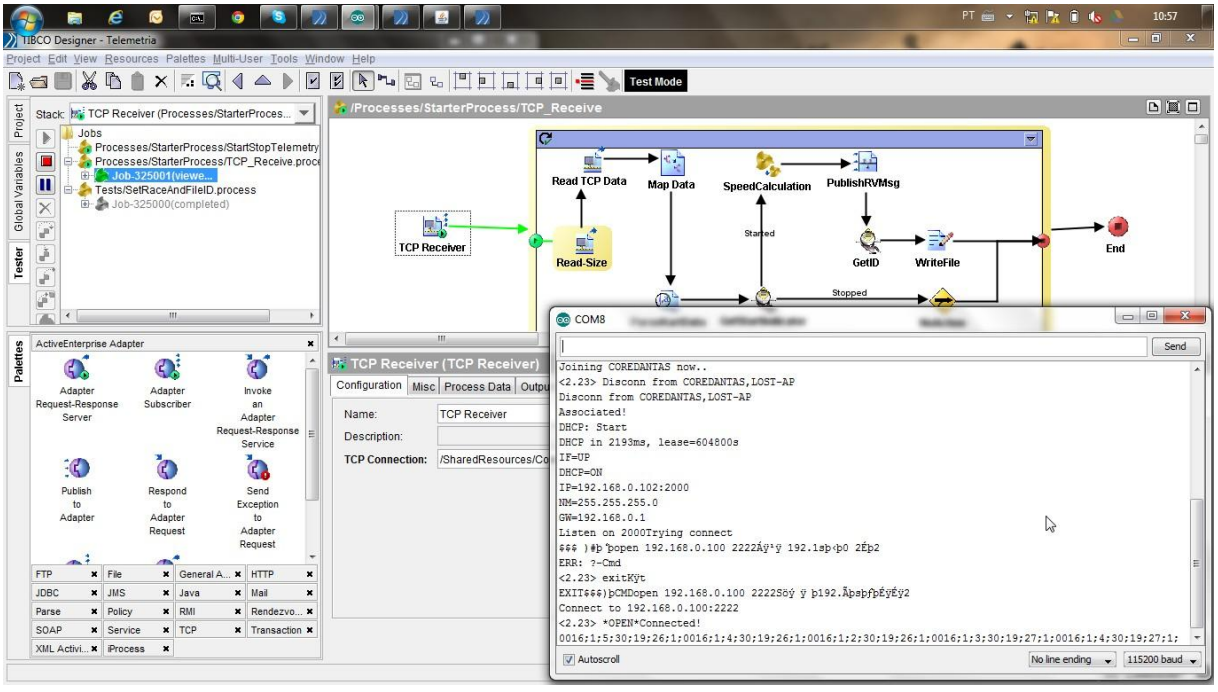


Figura 4.33 – Dados do kart recebidos

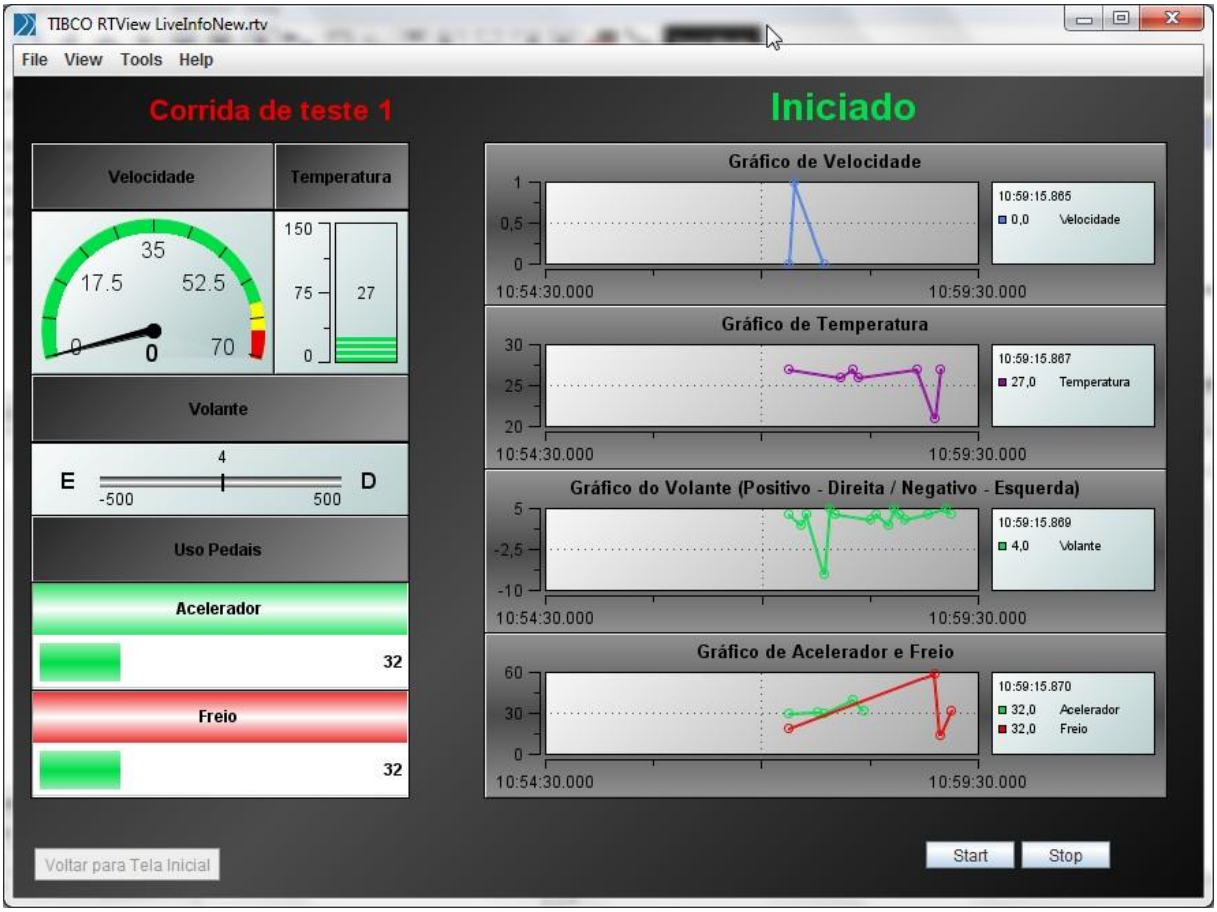


Figura 4.34 – Dados do kart exibidos nos gráficos online

5- CONCLUSÕES

Este projeto teve como finalidade o desenvolvimento de um protótipo de um sistema de telemetria em tempo real para kart, onde dados importantes foram enviados via WiFi, exibidos em um painel de gráficos online, gravados em um banco de dados e exibidos em um painel de gráficos para análise posterior. A aferição dos parâmetros de posição dos pedais, movimentação do volante, temperatura do motor e velocidade obtiveram um resultado satisfatório, os dados foram enviados e gravados de forma correta e os gráficos exibiram corretamente os dados. Pode-se comprovar que os sensores escolhidos atingiram os objetivos propostos.

Durante o desenvolvimento do projeto foram encontradas algumas dificuldades que atrasaram o cronograma. Houve uma demora na definição do meio de transmissão dos dados. Inicialmente, os dados seriam transmitidos por rádio frequência, porém isso aumentava a quantidade de dispositivos necessários para o funcionamento do projeto, então foi substituído para WiFi. Depois de definido o meio de transmissão, houve uma demora na decisão do protocolo em que a mensagem seria enviada. Inicialmente, seria utilizado o TRDP, que é o protocolo do TIBCO Rendezvous, porém o TIBCO Rendezvous não oferece suporte à microcontroladores. Depois foi pensado em utilizar o sistema de mensageria da CISCO, porém não foi encontrada nenhuma documentação informando o suporte a microcontroladores e também foi descartado. Por fim foi definido o TCP como protocolo de transmissão. Essa demora na definição do protocolo interferiu diretamente no desenvolvimento da solução de integração, que foi alterada diversas vezes para atender aos protocolos de transmissão. Outro atraso foi ocasionado pela WiFi Shield que queimou durante os testes e houve uma demora em conseguir uma nova. O giroscópio também queimou e teve de ser substituído por outro modelo, pois o modelo utilizado havia sido importado e no Brasil o preço dele é muito alto.

O desenvolvimento utilizando o Arduino foi fácil, pois o *framework* possibilitou muitas implementações, correções de código e testes em pouco tempo.

O desenvolvimento dos softwares utilizando a plataforma TIBCO acelerou o projeto, pois com um elevado conhecimento das ferramentas, o desenvolvimento dentro delas é muito rápido.

Como continuidade desse projeto pode ser implementada a marcação de tempo de volta com um Reed Swith, a adição de sensores como sensor de RPM, sensor de pressão dos pneus, sensor de força G, sensor de nível de combustível, sensor de pressão do óleo.

Outra frente de trabalho seria o desenvolvimento de uma interface entre o protótipo e o piloto, para que os dados sejam exibidos não apenas para a equipe, mas para o piloto também. As informações importantes exibir para o piloto seriam: a temperatura do motor, tempo de volta, nível do combustível e velocidade.

REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. ***Hardware*.** Disponível em: <http://arduino.cc/en/Main/Hardware>. Acesso em: 9 de setembro de 2012.

ARDUINO. **Language Reference.** Disponível em: <http://arduino.cc/en/Reference/HomePage>. Acesso em: 06 de outubro de 2012.

ARDUINO DUEMILANOVE. **Arduino Duemilanove.** Disponível em: <http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove>. Acesso em: 22 de outubro de 2012.

ATMEL. **ATmega48A/PA/88A/PA/168A/PA/328/P [DATASHEET SUMMARY]**. Disponível em: <http://www.atmel.com/Images/8271s.pdf>. Acesso em: 22 de outubro de 2012.

CAMPOS, Leonardo Barreto. **Conceitos Básicos da Linguagem C.** Disponível em: <http://www.univasf.edu.br/~leonardo.campos/Arquivos/Disciplinas/Alg_Prog_I_2007_1/Aula_01.pdf>. Acesso em: 11 de setembro de 2012.

COMMISSION INTERNATIONALE DE KARTING. **Definitions and Classification**. Disponível em: <http://www.cikfia.com/fileadmin/content/REGULATIONS/Sporting/Sporting%20Regulations/Web_DC_2012.pdf>. Acesso em: 9 de setembro de 2012.

CURRY, Edward, "Message-Oriented Middleware", *Capítulo em Middleware for Communications*, John Wiley & Sons, Chichester, Inglaterra, pp. 1-28, 2004.

IEEE802. **OFFICIAL IEEE 802.11 WORKING GROUP TIMELINES – 2012-11-16.** Disponível em: <http://www.ieee802.org/11/Reports/802.11_Timelines.htm>. Acesso em: 06 de dezembro de 2012.

KILIAN, Christopher. **Modern Control Technology: Components and Systems.** 2ª Edição, Delmar Learning, 2001.

FILHO, Angelo Lotierso. **Conceitos Básicos de Linguagem de Programação C.** Disponível em: <http://www.ancibe.com.br/Apostila%20de%20Algoritmos/apostila%20de%20linguagem%20C%20otima.pdf>. Acesso em: 11 de setembro de 2012.

MARTINS, Jorge. **Motores de combustão interna**, Publindústria, Porto, 2006.

MOTORSPORT. **Rubens Barrichello in Ferrari telemetry room.** Disponível em: <http://www.motorsport.com/f1/photo/main-gallery/rubens-barrichello-in-ferrari-telemetry-2/?i=206464&id=229346&sz=1&s=2>. Acesso em: 8 de outubro de 2012.

NATIONAL SEMICONDUCTOR. **Datasheet LM35.** Disponível em: <http://www.datasheetcatalog.org/datasheet/nationalsemiconductor/DS005516.PDF>. Acesso em: 9 de setembro de 2012.

ORACLE. **Introduction to Oracle Database.** Disponível em: <http://docs.oracle.com/cd/E11882_01/server.112/e25789/intro.htm#i68236>. Acesso em: 11 de setembro de 2012.

RADIOCONTROLADO. **Motores 2 tempos.** Disponível em: <http://radiocontrolado.com.br/2tempos.asp>. Acesso em: 06 de outubro de 2012.

RADIOCONTROLADO. **Motores 4 tempos.** Disponível em: <http://radiocontrolado.com.br/4tempos.asp>. Acesso em: 06 de outubro de 2012.

RCMASTER. Artigos – **Motores a Combustão, 4 e 2 tempos**. Disponível em: <http://www.rcmasters.com.br/files/motores2e4tempos.pdf> – Acesso em: 06 de outubro de 2012.

ROVING NETWORKS. **RN-171 Datasheet.** Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/WiFly-RN-171-DS-ver3.1r.pdf>. Acesso em: 30 de outubro de 2012.

SCARBSF1. **Telemetry and Data Analysis Introduction.** Disponível em: <http://scarbsf1.com/blog1/2011/08/>. Acesso em: 8 de outubro de 2012.

SEEED STUDIO. **Wifi Shield.** Disponível em: <http://www.seeedstudio.com/wiki/Wifi_Shield>. Acesso em: 30 de outubro de 2012.

SL. **RTView for TIBCO.** Disponível em: <http://www.sl.com/products/rtviewtibco_hawkmonitor.shtml>. Acesso em: 08 de outubro de 2012.

ST. **LPY530AL Datasheet.** Disponível em: <http://www.sparkfun.com/datasheets/Sensors/IMU/lpy530al.pdf>. Acesso em: 5 de novembro de 2012.

SOARES, Flávio Augusto; NOLL, Valdir. **Transdutores.** Disponível em: <http://florianopolis.ifsc.edu.br/vnoll/transdutores2.pdf>. Acesso em: 08 de outubro de 2012.

TIBCO. **Messaging.** Disponível em: <http://www.tibco.com/products/automation/messaging/default.jsp>. Acesso em: 11 de setembro de 2012.

TIBCO. **TIBCO ActiveMatrix BusinessWorks Concepts.** Disponível em: <https://docs.tibco.com/tibbr_integration?url=pub/activematrix_businessworks/5.9.3_march_2012/html/index.htm&product=396>. Acesso em: 20 de setembro de 2012.

TIBCO. **TIBCO Rendezvous Concepts.** Disponível em: <https://docs.tibco.com/emp/rendezvous/8.4.0-february-2012/doc/html/wwhelp/wwhimpl/js/html/wwhelp.htm#href=tib_rv_concepts/title.1.1.htm>. Acesso em: 11 de setembro de 2012.

WI-FI ALLIANCE. **The How and Why of Wi-Fi.** Disponível em: <https://www.wi-fi.org/knowledge-center/articles/how-and-why-wi-fi>. Acesso em: 30 de outubro de 2012.

ANEXO 1 – BIBLIOTECA WIFLY.H

```
#ifndef WIFLY_H
#define WIFLY_H

#include <Arduino.h>
#include "Hardware.h"
#include "Debug.h"
#include <SoftwareSerial.h>

#define NB_TRY_BEFORE_REBOOT 10
#define READ_TIMEOUT 10000
//
class WiflyClass: public SoftwareSerial {
//class WiflyClass {
public:
      WiflyClass(int,int);
      void init();

//WiflyClass:WiflySerial(int,int);
//          {
//                WiflySerial = new SoftwareSerial(int,int);
//          }
      //SoftwareSerial WiflySerial = new SoftwareSerial(tx,rx);
      //SoftwareSerial::SoftwareSerial WiflySerial = new
SoftwareSerial::SoftwareSerial(tx,rx);
      void join(const char *ssid);
      void waitForReady(boolean dhcp);
      void setConfig(const char *ssid, const char *passphrase);
      void setConfig(const char *ssid, const char *passphrase, const char
*ip, const char *mask ,const char *gateway);
      void reset();
      void closeAndExit();
      void reboot();
      bool checkConnection();
      void checkAssociated();
    bool connect(const char *host, const char *port);
    void writeToSocket(const char *data);
    char readFromSocket();
      bool canReadFromSocket();
      bool sendCommand(const char *cmd, char *expectedReturn, boolean
multipart=false);
      void factoryReset();
      char Read();
      void flush();
      bool find(char *target);
      bool findUntil(char *target, char *terminate);
      void skipChar(int count);
      void print(char data);
private:
      unsigned char disconCount;
      bool _dhcp;
      uint8_t unavailConn;
      //SoftwareSerial WiflySerial;
};

//extern WiflyClass Wifly;
//extern SoftwareSerial WiflySerial(int,int);
#endif // WIFLY_H
```

ANEXO 2 – BIBLIOTECA HARDWARE.H

```
#ifndef HARDWARE_H
#define HARDWARE_H

#define SDFAT_CHIP_SELECT 77
#define VS1011_RST 73
#define VS1011_XDCS 72
#define VS1011_XCS 71
#define VS1011_DREQ 70
#define WIFLY_RST 49
#define WIFLY_GPIO6 76
#define GREEN_LED 78
#define RED_LED 79

#endif // HARDWARE_H
```

ANEXO 3 – BIBLIOTECA DEBUG.H

```
#ifndef __DEBUG_H__
#define __DEBUG_H__

static const char sChar = 0x2;
static const char eChar = 0x3;
static const char dStringStart = ':';

//#define DEBUG
#define DEBUG_LEVEL 0

#define DEBUG_LOG(level, message) \
  if (DEBUG_LEVEL >= level) {\
      Serial.print(sChar);\
      Serial.print(dStringStart);\
    Serial.print(message);\
      Serial.print(eChar);\
  };
#endif
```

APÊNDICE A – WSDL CONCRETO UTILIZADO PARA REQUISIÇÕES SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Created by TIBCO WSDL-->
<wsdl:definitions
xmlns:ns2="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
DriverSchema.xsd"
xmlns:ns1="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
ResponseSchema.xsd"
xmlns:ns3="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
KartSchema.xsd" xmlns:tns="http://xmlns.example.com/1340980106678"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns0="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
RaceSchema.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
name="Untitled" targetNamespace="http://xmlns.example.com/1340980106678">
    <wsdl:types>
        <xs:schema
xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/Race
Schema.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Sc
hemas/RaceSchema.xsd" elementFormDefault="qualified"
attributeFormDefault="unqualified">
            <xs:element name="race">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Name" type="xs:string"/>
                        <xs:element name="Nickname" type="xs:string"/>
                        <xs:element name="Kart" type="xs:string"/>
                        <xs:element name="Date" type="xs:string"/>
                        <xs:element name="Karting" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
        <xs:schema
xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/Resp
onseSchema.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Sc
hemas/ResponseSchema.xsd" elementFormDefault="qualified"
attributeFormDefault="unqualified">
            <xs:element name="response">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Status" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
        <xs:schema
xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/Driv
erSchema.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Sc
hemas/DriverSchema.xsd" elementFormDefault="qualified"
attributeFormDefault="unqualified">
            <xs:element name="driver">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Name" type="xs:string"/>
                        <xs:element name="Nickname" type="xs:string"/>
```

```
                        <xs:element name="Category" type="xs:string"/>
                        <xs:element name="Age" type="xs:string"/>
                        <xs:element name="Gender" type="xs:string"/>
                        <xs:element name="Weight" type="xs:string"/>
                        <xs:element name="New" type="xs:string"
minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
        <xs:schema
xmlns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/Kart
Schema.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Sc
hemas/KartSchema.xsd" elementFormDefault="qualified"
attributeFormDefault="unqualified">
            <xs:element name="kart">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Number" type="xs:string"/>
                        <xs:element name="Karting" type="xs:string"/>
                        <xs:element name="Sensor" type="xs:string"/>
                        <xs:element name="New" type="xs:string"
minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
    <wsdl:service name="RecordService">
        <wsdl:port name="PortTypeEndpoint1"
binding="tns:PortTypeEndpoint1Binding">
            <soap:address
location="http://localhost:22/Services/Service.serviceagent/PortTypeEndpoin
t1"/>
        </wsdl:port>
    </wsdl:service>
    <wsdl:portType name="PortType">
        <wsdl:operation name="OperationRace">
            <wsdl:input message="tns:MessageInRace"/>
            <wsdl:output message="tns:MessageOut"/>
        </wsdl:operation>
        <wsdl:operation name="OperationDriver">
            <wsdl:input message="tns:MessageInDriver"/>
            <wsdl:output message="tns:MessageOut"/>
        </wsdl:operation>
        <wsdl:operation name="OperationKart">
            <wsdl:input message="tns:MessageInKart"/>
            <wsdl:output message="tns:MessageOut"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="PortTypeEndpoint1Binding" type="tns:PortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="OperationRace">
            <soap:operation style="document"
soapAction="/Services/Service.serviceagent/PortTypeEndpoint1/OperationRace"
/>
            <wsdl:input>
                <soap:body use="literal" parts="input"/>
            </wsdl:input>
```

```
            <wsdl:output>
                <soap:body use="literal" parts="output"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="OperationDriver">
            <soap:operation style="document"
soapAction="/Services/Service.serviceagent/PortTypeEndpoint1/OperationDrive
r"/>
            <wsdl:input>
                <soap:body use="literal" parts="input"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" parts="output"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="OperationKart">
            <soap:operation style="document"
soapAction="/Services/Service.serviceagent/PortTypeEndpoint1/OperationKart"
/>
            <wsdl:input>
                <soap:body use="literal" parts="Input"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" parts="output"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:message name="MessageInRace">
        <wsdl:part name="input" element="ns0:race"/>
    </wsdl:message>
    <wsdl:message name="MessageOut">
        <wsdl:part name="output" element="ns1:response"/>
    </wsdl:message>
    <wsdl:message name="MessageInDriver">
        <wsdl:part name="input" element="ns2:driver"/>
    </wsdl:message>
    <wsdl:message name="MessageInKart">
        <wsdl:part name="Input" element="ns3:kart"/>
    </wsdl:message>

</wsdl:definitions>
```

APÊNDICE B – WSDL ABSTRATO UTILIZADO NO WEBSERVICE

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
xmlns:ns2="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
ResponseSchema.xsd" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns4="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
DriverSchema.xsd"
xmlns:ns="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/K
artSchema.xsd"
xmlns:ns3="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
RaceSchema.xsd" xmlns:tns="http://xmlns.example.com/1340980165643"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://xmlns.example.com/1340980165643">
    <import
namespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
ResponseSchema.xsd" location="../Schemas/Response.xsd"/>
    <import
namespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
RaceSchema.xsd" location="../Schemas/Race.xsd"/>
    <import
namespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
KartSchema.xsd" location="../Schemas/Kart.xsd"/>
    <import
namespace="http://www.tibco.com/schemas/Telemetria/SharedResources/Schemas/
DriverSchema.xsd" location="../Schemas/Driver.xsd"/>
    <message name="MessageInKart">
        <part name="Input" element="ns:kart"/>
    </message>
    <message name="MessageOut">
        <part name="output" element="ns2:response"/>
    </message>
    <message name="MessageInRace">
        <part name="input" element="ns3:race"/>
    </message>
    <message name="MessageInDriver">
        <part name="input" element="ns4:driver"/>
    </message>
    <portType name="PortType">
        <operation name="OperationKart">
            <input message="tns:MessageInKart"/>
            <output message="tns:MessageOut"/>
        </operation>
        <operation name="OperationDriver">
            <input message="tns:MessageInDriver"/>
            <output message="tns:MessageOut"/>
        </operation>
        <operation name="OperationRace">
            <input message="tns:MessageInRace"/>
            <output message="tns:MessageOut"/>
        </operation>
    </portType>
</definitions>
```

APÊNDICE C – PROGRAMA DO ARDUINO

```
#include "Wifly.h"
#include <SoftwareSerial.h>
WiflyClass Wifly(2,3);

// Definições de pinos utilizados
const int aceleratorPin = 0; //Define o pino A0 para o potenciômetro do
acelerador
const int brakePin = 1; //Define o pino A1 para o potenciômetro do freio
const int temperaturePin = 2; //Define o pino A2 para o LM35
const int speedPin = 3; //Define o pino A3 para o sensor de velocidade
const int Xpin = 4; //Define o pino A4 para o eixo x do giroscópio
const int Zpin = 5; //Define o pino A5 para o eixo z do giroscópio

//Define variáveis globais
int IDSensor = 1;//Define valor do ID do Sensor
int x, z; // Variáveis de leitura do Giroscópio
int Acelerator; //Variável que recebe o valor da posição do acelerador
int Brake; //Variável que recebe o valor da posição do freio
int Temperature; //Variável que recebe o valor da temperatura do motor
int Speed; //Variável que recebe o valor da velocidade do kart
String data; //String que contém o valor de todos os sensores
String info; //String com todas informações necessárias para o envio dos
dados por TCP
char delimiter; //Define o caracter delimitador
char* msg; //Mensagem que será enviada por TCP
char temp[50];

void setup()
{
  //Define a velocidade da comunicação em 115200 bits por segundo
  Serial.begin(115200);

  //Inicializa e configura a WiFi Shield
  Wifly.init();
  Wifly.setConfig("SSID", "**********");//Define os parâmetros de conexão
  Wifly.join("SSID");//Conecta na rede
  Wifly.checkAssociated();//Verifica se o arduíno está conectado
  while(!Wifly.connect("192.168.2.1","2222"));//Conecta no serviço remoto
}

void loop()
{

  Acelerator = (int)analogRead(aceleratorPin)/10.23;//Define valor da
aceleração. A divisão serve para que o valor varie entre 0 e 100
  Brake = (int)analogRead(brakePin)/10.23;//Define valor do freio. A
divisão serve para que o valor varie entre 0 e 100
  Temperature = analogRead(temperaturePin);
  Temperature = (int)map(Temperature, 0, 1023, 0 ,500);
  Speed = 0;
  if(analogRead(speedPin) > 1000){
    Speed = 1;
  }
  z = 250 - analogRead(Zpin);

  data = String(';');
  delimiter = ';';
```

```
  //Atribui à variável data os valores dos sensores
  data.concat(IDSensor);
  data.concat(delimiter);
  data.concat(z);
  data.concat(delimiter);
  data.concat(Acelerator);
  data.concat(delimiter);
  data.concat(Brake);
  data.concat(delimiter);
  data.concat(Temperature);
  data.concat(delimiter);
  data.concat(Speed);
  data.concat(delimiter);

  //Mede o tamanho da string data e constrói a string info com as
informações:
  //Os 4 primeiros caracteres são o tamanho da string com dados
  //Os caracteres seguintes são os dados

  info = "00";
  if(data.length()<10){
    info.concat("0");
  }
  info.concat(data.length());
  info.concat(data);

  //Atribui o valor de info á uma variável char temporária chamada temp
  info.toCharArray(temp,info.length()+1);
  //Atribui o valor da variável temp à constante msg
  msg = temp;
  Wifly.writeToSocket(msg);
  delay(40);  // Define tempo de espera

}
```

APÊNDICE D – SQL UTILIZADO PELO SPOTFIRE

```
SELECT
   D1."ACELERADOR" AS "ACELERADOR",
   D1."FREIO" AS "FREIO",
   D1."ID_DADO" AS "IDDADO",
   C2."NM_CORRIDA" AS "NOMECORRIDA",
   P3."NOME" AS "NOMEPILOTOS",
   D1."TEMPERATURA" AS "TEMPERATURA",
   D1."TS_DADO" AS "TIMESTAMPDADOS",
   D1."VELOCIDADE" AS "VELOCIDADE",
   D1."VOLANTE" AS "VOLANTE"
FROM
   "DIOGO"."DADOS" D1,
   "DIOGO"."CORRIDAS" C2,
   "DIOGO"."PILOTOS" P3
WHERE
   (D1."NUM_SQC_CORRIDA" = C2."NUM_SQC_CORRIDA")
   AND (D1."ID_PILOTO" = P3."ID_PILOTO")
   AND <conditions>
```

APÊNDICE E – PROGRAMA DE TESTES DO PROTÓTIPO

```
// Definições de pinos utilizados
const int aceleratorPin = 0; //Define o pino A0 para o potenciômetro do
acelerador
const int brakePin = 1; //Define o pino A1 para o potenciômetro do freio
const int temperaturePin = 2; //Define o pino A2 para o LM35
const int speedPin = 3; //Define o pino A3 para o sensor de velocidade
const int Xpin = 4; //Define o pino A4 para o eixo x do giroscópio
const int Zpin = 5; //Define o pino A5 para o eixo z do giroscópio

//Define variáveis globais
int IDSensor = 1;//Define valor do ID do Sensor
int x, z; // Variáveis de leitura do Giroscópio
int Acelerator; //Variável que recebe o valor da posição do acelerador
int Brake; //Variável que recebe o valor da posição do freio
int Temperature; //Variável que recebe o valor da temperatura do motor
int Speed; //Variável que recebe o valor da velocidade do kart
String data; //String que contém o valor de todos os sensores
String info; //String com todas informações necessárias para o envio dos
dados por TCP
char delimiter; //Define o caracter delimitador
char* msg; //Mensagem que será enviada por TCP
char temp[30];

void setup()
{
  //Define a velocidade da comunicação em 115200 bits por seegundo
  Serial.begin(115200);

}

void loop()
{
  Acelerator = (int)analogRead(aceleratorPin)/10.23;//Define valor da
aceleração. A divisão serve para que o valor varie entre 0 e 100
  Brake = (int)analogRead(brakePin)/10.23;//Define valor do freio. A
divisão serve para que o valor varie entre 0 e 100
  Temperature = analogRead(temperaturePin);
  Temperature = (int)map(Temperature, 0, 1023, 0 ,500);
  Speed = 0;
  if(analogRead(speedPin) > 1000){
    Speed = 1;
  }

  z = 250 - analogRead(Zpin);

  //Exibe os valores dos sensores separadamente
  Serial.println("ID Sensor");
  Serial.println(IDSensor);
  Serial.println("Volante");
  Serial.println(z, DEC);
  Serial.println("Acelerador");
  Serial.println(Acelerator);
  Serial.println("Freio");
  Serial.println(Brake);
  Serial.println("Temperatura");
  Serial.println(Temperature);
```

```
  Serial.println("Velocidade");
  Serial.print(Speed);
  Serial.println();


  data = String(';');
  delimiter = ';';
  //Atribui à variável data os valores dos sensores
  data.concat(IDSensor);
  data.concat(delimiter);
  data.concat(z);
  data.concat(delimiter);
  data.concat(Acelerator);
  data.concat(delimiter);
  data.concat(Brake);
  data.concat(delimiter);
  data.concat(Temperature);
  data.concat(delimiter);
  data.concat(Speed);
  data.concat(delimiter);
  //Exibe os dados da variável data
  Serial.println("data:");
  Serial.println(data);

  //Atribui o tamanho da string data e seus valores à variável info
  info = "00";
  if(data.length()<10){
    info.concat("0");
  }
  info.concat(data.length());
  info.concat(data);
  //Exibe os dados da variável info
  Serial.println("info:");
  Serial.println(info);
  //Transforma a String info em char e atribui esse valor à variável
temporária temp
  info.toCharArray(temp,info.length()+1);

  //Exibe os dados da viriável temp
  Serial.println("temp:");
  Serial.println(temp);

  //Atribui o valor da variável temp à const char msg
  msg = temp;
  //Exibe os dados da constante msg
  Serial.println("msg:");
  Serial.println(msg);
  Serial.println();
  delay(2000);  // Define tempo de espera

}
```

APÊNDICE F – CÓDIGO CORRIGIDO DO ARQUIVO WIFLY.CPP

```
#include "Wifly.h"

void WiflyClass::init() {

	//pinMode(WIFLY_RST, OUTPUT);
	//pinMode(WIFLY_GPIO6, INPUT);
	SoftwareSerial::begin(9600);
	//reset();
	//unavailConn = 0;
}

void WiflyClass::waitForReady(boolean dhcp) {
	_dhcp = dhcp;
	while(!find("Associated!"));
	if(_dhcp) {

		while(!find("DHCP in"));

	} else {
		while(!find("Using Static IP"));

	}
	find("Listen on 2000");
}

void WiflyClass::setConfig(const char *ssid, const char *passphrase) {
	sendCommand("$$$", "CMD");//enter the setting model
	sendCommand("set w join 0\r","AOK");//set it not to join automatic
	sendCommand("set comm remote 0\r", "AOK");//set the message(*OPEN*)
when connected the remote host/client
	sendCommand("set w linkmon 5\r", "AOK");//set it connect the route
per 5 sec
	sendCommand("set wlan auth 4\r","AOK");//set the wifly's insecure
model
	sendCommand("set comm idle 0\r", "AOK");//set it when is connected
will not disconnect
	sendCommand("set w ssid ","");//set the name of the wifi you want to
join in
	sendCommand(ssid, "", true);
	sendCommand("\r", "AOK", true);
	if(passphrase != NULL) {
		sendCommand("set w phrase ","");//set the phrase of the wifi
		sendCommand(passphrase, "", true);
		sendCommand("\r", "AOK", true);
	}
	sendCommand("set ip dhcp 1\r","AOK");//open the DHCP,so it will get
the ip automatic
	sendCommand("save\r", "Storing in config");//save it
	sendCommand("exit\r", "EXIT");//exit the setting model
	reboot();
}

void WiflyClass::setConfig(const char *ssid, const char *passphrase, const
char *ip, const char *mask, const char *gateway) {
	sendCommand("$$$", "CMD");
	sendCommand("set w join 1\r", "AOK");
	sendCommand("set comm remote 0\r", "AOK");
```

```
	sendCommand("set w linkmon 5\r", "AOK");
	sendCommand("set comm idle 0\r", "AOK");
	sendCommand("set wlan auth 4\r","AOK");//set the wifly's insecure
model
	sendCommand("set dns address 192.168.1.2\r", "AOK");
	sendCommand("set w ssid ","");
	sendCommand(ssid, "", true);
	sendCommand("\r", "AOK", true);
	if(passphrase != NULL) {
		sendCommand("set w phrase ","");
		sendCommand(passphrase, "", true);
		sendCommand("\r", "AOK", true);
	}
	sendCommand("set ip dhcp 0\r","AOK");
	sendCommand("set ip address ", "");
	sendCommand(ip, "", true);
	sendCommand("\r", "AOK", true);
	sendCommand("set ip gateway ", "");
	sendCommand(gateway, "", true);
	sendCommand("\r", "AOK", true);
	sendCommand("set ip netmask ", "");
	sendCommand(mask, "", true);
	sendCommand("\r", "AOK", true);
	sendCommand("save\r", "Storing in config");
	sendCommand("exit\r", "EXIT");
	reboot();
}

void WiflyClass::checkAssociated() {
	while(!find("Associated!"));
	while(!find("DHCP in"));
	find("Listen on 2000");

}

void WiflyClass::reboot() {
	sendCommand("$$$", "CMD");
	sendCommand("reboot\r","*Reboot*");
	while(!find("*READY*"));
}

void WiflyClass::join(const char *ssid)
{
	sendCommand("$$$", "CMD");
	sendCommand("join ","" );
	sendCommand(ssid,"");
	sendCommand("\r","");
}

void WiflyClass::reset() {
	digitalWrite(WIFLY_RST, LOW);
	delay(1);
	digitalWrite(WIFLY_RST, HIGH);
	//while(!find("*READY*"));
}

bool WiflyClass::checkConnection() {
	unsigned long _start = millis();
	while((_start + 1000) > millis()) {
		int i = digitalRead(WIFLY_GPIO6);
		if(i == 1) {
```

```
                        unavailConn++;
                        if(unavailConn == 10) {
                              reboot();
                              waitForReady(_dhcp);
                              unavailConn = 0;
                        }
                        return false;
                  }
            }
            return true;
}

bool WiflyClass::connect(const char *host, const char *port) {
            sendCommand("$$$", "CMD");
            sendCommand("open ", "");
            sendCommand(host, "", true);
            sendCommand(" ", "", true);
            sendCommand(port, "", true);
            boolean ret = sendCommand("\r", "*OPEN*", true);
            if(!ret) {
                  sendCommand("exit\r", "EXIT");
                  return false;
            }
            return true;
}

void WiflyClass::closeAndExit() {
            sendCommand("close\r","");
            sendCommand("exit\r", "EXIT");
}

void WiflyClass::writeToSocket(const char *data) {
            sendCommand(data, "");
}

void WiflyClass::print(char data) {
            SoftwareSerial::print(data);
            Serial.print(data);
}

char WiflyClass::readFromSocket() {
            return Read();
}

bool WiflyClass::canReadFromSocket() {
            return (SoftwareSerial::available() > 0 ? true:false);
            //return (WiflySerial::available() > 0 ? true:false);
}

void WiflyClass::factoryReset() {
            sendCommand("$$$", "CMD");
            sendCommand("factory RESET\r", "");
            sendCommand("exit\r", "EXIT");
            reboot();
}

bool WiflyClass::sendCommand(const char *cmd, char *expectedReturn, boolean
multipart) {
            SoftwareSerial::print(cmd);
//#ifdef DEBUG
            Serial.print(cmd);
```

```
//#endif
      //WiflySerial::print(cmd);
      return find(expectedReturn);
}

char WiflyClass::Read() {
      long start = millis();
      char c;
      int temp;
      while(millis() < (start + READ_TIMEOUT)) {

            if (SoftwareSerial::available() > 0) {
                  char c = SoftwareSerial::read();
                  Serial.print(c);
                  return c;
            }
      }
      return NULL;
}

bool WiflyClass::find(char *target) {
    return findUntil(target, NULL);
}

// as above but search ends if the terminate string is found
bool WiflyClass::findUntil(char *target, char *terminate)
{
    unsigned char targetLen = strlen(target);
    unsigned char index = 0;  // maximum target string length is 255
unsigned chars
    unsigned char termIndex = 0;
    unsigned char termLen = (terminate == NULL ? 0 : strlen(terminate));
    char c;

    if( *target == 0)
            return true;   // return true if target is a null string
    while( (c = Read()) != 0){
            if( c == target[index]){
                  if(++index >= targetLen){ // return true if all chars in
the target match
                        return true;
                  }
            }
            else{
                  index = 0;  // reset index if any char does not match
            }
            if(termLen > 0 && c == terminate[termIndex]){
                  if(++termIndex >= termLen)
                        return false;       // return false if terminate
string found before target string
            }
            else
                  termIndex = 0;
    }
    return false;
}

void WiflyClass::skipChar(int count) {
      int i=0;
      for(i=0;i<count;i++)
            read();
```

```
}

void WiflyClass::flush() {
      SoftwareSerial::flush();
      //WiflySerial::flush();
}

WiflyClass::WiflyClass(int rx,int tx):SoftwareSerial(rx,tx)
{
      //int a = 5;
      //PenClass pen;
      //pen = new PenClass(blue);
      //SoftwareSerial WiflySerial = new SoftwareSerial(tx,rx)
}
```

APÊNDICE G – FLUXO DO PROJETO